Capitolo 95

Sorgenti della libreria generale

«

95.1 os32: file isolati della directory «lib/»

95.1.1 lib/NULL.h

Si veda la sezione 91.3.

```
#ifndef _NULL_H
#define _NULL_H       1
//----------------------------------------------------------------------
#define NULL ((void *) 0)
//----------------------------------------------------------------------
#endif
```

95.1.2 lib/SEEK.h

Si veda la sezione 91.3.

```
#ifndef _SEEK_H
#define _SEEK_H      1
//----------------------------------------------------------
// These values are used inside 'stdio.h' and
// 'unistd.h'.
//----------------------------------------------------------
#define SEEK_SET        0       // From the start.
#define SEEK_CUR        1       // From current
                                // position.
#define SEEK_END        2       // From the end.
//----------------------------------------------------------
#endif
```

95.1.3 lib/assert.h

Si veda la sezione 88.6.

```
#ifndef _ASSERT_H
#define _ASSERT_H       1
//----------------------------------------------------------
#include <stdio.h>
//----------------------------------------------------------
#ifdef NDEBUG
#define assert(ignore) ((void)0)
#else
#define assert(ASSERTION)                             \
    ({if ((ASSERTION)==0)                             \
       fprintf (stderr,                               \
                "Assertion failed: " # ASSERTION      \
                ", function %s, file %s, line %u.\n", \
                __func__, __FILE__, __LINE__);})
#endif
//----------------------------------------------------------
#endif
```

95.1.4 lib/clock_t.h

Si veda la sezione 91.3.

```
#ifndef _CLOCK_T_H
#define _CLOCK_T_H       1
//----------------------------------------------------------
#include <stdint.h>
//----------------------------------------------------------
typedef uint64_t clock_t;
//----------------------------------------------------------
#endif
```

95.1.5 lib/ctype.h

Si veda la sezione 91.3.

```
#ifndef _CTYPE_H
#define _CTYPE_H       1
//----------------------------------------------------------
#include <NULL.h>
//----------------------------------------------------------
#define isblank(C)  ((int) (C == ' ' || C == '\t'))
#define isspace(C)  ((int) (C == ' ' \
                            || C == '\f' \
                            || C == '\n' \
                            || C == '\n' \
                            || C == '\r' \
                            || C == '\t' \
                            || C == '\v'))
#define isdigit(C)  ((int) (C >= '0' && C <= '9'))
#define isxdigit(C) \
    ((int) ((C >= '0' && C <= '9') \
            || (C >= 'A' && C <= 'F') \
            || (C >= 'a' && C <= 'f')))
#define isupper(C)  ((int) (C >= 'A' && C <= 'Z'))
#define islower(C)  ((int) (C >= 'a' && C <= 'z'))
#define iscntrl(C)  ((int) ((C >= 0x00 && C <= 0x1F) \
                            || C == 0x7F))
#define isgraph(C)  ((int) (C >= 0x21 && C <= 0x7E))
#define isprint(C)  ((int) (C >= 0x20 && C <= 0x7E))
#define isalpha(C)  (isupper (C) || islower (C))
#define isalnum(C)  (isalpha (C) || isdigit (C))
#define ispunct(C)  (isgraph (C) && (!isspace (C)) \
                     && (!isalnum (C)))
#define tolower(C)  (isupper (C) ? ((C) + 0x20) : (C))
#define toupper(C)  (islower (C) ? ((C) - 0x20) : (C))
#define toascii(C)  (C & 0x7F)
#define _tolower(C) (isupper (C) ? ((C) + 0x20) : (C))
#define _toupper(C) (islower (C) ? ((C) - 0x20) : (C))
//----------------------------------------------------------
#endif
```

95.1.6 lib/limits.h

«

Si veda la sezione 91.3.

```
#ifndef _LIMITS_H
#define _LIMITS_H       1
//----------------------------------------------------------
#define CHAR_UNSIGNED   0
//----------------------------------------------------------
#define CHAR_BIT        (8)
//
#define SCHAR_MIN       (-0x80)
#define SCHAR_MAX        (0x7F)
#define UCHAR_MAX        (0xFF)
//
#ifdef CHAR_UNSIGNED
#define CHAR_MIN      (0)
#define CHAR_MAX       UCHAR_MAX
#else
#define CHAR_MIN       SCHAR_MIN
#define CHAR_MAX       SCHAR_MAX
#endif
```

```
//
#define MB_LEN_MAX       (16)
//
#define SHRT_MIN        (-0x8000)
#define SHRT_MAX         (0x7FFF)
#define USHRT_MAX        (0xFFFF)
//
#define INT_MIN         (-0x80000000)
#define INT_MAX          (0x7FFFFFFF)
#define UINT_MAX         (0xFFFFFFFFU)
//
#define LONG_MIN        (-0x80000000L)
#define LONG_MAX         (0x7FFFFFFFL)
#define ULONG_MAX        (0xFFFFFFFFUL)
//
#define LLONG_MIN       (-0x8000000000000000LL)
#define LLONG_MAX        (0x7FFFFFFFFFFFFFFFLL)
#define ULLONG_MAX       (0xFFFFFFFFFFFFFFFFULL)
#define WORD_BIT         (32)
#define LONG_BIT         (32)
#define SSIZE_MAX        (0x7FFFFFFFL)
//----------------------------------------------------------
#define ARG_MAX      8192          // Arguments+environment
                              // max length.
#define ATEXIT_MAX     32          // Max "at exit"
                              // functions.
#define FILESIZEBITS   32          // File size needs integer
                              // size...
#define LINK_MAX      254          // Max links per file.
#define NAME_MAX       14          // File name max
                              // (Minix 1 fs).
#define OPEN_MAX      128          // Max open files per
                              // process.
#define PATH_MAX     1024          // Path, including
                              // final '\0'.
#define MAX_CANON     256          // Max bytes in
                              // canonical tty queue.
```

```
#define MAX_INPUT       1       // Max bytes in tty
                           // input queue.
//----------------------------------------------------------
#define CHLD_MAX          INT_MAX       // Not used.
#define HOST_NAME_MAX     INT_MAX       // Not used.
#define LOGIN_NAME_MAX    INT_MAX       // Not used.
#define PAGE_SIZE         INT_MAX       // Not used.
#define RE_DUP_MAX        INT_MAX       // Not used.
#define STREAM_MAX        INT_MAX       // Not used.
#define SYMLOOP_MAX       INT_MAX       // Not used.
#define TTY_NAME_MAX      INT_MAX       // Not used.
#define TZNAME_MAX        INT_MAX       // Not used.
#define PIPE_MAX          INT_MAX       // Not used.
#define SYMLINK_MAX       INT_MAX       // Not used.
//----------------------------------------------------------
#endif
```

95.1.7 lib/ptrdiff_t.h

Si veda la sezione 91.3.

```
#ifndef _PTRDIFF_T_H
#define _PTRDIFF_T_H    1
//----------------------------------------------------------
typedef int ptrdiff_t;
//----------------------------------------------------------
#endif
```

95.1.8 lib/restrict.h

Si veda la sezione 91.3.

```
#ifndef _RESTRICT_H
#define _RESTRICT_H      1
//----------------------------------------------------------
// At the moment, the GCC compiler does not support
// the ‘restrict’ keyword.
```

```
//----------------------------------------------------------
#define restrict /**/
//----------------------------------------------------------
#endif
```

95.1.9 lib/size_t.h

Si veda la sezione 91.3.

```
#ifndef _SIZE_T_H
#define _SIZE_T_H       1
//----------------------------------------------------------
// The type `size_t' *must* be equal to an `int'.
//----------------------------------------------------------
typedef unsigned int size_t;
//----------------------------------------------------------
#endif
```

95.1.10 lib/stdarg.h

Si veda la sezione 91.3.

```
#ifndef _STDARG_H
#define _STDARG_H       1
//----------------------------------------------------------
typedef unsigned char *va_list;
//----------------------------------------------------------
#define va_start(ap, last) \
    ((void) ((ap) = \
        ((va_list) &(last)) + (sizeof (last))))
#define va_end(ap) ((void) ((ap) = 0))
#define va_copy(dest, src) \
    ((void) ((dest) = (va_list) (src)))
#define va_arg(ap, type) \
    (((ap) = (ap) + (sizeof (type))), \
        *((type *) ((ap) - (sizeof (type)))))
//----------------------------------------------------------
```

```
#endif
```

95.1.11 lib/stdbool.h

Si veda la sezione 91.3.

```
#ifndef _STDBOOL_H
#define _STDBOOL_H      1
//----------------------------------------------------------
#define bool    _Bool
#define true    1
#define false   0
#define __bool_true_false_are_defined   1
//----------------------------------------------------------
#endif
```

95.1.12 lib/stddef.h

Si veda la sezione 91.3.

```
#ifndef _STDDEF_H
#define _STDDEF_H      1
//----------------------------------------------------------
#include <ptrdiff_t.h>
#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>
//----------------------------------------------------------
#define offsetof(type, member) \
    ((size_t) &((type *)0)->member)
//----------------------------------------------------------
#endif
```

95.1.13 lib/stdint.h

Si veda la sezione 91.3.

```
#ifndef _STDINT_H
#define _STDINT_H       1
//----------------------------------------------------------
typedef signed char int8_t;
typedef short int int16_t;
typedef int int32_t;
typedef long long int int64_t;
//
typedef unsigned char uint8_t;
typedef unsigned short int uint16_t;
typedef unsigned int uint32_t;
typedef unsigned long long int uint64_t;
//
#define INT8_MIN              (-0x80)
#define INT16_MIN             (-0x8000)
#define INT32_MIN             (-0x80000000)
#define INT64_MIN             (-0x8000000000000000LL)
//
#define INT8_MAX                0x7F
#define INT16_MAX               0x7FFF
#define INT32_MAX               0x7FFFFFFF
#define INT64_MAX               0x7FFFFFFFFFFFFFFFLL
//
#define UINT8_MAX               0xFF
#define UINT16_MAX              0xFFFF
#define UINT32_MAX              0xFFFFFFFFU
#define UINT64_MAX              0xFFFFFFFFFFFFFFFFULL
//----------------------------------------------------------
typedef signed char int_least8_t;
typedef short int int_least16_t;
typedef int int_least32_t;
typedef long long int int_least64_t;
//
typedef unsigned char uint_least8_t;
typedef unsigned short int uint_least16_t;
typedef unsigned int uint_least32_t;
typedef unsigned long long int uint_least64_t;
//
#define INT_LEAST8_MIN        (-0x80)
#define INT_LEAST16_MIN       (-0x8000)
#define INT_LEAST32_MIN       (-0x80000000)
#define INT_LEAST64_MIN       (-0x8000000000000000LL)
//
#define INT_LEAST8_MAX          0x7F
#define INT_LEAST16_MAX         0x7FFF
#define INT_LEAST32_MAX         0x7FFFFFFF
#define INT_LEAST64_MAX         0x7FFFFFFFFFFFFFFFLL
//
#define UINT_LEAST8_MAX         0xFF
#define UINT_LEAST16_MAX        0xFFFF
#define UINT_LEAST32_MAX        0xFFFFFFFFU
#define UINT_LEAST64_MAX        0xFFFFFFFFFFFFFFFFULL
//----------------------------------------------------------
#define INT8_C(VAL)             VAL
#define INT16_C(VAL)            VAL
#define INT32_C(VAL)            VAL
#define INT64_C(VAL)            VAL ## LL
//
#define UINT8_C(VAL)            VAL
#define UINT16_C(VAL)           VAL
#define UINT32_C(VAL)           VAL ## U
#define UINT64_C(VAL)           VAL ## ULL
//----------------------------------------------------------
typedef signed char int_fast8_t;
typedef int int_fast16_t;
typedef int int_fast32_t;
typedef long long int int_fast64_t;
//
typedef unsigned char uint_fast8_t;
typedef unsigned int uint_fast16_t;
typedef unsigned int uint_fast32_t;
typedef unsigned long long int uint_fast64_t;
//
#define INT_FAST8_MIN         (-0x80)
#define INT_FAST16_MIN        (-0x80000000)
#define INT_FAST32_MIN        (-0x80000000)
#define INT_FAST64_MIN        (-0x8000000000000000LL)
//
#define INT_FAST8_MAX           0x7F
#define INT_FAST16_MAX          0x7FFFFFFF
#define INT_FAST32_MAX          0x7FFFFFFF
#define INT_FAST64_MAX          0x7FFFFFFFFFFFFFFFLL
//
#define UINT_FAST8_MAX          0xFF
#define UINT_FAST16_MAX         0xFFFFFFFFU
#define UINT_FAST32_MAX         0xFFFFFFFFU
#define UINT_FAST64_MAX         0xFFFFFFFFFFFFFFFFULL
//----------------------------------------------------------
typedef int intptr_t;
typedef unsigned int uintptr_t;
//
#define INTPTR_MIN            (-0x80000000)
#define INTPTR_MAX              0x7FFFFFFF
#define UINTPTR_MAX             0xFFFFFFFFU
//
typedef long long int intmax_t;
typedef unsigned long long int uintmax_t;
//
#define INTMAX_C(VAL)           VAL ## LL
#define UINTMAX_C(VAL)          VAL ## ULL
#define INTMAX_MIN   (-INTMAX_C(0x8000000000000000))
#define INTMAX_MAX    (INTMAX_C(0x7FFFFFFFFFFFFFFF))
#define UINTMAX_MAX  (UINTMAX_C(0xFFFFFFFFFFFFFFFF))
//----------------------------------------------------------
#define PTRDIFF_MIN           (-0x80000000)
#define PTRDIFF_MAX             0x7FFFFFFF
//
#define SIG_ATOMIC_MIN        (-0x80000000)
```

```
#define SIG_ATOMIC_MAX          0x7FFFFFFF
//
#define SIZE_MAX                0xFFFFFFFFU
//
#define WCHAR_MIN               0x00000000
#define WCHAR_MAX               0xFFFFFFFFU
//
#define WINT_MIN              (-0x8000000000000000LL)
#define WINT_MAX                0x7FFFFFFFFFFFFFFFLL
//----------------------------------------------------------
#endif
```

95.1.14 lib/time_t.h

Si veda la sezione 91.3.

```
#ifndef _TIME_T_H
#define _TIME_T_H       1
//----------------------------------------------------------
typedef long long int time_t;
//----------------------------------------------------------
#endif
```

95.1.15 lib/wchar_t.h

Si veda la sezione 91.3.

```
#ifndef _WCHAR_T_H
#define _WCHAR_T_H      1
//----------------------------------------------------------
typedef unsigned int wchar_t;
//----------------------------------------------------------
#endif
```

95.2 os32: «lib/_gcc.h»

Si veda la sezione 88.1.

```
#ifndef __GCC_H
#define __GCC_H        1
//----------------------------------------------------------
#include <stdlib.h>
//----------------------------------------------------------
typedef struct
{
  unsigned long long int quot;
  unsigned long long int rem;
} ulldiv_t;
//----------------------------------------------------------
lldiv_t _lldiv (long long int dividend,
                long long int divisor);
ulldiv_t _ulldiv (unsigned long long int dividend,
                  unsigned long long int divisor);
//----------------------------------------------------------
unsigned long long int __udivdi3 (unsigned long long
                                  int dividend,
                                  unsigned long long
                                  int divisor);
unsigned long long int __umoddi3 (unsigned long long
                                  int dividend,
                                  unsigned long long
                                  int divisor);
long long int __divdi3 (long long int dividend,
                        long long int divisor);
long long int __moddi3 (long long int dividend,
                        long long int divisor);
//----------------------------------------------------------
#endif
```

95.2.1 lib/_gcc/__divdi3.c

Si veda la sezione 88.1.

```
#include <_gcc.h>
//----------------------------------------------------------
long long int
__divdi3 (long long int dividend, long long int divisor)
{
  lldiv_t result;
  result = _lldiv (dividend, divisor);
  return result.quot;
}
```

95.2.2 lib/_gcc/__moddi3.c

Si veda la sezione 88.1.

```
#include <_gcc.h>
//----------------------------------------------------------
long long int
__moddi3 (long long int dividend, long long int divisor)
{
  lldiv_t result;
  result = _lldiv (dividend, divisor);
  return result.rem;
}
```

95.2.3 lib/_gcc/__udivdi3.c

Si veda la sezione 88.1.

```
#include <_gcc.h>
//----------------------------------------------------------
unsigned long long int
__udivdi3 (unsigned long long int dividend,
           unsigned long long int divisor)
{
  ulldiv_t result;
  result = _ulldiv (dividend, divisor);
  return result.quot;
}
```

95.2.4 lib/_gcc/__umoddi3.c

Si veda la sezione 88.1.

```
#include <_gcc.h>
//----------------------------------------------------------
unsigned long long int
__umoddi3 (unsigned long long int dividend,
           unsigned long long int divisor)
{
  ulldiv_t result;
  result = _ulldiv (dividend, divisor);
  return result.rem;
}
```

95.2.5 lib/_gcc/_lldiv.c

Si veda la sezione 88.1.

```
#include <_gcc.h>
//----------------------------------------------------------
// If DIVIDEND and DIVISOR have different sign,
// the QUOTIENT is negative.
```

```
//
// The REMINDER has the same sign as the DIVISOR.
//----------------------------------------------------------
lldiv_t
_lldiv (long long int dividend, long long int divisor)
{
  ulldiv_t uresult;
  lldiv_t result;
  //
  // Check for sign.
  //
  if (dividend >= 0 && divisor >= 0)
    {
      uresult = _ulldiv ((unsigned long long) dividend,
                         (unsigned long long) divisor);
      result.quot = uresult.quot;
      result.rem = uresult.rem;
    }
  else if (dividend < 0 && divisor < 0)
    {
      uresult =
        _ulldiv ((unsigned long long) -dividend,
                 (unsigned long long) -divisor);
      result.quot = uresult.quot;
      result.rem = -uresult.rem;
    }
  else if (dividend < 0 && divisor >= 0)
    {
      uresult =
        _ulldiv ((unsigned long long) -dividend,
                 (unsigned long long) divisor);
      result.quot = -uresult.quot;
      result.rem = uresult.rem;
    }
  else if (dividend >= 0 && divisor < 0)
    {
      uresult = _ulldiv ((unsigned long long) dividend,
                         (unsigned long long) -divisor);
      result.quot = uresult.quot;
      result.rem = -uresult.rem;
    }
  //
  return (result);
}
```

95.2.6 lib/_gcc/_ulldiv.c

Si veda la sezione 88.1.

```
#include <_gcc.h>
//----------------------------------------------------------
// DIVIDEND = DIVISOR * QUOTIENT + REMINDER
//
// If DIVISOR == 0,
// then QUOTIENT == 0 and REMINDER == DIVIDEND
//----------------------------------------------------------
ulldiv_t
_ulldiv (unsigned long long int dividend,
         unsigned long long int divisor)
{
  unsigned long long int sign;
  unsigned long long int mask;
  ulldiv_t result;
  int scroll;
  unsigned int size;    // Bits of a long long.
  //
  // Division of zero will return zero.
  //
  if (dividend == 0)
    {
      result.quot = 0;
      result.rem = 0;
      return (result);
    }
    //
    // Division by zero will return zero and all
    // reminder.
    //
    if (divisor == 0)
      {
        result.quot = 0;
        result.rem = dividend;
        return (result);
      }
    //
    // Calculate how much bits does have the type ‘long
    // long’.
    //
    size = 0;
    mask = ~0LL;
    //
    while (mask > 0)
      {
        size += 8;
        mask >>= 8;
      }
    //
    // Calculate the value for ‘sign’ that needs to have
    // the most
    // significant bit to one.
    //
    mask = ~0LL;
    mask >>= 1;
    sign = ~mask;
    //
    // Scroll divisor to the left, as long as the first
    // bit is zero.
    //
    for (scroll = 0; scroll < size; scroll++)
      {
        if (divisor & sign)
        {
          //
          // The most significant bit is one.
          //
          break;
        }
      //
      // The most significant bit is zero: scroll
      // left.
      //
      divisor <<= 1;
    }
  //
  //
  //
  result.quot = 0;
  result.rem = 0;
  //
  for (; scroll >= 0 && divisor > 0; scroll--)
    {
      result.quot <<= 1;
      if (dividend >= divisor)
        {
          result.quot |= 1LL;
          dividend -= divisor;
        }
      divisor >>= 1;
    }
  //
  result.rem = dividend;
  //
  return (result);
}
```

95.3 os32: «lib/arpa/inet.h»

Si veda la sezione 91.3.

```
#ifndef _ARPA_INET_H
#define _ARPA_INET_H       1
//----------------------------------------------------------
#include <stdint.h>
#include <sys/socklen_t.h>
//----------------------------------------------------------
uint32_t htonl (uint32_t host32);
uint16_t htons (uint16_t host16);
uint32_t ntohl (uint32_t net32);
uint16_t ntohs (uint16_t net16);
//----------------------------------------------------------
const char *inet_ntop (int family, const void *src,
                       char *dst, socklen_t size);
int inet_pton (int family, const char *src, void *dst);
//----------------------------------------------------------
#endif
```

95.3.1 lib/arpa/inet/htonl.c

Si veda la sezione 88.11.

```
#include <arpa/inet.h>
//----------------------------------------------------------
uint32_t
htonl (uint32_t host32)
{
  uint8_t *orig = (void *) &host32;
  union
  {
    uint32_t value;
    uint8_t b[4];
  } dest;
  //
  // Convert: must revert byte order.
  //
  dest.b[0] = orig[3];
  dest.b[1] = orig[2];
  dest.b[2] = orig[1];
  dest.b[3] = orig[0];
  //
  return (dest.value);
}
```

95.3.2 lib/arpa/inet/htons.c

Si veda la sezione 88.11.

```
#include <arpa/inet.h>
//----------------------------------------------------------
uint16_t
htons (uint16_t host16)
{
  uint8_t *orig = (void *) &host16;
  union
  {
    uint16_t value;
    uint8_t b[2];
  } dest;
  //
  // Convert: must revert byte order.
  //
  dest.b[0] = orig[1];
  dest.b[1] = orig[0];
  //
  return (dest.value);
}
```

95.3.3 lib/arpa/inet/inet_ntop.c

Si veda la sezione 88.66.

```
#include <arpa/inet.h>
#include <stdint.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
//----------------------------------------------------------
const char *
inet_ntop (int family, const void *src, char *dst,
           socklen_t size)
{
  //
  // Check family type: only IPv4 is available here.
  //
  if (family != AF_INET)
    {
      errset (EAFNOSUPPORT);
      return (NULL);
    }
  //
  // Check for NULL pointers.
  //
  if (src == NULL || dst == NULL)
    {
      errset (EINVAL);
      return (NULL);
    }
  //
  snprintf (dst, (size_t) size, "%i.%i.%i.%i",
            *((in_addr_t *) src) >> 0 & 0x000000FF,
            *((in_addr_t *) src) >> 8 & 0x000000FF,
            *((in_addr_t *) src) >> 16 & 0x000000FF,
            *((in_addr_t *) src) >> 24 & 0x000000FF);
  //
  // Return ok.
  //
  return (dst);
}
```

95.3.4 lib/arpa/inet/inet_pton.c

Si veda la sezione 88.67.

```
#include <arpa/inet.h>
#include <stdint.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
//-----------------------------------------------------------
#define INET_PTON_MAX_STRING_SIZE 31
//-----------------------------------------------------------
int
inet_pton (int family, const char *src, void *dst)
{
  char *t;
  int ipv4[4];
  int i;
  in_addr_t result;
  char source[INET_PTON_MAX_STRING_SIZE + 1];
```

```
    //
    // Check family type: only IPv4 is available here.
    //
    if (family != AF_INET)
      {
        errset (EAFNOSUPPORT);
        return (-1);
      }
    //
    // Check for NULL pointers.
    //
    if (src == NULL || dst == NULL)
      {
        errset (EINVAL);
        return (-1);
      }
    //
    // Check the source string size.
    //
    if (strlen (src) > INET_PTON_MAX_STRING_SIZE)
      {
        //
        // The IPv4 address scan is finished
        // prematurely:
        // return zero to tell that the address string
        // is
        // not correct.
        //
        return (0);
      }
    //
    // Copy the source address, to be able to modify
    // the string.
    //
    strcpy (source, src);
    //
    // Start ``tokenize'' the string: it is here
    // accepted also
    // the space as a delimiter.
    //
    t = strtok (source, ". ");
    //
    for (i = 0; i < 4 && t != NULL; i++)
      {
        ipv4[i] = atoi (t);
        if (ipv4[i] > 255 || ipv4[i] < 0)
          {
            //
            // An octet cannot have a value greater than
            // 255,
            // and cannot be negative.
            //
            break;
          }
        t = strtok (NULL, ". ");
      }
    //
    if (i < 4)
      {
        //
        // The IPv4 address scan is finished
        // prematurely:
        // return zero to tell that the address string
        // is
        // not correct.
        //
        return (0);
      }
    //
    // Translate into a network byte order IPv4 address:
    // the architecture is little-endian.
    //
    result = 0;
    result += (ipv4[0] << 0) & 0x000000FF;
```

```
  result += (ipv4[1] << 8) & 0x0000FF00;
  result += (ipv4[2] << 16) & 0x00FF0000;
  result += (ipv4[3] << 24) & 0xFF000000;
  //
  // Update the destination.
  //
  *((in_addr_t *) dst) = result;
  //
  // Return ok.
  //
  return (1);
}
```

95.3.5 lib/arpa/inet/ntohl.c

Si veda la sezione 88.11.

```
#include <arpa/inet.h>
//----------------------------------------------------------
uint32_t
ntohl (uint32_t net32)
{
  uint8_t *orig = (void *) &net32;
  union
  {
    uint32_t value;
    uint8_t b[4];
  } dest;
  //
  // Convert: must revert byte order.
  //
  dest.b[0] = orig[3];
  dest.b[1] = orig[2];
  dest.b[2] = orig[1];
  dest.b[3] = orig[0];
  //
  return (dest.value);
}
```

95.3.6 lib/arpa/inet/ntohs.c

Si veda la sezione 88.11.

```
#include <arpa/inet.h>
//----------------------------------------------------------------------
uint16_t
ntohs (uint16_t net16)
{
  uint8_t *orig = (void *) &net16;
  union
  {
    uint16_t value;
    uint8_t b[2];
  } dest;
  //
  // Convert: must revert byte order.
  //
  dest.b[0] = orig[1];
  dest.b[1] = orig[0];
  //
  return (dest.value);
}
```

95.4 os32: «lib/dirent.h»

Si veda la sezione 91.3.

```
#ifndef _DIRENT_H
#define _DIRENT_H        1

#include <sys/types.h>  // ino_t
#include <limits.h>     // NAME_MAX

```

```
//-----------------------------------------------------------
struct dirent
{
  ino_t d_ino;  // I-node number [1]
  char d_name[NAME_MAX + 1];    // NAME_MAX + Null
  // termination
} __attribute__ ((packed));
//
// [1] The type 'ino_t' must be equal to 'uint16_t',
//     because the directory inside the Minix 1 file
//     system has exactly such size.
//
//-----------------------------------------------------------
#define DOPEN_MAX   OPEN_MAX/2  // <limits.h> [1]
//
// [1] DOPEN_MAX is not standard, but it is used to
//     define how many directory slot to keep for open
//     directories. As directory streams are opened as
//     file descriptors, the sum of all kind of file
//     open cannot be more than OPEM_MAX.
//-----------------------------------------------------------
typedef struct
{
  int fdn;      // File descriptor number.
  struct dirent dir;    // Last directory item read.
} DIR;

extern DIR _directory_stream[]; // Defined inside
                                // 'lib/dirent/DIR.c'.
//-----------------------------------------------------------
// Function prototypes.
//-----------------------------------------------------------
int closedir (DIR * dp);
DIR *opendir (const char *name);
struct dirent *readdir (DIR * dp);
void rewinddir (DIR * dp);
//-----------------------------------------------------------
```

```

#endif
```

95.4.1 lib/dirent/DIR.c

Si veda la sezione 91.3.

```
#include <dirent.h>
//
// There must be room for at least 'DOPEN_MAX'
// elements.
//
DIR _directory_stream[DOPEN_MAX];

void
_dirent_directory_stream_setup (void)
{
    int d;
    //
    for (d = 0; d < DOPEN_MAX; d++)
      {
        _directory_stream[d].fdn = -1;
      }
}
```

95.4.2 lib/dirent/closedir.c

Si veda la sezione 88.13.

```
#include <dirent.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stddef.h>
//----------------------------------------------------------
int
closedir (DIR * dp)
{
  //
  // Check for a valid argument
  //
  if (dp == NULL)
    {
      //
      // Not a valid pointer.
      //
      errset (EBADF);   // Invalid directory.
      return (-1);
    }
  //
  // Check if it is an open directory stream.
  //
  if (dp->fdn < 0)
    {
      //
      // The stream is closed.
      //
      errset (EBADF);   // Invalid directory.
      return (-1);
    }
  //
```

```
    // Close the file descriptor. It there is an error,
    // the 'errno' variable will be set by 'close()'.
    //
    return (close (dp->fdn));
}
```

95.4.3 lib/dirent/opendir.c

«

Si veda la sezione 88.89.

```
#include <dirent.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stddef.h>
//----------------------------------------------------------------------
DIR *
opendir (const char *path)
{
    int fdn;
    int d;
    DIR *dp;
    struct stat file_status;
    //
    // Function 'opendir()' is used only for reading.
    //
    fdn = open (path, O_RDONLY);
    //
    // Check the file descriptor returned.
    //
    if (fdn < 0)
      {
        //
        // The variable 'errno' is already set:
```

```
        // EINVAL
        // EMFILE
        // ENFILE
        //
        errset (errno);
        return (NULL);
      }
    //
    // Set the ‘FD_CLOEXEC’ flag for that file
    // descriptor.
    //
    if (fcntl (fdn, F_SETFD, FD_CLOEXEC) != 0)
      {
        //
        // The variable ‘errno’ is already set:
        // EBADF
        //
        errset (errno);
        close (fdn);
        return (NULL);
      }
    //
    //
    //
    if (fstat (fdn, &file_status) != 0)
      {
        //
        // Error should be already set.
        //
        errset (errno);
        close (fdn);
        return (NULL);
      }
    //
    // Verify it is a directory
    //
    if (!S_ISDIR (file_status.st_mode))
    {
      //
      // It is not a directory!
      //
      close (fdn);
      errset (ENOTDIR); // Is not a directory.
      return (NULL);
    }
  //
  // A valid file descriptor is available: must find a
  // free
  // '_directory_stream[]' slot.
  //
  for (d = 0; d < DOPEN_MAX; d++)
    {
      if (_directory_stream[d].fdn < 0)
        {
          //
          // Found a free slot: set it up.
          //
          dp = &(_directory_stream[d]);
          dp->fdn = fdn;
          //
          // Return the directory pointer.
          //
          return (dp);
        }
    }
  //
  // If we are here, there was no free directory slot
  // available.
  //
  close (fdn);
  errset (EMFILE);      // Too many file open.
  return (NULL);
}
```

95.4.4 lib/dirent/readdir.c

«

Si veda la sezione 88.98.

```
#include <dirent.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stddef.h>
//----------------------------------------------------------
struct dirent *
readdir (DIR * dp)
{
  ssize_t size;
  //
  // Check for a valid argument.
  //
  if (dp == NULL)
    {
      //
      // Not a valid pointer.
      //
      errset (EBADF);   // Invalid directory.
      return (NULL);
    }
  //
  // Check if it is an open directory stream.
  //
  if (dp->fdn < 0)
    {
      //
      // The stream is closed.
      //
      errset (EBADF);   // Invalid directory.
      return (NULL);
    }
  //
  // Read the directory.
  //
  size = read (dp->fdn, &(dp->dir), (size_t) 16);
  //
  // Fix the null termination, if the name is very
  // long.
  //
  dp->dir.d_name[NAME_MAX] = '\0';
  //
  // Check what was read.
  //
  if (size == 0)
    {
      //
      // End of directory, but it is not an error.
      //
      return (NULL);
    }
  //
  if (size < 0)
    {
      //
      // This is an error. The variable 'errno' is
      // already set.
      //
      errset (errno);
      return (NULL);
    }
  //
  if (dp->dir.d_ino == 0)
    {
      //
      // This is a null directory record.
      // Should try to read the next one.
      //
      return (readdir (dp));
    }
  //
  if (strlen (dp->dir.d_name) == 0)
    {
      //
      // This is a bad directory record: try to read
      // next.
      //
      return (readdir (dp));
    }
  //
  // A valid directory record should be available now.
  //
  return (&(dp->dir));
}
```

95.4.5 lib/dirent/rewinddir.c

Si veda la sezione 88.101.

```
#include <dirent.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <stddef.h>
#include <stdio.h>
//----------------------------------------------------------
void
rewinddir (DIR * dp)
{
  FILE *fp;
  //
  // Check for a valid argument.
  //
  if (dp == NULL)
```

```
    {
      //
      // Nothing to rewind, and no error to set.
      //
      return;
    }
  //
  // Check if it is an open directory stream.
  //
  if (dp->fdn < 0)
    {
      //
      // The stream is closed.
      // Nothing to rewind, and no error to set.
      //
      return;
    }
  //
  //
  //
  fp = &_stream[dp->fdn];
  //
  rewind (fp);
}
```

95.5 os32: «lib/errno.h»

Si veda la sezione 88.20.

```
#ifndef _ERRNO_H
#define _ERRNO_H       1
//----------------------------------------------------------
#include <limits.h>
#include <string.h>
#include <sys/os32.h>
#include <kernel/lib_k.h>

```

```
//----------------------------------------------------------
// The variable 'errno' is standard, but 'errln' and
// 'errfn' are added to keep track of the error source.
// Variable 'errln' is used to save the source file
// line number; variable 'errfn' is used to save the
// source file name. To set these variable in a
// consistent way it is also added a macroinstruction:
// 'errset'.
//----------------------------------------------------------
extern int errno;
extern int errln;
extern char errfn[PATH_MAX];
//
#define errset(e) \
    (errln = __LINE__, \
     strncpy (errfn, __FILE__, PATH_MAX), \
     errno = e)
//----------------------------------------------------------
// Standard POSIX 'errno' macro variables.
//----------------------------------------------------------
#define E2BIG            1      // Argument list too
                                // long.
#define EACCES           2      // Permission denied.
#define EADDRINUSE       3      // Address in use.
#define EADDRNOTAVAIL    4      // Address not
                                // available.
#define EAFNOSUPPORT     5      // Address family not
                                // supported.
#define EAGAIN           6      // Resource
                                // unavailable, try
                                // again.
#define EALREADY         7      // Connection already
                                // in progress.
#define EBADF            8      // Bad file
                                // descriptor.
#define EBADMSG          9      // Bad message.
#define EBUSY           10      // Device or resource
```

```
                                // busy.
#define ECANCELED       11      // Operation canceled.
#define ECHILD          12      // No child processes.
#define ECONNABORTED    13      // Connection aborted.
#define ECONNREFUSED    14      // Connection refused.
#define ECONNRESET      15      // Connection reset.
#define EDEADLK         16      // Resource deadlock
                                // would occur.
#define EDESTADDRREQ    17      // Destination address
                                // required.
#define EDOM            18      // Mathematics
                                // argument out of
                                // domain of
                                // function.
#define EDQUOT          19      // Reserved.
#define EEXIST          20      // File exists.
#define EFAULT          21      // Bad address.
#define EFBIG           22      // File too large.
#define EHOSTUNREACH    23      // Host is
                                // unreachable.
#define EIDRM           24      // Identifier removed.
#define EILSEQ          25      // Illegal byte
                                // sequence.
#define EINPROGRESS     26      // Operation in
                                // progress.
#define EINTR           27      // Interrupted
                                // function.
#define EINVAL          28      // Invalid argument.
#define EIO             29      // I/O error.
#define EISCONN         30      // Socket is
                                // connected.
#define EISDIR          31      // Is a directory.
#define ELOOP           32      // Too many levels of
                                // symbolic links.
#define EMFILE          33      // Too many open
                                // files.
#define EMLINK          34      // Too many links.
```

```
#define EMSGSIZE        35      // Message too large.
#define EMULTIHOP       36      // Reserved.
#define ENAMETOOLONG    37      // Filename too long.
#define ENETDOWN        38      // Network is down.
#define ENETRESET       39      // Connection aborted
                                // by network.
#define ENETUNREACH     40      // Network
                                // unreachable.
#define ENFILE          41      // Too many files open
                                // in system.
#define ENOBUFS         42      // No buffer space
                                // available.
#define ENODATA         43      // No message is
                                // available on the
                                // stream head
                                // read queue.
#define ENODEV          44      // No such device.
#define ENOENT          45      // No such file or
                                // directory.
#define ENOEXEC         46      // Executable file
                                // format error.
#define ENOLCK          47      // No locks available.
#define ENOLINK         48      // Reserved.
#define ENOMEM          49      // Not enough space.
#define ENOMSG          50      // No message of the
                                // desired type.
#define ENOPROTOOPT     51      // Protocol not
                                // available.
#define ENOSPC          52      // No space left on
                                // device.
#define ENOSR           53      // No stream
                                // resources.
#define ENOSTR          54      // Not a stream.
#define ENOSYS          55      // Function not
                                // supported.
#define ENOTCONN        56      // The socket is not
                                // connected.
```

```
#define ENOTDIR         57      // Not a directory.
#define ENOTEMPTY       58      // Directory not
                                // empty.
#define ENOTSOCK        59      // Not a socket.
#define ENOTSUP         60      // Not supported.
#define ENOTTY          61      // Inappropriate I/O
                                // control operation.
#define ENXIO           62      // No such device or
                                // address.
#define EOPNOTSUPP      63      // Operation not
                                // supported on
                                // socket.
#define EOVERFLOW       64      // Value too large to
                                // be stored in data
                                // type.
#define EPERM           65      // Operation not
                                // permitted.
#define EPIPE           66      // Broken pipe.
#define EPROTO          67      // Protocol error.
#define EPROTONOSUPPORT 68      // Protocol not
                                // supported.
#define EPROTOTYPE      69      // Protocol wrong type
                                // for socket.
#define ERANGE          70      // Result too large.
#define EROFS           71      // Read-only file
                                // system.
#define ESPIPE          72      // Invalid seek.
#define ESRCH           73      // No such process.
#define ESTALE          74      // Reserved.
#define ETIME           75      // Stream ioctl()
                                // timeout.
#define ETIMEDOUT       76      // Connection timed
                                // out.
#define ETXTBSY         77      // Text file busy.
#define EWOULDBLOCK     78      // Operation would
                                // block (may be the
                                // same as EAGAIN).
```

```
#define EXDEV           79      // Cross-device link.
//----------------------------------------------------------
// Added os32 errors.
//----------------------------------------------------------
#define EUNKNOWN                   (-1) // Unknown
                                        // error.
#define E_NO_MEDIUM                 80  // No medium
                                        // found.
#define E_MEDIUM                    81  // Medium
                                        // reported
                                        // error.
#define E_FILE_TYPE                 82  // File type
                                        // not
                                        // compatible.
#define E_ROOT_INODE_NOT_CACHED     83  // The root
                                        // directory
                                        // inode is
                                        // not cached.
#define E_CANNOT_READ_SUPERBLOCK    84  // Cannot read
                                        // super
                                        // block.
#define E_MAP_INODE_TOO_BIG         85  // Map inode
                                        // too big.
#define E_MAP_ZONE_TOO_BIG          86  // Map zone
                                        // too big.
#define E_DATA_ZONE_TOO_BIG         87  // Data zone
                                        // too big.
#define E_CANNOT_FIND_ROOT_DEVICE   88  // Cannot find
                                        // root
                                        // device.
#define E_CANNOT_FIND_ROOT_INODE    89  // Cannot find
                                        // root inode.
#define E_FILE_TYPE_UNSUPPORTED     90  // File type
                                        // unsupported.
#define E_ENV_TOO_BIG               91  // Environment
                                        // too big.
#define E_LIMIT                     92  // Exceeded
```

```
                                        // implementa-
                                        // tion limits.
#define E_NOT_MOUNTED               93  // Not
                                        // mounted.
#define E_NOT_IMPLEMENTED           94  // Not
                                        // implemented.
#define E_HARDWARE_FAULT            95  // Hardware
                                        // fault.
#define E_DRIVER_FAULT              96  // Driver
                                        // fault.
#define E_PIPE_FULL                 97  // Pipe full.
#define E_PIPE_EMPTY                98  // Pipe empty.
#define E_PART_TYPE_NOT_MINIX       99  // Not a Minix
                                        // partition
                                        // type.
#define E_FS_TYPE_NOT_SUPPORTED    100  // File system
                                        // type not
                                        // supported.
#define E_PDU_TOO_BIG              101  // PDU too
                                        // big.
#define E_ARP_MISSING              102  // ARP missing
                                        // address.
//----------------------------------------------------------
// Default descriptions for errors.
//----------------------------------------------------------
#define TEXT_E2BIG           "Argument list too long."
#define TEXT_EACCES          "Permission denied."
#define TEXT_EADDRINUSE      "Address in use."
#define TEXT_EADDRNOTAVAIL   "Address not available."
#define TEXT_EAFNOSUPPORT    "Address family not " \
                             "supported."
#define TEXT_EAGAIN          "Resource unavailable, " \
                             "try again."
#define TEXT_EALREADY        "Connection already in " \
                             "progress."
#define TEXT_EBADF           "Bad file descriptor."
#define TEXT_EBADMSG         "Bad message."
```

```
#define TEXT_EBUSY           "Device or resource busy."
#define TEXT_ECANCELED       "Operation canceled."
#define TEXT_ECHILD          "No child processes."
#define TEXT_ECONNABORTED    "Connection aborted."
#define TEXT_ECONNREFUSED    "Connection refused."
#define TEXT_ECONNRESET      "Connection reset."
#define TEXT_EDEADLK         "Resource deadlock " \
                             "would occur."
#define TEXT_EDESTADDRREQ    "Destination address " \
                             "required."
#define TEXT_EDOM            "Mathematics argument " \
                             "out of " \
                             "domain of function."
#define TEXT_EDQUOT          "Reserved error: EDQUOT"
#define TEXT_EEXIST          "File exists."
#define TEXT_EFAULT          "Bad address."
#define TEXT_EFBIG           "File too large."
#define TEXT_EHOSTUNREACH    "Host is unreachable."
#define TEXT_EIDRM           "Identifier removed."
#define TEXT_EILSEQ          "Illegal byte sequence."
#define TEXT_EINPROGRESS     "Operation in progress."
#define TEXT_EINTR           "Interrupted function."
#define TEXT_EINVAL          "Invalid argument."
#define TEXT_EIO             "I/O error."
#define TEXT_EISCONN         "Socket is connected."
#define TEXT_EISDIR          "Is a directory."
#define TEXT_ELOOP           "Too many levels of " \
                             "symbolic links."
#define TEXT_EMFILE          "Too many open files."
#define TEXT_EMLINK          "Too many links."
#define TEXT_EMSGSIZE        "Message too large."
#define TEXT_EMULTIHOP       "Reserved error: " \
                             "EMULTIHOP"
#define TEXT_ENAMETOOLONG    "Filename too long."
#define TEXT_ENETDOWN        "Network is down."
#define TEXT_ENETRESET       "Connection aborted by " \
                             "network."
```

```
#define TEXT_ENETUNREACH     "Network unreachable."
#define TEXT_ENFILE          "Too many files open " \
                             "in system."
#define TEXT_ENOBUFS         "No buffer space " \
                             "available."
#define TEXT_ENODATA         "No message is " \
                             "available on the " \
                             "stream head read queue."
#define TEXT_ENODEV          "No such device."
#define TEXT_ENOENT          "No such file or " \
                             "directory."
#define TEXT_ENOEXEC         "Executable file " \
                             "format error."
#define TEXT_ENOLCK          "No locks available."
#define TEXT_ENOLINK         "Reserved error: ENOLINK"
#define TEXT_ENOMEM          "Not enough space."
#define TEXT_ENOMSG          "No message of the " \
                             "desired type."
#define TEXT_ENOPROTOOPT     "Protocol not available."
#define TEXT_ENOSPC          "No space left on device."
#define TEXT_ENOSR           "No stream resources."
#define TEXT_ENOSTR          "Not a stream."
#define TEXT_ENOSYS          "Function not supported."
#define TEXT_ENOTCONN        "The socket is not " \
                             "connected."
#define TEXT_ENOTDIR         "Not a directory."
#define TEXT_ENOTEMPTY       "Directory not empty."
#define TEXT_ENOTSOCK        "Not a socket."
#define TEXT_ENOTSUP         "Not supported."
#define TEXT_ENOTTY          "Inappropriate I/O " \
                             "control operation."
#define TEXT_ENXIO           "No such device or " \
                             "address."
#define TEXT_EOPNOTSUPP      "Operation not " \
                             "supported on socket."
#define TEXT_EOVERFLOW       "Value too large to be " \
                             "stored in data type."
```

```
#define TEXT_EPERM           "Operation not permitted."
#define TEXT_EPIPE           "Broken pipe."
#define TEXT_EPROTO          "Protocol error."
#define TEXT_EPROTONOSUPPORT "Protocol not supported."
#define TEXT_EPROTOTYPE      "Protocol wrong type " \
                             "for socket."
#define TEXT_ERANGE          "Result too large."
#define TEXT_EROFS           "Read-only file system."
#define TEXT_ESPIPE          "Invalid seek."
#define TEXT_ESRCH           "No such process."
#define TEXT_ESTALE          "Reserved error: ESTALE"
#define TEXT_ETIME           "Stream ioctl() timeout."
#define TEXT_ETIMEDOUT       "Connection timed out."
#define TEXT_ETXTBSY         "Text file busy."
#define TEXT_EWOULDBLOCK     "Operation would block."
#define TEXT_EXDEV           "Cross-device link."
//----------------------------------------------------------
#define TEXT_EUNKNOWN                   \
    "Unknown error."
#define TEXT_E_NO_MEDIUM                \
    "No medium found."
#define TEXT_E_MEDIUM                   \
    "Medium reported error"
#define TEXT_E_FILE_TYPE                \
    "File type not compatible."
#define TEXT_E_ROOT_INODE_NOT_CACHED    \
    "The root directory inode is not cached."
#define TEXT_E_CANNOT_READ_SUPERBLOCK   \
    "Cannot read super block."
#define TEXT_E_MAP_INODE_TOO_BIG        \
    "Map inode too big."
#define TEXT_E_MAP_ZONE_TOO_BIG         \
    "Map zone too big."
#define TEXT_E_DATA_ZONE_TOO_BIG        \
    "Data zone too big."
#define TEXT_E_CANNOT_FIND_ROOT_DEVICE  \
    "Cannot find root device."
```

```
#define TEXT_E_CANNOT_FIND_ROOT_INODE   \
    "Cannot find root inode."
#define TEXT_E_FILE_TYPE_UNSUPPORTED    \
    "File type unsupported."
#define TEXT_E_ENV_TOO_BIG              \
    "Environment too big."
#define TEXT_E_LIMIT                    \
    "Exceeded implementation limits."
#define TEXT_E_NOT_MOUNTED              \
    "Not mounted."
#define TEXT_E_NOT_IMPLEMENTED          \
    "Not implemented."
#define TEXT_E_HARDWARE_FAULT           \
    "Hardware fault."
#define TEXT_E_DRIVER_FAULT             \
    "Driver fault."
#define TEXT_E_PIPE_FULL                \
    "Pipe full."
#define TEXT_E_PIPE_EMPTY               \
    "Pipe empty."
#define TEXT_E_PART_TYPE_NOT_MINIX      \
    "Not a Minix partition type."
#define TEXT_E_FS_TYPE_NOT_SUPPORTED    \
    "File system type not supported."
#define TEXT_E_PDU_TOO_BIG              \
    "PDU too big."
#define TEXT_E_ARP_MISSING              \
    "ARP missing address."
//----------------------------------------------------------
#endif
```

95.5.1 lib/errno/errno.c 1846

95.5.1 lib/errno/errno.c

Si veda la sezione 88.20.

```
//----------------------------------------------------------
// This file does not include the 'errno.h' header,
// because here 'errno' should not be declared as an
// extern variable!
//----------------------------------------------------------
#include <limits.h>
//----------------------------------------------------------
// The variable 'errno' is standard, but 'errln' and
// 'errfn' are added to keep track of the error source.
// Variable 'errln' is used to save the source file
// line number; variable 'errfn' is used to save the
// source file name.
// To set these variable in a consistent way it is
// also added a macroinstruction: 'errset'.
//----------------------------------------------------------
int errno;
int errln;
char errfn[PATH_MAX];
//----------------------------------------------------------
```

95.6 os32: «lib/fcntl.h»

Si veda la sezione 91.3.

```
#ifndef _FCNTL_H
#define _FCNTL_H        1

#include <sys/types.h>  // mode_t
                        // off_t
                        // pid_t
//----------------------------------------------------------
// Values for the second parameter of function
// 'fcntl()'.
//----------------------------------------------------------
```

```
#define F_DUPFD         0       // Duplicate file
                                // descriptor.
#define F_GETFD         1       // Get file descriptor
                                // flags.
#define F_SETFD         2       // Set file descriptor
                                // flags.
#define F_GETFL         3       // Get file status
                                // flags.
#define F_SETFL         4       // Set file status
                                // flags.
#define F_GETLK         5       // Get record locking
                                // information.
#define F_SETLK         6       // Set record locking
                                // information.
#define F_SETLKW        7       // Set record locking
                                // information;
                                // wait if blocked.
#define F_GETOWN        8       // Set owner of
                                // socket.
#define F_SETOWN        9       // Get owner of
                                // socket.
//----------------------------------------------------------
// Flags to be set with:
//   fcntl (fd, F_SETFD, ...);
//----------------------------------------------------------
#define FD_CLOEXEC      1       // Close the file
                                // descriptor upon
                                // execution of an
                                // exec() family
                                // function.
//----------------------------------------------------------
// Values for type `l_type', used for record locking
// with `fcntl()'.
//----------------------------------------------------------
#define F_RDLCK         0       // Read lock.
#define F_WRLCK         1       // Write lock.
#define F_UNLCK         2       // Remove lock.
```

```
//-----------------------------------------------------------
// Flags for file creation, in place of 'oflag'
// parameter for function 'open()'.
//-----------------------------------------------------------
#define O_CREAT         000010  // Create file if it
                                // does not exist.
#define O_EXCL          000020  // Exclusive use flag.
#define O_NOCTTY        000040  // Do not assign a
                                // controlling
                                // terminal.
#define O_TRUNC         000100  // Truncation flag.
//-----------------------------------------------------------
// Flags for the file status, used with 'open()' and
// 'fcntl()'.
//-----------------------------------------------------------
#define O_APPEND        000200  // Write append.
#define O_DSYNC         000400  // Synchronized write
                                // operations.
#define O_NONBLOCK      001000  // Non-blocking mode.
#define O_RSYNC         002000  // Synchronized read
                                // operations.
#define O_SYNC          004000  // Synchronized read
                                // and write.
//-----------------------------------------------------------
// File access mask selection.
//-----------------------------------------------------------
#define O_ACCMODE       000003  // Mask to select the
                                // last three bits,
                                // used to specify the
                                // main access
                                // modes: read, write
                                // and both.
//-----------------------------------------------------------
// Main access modes.
//-----------------------------------------------------------
#define O_RDONLY        000001  // Read.
#define O_WRONLY        000002  // Write.
```

```
#define O_RDWR         (O_RDONLY | O_WRONLY)    // [1]
//
// [1]  Both read and write.
//
//----------------------------------------------------------
// Structure 'flock', used to file lock for POSIX
// standard. It is not used inside os32.
//----------------------------------------------------------
struct flock
{
  short int l_type;     // Type of lock: F_RDLCK,
  // F_WRLCK, or F_UNLCK.
  short int l_whence;   // Start reference point.
  off_t l_start;        // Offset, from 'l_whence',
  // for the area start.
  off_t l_len;  // Locked area size. Zero means up to
  // the end of the file.
  pid_t l_pid;  // The process id blocking the area.
};
//----------------------------------------------------------
// Function prototypes.
//----------------------------------------------------------
int creat (const char *path, mode_t mode);
int fcntl (int fdn, int cmd, ...);
int open (const char *path, int oflags, ...);
//----------------------------------------------------------

#endif
```

95.6.1 lib/fcntl/creat.c

Si veda la sezione 88.14.

```
#include <fcntl.h>
#include <sys/types.h>
//----------------------------------------------------------
int
creat (const char *path, mode_t mode)
{
  return (open (path, O_WRONLY | O_CREAT | O_TRUNC, mode));
}
```

95.6.2 lib/fcntl/fcntl.c

Si veda la sezione 87.18.

```
#include <fcntl.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <sys/os32.h>
#include <limits.h>
//----------------------------------------------------------
int
fcntl (int fdn, int cmd, ...)
{
  va_list ap;
  sysmsg_fcntl_t msg;
  va_start (ap, cmd);
  //
  // Well known arguments.
  //
  msg.fdn = fdn;
  msg.cmd = cmd;
  //
  // Select other arguments.
  //
  switch (cmd)
    {
    case F_DUPFD:
    case F_SETFD:
    case F_SETFL:
      msg.arg = va_arg (ap, int);
      break;
    case F_GETFD:
    case F_GETFL:
      break;
    case F_GETOWN:
    case F_SETOWN:
    case F_GETLK:
    case F_SETLK:
    case F_SETLKW:
      errset (E_NOT_IMPLEMENTED);       // Not
      // implemented.
      return (-1);
    default:
      errset (EINVAL);  // Not implemented.
      return (-1);
    }
  //
  // Do the system call.
  //
  sys (SYS_FCNTL, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.6.3 lib/fcntl/open.c

Si veda la sezione 87.37.

```
#include <fcntl.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <sys/os32.h>
#include <limits.h>
//----------------------------------------------------------
int
open (const char *path, int oflags, ...)
{
  va_list ap;
  sysmsg_open_t msg;
  va_start (ap, oflags);
  msg.path = path;
  msg.flags = oflags;
  msg.mode = va_arg (ap, mode_t);
  sys (SYS_OPEN, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.7 os32: «lib/grp.h»

Si veda la sezione 91.3.

```
#ifndef _GRP_H
#define _GRP_H        1
//----------------------------------------------------------
#include <restrict.h>
#include <sys/types.h>  // gid_t, uid_t
//----------------------------------------------------------
```

```
#define GR_MEM_MAX 32
struct group
{
  char *gr_name;
  char *gr_passwd;
  gid_t gr_gid;
  char *gr_mem[GR_MEM_MAX];
};
//----------------------------------------------------------
struct group *getgrent (void);
void setgrent (void);
void endgrent (void);
struct group *getgrnam (const char *name);
struct group *getgrgid (gid_t gid);
//----------------------------------------------------------
#endif
```

95.7.1 lib/grp/grent.c

Si veda la sezione 88.53.

```
#include <grp.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//----------------------------------------------------------
static char buffer[BUFSIZ];
static struct group gr;
static FILE *fp = NULL;
//----------------------------------------------------------
struct group *
getgrent (void)
{
  void *pstatus;
  char *char_gid;
  int i;
  //
  if (fp == NULL)
    {
      fp = fopen ("/etc/group", "r");
      if (fp == NULL)
        {
          return NULL;
        }
    }
  //
  pstatus = fgets (buffer, BUFSIZ, fp);
  if (pstatus == NULL)
    {
      return (NULL);
    }
  //
  // The parse is made with ‘strtok()’. Please notice
  // that
  // ‘strtok()’ will not parse a line like the
  // following:
  // user::233:
  // The password field *must* have something,
  // otherwise the
  // GID will take the password place.
  // ‘strtok()’ will consider ‘::’ the same as ‘:’!
  //
  gr.gr_name = strtok (buffer, ":");
  gr.gr_passwd = strtok (NULL, ":");
  char_gid = strtok (NULL, ":");
  for (i = 0; i < GR_MEM_MAX; i++)
    {
      gr.gr_mem[i] = strtok (NULL, ",\n");
    }
  gr.gr_gid = (gid_t) atoi (char_gid);
  //
  return (&gr);
```

```
}

//-----------------------------------------------------------
void
endgrent (void)
{
  int status;
  //
  if (fp != NULL)
    {
      status = fclose (fp);
      if (status != 0)
        {
          perror (NULL);
          fp = NULL;
        }
    }
}

//-----------------------------------------------------------
void
setgrent (void)
{
  if (fp != NULL)
    {
      rewind (fp);
    }
}

//-----------------------------------------------------------
struct group *
getgrnam (const char *name)
{
  struct group *gr;
  //
  setgrent ();
  //
  for (;;)
    {
      gr = getgrent ();
      if (gr == NULL)
        {
          return (NULL);
        }
      if (strcmp (gr->gr_name, name) == 0)
        {
          return (gr);
        }
    }
}

//----------------------------------------------------------
struct group *
getgrgid (gid_t gid)
{
  struct group *gr;
  //
  setgrent ();
  //
  for (;;)
    {
      gr = getgrent ();
      if (gr == NULL)
        {
          return (NULL);
        }
      if (gr->gr_gid == gid)
        {
          return (gr);
        }
    }
}
```

95.8 os32: «lib/inttypes.h»

Si veda la sezione 91.3.

```
#ifndef _INTTYPES_H
#define _INTTYPES_H     1
//----------------------------------------------------------
#include <stdint.h>
#include <wchar_t.h>
#include <restrict.h>
//----------------------------------------------------------
typedef struct
{
  intmax_t quot;
  intmax_t rem;
} imaxdiv_t;
//
imaxdiv_t imaxdiv (intmax_t numer, intmax_t denom);
//----------------------------------------------------------
// Output typesetting.
//----------------------------------------------------------
#define PRId8           "d"
#define PRId16          "d"
#define PRId32          "d"
#define PRId64          "lld"
//
#define PRIdLEAST8      "d"
#define PRIdLEAST16     "d"
#define PRIdLEAST32     "d"
#define PRIdLEAST64     "lld"
//
#define PRIdFAST8       "d"
#define PRIdFAST16      "d"
#define PRIdFAST32      "d"
#define PRIdFAST64      "lld"
//
#define PRIdMAX         "lld"
#define PRIdPTR         "d"
```

```
//
#define PRIi8           "i"
#define PRIi16          "i"
#define PRIi32          "i"
#define PRIi64          "lli"
//
#define PRIiLEAST8      "i"
#define PRIiLEAST16     "i"
#define PRIiLEAST32     "i"
#define PRIiLEAST64     "lli"
//
#define PRIiFAST8       "i"
#define PRIiFAST16      "i"
#define PRIiFAST32      "i"
#define PRIiFAST64      "lli"
//
#define PRIiMAX         "lli"
#define PRIiPTR         "i"
//
#define PRIb8           "b"     // PRIb... is not
                                // standard!
#define PRIb16          "b"     //
#define PRIb32          "b"     //
#define PRIb64          "llb"   //
//                              //
#define PRIbLEAST8      "b"     //
#define PRIbLEAST16     "b"     //
#define PRIbLEAST32     "b"     //
#define PRIbLEAST64     "llb"   //
//                              //
#define PRIbFAST8       "b"     //
#define PRIbFAST16      "b"     //
#define PRIbFAST32      "b"     //
#define PRIbFAST64      "llb"   //
//                              //
#define PRIbMAX         "llb"   //
#define PRIbPTR         "b"     //
```

```
//
#define PRIo8           "o"
#define PRIo16          "o"
#define PRIo32          "o"
#define PRIo64          "llo"
//
#define PRIoLEAST8      "o"
#define PRIoLEAST16     "o"
#define PRIoLEAST32     "o"
#define PRIoLEAST64     "llo"
//
#define PRIoFAST8       "o"
#define PRIoFAST16      "o"
#define PRIoFAST32      "o"
#define PRIoFAST64      "llo"
//
#define PRIoMAX         "llo"
#define PRIoPTR         "o"
//
#define PRIu8           "u"
#define PRIu16          "u"
#define PRIu32          "u"
#define PRIu64          "llu"
//
#define PRIuLEAST8      "u"
#define PRIuLEAST16     "u"
#define PRIuLEAST32     "u"
#define PRIuLEAST64     "llu"
//
#define PRIuFAST8       "u"
#define PRIuFAST16      "u"
#define PRIuFAST32      "u"
#define PRIuFAST64      "llu"
//
#define PRIuMAX         "llu"
#define PRIuPTR         "u"
//
```

```
#define PRIx8           "x"
#define PRIx16          "x"
#define PRIx32          "x"
#define PRIx64          "llx"
//
#define PRIxLEAST8      "x"
#define PRIxLEAST16     "x"
#define PRIxLEAST32     "x"
#define PRIxLEAST64     "llx"
//
#define PRIxFAST8       "x"
#define PRIxFAST16      "x"
#define PRIxFAST32      "x"
#define PRIxFAST64      "llx"
//
#define PRIxMAX         "llx"
#define PRIxPTR         "x"
//
#define PRIX8           "X"
#define PRIX16          "X"
#define PRIX32          "X"
#define PRIX64          "llX"
//
#define PRIXLEAST8      "X"
#define PRIXLEAST16     "X"
#define PRIXLEAST32     "X"
#define PRIXLEAST64     "llX"
//
#define PRIXFAST8       "X"
#define PRIXFAST16      "X"
#define PRIXFAST32      "X"
#define PRIXFAST64      "llX"
//
#define PRIXMAX         "llX"
#define PRIXPTR         "X"
//----------------------------------------------------------
// Input scan and evaluation.
```

```
//----------------------------------------------------------
#define SCNd8           "hhd"
#define SCNd16          "hd"
#define SCNd32          "d"
#define SCNd64          "lld"
//
#define SCNdLEAST8      "hhd"
#define SCNdLEAST16     "hd"
#define SCNdLEAST32     "d"
#define SCNdLEAST64     "lld"
//
#define SCNdFAST8       "hhd"
#define SCNdFAST16      "d"
#define SCNdFAST32      "d"
#define SCNdFAST64      "lld"
//
#define SCNdMAX         "lld"
#define SCNdPTR         "d"
//
#define SCNi8           "hhi"
#define SCNi16          "hi"
#define SCNi32          "i"
#define SCNi64          "lli"
//
#define SCNiLEAST8      "hhi"
#define SCNiLEAST16     "hi"
#define SCNiLEAST32     "i"
#define SCNiLEAST64     "lli"
//
#define SCNiFAST8       "hhi"
#define SCNiFAST16      "i"
#define SCNiFAST32      "i"
#define SCNiFAST64      "lli"
//
#define SCNiMAX         "lli"
#define SCNiPTR         "i"
//
```

```
#define SCNb8           "hhb"   // SCNb... is not
                                // standard!
#define SCNb16          "hb"    //
#define SCNb32          "b"     //
#define SCNb64          "llb"   //
//                              //
#define SCNbLEAST8      "hhb"   //
#define SCNbLEAST16     "hb"    //
#define SCNbLEAST32     "b"     //
#define SCNbLEAST64     "llb"   //
//                              //
#define SCNbFAST8       "hhb"   //
#define SCNbFAST16      "b"     //
#define SCNbFAST32      "b"     //
#define SCNbFAST64      "llb"   //
//                              //
#define SCNbMAX         "llb"   //
#define SCNbPTR         "b"     //
//
#define SCNo8           "hho"
#define SCNo16          "ho"
#define SCNo32          "o"
#define SCNo64          "llo"
//
#define SCNoLEAST8      "hho"
#define SCNoLEAST16     "ho"
#define SCNoLEAST32     "o"
#define SCNoLEAST64     "llo"
//
#define SCNoFAST8       "hho"
#define SCNoFAST16      "o"
#define SCNoFAST32      "o"
#define SCNoFAST64      "llo"
//
#define SCNoMAX         "llo"
#define SCNoPTR         "o"
//
```

```
#define SCNu8           "hhu"
#define SCNu16          "hu"
#define SCNu32          "u"
#define SCNu64          "llu"
//
#define SCNuLEAST8      "hhu"
#define SCNuLEAST16     "hu"
#define SCNuLEAST32     "u"
#define SCNuLEAST64     "llu"
//
#define SCNuFAST8       "hhu"
#define SCNuFAST16      "u"
#define SCNuFAST32      "u"
#define SCNuFAST64      "llu"
//
#define SCNuMAX         "llu"
#define SCNuPTR         "u"
//
#define SCNx8           "hhx"
#define SCNx16          "hx"
#define SCNx32          "x"
#define SCNx64          "llx"
//
#define SCNxLEAST8      "hhx"
#define SCNxLEAST16     "hx"
#define SCNxLEAST32     "x"
#define SCNxLEAST64     "llx"
//
#define SCNxFAST8       "hhx"
#define SCNxFAST16      "x"
#define SCNxFAST32      "x"
#define SCNxFAST64      "llx"
//
#define SCNxMAX         "llx"
#define SCNxPTR         "x"
//----------------------------------------------------------
intmax_t imaxabs (intmax_t j);
intmax_t strtoimax (const char *restrict nptr,
                    char **restrict endptr, int base);
uintmax_t strtouimax (const char *restrict nptr,
                      char **restrict endptr, int base);
intmax_t wcstoimax (const wchar_t * restrict nptr,
                    wchar_t ** restrict endptr, int base);
uintmax_t wcstouimax (const wchar_t * restrict nptr,
                      wchar_t ** restrict endptr, int base);
//----------------------------------------------------------
#endif
```

95.8.1 lib/inttypes/imaxabs.c

«

Si veda la sezione 88.3.

```
#include <inttypes.h>
//----------------------------------------------------------
intmax_t
imaxabs (intmax_t j)
{
  if (j < 0)
    {
      return -j;
    }
  else
    {
      return j;
    }
}
```

95.8.2 lib/inttypes/imaxdiv.c

Si veda la sezione 88.17.

```
#include <inttypes.h>
//----------------------------------------------------------
imaxdiv_t
imaxdiv (intmax_t numer, intmax_t denom)
{
  imaxdiv_t d;
  d.quot = numer / denom;
  d.rem = numer % denom;
  return d;
}
```

95.9 os32: «lib/libgen.h»

Si veda la sezione 91.3.

```
#ifndef _LIBGEN_H
#define _LIBGEN_H       1

//----------------------------------------------------------
char *basename (char *path);
char *dirname (char *path);
//----------------------------------------------------------

#endif
```

95.9.1 lib/libgen/basename.c

Si veda la sezione 88.10.

```
#include <libgen.h>
#include <limits.h>
#include <stddef.h>
#include <string.h>
//----------------------------------------------------------
char *
basename (char *path)
{
  static char *point = ".";    // When 'path' is
  // NULL.
  char *p;      // Pointer inside 'path'.
  int i;        // Scan index inside 'path'.
  //
  // Empty path.
  //
  if (path == NULL || strlen (path) == 0)
    {
      return (point);
    }
  //
  // Remove all final '/' if it exists, excluded the
  // first character:
  // 'i' is kept greater than zero.
  //
  for (i = (strlen (path) - 1);
       i > 0 && path[i] == '/'; i--)
    {
      path[i] = 0;
    }
  //
  // After removal of extra final '/', if there is
  // only one '/', this
  // is to be returned.
  //
  if (strncmp (path, "/", PATH_MAX) == 0)
    {
      return (path);
    }
  //
  // If there are no `/'.
  //
  if (strchr (path, '/') == NULL)
    {
      return (path);
    }
  //
  // Find the last `/' and calculate a pointer to the
  // base name.
  //
  p = strrchr (path, (unsigned int) '/');
  p++;
  //
  // Return the pointer to the base name.
  //
  return (p);
}
```

95.9.2 lib/libgen/dirname.c

Si veda la sezione 88.10.

```
#include <libgen.h>
#include <limits.h>
#include <stddef.h>
#include <string.h>
//----------------------------------------------------------
char *
dirname (char *path)
{
  static char *point = ".";     // When `path' is
  // NULL.
```

```
    char *p;      // Pointer inside ‘path’.
    int i;        // Scan index inside ‘path’.
    //
    // Empty path.
    //
    if (path == NULL || strlen (path) == 0)
      {
        return (point);
      }
    //
    // Simple cases.
    //
    if (strncmp (path, "/", PATH_MAX) == 0 ||
        strncmp (path, ".", PATH_MAX) == 0 ||
        strncmp (path, "..", PATH_MAX) == 0)
      {
        return (path);
      }
    //
    // Remove all final ‘/’ if it exists, excluded the
    // first character:
    // ‘i’ is kept greater than zero.
    //
    for (i = (strlen (path) - 1);
         i > 0 && path[i] == '/'; i--)
      {
        path[i] = 0;
      }
    //
    // After removal of extra final ‘/’, if there is
    // only one ‘/’, this
    // is to be returned.
    //
    if (strncmp (path, "/", PATH_MAX) == 0)
      {
        return (path);
      }
    //
    // If there are no ‘/’
    //
    if (strchr (path, '/') == NULL)
      {
        return (point);
      }
    //
    // If there is only a ‘/’ a the beginning.
    //
    if (path[0] == '/' &&
        strchr (&path[1], (unsigned int) '/') == NULL)
      {
        path[1] = 0;
        return (path);
      }
    //
    // Replace the last ‘/’ with zero.
    //
    p = strrchr (path, (unsigned int) '/');
    *p = 0;
    //
    // Now remove extra duplicated final ‘/’, except the
    // very first
    // character: ‘i’ is kept greater than zero.
    //
    for (i = (strlen (path) - 1);
         i > 0 && path[i] == '/'; i--)
      {
        path[i] = 0;
      }
    //
    // Now ‘path’ appears as a reduced string: the
    // original path string
    // is modified.
    //
    return (path);
```

```
}
```

95.10 os32: «lib/netinet/icmp.h»

Si veda la sezione 91.3.

```
#ifndef __NETINET_ICMP_H
#define __NETINET_ICMP_H    1
//------------------------------------------------------------
// GNU C compatible ICMPv4 header and definitions
//------------------------------------------------------------
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/ip.h>
//------------------------------------------------------------
struct icmphdr
{
  uint8_t type; // message type [1]
  uint8_t code; // type sub-code [2]
  uint16_t checksum;
  union
  {
    struct
    {
      uint16_t id;
      uint16_t sequence;
    } __attribute__ ((packed)) echo;    // echo
    // datagram
    uint32_t gateway;   // gateway address
    struct
    {
      uint16_t unused;
      uint16_t mtu;
    } __attribute__ ((packed)) frag;    // path mtu
    // discovery
  } un;
} __attribute__ ((packed));
```

```
//
// [1] message type:
//
#define ICMP_ECHOREPLY          0  // echo reply
#define ICMP_DEST_UNREACH       3  // destination
                                   // unreachable
#define ICMP_SOURCE_QUENCH      4  // source
                                   // quench
#define ICMP_REDIRECT           5  // redirect
                                   // (change
                                   // route)
#define ICMP_ECHO               8  // echo
                                   // request
#define ICMP_TIME_EXCEEDED      11 // time
                                   // exceeded
#define ICMP_PARAMETERPROB      12 // parameter
                                   // problem
#define ICMP_TIMESTAMP          13 // timestamp
                                   // request
#define ICMP_TIMESTAMPREPLY     14 // timestamp
                                   // reply
#define ICMP_INFO_REQUEST       15 // information
                                   // request
#define ICMP_INFO_REPLY         16 // information
                                   // reply
#define ICMP_ADDRESS            17 // address
                                   // mask
                                   // request
#define ICMP_ADDRESSREPLY       18 // address
                                   // mask reply
#define NR_ICMP_TYPES           18
//
// [2] type ICMP_DEST_UNREACH, code:
//
#define ICMP_NET_UNREACH        0  // network
                                   // unreachable
#define ICMP_HOST_UNREACH       1  // host
```

```
                                   // unreachable
#define ICMP_PROT_UNREACH       2  // protocol
                                   // unreachable
#define ICMP_PORT_UNREACH       3  // port
                                   // unreachable
#define ICMP_FRAG_NEEDED        4  // fragmentation
                                   // needed/DF
                                   // set
#define ICMP_SR_FAILED          5  // source
                                   // route
                                   // failed
#define ICMP_NET_UNKNOWN        6  // destination
                                   // network
                                   // unknown
#define ICMP_HOST_UNKNOWN       7  // destination
                                   // host
                                   // unknown
#define ICMP_HOST_ISOLATED      8  // source host
                                   // isolated
#define ICMP_NET_ANO            9  // destination
                                   // network
                                   // administratively
                                   // prohibited
#define ICMP_HOST_ANO          10  // destination
                                   // host
                                   // administratively
                                   // prohibited
#define ICMP_NET_UNR_TOS       11  // network
                                   // unreachable
                                   // for this
                                   // type of
                                   // service
#define ICMP_HOST_UNR_TOS      12  // host
                                   // unreachable
                                   // for this
                                   // type of
                                   // service
```

```
#define ICMP_PKT_FILTERED       13 // packet
                                   // filtered
#define ICMP_PREC_VIOLATION     14 // precedence
                                   // violation
#define ICMP_PREC_CUTOFF        15 // precedence
                                   // cut off
#define NR_ICMP_UNREACH         15 // instead of
                                   // hardcoding
                                   // immediate
                                   // value
//
// [2] type ICMP_REDIRECT, code:
//
#define ICMP_REDIR_NET          0  // redirect
                                   // net
#define ICMP_REDIR_HOST         1  // redirect
                                   // host
#define ICMP_REDIR_NETTOS       2  // redirect
                                   // net for TOS
#define ICMP_REDIR_HOSTTOS      3  // redirect
                                   // host for
                                   // TOS
//
// [2] type ICMP_TIME_EXCEEDED, code:
//
#define ICMP_EXC_TTL            0   // TTL count
                                    // exceeded
#define ICMP_EXC_FRAGTIME       1   // fragment
                                    // reass time
                                    // exceeded
//----------------------------------------------------------
#endif
```

95.11 os32: «lib/netinet/in.h»

Si veda la sezione 91.3.

```
#ifndef _NETINET_IN_H
#define _NETINET_IN_H    1
//----------------------------------------------------------
#include <stdint.h>
#include <sys/sa_family_t.h>
//----------------------------------------------------------
typedef uint16_t in_port_t;     // Port number. [1]
typedef uint32_t in_addr_t;     // IPv4 address.
//
// [1] Types 'in_port_t' and 'in_addr_t' are to be
//     intended for network byte order IPv4 integer
//     address, at least because this type is
//     used inside the type 'struct in_addr', that is
//     surely in network byte order. But attention must
//     be made to mistakes: for example,
//     inside the file <netinet/in.h> from GNU sources,
//     there are some macro defining default netmask
//     like this:
//
// #define IN_CLASSA(a)
//    ((((in_addr_t)(a)) & 0x80000000) == 0)
// #define IN_CLASSB(a)
//    ((((in_addr_t)(a)) & 0xc0000000) == 0x80000000)
// #define IN_CLASSC(a)
//    ((((in_addr_t)(a)) & 0xe0000000) == 0xc0000000)
//
//     Such macro can work only if the architecture is
//     big-endian.
//
//----------------------------------------------------------
//
// IPv4 address.
//
struct in_addr
```

```
{
  in_addr_t s_addr;
};
//
// struct sockaddr_in, members in *network*byte*order*.
//
struct sockaddr_in
{
  sa_family_t sin_family;       // AF_INET.
  in_port_t sin_port;   // Port number.
  struct in_addr sin_addr;      // IP address.
  uint8_t sin_zero[8];  // [2]
};
//
// [2] The type 'struct sockaddr_in' must be
//     replaceable with the type 'struct sockaddr',
//     with a cast. So it is necessary to fill the
//     unused space with a filler.
//
//----------------------------------------------------------
//
// IPv6 address, network byte order.
//
struct in6_addr
{
  uint8_t s6_addr[16];
};
//
// struct sockaddr_in6, members in network byte order.
//
struct sockaddr_in6
{
  sa_family_t sin6_family;      // AF_INET6.
  in_port_t sin6_port;  // Port number.
  uint32_t sin6_flowinfo;       // IPv6 traffic class
  // and flow info.
  struct in6_addr sin6_addr;    // IPv6 address.
  uint32_t sin6_scope_id;       // Set of interfaces
  // for a scope.
};
//----------------------------------------------------------
//external in6_addr in6addr_any;
//#define IN6ADDR_ANY_INIT  ...
//external struct in6_addr in6addr_loopback;
//#define IN6ADDR_LOOPBACK_INIT ...
//----------------------------------------------------------
//
//
//
struct ipv6_mreq
{
  struct in6_addr ipv6mr_multiaddr;     // IPv6
  // multicast
  // address.
  unsigned int ipv6mr_interface;        // Interface
  // index.
};
//----------------------------------------------------------
#define IPPROTO_IP      0       // Internet protocol.
#define IPPROTO_ICMP    1       // Contro message
                                // protocol.
#define IPPROTO_TCP     6       // Transmission
                                // control protocol.
#define IPPROTO_UDP    17       // User datagram
                                // protocol.
#define IPPROTO_IPV6   41       // Internet protocol
                                // version 6.
#define IPPROTO_RAW   255       // Raw IP packets
                                // protocol
//----------------------------------------------------------
//
// 0.0.0.0
//
#define INADDR_ANY         ((in_addr_t) 0x00000000)
```

```
//
// 255.255.255.255
//
#define INADDR_BROADCAST   ((in_addr_t) 0xffffffff)
//
// 127.0.0.1
//
#define INADDR_LOOPBACK    ((in_addr_t) 0x7f000001)
//
//
//
#define INET_ADDRSTRLEN  16     // IPv4 address string
                                // size.
#define INET6_ADDRSTRLEN 46     // IPv6 address string
                                // size.
//----------------------------------------------------------
#endif
```

95.12 os32: «lib/netinet/ip.h»

Si veda la sezione 91.3.

```
#ifndef _NETINET_IP_H
#define _NETINET_IP_H    1
//----------------------------------------------------------
// GNU C compatible IPv4 header.
//----------------------------------------------------------
#include <netinet/in.h>
//----------------------------------------------------------
struct iphdr
{
  uint16_t ihl:4,       // header length / 4
    version:4;  // IP version
  uint8_t tos;  // type of service
  uint16_t tot_len;     // total packet length
  uint16_t id;  // identification
  uint16_t frag_off;    // fragment offset field
  uint8_t ttl;  // time to live
  uint8_t protocol;     // contained protocol
  uint16_t check;       // header checksum
  in_addr_t saddr;      // source IP address
  in_addr_t daddr;      // destination IP address
  //
  // Options after this point.
  //
};
//----------------------------------------------------------
#define IPVERSION       4       // IP version number
#define IP_MAXPACKET    65535   // maximum packet size
//
#define MAXTTL          255     // maximum time to
                                // live (seconds)
#define IPDEFTTL        64      // default ttl, from
                                // RFC 1340
#define IPFRAGTTL       60      // time to live for
                                // fragments
#define IPTTLDEC        1       // subtracted when
                                // forwarding
//
#define IP_MSS          576     // default maximum
                                // segment size
//----------------------------------------------------------
#endif
```

95.13 os32: «lib/netinet/tcp.h»

Si veda la sezione 91.3.

```
#ifndef _NETINET_TCP_H
#define _NETINET_TCP_H  1
//----------------------------------------------------------
// GNU C compatible UDP header.
//----------------------------------------------------------
#include <sys/types.h>
//----------------------------------------------------------
struct tcphdr
{
  uint16_t source;
  uint16_t dest;
  uint32_t seq;
  uint32_t ack_seq;
  uint16_t res1:4,
    doff:4,
    fin:1, syn:1, rst:1, psh:1, ack:1, urg:1, res2:2;
  uint16_t window;
  uint16_t check;
  uint16_t urg_ptr;
};
//----------------------------------------------------------
// ATTENZIONE: per dare un significato allo stato di
// una connessione, occorre distinguere in che modo si
// trova inizialmente il socket:
// attivo o passivo (passivo quando rimane in ascolto
// per una connessione).
//
enum
{
  TCP_LISTEN = 1,       // waiting a connection
  // request
  TCP_SYN_SENT, // SYN was sent, waiting from the
  // response SYN
  TCP_SYN_RECV, // SYN received, waiting for ACK
  TCP_ESTABLISHED,      // SYN sent, SYN received and
  // ACK sent
  TCP_FIN_WAIT1,        // local close, FIN sent,
  // waiting ACK or FIN
  TCP_FIN_WAIT2,        // FIN sent, ACK received,
  // waiting FIN
  TCP_CLOSE_WAIT,       // FIN received, ACK sent,
  // waiting local close
  TCP_CLOSING,  // FIN sent, FIN received, ACK sent,
  // waiting ACK
  TCP_LAST_ACK, // FIN received, ACK and FIN sent,
  // waiting ACK
  TCP_TIME_WAIT,        // after TCP_LAST_ACK, wait a
  // little and remove
  TCP_CLOSE,    // connection removed
  TCP_RESET     // connection reset (not standard)
};

#define TCPOPT_EOL              0
#define TCPOPT_NOP              1
#define TCPOPT_MAXSEG           2
#define TCPOLEN_MAXSEG          4
#define TCPOPT_WINDOW           3
#define TCPOLEN_WINDOW          3
#define TCPOPT_SACK_PERMITTED   4
#define TCPOLEN_SACK_PERMITTED  2
#define TCPOPT_SACK             5
#define TCPOPT_TIMESTAMP        8
#define TCPOLEN_TIMESTAMP       10
//----------------------------------------------------------
//
// TCP max segment size: IP_MSS - IP header size.
// Suppose to have a max IP header of 56 bytes,
// TCP_MSS == 520.
//
#define TCP_MSS         520
//----------------------------------------------------------
```

```
// LA STRUTTURA SEGUENTE È DA VALUTARE, forse conviene
// fare una tabella a parte per le connessioni TCP.
//
struct tcp_info
{
  uint8_t tcpi_state;
  uint8_t tcpi_ca_state;
  uint8_t tcpi_retransmits;
  uint8_t tcpi_probes;
  uint8_t tcpi_backoff;
  uint8_t tcpi_options;
  uint8_t tcpi_snd_wscale:4, tcpi_rcv_wscale:4;

  uint32_t tcpi_rto;
  uint32_t tcpi_ato;
  uint32_t tcpi_snd_mss;
  uint32_t tcpi_rcv_mss;

  uint32_t tcpi_unacked;
  uint32_t tcpi_sacked;
  uint32_t tcpi_lost;
  uint32_t tcpi_retrans;
  uint32_t tcpi_fackets;

  /* Times. */
  uint32_t tcpi_last_data_sent;

  /* Not remembered, sorry.  */
  uint32_t tcpi_last_ack_sent;

  uint32_t tcpi_last_data_recv;
  uint32_t tcpi_last_ack_recv;

  /* Metrics. */
  uint32_t tcpi_pmtu;
  uint32_t tcpi_rcv_ssthresh;
  uint32_t tcpi_rtt;
  uint32_t tcpi_rttvar;
  uint32_t tcpi_snd_ssthresh;
  uint32_t tcpi_snd_cwnd;
  uint32_t tcpi_advmss;
  uint32_t tcpi_reordering;

  uint32_t tcpi_rcv_rtt;
  uint32_t tcpi_rcv_space;

  uint32_t tcpi_total_retrans;
};


//----------------------------------------------------------
#endif
```

95.14 os32: «lib/netinet/udp.h»

Si veda la sezione 91.3.

```
#ifndef __NETINET_UDP_H
#define __NETINET_UDP_H    1
//----------------------------------------------------------
// GNU C compatible UDP header.
//----------------------------------------------------------
#include <sys/types.h>
//----------------------------------------------------------
struct udphdr
{
  uint16_t source;      // source port
  uint16_t dest;        // destination port
  uint16_t len; // length
  uint16_t check;       // checksum
} __attribute__ ((packed));
//----------------------------------------------------------
```

```
#endif
```

95.15 os32: «lib/pwd.h»

Si veda la sezione 91.3.

```
#ifndef _PWD_H
#define _PWD_H        1
//-----------------------------------------------------------
#include <restrict.h>
#include <sys/types.h>  // gid_t, uid_t
//-----------------------------------------------------------
struct passwd
{
  char *pw_name;
  char *pw_passwd;
  uid_t pw_uid;
  gid_t pw_gid;
  char *pw_gecos;
  char *pw_dir;
  char *pw_shell;
};
//-----------------------------------------------------------
struct passwd *getpwent (void);
void setpwent (void);
void endpwent (void);
struct passwd *getpwnam (const char *name);
struct passwd *getpwuid (uid_t uid);
//-----------------------------------------------------------

#endif
```

95.15.1 lib/pwd/pwent.c

Si veda la sezione 88.57.

```
#include <pwd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//----------------------------------------------------------
static char buffer[BUFSIZ];
static struct passwd pw;
static FILE *fp = NULL;
//----------------------------------------------------------
struct passwd *
getpwent (void)
{
  void *pstatus;
  char *char_uid;
  char *char_gid;
  //
  if (fp == NULL)
    {
      fp = fopen ("/etc/passwd", "r");
      if (fp == NULL)
        {
          return NULL;
        }
    }
  //
  pstatus = fgets (buffer, BUFSIZ, fp);
  if (pstatus == NULL)
    {
      return (NULL);
    }
  //
  // The parse is made with 'strtok()'. Please notice
  // that
  // 'strtok()' will not parse a line like the
  // following:
  // user::1001:233:...
  // The password field *must* have something,
  // otherwise the
  // UID will take the password place.
  // `strtok()' will consider `::' the same as `:'!
  //
  pw.pw_name = strtok (buffer, ":");
  pw.pw_passwd = strtok (NULL, ":");
  char_uid = strtok (NULL, ":");
  char_gid = strtok (NULL, ":");
  pw.pw_gecos = strtok (NULL, ":");
  pw.pw_dir = strtok (NULL, ":");
  pw.pw_shell = strtok (NULL, "\n");
  pw.pw_uid = (uid_t) atoi (char_uid);
  pw.pw_gid = (gid_t) atoi (char_gid);
  //
  return (&pw);
}

//----------------------------------------------------------------------
void
endpwent (void)
{
  int status;
  //
  if (fp != NULL)
    {
      status = fclose (fp);
      if (status != 0)
        {
          perror (NULL);
          fp = NULL;
        }
      else
        {
          ;     // printf ("[%s] fclose (fp)\n",
          // __func__);
        }
    }
}

//----------------------------------------------------------
void
setpwent (void)
{
  if (fp != NULL)
    {
      rewind (fp);
    }
}

//----------------------------------------------------------
struct passwd *
getpwnam (const char *name)
{
  struct passwd *pw;
  //
  setpwent ();
  //
  for (;;)
    {
      pw = getpwent ();
      if (pw == NULL)
        {
          return (NULL);
        }
      if (strcmp (pw->pw_name, name) == 0)
        {
          return (pw);
        }
    }
}

```

```
//----------------------------------------------------------------------
struct passwd *
getpwuid (uid_t uid)
{
  struct passwd *pw;
  //
  setpwent ();
  //
  for (;;)
    {
      pw = getpwent ();
      if (pw == NULL)
        {
          return (NULL);
        }
      if (pw->pw_uid == uid)
        {
          return (pw);
        }
    }
}
```

95.16 os32: «lib/setjmp.h»

Si veda la sezione 87.49.

```
#ifndef _SETJMP_H
#define _SETJMP_H       1
//----------------------------------------------------------------------
#include <sys/os32.h>
#include <NULL.h>
//----------------------------------------------------------------------
typedef struct
{
  uint32_t eax0;
  uint32_t ecx0;
  uint32_t edx0;
  uint32_t ebx0;
  uint32_t ebp0;
  uint32_t esi0;
  uint32_t edi0;
  uint32_t ds0;
  uint32_t es0;
  uint32_t fs0;
  uint32_t gs0;
  uint32_t eip0;
  uint32_t cs0;
  uint32_t eflags0;
  //
  uint32_t eip1;
  uint32_t syscallnr;
  uint32_t msg_pointer;
  uint32_t msg_size;
  //
  uint32_t env;
  uint32_t ret;
  uint32_t ebp1;
  uint32_t eip2;
  //
} jmp_stack_t;

typedef struct
{
  uint32_t esp0;
  uint32_t eax0;
  uint32_t ecx0;
  uint32_t edx0;
  uint32_t ebx0;
  uint32_t ebp0;
  uint32_t esi0;
  uint32_t edi0;
  uint32_t ds0;
  uint32_t es0;
  uint32_t fs0;
  uint32_t gs0;
  uint32_t eip0;
  uint32_t cs0;
  uint32_t eflags0;
  //
  uint32_t eip1;
  uint32_t syscallnr;
  uint32_t msg_pointer;
  uint32_t msg_size;
  //
  uint32_t env;
  uint32_t ret;
  uint32_t ebp1;
  uint32_t eip2;
  //
} jmp_env_t;
//
typedef char jmp_buf[sizeof (jmp_env_t)];
//----------------------------------------------------------
int setjmp (jmp_buf env);
void longjmp (jmp_buf env, int val);
//----------------------------------------------------------
#endif
```

95.16.1 lib/setjmp/longjmp.c

Si veda la sezione 87.49.

```
#include <sys/os32.h>
#include <setjmp.h>
//----------------------------------------------------------
void
longjmp (jmp_buf env, int val)
```

```
{
  sysmsg_jmp_t msg;
  msg.env = env;
  msg.ret = val;
  sys (SYS_LONGJMP, &msg, sizeof msg);
}
```

95.16.2 lib/setjmp/setjmp.s

Si veda la sezione 87.49.

```
.global setjmp
.extern sys
#-------------------------------------------------------
.text
#-------------------------------------------------------
.align 4
setjmp:
    #
    # Previous pushes:
    #
    #   push &env
    #   push back_address   # made by a call to
    #                       # setjmp() function
    #
    enter $8, $0
    #
    # sysmsg_jmp_t msg;
    #
    movl  $0,  -4(%ebp)        # msg.ret = 0;
    #
    movl  8(%ebp), %eax        # msg.env = env;
    movl  %eax,-8(%ebp)
    #
    # sys (SYS_SETJMP, &msg, sizeof msg);
    #
    lea   -8(%ebp), %eax
    pushl $8                    # sizeof msg
    pushl %eax                  # &msg
    pushl $47                   # SYS_SETJMP
    call  sys
    add   $4, %esp
    add   $4, %esp
    add   $4, %esp
    #
    # return (msg.ret);
    #
    movl  -4(%ebp), %eax
    leave
    ret
```

95.17 os32: «lib/signal.h»

«

Si veda la sezione 91.3.

```
#ifndef _SIGNAL_H
#define _SIGNAL_H       1
//----------------------------------------------------------
#include <sys/types.h>
//----------------------------------------------------------
#define SIGHUP           1
#define SIGINT           2
#define SIGQUIT          3
#define SIGILL           4
#define SIGABRT          6
#define SIGFPE           8
#define SIGKILL          9
#define SIGSEGV         11
#define SIGPIPE         13
#define SIGALRM         14
#define SIGTERM         15
#define SIGSTOP         17
#define SIGTSTP         18
#define SIGCONT         19
```

```
#define SIGCHLD         20
#define SIGTTIN         21
#define SIGTTOU         22
#define SIGUSR1         30
#define SIGUSR2         31
//----------------------------------------------------------
typedef int sig_atomic_t;
typedef void (*sighandler_t) (int);     // [1]
//
// [1] The type ‘sighandler_t’ is a pointer to a
// function for the signal handling, with a parameter
// of type ‘int’, returning ‘void’.
//
//----------------------------------------------------------
// Special function used to call the real signal
// handler. This function will return to the ‘back’
// address, instead where it was called.
//
void _sighandler_wrapper (uint32_t handler,
                          uint32_t signal, uint32_t back);
//----------------------------------------------------------
// Special undeclarable functions.
//
#define SIG_ERR ((sighandler_t) -1)     // [2]
#define SIG_DFL ((sighandler_t) 0)      // [2]
#define SIG_IGN ((sighandler_t) 1)      // [2]
//
// [2] It transforms an integer number into a
//      ‘sighnandler_t’ type, that is, a pointer
//      to a function that does not exists really.
//
//----------------------------------------------------------
sighandler_t signal (int sig, sighandler_t handler);
int kill (pid_t pid, int sig);
int raise (int sig);
//----------------------------------------------------------
```

```
#endif
```

95.17.1 lib/signal/_sighandler_wrapper.s

Si veda la sezione 87.52.

```
.global _sighandler_wrapper
#------------------------------------------------------
.section .text
#------------------------------------------------------
# Port input byte.
#------------------------------------------------------
_sighandler_wrapper:
    #
    # Current stack is:
    #
    # push %eip              # Back from interrupted code.
    # push <sig_num>         # Signal number.
    # push <sig_handler>     # Signal handler address
    #
    # Please note that THERE IS NO RETURN ADDRESS!
    # Instead you find the signal handler address
    # there.
    #
    # This routine should have to call the signal
    # handler function, and then return back to the
    # interrupted code.
    #
    enter $0, $0                # No local variables.
    pushf
    pusha
    .equ SIG_HAND,  4           # First argument.  [1]
    .equ SIG_NUM,   8           # Second argument. [1]
    #
    # [1] This function is called without the return
    #     address inside the stack. So the arguments
    #     are 4 bytes nearer than the usual.
    #
    mov  SIG_NUM(%ebp), %edx    # Copy the signal
                                # number into EDX.
    mov  SIG_HAND(%ebp), %eax   # Copy the signal
                                # handler function
                                # address into EAX.
    push %edx                   # Prepare argument for
                                # the signal
                                # handler function.
    call *%eax                  # Call the signal
                                # handler function.
    add  $4, %esp               # Pop the signal
                                # number argument.
    popa
    popf
    leave
    #
    # Now we are back to the same stack as the
    # beginning:
    #
    # push %eip         # back from interrupted code.
    # push <sig_num>
    # push <sig_handler>
    # push %eip         # back from
    #                   # _sighandler_wrapper()
    #
    # The stack pointer must be modified before
    # returning, so that the address to the original
    # interrupted instruction is used for return.
    # Without such modification, the RET
    # instruction would find the signal handler address
    # instead!
    #
    add $4, %esp
    add $4, %esp
    #
    # Now we are ready to return to the original
    # interrupted address!
    #
    ret

```

95.17.2 lib/signal/kill.c

Si veda la sezione 87.29.

```
#include <sys/os32.h>
#include <sys/types.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
//----------------------------------------------------------
int
kill (pid_t pid, int sig)
{
  sysmsg_kill_t msg;
  if (pid < -1) // Currently unsupported.
    {
      errset (ESRCH);
      return (-1);
    }
  msg.pid = pid;
  msg.signal = sig;
  msg.ret = 0;
  msg.errno = 0;
  sys (SYS_KILL, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.17.3 lib/signal/signal.c

«

Si veda la sezione 87.52.

```
#include <sys/os32.h>
#include <sys/types.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
//----------------------------------------------------------------------
sighandler_t
signal (int sig, sighandler_t handler)
{
  sysmsg_signal_t msg;

  msg.signal = sig;
  msg.handler = handler;
  msg.wrapper = (uintptr_t) _sighandler_wrapper;
  msg.ret = SIG_DFL;
  msg.errno = 0;
  sys (SYS_SIGNAL, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.18 os32: «lib/stdio.h»

Si veda la sezione 88.112.

```
#ifndef _STDIO_H
#define _STDIO_H        1
//----------------------------------------------------------
#include <restrict.h>
#include <stdarg.h>
#include <stdint.h>
#include <limits.h>
#include <NULL.h>
#include <size_t.h>
#include <sys/types.h>
#include <SEEK.h>       // SEEK_CUR, SEEK_SET,
                        // SEEK_END
//----------------------------------------------------------
#define BUFSIZ              8192 // At least the
                                 // file
                                 // system max zone
                                 // size.
#define _IOFBF                 0 // Input-output
                                 // fully
                                 // buffered.
#define _IOLBF                 1 // Input-output
                                 // line
                                 // buffered.
#define _IONBF                 2 // Input-output
                                 // with
                                 // no buffering.

#define L_tmpnam    FILENAME_MAX // <limits.h>

#define FOPEN_MAX       OPEN_MAX // <limits.h>
#define FILENAME_MAX    NAME_MAX // <limits.h>
#define TMP_MAX           0x7FFF

#define EOF                 (-1) // Must be a
```

```
                                 // negative
                                 // value.
//----------------------------------------------------------
typedef off_t fpos_t;   // ‘off_t’ defined in
                        // <sys/types.h>.

typedef struct
{
  int fdn;      // File descriptor number.
  char error;   // Error indicator.
  char eof;     // End of file indicator.
} FILE;

extern FILE _stream[];  // Defined inside
                        // ‘lib/stdio/FILE.c’.

#define stdin   (&_stream[0])
#define stdout  (&_stream[1])
#define stderr  (&_stream[2])
//----------------------------------------------------------
void clearerr (FILE * fp);
int fclose (FILE * fp);
int feof (FILE * fp);
int ferror (FILE * fp);
int fflush (FILE * fp);
int fgetc (FILE * fp);
int fgetpos (FILE * restrict fp, fpos_t * restrict pos);
char *fgets (char *restrict string, int n,
             FILE * restrict fp);
int fileno (FILE * fp);
FILE *fopen (const char *path, const char *mode);
int fprintf (FILE * fp, char *restrict format, ...);
int fputc (int c, FILE * fp);
int fputs (const char *restrict string, FILE * restrict fp);
size_t fread (void *restrict buffer, size_t size,
              size_t nmemb, FILE * restrict fp);
FILE *freopen (const char *restrict path,
               const char *restrict mode,
               FILE * restrict fp);
int fscanf (FILE * restrict fp,
            const char *restrict format, ...);
int fseek (FILE * fp, long int offset, int whence);
int fsetpos (FILE * fp, fpos_t * pos);
long int ftell (FILE * fp);
off_t ftello (FILE * fp);
size_t fwrite (const void *restrict buffer,
               size_t size, size_t nmemb,
               FILE * restrict fp);
#define  getc(p)    (fgetc (p))
int getchar (void);
char *gets (char *string);
void perror (const char *string);
int printf (const char *restrict format, ...);
#define  putc(c, p) (fputc ((c), (p)))
int putchar (int c);
int puts (const char *string);
void rewind (FILE * fp);
int scanf (const char *restrict format, ...);
void setbuf (FILE * restrict fp, char *restrict buffer);
int setvbuf (FILE * restrict fp, char *restrict buffer,
             int buf_mode, size_t size);
int snprintf (char *restrict string, size_t size,
              const char *restrict format, ...);
int sprintf (char *restrict string,
             const char *restrict format, ...);
int sscanf (char *restrict string,
            const char *restrict format, ...);
int vfprintf (FILE * fp, char *restrict format,
              va_list arg);
int vfscanf (FILE * restrict fp,
             const char *restrict format, va_list arg);
int vprintf (const char *restrict format, va_list arg);
int vscanf (const char *restrict format, va_list ap);
int vsnprintf (char *restrict string, size_t size,
               const char *restrict format, va_list arg);
int vsprintf (char *restrict string,
              const char *restrict format, va_list arg);
int vsscanf (const char *string, const char *format,
             va_list ap);
//----------------------------------------------------------
#endif
```

95.18.1 lib/stdio/FILE.c

«

Si veda la sezione 91.3.

```
#include <stdio.h>
//
// There must be room for at least 'FOPEN_MAX'
// elements.
//
FILE _stream[FOPEN_MAX];
//----------------------------------------------------------
void
_stdio_stream_setup (void)
{
  _stream[0].fdn = 0;
  _stream[0].error = 0;
  _stream[0].eof = 0;

  _stream[1].fdn = 1;
  _stream[1].error = 0;
  _stream[1].eof = 0;

  _stream[2].fdn = 2;
  _stream[2].error = 0;
  _stream[2].eof = 0;
}
```

95.18.2 lib/stdio/clearerr.c

Si veda la sezione 88.12.

```
#include <stdio.h>
//----------------------------------------------------------
void
clearerr (FILE * fp)
{
  if (fp != NULL)
    {
      fp->error = 0;
      fp->eof = 0;
    }
}
```

95.18.3 lib/stdio/fclose.c

Si veda la sezione 88.28.

```
#include <stdio.h>
#include <unistd.h>
//----------------------------------------------------------
int
fclose (FILE * fp)
{
  return (close (fp->fdn));
}
```

95.18.4 lib/stdio/feof.c

Si veda la sezione 88.29.

```
#include <stdio.h>
//----------------------------------------------------------
int
feof (FILE * fp)
{
  if (fp != NULL)
    {
      return (fp->eof);
    }
  return (0);
}
```

95.18.5 lib/stdio/ferror.c

Si veda la sezione 88.30.

```
#include <stdio.h>
//----------------------------------------------------------
int
ferror (FILE * fp)
{
  if (fp != NULL)
    {
      return (fp->error);
    }
  return (0);
}
```

95.18.6 lib/stdio/fflush.c

Si veda la sezione 88.31.

```
#include <stdio.h>
//----------------------------------------------------------
int
fflush (FILE * fp)
{
  //
  // The os32 library does not have any buffered data.
  //
  return (0);
}
```

95.18.7 lib/stdio/fgetc.c

Si veda la sezione 88.32.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
//----------------------------------------------------------
int
fgetc (FILE * fp)
{
  ssize_t size_read;
  int c;        // Character read.
  //
  for (c = 0;;)
    {
      size_read = read (fp->fdn, &c, (size_t) 1);
      //
      if (size_read <= 0)
        {
          //
          // It is the end of file (zero) otherwise
          // there is a
          // problem (a negative value): return `EOF'.
          //
          return (EOF);
        }
      //
      // Valid read: end of scan.
      //
      return (c);
    }
}
```

95.18.8 lib/stdio/fgetpos.c

«

Si veda la sezione 88.33.

```
#include <stdio.h>
//----------------------------------------------------------
int
fgetpos (FILE * restrict fp, fpos_t * restrict pos)
{
  long int position;
  //
  if (fp != NULL)
    {
      position = ftell (fp);
      if (position >= 0)
        {
          *pos = position;
          return (0);
        }
    }
  return (-1);
}
```

95.18.9 lib/stdio/fgets.c

«

Si veda la sezione 88.34.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stddef.h>
//----------------------------------------------------------
char *
fgets (char *restrict string, int n, FILE * restrict fp)
{
  ssize_t size_read;
  int b;          // Index inside the string buffer.
  //
  for (b = 0; b < (n - 1); b++, string[b] = 0)
    {
      size_read = read (fp->fdn, &string[b], (size_t) 1);
      //
      if (size_read <= 0)
        {
          //
          // It is the end of file (zero) otherwise
          // there is a
          // problem (a negative value).
          //
          string[b] = 0;
          break;
        }
      //
      if (string[b] == ’\n’)
        {
          b++;
          string[b] = 0;
          break;
        }
    }
  //
  // If ‘b‘ is zero, nothing was read and ‘NULL’ is
  // returned.
  //
  if (b == 0)
    {
      return (NULL);
    }
  else
    {
      return (string);
    }
}
```

95.18.10 lib/stdio/fileno.c

Si veda la sezione 88.35.

```
#include <stdio.h>
#include <errno.h>
//----------------------------------------------------------
int
fileno (FILE * fp)
{
  if (fp != NULL)
    {
      return (fp->fdn);
    }
  errset (EBADF);       // Bad file descriptor.
  return (-1);
}
```

95.18.11 lib/stdio/fopen.c

Si veda la sezione 88.36.

```
#include <fcntl.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <sys/os32.h>
#include <limits.h>
#include <stdio.h>
//----------------------------------------------------------
FILE *
fopen (const char *path, const char *mode)
{
  int fdn;
  //
  if (strcmp (mode, "r") || strcmp (mode, "rb"))
    {
        fdn = open (path, O_RDONLY);
      }
    else if (strcmp (mode, "r+") ||
             strcmp (mode, "r+b") || strcmp (mode, "rb+"))
      {
        fdn = open (path, O_RDWR);
      }
    else if (strcmp (mode, "w") || strcmp (mode, "wb"))
      {
        fdn = open (path, O_WRONLY | O_CREAT | O_TRUNC, 0666);
      }
    else if (strcmp (mode, "w+") ||
             strcmp (mode, "w+b") || strcmp (mode, "wb+"))
      {
        fdn = open (path, O_RDWR | O_CREAT | O_TRUNC, 0666);
      }
    else if (strcmp (mode, "a") || strcmp (mode, "ab"))
      {
        fdn =
          open (path,
                O_WRONLY | O_APPEND | O_CREAT | O_TRUNC,
                0666);
      }
    else if (strcmp (mode, "a+") ||
             strcmp (mode, "a+b") || strcmp (mode, "ab+"))
      {
        fdn =
          open (path,
                O_RDWR | O_APPEND | O_CREAT | O_TRUNC, 0666);
      }
    else
      {
        errset (EINVAL);  // Invalid argument.
        return (NULL);
      }
    //
    // Check the file descriptor returned.
```

```
  //
  if (fdn < 0)
    {
      //
      // The variable 'errno' is already set.
      //
      errset (errno);
      return (NULL);
    }
  //
  // A valid file descriptor is available: convert it
  // into a file
  // stream. Please note that the file descriptor
  // number must be
  // saved inside the corresponding '_stream[]' array,
  // because the
  // file pointer do not have knowledge of the
  // relative position
  // inside the array.
  //
  _stream[fdn].fdn = fdn;       // Saved the file
  // descriptor number.
  //
  return (&_stream[fdn]);       // Returned the file
  // stream pointer.
}
```

95.18.12 lib/stdio/fprintf.c

Si veda la sezione 88.91.

```
#include <stdio.h>
//-----------------------------------------------------------
int
fprintf (FILE * fp, char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return (vfprintf (fp, format, ap));
}
```

95.18.13 lib/stdio/fputc.c

Si veda la sezione 88.38.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <string.h>
#include <unistd.h>
//----------------------------------------------------------------------
int
fputc (int c, FILE * fp)
{
  ssize_t size_written;
  char character = (char) c;
  size_written = write (fp->fdn, &character, (size_t) 1);
  if (size_written < 0)
    {
      fp->eof = 1;
      return (EOF);
    }
  return (c);
}
```

95.18.14 lib/stdio/fputs.c

Si veda la sezione 88.39.

```
#include <stdio.h>
#include <string.h>
//----------------------------------------------------------------------
int
fputs (const char *restrict string, FILE * restrict fp)
```

```
{
  int i;        // Index inside the string to be
  // printed.
  int status;

  for (i = 0; i < strlen (string); i++)
    {
      status = fputc (string[i], fp);
      if (status == EOF)
        {
          fp->eof = 1;
          return (EOF);
        }
    }
  return (0);
}
```

95.18.15 lib/stdio/fread.c

Si veda la sezione 88.40.

```
#include <unistd.h>
#include <stdio.h>
//----------------------------------------------------------
size_t
fread (void *restrict buffer, size_t size,
       size_t nmemb, FILE * restrict fp)
{
  ssize_t size_read;
  size_read =
    read (fp->fdn, buffer, (size_t) (size * nmemb));
  if (size_read == 0)
    {
      fp->eof = 1;
      return ((size_t) 0);
    }
  else if (size_read < 0)
```

```
    {
      fp->error = 1;
      return ((size_t) 0);
    }
  else
    {
      return ((size_t) (size_read / size));
    }
}
```

95.18.16 lib/stdio/freopen.c

Si veda la sezione 88.36.

```
#include <fcntl.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <sys/os32.h>
#include <limits.h>
#include <stdio.h>
//----------------------------------------------------------
FILE *
freopen (const char *restrict path,
         const char *restrict mode, FILE * restrict fp)
{
  int status;
  FILE *fp_new;
  //
  if (fp == NULL)
    {
      return (NULL);
    }
  //
  status = fclose (fp);
  if (status != 0)
```

```
    {
      fp->error = 1;
      return (NULL);
    }
  //
  fp_new = fopen (path, mode);
  //
  if (fp_new == NULL)
    {
      return (NULL);
    }
  //
  if (fp_new != fp)
    {
      fclose (fp_new);
      return (NULL);
    }
  //
  return (fp_new);
}
```

95.18.17 lib/stdio/fscanf.c

Si veda la sezione 88.102.

```
#include <stdio.h>
//----------------------------------------------------------
int
fscanf (FILE * restrict fp,
        const char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return vfscanf (fp, format, ap);
}
```

95.18.18 lib/stdio/fseek.c

Si veda la sezione 88.44.

```
#include <stdio.h>
#include <unistd.h>
//----------------------------------------------------------------------
int
fseek (FILE * fp, long int offset, int whence)
{
  off_t off_new;
  off_new = lseek (fp->fdn, (off_t) offset, whence);
  if (off_new < 0)
    {
      fp->error = 1;
      return (-1);
    }
  else
    {
      fp->eof = 0;
      return (0);
    }
}
```

95.18.19 lib/stdio/fseeko.c

Si veda la sezione 88.44.

```
#include <stdio.h>
#include <unistd.h>
//----------------------------------------------------------------------
int
fseeko (FILE * fp, off_t offset, int whence)
{
  off_t off_new;
  off_new = lseek (fp->fdn, offset, whence);
  if (off_new < 0)
    {
      fp->error = 1;
      return (-1);
    }
  else
    {
      return (0);
    }
}
```

95.18.20 lib/stdio/fsetpos.c

Si veda la sezione 88.33.

```
#include <stdio.h>
//----------------------------------------------------------------------
int
fsetpos (FILE * fp, fpos_t * pos)
{
  long int position;
  //
  if (fp != NULL)
    {
      position = fseek (fp, (long int) *pos, SEEK_SET);
      if (position >= 0)
        {
          *pos = position;
          return (0);
        }
    }
  return (-1);
}
```

95.18.21 lib/stdio/ftell.c

Si veda la sezione 88.47.

```
#include <stdio.h>
#include <unistd.h>
//----------------------------------------------------------
long int
ftell (FILE * fp)
{
  return ((long int) lseek (fp->fdn, (off_t) 0, SEEK_CUR));
}
```

95.18.22 lib/stdio/ftello.c

Si veda la sezione 88.47.

```
#include <stdio.h>
#include <unistd.h>
//----------------------------------------------------------
off_t
ftello (FILE * fp)
{
  return (lseek (fp->fdn, (off_t) 0, SEEK_CUR));
}
```

95.18.23 lib/stdio/fwrite.c

Si veda la sezione 88.49.

```
#include <unistd.h>
#include <stdio.h>
//----------------------------------------------------------
size_t
fwrite (const void *restrict buffer, size_t size,
        size_t nmemb, FILE * restrict fp)
{
  ssize_t size_written;
  size_written =
    write (fp->fdn, buffer, (size_t) (size * nmemb));
  if (size_written < 0)
    {
      fp->error = 1;
      return ((size_t) 0);
    }
  else
    {
      return ((size_t) (size_written / size));
    }
}
```

95.18.24 lib/stdio/getchar.c

Si veda la sezione 88.32.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
//------------------------------------------------------------
int
getchar (void)
{
  ssize_t size_read;
  int c;        // Character read.
  //
  for (c = 0;;)
    {
      size_read = read (STDIN_FILENO, &c, (size_t) 1);
      //
      if (size_read <= 0)
        {
          //
          // It is the end of file (zero) otherwise
          // there is a
          // problem (a negative value): return 'EOF'.
          //
          _stream[STDIN_FILENO].eof = 1;
          return (EOF);
        }
      //
      // Valid read.
      //
      if (size_read == 0)
        {
          //
          // If no character is ready inside the
          // keyboard buffer, just
          // retry.
          //
          continue;
        }
      //
      // End of scan.
      //
      return (c);
    }
}
```

95.18.25 lib/stdio/gets.c

Si veda la sezione 88.34.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stddef.h>
//----------------------------------------------------------------------
char *
gets (char *string)
{
  ssize_t size_read;
  int b;              // Index inside the string buffer.
```

```
    //
    for (b = 0;; b++, string[b] = 0)
      {
        size_read =
          read (STDIN_FILENO, &string[b], (size_t) 1);
        //
        if (size_read <= 0)
          {
            //
            // It is the end of file (zero) otherwise
            // there is a
            // problem (a negative value).
            //
            _stream[STDIN_FILENO].eof = 1;
            string[b] = 0;
            break;
          }
        //
        if (string[b] == ’\n’)
          {
            b++;
            string[b] = 0;
            break;
          }
      }
    //
    // If ‘b‘ is zero, nothing was read and ‘NULL’ is
    // returned.
    //
    if (b == 0)
      {
        return (NULL);
      }
    else
      {
        return (string);
      }
```

```
}
```

95.18.26 lib/stdio/perror.c

Si veda la sezione 88.90.

```
#include <stdio.h>
#include <errno.h>
#include <stddef.h>
#include <string.h>
//----------------------------------------------------------------------
void
perror (const char *string)
{
  //
  // If errno is zero, there is nothing to show.
  //
  if (errno == 0)
    {
      return;
    }
  //
  // Show the string if there is one.
  //
  if (string != NULL && strlen (string) > 0)
    {
      printf ("%s: ", string);
    }
  //
  // Show the translated error.
  //
  if (errfn[0] != 0 && errln != 0)
    {
      printf ("[%s:%u:%i] %s\n",
              errfn, errln, errno, strerror (errno));
    }
  else
```

```
    {
      printf ("[%i] %s\n", errno, strerror (errno));
    }
}
```

95.18.27 lib/stdio/printf.c

Si veda la sezione 88.91.

```
#include <stdio.h>
//----------------------------------------------------------
int
printf (const char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return (vprintf (format, ap));
}
```

95.18.28 lib/stdio/putchar.c

Si veda la sezione 88.38.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <string.h>
#include <unistd.h>
//----------------------------------------------------------
int
putchar (int c)
{
  return (fputc (c, stdout));
}
```

95.18.29 lib/stdio/puts.c

Si veda la sezione 88.39.

```
#include <stdio.h>
//----------------------------------------------------------
int
puts (const char *string)
{
  int status;
  status = printf ("%s\n", string);
  if (status < 0)
    {
      return (EOF);
    }
  else
    {
      return (status);
    }
}
```

95.18.30 lib/stdio/rewind.c

Si veda la sezione 88.100.

```
#include <stdio.h>
//----------------------------------------------------------
void
rewind (FILE * fp)
{
  (void) fseek (fp, 0L, SEEK_SET);
  fp->error = 0;
}
```

95.18.31 lib/stdio/scanf.c

Si veda la sezione 88.102.

```
#include <stdio.h>
//----------------------------------------------------------
int
scanf (const char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return vfscanf (stdin, format, ap);
}
```

95.18.32 lib/stdio/setbuf.c

Si veda la sezione 88.103.

```
#include <stdio.h>
//----------------------------------------------------------
void
setbuf (FILE * restrict fp, char *restrict buffer)
{
  //
  // The os32 library does not have any buffered data.
  //
  return;
}
```

95.18.33 lib/stdio/setvbuf.c

Si veda la sezione 88.103.

```
#include <stdio.h>
//----------------------------------------------------------
int
setvbuf (FILE * restrict fp, char *restrict buffer,
         int buf_mode, size_t size)
```

```
{
  //
  // The os32 library does not have any buffered data.
  //
  return (0);
}
```

95.18.34 lib/stdio/snprintf.c

Si veda la sezione 88.91.

```
#include <stdio.h>
#include <stdarg.h>
//----------------------------------------------------------
int
snprintf (char *restrict string, size_t size,
          const char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return vsnprintf (string, size, format, ap);
}
```

95.18.35 lib/stdio/sprintf.c

Si veda la sezione 88.91.

```
#include <stdio.h>
#include <stdarg.h>
//----------------------------------------------------------
int
sprintf (char *restrict string,
         const char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return vsnprintf (string, (size_t) BUFSIZ, format, ap);
}
```

95.18.36 lib/stdio/sscanf.c

Si veda la sezione 88.102.

```
#include <stdio.h>
//----------------------------------------------------------------------
int
sscanf (char *restrict string,
        const char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return vsscanf (string, format, ap);
}
```

95.18.37 lib/stdio/vfprintf.c

Si veda la sezione 88.137.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <string.h>
#include <unistd.h>
//----------------------------------------------------------------------
int
vfprintf (FILE * fp, char *restrict format, va_list arg)
{
  ssize_t size_written;
  size_t size;
  size_t size_total;
  int status;
  char string[BUFSIZ];
  char *buffer = string;
  //
  buffer[0] = 0;
  status = vsprintf (buffer, format, arg);
  //
  size = strlen (buffer);
  if (size >= BUFSIZ)
    {
      size = BUFSIZ;
    }
  //
  for (size_total = 0, size_written = 0;
       size_total < size;
       size_total += size_written, buffer += size_written)
    {
      size_written =
        write (fp->fdn, buffer, size - size_total);
      if (size_written < 0)
        {
          return (size_total);
        }
    }
  return (size);
}
```

95.18.38 lib/stdio/vfscanf.c

Si veda la sezione 88.138.

```
#include <stdio.h>

//----------------------------------------------------------
int vfsscanf (FILE * restrict fp, const char *string,
              const char *restrict format, va_list ap);
//----------------------------------------------------------
int
vfscanf (FILE * restrict fp,
         const char *restrict format, va_list ap)
{
  return (vfsscanf (fp, NULL, format, ap));
}

//----------------------------------------------------------
```

95.18.39 lib/stdio/vfsscanf.c

Si veda la sezione 88.138.

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <stdarg.h>
#include <ctype.h>
#include <errno.h>
#include <stddef.h>
//----------------------------------------------------------
//
// This function is not standard and is able to do the
// work of both 'vfscanf()' and 'vsscanf()'.
//
//----------------------------------------------------------
#define WIDTH_MAX       64
//----------------------------------------------------------
static intmax_t strtointmax (const char *restrict
                             string,
                             const char **restrict
                             endptr, int base,
                             size_t max_width);
static int ass_or_eof (int consumed, int assigned);
//----------------------------------------------------------
int
vfsscanf (FILE * restrict fp, const char *string,
          const char *restrict format, va_list ap)
{
    int f = 0;    // Format index.
    char buffer[BUFSIZ];
    const char *input = string;   // Default.
    const char *start = input;    // Default.
    const char *restrict next = NULL;
    int scanned = 0;
    //
    bool stream = 0;
    bool flag_star = 0;
    bool specifier = 0;
    bool specifier_flags = 0;
    bool specifier_width = 0;
    bool specifier_type = 0;
    bool inverted = 0;
    //
    char *ptr_char;
    signed char *ptr_schar;
    unsigned char *ptr_uchar;
    short int *ptr_sshort;
    unsigned short int *ptr_ushort;
    int *ptr_sint;
    unsigned int *ptr_uint;
    long int *ptr_slong;
    unsigned long int *ptr_ulong;
    intmax_t *ptr_simax;
    uintmax_t *ptr_uimax;
    size_t *ptr_size;
    ptrdiff_t *ptr_ptrdiff;
    void **ptr_void;
    //
    size_t width;
    char width_string[WIDTH_MAX + 1];
    int w;        // Index inside width string.
    int assigned = 0;     // Assignment counter.
    int consumed = 0;     // Consumed counter.
    //
    intmax_t value_i;
    uintmax_t value_u;
    //
    const char *end_format;
    const char *end_input;
    int count;    // Generic counter.
    int index;    // Generic index.
    bool ascii[128];
    //
    void *pstatus;
    //
    // Initialize some data.
    //
    width_string[0] = '\0';
    end_format = format + (strlen (format));
    //
    // Check arguments and find where input comes.
    //
    if (fp == NULL && (string == NULL || string[0] == 0))
      {
        errset (EINVAL);  // Invalid argument.
        return (EOF);
      }
    //
    if (fp != NULL && string != NULL && string[0] != 0)
      {
        errset (EINVAL);  // Invalid argument.
        return (EOF);
      }
    //
    if (fp != NULL)
      {
        stream = 1;
      }
    //
    //
    //
    for (;;)
```

```
      {
        if (stream)
          {
            pstatus = fgets (buffer, BUFSIZ, fp);
            //
            if (pstatus == NULL)
              {
                return (ass_or_eof (consumed, assigned));
              }
            //
            input = buffer;
            start = input;
            next = NULL;
          }
        //
        // Calculate end input.
        //
        end_input = input + (strlen (input));
        //
        // Scan format and input strings. Index `f' is
        // not reset.
        //
        while (&format[f] < end_format && input < end_input)
          {
            if (!specifier)
              {
                // --------------------------------------
                // The context is not
                // inside a specifier.
                // --------------------------------------
                if (isspace (format[f]))
                  {
                    // --------------------------- Space.
                    while (isspace (*input))
                      {
                        input++;
                      }
```

```
                   //
                   // Verify that the input string is
                   // not finished.
                   //
                   if (input[0] == 0)
                     {
                       //
                       // As the input string is
                       // finished, the format
                       // string index is not advanced,
                       // because there
                       // might be more spaces on the
                       // next line (if
                       // there is a next line, of
                       // course).
                       //
                       continue;
                     }
                   else
                     {
                       f++;
                       continue;
                     }
                 }
               if (format[f] != '%')
                 {
                   // -------------- Ordinary character.
                   if (format[f] == *input)
                     {
                       input++;
                       f++;
                       continue;
                     }
                   else
                     {
                       return (ass_or_eof
                               (consumed, assigned));
```

```
          }
      }
    if (format[f] == '%' && format[f + 1] == '%')
      {
        // ---------- Matching a literal '%'.
        f++;
        if (format[f] == *input)
          {
            input++;
            f++;
            continue;
          }
        else
          {
            return (ass_or_eof
                    (consumed, assigned));
          }
      }
    if (format[f] == '%')
      {
        // ---------- Percent of a specifier.
        f++;
        specifier = 1;
        specifier_flags = 1;
        continue;
      }
  }
//
if (specifier && specifier_flags)
  {
    // -------------------------------------
    // The context is inside
    // specifier flags.
    // -------------------------------------
    if (format[f] == '*')
      {
        // ----- Assignment suppression star.
```

```
                    flag_star = 1;
                    f++;
                  }
                else
                  {
                    // -----------------------------------
                    // End of flags and begin of
                    // specifier length.
                    // -----------------------------------
                    specifier_flags = 0;
                    specifier_width = 1;
                  }
              }
            //
            if (specifier && specifier_width)
              {
                // ---------------------------------------
                // The context is inside a
                // specifier width.
                // ---------------------------------------
                for (w = 0;
                     format[f] >= '0'
                     && format[f] <= '9'
                     && w < WIDTH_MAX; w++)
                  {
                    width_string[w] = format[f];
                    f++;
                  }
                width_string[w] = '\0';
                width = atoi (width_string);
                if (width > WIDTH_MAX)
                  {
                    width = WIDTH_MAX;
                  }
                //
                // ---------------------------------------
                // A zero width means an unspecified
              // limit for the field
              // length.
              // --------------------------------------
              // End of spec. width and
              // begin of spec. type.
              // --------------------------------------
              specifier_width = 0;
              specifier_type = 1;
            }
          //
          if (specifier && specifier_type)
            {
              //
              // Specifiers with length modifier.
              //
              if (format[f] == 'h' && format[f + 1] == 'h')
                {
                  // ---------------------------- char.
                  if (format[f + 2] == 'd')
                    {
                      // -------- signed char, base 10.
                      value_i =
                        strtointmax (input, &next, 10,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_schar =
                            va_arg (ap, signed char *);
                          *ptr_schar = value_i;
                          assigned++;
                        }
                      f += 3;
                      input = next;
                    }
                  else if (format[f + 2] == 'i')
                    {
                      // ------------------------------
                      // signed char, base unknown.
                      // ------------------------------
                      value_i =
                        strtointmax (input, &next, 0,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_schar =
                            va_arg (ap, signed char *);
                          *ptr_schar = value_i;
                          assigned++;
                        }
                      f += 3;
                      input = next;
                    }
                  else if (format[f + 2] == 'o')
                    {
                      // ------------------------------
                      // signed char, base 8.
                      // ------------------------------
                      value_i =
                        strtointmax (input, &next, 8,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_schar =
                            va_arg (ap, signed char *);
                          *ptr_schar = value_i;
                          assigned++;
                        }
                      f += 3;
                      input = next;
                    }
                  else if (format[f + 2] == 'u')
                    {
                      // -----------------------------
                      // unsigned char, base 10.
                      // -----------------------------
                      value_u =
                        strtointmax (input, &next, 10,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_uchar =
                            va_arg (ap, unsigned char *);
                          *ptr_uchar = value_u;
                          assigned++;
                        }
                      f += 3;
                      input = next;
```

```
                     }
                   else if (format[f + 2] == 'x'
                            || format[f + 2] == 'X')
                     {
                       // -------------------------------
                       // signed char, base 16.
                       // -------------------------------
                       value_i =
                         strtointmax (input, &next, 16,
                                      width);
                       if (input == next)
                         {
                           return (ass_or_eof
                                   (consumed, assigned));
                         }
                       consumed++;
                       if (!flag_star)
                         {
                           ptr_schar =
                             va_arg (ap, signed char *);
                           *ptr_schar = value_i;
                           assigned++;
                         }
                       f += 3;
                       input = next;
                     }
                   else if (format[f + 2] == 'n')
                     {
                       // -------------------------------
                       // signed char,
                       // string index counter.
                       // -------------------------------
                       ptr_schar =
                         va_arg (ap, signed char *);
                       *ptr_schar =
                         (signed char) (input - start +
                                        scanned);
```

```
        f += 3;
      }
    else
      {
        // ------------------------------
        // unsupported or
        // unknown specifier.
        // ------------------------------
        f += 2;
      }
  }
else if (format[f] == 'h')
  {
    // --------------------------- short.
    if (format[f + 1] == 'd')
      {
        // ------------------------------
        // signed short, base 10.
        // ------------------------------
        value_i =
          strtointmax (input, &next, 10,
                       width);
        if (input == next)
          {
            return (ass_or_eof
                    (consumed, assigned));
          }
        consumed++;
        if (!flag_star)
          {
            ptr_sshort =
              va_arg (ap, signed short *);
            *ptr_sshort = value_i;
            assigned++;
          }
        f += 2;
        input = next;
```

```
                    }
                  else if (format[f + 1] == 'i')
                    {
                      // ------------------------------
                      // signed
                      // short, base unknown.
                      // ------------------------------
                      value_i =
                        strtointmax (input, &next, 0,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_sshort =
                            va_arg (ap, signed short *);
                          *ptr_sshort = value_i;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
                  else if (format[f + 1] == 'o')
                    {
                      // ------------------------------
                      // signed short, base 8.
                      // ------------------------------
                      value_i =
                        strtointmax (input, &next, 8,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_sshort =
          va_arg (ap, signed short *);
        *ptr_sshort = value_i;
        assigned++;
      }
    f += 2;
    input = next;
  }
else if (format[f + 1] == 'u')
  {
    // -------------------------------
    // unsigned short, base 10.
    // -------------------------------
    value_u =
      strtointmax (input, &next, 10,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_ushort =
          va_arg (ap, unsigned short *);
        *ptr_ushort = value_u;
        assigned++;
      }
    f += 2;
    input = next;
  }
```

```
                  else if (format[f + 1] == 'x'
                           || format[f + 2] == 'X')
                    {
                      // ------------------------------
                      // signed short, base 16.
                      // ------------------------------
                      value_i =
                        strtointmax (input, &next, 16,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_sshort =
                            va_arg (ap, signed short *);
                          *ptr_sshort = value_i;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
                  else if (format[f + 1] == 'n')
                    {
                      // ------------------------------
                      // signed char,
                      // string index counter.
                      // ------------------------------
                      ptr_sshort =
                        va_arg (ap, signed short *);
                      *ptr_sshort =
                        (signed short) (input - start +
                                        scanned);
                      f += 2;
```

```
      }
    else
      {
        // ------------------------------
        // unsupported or
        // unknown specifier.
        // ------------------------------
        f += 1;
      }
  }
// --------- There is no ‘long long int’.
else if (format[f] == 'l')
  {
    // ------------------------ long int.
    if (format[f + 1] == 'd')
      {
        // ------------------------------
        // signed long, base 10.
        // ------------------------------
        value_i =
          strtointmax (input, &next, 10,
                       width);
        if (input == next)
          {
            return (ass_or_eof
                    (consumed, assigned));
          }
        consumed++;
        if (!flag_star)
          {
            ptr_slong =
              va_arg (ap, signed long *);
            *ptr_slong = value_i;
            assigned++;
          }
        f += 2;
        input = next;
  }
else if (format[f + 1] == 'i')
  {
    // ------------------------------
    // signed
    // long, base unknown.
    // ------------------------------
    value_i =
      strtointmax (input, &next, 0,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_slong =
          va_arg (ap, signed long *);
        *ptr_slong = value_i;
        assigned++;
      }
    f += 2;
    input = next;
  }
else if (format[f + 1] == 'o')
  {
    // ------------------------------
    // signed long, base 8.
    // ------------------------------
    value_i =
      strtointmax (input, &next, 8,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_slong =
                            va_arg (ap, signed long *);
                          *ptr_slong = value_i;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
                  else if (format[f + 1] == 'u')
                    {
                      // -----------------------------
                      // unsigned long, base 10.
                      // -----------------------------
                      value_u =
                        strtointmax (input, &next, 10,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_ulong =
                            va_arg (ap, unsigned long *);
                          *ptr_ulong = value_u;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
```

```
else if (format[f + 1] == 'x'
         || format[f + 2] == 'X')
  {
    // ------------------------------
    // signed long, base 16.
    // ------------------------------
    value_i =
      strtointmax (input, &next, 16,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_slong =
          va_arg (ap, signed long *);
        *ptr_slong = value_i;
        assigned++;
      }
    f += 2;
    input = next;
  }
else if (format[f + 1] == 'n')
  {
    // ------------------------------
    // signed char,
    // string index counter.
    // ------------------------------
    ptr_slong =
      va_arg (ap, signed long *);
    *ptr_slong =
      (signed long) (input - start +
                     scanned);
    f += 2;
```

```
                    }
                  else
                    {
                      // ------------------------------
                      // unsupported or
                      // unknown specifier.
                      // ------------------------------
                      f += 1;
                    }
                }
              else if (format[f] == 'j')
                {
                  // -----------------.------ intmax_t.
                  if (format[f + 1] == 'd')
                    {
                      // ----------- intmax_t, base 10.
                      value_i =
                        strtointmax (input, &next, 10,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_simax =
                            va_arg (ap, intmax_t *);
                          *ptr_simax = value_i;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
                  else if (format[f + 1] == 'i')
                    {
```

```
    // ------------------------------
    // intmax_t, base unknown.
    // ------------------------------
    value_i =
      strtointmax (input, &next, 0,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_simax =
          va_arg (ap, intmax_t *);
        *ptr_simax = value_i;
        assigned++;
      }
    f += 2;
    input = next;
  }
else if (format[f + 1] == 'o')
  {
    // ------------------------------
    // intmax_t, base 8.
    // ------------------------------
    value_i =
      strtointmax (input, &next, 8,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
```

```
                         {
                           ptr_simax =
                             va_arg (ap, intmax_t *);
                           *ptr_simax = value_i;
                           assigned++;
                         }
                       f += 2;
                       input = next;
                     }
                   else if (format[f + 1] == 'u')
                     {
                       // -------------------------------
                       // uintmax_t, base 10.
                       // -------------------------------
                       value_u =
                         strtointmax (input, &next, 10,
                                      width);
                       if (input == next)
                         {
                           return (ass_or_eof
                                   (consumed, assigned));
                         }
                       consumed++;
                       if (!flag_star)
                         {
                           ptr_uimax =
                             va_arg (ap, uintmax_t *);
                           *ptr_uimax = value_u;
                           assigned++;
                         }
                       f += 2;
                       input = next;
                     }
                   else if (format[f + 1] == 'x'
                            || format[f + 2] == 'X')
                     {
                       // -------------------------------
```

```
                    // intmax_t, base 16.
                    // ------------------------------
                    value_i =
                      strtointmax (input, &next, 16,
                                   width);
                    if (input == next)
                      {
                        return (ass_or_eof
                                (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_simax =
                          va_arg (ap, intmax_t *);
                        *ptr_simax = value_i;
                        assigned++;
                      }
                    f += 2;
                    input = next;
                  }
                else if (format[f + 1] == 'n')
                  {
                    // ------------------------------
                    // signed char,
                    // string index counter.
                    // ------------------------------
                    ptr_simax = va_arg (ap, intmax_t *);
                    *ptr_simax =
                      (intmax_t) (input - start +
                                  scanned);
                    f += 2;
                  }
                else
                  {
                    // ------------------------------
                    // unsupported or
```

```
        // unknown specifier.
        // ------------------------------
        f += 1;
      }
  }
else if (format[f] == 'z')
  {
    // -------------------------- size_t.
    if (format[f + 1] == 'd')
      {
        // ------------------------------
        // size_t, base 10.
        // ------------------------------
        value_i =
          strtointmax (input, &next, 10,
                       width);
        if (input == next)
          {
            return (ass_or_eof
                    (consumed, assigned));
          }
        consumed++;
        if (!flag_star)
          {
            ptr_size = va_arg (ap, size_t *);
            *ptr_size = value_i;
            assigned++;
          }
        f += 2;
        input = next;
      }
    else if (format[f + 1] == 'i')
      {
        // ------------------------------
        // size_t, base unknown.
        // ------------------------------
        value_i =
                        strtointmax (input, &next, 0,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_size = va_arg (ap, size_t *);
                          *ptr_size = value_i;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
                  else if (format[f + 1] == 'o')
                    {
                      // -------------------------------
                      // size_t, base 8.
                      // -------------------------------
                      value_i =
                        strtointmax (input, &next, 8,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_size = va_arg (ap, size_t *);
                          *ptr_size = value_i;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
                  else if (format[f + 1] == 'u')
                    {
                      // ------------------------------
                      // size_t, base 10.
                      // ------------------------------
                      value_u =
                        strtointmax (input, &next, 10,
                                     width);
                      if (input == next)
                        {
                          return (ass_or_eof
                                  (consumed, assigned));
                        }
                      consumed++;
                      if (!flag_star)
                        {
                          ptr_size = va_arg (ap, size_t *);
                          *ptr_size = value_u;
                          assigned++;
                        }
                      f += 2;
                      input = next;
                    }
                  else if (format[f + 1] == 'x'
                           || format[f + 2] == 'X')
                    {
                      // ------------------------------
                      // size_t, base 16.
                      // ------------------------------
                      value_i =
                        strtointmax (input, &next, 16,
                                     width);
                      if (input == next)
                        {
```

```
            return (ass_or_eof
                    (consumed, assigned));
          }
        consumed++;
        if (!flag_star)
          {
            ptr_size = va_arg (ap, size_t *);
            *ptr_size = value_i;
            assigned++;
          }
        f += 2;
        input = next;
      }
    else if (format[f + 1] == 'n')
      {
        // ------------------------------
        // signed char,
        // string index counter.
        // ------------------------------
        ptr_size = va_arg (ap, size_t *);
        *ptr_size =
          (size_t) (input - start + scanned);
        f += 2;
      }
    else
      {
        // ------------------------------
        // unsupported or
        // unknown specifier.
        // ------------------------------
        f += 1;
      }
  }
else if (format[f] == 't')
  {
    // ----------------------- ptrdiff_t.
    if (format[f + 1] == 'd')
  {
    // ------------------------------
    // ptrdiff_t, base 10.
    // ------------------------------
    value_i =
      strtointmax (input, &next, 10,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_ptrdiff =
          va_arg (ap, ptrdiff_t *);
        *ptr_ptrdiff = value_i;
        assigned++;
      }
    f += 2;
    input = next;
  }
else if (format[f + 1] == 'i')
  {
    // ------------------------------
    // ptrdiff_t, base unknown.
    // ------------------------------
    value_i =
      strtointmax (input, &next, 0,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_ptrdiff =
          va_arg (ap, ptrdiff_t *);
        *ptr_ptrdiff = value_i;
        assigned++;
      }
    f += 2;
    input = next;
  }
else if (format[f + 1] == 'o')
  {
    // -------------------------------
    // ptrdiff_t, base 8.
    // -------------------------------
    value_i =
      strtointmax (input, &next, 8,
                   width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_ptrdiff =
          va_arg (ap, ptrdiff_t *);
        *ptr_ptrdiff = value_i;
        assigned++;
      }
    f += 2;
    input = next;
  }
else if (format[f + 1] == 'u')
  {
    // -------------------------------
```

```
                        // ptrdiff_t, base 10.
                        // -------------------------------
                        value_u =
                          strtointmax (input, &next, 10,
                                       width);
                        if (input == next)
                          {
                            return (ass_or_eof
                                    (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_ptrdiff =
                              va_arg (ap, ptrdiff_t *);
                            *ptr_ptrdiff = value_u;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f + 1] == 'x'
                             || format[f + 2] == 'X')
                      {
                        // -------------------------------
                        // ptrdiff_t, base 16.
                        // -------------------------------
                        value_i =
                          strtointmax (input, &next, 16,
                                       width);
                        if (input == next)
                          {
                            return (ass_or_eof
                                    (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
```

```
                          {
                            ptr_ptrdiff =
                              va_arg (ap, ptrdiff_t *);
                            *ptr_ptrdiff = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f + 1] == 'n')
                      {
                        // ------------------------------
                        // signed char,
                        // string index counter.
                        // ------------------------------
                        ptr_ptrdiff =
                          va_arg (ap, ptrdiff_t *);
                        *ptr_ptrdiff =
                          (ptrdiff_t) (input - start +
                                       scanned);
                        f += 2;
                      }
                    else
                      {
                        // ------------------------------
                        // unsupported or
                        // unknown specifier.
                        // ------------------------------
                        f += 1;
                      }
                  }
                //
                // Specifiers with no length modifier.
                //
                if (format[f] == 'd')
                  {
                    // ---------- signed short, base 10.
                    value_i =
                      strtointmax (input, &next, 10, width);
                    if (input == next)
                      {
                        return (ass_or_eof
                                (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_sshort =
                          va_arg (ap, signed short *);
                        *ptr_sshort = value_i;
                        assigned++;
                      }
                    f += 1;
                    input = next;
                  }
                else if (format[f] == 'i')
                  {
                    // -----------------------------------
                    // signed
                    // int, base unknown.
                    // -----------------------------------
                    value_i =
                      strtointmax (input, &next, 0, width);
                    if (input == next)
                      {
                        return (ass_or_eof
                                (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_sint = va_arg (ap, signed int *);
                        *ptr_sint = value_i;
                        assigned++;
                    }
                  f += 1;
                  input = next;
                }
              else if (format[f] == 'o')
                {
                  // -------------------------------------
                  // signed int, base 8.
                  // -------------------------------------
                  value_i =
                    strtointmax (input, &next, 8, width);
                  if (input == next)
                    {
                      return (ass_or_eof
                              (consumed, assigned));
                    }
                  consumed++;
                  if (!flag_star)
                    {
                      ptr_sint = va_arg (ap, signed int *);
                      *ptr_sint = value_i;
                      assigned++;
                    }
                  f += 1;
                  input = next;
                }
              else if (format[f] == 'u')
                {
                  // -------------------------------------
                  // unsigned short, base 10.
                  // -------------------------------------
                  value_u =
                    strtointmax (input, &next, 10, width);
                  if (input == next)
                    {
                      return (ass_or_eof
                              (consumed, assigned));
```

```
               }
              consumed++;
              if (!flag_star)
               {
                ptr_uint =
                 va_arg (ap, unsigned int *);
                *ptr_uint = value_u;
                assigned++;
               }
              f += 1;
              input = next;
             }
            else if (format[f] == 'x' || format[f] == 'X')
             {
              // ---------------------------------
              // signed short, base 16.
              // ---------------------------------
              value_i =
               strtointmax (input, &next, 16, width);
              if (input == next)
               {
                return (ass_or_eof
                        (consumed, assigned));
               }
              consumed++;
              if (!flag_star)
               {
                ptr_sint = va_arg (ap, signed int *);
                *ptr_sint = value_i;
                assigned++;
               }
              f += 1;
              input = next;
             }
            else if (format[f] == 'c')
             {
              // -------------------------- char[].
```

```
                   if (width == 0)
                     width = 1;
                   //
                   if (!flag_star)
                     ptr_char = va_arg (ap, char *);
                   //
                   for (count = 0;
                        width > 0 && *input != 0;
                        width--, ptr_char++, input++)
                     {
                       if (!flag_star)
                         *ptr_char = *input;
                       //
                       count++;
                     }
                   //
                   if (count)
                     consumed++;
                   if (count && !flag_star)
                     assigned++;
                   //
                   f += 1;
                 }
               else if (format[f] == 's')
                 {
                   // -------------------------- string.
                   if (!flag_star)
                     ptr_char = va_arg (ap, char *);
                   //
                   for (count = 0;
                        !isspace (*input)
                        && *input != 0; ptr_char++, input++)
                     {
                       if (!flag_star)
                         *ptr_char = *input;
                       //
                       count++;
```

```
                      }
                    if (!flag_star)
                      *ptr_char = 0;
                    //
                    if (count)
                      consumed++;
                    if (count && !flag_star)
                      assigned++;
                    //
                    f += 1;
                  }
                else if (format[f] == '[')
                  {
                    //
                    f++;
                    //
                    if (format[f] == '^')
                      {
                        inverted = 1;
                        f++;
                      }
                    else
                      {
                        inverted = 0;
                      }
                    //
                    // Reset ascii array.
                    //
                    for (index = 0; index < 128; index++)
                      {
                        ascii[index] = inverted;
                      }
                    //
                    //
                    //
                    for (count = 0;
                         &format[f] < end_format; count++)
                    {
                      if (format[f] == ']' && count > 0)
                        {
                          break;
                        }
                      //
                      // Check for an interval.
                      //
                      if (format[f + 1] == '-'
                          && format[f + 2] != ']'
                          && format[f + 2] != 0)
                        {
                          //
                          // Interval.
                          //
                          for (index = format[f];
                               index <= format[f + 2];
                               index++)
                            {
                              ascii[index] = !inverted;
                            }
                          f += 3;
                          continue;
                        }
                      //
                      // Single character.
                      //
                      index = format[f];
                      ascii[index] = !inverted;
                      f++;
                    }
                  //
                  // Is the scan correctly finished?.
                  //
                  if (format[f] != ']')
                    {
                      return (ass_or_eof
              (consumed, assigned));
    }
  //
  // The ascii table is populated.
  //
  if (width == 0)
    width = SIZE_MAX;
  //
  // Scan the input string.
  //
  if (!flag_star)
    ptr_char = va_arg (ap, char *);
  //
  for (count = 0;
       width > 0 && *input != 0;
       width--, ptr_char++, input++)
    {
      index = *input;
      if (ascii[index])
        {
          if (!flag_star)
            *ptr_char = *input;
          count++;
        }
      else
        {
          break;
        }
    }
  //
  if (count)
    consumed++;
  if (count && !flag_star)
    assigned++;
  //
  f += 1;
}
```

```
else if (format[f] == 'p')
  {
    // -------------------------- void *.
    value_i =
      strtointmax (input, &next, 16, width);
    if (input == next)
      {
        return (ass_or_eof
                (consumed, assigned));
      }
    consumed++;
    if (!flag_star)
      {
        ptr_void = va_arg (ap, void **);
        *ptr_void = (void *) ((int) value_i);
        assigned++;
      }
    f += 1;
    input = next;
  }
else if (format[f] == 'n')
  {
    // ----------------------------------
    // signed char,
    // string index counter.
    // ----------------------------------
    ptr_sint = va_arg (ap, signed int *);
    *ptr_sint =
      (signed char) (input - start + scanned);
    f += 1;
  }
else
  {
    // ----------------------------------
    // unsupported or
    // unknown specifier.
    // ----------------------------------
```

```
                  ;
                }

              // --------------------------------------
              // End of specifier.
              // --------------------------------------

              width_string[0] = '\0';
              specifier = 0;
              specifier_flags = 0;
              specifier_width = 0;
              specifier_type = 0;
              flag_star = 0;

            }
        }
      //
      // The format or the input string is terminated.
      //
      if (&format[f] < end_format && stream)
        {
          //
          // Only the input string is finished, and
          // the input comes
          // from a stream, so another read will be
          // done.
          //
          scanned += (int) (input - start);
          continue;
        }
      //
      // The format string is terminated.
      //
      return (ass_or_eof (consumed, assigned));
    }
}
```

```
//----------------------------------------------------------
static intmax_t
strtointmax (const char *restrict string,
             const char **restrict endptr, int base,
             size_t max_width)
{
  int i;
  int d;        // Digits counter.
  int sign = +1;
  intmax_t number;
  intmax_t previous;
  int digit;
  //
  bool flag_prefix_oct = 0;
  bool flag_prefix_exa = 0;
  bool flag_prefix_dec = 0;
  //
  // If the 'max_width' value is zero, fix it to the
  // maximum
  // that it can represent.
  //
  if (max_width == 0)
    {
      max_width = SIZE_MAX;
    }
  //
  // Eat initial spaces, but if there are spaces,
  // there is an
  // error inside the calling function!
  //
  for (i = 0; isspace (string[i]); i++)
    {
      fprintf (stderr,
               "libc error: file \"%s\", line %i\n",
               __FILE__, __LINE__);
      ;
    }
    //
    // Check sign. The ‘max_width’ counts also the sign,
    // if there is
    // one.
    //
    if (string[i] == '+')
      {
        sign = +1;
        i++;
        max_width--;
      }
    else if (string[i] == '-')
      {
        sign = -1;
        i++;
        max_width--;
      }
    //
    // Check for prefix.
    //
    if (string[i] == '0')
      {
        if (string[i + 1] == 'x' || string[i + 1] == 'X')
          {
            flag_prefix_exa = 1;
          }
        if (isdigit (string[i + 1]))
          {
            flag_prefix_oct = 1;
          }
      }
    //
    if (string[i] > '0' && string[i] <= '9')
      {
        flag_prefix_dec = 1;
      }
    //
    // Check compatibility with requested base.
    //
    if (flag_prefix_exa)
      {
        if (base == 0)
          {
            base = 16;
          }
        else if (base == 16)
          {
            ;     // Ok.
          }
        else
          {
            //
            // Incompatible sequence: only the initial
            // zero is reported.
            //
            *endptr = &string[i + 1];
            return ((intmax_t) 0);
          }
        //
        // Move on, after the '0x' prefix.
        //
        i += 2;
      }
    //
    if (flag_prefix_oct)
      {
        if (base == 0)
          {
            base = 8;
          }
        //
        // Move on, after the '0' prefix.
        //
        i += 1;
      }
    //
    if (flag_prefix_dec)
      {
        if (base == 0)
          {
            base = 10;
          }
      }
    //
    // Scan the string.
    //
    for (d = 0, number = 0;
         d < max_width && string[i] != 0; i++, d++)
      {
        if (string[i] >= '0' && string[i] <= '9')
          {
            digit = string[i] - '0';
          }
        else if (string[i] >= 'A' && string[i] <= 'F')
          {
            digit = string[i] - 'A' + 10;
          }
        else if (string[i] >= 'a' && string[i] <= 'f')
          {
            digit = string[i] - 'a' + 10;
          }
        else
          {
            digit = 999;
          }
        //
        // Give a sign to the digit.
        //
        digit *= sign;
        //
        // Compare with the base.
      //
      if (base > (digit * sign))
        {
          //
          // Check if the current digit can be safely
          // computed.
          //
          previous = number;
          number *= base;
          number += digit;
          if (number / base != previous)
            {
              //
              // Out of range.
              //
              *endptr = &string[i + 1];
              errset (ERANGE);  // Result too large.
              if (sign > 0)
                {
                  return (INTMAX_MAX);
                }
              else
                {
                  return (INTMAX_MIN);
                }
            }
        }
      else
        {
          *endptr = &string[i];
          return (number);
        }
    }
  //
  // The string is finished or the max digits length
  // is reached.
  //
  *endptr = &string[i];
  //
  return (number);
}

//----------------------------------------------------------
static int
ass_or_eof (int consumed, int assigned)
{
  if (consumed == 0)
    {
      return (EOF);
    }
  else
    {
      return (assigned);
    }
}

//----------------------------------------------------------
```

95.18.40 lib/stdio/vprintf.c

Si veda la sezione 88.137.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <string.h>
#include <unistd.h>
//----------------------------------------------------------
int
vprintf (const char *restrict format, va_list arg)
{
  ssize_t size_written;
  size_t size;
  size_t size_total;
  int status;
  char string[BUFSIZ];
  char *buffer = string;

  buffer[0] = 0;
  status = vsprintf (buffer, format, arg);

  size = strlen (buffer);
  if (size >= BUFSIZ)
    {
      size = BUFSIZ;
    }

  for (size_total = 0, size_written = 0;
       size_total < size;
       size_total += size_written, buffer += size_written)
    {
      //
      // Write to the standard output: file descriptor
      // n. 1.
      //
      size_written =
        write (STDOUT_FILENO, buffer, size - size_total);
      if (size_written < 0)
        {
          return (size_total);
        }
    }
  return (size);
}
```

95.18.41 lib/stdio/vscanf.c

Si veda la sezione 88.138.

```
#include <stdio.h>
//------------------------------------------------------------
```

```
int
vscanf (const char *restrict format, va_list ap)
{
  return (vfscanf (stdin, format, ap));
}

//------------------------------------------------------------
```

95.18.42 lib/stdio/vsnprintf.c

Si veda la sezione 88.137.

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
//------------------------------------------------------------
static size_t uimaxtoa (uintmax_t integer,
                         char *buffer, int base,
                         int uppercase, size_t size);
static size_t imaxtoa (intmax_t integer, char *buffer,
                        int base, int uppercase,
                        size_t size);
static size_t simaxtoa (intmax_t integer, char *buffer,
                         int base, int uppercase,
                         size_t size);
static size_t uimaxtoa_fill (uintmax_t integer,
                              char *buffer, int base,
                              int uppercase, int width,
                              int filler, int max);
static size_t imaxtoa_fill (intmax_t integer,
                             char *buffer, int base,
                             int uppercase, int width,
                             int filler, int max);
static size_t simaxtoa_fill (intmax_t integer,
                              char *buffer, int base,
                             int uppercase, int width,
                             int filler, int max);
static size_t strtostr_fill (char *string,
                             char *buffer, int width,
                             int filler, int max);
//----------------------------------------------------------
int
vsnprintf (char *restrict string, size_t size,
           const char *restrict format, va_list ap)
{
  //
  // We produce at most ‘size-1’ characters, + ’\0’.
  // ‘size’ is used also as the max size for internal
  // strings, but only if it is not too big.
  //
  int f = 0;
  int s = 0;
  int remain = size - 1;
  //
  bool specifier = 0;
  bool specifier_flags = 0;
  bool specifier_width = 0;
  bool specifier_precision = 0;
  bool specifier_type = 0;
  //
  bool flag_plus = 0;
  bool flag_minus = 0;
  bool flag_space = 0;
  bool flag_alternate = 0;
  bool flag_zero = 0;
  //
  int alignment;
  int filler;
  //
  intmax_t value_i;
  uintmax_t value_ui;
  char *value_cp;
  //
  size_t width;
  size_t precision;
  size_t str_size =
    (size > (BUFSIZ / 2) ? (BUFSIZ / 2) : size);
  char width_string[str_size];
  char precision_string[str_size];
  int w;
  int p;
  //
  width_string[0] = '\0';
  precision_string[0] = '\0';
  //
  while (format[f] != 0 && s < (size - 1))
    {
      if (!specifier)
        {
          // ------------------- The context is not
          // inside a specifier.
          if (format[f] != '%')
            {
              string[s] = format[f];
              s++;
              remain--;
              f++;
              continue;
            }
          if (format[f] == '%' && format[f + 1] == '%')
            {
              string[s] = '%';
              f++;
              f++;
              s++;
              remain--;
              continue;
            }
          if (format[f] == '%')
```

```
            {
              f++;
              specifier = 1;
              specifier_flags = 1;
              continue;
            }
        }
      //
      if (specifier && specifier_flags)
        {
          // ------------------- The context is inside
          // specifier flags.
          if (format[f] == '+')
            {
              flag_plus = 1;
              f++;
              continue;
            }
          else if (format[f] == '-')
            {
              flag_minus = 1;
              f++;
              continue;
            }
          else if (format[f] == ' ')
            {
              flag_space = 1;
              f++;
              continue;
            }
          else if (format[f] == '#')
            {
              flag_alternate = 1;
              f++;
              continue;
            }
          else if (format[f] == '0')
```

```
            {
              flag_zero = 1;
              f++;
              continue;
            }
          else
            {
              specifier_flags = 0;
              specifier_width = 1;
            }
        }
      //
      if (specifier && specifier_width)
        {
          // ------------------- The context is inside
          // specifier width.
          for (w = 0;
               format[f] >= '0' && format[f] <= '9'
               && w < str_size; w++)
            {
              width_string[w] = format[f];
              f++;
            }
          width_string[w] = '\0';

          specifier_width = 0;

          if (format[f] == '.')
            {
              specifier_precision = 1;
              f++;
            }
          else
            {
              specifier_precision = 0;
              specifier_type = 1;
            }
          }
        //
        if (specifier && specifier_precision)
          {
            // --------------- The context is inside
            // specifier precision.
            for (p = 0;
                 format[f] >= '0' && format[f] <= '9'
                 && p < str_size; p++)
              {
                precision_string[p] = format[f];
                p++;
              }
            precision_string[p] = '\0';

            specifier_precision = 0;
            specifier_type = 1;
          }
        //
        if (specifier && specifier_type)
          {
            // -------------------- The context is
            // inside specifier type.
            width = atoi (width_string);
            precision = atoi (precision_string);
            filler = ' ';
            if (flag_zero)
              filler = '0';
            if (flag_space)
              filler = ' ';
            alignment = width;
            if (flag_minus)
              {
                alignment = -alignment;
                filler = ' ';     // The filler
                // character cannot
                // be zero, so it is black.
```

```
  }
//
if (format[f] == 'h' && format[f + 1] == 'h')
  {
    if (format[f + 2] == 'd'
        || format[f + 2] == 'i')
      {
        // ----------------------------
        // signed char, base 10.
        value_i = va_arg (ap, int);
        if (flag_plus)
          {
            s +=
              simaxtoa_fill (value_i,
                             &string[s], 10,
                             0, alignment,
                             filler, remain);
          }
        else
          {
            s +=
              imaxtoa_fill (value_i,
                            &string[s], 10,
                            0, alignment,
                            filler, remain);
          }
        f += 3;
      }
    else if (format[f + 2] == 'u')
      {
        // --------------------------
        // unsigned char, base 10.
        value_ui = va_arg (ap, unsigned int);
        s +=
          uimaxtoa_fill (value_ui,
                         &string[s], 10, 0,
                         alignment, filler,
                                   remain);
                  f += 3;
                }
              else if (format[f + 2] == 'o')
                {
                  // -------------------------
                  // unsigned char, base 8.
                  value_ui = va_arg (ap, unsigned int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 8, 0,
                                   alignment, filler,
                                   remain);
                  f += 3;
                }
              else if (format[f + 2] == 'x')
                {
                  // -------------------------
                  // unsigned char, base 16.
                  value_ui = va_arg (ap, unsigned int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 16, 0,
                                   alignment, filler,
                                   remain);
                  f += 3;
                }
              else if (format[f + 2] == 'X')
                {
                  // -------------------------
                  // unsigned char, base 16.
                  value_ui = va_arg (ap, unsigned int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 16, 1,
                                   alignment, filler,
                                   remain);
                  f += 3;
                }
              else if (format[f + 2] == 'b')
                {
                  // ---------------- unsigned char,
                  // base 2 (extention).
                  value_ui = va_arg (ap, unsigned int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 2, 0,
                                   alignment, filler,
                                   remain);
                  f += 3;
                }
              else
                {
                  // ---------------- unsupported or
                  // unknown specifier.
                  f += 2;
                }
            }
          else if (format[f] == 'h')
            {
              if (format[f + 1] == 'd'
                  || format[f + 1] == 'i')
                {
                  // ------------------------------
                  // short int, base 10.
                  value_i = va_arg (ap, int);
                  if (flag_plus)
                    {
                      s +=
                        simaxtoa_fill (value_i,
                                       &string[s], 10,
                                       0, alignment,
                                       filler, remain);
                    }
```

```
                   else
                     {
                       s +=
                         imaxtoa_fill (value_i,
                                       &string[s], 10,
                                       0, alignment,
                                       filler, remain);
                     }
                   f += 2;
                 }
               else if (format[f + 1] == ’u’)
                 {
                   // --------------------- unsigned
                   // short int, base 10.
                   value_ui = va_arg (ap, unsigned int);
                   s +=
                     uimaxtoa_fill (value_ui,
                                    &string[s], 10, 0,
                                    alignment, filler,
                                    remain);
                   f += 2;
                 }
               else if (format[f + 1] == ’o’)
                 {
                   // ---------------------- unsigned
                   // short int, base 8.
                   value_ui = va_arg (ap, unsigned int);
                   s +=
                     uimaxtoa_fill (value_ui,
                                    &string[s], 8, 0,
                                    alignment, filler,
                                    remain);
                   f += 2;
                 }
               else if (format[f + 1] == ’x’)
                 {
                   // --------------------- unsigned
```

```
    // short int, base 16.
    value_ui = va_arg (ap, unsigned int);
    s +=
      uimaxtoa_fill (value_ui,
                     &string[s], 16, 0,
                     alignment, filler,
                     remain);
    f += 2;
  }
else if (format[f + 1] == 'X')
  {
    // --------------------- unsigned
    // short int, base 16.
    value_ui = va_arg (ap, unsigned int);
    s +=
      uimaxtoa_fill (value_ui,
                     &string[s], 16, 1,
                     alignment, filler,
                     remain);
    f += 2;
  }
else if (format[f + 1] == 'b')
  {
    // ---------- unsigned short int,
    // base 2 (extention).
    value_ui = va_arg (ap, unsigned int);
    s +=
      uimaxtoa_fill (value_ui,
                     &string[s], 2, 0,
                     alignment, filler,
                     remain);
    f += 2;
  }
else
  {
    // ---------------- unsupported or
    // unknown specifier.
```

```
                  f += 1;
                }
            }
          else if (format[f] == 'l' && format[f + 1] != 'l')
            {
              if (format[f + 1] == 'd'
                  || format[f + 1] == 'i')
                {
                  // ---------------------------------
                  // long int base 10.
                  value_i = va_arg (ap, long int);
                  if (flag_plus)
                    {
                      s +=
                        simaxtoa_fill (value_i,
                                       &string[s], 10,
                                       0, alignment,
                                       filler, remain);
                    }
                  else
                    {
                      s +=
                        imaxtoa_fill (value_i,
                                      &string[s], 10,
                                      0, alignment,
                                      filler, remain);
                    }
                  f += 2;
                }
              else if (format[f + 1] == 'u')
                {
                  // ----------------------- Unsigned
                  // long int base 10.
                  value_ui = va_arg (ap, unsigned long int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 10, 0,
                                   alignment, filler,
                                   remain);
                  f += 2;
                }
              else if (format[f + 1] == 'o')
                {
                  // ------------------------ Unsigned
                  // long int base 8.
                  value_ui = va_arg (ap, unsigned long int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 8, 0,
                                   alignment, filler,
                                   remain);
                  f += 2;
                }
              else if (format[f + 1] == 'x')
                {
                  // ----------------------- Unsigned
                  // long int base 16.
                  value_ui = va_arg (ap, unsigned long int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 16, 0,
                                   alignment, filler,
                                   remain);
                  f += 2;
                }
              else if (format[f + 1] == 'X')
                {
                  // ----------------------- Unsigned
                  // long int base 16.
                  value_ui = va_arg (ap, unsigned long int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 16, 1,
                                   alignment, filler,
                                    remain);
                   f += 2;
                 }
               else if (format[f + 1] == 'b')
                 {
                   // ------------ Unsigned long int
                   // base 2 (extention).
                   value_ui = va_arg (ap, unsigned long int);
                   s +=
                     uimaxtoa_fill (value_ui,
                                    &string[s], 2, 0,
                                    alignment, filler,
                                    remain);
                   f += 2;
                 }
               else
                 {
                   // ---------------- unsupported or
                   // unknown specifier.
                   f += 1;
                 }
             }
           else if (format[f] == 'l' && format[f + 1] == 'l')
             {
               if (format[f + 2] == 'd'
                   || format[f + 2] == 'i')
                 {
                   // --------------------------------
                   // long int base 10.
                   value_i = va_arg (ap, long long int);
                   if (flag_plus)
                     {
                       s +=
                         simaxtoa_fill (value_i,
                                        &string[s], 10,
                                        0, alignment,
                                        filler, remain);
```

```
                    }
                  else
                    {
                      s +=
                        imaxtoa_fill (value_i,
                                      &string[s], 10,
                                      0, alignment,
                                      filler, remain);
                    }
                  f += 3;
                }
              else if (format[f + 2] == 'u')
                {
                  // ----------------------- Unsigned
                  // long int base 10.
                  value_ui =
                    va_arg (ap, unsigned long long int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 10, 0,
                                   alignment, filler,
                                   remain);
                  f += 3;
                }
              else if (format[f + 2] == 'o')
                {
                  // ------------------------ Unsigned
                  // long int base 8.
                  value_ui =
                    va_arg (ap, unsigned long long int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 8, 0,
                                   alignment, filler,
                                   remain);
                  f += 3;
                }
```

```
            else if (format[f + 2] == 'x')
              {
                // ----------------------- Unsigned
                // long int base 16.
                value_ui =
                  va_arg (ap, unsigned long long int);
                s +=
                  uimaxtoa_fill (value_ui,
                                 &string[s], 16, 0,
                                 alignment, filler,
                                 remain);
                f += 3;
              }
            else if (format[f + 2] == 'X')
              {
                // ----------------------- Unsigned
                // long int base 16.
                value_ui =
                  va_arg (ap, unsigned long long int);
                s +=
                  uimaxtoa_fill (value_ui,
                                 &string[s], 16, 1,
                                 alignment, filler,
                                 remain);
                f += 3;
              }
            else if (format[f + 2] == 'b')
              {
                // ------------ Unsigned long int
                // base 2 (extention).
                value_ui =
                  va_arg (ap, unsigned long long int);
                s +=
                  uimaxtoa_fill (value_ui,
                                 &string[s], 2, 0,
                                 alignment, filler,
                                 remain);
                  f += 3;
                }
              else
                {
                  // ---------------- unsupported or
                  // unknown specifier.
                  f += 2;
                }
            }
          else if (format[f] == 'j')
            {
              if (format[f + 1] == 'd'
                  || format[f + 1] == 'i')
                {
                  // -------------------------------
                  // intmax_t base 10.
                  value_i = va_arg (ap, intmax_t);
                  if (flag_plus)
                    {
                      s +=
                        simaxtoa_fill (value_i,
                                       &string[s], 10,
                                       0, alignment,
                                       filler, remain);
                    }
                  else
                    {
                      s +=
                        imaxtoa_fill (value_i,
                                      &string[s], 10,
                                      0, alignment,
                                      filler, remain);
                    }
                  f += 2;
                }
              else if (format[f + 1] == 'u')
                {
                  // -------------------------------
                  // uintmax_t base 10.
                  value_ui = va_arg (ap, uintmax_t);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 10, 0,
                                   alignment, filler,
                                   remain);
                  f += 2;
                }
              else if (format[f + 1] == 'o')
                {
                  // --------------------------------
                  // uintmax_t base 8.
                  value_ui = va_arg (ap, uintmax_t);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 8, 0,
                                   alignment, filler,
                                   remain);
                  f += 2;
                }
              else if (format[f + 1] == 'x')
                {
                  // -------------------------------
                  // uintmax_t base 16.
                  value_ui = va_arg (ap, uintmax_t);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 16, 0,
                                   alignment, filler,
                                   remain);
                  f += 2;
                }
              else if (format[f + 1] == 'X')
                {
                  // -------------------------------
```

```
                    // uintmax_t base 16.
                    value_ui = va_arg (ap, uintmax_t);
                    s +=
                      uimaxtoa_fill (value_ui,
                                     &string[s], 16, 1,
                                     alignment, filler,
                                     remain);
                    f += 2;
                  }
                else if (format[f + 1] == 'b')
                  {
                    // -------------------- uintmax_t
                    // base 2 (extention).
                    value_ui = va_arg (ap, uintmax_t);
                    s +=
                      uimaxtoa_fill (value_ui,
                                     &string[s], 2, 0,
                                     alignment, filler,
                                     remain);
                    f += 2;
                  }
                else
                  {
                    // ---------------- unsupported or
                    // unknown specifier.
                    f += 1;
                  }
              }
            else if (format[f] == 'z')
              {
                if (format[f + 1] == 'd'
                    || format[f + 1] == 'i'
                    || format[f + 1] == 'i')
                  {
                    // ---------------- size_t base 10.
                    value_ui = va_arg (ap, unsigned long int);
                    s +=
                     uimaxtoa_fill (value_ui,
                                    &string[s], 10, 0,
                                    alignment, filler,
                                    remain);
                   f += 2;
                 }
               else if (format[f + 1] == 'o')
                 {
                   // ------------------ size_t base 8.
                   value_ui = va_arg (ap, unsigned long int);
                   s +=
                     uimaxtoa_fill (value_ui,
                                    &string[s], 8, 0,
                                    alignment, filler,
                                    remain);
                   f += 2;
                 }
               else if (format[f + 1] == 'x')
                 {
                   // ----------------- size_t base 16.
                   value_ui = va_arg (ap, unsigned long int);
                   s +=
                     uimaxtoa_fill (value_ui,
                                    &string[s], 16, 0,
                                    alignment, filler,
                                    remain);
                   f += 2;
                 }
               else if (format[f + 1] == 'X')
                 {
                   // ----------------- size_t base 16.
                   value_ui = va_arg (ap, unsigned long int);
                   s +=
                     uimaxtoa_fill (value_ui,
                                    &string[s], 16, 1,
                                    alignment, filler,
                                    remain);
```

```
                  f += 2;
                }
              else if (format[f + 1] == 'b')
                {
                  // ---------------------- size_t
                  // base 2 (extention).
                  value_ui = va_arg (ap, unsigned long int);
                  s +=
                    uimaxtoa_fill (value_ui,
                                   &string[s], 2, 0,
                                   alignment, filler,
                                   remain);
                  f += 2;
                }
              else
                {
                  // --------------- unsupported or
                  // unknown specifier.
                  f += 1;
                }
            }
          else if (format[f] == 't')
            {
              if (format[f + 1] == 'd'
                  || format[f + 1] == 'i')
                {
                  // ------------------------------
                  // ptrdiff_t base 10.
                  value_i = va_arg (ap, long int);
                  if (flag_plus)
                    {
                      s +=
                        simaxtoa_fill (value_i,
                                       &string[s], 10,
                                       0, alignment,
                                       filler, remain);
                    }
                    else
                      {
                        s +=
                          imaxtoa_fill (value_i,
                                        &string[s], 10,
                                        0, alignment,
                                        filler, remain);
                      }
                    f += 2;
                  }
                else if (format[f + 1] == 'u')
                  {
                    // ----------------- ptrdiff_t base
                    // 10, without sign.
                    value_ui = va_arg (ap, unsigned long int);
                    s +=
                      uimaxtoa_fill (value_ui,
                                     &string[s], 10, 0,
                                     alignment, filler,
                                     remain);
                    f += 2;
                  }
                else if (format[f + 1] == 'o')
                  {
                    // ------------------ ptrdiff_t base
                    // 8, without sign.
                    value_ui = va_arg (ap, unsigned long int);
                    s +=
                      uimaxtoa_fill (value_ui,
                                     &string[s], 8, 0,
                                     alignment, filler,
                                     remain);
                    f += 2;
                  }
                else if (format[f + 1] == 'x')
                  {
                    // ----------------- ptrdiff_t base
                    // 16, without sign.
                    value_ui = va_arg (ap, unsigned long int);
                    s +=
                      uimaxtoa_fill (value_ui,
                                     &string[s], 16, 0,
                                     alignment, filler,
                                     remain);
                    f += 2;
                  }
                else if (format[f + 1] == 'X')
                  {
                    // ----------------- ptrdiff_t base
                    // 16, without sign.
                    value_ui = va_arg (ap, unsigned long int);
                    s +=
                      uimaxtoa_fill (value_ui,
                                     &string[s], 16, 1,
                                     alignment, filler,
                                     remain);
                    f += 2;
                  }
                else if (format[f + 1] == 'b')
                  {
                    // ------ ptrdiff_t base 2, without
                    // sign (extention).
                    value_ui = va_arg (ap, unsigned long int);
                    s +=
                      uimaxtoa_fill (value_ui,
                                     &string[s], 2, 0,
                                     alignment, filler,
                                     remain);
                    f += 2;
                  }
                else
                  {
                    // --------------- unsupported or
                    // unknown specifier.
                  f += 1;
                }
            }
          if (format[f] == 'd' || format[f] == 'i')
            {
              // ------------------------ int base 10.
              value_i = va_arg (ap, int);
              if (flag_plus)
                {
                  s +=
                    simaxtoa_fill (value_i, &string[s],
                                   10, 0, alignment,
                                   filler, remain);
                }
              else
                {
                  s +=
                    imaxtoa_fill (value_i, &string[s],
                                  10, 0, alignment,
                                  filler, remain);
                }
              f += 1;
            }
          else if (format[f] == 'u')
            {
              // --------------------------------
              // unsigned int base 10.
              value_ui = va_arg (ap, unsigned int);
              s +=
                uimaxtoa_fill (value_ui, &string[s],
                               10, 0, alignment,
                               filler, remain);
              f += 1;
            }
          else if (format[f] == 'o')
            {
              // ---------------- unsigned int base 8.
              value_ui = va_arg (ap, unsigned int);
              s +=
                uimaxtoa_fill (value_ui, &string[s], 8,
                               0, alignment, filler,
                               remain);
              f += 1;
            }
          else if (format[f] == 'x')
            {
              // ---------------------------------
              // unsigned int base 16.
              value_ui = va_arg (ap, unsigned int);
              s +=
                uimaxtoa_fill (value_ui, &string[s],
                               16, 0, alignment,
                               filler, remain);
              f += 1;
            }
          else if (format[f] == 'X')
            {
              // ---------------------------------
              // unsigned int base 16.
              value_ui = va_arg (ap, unsigned int);
              s +=
                uimaxtoa_fill (value_ui, &string[s],
                               16, 1, alignment,
                               filler, remain);
              f += 1;
            }
          else if (format[f] == 'b')
            {
              // --------------------- unsigned int
              // base 2 (extention).
              value_ui = va_arg (ap, unsigned int);
              s +=
                uimaxtoa_fill (value_ui, &string[s], 2,
                               0, alignment, filler,
                               remain);
              f += 1;
            }
          else if (format[f] == 'c')
            {
              // ---------------------- unsigned char.
              value_ui = va_arg (ap, unsigned int);
              string[s] = (char) value_ui;
              s += 1;
              f += 1;
            }
          else if (format[f] == 's')
            {
              // ----------------------------- string.
              value_cp = va_arg (ap, char *);
              filler = ' ';

              s +=
                strtostr_fill (value_cp, &string[s],
                               alignment, filler, remain);
              f += 1;
            }
          else
            {
              // -------------------- unsupported or
              // unknown specifier.
              ;
            }
          // -----------------------------------------
          // End of specifier.
          // -----------------------------------------
          width_string[0] = '\0';
          precision_string[0] = '\0';

          specifier = 0;
          specifier_flags = 0;
          specifier_width = 0;
          specifier_precision = 0;
          specifier_type = 0;

          flag_plus = 0;
          flag_minus = 0;
          flag_space = 0;
          flag_alternate = 0;
          flag_zero = 0;
        }
    }
  string[s] = '\0';
  return s;
}

//----------------------------------------------------------
// Static functions.
//----------------------------------------------------------
static size_t
uimaxtoa (uintmax_t integer, char *buffer, int base,
          int uppercase, size_t size)
{
  // -------------------------------------------------
  // Convert a maximum rank integer into a string.
  // -------------------------------------------------

  uintmax_t integer_copy = integer;
  size_t digits;
  int b;
  unsigned char remainder;

  for (digits = 0; integer_copy > 0; digits++)
    {
      integer_copy = integer_copy / base;
    }

  if (buffer == NULL && integer == 0)
    return 1;
    if (buffer == NULL && integer > 0)
      return digits;

    if (integer == 0)
      {
        buffer[0] = '0';
        buffer[1] = '\0';
        return 1;
      }
    //
    // Fix the maximum number of digits.
    //
    if (size > 0 && digits > size)
      digits = size;
    //
    *(buffer + digits) = '\0';    // End of string.

    for (b = digits - 1; integer != 0 && b >= 0; b--)
      {
        remainder = integer % base;
        integer = integer / base;

        if (remainder <= 9)
          {
            *(buffer + b) = remainder + '0';
          }
        else
          {
            if (uppercase)
              {
                *(buffer + b) = remainder - 10 + 'A';
              }
            else
              {
                *(buffer + b) = remainder - 10 + 'a';
              }
          }
    }
  return digits;
}

//----------------------------------------------------------
static size_t
imaxtoa (intmax_t integer, char *buffer, int base,
         int uppercase, size_t size)
{
  // -------------------------------------------------
  // Convert a maximum rank integer with sign into a
  // string.
  // -------------------------------------------------

  if (integer >= 0)
    {
      return uimaxtoa (integer, buffer, base,
                       uppercase, size);
    }
  //
  // At this point, there is a negative number, less
  // than zero.
  //
  if (buffer == NULL)
    {
      return uimaxtoa (-integer, NULL, base, uppercase,
                       size) + 1;
    }

  *buffer = '-';          // The minus sign is needed at
  // the beginning.
  if (size == 1)
    {
      *(buffer + 1) = '\0';
      return 1;
    }
  else
```

```
    {
      return uimaxtoa (-integer, buffer + 1, base,
                       uppercase, size - 1) + 1;
    }
}

//----------------------------------------------------------
static size_t
simaxtoa (intmax_t integer, char *buffer, int base,
          int uppercase, size_t size)
{
  // -------------------------------------------------
  // Convert a maximum rank integer with sign into a
  // string, placing
  // the sign also if it is positive.
  // -------------------------------------------------

  if (buffer == NULL && integer >= 0)
    {
      return uimaxtoa (integer, NULL, base, uppercase,
                       size) + 1;
    }

  if (buffer == NULL && integer < 0)
    {
      return uimaxtoa (-integer, NULL, base, uppercase,
                       size) + 1;
    }
  //
  // At this point, 'buffer' is different from NULL.
  //
  if (integer >= 0)
    {
      *buffer = '+';
    }
  else
    {
      *buffer = '-';
    }

  if (size == 1)
    {
      *(buffer + 1) = '\0';
      return 1;
    }

  if (integer >= 0)
    {
      return uimaxtoa (integer, buffer + 1, base,
                       uppercase, size - 1) + 1;
    }
  else
    {
      return uimaxtoa (-integer, buffer + 1, base,
                       uppercase, size - 1) + 1;
    }
}

//----------------------------------------------------------
static size_t
uimaxtoa_fill (uintmax_t integer, char *buffer,
               int base, int uppercase, int width,
               int filler, int max)
{
  // -------------------------------------------------
  // Convert a maximum rank integer without sign into
  // a string,
  // takeing care of the alignment.
  // -------------------------------------------------

  size_t size_i;
  size_t size_f;

  if (max < 0)
```

```
    return 0;   // «max» deve essere un valore
  // positivo.

  size_i = uimaxtoa (integer, NULL, base, uppercase, 0);

  if (width > 0 && max > 0 && width > max)
    width = max;
  if (width < 0 && -max < 0 && width < -max)
    width = -max;

  if (size_i > abs (width))
    {
      return uimaxtoa (integer, buffer, base,
                       uppercase, abs (width));
    }

  if (width == 0 && max > 0)
    {
      return uimaxtoa (integer, buffer, base,
                       uppercase, max);
    }

  if (width == 0)
    {
      return uimaxtoa (integer, buffer, base,
                       uppercase, abs (width));
    }
  //
  // size_i <= abs (width).
  //
  size_f = abs (width) - size_i;

  if (width < 0)
    {
      // Left alignment.
      uimaxtoa (integer, buffer, base, uppercase, 0);
      memset (buffer + size_i, filler, size_f);
    }
  else
    {
      // Right alignment.
      memset (buffer, filler, size_f);
      uimaxtoa (integer, buffer + size_f, base,
                uppercase, 0);
    }
  *(buffer + abs (width)) = ’\0’;

  return abs (width);
}

//----------------------------------------------------------
static size_t
imaxtoa_fill (intmax_t integer, char *buffer, int base,
              int uppercase, int width, int filler, int max)
{
  // -------------------------------------------------
  // Convert a maximum rank integer with sign into a
  // string,
  // takeing care of the alignment.
  // -------------------------------------------------

  size_t size_i;
  size_t size_f;

  if (max < 0)
    return 0;   // ‘max’ must be a positive value.

  size_i = imaxtoa (integer, NULL, base, uppercase, 0);

  if (width > 0 && max > 0 && width > max)
    width = max;
  if (width < 0 && -max < 0 && width < -max)
    width = -max;

    if (size_i > abs (width))
      {
        return imaxtoa (integer, buffer, base, uppercase,
                        abs (width));
      }

    if (width == 0 && max > 0)
      {
        return imaxtoa (integer, buffer, base, uppercase,
                        max);
      }

    if (width == 0)
      {
        return imaxtoa (integer, buffer, base, uppercase,
                        abs (width));
      }

    // size_i <= abs (width).

    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Left alignment.
        imaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Right alignment.
        memset (buffer, filler, size_f);
        imaxtoa (integer, buffer + size_f, base,
                 uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

```

```
  return abs (width);
}

//----------------------------------------------------------
static size_t
simaxtoa_fill (intmax_t integer, char *buffer,
               int base, int uppercase, int width,
               int filler, int max)
{
  // -------------------------------------------------
  // Convert a maximum rank integer with sign into a
  // string,
  // placing the sign also if it is positive and
  // takeing care of the
  // alignment.
  // -------------------------------------------------

  size_t size_i;
  size_t size_f;

  if (max < 0)
    return 0;   // `max' must be a positive value.

  size_i = simaxtoa (integer, NULL, base, uppercase, 0);

  if (width > 0 && max > 0 && width > max)
    width = max;
  if (width < 0 && -max < 0 && width < -max)
    width = -max;

  if (size_i > abs (width))
    {
      return simaxtoa (integer, buffer, base,
                       uppercase, abs (width));
    }

  if (width == 0 && max > 0)
```

```
    {
      return simaxtoa (integer, buffer, base,
                       uppercase, max);
    }

  if (width == 0)
    {
      return simaxtoa (integer, buffer, base,
                       uppercase, abs (width));
    }
  //
  // size_i <= abs (width).
  //
  size_f = abs (width) - size_i;

  if (width < 0)
    {
      // Left alignment.
      simaxtoa (integer, buffer, base, uppercase, 0);
      memset (buffer + size_i, filler, size_f);
    }
  else
    {
      // Right alignment.
      memset (buffer, filler, size_f);
      simaxtoa (integer, buffer + size_f, base,
                uppercase, 0);
    }
  *(buffer + abs (width)) = '\0';

  return abs (width);
}

//----------------------------------------------------------
static size_t
strtostr_fill (char *string, char *buffer, int width,
               int filler, int max)
```

```
{
  // -------------------------------------------------
  // Transfer a string with care for the alignment.
  // -------------------------------------------------

  size_t size_s;
  size_t size_f;

  if (max < 0)
    return 0;   // ‘max’ must be a positive value.

  size_s = strlen (string);

  if (width > 0 && max > 0 && width > max)
    width = max;
  if (width < 0 && -max < 0 && width < -max)
    width = -max;

  if (width != 0 && size_s > abs (width))
    {
      memcpy (buffer, string, abs (width));
      buffer[width] = ’\0’;
      return width;
    }

  if (width == 0 && max > 0 && size_s > max)
    {
      memcpy (buffer, string, max);
      buffer[max] = ’\0’;
      return max;
    }

  if (width == 0 && max > 0 && size_s < max)
    {
      memcpy (buffer, string, size_s);
      buffer[size_s] = ’\0’;
      return size_s;
    }
  //
  // width =! 0
  // size_s <= abs (width)
  //
  size_f = abs (width) - size_s;

  if (width < 0)
    {
      // Right alignment.
      memset (buffer, filler, size_f);
      strncpy (buffer + size_f, string, size_s);
    }
  else
    {
      // Left alignment.
      strncpy (buffer, string, size_s);
      memset (buffer + size_s, filler, size_f);
    }
  *(buffer + abs (width)) = '\0';

  return abs (width);
}
```

95.18.43 lib/stdio/vsprintf.c

Si veda la sezione 88.137.

```
#include <stdio.h>
//----------------------------------------------------------
int
vsprintf (char *restrict string,
          const char *restrict format, va_list arg)
{
  return (vsnprintf (string, BUFSIZ, format, arg));
}
```

95.18.44 lib/stdio/vsscanf.c

Si veda la sezione 88.138.

```
#include <stdio.h>

//----------------------------------------------------------
int vfsscanf (FILE * restrict fp, const char *string,
              const char *restrict format, va_list ap);
//----------------------------------------------------------
int
vsscanf (const char *string,
         const char *restrict format, va_list ap)
{
  return (vfsscanf (NULL, string, format, ap));
}

//----------------------------------------------------------
```

95.19 os32: «lib/stdlib.h»

Si veda la sezione 91.3.

```
#ifndef _STDLIB_H
#define _STDLIB_H       1
//----------------------------------------------------------
#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>
#include <limits.h>
#include <restrict.h>
#include <stdint.h>
//----------------------------------------------------------
typedef struct
{
  int quot;
  int rem;
} div_t;
```

```
//-----------------------------------------------------------
typedef struct
{
  long int quot;
  long int rem;
} ldiv_t;
//-----------------------------------------------------------
typedef struct
{
  long long int quot;
  long long int rem;
} lldiv_t;
//-----------------------------------------------------------
typedef void (*atexit_t) (void);        // Non standard.
                                  // [1]
//
// [1] The type 'atexit_t' is a pointer to a function
//     for the "at exit" procedure, with no parameters
//     and returning void. With the declaration of type
//     'atexit_t', the function prototype of 'atexit()'
//     is easier to declare and to understand. Original
//     declaration is:
//
//     int atexit (void (*function) (void));
//
//-----------------------------------------------------------
typedef struct
{
  uintptr_t allocated:1, filler:1, next:30;
} _alloc_head_t;        // Non standard [2]
//
// [2] This is used for the 'malloc()' management, as
//     the pointer to the following element of memory,
//     that might be free or allocated.
//
// La dimensione di «uintptr_t» condiziona la struttura
// «mm_head_t» e la dimensione delle unità minime di
// memoria allocata. «uintptr_t» è da 32 bit, così
// l'immagine del kernel è allineata a blocchi da
// 32 bit e così deve essere anche per gli altri
// blocchi di memoria.
// Essendo i blocchi di memoria multipli di 32 bit, gli
// indirizzi sono sempre multipli di 4 (4 byte);
// pertanto, servono solo 30 bit per rappresentare
// l'indirizzo, che poi viene ottenuto moltiplicandolo
// per quattro. Di conseguenza, il bit meno
// significativo viene usato per annotare se il blocco
// di memoria è libero e il bit successivo non viene
// usato. Questo meccanismo potrebbe essere usato anche
// con un indirizzamento a 16 bit, dove servirebbero 15
// bit per indirizzi multipli di due byte.
//
//----------------------------------------------------------
#define EXIT_FAILURE    1
#define EXIT_SUCCESS    0
#define RAND_MAX        INT_MAX
#define MB_CUR_MAX      ((size_t) MB_LEN_MAX)
//----------------------------------------------------------
void _Exit (int status);
void abort (void);
int abs (int j);
int atexit (atexit_t function);
int atoi (const char *string);
long int atol (const char *string);
#define  calloc(b, s) (malloc ((b) * (s)))
div_t div (int numer, int denom);
void exit (int status);
void free (void *ptr);
char *getenv (const char *name);
long int labs (long int j);
long long int llabs (long long int j);
ldiv_t ldiv (long int numer, long int denom);
lldiv_t lldiv (long long int numer, long long int denom);
void *malloc (size_t size);
```

```
int putenv (const char *string);
void qsort (void *base, size_t nmemb, size_t size,
            int (*compare) (const void *, const void *));
int rand (void);
void *realloc (void *ptr, size_t size);
int setenv (const char *name, const char *value,
            int overwrite);
void srand (unsigned int seed);
long int strtol (const char *restrict string,
                 char **restrict endptr, int base);
unsigned long int strtoul (const char *restrict string,
                           char **restrict endptr,
                           int base);
//int             system    (const char *string);
int unsetenv (const char *name);
//----------------------------------------------------------
#endif
```

95.19.1 lib/stdlib/_Exit.c

Si veda la sezione 87.2.

```
#include <stdlib.h>
#include <sys/os32.h>
//----------------------------------------------------------
void
_Exit (int status)
{
  sysmsg_exit_t msg;
  //
  // Only the low eight bit are returned.
  //
  msg.status = (status & 0xFF);
  //
  //
  //
  sys (SYS_EXIT, &msg, (sizeof msg));
  //
  // Should not return from system call, but if it
  // does, loop
  // forever:
  //
  while (1);
}
```

95.19.2 lib/stdlib/abort.c

Si veda la sezione 88.2.

```
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
//----------------------------------------------------------
void
abort (void)
{
  pid_t pid;
  sighandler_t sig_previous;
  //
  // Set 'SIGABRT' to a default action.
  //
  sig_previous = signal (SIGABRT, SIG_DFL);
  //
  // If the previous action was something different
  // than symbolic
  // ones, configure again the previous action.
  //
  if (sig_previous != SIG_DFL &&
      sig_previous != SIG_IGN && sig_previous != SIG_ERR)
    {
      signal (SIGABRT, sig_previous);
    }
  //
  // Get current process ID and sent the signal.
  //
  pid = getpid ();
  kill (pid, SIGABRT);
  //
  // Second chance
  //
  for (;;)
    {
      signal (SIGABRT, SIG_DFL);
      pid = getpid ();
      kill (pid, SIGABRT);
    }
}
```

95.19.3 lib/stdlib/abs.c

Si veda la sezione 88.3.

```
#include <stdlib.h>
//-----------------------------------------------------------
int
abs (int j)
{
  if (j < 0)
    {
      return -j;
    }
  else
    {
      return j;
    }
}
```

95.19.4 lib/stdlib/atexit.c

«

Si veda la sezione 88.7.

```
#include <stdlib.h>
//----------------------------------------------------------
atexit_t _atexit_table[ATEXIT_MAX];
//----------------------------------------------------------
void
_atexit_setup (void)
{
  int a;
  //
  for (a = 0; a < ATEXIT_MAX; a++)
    {
      _atexit_table[a] = NULL;
    }
}

//----------------------------------------------------------
int
atexit (atexit_t function)
{
  int a;
  //
  if (function == NULL)
    {
      return (-1);
    }
  //
  for (a = 0; a < ATEXIT_MAX; a++)
    {
      if (_atexit_table[a] == NULL)
        {
          _atexit_table[a] = function;
          return (0);
        }
    }
  //
  return (-1);
}
```

95.19.5 lib/stdlib/atoi.c

Si veda la sezione 88.8.

```
#include <stdlib.h>
#include <ctype.h>
//----------------------------------------------------------
int
atoi (const char *string)
{
  int i;
  int sign = +1;
  int number;
  //
  for (i = 0; isspace (string[i]); i++)
    {
      ;
    }
  //
  if (string[i] == '+')
    {
      sign = +1;
      i++;
    }
  else if (string[i] == '-')
    {
      sign = -1;
      i++;
    }
  //
  for (number = 0; isdigit (string[i]); i++)
    {
      number *= 10;
      number += (string[i] - '0');
    }
  //
  number *= sign;
  //
  return number;
}
```

95.19.6 lib/stdlib/atol.c

Si veda la sezione 88.8.

```
#include <stdlib.h>
#include <ctype.h>
//----------------------------------------------------------
long int
atol (const char *string)
{
  int i;
  int sign = +1;
  long int number;
  //
  for (i = 0; isspace (string[i]); i++)
    {
      ;
    }
  //
  if (string[i] == '+')
    {
      sign = +1;
      i++;
    }
  else if (string[i] == '-')
```

```
    {
      sign = -1;
      i++;
    }
  //
  for (number = 0; isdigit (string[i]); i++)
    {
      number *= 10;
      number += (string[i] - '0');
    }
  //
  number *= sign;
  //
  return number;
}
```

95.19.7 lib/stdlib/div.c

Si veda la sezione 88.17.

```
#include <stdlib.h>
//----------------------------------------------------------
div_t
div (int numer, int denom)
{
  div_t d;
  d.quot = numer / denom;
  d.rem = numer % denom;
  return d;
}
```

95.19.8 lib/stdlib/environment.c

Si veda la sezione 91.1.

```
#include <stdlib.h>
#include <string.h>
//----------------------------------------------------------
// This file contains a non standard definition,
// related to the environment handling.
//
// The file `crt0.s', before calling the main function,
// calls the function `_environment_setup(), that is
// responsible for initializing the array
// `_environment_table[][]' and for copying the content
// of the environment, as it comes from the `exec()'
// system call.
//
// The pointers to the environment strings organised
// inside the array `_environment_table[][]', are also
// copied inside the array of pointers
// `_environment[]'.
//
// After all that is done, inside `crt0.s', the pointer
// to `_environment[]' is copied to the traditional
// variable `environ' and also to the previous value of
// the pointer variable `envp'.
//
// This way, applications will get the environment, but
// organised inside the table `_environment_table[][]'.
// So, functions like `getenv()' and `setenv()' do know
// where to look for.
//
// It is useful to notice that there is no prototype
// and no extern declaration inside the file
// <stdlib.h>, about this function and these arrays,
// because applications do not have to know about it.
//
// Please notice that `environ' could be just the same
// as '_environment' here, but the common use puts
// 'environ' inside <unistd.h>, although for this
// implementation it should be better placed inside
// <stdlib.h>.
//
//----------------------------------------------------------
char _environment_table[ARG_MAX / 32][ARG_MAX / 16];
char *_environment[ARG_MAX / 32 + 1];
//----------------------------------------------------------
void
_environment_setup (char *envp[])
{
  int e;
  int s;
  //
  // Reset the '_environment_table[][]' array.
  //
  for (e = 0; e < ARG_MAX / 32; e++)
    {
      for (s = 0; s < ARG_MAX / 16; s++)
        {
          _environment_table[e][s] = 0;
        }
    }
  //
  // Set the '_environment[]' pointers. The final
  // extra element must
  // be a NULL pointer.
  //
  for (e = 0; e < ARG_MAX / 32; e++)
    {
      _environment[e] = _environment_table[e];
    }
  _environment[ARG_MAX / 32] = NULL;
  //
  // Copy the environment inside the array, but only
  // if 'envp' is
  // not NULL.
  //
  if (envp != NULL)
    {
      for (e = 0; envp[e] != NULL && e < ARG_MAX / 32; e++)
        {
          strncpy (_environment_table[e], envp[e],
                   (ARG_MAX / 16) - 1);
        }
    }
}
```

95.19.9 lib/stdlib/exit.c

«

Si veda la sezione 88.7.

```
#include <stdlib.h>
#include <stdio.h>
//----------------------------------------------------------------------
extern atexit_t _atexit_table[];
//----------------------------------------------------------------------
void
exit (int status)
{
  int a;
  //
  // The "at exit" functions must be called in reverse
  // order.
  //
  for (a = (ATEXIT_MAX - 1); a >= 0; a--)
    {
      if (_atexit_table[a] != NULL)
        {
          (*_atexit_table[a]) ();
        }
    }
  //
  // Now: really exit.
  //
  _Exit (status);
  //
  // Should not return from system call, but if it
  // does, loop
  // forever:
  //
  while (1);
}
```

95.19.10 lib/stdlib/getenv.c

Si veda la sezione 88.52.

```
#include <stdlib.h>
#include <string.h>
//----------------------------------------------------------
extern char *_environment[];
//----------------------------------------------------------
char *
getenv (const char *name)
{
  int e;          // First index: environment table
  // items.
  int f;          // Second index: environment string
  // scan.
  char *value;    // Pointer to the environment value
  // found.
  //
  // Check if the input is valid. No error is
  // reported.
  //
  if (name == NULL || strlen (name) == 0)
    {
      return (NULL);
    }
    //
    // Scan the environment table items, with index 'e'.
    // The pointer
    // 'value' is initialized to NULL. If the pointer
    // 'value' gets a
    // valid pointer, the environment variable was found
    // and a
    // pointer to the beginning of its value is
    // available.
    //
    for (value = NULL, e = 0; e < ARG_MAX / 32; e++)
      {
        //
        // Scan the string of the environment item, with
        // index 'f'.
        // The scan continue until 'name[f]' and
        // '_environment[e][f]'
        // are equal.
        //
        for (f = 0;
             f < ARG_MAX / 16 - 1
             && name[f] == _environment[e][f]; f++)
          {
            ;     // Just scan.
          }
        //
        // At this point, 'name[f]' and
        // '_environment[e][f]' are
        // different: if 'name[f]' is zero the name
        // string is
        // terminated; if '_environment[e][f]' is also
        // equal to '=',
        // the environment item is corresponding to the
        // requested name.
        //
        if (name[f] == 0 && _environment[e][f] == '=')
          {
```

```
          //
          // The pointer to the beginning of the
          // environment value is
          // calculated, and the external loop exit.
          //
          value = &_environment[e][f + 1];
          break;
        }
    }
  //
  // The 'value' is returned: if it is still NULL,
  // then, no
  // environment variable with the requested name was
  // found.
  //
  return (value);
}
```

95.19.11 lib/stdlib/labs.c

Si veda la sezione 88.3.

```
#include <stdlib.h>
//----------------------------------------------------------
long int
labs (long int j)
{
  if (j < 0)
    {
      return -j;
    }
  else
    {
      return j;
    }
}
```

95.19.12 lib/stdlib/ldiv.c

Si veda la sezione 88.17.

```
#include <stdlib.h>
//----------------------------------------------------------
ldiv_t
ldiv (long int numer, long int denom)
{
  ldiv_t d;
  d.quot = numer / denom;
  d.rem = numer % denom;
  return d;
}
```

95.19.13 lib/stdlib/llabs.c

Si veda la sezione 88.3.

```
#include <stdlib.h>
//----------------------------------------------------------
long long int
llabs (long long int j)
{
  if (j < 0)
    {
      return -j;
    }
  else
    {
      return j;
    }
}
```

95.19.14 lib/stdlib/lldiv.c

Si veda la sezione 88.17.

```
#include <stdlib.h>
//----------------------------------------------------------------------
lldiv_t
lldiv (long long int numer, long long int denom)
{
  lldiv_t d;
  d.quot = numer / denom;
  d.rem = numer % denom;
  return d;
}
```

95.19.15 lib/stdlib/putenv.c

Si veda la sezione 88.94.

```
#include <stdlib.h>
#include <string.h>
#include <errno.h>
//----------------------------------------------------------------------
extern char *_environment[];
//----------------------------------------------------------------------
int
putenv (const char *string)
{
  int e;          // First index: environment table
  // items.
  int f;          // Second index: environment string
  // scan.
  //
  // Check if the input is empty. No error is
  // reported.
  //
  if (string == NULL || strlen (string) == 0)
    {
        return (0);
      }
    //
    // Check if the input is valid: there must be a '='
    // sign.
    // Error here is reported.
    //
    if (strchr (string, '=') == NULL)
      {
        errset (EINVAL);  // Invalid argument.
        return (-1);
      }
    //
    // Scan the environment table items, with index 'e'.
    // The intent is
    // to find a previous environment variable with the
    // same name.
    //
    for (e = 0; e < ARG_MAX / 32; e++)
      {
        //
        // Scan the string of the environment item, with
        // index 'f'.
        // The scan continue until 'string[f]' and
        // '_environment[e][f]'
        // are equal.
        //
        for (f = 0;
             f < ARG_MAX / 16 - 1
             && string[f] == _environment[e][f]; f++)
          {
            ;     // Just scan.
          }
        //
        // At this point, 'string[f-1]' and
        // '_environment[e][f-1]'
        // should contain '='. If it is so, the
        // environment is replaced.
        //
        if (string[f - 1] == '='
            && _environment[e][f - 1] == '=')
          {
            //
            // The environment item was found: now
            // replace the pointer.
            //
            _environment[e] = (char *) string;
            //
            // Return.
            //
            return (0);
          }
      }
    //
    // The item was not found. Scan again for a free
    // slot.
    //
    for (e = 0; e < ARG_MAX / 32; e++)
      {
        if (_environment[e] == NULL
            || _environment[e][0] == 0)
          {
            //
            // An empty item was found and the pointer
            // will be
            // replaced.
            //
            _environment[e] = (char *) string;
            //
            // Return.
            //
            return (0);
          }
      }
```

```
  //
  // Sorry: the empty slot was not found!
  //
  errset (ENOMEM);      // Not enough space.
  return (-1);
}
```

95.19.16 lib/stdlib/qsort.c

Si veda la sezione 88.96.

```
#include <stdlib.h>
#include <string.h>
#include <errno.h>
//----------------------------------------------------------
static int part (char *array, size_t size, int a,
                 int z, int (*compare) (const void *,
                                        const void *));
static void sort (char *array, size_t size, int a,
                  int z, int (*compare) (const void *,
                                         const void *));
//----------------------------------------------------------
void
qsort (void *base, size_t nmemb, size_t size,
       int (*compare) (const void *, const void *))
{
  if (size <= 1)
    {
      //
      // There is nothing to sort!
      //
      return;
    }
  else
    {
      sort ((char *) base, size, 0, (int) (nmemb - 1),
            compare);
    }
}

//----------------------------------------------------------
static void
sort (char *array, size_t size, int a, int z,
      int (*compare) (const void *, const void *))
{
  int loc;
  //
  if (z > a)
    {
      loc = part (array, size, a, z, compare);
      if (loc >= 0)
        {
          sort (array, size, a, loc - 1, compare);
          sort (array, size, loc + 1, z, compare);
        }
    }
}

//----------------------------------------------------------
static int
part (char *array, size_t size, int a, int z,
      int (*compare) (const void *, const void *))
{
  int i;
  int loc;
  char *swap;
  //
  if (z <= a)
    {
      errset (EUNKNOWN);        // Should never
                                // happen.
      return (-1);
    }
  //
```

```
    // Index 'i' after the first element; index 'loc' at
    // the last
    // position.
    //
    i = a + 1;
    loc = z;
    //
    // Prepare space in memory for element swap.
    //
    swap = malloc (size);
    if (swap == NULL)
      {
        errset (ENOMEM);
        return (-1);
      }
    //
    // Loop as long as index 'loc' is higher than index
    // 'i'.
    // When index 'loc' is less or equal to index 'i',
    // then, index 'loc' is the right position for the
    // first element of the current piece of array.
    //
    for (;;)
      {
        //
        // Index 'i' goes up...
        //
        for (; i < loc; i++)
          {
            if (compare
                (&array[i * size], &array[a * size]) > 0)
              {
                break;
              }
          }
        //
        // Index 'loc' gose down...
        //
        for (;; loc--)
          {
            if (compare
                (&array[loc * size], &array[a * size]) <= 0)
              {
                break;
              }
          }
        //
        // Swap elements related to index `i' and `loc'.
        //
        if (loc <= i)
          {
            //
            // The array is completely scanned.
            //
            break;
          }
        else
          {
            memcpy (swap, &array[loc * size], size);
            memcpy (&array[loc * size], &array[i * size],
                    size);
            memcpy (&array[i * size], swap, size);
          }
      }
    //
    // Swap the first element with the one related to
    // the
    // index `loc'.
    //
    memcpy (swap, &array[loc * size], size);
    memcpy (&array[loc * size], &array[a * size], size);
    memcpy (&array[a * size], swap, size);
    //
    // Free the swap memory.
```

```
  //
  free (swap);
  //
  // Return the index 'loc'.
  //
  return (loc);
}
```

95.19.17 lib/stdlib/rand.c

Si veda la sezione 88.97.

```
#include <stdlib.h>
//----------------------------------------------------------
static unsigned int _srand = 1; // The '_srand' rank
                                // must be at least
                                // 'unsigned int' and
                                // must be able to
                                // represent the value
                                // 'RAND_MAX'.
//----------------------------------------------------------
int
rand (void)
{
  _srand = _srand * 12345 + 123;
  return _srand % ((unsigned int) RAND_MAX + 1);
}

//----------------------------------------------------------
void
srand (unsigned int seed)
{
  _srand = seed;
}
```

95.19.18 lib/stdlib/setenv.c

«

Si veda la sezione 88.104.

```
#include <stdlib.h>
#include <string.h>
#include <errno.h>
//----------------------------------------------------------
extern char *_environment[];
extern char *_environment_table[];
//----------------------------------------------------------
int
setenv (const char *name, const char *value, int overwrite)
{
  int e;          // First index: environment table
                  // items.
  int f;          // Second index: environment string
                  // scan.
  //
  // Check if the input is empty. No error is
  // reported.
  //
  if (name == NULL || strlen (name) == 0)
    {
      return (0);
    }
  //
  // Check if the input is valid: error here is
  // reported.
  //
  if (strchr (name, '=') != NULL)
    {
      errset (EINVAL);  // Invalid argument.
      return (-1);
    }
  //
  // Check if the input is too big.
  //
```

```
    if ((strlen (name) + strlen (value) + 2) > ARG_MAX / 16)
      {
        //
        // The environment to be saved is bigger than
        // the
        // available string size, inside
        // `_environment_table[]'.
        //
        errset (ENOMEM);  // Not enough space.
        return (-1);
      }
    //
    // Scan the environment table items, with index `e'.
    // The intent is
    // to find a previous environment variable with the
    // same name.
    //
    for (e = 0; e < ARG_MAX / 32; e++)
      {
        //
        // Scan the string of the environment item, with
        // index `f'.
        // The scan continue until `name[f]' and
        // `_environment[e][f]'
        // are equal.
        //
        for (f = 0;
             f < ARG_MAX / 16 - 1
             && name[f] == _environment[e][f]; f++)
          {
            ;     // Just scan.
          }
        //
        // At this point, `name[f]' and
        // `_environment[e][f]' are
        // different: if `name[f]' is zero the name
        // string is
      // terminated; if ‘_environment[e][f]’ is also
      // equal to ‘=’,
      // the environment item is corresponding to the
      // requested name.
      //
      if (name[f] == 0 && _environment[e][f] == '=')
        {
          //
          // The environment item was found; if it can
          // be overwritten,
          // the write is done.
          //
          if (overwrite)
            {
              //
              // To be able to handle both ‘setenv()’
              // and ‘putenv()’,
              // before removing the item, it is fixed
              // the pointer to
              // the global environment table.
              //
              _environment[e] = _environment_table[e];
              //
              // Now copy the new environment. The
              // string size was
              // already checked.
              //
              strcpy (_environment[e], name);
              strcat (_environment[e], "=");
              strcat (_environment[e], value);
              //
              // Return.
              //
              return (0);
            }
          //
          // Cannot overwrite!
```

```
            //
            errset (EUNKNOWN);
            return (-1);
          }
      }
    //
    // The item was not found. Scan again for a free
    // slot.
    //
    for (e = 0; e < ARG_MAX / 32; e++)
      {
        if (_environment[e] == NULL
            || _environment[e][0] == 0)
          {
            //
            // An empty item was found. To be able to
            // handle both
            // `setenv()' and `putenv()', it is fixed
            // the pointer to
            // the global environment table.
            //
            _environment[e] = _environment_table[e];
            //
            // Now copy the new environment. The string
            // size was
            // already checked.
            //
            strcpy (_environment[e], name);
            strcat (_environment[e], "=");
            strcat (_environment[e], value);
            //
            // Return.
            //
            return (0);
          }
      }
    //
```

```
  // Sorry: the empty slot was not found!
  //
  errset (ENOMEM);      // Not enough space.
  return (-1);
}
```

95.19.19 lib/stdlib/strtol.c

Si veda la sezione 88.130.

```
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <stdbool.h>
//----------------------------------------------------------
#define isoctal(C)  ((int) (C >= '0' && C <= '7'))
//----------------------------------------------------------
long int
strtol (const char *restrict string,
        char **restrict endptr, int base)
{
  int i;
  int sign = +1;
  long int number;
  long int previous;
  int digit;
  //
  bool flag_prefix_oct = 0;
  bool flag_prefix_exa = 0;
  bool flag_prefix_dec = 0;
  //
  // Check base and string.
  //
  // With base 1 cannot do anything.
  //
  if (base < 0 || base > 36 || base == 1
        || string == NULL || string[0] == 0)
      {
        if (endptr != NULL)
          *endptr = (char *) string;
        errset (EINVAL);  // Invalid argument.
        return ((long int) 0);
      }
    //
    // Eat initial spaces.
    //
    for (i = 0; isspace (string[i]); i++)
      {
        ;
      }
    //
    // Check sign.
    //
    if (string[i] == '+')
      {
        sign = +1;
        i++;
      }
    else if (string[i] == '-')
      {
        sign = -1;
        i++;
      }
    //
    // Check for prefix.
    //
    if (string[i] == '0')
      {
        if (string[i + 1] == 'x' || string[i + 1] == 'X')
          {
            flag_prefix_exa = 1;
          }
        else if (isoctal (string[i + 1]))
```

```
          {
            flag_prefix_oct = 1;
          }
        else
          {
            flag_prefix_dec = 1;
          }
      }
    else if (isdigit (string[i]))
      {
        flag_prefix_dec = 1;
      }
    //
    // Check compatibility with requested base.
    //
    if (flag_prefix_exa)
      {
        //
        // At the moment, there is a zero and a 'x'.
        // Might be
        // exadecimal, or might be a number base 33 or
        // more.
        //
        if (base == 0)
          {
            base = 16;
          }
        else if (base == 16)
          {
            ;     // Ok.
          }
        else if (base >= 33)
          {
            ;     // Ok.
          }
        else
          {
```

```
          //
          // Incompatible sequence: only the initial
          // zero is reported.
          //
          if (endptr != NULL)
            *endptr = (char *) &string[i + 1];
          return ((long int) 0);
        }
      //
      // Move on, after the ‘0x’ prefix.
      //
      i += 2;
    }
  //
  if (flag_prefix_oct)
    {
      //
      // There is a zero and a digit.
      //
      if (base == 0)
        {
          base = 8;
        }
      //
      // Move on, after the ‘0’ prefix.
      //
      i += 1;
    }
  //
  if (flag_prefix_dec)
    {
      if (base == 0)
        {
          base = 10;
        }
    }
  //
```

```
    // Scan the string.
    //
    for (number = 0; string[i] != 0; i++)
      {
        if (string[i] >= '0' && string[i] <= '9')
          {
            digit = string[i] - '0';
          }
        else if (string[i] >= 'A' && string[i] <= 'Z')
          {
            digit = string[i] - 'A' + 10;
          }
        else if (string[i] >= 'a' && string[i] <= 'z')
          {
            digit = string[i] - 'a' + 10;
          }
        else
          {
            //
            // This is an out of range digit.
            //
            digit = 999;
          }
        //
        // Give a sign to the digit.
        //
        digit *= sign;
        //
        // Compare with the base.
        //
        if (base > (digit * sign))
          {
            //
            // Check if the current digit can be safely
            // computed.
            //
            previous = number;
          number *= base;
          number += digit;
          if (number / base != previous)
            {
              //
              // Out of range.
              //
              if (endptr != NULL)
                *endptr = (char *) &string[i + 1];
              errset (ERANGE);  // Result too large.
              if (sign > 0)
                {
                  return (LONG_MAX);
                }
              else
                {
                  return (LONG_MIN);
                }
            }
        }
      else
        {
          if (endptr != NULL)
            *endptr = (char *) &string[i];
          return (number);
        }
    }
  //
  // The string is finished.
  //
  if (endptr != NULL)
    *endptr = (char *) &string[i];
  //
  return (number);
}
```

95.19.20 lib/stdlib/strtoul.c

Si veda la sezione 88.130.

```
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
// A really poor implementation. ,-(
//
unsigned long int
strtoul (const char *restrict string,
         char **restrict endptr, int base)
{
  return ((unsigned long int)
          strtol (string, endptr, base));
}
```

95.19.21 lib/stdlib/unsetenv.c

Si veda la sezione 88.104.

```
#include <stdlib.h>
#include <string.h>
#include <errno.h>
//----------------------------------------------------------
extern char *_environment[];
extern char *_environment_table[];
//----------------------------------------------------------
int
unsetenv (const char *name)
{
  int e;        // First index: environment table
  // items.
  int f;        // Second index: environment string
  // scan.
  //
```

```
    // Check if the input is empty. No error is
    // reported.
    //
    if (name == NULL || strlen (name) == 0)
      {
        return (0);
      }
    //
    // Check if the input is valid: error here is
    // reported.
    //
    if (strchr (name, ’=’) != NULL)
      {
        errset (EINVAL);  // Invalid argument.
        return (-1);
      }
    //
    // Scan the environment table items, with index ‘e’.
    //
    for (e = 0; e < ARG_MAX / 32; e++)
      {
        //
        // Scan the string of the environment item, with
        // index ‘f’.
        // The scan continue until ‘name[f]’ and
        // ‘_environment[e][f]’
        // are equal.
        //
        for (f = 0;
             f < ARG_MAX / 16 - 1
             && name[f] == _environment[e][f]; f++)
          {
            ;     // Just scan.
          }
        //
        // At this point, ‘name[f]’ and
        // ‘_environment[e][f]’ are
      // different: if `name[f]' is zero the name
      // string is
      // terminated; if `_environment[e][f]' is also
      // equal to `=',
      // the environment item is corresponding to the
      // requested name.
      //
      if (name[f] == 0 && _environment[e][f] == '=')
        {
          //
          // The environment item was found and it
          // have to be removed.
          // To be able to handle both `setenv()' and
          // `putenv()',
          // before removing the item, it is fixed the
          // pointer to
          // the global environment table.
          //
          _environment[e] = _environment_table[e];
          //
          // Now remove the environment item.
          //
          _environment[e][0] = 0;
          break;
        }
    }
  //
  // Work done fine.
  //
  return (0);
}
```

95.19.22 lib/stdlib_alloc/_alloc_list.c

«

Si veda la sezione 88.76.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
//----------------------------------------------------------
extern uintptr_t _alloc_start;
//----------------------------------------------------------
void
_alloc_list (void)
{
  uintptr_t start = _alloc_start;
  uintptr_t end = (uintptr_t) sbrk (0);
  _alloc_head_t *head = (void *) start;
  size_t actual_size;
  uintptr_t current;
  uintptr_t next;
  uintptr_t up_to;
  int counter;
  //
  // Scandisce la lista di blocchi di memoria.
  //
  counter = 2;
  while (counter)
    {
      //
      // Annota la posizione attuale e quella
      // successiva.
      //
      current = (uintptr_t) head;
      next = head->next * (sizeof (_alloc_head_t));
      if (next == start)
        {
          up_to = end;
        }
        else
          {
            up_to = next;
          }
        //
        // Se è stato raggiunto il primo elemento,
        // decrementa il
        // contatore di una unità. Se è già a zero,
        // esce.
        //
        if (current == start)
          {
            counter--;
            if (counter == 0)
              break;
          }
        //
        // Determina la dimensione del blocco attuale.
        //
        if (current == start && next == start)
          {
            //
            // Si tratta del primo e unico elemento
            // della lista.
            //
            actual_size =
              end - start - (sizeof (_alloc_head_t));
          }
        else
          {
            actual_size =
              up_to - current - (sizeof (_alloc_head_t));
          }
        //
        // Si mostra lo stato del blocco di memoria.
        //
        if (head->allocated)
        {
          printf ("[%s] used %08X..%08X size %08zX\n",
                  __func__,
                  current + (sizeof (_alloc_head_t)),
                  up_to, actual_size);
        }
      else
        {
          printf ("[%s] free %08X..%08X size %08zX\n",
                  __func__,
                  current + (sizeof (_alloc_head_t)),
                  up_to, actual_size);
        }
      //
      // Si passa alla posizione successiva.
      //
      head = (void *) next;
    }
}
```

95.19.23 lib/stdlib_alloc/free.c

Si veda la sezione 88.76.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
//----------------------------------------------------------
extern uintptr_t _alloc_start;
//----------------------------------------------------------
void
free (void *ptr)
{
  _alloc_head_t *start = (_alloc_head_t *) _alloc_start;
  _alloc_head_t *head_current = ((_alloc_head_t *) ptr) - 1;
  _alloc_head_t *head_next;
  //
```

```
    // Verifica il blocco attuale e, se è possibile, lo
    // libera.
    //
    if (head_current->allocated == 1)
      {
        head_current->allocated = 0;
      }
    else
      {
        printf ("[%s] ERROR: cannot free %08X!\n",
                __func__,
                (uintptr_t) head_current +
                (sizeof (_alloc_head_t)));
      }
    //
    // Scandisce i blocchi liberi, cercando quelli
    // adiacenti per
    // allungarli. Se il blocco successivo è il primo,
    // termina,
    // perché non può avvenire alcuna fusione con
    // quello precedente.
    //
    head_current = start;
    while (1)
      {
        //
        // Individua il blocco successivo.
        //
        head_next =
          (_alloc_head_t *) (head_current->next
                             * (sizeof (_alloc_head_t)));
        //
        // Controlla se è il primo.
        //
        if (head_next == start)
          {
            break;
        }
      //
      //
      //
      if (head_current->allocated == 0)
        {
          //
          // Controlla se si può espandere.
          //
          if (head_next->allocated == 0)
            {
              head_current->next = head_next->next;
            }
          else
            {
              head_current = head_next;
            }
        }
      else
        {
          head_current = (_alloc_head_t *)
            (head_current->next * (sizeof (_alloc_head_t)));
        }
    }
}
```

95.19.24 lib/stdlib_alloc/malloc.c

«

Si veda la sezione 88.76.

```
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
//----------------------------------------------------------
uintptr_t _alloc_start = 0;
//----------------------------------------------------------
static int _alloc_init (void);
static void *_malloc (size_t size);
//----------------------------------------------------------
void *
malloc (size_t size)
{
  void *pstatus;
  int status;
  //
  // Verify to have initialized the allocation memory.
  //
  if (_alloc_start == 0)
    {
      status = _alloc_init ();
      if (status < 0)
        {
          errset (ENOMEM);
          return (NULL);
        }
    }
  //
  // Try to allocate as usual.
  //
  pstatus = _malloc (size);
  //
  if (pstatus == NULL)
    {
      //
      // Try to increase memory for the process.
      //
      pstatus = sbrk (size);
      if (pstatus == NULL)
        {
          //
          // Sorry: no way to get memory.
          //
          errset (ENOMEM);
          return (NULL);
        }
      //
      // Ok. Now try again to allocate memory.
      //
      return (_malloc (size));
    }
  else
    {
      //
      // The first allocation was successful.
      //
      return (pstatus);
    }
}

//----------------------------------------------------------
static int
_alloc_init (void)
{
  uintptr_t start;
  uintptr_t end;
  _alloc_head_t *head;
  size_t available;
  //
  // Get size.
  //
  if (_alloc_start == 0)
    {
      _alloc_start = (uintptr_t) sbrk (0);
    }
  //
  start = _alloc_start;
  end = (uintptr_t) sbrk (0);
  available = end - start;
  //
  // Check available space.
  //
  if (available < ((sizeof (_alloc_head_t)) * 2))
    {
      //
      // Try to get a little memory.
      //
      sbrk ((sizeof (_alloc_head_t)) * 2);
      end = (uintptr_t) sbrk (0);
      available = end - start;
      if (available < ((sizeof (_alloc_head_t)) * 2))
        {
          //
          // Sorry!
          //
          return (-1);
        }
    }
  //
  // Prepare the list main node.
  //
  head = (_alloc_head_t *) start;
  //
  // Init the first free block, that points to itself,
  // as it is
  // the only one.
  //
  head->allocated = 0;
  head->next = (start / (sizeof (_alloc_head_t)));
  //
  // Ok.
  //
  return (0);
}

//----------------------------------------------------------
static void *
_malloc (size_t size)
{
  uintptr_t start = _alloc_start;
  uintptr_t end = (uintptr_t) sbrk (0);
  _alloc_head_t *head = (void *) start;
  size_t actual_size;
  uintptr_t current;
  uintptr_t next;
  uintptr_t new;
  uintptr_t up_to;
  int counter;
  //
  // Arrotonda in eccesso il valore di «size», in
  // modo che sia un
  // multiplo della dimensione di «_alloc_head_t».
  // Altrimenti, la
  // collocazione dei blocchi successivi può avvenire
  // in modo
  // non allineato.
  //
  size = (size + (sizeof (_alloc_head_t)) - 1);
  size = size / (sizeof (_alloc_head_t));
  size = size * (sizeof (_alloc_head_t));
  //
  // Cerca un blocco libero di dimensione sufficiente.
  //
  counter = 2;
  while (counter)
    {
      //
      // Annota la posizione attuale e quella
      // successiva.
      //
      current = (uintptr_t) head;
      next = head->next * (sizeof (_alloc_head_t));
      //
      if (next == start)
        {
          up_to = end;
          }
        else
          {
            up_to = next;
          }
        //
        // Se è stato raggiunto il primo elemento,
        // decrementa il
        // contatore di una unità. Se è già a zero,
        // esce.
        //
        if (current == start)
          {
            counter--;
            if (counter == 0)
              break;
          }
        //
        // Controlla se si tratta di un blocco libero.
        //

        if (!head->allocated)
          {
            //
            // Il blocco è libero: si deve determinarne
            // la dimensione.
            //
            if (current == start && next == start)
              {
                //
                // Si tratta del primo e unico elemento
                // della lista.
                //
                actual_size =
                  end - start - (sizeof (_alloc_head_t));
              }
            else
            {
              actual_size =
                up_to - current - (sizeof (_alloc_head_t));
            }
          //
          // Si verifica che sia capiente.
          //
          if (actual_size >=
              size + ((sizeof (_alloc_head_t)) * 2))
            {
              //
              // C'è spazio per dividere il blocco.
              //
              new =
                current + size + (sizeof (_alloc_head_t));
              //
              // Aggiorna l'intestazione attuale.
              //
              head->allocated = 1;
              head->next = new / (sizeof (_alloc_head_t));
              //
              // Predispone l'intestazione successiva.
              //
              head = (void *) new;
              head->allocated = 0;
              head->next = next / (sizeof (_alloc_head_t));
              //
              // Restituisce l'indirizzo iniziale
              // dello spazio libero,
              // successivo all'intestazione.
              //
              return (void *) (current +
                               (sizeof (_alloc_head_t)));
            }
          else if (actual_size >= size)
            {
              //
              // Il blocco va usato per intero.
              //
              head->allocated = 1;
              //
              // Restituisce l'indirizzo iniziale
              // dello spazio libero,
              // successivo all'intestazione.
              //
              return (void *) (current +
                               (sizeof (_alloc_head_t)));
            }
        }
      //
      // Il blocco è allocato, oppure è di
      // dimensione insufficiente;
      // pertanto occorre passare alla posizione
      // successiva.
      //
      head = (void *) next;
    }
  //
  // Essendo terminato il ciclo precedente, vuol dire
  // che non ci sono spazi disponibili.
  //
  errset (ENOMEM);
  return NULL;
}
```

95.19.25 lib/stdlib_alloc/realloc.c

Si veda la sezione 88.76.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
//----------------------------------------------------------
```

```
extern uintptr_t _alloc_start;
//----------------------------------------------------------
void *
realloc (void *ptr, size_t size)
{
  uintptr_t start = _alloc_start;
  uintptr_t end = (uintptr_t) sbrk (0);
  size_t actual_size;
  _alloc_head_t *head = ((_alloc_head_t *) ptr) - 1;
  _alloc_head_t *head_new;
  void *ptr_new;
  //
  // Verifica che il puntatore riguardi effettivamente
  // un'area occupata.
  //
  if (!head->allocated)
    {
      printf
        ("[%s] ERROR: cannot re-allocate %08X that is "
         "not already allocated!", __func__,
         (uintptr_t) ptr);
    }
  //
  // Arrotonda in eccesso il valore di «size», in
  // modo che sia un
  // multiplo della dimensione di «_alloc_head_t».
  // Altrimenti, la
  // collocazione dei blocchi successivi può avvenire
  // in modo
  // non allineato.
  //
  size = (size + (sizeof (_alloc_head_t)) - 1);
  size = size / (sizeof (_alloc_head_t));
  size = size * (sizeof (_alloc_head_t));
  //
  // Determina la dimensione attuale.
  //
```

```
    if ((head->next * (sizeof (_alloc_head_t))) == start)
      {
        actual_size = end - ((uintptr_t) ptr);
      }
    else
      {
        actual_size =
          (head->next * (sizeof (_alloc_head_t))) -
          ((uintptr_t) ptr);
      }
    //
    // Se la dimensione richiesta è inferiore, può
    // ridurre
    // l'estensione del blocco.
    //
    if (size == actual_size)
      {
        return ptr;
      }
    else if (size <=
             (actual_size - (sizeof (_alloc_head_t)) * 2))
      {
        //
        // Si può ricavare lo spazio libero rimanente.
        //
        head_new = (_alloc_head_t *) (((char *) ptr) + size);
        //
        head_new->next = head->next;
        head_new->allocated = 0;
        //
        head->next =
          ((uintptr_t) head_new) / (sizeof (_alloc_head_t));
        //
        return ptr;
      }
    else if (size < actual_size)
      {
```

```
      //
      // Anche se è minore, non si può ridurre lo
      // spazio usato
      // effettivamente.
      //
      return ptr;
    }
  else
    {
      //
      // La dimensione richiesta è maggiore.
      //
      ptr_new = malloc (size);
      //
      if (ptr_new)
        {
          //
          // Ricopia i dati nella nuova collocazione.
          //
          memcpy (ptr_new, ptr, actual_size);
          //
          // Libera la collocazione vecchia.
          //
          free (ptr);
          //
          return ptr_new;
        }
      else
        {
          return NULL;
        }
    }
}
```

95.20 os32: «lib/string.h»

Si veda la sezione 91.3.

```
#ifndef _STRING_H
#define _STRING_H       1
//----------------------------------------------------------
#include <size_t.h>
#include <NULL.h>
#include <restrict.h>
//----------------------------------------------------------
void *memccpy (void *restrict dst,
               const void *restrict org, int c, size_t n);
void *memchr (const void *memory, int c, size_t n);
int memcmp (const void *memory1, const void *memory2,
            size_t n);
void *memcpy (void *restrict dst,
              const void *restrict org, size_t n);
void *memmove (void *dst, const void *org, size_t n);
void *memset (void *memory, int c, size_t n);
char *strcat (char *restrict dst, const char *restrict org);
char *strchr (const char *string, int c);
int strcmp (const char *string1, const char *string2);
int strcoll (const char *string1, const char *string2);
char *strcpy (char *restrict dst, const char *restrict org);
size_t strcspn (const char *string, const char *reject);
char *strdup (const char *string);
char *strerror (int errnum);
size_t strlen (const char *string);
char *strncat (char *restrict dst,
               const char *restrict org, size_t n);
int strncmp (const char *string1, const char *string2,
             size_t n);
char *strncpy (char *restrict dst,
               const char *restrict org, size_t n);
char *strpbrk (const char *string, const char *accept);
char *strrchr (const char *string, int c);
size_t strspn (const char *string, const char *accept);
```

```
char *strstr (const char *string, const char *substring);
char *strtok (char *restrict string,
              const char *restrict delim);
size_t strxfrm (char *restrict dst,
                const char *restrict org, size_t n);
//----------------------------------------------------------

#endif
```

95.20.1 lib/string/memccpy.c

Si veda la sezione 88.77.

```
#include <string.h>
//----------------------------------------------------------
void *
memccpy (void *restrict dst, const void *restrict org,
         int c, size_t n)
{
  char *d = (char *) dst;
  char *o = (char *) org;
  size_t i;
  for (i = 0; n > 0 && i < n; i++)
    {
      d[i] = o[i];
      if (d[i] == (char) c)
        {
          return ((void *) &d[i + 1]);
        }
    }
  return (NULL);
}
```

95.20.2 lib/string/memchr.c

Si veda la sezione 88.78.

```
#include <string.h>
//----------------------------------------------------------
void *
memchr (const void *memory, int c, size_t n)
{
  char *m = (char *) memory;
  size_t i;
  for (i = 0; n > 0 && i < n; i++)
    {
      if (m[i] == (char) c)
        {
          return (void *) (m + i);
        }
    }
  return NULL;
}
```

95.20.3 lib/string/memcmp.c

Si veda la sezione 88.79.

```
#include <string.h>
//----------------------------------------------------------
int
memcmp (const void *memory1, const void *memory2, size_t n)
{
  char *a = (char *) memory1;
  char *b = (char *) memory2;
  size_t i;
  for (i = 0; n > 0 && i < n; i++)
    {
      if (a[i] > b[i])
        {
          return 1;
        }
      else if (a[i] < b[i])
        {
          return -1;
        }
    }
  return 0;
}
```

95.20.4 lib/string/memcpy.c

Si veda la sezione 88.80.

```
#include <string.h>
//----------------------------------------------------------
void *
memcpy (void *restrict dst, const void *restrict org,
        size_t n)
{
  char *d = (char *) dst;
  char *o = (char *) org;
  size_t i;
  for (i = 0; n > 0 && i < n; i++)
    {
      d[i] = o[i];
    }
  return dst;
}
```

95.20.5 lib/string/memmove.c

Si veda la sezione 88.81.

```
#include <string.h>
//----------------------------------------------------------
void *
memmove (void *dst, const void *org, size_t n)
```

```
{
  char *d = (char *) dst;
  char *o = (char *) org;
  size_t i;
  //
  // Depending on the memory start locations, copy may
  // be direct or
  // reverse, to avoid overwriting before the
  // relocation is done.
  //
  if (d < o)
    {
      for (i = 0; i < n; i++)
        {
          d[i] = o[i];
        }
    }
  else if (d == o)
    {
      //
      // Memory locations are already the same.
      //
      ;
    }
  else
    {
      for (i = n - 1; i >= 0; i--)
        {
          d[i] = o[i];
        }
    }
  return dst;
}
```

95.20.6 lib/string/memset.c

Si veda la sezione 88.82.

```
#include <string.h>
//----------------------------------------------------------
void *
memset (void *memory, int c, size_t n)
{
    char *m = (char *) memory;
    size_t i;
    for (i = 0; n > 0 && i < n; i++)
      {
        m[i] = (char) c;
      }
    return memory;
}
```

95.20.7 lib/string/strcat.c

Si veda la sezione 88.113.

```
#include <string.h>
//----------------------------------------------------------
char *
strcat (char *restrict dst, const char *restrict org)
{
    size_t i;
    size_t j;
    for (i = 0; dst[i] != 0; i++)
      {
        ; // Just look for the null character.
      }
    for (j = 0; org[j] != 0; i++, j++)
      {
        dst[i] = org[j];
      }
    dst[i] = 0;
```

```
  return dst;
}
```

95.20.8 lib/string/strchr.c

Si veda la sezione 88.114.

```
#include <string.h>
//----------------------------------------------------------
char *
strchr (const char *string, int c)
{
  size_t i;
  for (i = 0;; i++)
    {
      if (string[i] == (char) c)
        {
          return (char *) (string + i);
        }
      else if (string[i] == 0)
        {
          return NULL;
        }
    }
}
```

95.20.9 lib/string/strcmp.c

Si veda la sezione 88.115.

```
#include <string.h>
//----------------------------------------------------------
int
strcmp (const char *string1, const char *string2)
{
  char *a = (char *) string1;
  char *b = (char *) string2;
  size_t i;
  for (i = 0;; i++)
    {
      if (a[i] > b[i])
        {
          return 1;
        }
      else if (a[i] < b[i])
        {
          return -1;
        }
      else if (a[i] == 0 && b[i] == 0)
        {
          return 0;
        }
    }
}
```

95.20.10 lib/string/strcoll.c

Si veda la sezione 88.115.

```
#include <string.h>
//----------------------------------------------------------
int
strcoll (const char *string1, const char *string2)
{
  return (strcmp (string1, string2));
}
```

95.20.11 lib/string/strcpy.c

Si veda la sezione 88.117.

```
#include <string.h>
//----------------------------------------------------------
char *
```

```
strcpy (char *restrict dst, const char *restrict org)
{
  size_t i;
  for (i = 0; org[i] != 0; i++)
    {
      dst[i] = org[i];
    }
  dst[i] = 0;
  return dst;
}
```

95.20.12 lib/string/strcspn.c

Si veda la sezione 88.127.

```
#include <string.h>
//----------------------------------------------------------
size_t
strcspn (const char *string, const char *reject)
{
  size_t i;
  size_t j;
  int found;
  for (i = 0; string[i] != 0; i++)
    {
      for (j = 0, found = 0; reject[j] != 0 || found; j++)
        {
          if (string[i] == reject[j])
            {
              found = 1;
              break;
            }
        }
      if (found)
        {
          break;
        }
    }
  return i;
}
```

95.20.13 lib/string/strdup.c

Si veda la sezione 88.119.

```
#include <string.h>
#include <stdlib.h>
#include <errno.h>
//----------------------------------------------------------
char *
strdup (const char *string)
{
  size_t size;
  char *copy;
  //
  // Get string size: must be added 1, to count the
  // termination null
  // character.
  //
  size = strlen (string) + 1;
  //
  copy = malloc (size);
  //
  if (copy == NULL)
    {
      errset (ENOMEM);  // Not enough memory.
      return (NULL);
    }
  //
  strcpy (copy, string);
  //
  return (copy);
}
```

95.20.14 lib/string/strerror.c

Si veda la sezione 88.120.

```
#include <string.h>
#include <errno.h>
//----------------------------------------------------------
#define ERROR_MAX 120
//----------------------------------------------------------
char *
strerror (int errnum)
{
  static char *err[ERROR_MAX];
  //
  err[0] = "No error";
  err[E2BIG] = TEXT_E2BIG;
  err[EACCES] = TEXT_EACCES;
  err[EADDRINUSE] = TEXT_EADDRINUSE;
  err[EADDRNOTAVAIL] = TEXT_EADDRNOTAVAIL;
  err[EAFNOSUPPORT] = TEXT_EAFNOSUPPORT;
  err[EAGAIN] = TEXT_EAGAIN;
  err[EALREADY] = TEXT_EALREADY;
  err[EBADF] = TEXT_EBADF;
  err[EBADMSG] = TEXT_EBADMSG;
  err[EBUSY] = TEXT_EBUSY;
  err[ECANCELED] = TEXT_ECANCELED;
  err[ECHILD] = TEXT_ECHILD;
  err[ECONNABORTED] = TEXT_ECONNABORTED;
  err[ECONNREFUSED] = TEXT_ECONNREFUSED;
  err[ECONNRESET] = TEXT_ECONNRESET;
  err[EDEADLK] = TEXT_EDEADLK;
  err[EDESTADDRREQ] = TEXT_EDESTADDRREQ;
  err[EDOM] = TEXT_EDOM;
  err[EDQUOT] = TEXT_EDQUOT;
  err[EEXIST] = TEXT_EEXIST;
  err[EFAULT] = TEXT_EFAULT;
  err[EFBIG] = TEXT_EFBIG;
  err[EHOSTUNREACH] = TEXT_EHOSTUNREACH;
```

```
    err[EIDRM] = TEXT_EIDRM;
    err[EILSEQ] = TEXT_EILSEQ;
    err[EINPROGRESS] = TEXT_EINPROGRESS;
    err[EINTR] = TEXT_EINTR;
    err[EINVAL] = TEXT_EINVAL;
    err[EIO] = TEXT_EIO;
    err[EISCONN] = TEXT_EISCONN;
    err[EISDIR] = TEXT_EISDIR;
    err[ELOOP] = TEXT_ELOOP;
    err[EMFILE] = TEXT_EMFILE;
    err[EMLINK] = TEXT_EMLINK;
    err[EMSGSIZE] = TEXT_EMSGSIZE;
    err[EMULTIHOP] = TEXT_EMULTIHOP;
    err[ENAMETOOLONG] = TEXT_ENAMETOOLONG;
    err[ENETDOWN] = TEXT_ENETDOWN;
    err[ENETRESET] = TEXT_ENETRESET;
    err[ENETUNREACH] = TEXT_ENETUNREACH;
    err[ENFILE] = TEXT_ENFILE;
    err[ENOBUFS] = TEXT_ENOBUFS;
    err[ENODATA] = TEXT_ENODATA;
    err[ENODEV] = TEXT_ENODEV;
    err[ENOENT] = TEXT_ENOENT;
    err[ENOEXEC] = TEXT_ENOEXEC;
    err[ENOLCK] = TEXT_ENOLCK;
    err[ENOLINK] = TEXT_ENOLINK;
    err[ENOMEM] = TEXT_ENOMEM;
    err[ENOMSG] = TEXT_ENOMSG;
    err[ENOPROTOOPT] = TEXT_ENOPROTOOPT;
    err[ENOSPC] = TEXT_ENOSPC;
    err[ENOSR] = TEXT_ENOSR;
    err[ENOSTR] = TEXT_ENOSTR;
    err[ENOSYS] = TEXT_ENOSYS;
    err[ENOTCONN] = TEXT_ENOTCONN;
    err[ENOTDIR] = TEXT_ENOTDIR;
    err[ENOTEMPTY] = TEXT_ENOTEMPTY;
    err[ENOTSOCK] = TEXT_ENOTSOCK;
    err[ENOTSUP] = TEXT_ENOTSUP;
    err[ENOTTY] = TEXT_ENOTTY;
    err[ENXIO] = TEXT_ENXIO;
    err[EOPNOTSUPP] = TEXT_EOPNOTSUPP;
    err[EOVERFLOW] = TEXT_EOVERFLOW;
    err[EPERM] = TEXT_EPERM;
    err[EPIPE] = TEXT_EPIPE;
    err[EPROTO] = TEXT_EPROTO;
    err[EPROTONOSUPPORT] = TEXT_EPROTONOSUPPORT;
    err[EPROTOTYPE] = TEXT_EPROTOTYPE;
    err[ERANGE] = TEXT_ERANGE;
    err[EROFS] = TEXT_EROFS;
    err[ESPIPE] = TEXT_ESPIPE;
    err[ESRCH] = TEXT_ESRCH;
    err[ESTALE] = TEXT_ESTALE;
    err[ETIME] = TEXT_ETIME;
    err[ETIMEDOUT] = TEXT_ETIMEDOUT;
    err[ETXTBSY] = TEXT_ETXTBSY;
    err[EWOULDBLOCK] = TEXT_EWOULDBLOCK;
    err[EXDEV] = TEXT_EXDEV;
    err[E_NO_MEDIUM] = TEXT_E_NO_MEDIUM;
    err[E_MEDIUM] = TEXT_E_MEDIUM;
    err[E_FILE_TYPE] = TEXT_E_FILE_TYPE;
    err[E_ROOT_INODE_NOT_CACHED] =
      TEXT_E_ROOT_INODE_NOT_CACHED;
    err[E_CANNOT_READ_SUPERBLOCK] =
      TEXT_E_CANNOT_READ_SUPERBLOCK;
    err[E_MAP_INODE_TOO_BIG] = TEXT_E_MAP_INODE_TOO_BIG;
    err[E_MAP_ZONE_TOO_BIG] = TEXT_E_MAP_ZONE_TOO_BIG;
    err[E_DATA_ZONE_TOO_BIG] = TEXT_E_DATA_ZONE_TOO_BIG;
    err[E_CANNOT_FIND_ROOT_DEVICE] =
      TEXT_E_CANNOT_FIND_ROOT_DEVICE;
    err[E_CANNOT_FIND_ROOT_INODE] =
      TEXT_E_CANNOT_FIND_ROOT_INODE;
    err[E_FILE_TYPE_UNSUPPORTED] =
      TEXT_E_FILE_TYPE_UNSUPPORTED;
    err[E_ENV_TOO_BIG] = TEXT_E_ENV_TOO_BIG;
    err[E_LIMIT] = TEXT_E_LIMIT;
```

```
  err[E_NOT_MOUNTED] = TEXT_E_NOT_MOUNTED;
  err[E_NOT_IMPLEMENTED] = TEXT_E_NOT_IMPLEMENTED;
  err[E_HARDWARE_FAULT] = TEXT_E_HARDWARE_FAULT;
  err[E_DRIVER_FAULT] = TEXT_E_DRIVER_FAULT;
  err[E_PIPE_FULL] = TEXT_E_PIPE_FULL;
  err[E_PIPE_EMPTY] = TEXT_E_PIPE_EMPTY;
  err[E_PART_TYPE_NOT_MINIX] = TEXT_E_PART_TYPE_NOT_MINIX;
  err[E_FS_TYPE_NOT_SUPPORTED] =
    TEXT_E_FS_TYPE_NOT_SUPPORTED;
  err[E_PDU_TOO_BIG] = TEXT_E_PDU_TOO_BIG;
  err[E_ARP_MISSING] = TEXT_E_ARP_MISSING;
  //
  if (errnum >= ERROR_MAX || errnum < 0)
    {
      return ("Unknown error");
    }
  //
  return (err[errnum]);
}
```

95.20.15 lib/string/strlen.c

Si veda la sezione 88.121.

```
#include <string.h>
//----------------------------------------------------------------------
size_t
strlen (const char *string)
{
  size_t i;
  for (i = 0; string[i] != 0; i++)
    {
      ; // Just count.
    }
  return i;
}
```

95.20.16 lib/string/strncat.c

Si veda la sezione 88.113.

```
#include <string.h>
//----------------------------------------------------------
char *
strncat (char *restrict dst, const char *restrict org,
         size_t n)
{
  size_t i;
  size_t j;
  for (i = 0; n > 0 && dst[i] != 0; i++)
    {
      ; // Just seek the null character.
    }
  for (j = 0; n > 0 && j < n && org[j] != 0; i++, j++)
    {
      dst[i] = org[j];
    }
  dst[i] = 0;
  return dst;
}
```

95.20.17 lib/string/strncmp.c

Si veda la sezione 88.115.

```
#include <string.h>
//----------------------------------------------------------
int
strncmp (const char *string1, const char *string2, size_t n)
{
  size_t i;
  for (i = 0; i < n; i++)
    {
      if (string1[i] > string2[i])
        {
          return 1;
        }
      else if (string1[i] < string2[i])
        {
          return -1;
        }
      else if (string1[i] == 0 && string2[i] == 0)
        {
          return 0;
        }
    }
  return 0;
}
```

95.20.18 lib/string/strncpy.c

Si veda la sezione 88.117.

```
#include <string.h>
//----------------------------------------------------------------------
char *
strncpy (char *restrict dst, const char *restrict org,
         size_t n)
{
  size_t i;
  for (i = 0; n > 0 && i < n && org[i] != 0; i++)
    {
      dst[i] = org[i];
    }
  for (; n > 0 && i < n; i++)
    {
      dst[i] = 0;
    }
  return dst;
}
```

95.20.19 lib/string/strpbrk.c

Si veda la sezione 88.125.

```
#include <string.h>
//----------------------------------------------------------
char *
strpbrk (const char *string, const char *accept)
{
  //
  // The first parameter not `const char *' because
  // otherwise
  // the return value should be `const char *' too!
  //
  size_t i;
  size_t j;
  //
  for (i = 0; string[i] != 0; i++)
    {
      for (j = 0; accept[j] != 0; j++)
        {
          if (string[i] == accept[j])
            {
              return (char *) (string + i);
            }
        }
    }
  return NULL;
}
```

95.20.20 lib/string/strrchr.c

Si veda la sezione 88.114.

```
#include <string.h>
//----------------------------------------------------------
char *
strrchr (const char *string, int c)
```

```
{
  int i;
  for (i = strlen (string); i >= 0; i--)
    {
      if (string[i] == (char) c)
        {
          break;
        }
    }
  if (i < 0)
    {
      return NULL;
    }
  else
    {
      return (char *) (string + i);
    }
}
```

95.20.21 lib/string/strspn.c

Si veda la sezione 88.127.

```
#include <string.h>
//----------------------------------------------------------
size_t
strspn (const char *string, const char *accept)
{
  size_t i;
  size_t j;
  int found;
  for (i = 0; string[i] != 0; i++)
    {
      for (j = 0, found = 0; accept[j] != 0; j++)
        {
          if (string[i] == accept[j])
            {
              found = 1;
              break;
            }
        }
      if (!found)
        {
          break;
        }
    }
  return i;
}
```

95.20.22 lib/string/strstr.c

Si veda la sezione 88.128.

```
#include <string.h>
//----------------------------------------------------------
char *
strstr (const char *string, const char *substring)
{
  size_t i;
  size_t j;
  size_t k;
  int found;
  if (substring[0] == 0)
    {
      return (char *) string;
    }
  for (i = 0, j = 0, found = 0; string[i] != 0; i++)
    {
      if (string[i] == substring[0])
        {
          for (k = i, j = 0;
               string[k] == substring[j] &&
               string[k] != 0 &&
               substring[j] != 0; j++, k++)
```

```
            {
              ;
            }
          if (substring[j] == 0)
            {
              found = 1;
            }
        }
      if (found)
        {
          return (char *) (string + i);
        }
    }
  return NULL;
}
```

95.20.23 lib/string/strtok.c

Si veda la sezione 88.129.

```
#include <string.h>
//----------------------------------------------------------
char *
strtok (char *restrict string, const char *restrict delim)
{
  static char *next = NULL;
  size_t i = 0;
  size_t j;
  int found_token;
  int found_delim;
  //
  // If the string received a the first parameter is a
  // null pointer,
  // the static pointer is used. But if it is already
  // NULL,
  // the scan cannot start.
  //
```

```
    if (string == NULL)
      {
        if (next == NULL)
          {
            return NULL;
          }
        else
          {
            string = next;
          }
      }
    //
    // If the string received as the first parameter is
    // empty, the scan
    // cannot start.
    //
    if (string[0] == 0)
      {
        next = NULL;
        return NULL;
      }
    else
      {
        if (delim[0] == 0)
          {
            return string;
          }
      }
    //
    // Find the next token.
    //
    for (i = 0, found_token = 0, j = 0;
         string[i] != 0 && (!found_token); i++)
      {
        //
        // Look inside delimiters.
        //
        for (j = 0, found_delim = 0; delim[j] != 0; j++)
          {
            if (string[i] == delim[j])
              {
                found_delim = 1;
              }
          }
        //
        // If current character inside the string is not
        // a delimiter,
        // it is the start of a new token.
        //
        if (!found_delim)
          {
            found_token = 1;
            break;
          }
      }
    //
    // If a token was found, the pointer is updated.
    // If otherwise the token is not found, this means
    // that
    // there are no more.
    //
    if (found_token)
      {
        string += i;
      }
    else
      {
        next = NULL;
        return NULL;
      }
    //
    // Find the end of the token.
    //
    for (i = 0, found_delim = 0; string[i] != 0; i++)
    {
      for (j = 0; delim[j] != 0; j++)
        {
          if (string[i] == delim[j])
            {
              found_delim = 1;
              break;
            }
        }
      if (found_delim)
        {
          break;
        }
    }
  //
  // If a delimiter was found, the corresponding
  // character must be
  // reset to zero. If otherwise the string is
  // terminated, the
  // scan is terminated.
  //
  if (found_delim)
    {
      string[i] = 0;
      next = &string[i + 1];
    }
  else
    {
      next = NULL;
    }
  //
  // At this point, the current string represent the
  // token found.
  //
  return string;
}
```

95.20.24 lib/string/strxfrm.c

Si veda la sezione 88.132.

```
#include <string.h>
//----------------------------------------------------------
size_t
strxfrm (char *restrict dst, const char *restrict org,
         size_t n)
{
  size_t i;
  if (n == 0 && dst == NULL)
    {
      return strlen (org);
    }
  else
    {
      for (i = 0; i < n; i++)
        {
          dst[i] = org[i];
          if (org[i] == 0)
            {
              break;
            }
        }
      return i;
    }
}
```

95.21 os32: «lib/sys/os32.h»

Si veda la sezione 91.3.

```
#ifndef _SYS_OS32_H
#define _SYS_OS32_H      1
//----------------------------------------------------------
// This file contains all the declarations that don't
// have a better place inside standard headers files.
// Even declarations related to device numbers and
// system calls is contained here.
//----------------------------------------------------------
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdint.h>
#include <signal.h>
#include <limits.h>
#include <stdio.h>
#include <stddef.h>
#include <restrict.h>
#include <stdarg.h>
#include <termios.h>
//----------------------------------------------------------
typedef uint16_t h_port_t;      // Port number in host
                                // byte order.
typedef uint32_t h_addr_t;      // IPv4 address in
                                // host byte order.
//----------------------------------------------------------
// Please remember that system calls should never be
// used (called) inside the kernel code, because system
// calls cannot be nested for the os32 simple
// architecture!
// If a particular function is necessary inside the
// kernel, that usually is made by a system call, an
// appropriate k_...() function must be
// made, to avoid the problem.
//----------------------------------------------------------
// Device numbers.
//----------------------------------------------------------
#define DEV_UNDEFINED_MAJOR     ((dev_t) 0x00)
#define DEV_UNDEFINED           ((dev_t) 0x0000)
#define DEV_MEM_MAJOR           ((dev_t) 0x01)
#define DEV_MEM                 ((dev_t) 0x0101)
```

```
#define DEV_NULL                ((dev_t) 0x0102)
#define DEV_PORT                ((dev_t) 0x0103)
#define DEV_ZERO                ((dev_t) 0x0104)
#define DEV_TTY_MAJOR           ((dev_t) 0x02)
#define DEV_TTY                 ((dev_t) 0x0200)
//
#define DEV_KMEM_MAJOR          ((dev_t) 0x04)
#define DEV_KMEM_PS             ((dev_t) 0x0401)
#define DEV_KMEM_MMP            ((dev_t) 0x0402)
#define DEV_KMEM_SB             ((dev_t) 0x0403)
#define DEV_KMEM_INODE          ((dev_t) 0x0404)
#define DEV_KMEM_FILE           ((dev_t) 0x0405)
#define DEV_KMEM_ARP            ((dev_t) 0x0406)
#define DEV_KMEM_NET            ((dev_t) 0x0407)
#define DEV_KMEM_ROUTE          ((dev_t) 0x0408)
//
#define DEV_CONSOLE_MAJOR       ((dev_t) 0x05)
#define DEV_CONSOLE             ((dev_t) 0x05FF)
#define DEV_CONSOLE0            ((dev_t) 0x0500)
#define DEV_CONSOLE1            ((dev_t) 0x0501)
#define DEV_CONSOLE2            ((dev_t) 0x0502)
#define DEV_CONSOLE3            ((dev_t) 0x0503)
#define DEV_CONSOLE4            ((dev_t) 0x0504)
//
#define DEV_DM_MAJOR            ((dev_t) 0x08)
#define DEV_DM00                ((dev_t) 0x0800)
#define DEV_DM01                ((dev_t) 0x0801)
#define DEV_DM02                ((dev_t) 0x0802)
#define DEV_DM03                ((dev_t) 0x0803)
#define DEV_DM04                ((dev_t) 0x0804)
#define DEV_DM10                ((dev_t) 0x0810)
#define DEV_DM11                ((dev_t) 0x0811)
#define DEV_DM12                ((dev_t) 0x0812)
#define DEV_DM13                ((dev_t) 0x0813)
#define DEV_DM14                ((dev_t) 0x0814)
#define DEV_DM20                ((dev_t) 0x0820)
#define DEV_DM21                ((dev_t) 0x0821)
```

```
#define DEV_DM22                ((dev_t) 0x0822)
#define DEV_DM23                ((dev_t) 0x0823)
#define DEV_DM24                ((dev_t) 0x0824)
#define DEV_DM30                ((dev_t) 0x0830)
#define DEV_DM31                ((dev_t) 0x0831)
#define DEV_DM32                ((dev_t) 0x0832)
#define DEV_DM33                ((dev_t) 0x0833)
#define DEV_DM34                ((dev_t) 0x0834)
//
//----------------------------------------------------------
#define min(a, b) (a < b ? a : b)
#define max(a, b) (a > b ? a : b)
#define sizeof_array(x) (sizeof(x) / sizeof((x)[0]))
#define sizeof_field(t, f) (sizeof(((t*)0)->f))
//----------------------------------------------------------
#define INPUT_LINE_HIDDEN 0
#define INPUT_LINE_ECHO   1
//----------------------------------------------------------
#define MOUNT_DEFAULT     0     // Default mount
                                // options.
#define MOUNT_RO          1     // Read only mount
                                // option.
//----------------------------------------------------------
#define SYS_0                    0      // Nothing to
                                        // do.
#define SYS_CHDIR                1
#define SYS_CHMOD                2
#define SYS_CLOCK                3
#define SYS_CLOSE                4
#define SYS_EXEC                 5
#define SYS_EXIT                 6      // [1] see
                                        // below.
#define SYS_FCHMOD               7
#define SYS_FORK                 8
#define SYS_FSTAT                9
#define SYS_KILL                10
#define SYS_LSEEK               11
```

```
#define SYS_MKDIR               12
#define SYS_MKNOD               13
#define SYS_MOUNT               14
#define SYS_OPEN                15
#define SYS_PGRP                16
#define SYS_READ                17
#define SYS_SETEUID             18
#define SYS_SETUID              19
#define SYS_SIGNAL              20
#define SYS_SLEEP               21
#define SYS_STAT                22
#define SYS_TIME                23
#define SYS_UAREA               24
#define SYS_UMASK               25
#define SYS_UMOUNT              26
#define SYS_WAIT                27
#define SYS_WRITE               28
#define SYS_ZPCHAR              29      // [2]
#define SYS_ZPSTRING            30      // [2]
#define SYS_CHOWN               31
#define SYS_DUP                 33
#define SYS_DUP2                34
#define SYS_LINK                35
#define SYS_UNLINK              36
#define SYS_FCNTL               37
#define SYS_STIME               38
#define SYS_FCHOWN              39
#define SYS_BRK                 40
#define SYS_SBRK                41
#define SYS_PIPE                42
#define SYS_TCGETATTR           43
#define SYS_TCSETATTR           44
#define SYS_SETEGID             45
#define SYS_SETGID              46
#define SYS_SETJMP              47
#define SYS_LONGJMP             48
#define SYS_RECVFROM            49
```

```
#define SYS_SOCKET              50
#define SYS_CONNECT             51
#define SYS_SEND                52
#define SYS_IPCONFIG            53
#define SYS_ROUTEADD            54
#define SYS_ROUTEDEL            55
#define SYS_BIND                56
#define SYS_LISTEN              57
#define SYS_ACCEPT              58
//
// [1] The files ‘crt0...’ need to know the value used
//     for the exit system call. If this value is
//     modified, all the file ‘crt0...’ have also to be
//     modified the same way.
//
// [2] These system calls were developed at the
//     beginning, when no standard I/O was available.
//     They are to be considered as a last resort for
//     debugging purposes.
//
//----------------------------------------------------------
// The following values must be: 1, 2, 4, 8, 16, 32,...
// so that can be ‘OR’ combined.
//
#define WAKEUP_EVENT_SIGNAL       0x0001
#define WAKEUP_EVENT_TIMER        0x0002
#define WAKEUP_EVENT_DEV_READ     0x0004
#define WAKEUP_EVENT_DEV_WRITE    0x0008
#define WAKEUP_EVENT_PIPE_READ    0x0010
#define WAKEUP_EVENT_PIPE_WRITE   0x0020
#define WAKEUP_EVENT_SOCK_READ    0x0040
#define WAKEUP_EVENT_SOCK_WRITE   0x0080
//----------------------------------------------------------
typedef struct
{
  int sfdn;
  struct sockaddr addr;
  socklen_t addrlen;
  int fl_flags;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_accept_t;
//-----------------------------------------------------------
typedef struct
{
  int sfdn;
  struct sockaddr addr;
  socklen_t addrlen;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_bind_t;
//-----------------------------------------------------------
typedef struct
{
  void *address;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_brk_t;
//-----------------------------------------------------------
typedef struct
{
  const char *path;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_chdir_t;
//-----------------------------------------------------------
```

```
typedef struct
{
  const char *path;
  mode_t mode;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_chmod_t;
//-----------------------------------------------------------
typedef struct
{
  const char *path;
  uid_t uid;
  uid_t gid;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_chown_t;
//-----------------------------------------------------------
typedef struct
{
  clock_t ret;
} sysmsg_clock_t;
//-----------------------------------------------------------
typedef struct
{
  int fdn;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_close_t;
//-----------------------------------------------------------
typedef struct
{
  int sfdn;
  struct sockaddr addr;
  socklen_t addrlen;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_connect_t;
//----------------------------------------------------------
typedef struct
{
  int fdn_old;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_dup_t;
//----------------------------------------------------------
typedef struct
{
  int fdn_old;
  int fdn_new;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_dup2_t;
//----------------------------------------------------------
typedef struct
{
  const char *path;
  int argc;
  int envc;
  char arg_data[ARG_MAX / 2];
  char env_data[ARG_MAX / 2];
  uid_t uid;
  uid_t euid;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_exec_t;
//----------------------------------------------------------
typedef struct
{
  int status;
} sysmsg_exit_t;
//----------------------------------------------------------
typedef struct
{
  int fdn;
  mode_t mode;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_fchmod_t;
//----------------------------------------------------------
typedef struct
{
  int fdn;
  uid_t uid;
  uid_t gid;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_fchown_t;
//----------------------------------------------------------
typedef struct
{
  int fdn;
  int cmd;
  int arg;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_fcntl_t;
//-----------------------------------------------------------
typedef struct
{
  pid_t ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_fork_t;
//-----------------------------------------------------------
typedef struct
{
  int fdn;
  struct stat stat;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_fstat_t;
//-----------------------------------------------------------
typedef struct
{
  int n;
  in_addr_t address;
  int m;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_ipconfig_t;
//-----------------------------------------------------------
typedef struct
{
  void *env;
  int ret;
  //
  // This structure is intentionally reduced.
  //
} sysmsg_jmp_t;
//----------------------------------------------------------
typedef struct
{
  pid_t pid;
  int signal;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_kill_t;
//----------------------------------------------------------
typedef struct
{
  const char *path_old;
  const char *path_new;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_link_t;
//----------------------------------------------------------
typedef struct
{
  int sfdn;
  int backlog;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_listen_t;
//----------------------------------------------------------
```

```
typedef struct
{
  int fdn;
  off_t offset;
  int whence;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_lseek_t;
//-----------------------------------------------------------
typedef struct
{
  const char *path;
  mode_t mode;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_mkdir_t;
//-----------------------------------------------------------
typedef struct
{
  const char *path;
  mode_t mode;
  dev_t device;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_mknod_t;
//-----------------------------------------------------------
typedef struct
{
  const char *path_dev;
  const char *path_mnt;
  int options;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_mount_t;
//----------------------------------------------------------
typedef struct
{
  const char *path;
  int flags;
  mode_t mode;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_open_t;
//----------------------------------------------------------
typedef struct
{
  int pipefd[2];
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_pipe_t;
//----------------------------------------------------------
typedef struct
{
  int fdn;
  void *buffer;
  size_t count;
  int fl_flags;
  ssize_t ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_read_t;
```

```
//-----------------------------------------------------------
typedef struct
{
  int sfdn;
  void *buffer;
  size_t count;
  int flags;
  void *addrfrom;
  void *addrsize;
  int fl_flags;
  ssize_t ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_recvfrom_t;
//-----------------------------------------------------------
typedef struct
{
  in_addr_t destination;
  int m;
  in_addr_t router;
  int device;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_route_t;
//-----------------------------------------------------------
typedef struct
{
  intptr_t increment;
  void *ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_sbrk_t;
//-----------------------------------------------------------
```

```
typedef struct
{
  int sfdn;
  const void *buffer;
  size_t count;
  int flags;
  ssize_t ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_send_t;
//----------------------------------------------------------
typedef struct
{
  gid_t egid;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_setegid_t;
//----------------------------------------------------------
typedef struct
{
  uid_t euid;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_seteuid_t;
//----------------------------------------------------------
typedef struct
{
  gid_t gid;
  gid_t egid;
  gid_t sgid;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_setgid_t;
//----------------------------------------------------------
typedef struct
{
  uid_t uid;
  uid_t euid;
  uid_t suid;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_setuid_t;
//----------------------------------------------------------
typedef struct
{
  uintptr_t wrapper;
  sighandler_t handler;
  int signal;
  sighandler_t ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_signal_t;
//----------------------------------------------------------
typedef struct
{
  int family;
  int type;
  int protocol;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_socket_t;
//----------------------------------------------------------
```

```
typedef struct
{
  int events;
  int signal;
  unsigned int seconds;
  time_t ret;
} sysmsg_sleep_t;
//----------------------------------------------------------
typedef struct
{
  const char *path;
  struct stat stat;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_stat_t;
//----------------------------------------------------------
typedef struct
{
  time_t ret;
} sysmsg_time_t;
//----------------------------------------------------------
typedef struct
{
  time_t timer;
  int ret;
} sysmsg_stime_t;
//----------------------------------------------------------
typedef struct
{
  int fdn;
  int action;
  struct termios *attr;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_tcattr_t;
//----------------------------------------------------------
typedef struct
{
  uid_t uid;    // Read user ID.
  uid_t euid;   // Effective user ID.
  uid_t suid;   // Saved user ID.
  gid_t gid;    // Read group ID.
  gid_t egid;   // Effective group ID.
  gid_t sgid;   // Saved group ID.
  pid_t pid;    // Process ID.
  pid_t ppid;   // Parent PID.
  pid_t pgrp;   // Process group.
  mode_t umask; // Access permission mask.
  char *path_cwd;
  size_t path_cwd_size; // Max path size.
} sysmsg_uarea_t;
//----------------------------------------------------------
typedef struct
{
  mode_t umask;
  mode_t ret;
} sysmsg_umask_t;
//----------------------------------------------------------
typedef struct
{
  const char *path_mnt;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_umount_t;
//----------------------------------------------------------
typedef struct
{
  const char *path;
  int ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_unlink_t;
//----------------------------------------------------------
typedef struct
{
  int status;
  pid_t ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_wait_t;
//----------------------------------------------------------
typedef struct
{
  int fdn;
  const void *buffer;
  size_t count;
  ssize_t ret;
  int errno;
  int errln;
  char errfn[PATH_MAX];
} sysmsg_write_t;
//----------------------------------------------------------
typedef struct
{
  char c;
} sysmsg_zpchar_t;
//----------------------------------------------------------
typedef struct
{
  char string[BUFSIZ];
} sysmsg_zpstring_t;
//----------------------------------------------------------
void input_line (char *line, char *prompt, size_t size,
                 int type);
int mount (const char *path_dev, const char *path_mnt,
           int options);
int namep (const char *name, char *path, size_t size);
void sys (int syscallnr, void *message, size_t size);
int umount (const char *path_mnt);
void z_perror (const char *string);
int z_printf (const char *restrict format, ...);
int z_vprintf (const char *restrict format, va_list arg);
int ipconfig (int n, h_addr_t address, int m);
int routedel (h_addr_t destination, int m);
int routeadd (h_addr_t destination, int m,
              h_addr_t router, int device);
//----------------------------------------------------------
#endif
```

95.21.1 lib/sys/os32/input_line.c

«

Si veda la sezione 88.68.

```
#include <sys/os32.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
//----------------------------------------------------------
static int terminal_echo (struct termios *orig);
static int terminal_noecho (struct termios *orig);
static int terminal_restore (struct termios *orig);
//----------------------------------------------------------
void
input_line (char *line, char *prompt, size_t size, int type)
{
  void *pstatus;
  int i;
  struct termios attr;
  //
  // Set terminal configuration.
  //
  if (type == INPUT_LINE_HIDDEN)
    {
      terminal_noecho (&attr);
    }
  else
    {
      terminal_echo (&attr);
    }
  //
  if (prompt != NULL || strlen (prompt) > 0)
    {
      printf ("%s", prompt);
    }
  //
  errno = 0;
  pstatus = fgets (line, (int) size, stdin);
  if (pstatus == NULL)
    {
      if (errno)
        {
          perror (NULL);
        }
      line[0] = 0;
      //
      // Reset terminal mode.
      //
      terminal_restore (&attr);
      return;
    }
  //
  // Find the last position and, if there is a new
  // line code,
  // replace it with zero. If the string is empty, a
  // ^D was
  // received.
  //
  i = strlen (line);
  if (i > 0 && line[i - 1] == '\n')
    {
      line[i - 1] = '\0';
    }
  //
  // Restore terminal mode.
  //
  terminal_restore (&attr);
}

//----------------------------------------------------------
static int
terminal_echo (struct termios *orig)
{
  int status;
  struct termios attr;
  //
  // Save previous.
  //
  status = tcgetattr (STDIN_FILENO, orig);
  if (status < 0)
    {
      return (-1);
    }
  //
  // Get again.
  //
  status = tcgetattr (STDIN_FILENO, &attr);
  if (status < 0)
    {
      return (-1);
    }
  //
  attr.c_iflag |= (BRKINT | ICRNL);
  attr.c_iflag &= ~(IGNBRK | INLCR);
  //
  attr.c_lflag |=
    (ECHO | ECHOE | ECHOK | ECHONL | ICANON | ISIG);
  attr.c_lflag &= ~(IEXTEN);
  //
  status = tcsetattr (STDIN_FILENO, TCSANOW, &attr);
  //
  return (status);
}

//----------------------------------------------------------
static int
terminal_noecho (struct termios *orig)
{
  int status;
  struct termios attr;
  //
  // Save previous.
  //
  status = tcgetattr (STDIN_FILENO, orig);
  if (status < 0)
    {
      return (-1);
    }
  //
  // Get again.
  //
  status = tcgetattr (STDIN_FILENO, &attr);
  if (status < 0)
    {
      return (-1);
    }
  //
  attr.c_iflag |= (BRKINT | ICRNL);
  attr.c_iflag &= ~(IGNBRK | INLCR);
  //
  attr.c_lflag |= (ICANON | ISIG);
  attr.c_lflag &= ~(ECHO | IEXTEN);
  //
  status = tcsetattr (STDIN_FILENO, TCSANOW, &attr);
  //
  return (status);
}

//----------------------------------------------------------
static int
terminal_restore (struct termios *orig)
{
  int status;
  //
  // For an unknown reason, when running with Bochs,
  // before
  // restoring the termios configuration, the previous
  // one
  // is to be read. Here, 'attr' is just a placeholder
  // and
  // the updated content is not used for anything
  // else.
  //
  struct termios attr;
  status = tcgetattr (STDIN_FILENO, &attr);
  if (status < 0)
    {
      return (-1);
    }
  //
  //
  //
  status = tcsetattr (STDIN_FILENO, TCSANOW, orig);
  //
  return (status);
}
```

95.21.2 lib/sys/os32/ipconfig.c

Si veda la sezione 87.28.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
//----------------------------------------------------------
int
ipconfig (int n, in_addr_t address, int m)
{
  sysmsg_ipconfig_t msg;
  //
  // Fill the message.
  //
  msg.n = n;
  msg.address = address;
  msg.m = m;
  msg.ret = 0;
  //
  // Syscall.
  //
  sys (SYS_IPCONFIG, &msg, (sizeof msg));
  //
  // Check return value.
  //
  if (msg.ret < 0)
    {
      //
      // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.21.3 lib/sys/os32/mount.c

«

Si veda la sezione 87.36.

```
#include <sys/types.h>
#include <errno.h>
#include <sys/os32.h>
#include <stddef.h>
#include <string.h>
//----------------------------------------------------------------------
int
mount (const char *path_dev, const char *path_mnt,
       int options)
```

```
{
  sysmsg_mount_t msg;
  //
  msg.path_dev = path_dev;
  msg.path_mnt = path_mnt;
  msg.options = options;
  msg.ret = 0;
  msg.errno = 0;
  //
  sys (SYS_MOUNT, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.21.4 lib/sys/os32/namep.c

Si veda la sezione 88.85.

```
#include <sys/os32.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
//----------------------------------------------------------
int
namep (const char *name, char *path, size_t size)
{
  char command[PATH_MAX];
  char *env_path;
  int p;          // Index used inside the path
  // environment.
  int c;          // Index used inside the command
  // string.
  int status;
  //
```

```
    // Check for valid input.
    //
    if (name == NULL || name[0] == 0 || path == NULL
        || name == path)
      {
        errset (EINVAL);  // Invalid argument.
        return (-1);
      }
    //
    // Check if the original command contains at least a
    // '/'. Otherwise
    // a scan for the environment variable 'PATH' must
    // be done.
    //
    if (strchr (name, '/') == NULL)
      {
        //
        // Ok: no '/' there. Get the environment
        // variable 'PATH'.
        //
        env_path = getenv ("PATH");
        if (env_path == NULL)
          {
            //
            // There is no 'PATH' environment value.
            //
            errset (ENOENT);      // No such file or
            // directory.
            return (-1);
          }
        //
        // Scan paths and try to find a file with that
        // name.
        //
        for (p = 0; env_path[p] != 0;)
          {
            for (c = 0;
             c < (PATH_MAX - strlen (name) - 2) &&
             env_path[p] != 0 &&
             env_path[p] != ':'; c++, p++)
          {
            command[c] = env_path[p];
          }
        //
        // If the loop is ended because the command
        // array does not
        // have enough room for the full path, then
        // must return an
        // error.
        //
        if (env_path[p] != ':' && env_path[p] != 0)
          {
            errset (ENAMETOOLONG);    // Filename
            // too long.
            return (-1);
          }
        //
        // The command array has enough space. At
        // index 'c' must
        // place a zero, to terminate current
        // string.
        //
        command[c] = 0;
        //
        // Add the rest of the path.
        //
        strcat (command, "/");
        strcat (command, name);
        //
        // Verify to have something with that full
        // path name.
        //
        status = access (command, F_OK);
        if (status == 0)
```

```
          {
            //
            // Verify to have enough room inside the
            // destination
            // path.
            //
            if (strlen (command) >= size)
              {
                //
                // Sorry: too big. There must be
                // room also for
                // the string termination null
                // character.
                //
                errset (ENAMETOOLONG);        // Filename
                // too long.
                return (-1);
              }
            //
            // Copy the path and return.
            //
            strncpy (path, command, size);
            return (0);
          }
        //
        // That path was not good: try again. But
        // before returning
        // to the external loop, must verify if ‘p’
        // is to be
        // incremented, after a ‘:’, because the
        // external loop
        // does not touch the index ‘p’,
        //
        if (env_path[p] == ':')
          {
            p++;
          }
        }
      //
      // At this point, there is no match with the
      // paths.
      //
      errset (ENOENT);  // No such file or directory.
      return (-1);
    }
  //
  // At this point, a path was given and the
  // environment variable
  // 'PATH' was not scanned. Just copy the same path.
  // But must verify
  // that the receiving path has enough room for it.
  //
  if (strlen (name) >= size)
    {
      //
      // Sorry: too big.
      //
      errset (ENAMETOOLONG);    // Filename too long.
      return (-1);
    }
  //
  // Ok: copy and return.
  //
  strncpy (path, name, size);
  return (0);
}
```

95.21.5 lib/sys/os32/routeadd.c

Si veda la sezione 87.42.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
```

```
#include <stdio.h>
//----------------------------------------------------------
int
routeadd (in_addr_t destination, int m,
          in_addr_t router, int device)
{
  sysmsg_route_t msg;
  //
  // Fill the message.
  //
  msg.destination = destination;
  msg.m = m;
  msg.router = router;
  msg.device = device;
  //
  // Syscall.
  //
  sys (SYS_ROUTEADD, &msg, (sizeof msg));
  //
  // Check return value.
  //
  if (msg.ret < 0)
    {
      //
      // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.21.6 lib/sys/os32/routedel.c

Si veda la sezione 87.43.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
//----------------------------------------------------------
int
routedel (in_addr_t destination, int m)
{
  sysmsg_route_t msg;
  //
  // Fill the message.
  //
  msg.destination = destination;
  msg.m = m;
  //
  // Syscall.
  //
  sys (SYS_ROUTEDEL, &msg, (sizeof msg));
  //
  // Check return value.
  //
  if (msg.ret < 0)
    {
      //
      // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.21.7 lib/sys/os32/sys.s

Si veda la sezione 87.56.

```
.global sys
#-------------------------------------------------------
.text
#-------------------------------------------------------
# Call a system call.
#
# Please remember that system calls should never be
# used (called) inside the kernel code, because system
# calls cannot be nested for the os32 simple
# architecture!
# If a particular function is necessary inside the
# kernel, that usually is made by a system call, an
# appropriate k_...() function must be made, to avoid
# the problem.
#
#-------------------------------------------------------
.align 4
sys:
    int   $128  # 0x80
    ret
```

95.21.8 lib/sys/os32/umount.c

Si veda la sezione 87.36.

```
#include <sys/types.h>
#include <errno.h>
#include <sys/os32.h>
#include <stddef.h>
#include <string.h>
//-----------------------------------------------------------
```

```
int
umount (const char *path_mnt)
{
  sysmsg_umount_t msg;
  //
  msg.path_mnt = path_mnt;
  msg.ret = 0;
  msg.errno = 0;
  //
  sys (SYS_UMOUNT, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.21.9 lib/sys/os32/z_perror.c

Si veda la sezione 87.65.

```
#include <sys/os32.h>
#include <errno.h>
#include <stddef.h>
#include <string.h>
//-----------------------------------------------------------
void
z_perror (const char *string)
{
  //
  // If errno is zero, there is nothing to show.
  //
  if (errno == 0)
    {
      return;
    }
  //
  // Show the string if there is one.
  //
  if (string != NULL && strlen (string) > 0)
    {
      z_printf ("%s: ", string);
    }
  //
  // Show the translated error.
  //
  if (errfn[0] != 0 && errln != 0)
    {
      z_printf ("[%s:%u:%i] %s\n",
                errfn, errln, errno, strerror (errno));
    }
  else
    {
      z_printf ("[%i] %s\n", errno, strerror (errno));
    }
}
```

95.21.10 lib/sys/os32/z_printf.c

Si veda la sezione 87.65.

```
#include <sys/os32.h>
#include <restrict.h>
//----------------------------------------------------------
int
z_printf (const char *restrict format, ...)
{
  va_list ap;
  va_start (ap, format);
  return z_vprintf (format, ap);
}
```

95.21.11 lib/sys/os32/z_vprintf.c

Si veda la sezione 87.65.

```
#include <sys/os32.h>
#include <restrict.h>
//----------------------------------------------------------
int
z_vprintf (const char *restrict format, va_list arg)
{
  int ret;
  sysmsg_zpstring_t msg;
  msg.string[0] = 0;
  ret = vsprintf (msg.string, format, arg);
  sys (SYS_ZPSTRING, &msg, (sizeof msg));
  return ret;
}
```

95.22 os32: «lib/sys/sa_family_t.h»

Si veda la sezione 91.3.

```
#ifndef _SYS_SA_FAMILY_T_H
#define _SYS_SA_FAMILY_T_H    1
//----------------------------------------------------------
#include <inttypes.h>
//----------------------------------------------------------
typedef uint16_t sa_family_t;   // Address family.
//----------------------------------------------------------
#endif
```

95.23 os32: «lib/sys/socket.h»

«

Si veda la sezione 91.3.

```
#ifndef _SYS_SOCKET_H
#define _SYS_SOCKET_H    1
//----------------------------------------------------------
#include <stdint.h>
#include <unistd.h>
#include <sys/socklen_t.h>
#include <sys/sa_family_t.h>
//----------------------------------------------------------
struct sockaddr
{
  sa_family_t sa_family;        // Address family.
  char sa_data[14];     // Socket address.
};
//
//
//
struct sockaddr_storage
{
  sa_family_t ss_family;        // Socket storage
  // family.
  uint8_t ss_zero[14];  // Filler.
};
//
//
//
#define SOCK_STREAM    1        // Byte-stream socket.
#define SOCK_DGRAM     2        // Datagram socket.
#define SOCK_RAW       3        // Raw protocol
                                // interface.
#define SOCK_SEQPACKET 5        // Sequenced-packet
                                // socket.
//
// Protocol families:
//
```

```
#define PF_UNSPEC       0       // Unspecified.
#define PF_UNIX         1       // Unix domain socket.
#define PF_INET         2       // IPv4 protocol
                                // family.
#define PF_INET6        10      // IPv6 protocol
                                // family.
//
// Address families.
//
#define AF_UNSPEC  PF_UNSPEC    // Unspecified.
#define AF_UNIX    PF_UNIX      // Unix domain socket.
#define AF_INET    PF_INET      // IPv4 address
                                // family.
#define AF_INET6   PF_INET6     // IPv6 address
                                // family.
//----------------------------------------------------------
int accept (int sfdn, struct sockaddr *addr,
            socklen_t * addrlen);
int bind (int sfdn, const struct sockaddr *addr,
          socklen_t addrlen);
int connect (int sfdn, const struct sockaddr *addr,
             socklen_t addrlen);
int listen (int sfdn, int backlog);
ssize_t send (int sfdn, const void *buffer,
              size_t count, int flags);
ssize_t recvfrom (int sfdn, void *buffer, size_t count,
                  int flags, struct sockaddr *addrfrom,
                  socklen_t * addrlen);
int socket (int family, int type, int protocol);

#define recv(sdfn, buffer, count, flags) \
    recvfrom (sdfn, buffer, count, flags, NULL, NULL)
//----------------------------------------------------------
#endif
```

95.23.1 lib/sys/socket/accept.c 2131

95.23.1 lib/sys/socket/accept.c

Si veda la sezione 87.3.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
//----------------------------------------------------------
int
accept (int sfdn, struct sockaddr *addr,
        socklen_t * addrlen)
{
  sysmsg_accept_t msg;
  //
  // Fill the message.
  //
  msg.sfdn = sfdn;
  memset (&msg.addr, 0x00, sizeof (msg.addr));
  msg.addrlen = *addrlen;
  msg.fl_flags = 0;     // Not necessary.
  msg.ret = 0;
  //
  // Syscall.
  //
  while (1)
```

```
      {
        sys (SYS_ACCEPT, &msg, (sizeof msg));
        //
        if (msg.ret < 0
            && (msg.errno == EAGAIN
                || msg.errno == EWOULDBLOCK))
          {
            //
            // No request at the moment.
            //
            if (msg.fl_flags & O_NONBLOCK)
              {
                //
                // Don't block.
                //
                break;
              }
            else
              {
                //
                // Keep trying.
                //
                continue;
              }
          }
        else
          {
            break;
          }
      }
    //
    // Check return value.
    //
    if (msg.ret < 0)
      {
        //
        // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  else
    {
      //
      // Update the socket address and the address
      // length.
      //
      if (addrlen != NULL && addr != NULL && *addrlen > 0)
        {
          memcpy (addr, &msg.addr,
                  min (msg.addrlen, *addrlen));
          *addrlen = msg.addrlen;
        }
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.23.2 lib/sys/socket/bind.c

«

Si veda la sezione 87.4.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
//----------------------------------------------------------------
int
bind (int sfdn, const struct sockaddr *addr,
      socklen_t addrlen)
{
  sysmsg_bind_t msg;
  //
  // Fill the message.
  //
  msg.sfdn = sfdn;
  memcpy (&msg.addr, addr, (size_t) addrlen);
  msg.addrlen = addrlen;
  msg.ret = 0;
  //
  // Syscall.
  //
  sys (SYS_BIND, &msg, (sizeof msg));
  //
  // Check return value.
  //
  if (msg.ret < 0)
    {
      //
      // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.23.3 lib/sys/socket/connect.c

Si veda la sezione 87.11.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
```

```
#include <stdio.h>
//----------------------------------------------------------
int
connect (int sfdn, const struct sockaddr *addr,
         socklen_t addrlen)
{
  sysmsg_connect_t msg;
  //
  // Fill the message.
  //
  msg.sfdn = sfdn;
  memcpy (&msg.addr, addr, (size_t) addrlen);
  msg.addrlen = addrlen;
  msg.ret = 0;
  //
  // Syscall.
  //
  while (1)
    {
      sys (SYS_CONNECT, &msg, (sizeof msg));
      //
      if (msg.ret < 0)
        {
          if (msg.errno == EINPROGRESS
              || msg.errno == EALREADY)
            {
              //
              // Loop until the connection is
              // established, or a
              // different error comes.
              //
              continue;
            }
          else
            {
              break;
            }
        }
      else
        {
          break;
        }
    }
  //
  // Check return value.
  //
  if (msg.ret < 0)
    {
      //
      // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.23.4 lib/sys/socket/listen.c

Si veda la sezione 87.31.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
//----------------------------------------------------------------------
int
listen (int sfdn, int backlog)
{
  sysmsg_listen_t msg;
  //
  // Fill the message.
  //
  msg.sfdn = sfdn;
  msg.backlog = backlog;
  msg.ret = 0;
  //
  // Syscall.
  //
  sys (SYS_LISTEN, &msg, (sizeof msg));
  //
  // Check return value.
  //
  if (msg.ret < 0)
    {
      //
      // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.23.5 lib/sys/socket/recvfrom.c

Si veda la sezione 87.40.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
```

```
//----------------------------------------------------------
ssize_t
recvfrom (int sfdn, void *buffer, size_t count,
          int flags, struct sockaddr *addrfrom,
          socklen_t * addrlen)
{
  sysmsg_recvfrom_t msg;
  //
  // Reduce size of read if necessary.
  //
  if (count > BUFSIZ)
    {
      count = BUFSIZ;
    }
  //
  // Fill the message.
  //
  msg.sfdn = sfdn;
  msg.buffer = buffer;
  msg.count = count;
  msg.flags = flags;
  msg.addrfrom = addrfrom;
  msg.addrsize = addrlen;
  msg.fl_flags = 0;     // Not necessary.
  msg.ret = 0;
  //
  // Repeat syscall, until something is received or
  // end of file is
  // reached.
  //
  while (1)
    {
      sys (SYS_RECVFROM, &msg, (sizeof msg));
      if (msg.ret == 0)
        {
          //
          // Stream closed from the other side.
```

```
            //
            break;
          }
        if (msg.ret < 0
            && (msg.errno == EAGAIN
                || msg.errno == EWOULDBLOCK))
          {
            //
            // No data at the moment.
            //
            if (msg.fl_flags & O_NONBLOCK)
              {
                //
                // Don't block.
                //
                break;
              }
            else
              {
                //
                // Keep trying.
                //
                continue;
              }
          }
        //
        // Otherwise, we have received something.
        //
        break;
      }
    //
    //
    //
    if (msg.ret < 0)
      {
        //
        // No valid read.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
      return (msg.ret);
    }
  //
  if (msg.ret > count)
    {
      //
      // A strange value was returned. Considering it
      // a read error.
      //
      errset (EIO);     // I/O error.
      return (-1);
    }
  //
  // A valid read: return.
  //
  return (msg.ret);
}
```

95.23.6 lib/sys/socket/send.c

Si veda la sezione 87.45.

```
#include <unistd.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
//-----------------------------------------------------------
ssize_t
send (int sfdn, const void *buffer, size_t count, int flags)
{
  sysmsg_send_t msg;
  int retry = 3;
  //
  // Reduce size of write if necessary.
  //
  if (count > BUFSIZ)
    {
      count = BUFSIZ;
    }
  //
  // Fill the message.
  //
  msg.sfdn = sfdn;
  msg.buffer = buffer;
  msg.count = count;
  msg.flags = flags;
  //
  // Syscall.
  //
  for (; retry > 0; retry--)
    {
      sys (SYS_SEND, &msg, (sizeof msg));
      //
      // Check.
      //
      if ((msg.ret < 0) && (msg.errno == E_ARP_MISSING))
        {
          sleep (1);
          continue;     // Retry.
        }
      else
        {
          break;
        }
    }
  //
  // Check the final result and return.
  //
  if (msg.ret < 0)
```

```
    {
      //
      // No valid write.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
      return (msg.ret);
    }
  //
  if (msg.ret > count)
    {
      //
      // A strange value was returned. Considering it
      // a read error.
      //
      errset (EIO);     // I/O error.
      return (-1);
    }
  //
  // A valid write return.
  //
  return (msg.ret);
}
```

95.23.7 lib/sys/socket/socket.c

Si veda la sezione 87.54.

```
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
//----------------------------------------------------------
int
socket (int family, int type, int protocol)
{
  sysmsg_socket_t msg;
  //
  // Fill the message.
  //
  msg.family = family;
  msg.type = type;
  msg.protocol = protocol;
  msg.ret = 0;
  //
  // Syscall.
  //
  sys (SYS_SOCKET, &msg, (sizeof msg));
  //
  // Check return value.
  //
  if (msg.ret < 0)
    {
      //
      // Something wrong.
      //
      errno = msg.errno;
      errln = msg.errln;
      strncpy (errfn, msg.errfn, PATH_MAX);
    }
  //
  // Return.
  //
  return (msg.ret);
}
```

95.24 os32: «lib/sys/socklen_t.h»

Si veda la sezione 91.3.

```
#ifndef _SYS_SOCKLEN_T_H
#define _SYS_SOCKLEN_T_H    1
//----------------------------------------------------------
```

```
#include <stdint.h>
//----------------------------------------------------------
typedef uint32_t socklen_t;
//----------------------------------------------------------
#endif
```

95.25 os32: «lib/sys/stat.h»

«

Si veda la sezione 91.3.

```
#ifndef _SYS_STAT_H
#define _SYS_STAT_H     1

#include <restrict.h>
#include <sys/types.h>  // dev_t
                        // off_t
                        // blkcnt_t
                        // blksize_t
                        // ino_t
                        // mode_t
                        // nlink_t
                        // uid_t
                        // gid_t
                        // time_t
//----------------------------------------------------------
// File type.
//----------------------------------------------------------
#define S_IFMT   0170000        // File type mask.
//
#define S_IFBLK  0060000        // Block device file.
#define S_IFCHR  0020000        // Character device
                                // file.
#define S_IFIFO  0010000        // Pipe (FIFO) file.
#define S_IFREG  0100000        // Regular file.
#define S_IFDIR  0040000        // Directory.
```

```
#define S_IFLNK  0120000        // Symbolic link.
#define S_IFSOCK 0140000        // Unix domain socket.
//----------------------------------------------------------
// Owner user access permissions.
//----------------------------------------------------------
#define S_IRWXU  0000700        // Owner user access
                                // permissions mask.
//
#define S_IRUSR  0000400        // Owner user read
                                // access permission.
#define S_IWUSR  0000200        // Owner user write
                                // access permission.
#define S_IXUSR  0000100        // Owner user
                                // execution or cross
                                // perm.
//----------------------------------------------------------
// Group owner access permissions.
//----------------------------------------------------------
#define S_IRWXG  0000070        // Owner group access
                                // permissions mask.
//
#define S_IRGRP  0000040        // Owner group read
                                // access permission.
#define S_IWGRP  0000020        // Owner group write
                                // access permission.
#define S_IXGRP  0000010        // Owner group
                                // execution or cross
                                // perm.
//----------------------------------------------------------
// Other users access permissions.
//----------------------------------------------------------
#define S_IRWXO  0000007        // Other users access
                                // permissions mask.
//
#define S_IROTH  0000004        // Other users read
                                // access permission.
#define S_IWOTH  0000002        // Other users write
```

```
                                // access permissions.
#define S_IXOTH  0000001        // Other users
                                // execution or cross
                                // perm.
//----------------------------------------------------------
// S-bit: in this case there is no mask to select all
// of them.
//----------------------------------------------------------
#define S_ISUID  0004000        // S-UID.
#define S_ISGID  0002000        // S-GID.
#define S_ISVTX  0001000        // Sticky.
//----------------------------------------------------------
// Macroinstructions to verify the type of file.
//----------------------------------------------------------
//
// Block device:
//
#define S_ISBLK(m)   (((m) & S_IFMT) == S_IFBLK)
//
// Character device:
//
#define S_ISCHR(m)   (((m) & S_IFMT) == S_IFCHR)
//
// FIFO.
//
#define S_ISFIFO(m)  (((m) & S_IFMT) == S_IFIFO)
//
// Regular file.
//
#define S_ISREG(m)   (((m) & S_IFMT) == S_IFREG)
//
// Directory.
//
#define S_ISDIR(m)   (((m) & S_IFMT) == S_IFDIR)
//
// Symbolic link.
//
```

```
#define S_ISLNK(m)    (((m) & S_IFMT) == S_IFLNK)
//
// Socket (Unix domain socket).
//
#define S_ISSOCK(m)   (((m) & S_IFMT) == S_IFSOCK)
//----------------------------------------------------------
// Structure 'stat'.
//----------------------------------------------------------
struct stat
{
  dev_t st_dev; // Device containing the file.
  ino_t st_ino; // File serial number (inode number).
  mode_t st_mode;       // File type and permissions.
  nlink_t st_nlink;     // Links to the file.
  uid_t st_uid; // Owner user id.
  gid_t st_gid; // Owner group id.
  dev_t st_rdev;        // Device number if it is a
  // device file.
  off_t st_size;        // File size.
  time_t st_atime;      // Last access time.
  time_t st_mtime;      // Last modification time.
  time_t st_ctime;      // Last inode modification.
  blksize_t st_blksize; // Block size for I/O
  // operations.
  blkcnt_t st_blocks;   // File size / block size.
};
//----------------------------------------------------------
// Function prototypes.
//----------------------------------------------------------
int chmod (const char *path, mode_t mode);
int fchmod (int fdn, mode_t mode);
int fstat (int fdn, struct stat *buffer);
int lstat (const char *restrict path,
           struct stat *restrict buffer);
int mkdir (const char *path, mode_t mode);
int mkfifo (const char *path, mode_t mode);
int mknod (const char *path, mode_t mode, dev_t dev);
int stat (const char *restrict path,
          struct stat *restrict buffer);
mode_t umask (mode_t mask);

#endif // _SYS_STAT_H
```

95.25.1 lib/sys/stat/chmod.c

«

Si veda la sezione 87.7.

```
#include <sys/stat.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
chmod (const char *path, mode_t mode)
{
  sysmsg_chmod_t msg;
  //
  msg.path = path;
  msg.mode = mode;
  //
  sys (SYS_CHMOD, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.25.2 lib/sys/stat/fchmod.c

«

Si veda la sezione 87.7.

```
#include <sys/stat.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
fchmod (int fdn, mode_t mode)
{
  sysmsg_fchmod_t msg;
  //
  msg.fdn = fdn;
  msg.mode = mode;
  //
  sys (SYS_FCHMOD, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.25.3 lib/sys/stat/fstat.c

Si veda la sezione 87.55.

```
#include <unistd.h>
#include <errno.h>
#include <sys/os32.h>
#include <string.h>
//----------------------------------------------------------
int
fstat (int fdn, struct stat *buffer)
{
  sysmsg_fstat_t msg;
  //
  msg.fdn = fdn;
  msg.stat.st_dev = buffer->st_dev;
  msg.stat.st_ino = buffer->st_ino;
  msg.stat.st_mode = buffer->st_mode;
  msg.stat.st_nlink = buffer->st_nlink;
  msg.stat.st_uid = buffer->st_uid;
  msg.stat.st_gid = buffer->st_gid;
  msg.stat.st_rdev = buffer->st_rdev;
  msg.stat.st_size = buffer->st_size;
  msg.stat.st_atime = buffer->st_atime;
  msg.stat.st_mtime = buffer->st_mtime;
  msg.stat.st_ctime = buffer->st_ctime;
  msg.stat.st_blksize = buffer->st_blksize;
  msg.stat.st_blocks = buffer->st_blocks;
  //
  sys (SYS_FSTAT, &msg, (sizeof msg));
  //
  buffer->st_dev = msg.stat.st_dev;
  buffer->st_ino = msg.stat.st_ino;
  buffer->st_mode = msg.stat.st_mode;
  buffer->st_nlink = msg.stat.st_nlink;
  buffer->st_uid = msg.stat.st_uid;
  buffer->st_gid = msg.stat.st_gid;
  buffer->st_rdev = msg.stat.st_rdev;
  buffer->st_size = msg.stat.st_size;
  buffer->st_atime = msg.stat.st_atime;
  buffer->st_mtime = msg.stat.st_mtime;
  buffer->st_ctime = msg.stat.st_ctime;
  buffer->st_blksize = msg.stat.st_blksize;
  buffer->st_blocks = msg.stat.st_blocks;
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.25.4 lib/sys/stat/mkdir.c

«

Si veda la sezione 87.34.

```
#include <sys/stat.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
mkdir (const char *path, mode_t mode)
{
  sysmsg_mkdir_t msg;
  //
  msg.path = path;
  msg.mode = mode;
  //
  sys (SYS_MKDIR, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.25.5 lib/sys/stat/mknod.c

Si veda la sezione 87.35.

```
#include <unistd.h>
#include <errno.h>
#include <sys/os32.h>
#include <string.h>
//----------------------------------------------------------
int
mknod (const char *path, mode_t mode, dev_t device)
{
  sysmsg_mknod_t msg;
  //
  msg.path = path;
  msg.mode = mode;
  msg.device = device;
  //
  sys (SYS_MKNOD, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.25.6 lib/sys/stat/stat.c

Si veda la sezione 87.55.

```
#include <unistd.h>
#include <errno.h>
#include <sys/os32.h>
#include <string.h>
//----------------------------------------------------------
```

```
int
stat (const char *path, struct stat *buffer)
{
  sysmsg_stat_t msg;
  //
  msg.path = path;
  //
  msg.stat.st_dev = buffer->st_dev;
  msg.stat.st_ino = buffer->st_ino;
  msg.stat.st_mode = buffer->st_mode;
  msg.stat.st_nlink = buffer->st_nlink;
  msg.stat.st_uid = buffer->st_uid;
  msg.stat.st_gid = buffer->st_gid;
  msg.stat.st_rdev = buffer->st_rdev;
  msg.stat.st_size = buffer->st_size;
  msg.stat.st_atime = buffer->st_atime;
  msg.stat.st_mtime = buffer->st_mtime;
  msg.stat.st_ctime = buffer->st_ctime;
  msg.stat.st_blksize = buffer->st_blksize;
  msg.stat.st_blocks = buffer->st_blocks;
  //
  sys (SYS_STAT, &msg, (sizeof msg));
  //
  buffer->st_dev = msg.stat.st_dev;
  buffer->st_ino = msg.stat.st_ino;
  buffer->st_mode = msg.stat.st_mode;
  buffer->st_nlink = msg.stat.st_nlink;
  buffer->st_uid = msg.stat.st_uid;
  buffer->st_gid = msg.stat.st_gid;
  buffer->st_rdev = msg.stat.st_rdev;
  buffer->st_size = msg.stat.st_size;
  buffer->st_atime = msg.stat.st_atime;
  buffer->st_mtime = msg.stat.st_mtime;
  buffer->st_ctime = msg.stat.st_ctime;
  buffer->st_blksize = msg.stat.st_blksize;
  buffer->st_blocks = msg.stat.st_blocks;
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.25.7 lib/sys/stat/umask.c

Si veda la sezione 87.60.

```
#include <sys/stat.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------------------
mode_t
umask (mode_t mask)
{
  sysmsg_umask_t msg;
  msg.umask = mask;
  sys (SYS_UMASK, &msg, (sizeof msg));
  return (msg.ret);
}
```

95.26 os32: «lib/sys/types.h»

Si veda la sezione 91.3.

```
#ifndef _SYS_TYPES_H
#define _SYS_TYPES_H    1
//----------------------------------------------------------------------
#include <clock_t.h>
#include <time_t.h>
#include <size_t.h>
//----------------------------------------------------------------------
```

```
typedef long int blkcnt_t;
typedef long int blksize_t;
typedef uint16_t dev_t; // Traditional device size.
typedef unsigned int id_t;
typedef unsigned int gid_t;
typedef unsigned int uid_t;
typedef uint16_t ino_t; // Minix 1 file system inode
                        // size.
typedef uint16_t mode_t;        // Minix 1 file system
                                // mode size.
typedef unsigned int nlink_t;
typedef long long int off_t;
typedef int pid_t;
typedef unsigned long int pthread_t;
typedef int ssize_t;
//----------------------------------------------------------
// Common extentions.
//
dev_t makedev (int major, int minor);
int major (dev_t device);
int minor (dev_t device);
//----------------------------------------------------------
#endif
```

95.26.1 lib/sys/types/major.c

Si veda la sezione 88.75.

```
#include <sys/types.h>
//----------------------------------------------------------
int
```

```
major (dev_t device)
{
    return ((int) (device / 256));
}
```

95.26.2 lib/sys/types/makedev.c

Si veda la sezione 88.75.

```
#include <sys/types.h>
//----------------------------------------------------------
dev_t
makedev (int major, int minor)
{
    return ((dev_t) (major * 256 + minor));
}
```

95.26.3 lib/sys/types/minor.c

Si veda la sezione 88.75.

```
#include <sys/types.h>
//----------------------------------------------------------
int
minor (dev_t device)
{
    return ((dev_t) (device & 0x00FF));
}
```

95.27 os32: «lib/sys/wait.h»

Si veda la sezione 91.3.

```
#ifndef _SYS_WAIT_H
#define _SYS_WAIT_H       1

```

```
#include <sys/types.h>

//----------------------------------------------------------
pid_t wait (int *status);
//----------------------------------------------------------

#endif
```

95.27.1 lib/sys/wait/wait.c

Si veda la sezione 87.63.

```
#include <sys/types.h>
#include <errno.h>
#include <sys/os32.h>
#include <stddef.h>
#include <string.h>
//----------------------------------------------------------
pid_t
wait (int *status)
{
  sysmsg_wait_t msg;
  msg.ret = 0;
  msg.errno = 0;
  msg.status = 0;
  while (msg.ret == 0)
    {
      //
      // Loop as long as there are children, an none
      // is dead.
      //
      sys (SYS_WAIT, &msg, (sizeof msg));
    }
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  //
  if (status != NULL)
    {
      //
      // Only the low eight bits are returned.
      //
      *status = (msg.status & 0x00FF);
    }
  return (msg.ret);
}
```

95.28 os32: «lib/termios.h»

Si veda la sezione 87.58.

```
#ifndef _TERMIOS_H
#define _TERMIOS_H       1
//----------------------------------------------------------
#include <stdint.h>
//----------------------------------------------------------
typedef uint16_t tcflag_t;
typedef unsigned char cc_t;
//----------------------------------------------------------
#define NCCS    11      // 'c_cc[]' size.
//
struct termios
{
  tcflag_t c_iflag;
  tcflag_t c_oflag;
  tcflag_t c_cflag;
  tcflag_t c_lflag;
  cc_t c_cc[NCCS];
};
//
// Subscript names for 'c_cc[]' array, inside the
// 'termios' structure:
```

```
//
#define VEOF      0     // EOF character
#define VEOL      1     // EOL character
#define VERASE    2     // ERASE character
#define VINTR     3     // INTR character
#define VKILL     4     // KILL character
#define VMIN      5     // MIN value
#define VQUIT     6     // QUIT character
#define VSTART    7     // START character
#define VSTOP     8     // STOP character
#define VSUSP     9     // SUSP character
#define VTIME    10     // TIME value
//
// Input modes, for `c_iflag' inside the `termios'
// structure:
//
#define BRKINT    1     // signal interrupt on break
#define ICRNL     2     // map CR to NL on input
#define IGNBRK    4     // ignore break condition
#define IGNCR     8     // ignore CR
#define IGNPAR   16     // ignore characters with
                        // parity errors
#define INLCR    32     // map NL to CR on input
#define INPCK    64     // enable input parity check
#define ISTRIP  128     // strip off eighth bit
#define IXOFF   256     // enable start/stop input
                        // control
#define IXON    512     // enable start/stop output
                        // control
#define PARMRK 1024     // mark parity errors
//
// Output modes, for `c_oflag' inside the `termios'
// structure:
//
#define OPOST     1     // post-process output
//
// Control modes, for `c_cflag' inside the `termios'
// structure:
//   not implemented.
//
//
// Local modes, for ‘c_lflag’ inside the ‘termios’
// structure:
//
#define ECHO      1     // enable echo
#define ECHOE     2     // echo erase character as
                        // backspace
#define ECHOK     4     // echo KILL
#define ECHONL    8     // echo NL
#define ICANON   16     // canonical input mode
#define IEXTEN   32     // extended input mode
#define ISIG     64     // enable signals
#define NOFLSH  128     // disable flush after
                        // interrupt or quit
#define TOSTOP  256     // send SIGTTOU for background
                        // output
//----------------------------------------------------------
// Optional action for use with ‘tcsetattr()’:
//
#define TCSANOW   1     // change attributes
                        // immediately
#define TCSADRAIN 2     // change attributes when
                        // output has drained
#define TCSAFLUSH 3     // change attributes when
                        // output has drained,
                        // and also flush pending
                        // input
//----------------------------------------------------------
int tcgetattr (int fdn, struct termios *termios_p);
int tcsetattr (int fdn, int action,
               struct termios *termios_p);
//----------------------------------------------------------
#endif
```

95.28.1 lib/termios/tcgetattr.c

Si veda la sezione 87.58.

```
#include <termios.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
#define DEBUG 0
//----------------------------------------------------------
int
tcgetattr (int fdn, struct termios *termios_p)
{
  sysmsg_tcattr_t msg;
  msg.fdn = fdn;
  msg.attr = termios_p;
  sys (SYS_TCGETATTR, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.28.2 lib/termios/tcsetattr.c

Si veda la sezione 87.58.

```
#include <termios.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
#define DEBUG 0
//----------------------------------------------------------
int
```

```
tcsetattr (int fdn, int action, struct termios *termios_p)
{
  sysmsg_tcattr_t msg;
  msg.fdn = fdn;
  msg.action = action;
  msg.attr = termios_p;
  sys (SYS_TCSETATTR, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.29 os32: «lib/time.h»

Si veda la sezione 91.3.

```
#ifndef _TIME_H
#define _TIME_H       1
//----------------------------------------------------------
#include <restrict.h>
#include <size_t.h>
#include <time_t.h>
#include <clock_t.h>
#include <NULL.h>
#include <stdint.h>
//----------------------------------------------------------
#define CLOCKS_PER_SEC  ((clock_t) 100)
//----------------------------------------------------------
struct tm
{
  int tm_sec;
  int tm_min;
  int tm_hour;
  int tm_mday;
  int tm_mon;
  int tm_year;
  int tm_wday;
  int tm_yday;
  int tm_isdst;
};
//----------------------------------------------------------
clock_t clock (void);
time_t time (time_t * timer);
int stime (time_t * timer);
double difftime (time_t time1, time_t time0);
time_t mktime (const struct tm *timeptr);
struct tm *gmtime (const time_t * timer);
struct tm *localtime (const time_t * timer);
char *asctime (const struct tm *timeptr);
char *ctime (const time_t * timer);
size_t strftime (char *restrict s, size_t maxsize,
                 const char *restrict format,
                 const struct tm *restrict timeptr);
//----------------------------------------------------------
#define difftime(t1,t0) ((double)((t1)-(t0)))
#define ctime(t)        (asctime (localtime (t)))
#define localtime(t)    (gmtime (t))
//----------------------------------------------------------
#endif
```

95.29.1 lib/time/asctime.c

Si veda la sezione 88.15.

```
#include <time.h>
#include <string.h>
#include <stdio.h>
//----------------------------------------------------------
char *
asctime (const struct tm *timeptr)
{
  static char time_string[25];  // ‘Sun Jan 30
  // 24:00:00 2111’
  //
  // Check argument.
  //
  if (timeptr == NULL)
    {
      return (NULL);
    }
  //
  // Set week day.
  //
  switch (timeptr->tm_wday)
    {
    case 0:
      strcpy (&time_string[0], "Sun");
      break;
    case 1:
      strcpy (&time_string[0], "Mon");
      break;
    case 2:
      strcpy (&time_string[0], "Tue");
      break;
    case 3:
      strcpy (&time_string[0], "Wed");
      break;
    case 4:
        strcpy (&time_string[0], "Thu");
        break;
      case 5:
        strcpy (&time_string[0], "Fri");
        break;
      case 6:
        strcpy (&time_string[0], "Sat");
        break;
      default:
        strcpy (&time_string[0], "Err");
      }
    //
    // Set month.
    //
    switch (timeptr->tm_mon)
      {
      case 1:
        strcpy (&time_string[3], " Jan");
        break;
      case 2:
        strcpy (&time_string[3], " Feb");
        break;
      case 3:
        strcpy (&time_string[3], " Mar");
        break;
      case 4:
        strcpy (&time_string[3], " Apr");
        break;
      case 5:
        strcpy (&time_string[3], " May");
        break;
      case 6:
        strcpy (&time_string[3], " Jun");
        break;
      case 7:
        strcpy (&time_string[3], " Jul");
        break;
    case 8:
      strcpy (&time_string[3], " Aug");
      break;
    case 9:
      strcpy (&time_string[3], " Sep");
      break;
    case 10:
      strcpy (&time_string[3], " Oct");
      break;
    case 11:
      strcpy (&time_string[3], " Nov");
      break;
    case 12:
      strcpy (&time_string[3], " Dec");
      break;
    default:
      strcpy (&time_string[3], " Err");
    }
  //
  // Set day of month, hour, minute, second and year.
  //
  sprintf (&time_string[7], " %2i %2i:%2i:%2i %4i",
           timeptr->tm_mday, timeptr->tm_hour,
           timeptr->tm_min, timeptr->tm_sec,
           timeptr->tm_year);
  //
  //
  //
  return (&time_string[0]);
}
```

95.29.2 lib/time/clock.c

«

Si veda la sezione 87.9.

```
#include <time.h>
#include <sys/os32.h>
```

```
//-----------------------------------------------------------
clock_t
clock (void)
{
  sysmsg_clock_t msg;
  msg.ret = 0;
  sys (SYS_CLOCK, &msg, (sizeof msg));
  return (msg.ret);
}
```

95.29.3 lib/time/gmtime.c

«

Si veda la sezione 88.15.

```
#include <time.h>
//-----------------------------------------------------------
static int leap_year (int year);
//-----------------------------------------------------------
struct tm *
gmtime (const time_t * timer)
{
  static struct tm tms;
  int loop;
  unsigned int remainder;
  unsigned int days;
  //
  // Check argument.
  //
  if (timer == NULL)
    {
      return (NULL);
    }
  //
  // Days since epoch. There are 86400 seconds per
  // day.
  // At the moment, the field 'tm_yday' will contain
  // all days since epoch.
```

```
    //
    days = *timer / 86400L;
    remainder = *timer % 86400L;
    //
    // Minutes, after full days.
    //
    tms.tm_min = remainder / 60U;
    //
    // Seconds, after full minutes.
    //
    tms.tm_sec = remainder % 60U;
    //
    // Hours, after full days.
    //
    tms.tm_hour = tms.tm_min / 60;
    //
    // Minutes, after full hours.
    //
    tms.tm_min = tms.tm_min % 60;
    //
    // Find the week day. Must remove some days to align
    // the
    // calculation. So: the week days of the first week
    // of 1970
    // are not valid! After 1970-01-04 calculations are
    // right.
    //
    tms.tm_wday = (days - 3) % 7;
    //
    // Find the year: the field 'tm_yday' will be
    // reduced to the days
    // of current year.
    //
    for (tms.tm_year = 1970; days > 0; tms.tm_year++)
      {
        if (leap_year (tms.tm_year))
          {
            if (days >= 366)
              {
                days -= 366;
                continue;
              }
            else
              {
                break;
              }
          }
        else
          {
            if (days >= 365)
              {
                days -= 365;
                continue;
              }
            else
              {
                break;
              }
          }
      }
    //
    // Day of the year.
    //
    tms.tm_yday = days + 1;
    //
    // Find the month.
    //
    tms.tm_mday = days + 1;
    //
    for (tms.tm_mon = 0, loop = 1; tms.tm_mon <= 12 && loop;)
      {
        tms.tm_mon++;
        //
        switch (tms.tm_mon)
        {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
          if (tms.tm_mday >= 31)
            {
              tms.tm_mday -= 31;
            }
          else
            {
              loop = 0;
            }
          break;
        case 4:
        case 6:
        case 9:
        case 11:
          if (tms.tm_mday >= 30)
            {
              tms.tm_mday -= 30;
            }
          else
            {
              loop = 0;
            }
          break;
        case 2:
          if (leap_year (tms.tm_year))
            {
              if (tms.tm_mday >= 29)
                {
                  tms.tm_mday -= 29;
                }
              else
                {
                  loop = 0;
                }
            }
          else
            {
              if (tms.tm_mday >= 28)
                {
                  tms.tm_mday -= 28;
                }
              else
                {
                  loop = 0;
                }
            }
          break;
        }
    }
  //
  // No check for day light saving time.
  //
  tms.tm_isdst = 0;
  //
  // Return.
  //
  return (&tms);
}

//----------------------------------------------------------
static int
leap_year (int year)
{
  if ((year % 4) == 0)
    {
      if ((year % 100) == 0)
        {
          if ((year % 400) == 0)
            {
              return (1);
            }
          else
            {
              return (0);
            }
        }
      else
        {
          return (1);
        }
    }
  else
    {
      return (0);
    }
}
```

95.29.4 lib/time/mktime.c

Si veda la sezione 88.15.

```
#include <time.h>
#include <string.h>
#include <stdio.h>
//-----------------------------------------------------------
static int leap_year (int year);
//-----------------------------------------------------------
time_t
mktime (const struct tm *timeptr)
{
  time_t timer_total;
  time_t timer_aux;
  int days;
  int month;
```

```
    int year;
    //
    // From seconds to days.
    //
    timer_total = timeptr->tm_sec;
    //
    timer_aux = timeptr->tm_min;
    timer_aux *= 60;
    timer_total += timer_aux;
    //
    timer_aux = timeptr->tm_hour;
    timer_aux *= (60 * 60);
    timer_total += timer_aux;
    //
    timer_aux = timeptr->tm_mday;
    timer_aux *= 24;
    timer_aux *= (60 * 60);
    timer_total += timer_aux;
    //
    // Month: add the days of months.
    // Will scan the months, from the first, but before
    // the
    // months of the value inside field 'tm_mon'.
    //
    for (month = 1, days = 0; month < timeptr->tm_mon;
         month++)
      {
        switch (month)
          {
          case 1:
          case 3:
          case 5:
          case 7:
          case 8:
          case 10:
            //
            // There is no December, because the scan
            // can go up to
            // the month before the value inside field
            // `tm_mon'.
            //
            days += 31;
            break;
          case 4:
          case 6:
          case 9:
          case 11:
            days += 30;
            break;
          case 2:
            if (leap_year (timeptr->tm_year))
              {
                days += 29;
              }
            else
              {
                days += 28;
              }
            break;
          }
      }
    //
    timer_aux = days;
    timer_aux *= 24;
    timer_aux *= (60 * 60);
    timer_total += timer_aux;
    //
    // Year. The work is similar to the one of months:
    // days of
    // years are counted, up to the year before the one
    // reported
    // by the field `tm_year'.
    //
    for (year = 1970, days = 0; year < timeptr->tm_year;
       year++)
    {
      if (leap_year (year))
        {
          days += 366;
        }
      else
        {
          days += 365;
        }
    }
  //
  // After all, must subtract a day from the total.
  //
  days--;
  //
  timer_aux = days;
  timer_aux *= 24;
  timer_aux *= (60 * 60);
  timer_total += timer_aux;
  //
  // That's all.
  //
  return (timer_total);
}

//----------------------------------------------------------
int
leap_year (int year)
{
  if ((year % 4) == 0)
    {
      if ((year % 100) == 0)
        {
          if ((year % 400) == 0)
            {
              return (1);
            }
          else
            {
              return (0);
            }
        }
      else
        {
          return (1);
        }
    }
  else
    {
      return (0);
    }
}
```

95.29.5 lib/time/stime.c

Si veda la sezione 87.59.

```
#include <time.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
int
stime (time_t * timer)
{
  sysmsg_stime_t msg;
  //
  if (timer == NULL)
    {
      errset (EINVAL);
      return (-1);
    }
  //
  msg.timer = *timer;
  msg.ret = 0;
  sys (SYS_STIME, &msg, (sizeof msg));
  return (msg.ret);
}
```

95.29.6 lib/time/time.c

Si veda la sezione 87.59.

```
#include <time.h>
#include <sys/os32.h>
//-----------------------------------------------------------
time_t
time (time_t * timer)
{
  sysmsg_time_t msg;
  msg.ret = ((time_t) 0);
  sys (SYS_TIME, &msg, (sizeof msg));
  if (timer != NULL)
    {
      *timer = msg.ret;
    }
  return (msg.ret);
}
```

95.30 os32: «lib/unistd.h»

Si veda la sezione 91.3.

```
#ifndef _UNISTD_H
#define _UNISTD_H        1
//-----------------------------------------------------------
#include <sys/stat.h>
#include <sys/types.h>  // size_t, ssize_t, uid_t,
                        // gid_t, off_t, pid_t
#include <inttypes.h>   // intptr_t
```

```
#include <SEEK.h>       // SEEK_CUR, SEEK_SET,
                        // SEEK_END
//----------------------------------------------------------------------
typedef unsigned int useconds_t;        // This type
                                        // should be
                                        // used for
                                        // the
                                 // obsolete function
                                 // 'usleep()', that
                                 // is only
                                 // implemented inside
                                 // the
                                 // kernel, as
                                 // 'k_usleep()', for
                                 // the
                                 // drivers
                                 // management.
//----------------------------------------------------------------------
extern char **environ;  // Variable 'environ' is used
                        // by functions like
                        // 'execv()' in replacement
                        // for 'envp[][]'.
//----------------------------------------------------------------------
extern char *optarg;    // Used by 'optarg()'.
extern int optind;      //
extern int opterr;      //
extern int optopt;      //
//----------------------------------------------------------------------
#define STDIN_FILENO    0       //
#define STDOUT_FILENO   1       // Standard file
                                // descriptors.
#define STDERR_FILENO   2       //
//----------------------------------------------------------------------
#define R_OK            4       // Read permission.
#define W_OK            2       // Write permission.
#define X_OK            1       // Execute or traverse
                                // permission.
```

```
#define F_OK             0       // File exists.
//-----------------------------------------------------------

int access (const char *path, int mode);
int brk (void *address);
int chdir (const char *path);
int chown (const char *path, uid_t uid, gid_t gid);
int close (int fdn);
int dup (int fdn_old);
int dup2 (int fdn_old, int fdn_new);
int execl (const char *path, char *arg, ...);
int execle (const char *path, char *arg, ...);
int execlp (const char *path, char *arg, ...);
int execv (const char *path, char *const argv[]);
int execve (const char *path, char *const argv[],
            char *const envp[]);
int execvp (const char *path, char *const argv[]);
void _exit (int status);
int fchown (int fdn, uid_t uid, gid_t gid);
pid_t fork (void);
char *getcwd (char *buffer, size_t size);
gid_t getegid (void);
uid_t geteuid (void);
gid_t getgid (void);
int getopt (int argc, char *const argv[],
            const char *optstring);
pid_t getpgrp (void);
pid_t getppid (void);
pid_t getpid (void);
uid_t getuid (void);
int isatty (int fdn);
int link (const char *path_old, const char *path_new);
off_t lseek (int fdn, off_t offset, int whence);
#define      nice(n)      (0)
int pipe (int pipefd[2]);
ssize_t read (int fdn, void *buffer, size_t count);
#define      readlink(p,b,s)  ((ssize_t) -1)
```

```
int rmdir (const char *path);
void *sbrk (intptr_t increment);
int setegid (gid_t gid);
int seteuid (uid_t uid);
int setgid (gid_t gid);
int setpgrp (void);
int setuid (uid_t uid);
unsigned int sleep (unsigned int s);
#define      sync() /**/
char *ttyname (int fdn);
int unlink (const char *path);
ssize_t write (int fdn, const void *buffer, size_t count);
//----------------------------------------------------------
#endif
```

95.30.1 lib/unistd/_exit.c

Si veda la sezione 87.2.

```
#include <unistd.h>
#include <sys/os32.h>
//----------------------------------------------------------
void
_exit (int status)
{
  sysmsg_exit_t msg;
  //
  // Only the low eight bit are returned.
  //
  msg.status = (status & 0xFF);
  //
  //
  //
  sys (SYS_EXIT, &msg, (sizeof msg));
  //
  // Should not return from system call, but if it
  // does, loop
  // forever:
```

```
  //
  while (1);
}
```

95.30.2 lib/unistd/access.c

Si veda la sezione 88.4.

```
#include <unistd.h>
#include <sys/stat.h>
#include <errno.h>
//----------------------------------------------------------
int
access (const char *path, int mode)
{
  struct stat st;
  int status;
  uid_t euid;
  //
  status = stat (path, &st);
  if (status != 0)
    {
      return (-1);
    }
  //
  // File exists?
  //
  if (mode == F_OK)
    {
      return (0);
    }
  //
  // Some access permissions are requested: get
  // effective user id.
  //
  euid = geteuid ();
  //
  // Check owner access permissions.
  //
  if (st.st_uid == euid
      && ((st.st_mode & S_IRWXU) == (mode << 6)))
    {
      return (0);
    }
  //
  // Check others access permissions.
  //
  if ((st.st_mode & S_IRWXO) == (mode))
    {
      return (0);
    }
  //
  // Otherwise there are no access permissions.
  //
  errset (EACCES);      // Permission denied.
  return (-1);
}
```

95.30.3 lib/unistd/brk.c

Si veda la sezione 87.5.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
brk (void *address)
{
  sysmsg_brk_t msg;
  //
  if (address == NULL)
```

```
    {
      errset (EINVAL);
      return (-1);
    }
  //
  msg.address = address;
  //
  sys (SYS_BRK, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.4 lib/unistd/chdir.c

«

Si veda la sezione 87.6.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
chdir (const char *path)
{
  sysmsg_chdir_t msg;
  //
  msg.path = path;
  msg.ret = 0;
  msg.errno = 0;
  //
  sys (SYS_CHDIR, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.5 lib/unistd/chown.c

Si veda la sezione 87.8.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
chown (const char *path, uid_t uid, gid_t gid)
{
  sysmsg_chown_t msg;
  //
  msg.path = path;
  msg.uid = uid;
  msg.gid = gid;
  //
  sys (SYS_CHOWN, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.6 lib/unistd/close.c

Si veda la sezione 87.10.

```
#include <unistd.h>
#include <errno.h>
#include <sys/os32.h>
#include <string.h>
//----------------------------------------------------------
int
close (int fdn)
{
  sysmsg_close_t msg;
  msg.fdn = fdn;
  //
  while (1)
    {
      sys (SYS_CLOSE, &msg, (sizeof msg));
      if (msg.ret < 0 && (msg.errno == EINPROGRESS
                          || msg.errno == EALREADY))
        {
          continue;
        }
      //
      break;
    }
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.7 lib/unistd/dup.c

Si veda la sezione 87.12.

```
#include <unistd.h>
#include <sys/os32.h>
```

```
#include <string.h>
#include <errno.h>
//----------------------------------------------------------
int
dup (int fdn_old)
{
  sysmsg_dup_t msg;
  //
  msg.fdn_old = fdn_old;
  //
  sys (SYS_DUP, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.8 lib/unistd/dup2.c

Si veda la sezione 87.12.

```
#include <unistd.h>
#include <sys/os32.h>
#include <string.h>
#include <errno.h>
//----------------------------------------------------------
int
dup2 (int fdn_old, int fdn_new)
{
  sysmsg_dup2_t msg;
  //
  msg.fdn_old = fdn_old;
  msg.fdn_new = fdn_new;
  //
  sys (SYS_DUP2, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.9 lib/unistd/environ.c

Si veda la sezione 91.1.

```
#include <unistd.h>
//----------------------------------------------------------------------
char **environ;
```

95.30.10 lib/unistd/execl.c

Si veda la sezione 88.21.

```
#include <unistd.h>
#include <limits.h>
#include <stdarg.h>
#include <stddef.h>
//----------------------------------------------------------------------
int
execl (const char *path, char *arg, ...)
{
  int argc;
  char *arg_next;
  char *argv[ARG_MAX / 2];
  //
  va_list ap;
  va_start (ap, arg);
  //
  arg_next = arg;
  //
  for (argc = 0; argc < ARG_MAX / 2; argc++)
    {
      argv[argc] = arg_next;
      if (argv[argc] == NULL)
        {
          break;              // End of arguments.
        }
      arg_next = va_arg (ap, char *);
    }
  //
  return (execve (path, argv, environ));          // [1]
}

//
// The variable 'environ' is declared as
// 'char **environ' and is
// included from <unistd.h>.
//
```

95.30.11 lib/unistd/execle.c

«

Si veda la sezione 88.21.

```
#include <unistd.h>
#include <limits.h>
#include <stdarg.h>
#include <stddef.h>
//----------------------------------------------------------
int
execle (const char *path, char *arg, ...)
{
  int argc;
  char *arg_next;
  char *argv[ARG_MAX / 2];
  char **envp;
  //
  va_list ap;
  va_start (ap, arg);
  //
  arg_next = arg;
  //
  for (argc = 0; argc < ARG_MAX / 2; argc++)
    {
      argv[argc] = arg_next;
      if (argv[argc] == NULL)
        {
          break;        // End of arguments.
        }
      arg_next = va_arg (ap, char *);
    }
  //
  envp = va_arg (ap, char **);
  //
  return (execve (path, argv, envp));
}
```

95.30.12 lib/unistd/execlp.c

Si veda la sezione 88.21.

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/os32.h>
//----------------------------------------------------------
int
execlp (const char *path, char *arg, ...)
{
  int argc;
  char *arg_next;
  char *argv[ARG_MAX / 2];
  char command[PATH_MAX];
  int status;
  //
  va_list ap;
  va_start (ap, arg);
  //
  arg_next = arg;
  //
  for (argc = 0; argc < ARG_MAX / 2; argc++)
    {
      argv[argc] = arg_next;
      if (argv[argc] == NULL)
        {
          break;        // End of arguments.
        }
      arg_next = va_arg (ap, char *);
    }
  //
  // Get a full command path if necessary.
  //
  status = namep (path, command, (size_t) PATH_MAX);
  if (status != 0)
    {
      //
      // Variable 'errno' is already set by
      // 'commandp()'.
      //
      return (-1);
    }
  //
  // Return calling 'execve()'
  //
  return (execve (command, argv, environ));    // [1]
}

//
// The variable 'environ' is declared as
// 'char **environ' and is
// included from <unistd.h>.
//
```

95.30.13 lib/unistd/execv.c

Si veda la sezione 88.21.

```
#include <unistd.h>
//----------------------------------------------------------
int
execv (const char *path, char *const argv[])
{
  return (execve (path, argv, environ));        // [1]
}

//
// The variable `environ' is declared as
// `char **environ' and is
// included from <unistd.h>.
//
```

95.30.14 lib/unistd/execve.c

Si veda la sezione 87.14.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <string.h>
//----------------------------------------------------------
int
execve (const char *path, char *const argv[],
        char *const envp[])
{
  sysmsg_exec_t msg;
  size_t size;
  size_t arg_size;
  int argc;
  size_t env_size;
  int envc;
  char *arg_data = msg.arg_data;
  char *env_data = msg.env_data;
  //
  msg.path = path;
  msg.ret = 0;
  msg.errno = 0;
  //
  // Copy 'argv[]' inside a the message buffer
  // 'msg.arg_data',
  // separating each string with a null character and
  // counting the
  // number of strings inside 'argc'.
  //
  for (argc = 0, arg_size = 0, size = 0;
       argv != NULL &&
       argc < (ARG_MAX / 16) &&
       arg_size < ARG_MAX / 2 &&
       argv[argc] != NULL; argc++, arg_size += size)
    {
      size = strlen (argv[argc]);
      size++;   // Count also the final null
      // character.
      if (size > (ARG_MAX / 2 - arg_size))
        {
          errset (E2BIG);       // Argument list too
          // long.
          return (-1);
        }
      strncpy (arg_data, argv[argc], size);
      arg_data += size;
    }
  msg.argc = argc;
  //
  // Copy 'envp[]' inside a the message buffer
  // 'msg.env_data',
  // separating each string with a null character and
  // counting the
  // number of strings inside `envc'.
  //
  for (envc = 0, env_size = 0, size = 0;
       envp != NULL &&
       envc < (ARG_MAX / 16) &&
       env_size < ARG_MAX / 2 &&
       envp[envc] != NULL; envc++, env_size += size)
    {
      size = strlen (envp[envc]);
      size++;   // Count also the final null
      // character.
      if (size > (ARG_MAX / 2 - env_size))
        {
          errset (E2BIG);       // Argument list too
          // long.
          return (-1);
        }
      strncpy (env_data, envp[envc], size);
      env_data += size;
    }
  msg.envc = envc;
  //
  // System call.
  //
  sys (SYS_EXEC, &msg, (sizeof msg));
  //
  // Should not return, but if it does, then there is
  // an error.
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.15 lib/unistd/execvp.c

Si veda la sezione 88.21.

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/os32.h>
//----------------------------------------------------------
int
execvp (const char *path, char *const argv[])
{
  char command[PATH_MAX];
  int status;
  //
  // Get a full command path if necessary.
  //
  status = namep (path, command, (size_t) PATH_MAX);
  if (status != 0)
    {
      //
      // Variable 'errno' is already set by 'namep()'.
      //
      return (-1);
    }
  //
  // Return calling 'execve()'
  //
  return (execve (command, argv, environ));     // [1]
}

//
// The variable 'environ' is declared as
// 'char **environ' and is
// included from <unistd.h>.
//
```

95.30.16 lib/unistd/fchdir.c

Si veda la sezione 87.6.

```
#include <unistd.h>
#include <errno.h>
//----------------------------------------------------------
int
fchdir (int fdn)
{
  //
  // os32 requires to keep track of the path for the
  // current working
  // directory. The standard function 'fchdir()' is
  // not applicable.
  //
  errset (E_NOT_IMPLEMENTED);
  return (-1);
}
```

95.30.17 lib/unistd/fchown.c

Si veda la sezione 87.8.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
fchown (int fdn, uid_t uid, gid_t gid)
{
  sysmsg_fchown_t msg;
  //
  msg.fdn = fdn;
  msg.uid = uid;
  msg.gid = gid;
```

```
    //
    sys (SYS_FCHOWN, &msg, (sizeof msg));
    //
    errno = msg.errno;
    errln = msg.errln;
    strncpy (errfn, msg.errfn, PATH_MAX);
    return (msg.ret);
}
```

95.30.18 lib/unistd/fork.c

Si veda la sezione 87.19.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
//----------------------------------------------------------
pid_t
fork (void)
{
    sysmsg_fork_t msg;
    //
    // Set the return value for the child process.
    //
    msg.ret = 0;
    //
    // Do the system call.
    //
    sys (SYS_FORK, &msg, (sizeof msg));
    //
    // If the system call has successfully generated a
    // copy of
    // the original process, the following code is
    // executed from
    // the parent and the child. But the child has the
```

```
  // `msg'
  // structure untouched, while the parent has, at
  // least, the
  // pid number inside `msg.ret'.
  // If the system call fails, there is no child, and
  // the
  // parent finds the return value equal to -1, with
  // an
  // error number.
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.19 lib/unistd/getcwd.c

Si veda la sezione 87.21.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <stddef.h>
#include <string.h>
//----------------------------------------------------------
char *
getcwd (char *buffer, size_t size)
{
  sysmsg_uarea_t msg;
  //
  // Check arguments: the buffer must be given.
  //
  if (buffer == NULL)
    {
      errset (EINVAL);
      return (NULL);
    }
  //
  msg.path_cwd = buffer;
  msg.path_cwd_size = size;
  //
  // Set the last buffer element to zero, for later
  // verification.
  //
  buffer[size - 1] = 0;
  //
  // Just get the user area data.
  //
  sys (SYS_UAREA, &msg, (sizeof msg));
  //
  // Check that the path is still correctly
  // terminated. If it isn't,
  // the path is longer than the buffer size, because
  // the last null
  // character was overwritten.
  //
  if (buffer[size - 1] != 0)
    {
      errset (ERANGE);
      return (NULL);
    }
  //
  // Everything is fine.
  //
  return (buffer);
}
```

95.30.20 lib/unistd/getegid.c

Si veda la sezione 87.22.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
gid_t
getegid (void)
{
  sysmsg_uarea_t msg;
  msg.path_cwd = NULL;
  msg.path_cwd_size = 0;
  sys (SYS_UAREA, &msg, (sizeof msg));
  return (msg.egid);
}
```

95.30.21 lib/unistd/geteuid.c

Si veda la sezione 87.27.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
uid_t
geteuid (void)
{
  sysmsg_uarea_t msg;
  msg.path_cwd = NULL;
  msg.path_cwd_size = 0;
  sys (SYS_UAREA, &msg, (sizeof msg));
  return (msg.euid);
}
```

95.30.22 lib/unistd/getgid.c

Si veda la sezione 87.22.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
gid_t
getgid (void)
{
  sysmsg_uarea_t msg;
  msg.path_cwd = NULL;
  msg.path_cwd_size = 0;
  sys (SYS_UAREA, &msg, (sizeof msg));
  return (msg.gid);
}
```

95.30.23 lib/unistd/getopt.c

Si veda la sezione 88.56.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
char *optarg;
int optind = 1;
int opterr = 1;
int optopt = 0;
//----------------------------------------------------------
static void getopt_no_argument (int opt);
//----------------------------------------------------------
int
getopt (int argc, char *const argv[], const char *optstring)
{
    static int o = 0;     // Index to scan grouped
    // options.
    int s;         // Index to scan ‘optstring’
    int opt;       // Current option letter.
    int flag_argument;    // If there should be an
    // argument.
    //
    // Entering the function, ‘flag_argument’ is zero.
    // Just to make
    // it clear:
    //
    flag_argument = 0;
    //
    // Scan ‘argv[]’ elements, starting form the value
    // that ‘optind’
    // already have.
    //
    for (; optind < argc; optind++)
      {
        //
        // If an option is expected, some check must be
        // done at
        // the beginning.
        //
        if (!flag_argument)
          {
            //
            // Check if the scan is finished and
            // ‘optind’ should be kept
            // untouched:
            // ‘argv[optind]’ is a null pointer;
            // ‘argv[optind][0]’ is not the character
            // ‘-’;
            // ‘argv[optind]’ points to the string "-";
            // all ‘argv[]’ elements are parsed.
            //
            if (argv[optind] == NULL
              || argv[optind][0] != '-'
              || argv[optind][1] == 0 || optind >= argc)
            {
              return (-1);
            }
          //
          // Check if the scan is finished and
          // 'optind' is to be
          // incremented:
          // 'argv[optind]' points to the string "--".
          //
          if (argv[optind][0] == '-'
              && argv[optind][1] == '-'
              && argv[optind][2] == 0)
            {
              optind++;
              return (-1);
            }
        }
      //
      // Scan 'argv[optind]' using the static index
      // 'o'.
      //
      for (; o < strlen (argv[optind]); o++)
        {
          //
          // If there should be an option, index 'o'
          // should
          // start from 1, because 'argv[optind][0]'
          // must
          // be equal to '-'.
          //
          if (!flag_argument && (o == 0))
            {
              //
              // As there is no options, 'o' cannot
              // start
              // from zero, so a new loop is done.
              //
              continue;
            }
          //
          if (flag_argument)
            {
              //
              // There should be an argument, starting
              // from
              // `argv[optind][o]'.
              //
              if ((o == 0) && (argv[optind][o] == '-'))
                {
                  //
                  // `argv[optind][0]' is equal to
                  // '-', but there
                  // should be an argument instead:
                  // the argument
                  // is missing.
                  //
                  optarg = NULL;
                  //
                  if (optstring[0] == ':')
                    {
                      //
                      // As the option string starts
                      // with `:' the
                      // function must return `:'.
                      //
                      optopt = opt;
                      opt = ':';
                    }
                  else
                    {
                      //
                      // As the option string does not
                      // start with `:'
                      // the function must return `?'.
                      //
                      getopt_no_argument (opt);
                      optopt = opt;
                      opt = '?';
                    }
                  //
                  // `optind' is left untouched.
                  //
                }
              else
                {
                  //
                  // The argument is found: `optind'
                  // is to be
                  // incremented and `o' is reset.
                  //
                  optarg = &argv[optind][o];
                  optind++;
                  o = 0;
                }
              //
              // Return the option, or ':', or '?'.
              //
              return (opt);
            }
          else
            {
              //
              // It should be an option: `optstring[]'
              // must be
              // scanned.
              //
              opt = argv[optind][o];
              //
              for (s = 0, optopt = 0;
     s < strlen (optstring); s++)
  {
    //
    // If 'optsting[0]' is equal to ':',
    // index 's' must
    // start at 1.
    //
    if ((s == 0) && (optstring[0] == ':'))
      {
        continue;
      }
    //
    if (opt == optstring[s])
      {
        //
        if (optstring[s + 1] == ':')
          {
            //
            // There is an argument.
            //
            flag_argument = 1;
            break;
          }
        else
          {
            //
            // There is no argument.
            //
            o++;
            return (opt);
          }
      }
  }
//
if (s >= strlen (optstring))
  {
    //
```

```
                    // The 'optstring' scan is concluded
                    // with no
                    // match.
                    //
                    o++;
                    optopt = opt;
                    return ('?');
                  }
                //
                // Otherwise the loop was broken.
                //
              }
          }
        //
        // Check index 'o'.
        //
        if (o >= strlen (argv[optind]))
          {
            //
            // There are no more options or there is no
            // argument
            // inside current 'argv[optind]' string.
            // Index 'o' is
            // reset before the next loop.
            //
            o = 0;
          }
      }
    //
    // No more elements inside 'argv' or loop broken:
    // there might be a
    // missing argument.
    //
    if (flag_argument)
      {
        //
        // Missing option argument.
      //
      optarg = NULL;
      //
      if (optstring[0] == ':')
        {
          return (':');
        }
      else
        {
          getopt_no_argument (opt);
          return ('?');
        }
    }
  //
  return (-1);
}

//----------------------------------------------------------
static void
getopt_no_argument (int opt)
{
  if (opterr)
    {
      fprintf (stderr,
               "Missing argument for option `-%c'\n", opt);
    }
}
```

95.30.24 lib/unistd/getpgrp.c

Si veda la sezione 87.25.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
```

```
pid_t
getpgrp (void)
{
  sysmsg_uarea_t msg;
  msg.path_cwd = NULL;
  msg.path_cwd_size = 0;
  sys (SYS_UAREA, &msg, (sizeof msg));
  return (msg.pgrp);
}
```

95.30.25 lib/unistd/getpid.c

Si veda la sezione 87.25.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
//----------------------------------------------------------
pid_t
getpid (void)
{
  sysmsg_uarea_t msg;
  msg.path_cwd = NULL;
  msg.path_cwd_size = 0;
  sys (SYS_UAREA, &msg, (sizeof msg));
  return (msg.pid);
}
```

95.30.26 lib/unistd/getppid.c

Si veda la sezione 87.25.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
```

```
//------------------------------------------------------------
pid_t
getppid (void)
{
  sysmsg_uarea_t msg;
  msg.path_cwd = NULL;
  msg.path_cwd_size = 0;
  sys (SYS_UAREA, &msg, (sizeof msg));
  return (msg.ppid);
}
```

95.30.27 lib/unistd/getuid.c

Si veda la sezione 87.27.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
//------------------------------------------------------------
uid_t
getuid (void)
{
  sysmsg_uarea_t msg;
  msg.path_cwd = NULL;
  msg.path_cwd_size = 0;
  sys (SYS_UAREA, &msg, (sizeof msg));
  return (msg.uid);
}
```

95.30.28 lib/unistd/isatty.c

Si veda la sezione 88.69.

```
#include <sys/stat.h>
#include <sys/os32.h>
#include <unistd.h>
```

```
#include <sys/types.h>
#include <errno.h>
//----------------------------------------------------------
int
isatty (int fdn)
{
  struct stat file_status;
  //
  // Verify to have valid input data.
  //
  if (fdn < 0)
    {
      errset (EBADF);
      return (0);
    }
  //
  // Verify the standard input.
  //
  if (fstat (fdn, &file_status) == 0)
    {
      if (major (file_status.st_rdev) == DEV_CONSOLE_MAJOR)
        {
          return (1);   // Meaning it is ok!
        }
      if (major (file_status.st_rdev) == DEV_TTY_MAJOR)
        {
          return (1);   // Meaning it is ok!
        }
    }
  else
    {
      errset (errno);
      return (0);
    }
  //
  // If here, it is not a terminal of any kind.
  //
  errset (EINVAL);
  return (0);
}
```

95.30.29 lib/unistd/link.c

Si veda la sezione 87.30.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------------
int
link (const char *path_old, const char *path_new)
{
  sysmsg_link_t msg;
  //
  msg.path_old = path_old;
  msg.path_new = path_new;
  //
  sys (SYS_LINK, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.30 lib/unistd/lseek.c

Si veda la sezione 87.33.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
```

```
#include <errno.h>
#include <string.h>
//----------------------------------------------------------
off_t
lseek (int fdn, off_t offset, int whence)
{
  sysmsg_lseek_t msg;
  msg.fdn = fdn;
  msg.offset = offset;
  msg.whence = whence;
  sys (SYS_LSEEK, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.31 lib/unistd/pipe.c

Si veda la sezione 87.38.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
pipe (int pipefd[2])
{
  sysmsg_pipe_t msg;
  //
  if (pipefd == NULL)
    {
      errset (EINVAL);
      return (-1);
    }
  //
  sys (SYS_PIPE, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  //
  pipefd[0] = msg.pipefd[0];
  pipefd[1] = msg.pipefd[1];
  //
  return (msg.ret);
}
```

95.30.32 lib/unistd/read.c

Si veda la sezione 87.39.

```
#include <unistd.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
//----------------------------------------------------------
ssize_t
read (int fdn, void *buffer, size_t count)
{
  sysmsg_read_t msg;
  //
  // Reduce size of read if necessary.
  //
  if (count > BUFSIZ)
    {
      count = BUFSIZ;
    }
  //
  // Fill the message.
  //
```

```
    msg.fdn = fdn;
    msg.buffer = buffer;
    msg.count = count;
    msg.fl_flags = 0;     // Not necessary.
    msg.ret = 0;
    //
    // Repeat syscall, until something is received or
    // end of file is
    // reached.
    //
    while (1)
      {
        sys (SYS_READ, &msg, (sizeof msg));
        if (msg.ret == 0)
          {
            //
            // End of file.
            //
            break;
          }
        if (msg.ret < 0
            && (msg.errno == EAGAIN
                || msg.errno == EWOULDBLOCK))
          {
            //
            // No data at the moment.
            //
            if (msg.fl_flags & O_NONBLOCK)
              {
                //
                // Don't block.
                //
                break;
              }
            else
              {
                //
                // Keep trying.
                //
                continue;
              }
          }
        //
        // Otherwise, we have read something.
        //
        break;
      }
    //
    //
    //
    if (msg.ret < 0)
      {
        //
        // No valid read.
        //
        errno = msg.errno;
        errln = msg.errln;
        strncpy (errfn, msg.errfn, PATH_MAX);
        return (msg.ret);
      }
    //
    if (msg.ret > count)
      {
        //
        // A strange value was returned. Considering it
        // a read error.
        //
        errset (EIO);     // I/O error.
        return (-1);
      }
    //
    // A valid read: return.
    //
    return (msg.ret);
```

```
}
```

95.30.33 lib/unistd/rmdir.c

Si veda la sezione 87.41.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
rmdir (const char *path)
{
  sysmsg_stat_t msg_stat;
  sysmsg_unlink_t msg_unlink;
  //
  msg_stat.path = path;
  //
  sys (SYS_STAT, &msg_stat, (sizeof msg_stat));
  //
  if (msg_stat.ret != 0)
    {
      errno = msg_stat.errno;
      errln = msg_stat.errln;
      strncpy (errfn, msg_stat.errfn, PATH_MAX);
      return (msg_stat.ret);
    }
  //
  if (!S_ISDIR (msg_stat.stat.st_mode))
    {
      errset (ENOTDIR); // Not a directory.
      return (-1);
    }
  //
  msg_unlink.path = path;
  //
  sys (SYS_UNLINK, &msg_unlink, (sizeof msg_unlink));
  //
  errno = msg_unlink.errno;
  errln = msg_unlink.errln;
  strncpy (errfn, msg_unlink.errfn, PATH_MAX);
  return (msg_unlink.ret);
}
```

95.30.34 lib/unistd/sbrk.c

Si veda la sezione 87.5.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//-----------------------------------------------------------
void *
sbrk (intptr_t increment)
{
  sysmsg_sbrk_t msg_sbrk;
  //
  msg_sbrk.increment = increment;
  //
  sys (SYS_SBRK, &msg_sbrk, (sizeof msg_sbrk));
  //
  errno = msg_sbrk.errno;
  errln = msg_sbrk.errln;
  strncpy (errfn, msg_sbrk.errfn, PATH_MAX);
  return (msg_sbrk.ret);
}
```

95.30.35 lib/unistd/setegid.c

Si veda la sezione 87.48.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
//----------------------------------------------------------
int
setegid (gid_t gid)
{
  sysmsg_setegid_t msg;
  msg.ret = 0;
  msg.errno = 0;
  msg.egid = gid;
  sys (SYS_SETEGID, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.36 lib/unistd/seteuid.c

Si veda la sezione 87.51.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
//----------------------------------------------------------
int
seteuid (uid_t uid)
{
  sysmsg_seteuid_t msg;
  msg.ret = 0;
  msg.errno = 0;
  msg.euid = uid;
  sys (SYS_SETEGID, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.37 lib/unistd/setgid.c

Si veda la sezione 87.48.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
//----------------------------------------------------------
int
setgid (gid_t gid)
{
  sysmsg_setgid_t msg;
  msg.ret = 0;
  msg.errno = 0;
  msg.egid = gid;
  sys (SYS_SETGID, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.38 lib/unistd/setpgrp.c

«

Si veda la sezione 87.50.

```
#include <unistd.h>
#include <sys/os32.h>
#include <stddef.h>
//----------------------------------------------------------------------
int
setpgrp (void)
{
  sys (SYS_PGRP, NULL, (size_t) 0);
  return (0);
}
```

95.30.39 lib/unistd/setuid.c

«

Si veda la sezione 87.51.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
//----------------------------------------------------------------------
int
setuid (uid_t uid)
{
  sysmsg_setuid_t msg;
  msg.ret = 0;
  msg.errno = 0;
  msg.euid = uid;
  sys (SYS_SETUID, &msg, (sizeof msg));
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.40 lib/unistd/sleep.c

«

Si veda la sezione 87.53.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os32.h>
#include <errno.h>
#include <time.h>
//----------------------------------------------------------
unsigned int
sleep (unsigned int seconds)
{
  sysmsg_sleep_t msg;
  time_t start;
  time_t end;
  int slept;
  //
  if (seconds == 0)
    {
      return (0);
    }
  //
  msg.events = WAKEUP_EVENT_TIMER;
  msg.seconds = seconds;
  sys (SYS_SLEEP, &msg, (sizeof msg));
  start = msg.ret;
  end = time (NULL);
  slept = end - msg.ret;
  //
  if (slept < 0)
    {
      return (seconds);
    }
  else if (slept < seconds)
    {
      return (seconds - slept);
    }
  else
    {
      return (0);
    }
}
```

95.30.41 lib/unistd/ttyname.c

«

Si veda la sezione 88.133.

```
#include <sys/os32.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
char *
ttyname (int fdn)
{
  dev_t dev_minor;
  struct stat file_status;
  static char name[PATH_MAX];
  //
  // Verify to have valid input data.
  //
  if (fdn < 0)
    {
      errset (EBADF);
      return (NULL);
    }
  //
  // Verify the file descriptor.
  //
  if (fstat (fdn, &file_status) == 0)
    {
      if (major (file_status.st_rdev) == DEV_CONSOLE_MAJOR)
    {
     dev_minor = minor (file_status.st_rdev);
     //
     // If minor is equal to 0xFF, it is
     // `/dev/console'.
     //
     if (dev_minor < 0xFF)
      {
       sprintf (name, "/dev/console%i", dev_minor);
      }
     else
      {
       strcpy (name, "/dev/console");
      }
     return (name);
    }
   else if (file_status.st_rdev == DEV_TTY)
    {
     strcpy (name, "/dev/tty");
     return (name);
    }
   else
    {
     errset (ENOTTY);
     return (NULL);
    }
  }
 else
  {
   errset (errno);
   return (NULL);
  }
}
```

95.30.42 lib/unistd/unlink.c

Si veda la sezione 87.62.

```
#include <unistd.h>
#include <string.h>
#include <sys/os32.h>
#include <errno.h>
#include <limits.h>
//----------------------------------------------------------
int
unlink (const char *path)
{
  sysmsg_unlink_t msg;
  //
  msg.path = path;
  //
  sys (SYS_UNLINK, &msg, (sizeof msg));
  //
  errno = msg.errno;
  errln = msg.errln;
  strncpy (errfn, msg.errfn, PATH_MAX);
  return (msg.ret);
}
```

95.30.43 lib/unistd/write.c

Si veda la sezione 87.64.

```
#include <unistd.h>
#include <sys/os32.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
//----------------------------------------------------------
ssize_t
write (int fdn, const void *buffer, size_t count)
{
    sysmsg_write_t msg;
    //
    // Reduce size of write if necessary.
    //
    if (count > BUFSIZ)
      {
        count = BUFSIZ;
      }
    //
    // Fill the message.
    //
    msg.fdn = fdn;
    msg.buffer = buffer;
    msg.count = count;
    //
    // Syscall.
    //
    sys (SYS_WRITE, &msg, (sizeof msg));
    //
    // Check result and return.
    //
    if (msg.ret < 0)
      {
        //
        // No valid write.
        //
        errno = msg.errno;
        errln = msg.errln;
        strncpy (errfn, msg.errfn, PATH_MAX);
        return (msg.ret);
      }
    //
    if (msg.ret > count)
      {
        //
        // A strange value was returned. Considering it
        // a read error.
      //
      errset (EIO);     // I/O error.
      return (-1);
    }
  //
  // A valid write return.
  //
  return (msg.ret);
}
```

95.31 os32: «lib/utime.h»

Si veda la sezione 91.3.

```
#ifndef _UTIME_H
#define _UTIME_H        1
//-----------------------------------------------------------
#include <restrict.h>
#include <sys/types.h>  // time_t
//-----------------------------------------------------------
struct utimbuf
{
  time_t actime;
  time_t modtime;
};
//-----------------------------------------------------------
int utime (const char *path, const struct utimbuf *times);
//-----------------------------------------------------------

#endif
```

95.31.1 lib/utime/utime.c

«

Si veda la sezione 91.3.

```
#include <utime.h>
#include <errno.h>
//----------------------------------------------------------
int
utime (const char *path, const struct utimbuf *times)
{
  //
  // Currently not implemented.
  //
  return (0);
}
```