UNIX nasce negli elaboratori PDP-11. La diffusione e lo studio del sistema operativo ha favorito la fortuna di questa linea di elaboratori, tanto che esiste ancora un discreto interesse verso la preservazione dell'hardware del PDP-11 e delle architetture derivate.

Negli anni 1970 UNIX è un sistema operativo di ricerca, dal quale emerge la filosofia della condivisione della conoscenza informatica. Il desiderio di preservare il PDP-11 va di pari passo con quello di conservare, per ciò che è possibile, quanto resta di tali versioni di Unix, assieme alle varianti ed estensioni dell'università di Berkeley, dello stesso periodo.

La proprietà dei diritti sul codice UNIX originale si è trasferita più volte. Ogni «proprietario» ha gestito --o preteso di gestire-- questi diritti a propria discrezione; in particolare si ricorda nel 1994 la controversia tra Novell e l'università di Berkeley e nel 2003 quella tra SCO Group e IBM. Ma nel 2002 va ricordato un evento importante: Caldera (successivamente divenuta SCO Group), la quale in quel momento ne aveva i diritti, rilascia le prime edizioni di UNIX con una licenza simile a quella di BSD. Purtroppo, nel momento in cui SCO Group ha iniziato la causa contro IBM, questa licenza è scomparsa dal sito originale, ma continua a essere conservata e pubblicata dagli amatori del vecchio UNIX. Si veda a questo proposito:

- Caldera, *Dear UNIX® enthusiast*, 2002
  http://minnie.tuhs.org/Archive/Caldera-license.pdf
- Dion L. Johnson II, *Liberal license for ancient UNIX sources*, 2002
  http://www.lemis.com/grog/UNIX/
  http://www.lemis.com/grog/UNIX/ancient-source-all.pdf

Figura u193.1. La licenza, in stile «BSD», per le prime versioni di UNIX. Il file originale, ottenuto da *http://minnie.tuhs.org/Archive/* , il giorno 3 febbraio 2007, dovrebbe essere disponibile anche presso *allegati/Caldera-license.pdf* .

240 West Center Street
Orem, Utah 84057
801-765-4999 Fax 801-765-4481

January 23, 2002

Dear UNIX® enthusiasts,

Caldera International, Inc. hereby grants a fee free license that includes the rights use, modify and distribute this named source code, including creating derived binary products created from the source code. The source code for which Caldera International, Inc. grants rights are limited to the following UNIX Operating Systems that operate on the 16-Bit PDP-11 CPU and early versions of the 32-Bit UNIX Operating System, with specific exclusion of UNIX System III and UNIX System V and successor operating systems:

32-bit 32V UNIX
16 bit UNIX Versions 1, 2, 3, 4, 5, 6, 7

Caldera International, Inc. makes no guarantees or commitments that any source code is available from Caldera International, Inc.

The following copyright notice applies to the source code files for which this license is granted.

Copyright(C) Caldera International Inc. 2001-2002. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code and documentation must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed or owned by Caldera International, Inc.

Neither the name of Caldera International, Inc. nor the names of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

USE OF THE SOFTWARE PROVIDED FOR UNDER THIS LICENSE BY CALDERA INTERNATIONAL, INC. AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL CALDERA INTERNATIONAL, INC. BE LIABLE FOR ANY DIRECT, INDIRECT INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Very truly yours,

/signed/ Bill Broderick

Bill Broderick
Director, Licensing Services

* UNIX is a registered trademark of The Open Group in the US and other countries.

Caratteristiche dei primi sistemi UNIX e BSD

Le prime versioni di UNIX e di BSD (quelle per il PDP-11) hanno delle caratteristiche comuni, anche se non si può avere la certezza che siano sempre perfettamente uniformi. Queste caratteristiche vengono riassunte in questo capitolo.

Partizioni

Un disco utilizzato da una distribuzione BSD tradizionale è sempre diviso in partizioni, mentre con UNIX queste non ci sono, in quanto lo spazio finale, dopo la conclusione del file system viene usato per lo scambio della memoria virtuale.

Le partizioni BSD sono identificate da «etichette», definite *disklabel*. I nomi di queste etichette sono dati da una lettera alfabetica seguita da due punti: «***x***:». Una distribuzione BSD tradizionale può gestire per ogni disco un massimo di otto partizioni e le prime hanno generalmente un ruolo prestabilito.

Tabella u193.2. Utilizzo normale delle partizioni nei sistemi BSD tradizionali.

Etichetta	Utilizzo
`a:`	Partizione principale.
`b:`	Partizione di scambio per la memoria virtuale.
`c:`	Spazio complessivo di tutto il disco.

Etichetta	Utilizzo

`d:`
`e:`
`f:`
`g:`
`h:`
	Partizioni disponibili.

Sempre per quanto riguarda BSD, la partizione identificata dall'etichetta '**`c:`**' serve generalmente a delimitare lo spazio complessivo del disco. Si osservi che non è possibile collocare una partizione per lo scambio della memoria virtuale in una partizione diversa dalla seconda; inoltre, tra i file di dispositivo non ne è previsto uno che rappresenti il disco complessivo: da ciò deriva la necessità di avere l'etichetta '**`c:`**'.

Dispositivi

«

Le distribuzioni UNIX e BSD tradizionali classificano i dispositivi di memorizzazione, che nel PDP-11 potevano essere molto diversi, in gruppi omogenei, in base all'organizzazione del codice necessario per accedervi. Questi gruppi hanno delle sigle che è necessario conoscere.

Tabella u193.3. Sigle e file di dispositivo usati per le unità di memorizzazione. Le associazioni sono indicative.

unità di controllo	modello	tipo	BSD	
RK11	RK05	cartridge	rk	/dev/**rk**0a /dev/r**rk**0a
RK611	RK06 RK07		hk	/dev/**hk**0a /dev/r**hk**0a
RL11 RLV12	RL01 RL02		rl	/dev/**rl**0a /dev/r**rl**0a
RP11	RP01 RP02 RP03	pack	rp	/dev/**rp**0a /dev/r**rp**0a
Massbus: RH70 RH11	RP04 RP05 RP06	pack	xp	/dev/**xp**0a /dev/r**xp**0a
	RP07	fixed		
	RM02 RM03 RM05	pack		
	RM80	fixed		
MSCP: RQDX3	RA60	pack	ra	/dev/**ra**0a /dev/r**ra**0a
	RA70 RA71 RA72 RA73 RA80 RA81 RA82 RA90 RA92	fixed		
	RD51 RD52 RD53 RD54 RD31 RD32	fixed		
	RX33	fd 5,25 in		
	RX50	fd 5,25 in		
RX11 RX211	RX01 RX02	fd 8 in		

unità di controllo	modello	tipo	BSD	
MSCP: RQDX3	RA60	pack	ra	/dev/**ra**0a /dev/r**ra**0a
	RA70 RA71 RA72 RA73 RA80 RA81 RA82 RA90 RA92	fixed		
	RD51 RD52 RD53 RD54 RD31 RD32	fixed		
	RX33	fd 5,25 in		
	RX50	fd 5,25 in		
RX11 RX211	RX01 RX02	fd 8 in		

TM02 TM03	TE16 TU45 TU77	tape	ht	
	TE10 TM11		tm	
	TS11 TSV05		ts	

«

I file di dispositivo delle unità di memorizzazione a disco sono classificate in base al gruppo di dispositivi di cui fanno parte; inoltre, possono essere disponibili a coppie: una versione a blocchi e un'altra a caratteri.

Nello UNIX di ricerca, la struttura del nome dei file di dispositivo per le unità a disco segue una regola abbastanza semplice:

```
[r]hhn
```

La lettera «r» iniziale, se appare indica un dispositivo a caratteri e sta per *raw*. I dispositivi di questo tipo (a caratteri), se previsti, si usano solo nella fase di inizializzazione e creazione dei file system, oltre che nella copia brutale dell'immagine della partizione.

Le due lettere successive, che nel modello appaiono come ***hh***, richiamano il nome del tipo di dispositivo, come annotato nella tabella della sezione precedente. Per esempio, il file '`/dev/rk0`' è un dispositivo a blocchi per l'accesso al primo disco di tipo '**`rk`**' (un disco RK05). Nello stesso modo, il file '`/dev/rrk0`' è il dispositivo a caratteri (*raw*) dello stesso disco.

Nel caso di BSD, la forma del nome di questi file di dispositivo rimane la stessa, con l'aggiunta della lettera della partizione. Per esempio, il file '`/dev/ra0a`' è un dispositivo a blocchi per l'accesso diretto alla prima partizione del primo disco di tipo '**`ra`**'. Nello stesso modo, il file '`/dev/rra0a`' è un dispositivo a caratteri della stessa partizione. È da osservare che con BSD un file di dispositivo a blocchi per il disco intero non è più disponibile, mentre rimane per

la versione a caratteri; per esempio, esiste '`/dev/rhk0`', ma **non esiste** più '`/dev/hk0`'.

Unità a nastro

I file di dispositivo delle unità a nastro non sono differenziati e vanno ricreati al volo, in base al tipo di nastro effettivamente esistente.

Collocazione dei file eseguibili

Nei primi sistemi UNIX non esistono le directory '`/sbin/`' e '`/usr/sbin/`', in quanto i programmi più delicati si trovano invece nella directory '`/etc/`' che non è inclusa nei percorsi predefiniti per questo. Pertanto, per avviare programmi come '**`mknod`**' o '**`mkfs`**', occorre anteporre tutto il percorso:

```
# /etc/mknod …
```

SIMH e il PDP-11

SIMH[1] è un simulatore di hardware per una serie di vecchi elaboratori, tra i quali anche il famoso PDP-11. La simulazione implica generalmente l'accesso a file su disco che rappresentano, di volta in volta, l'immagine di un disco, di un nastro, di una stampante.

Per poter utilizzare SIMH occorre leggere la documentazione originale annotata alla fine del capitolo. Qui vengono annotate solo poche cose e in particolare ciò che riguarda il PDP-11.

«

SIMH è costituito da un gruppo di programmi, ognuno specializzato per la simulazione di un certo tipo di elaboratore. Per esempio, per avviare la simulazione di un PDP-11, si usa normalmente il programma '**pdp11**'. Se si avvia il programma senza argomenti, si interagisce con il simulatore:

```
$ pdp11 [Invio]


  PDP-11 simulator V3.6-1

sim>
```

Da questa modalità interattiva, si danno dei comandi, con i quali si specificano delle opzioni di funzionamento e si definisce l'uso di file-immagine di unità che devono essere gestite. Di norma si prepara uno script per non perdere tempo, come nell'esempio seguente:

```
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
ATTACH  RL0  bsd_2.9_root_rl02.dsk
SHOW    RL0
;
; Boot.
;
BOOT    RL0
```

Le istruzioni che si possono dare dipendono molto dal tipo particolare di simulazione prescelto e sono documentate separatamente, rispetto alla guida generale sull'uso di SIMH. È comunque utile osservare che questi comandi non fanno differenza nell'uso di

lettere maiuscole o minuscole, a parte quando si fa riferimento a file-immagine, che vanno scritti come richiede il sistema operativo esterno.

Per eseguire uno script è sufficiente avviare il programma seguito dal nome dello stesso:

```
$ pdp11 avvio.ini[Invio]
```

In alternativa, durante il funzionamento interattivo, è possibile usare il comando '**DO**':

```
sim> do avvio.ini[Invio]
```

Durante il funzionamento interattivo del simulatore, è possibile concludere l'attività con il comando '**QUIT**':

```
sim> quit[Invio]
```

Oltre al fatto che i comandi di SIMH possono essere espressi indifferentemente con lettere maiuscole o minuscole, va osservato che questi comandi possono essere abbreviati; pertanto, spesso si vedono esempi di utilizzo di SIMH con comandi apparentemente differenti, per il solo fatto che sono stati abbreviati.

Avvio e sospensione di una simulazione

In condizioni normali, salvo configurazione differente, SIMH associa alla combinazione di tasti [*Ctrl e*] la sospensione della simulazione. In pratica, una volta avviata la simulazione, questa combinazione ne produce la sospensione.

Una volta sospesa una simulazione, la si può riprendere, allo stato in cui si trovava, con il comando '**CONT**'; inoltre, è possibile salvare lo stato di funzionamento di una simulazione sospesa in un file, per

poi recuperarla in un secondo momento e riprendere la simulazione da quella condizione.

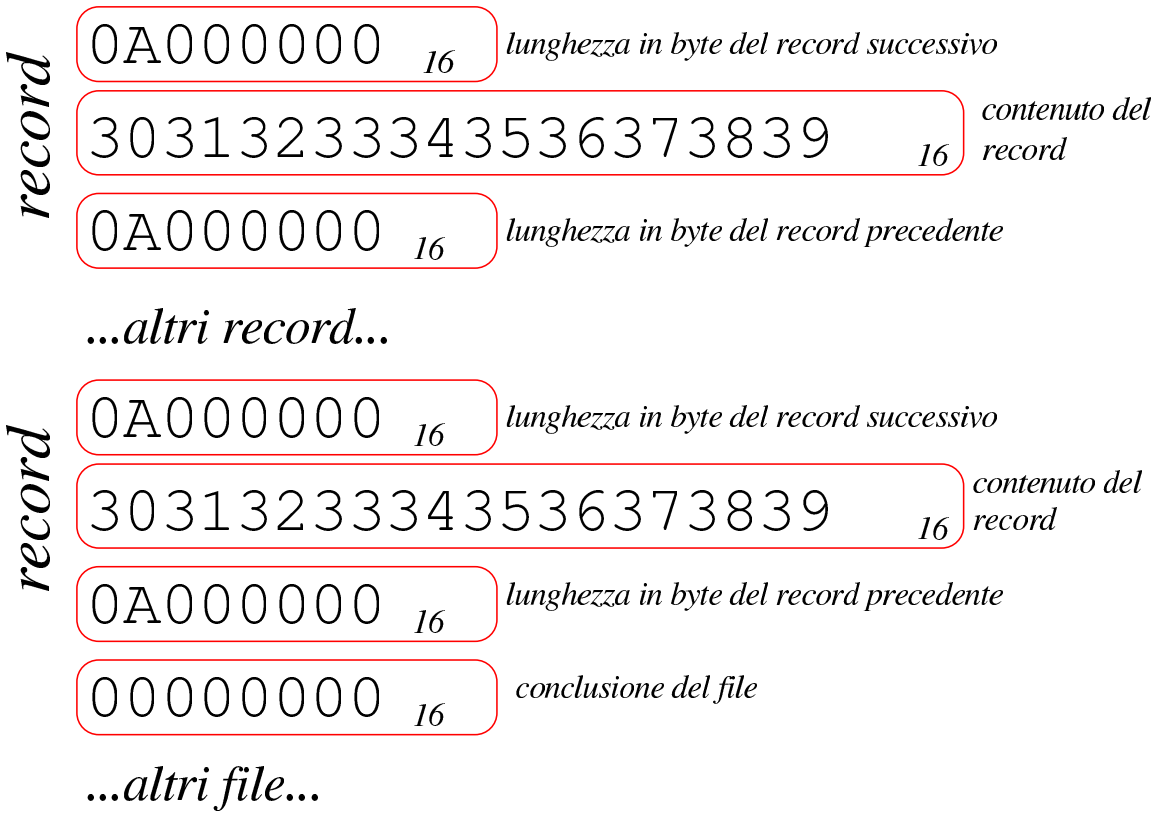
Comando	Descrizione
[*Ctrl e*]	Sospende la simulazione.
`CONT`	Riprende la simulazione sospesa.
`SAVE` ***file***	Salva in un file una simulazione sospesa.
`RESTORE` ***file***	Recupera da un file una simulazione sospesa (da riprendere poi con il comando '**`CONT`**').

Nastri virtuali

«

SIMH gestisce i nastri magnetici come file su disco, aventi però una struttura particolare che riproduce l'organizzazione dei dati nel nastro stesso. Il nastro, per sua natura, è suddiviso in ***record***; pertanto, anche i file di SIMH devono riprodurre tale informazione.

Figura u193.7. Struttura dei dati che rappresentano un nastro virtuale per SIMH.



In pratica, ogni record (che deve avere una quantità pari di byte) è preceduto e anche seguito da 32 bit che rappresentano la dimensione dello stesso, in byte, tenendo conto che il valore deve essere rappresentato in modalità *little-endian*. Alla fine del file, altri 32 bit a zero indicano la conclusione dello stesso (come se fosse l'inizio di un record vuoto. L'esempio che appare nella figura mostra la sequenza di questi dati; in particolare, il numero $0A000000_{16}$, va interpretato effettivamente come $0000000A_{16}$ (a causa dell'inversione *little-endian*), pari a 10_{10}; pertanto, il record è composto da 10 byte.

Figura u193.8. Esempio di un nastro virtuale per SIMH, visto in esadecimale e secondo il codice ASCII. Il file originale è una sequenza di 100 byte contenenti le cifre numeriche da zero a nove (da 30_{16} a 39_{16} secondo il codice ASCII), suddiviso in record da 10 byte.

```
00000000  0a 00 00 00 30 31 32 33  34 35 36 37 38 39 0a 00  |....0123456789..|
00000010  00 00 0a 00 00 00 30 31  32 33 34 35 36 37 38 39  |......0123456789|
00000020  0a 00 00 00 0a 00 00 00  30 31 32 33 34 35 36 37  |........01234567|
00000030  38 39 0a 00 00 00 0a 00  00 00 30 31 32 33 34 35  |89........012345|
00000040  36 37 38 39 0a 00 00 00  0a 00 00 00 30 31 32 33  |6789........0123|
00000050  34 35 36 37 38 39 0a 00  00 00 0a 00 00 00 30 31  |456789........01|
00000060  32 33 34 35 36 37 38 39  0a 00 00 00 0a 00 00 00  |23456789........|
00000070  30 31 32 33 34 35 36 37  38 39 0a 00 00 00 0a 00  |0123456789......|
00000080  00 00 30 31 32 33 34 35  36 37 38 39 0a 00 00 00  |..0123456789....|
00000090  0a 00 00 00 30 31 32 33  34 35 36 37 38 39 0a 00  |....0123456789..|
000000a0  00 00 0a 00 00 00 30 31  32 33 34 35 36 37 38 39  |......0123456789|
000000b0  0a 00 00 00 00 00 00 00                           |........|
000000b8
```

SIMH offre solo strumenti di conversione da altri formati o di analisi del contenuto, ma manca la possibilità di creare un nastro a partire da file comuni e di estrarre poi i file stessi. Per questo occorre realizzare un proprio programma. Gli esempi che si vedono nei listati successivi, funzionano correttamente se la piattaforma prevede interi da 32 bit, rappresentati in modalità *little-endian*.

Listato u193.9. Programma per la conversione di un file comune nel formato adatto a SIMH per simulare i nastri magnetici. Dovrebbe esserne disponibile una copia presso *allegati/convert_file_to_simh_tape.c* .

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    int  iRecordLength;          // 32 bit, little endian.
    char acRecord[65535];        // 65535 is the max record length.
    int  iZero;
```

```
    int  iByteRead;
    int  iByteLeft;
    //
    iZero = '\0';
    //
    // Get the record length from command line argument.
    //
    sscanf (argv[1], "%d", &iRecordLength);
    //
    // Read and write data.
    //
    while (1)   // Loop forever.
      {
        //
        // Read from standard input one record.
        //
        iByteRead = fread (acRecord, 1, iRecordLength, stdin);
        //
        if (iByteRead == iRecordLength)
          {
            //
            // The record was read completely.
            //
            fwrite (&iRecordLength, 4, 1, stdout);
            fwrite (acRecord, iByteRead, 1, stdout);
            fwrite (&iRecordLength, 4, 1, stdout);
          }
        else if (iByteRead == 0)
          {
            //
            // Nothing was read. The file is finished.
            //
            fwrite (&iZero, 1, 4, stdout);
            break;
          }
        else if (iByteRead < iRecordLength)
          {
            //
            // The record was read partially: it must be
            // filled with zeroes.
            //
            iByteLeft = iRecordLength - iByteRead;
            //
            fwrite (&iRecordLength, 4, 1, stdout);
```

```
            fwrite (acRecord, iByteRead, 1, stdout);
            while (iByteLeft > 0)
              {
                fwrite (&iZero, 1, 1, stdout);
                iByteLeft--;
              }
            fwrite (&iRecordLength, 4, 1, stdout);
            //
            // The file is finished.
            //
            fwrite (&iZero, 1, 4, stdout);
            break;
          }
      }
    return 0;
}
```

Il programma che si vede nel listato precedente converte un file normale in un «file su nastro», leggendo lo standard input e generando il risultato attraverso lo standard output. Se il file si chiama ‘`convert_file_to_simh_tape.c`’, si compila semplicemente così:

```
$ cc convert_file_to_simh_tape.c[Invio]

$ mv a.out convert_file_to_simh_tape[Invio]
```

Supponendo di volere convertire il file ‘`mio_file`’ in un nastro virtuale, avente record da 1024 byte, si può procedere così:

```
$ ./convert_file_to_simh_tape 1024 < mio_file > mio_file.tap
[Invio]
```

Una volta convertiti tutti i file che si vogliono usare, si possono mettere assieme in uno stesso «nastro virtuale», semplicemente concatenandoli, per esempio così:

```
$ cat file_0.tap file_1.tap … > nastro_completo.tap[Invio]
```

L'estrazione di un file da un nastro richiede invece un procedimento più complesso. L'esempio riportato nel listato successivo mostra un programma che si limita a estrarre il primo file.

Listato u193.10. Programma per l'estrazione del primo file contenuto nell'immagine di un nastro virtuale di SIMH.

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    int  iRecordLength;         // 32 bit, little endian.
    char acRecord[65535];       // 65535 is the max record length.
    int  iZero;
    int  iByteRead;
    int  iByteLeft;
    //
    iZero = '\0';
    //
    // Read and write data.
    //
    while (1)  // Loop forever.
      {
        //
        // Read from standard input the record length.
        //
        iByteRead = fread (&iRecordLength, 1, 4, stdin);
        //
        if (iByteRead == 0)
          {
            //
            // The file is finished.
            //
            break;
          }
        else if (iByteRead < 4)
          {
            //
            // This should not happen.
            //
            return 1;
          }
        else if (iRecordLength == 0)
          {
```

```
            //
            // As the value is zero, this is the end of the
            // first file, and no other file is saved.
            //
            break;
          }
        else
          {
            //
            // Continue reading a record.
            //
            iByteRead = fread (acRecord, 1, iRecordLength, stdin);
            //
            if (iByteRead < iRecordLength)
              {
                //
                // The record is not complete.
                //
                return 1;
              }
            else
              {
                //
                // The record seems ok: write to output.
                //
                fwrite (acRecord, iByteRead, 1, stdout);
                //
                // Try to read from standard input the same
                // old record length (and ignore it).
                //
                iByteRead = fread (&iRecordLength, 1, 4, stdin);
                //
                if (iByteRead < 4)
                  {
                    //
                    // this should not happen!
                    //
                    return 1;
                  }
              }
          }
        //
        // Continue the loop.
        //
```

```
        }
    return 0;
}
```

Il programma appena mostrato, si aspetta di leggere un file-immagine conforme alle specifiche di SIMH e non è in grado di gestire altre situazioni. Se si verificano errori, il programma termina di funzionare senza avvertimenti di qualunque sorta. Il file in ingresso viene atteso dallo standard input e il file estratto viene emesso attraverso lo standard output. Se il file si chiama '`estrai.c`', si compila semplicemente così:

```
$ cc estrai.c[Invio]

$ mv a.out estrai[Invio]
```

Supponendo di volere estrarre il primo file contenuto nell'immagine '`mio_file.tap`' si può procedere così:

```
$ ./estrai < mio_file.tap > file_0[Invio]
```

Dal momento che i dati in un nastro sono organizzati in record di dimensione uniforme, è normale che quanto estratto contenga qualche byte in più, ma a zero (00_{16}). Di norma, i file che vengono archiviati su nastro hanno una struttura tale per cui questa aggiunta diventa ininfluente.

Listato u193.11. Programma completo per l'estrazione di tutti i file da un'immagine di un nastro di SIMH. Dovrebbe esserne disponibile una copia presso *allegati/convert_simh_tape_to_file.c* .

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    int  iRecordLength;         // 32 bit, little endian
```

```
    char acRecord[65535];       // 65535 is the max record length.
    int  iZero;
    int  iByteRead;
    int  iByteLeft;
    char aczRootFileName[252];
    char aczFileName[255];
    int  iFileCounter;
    FILE *pfOut;
    //
    iZero        = '\0';
    iFileCounter = '\0';
    //
    // Get root file name.
    //
    sscanf (argv[1], "%s", aczRootFileName);
    //
    // Open the first output file.
    //
    sprintf (aczFileName, "%s-%03d", aczRootFileName, iFileCounter);
    pfOut = fopen (aczFileName, "w");
    printf ("%s\n", aczFileName);
    //
    // Read and write data.
    //
    while (1)
      {
        //
        // Read from standard input the record length.
        //
        iByteRead = fread (&iRecordLength, 1, 4, stdin);
        //
        if (iByteRead == 0)
          {
            //
            // The file is finished, although it is not
            // correctly ended. There are no more files.
            //
            fclose (pfOut);
            //
            break;
          }
        else if (iByteRead < 4)
          {
            //
```

```
            // This should not happen, but close anyway.
            //
            fclose (pfOut);
            //
            return 1;
          }
        else if (iRecordLength == 0)
          {
            //
            // Then length of the next record is zero.
            // This is the end of the first file: prepare the next one.
            //
            fclose (pfOut);
            iFileCounter++;
            sprintf (aczFileName, "%s-%03d", aczRootFileName, iFileCounter);
            pfOut = fopen (aczFileName, "w");
            printf ("%s\n", aczFileName);
            //
          }
        else
          {
            //
            // The record length was read: no read the record.
            //
            iByteRead = fread (acRecord, 1, iRecordLength, stdin);
            //
            if (iByteRead < iRecordLength)
              {
                //
                // The record is not complete: close.
                //
                fwrite (acRecord, iByteRead, 1, pfOut);
                fclose (pfOut);
                //
                return 1;
              }
            else
              {
                //
                // The record seems ok: write to output.
                //
                fwrite (acRecord, iByteRead, 1, pfOut);
                //
                // Try to read from standard input the same
```

```
                // old record length (and ignore it).
                //
                iByteRead = fread (&iRecordLength, 1, 4, stdin);
                //
                if (iByteRead < 4)
                  {
                    //
                    // this should not happen: close.
                    //
                    fclose (pfOut);
                    //
                    return 1;
                  }
              }
          }
        //
        // Continue the loop.
        //
      }
    //
    return 0;
}
```

Il programma che appare nell'ultimo listato è completo, in quanto estrapola tutti i file contenuti in un'immagine di nastro secondo SIMH. Il programma riceve dalla riga di comando la radice del nome dei file da creare, quindi genera una sequenza numerata con quella radice. In generale, l'ultimo file è vuoto e va ignorato.

Se il programma del listato è contenuto nel file '`simh_tape_to_file.c`', si compila così:

```
$ cc convert_simh_tape_to_file.c[Invio]

$ mv a.out convert_simh_tape_to_file[Invio]
```

Supponendo di volere estrarre i file contenuti nell'immagine '`mio_file.tap`' si può procedere così:

```
$ ./convert_simh_tape_to_file radice < mio_file.tap[Invio]

  radice-000
  radice-001
  radice-002
  …
```

In tal caso si ottengono i file ‘`radice-000`’, ‘`radice-001`’ e così di seguito.

Caratteristiche generali dei vari modelli PDP-11

I modelli di PDP-11 che sono esistiti hanno avuto caratteristiche abbastanza varie. Anche se si intende utilizzare solo un simulatore, è necessario scegliere il modello adatto al sistema operativo che si vuole installare. Le cose più importanti da sapere sono i tipi di bus ammissibili e la dimensione massima della memoria centrale.

Tabella u193.13. Caratteristiche generali dell’unità centrale, in ordine di anno di produzione.

Modello	Bus	Memoria centrale	Anno
PDP11/15	Unibus	64 Kibyte	1970
PDP11/20	Unibus	64 Kibyte	1970
PDP11/10	Unibus	64 Kibyte	1972
PDP11/05	Unibus	64 Kibyte	1972
PDP11/45	Unibus, Fastbus	256 Kibyte	1972
PDP11/50	Unibus, Fastbus	256 Kibyte	1972
PDP11/55	Unibus, Fastbus	256 Kibyte	1972
PDP11/35	Unibus	256 Kibyte	1973
PDP11/40	Unibus	256 Kibyte	1973
PDP11/03	Q-bus	64 Kibyte	1975
PDP11/04	Unibus	64 Kibyte	1975
PDP11/70	Unibus, Massbus	4 Mibyte	1975

Modello	Bus	Memoria centrale	Anno
PDP11/34	Unibus	256 Kibyte	1976
PDP11/60	Unibus	256 Kibyte	1976
PDP11/23	Q-bus	4 Mibyte	1979
PDP11/44	Unibus, Private memory bus	4 Mibyte	1979
PDP11/24	Unibus	4 Mibyte	1980
PDP11/73	Q-bus	4 Mibyte	1983
PDP11/21	Q-bus	64 Kibyte	1983
PDP11/83	Q-bus, Private memory bus	4 Mibyte	1985
PDP11/84	Unibus, Private memory bus	4 Mibyte	1985
PDP11/53	Q-bus	4 Mibyte	1987
PDP11/93	Q-bus, Private memory bus	4 Mibyte	1990
PDP11/94	Unibus, Private memory bus	4 Mibyte	1990

Simulazione unità a disco e a nastro

«

SIMH offre la simulazione di una gamma di dischi e nastri magnetici abbastanza ampia. La tabella successiva associa i tipi principali di dischi e nastri simulati all'unità di controllo e alla sigla usata da SIMH.

Tabella u193.14. I tipi principali di dischi e nastri gestiti da SIMH, associati ai nomi usati dai primi sistemi UNIX e BSD. Lo spazio disponibile viene calcolato in modo compatibile all'uso consueto: byte e multipli di byte.

SIMH	unità di controllo	modello	tipo	BSD		capacità byte	byte/ settore	settori/ traccia	tracce/ cilindro	cilindri
RK	RK11	RK05	cartridge	rk	/dev/**rk**0a /dev/r**rk**0a	2 494 464	512	12	2	203
HK	RK611	RK06		hk	/dev/**hk**0a /dev/r**hk**0a	13 888 512	512	22	3	411
		RK07				27 540 480	512	22	3	815
RL	RL11 RLV12	RL01		rl	/dev/**rl**0a /dev/r**rl**0a	5 242 880	256	40	2	256
		RL02				10 485 760	512	40	2	256
	RP11	RP01	pack	rp	/dev/**rp**0a /dev/r**rp**0a	5 196 800	512	5	10	203
		RP02				20 787 200	512	10	20	203
		RP03				40 960 000	512	10	20	400
RP	Massbus: RH70 RH11	RP04		xp	/dev/**xp**0a /dev/r**xp**0a	79 964 160	512	20	19	411
		RP05				79 964 160	512	20	19	411
		RP06				174 423 040	512	22	19	815
		RP07	fixed			516 096 000	512	50	32	630
		RM02	pack			67 420 160	512	32	5	823
		RM03				67 420 160	512	32	5	823
		RM05				256 196 608	512	32	19	823
		RM80	fixed			128 221 184	512	32	14	559
RQ	MSCP: RQDX3	RA60	pack	ra	/dev/**ra**0a /dev/r**ra**0a	204 890 112	512	42	4	2382
		RA70	fixed			280 084 992	512	33	11	1507
		RA71				700 062 720	512	51	14	1915
		RA72				1 000 089 600	512	51	20	1915
		RA73				2 007 290 880	512	70	21	2667
		RA80				121 325 568	512	31	14	546
		RA81				456 228 864	512	51	14	1248
		RA82				622 932 480	512	57	15	1423
		RA90				1 216 590 336	512	69	13	2649
		RA92				1 505 926 656	512	69	13	3279
		RD51	fixed			10 653 696	512	17	4	306
		RD52				30 965 760	512	18	7	480
		RD53				75 497 472	512	18	8	1024
		RD54				159 936 000	512	17	15	1225
		RD31				21 411 840	512	17	4	615
		RD32				42 823 680	512	17	6	820
		RX33	fd 5,25 in			1 228 800	512	15	2	80
						737 280	512	9	2	80
						409 600	512	10	1	80
						368 640	512	9	2	40
		RX50	fd 5,25 in			409 600	512	10	1	80
RX	RX11	RX01	fd 8 in			256 256	128	26	1	77
RY	RX211	RX02				512 512	256	26	1	77

TU	TM02 TM03	TE16	tape	ht	
		TU45			
		TU77			
TM		TE10		tm	
		TM11			
TS		TS11		ts	
		TSV05			

2.11BSD

Questo capitolo descrive un procedimento per installare 2.11BSD in un PDP-11 simulato attraverso SIMH, utilizzato a sua volta in un sistema GNU/Linux.

La lettura di questo capitolo è utile solo dopo quella della guida originale di Steven Schultz, *Installing and operating 2.11BSD on the PDP-11*, che può essere ottenuta dall'indirizzo *http://minnie.tuhs.org/PUPS/Setup/2.11bs11d_setup.pdf* .

I file usati per l'installazione descritta in questo capitolo provengono da *http://minnie.tuhs.org/Archive/PDP-11/Distributions/ucb/2.11BS11D/* . Eventualmente è disponibile anche una forma diversa della stessa distribuzione che parte direttamente da un file-immagine del disco contenente il file system principale, con tutto ciò che serve, già pronto per SIMH: *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/2.11_on_Simh/* .

Preparazione del nastro virtuale

Tra i file che compongono la distribuzione, quelli che servono per costruire il nastro di installazione sono i seguenti:

File	Descrizione
'mtboot'	Contiene il settore di avvio da usare con il nastro.
'boot'	Contiene il programma di avvio per la gestione iniziale del nastro.
'disklabel'	Contiene il programma, da avviare attraverso il nastro, usato per suddividere i dischi in partizioni.

File	Descrizione
‘`mkfs`’	Contiene il programma, da avviare attraverso il nastro, con il quale inizializzare le partizioni dei dischi.
‘`restore`’	Contiene il programma, da avviare attraverso il nastro, con il quale è possibile recuperare i dati archiviati con ‘**`dump`**’.
‘`icheck`’	Contiene il programma, da avviare attraverso il nastro, per il controllo del file system di una partizione.
‘`root.dump`’	Contiene l’archivio del file system principale, da estrarre con ‘**`restor`**’.
‘`file6.tar.gz`’	Contiene l’archivio di quanto si articola a partire da ‘`/usr/`’.
‘`file7.tar.gz`’	Contiene l’archivio di quanto si articola a partire da ‘`/usr/src/`’.
‘`file8.tar.gz`’	Contiene l’archivio di altro materiale che si articola sempre a partire da ‘`/usr/src/`’.

I tre file con estensione ‘`.tar.gz`’ vanno decompressi:

GNU/Linux `$` **`gunzip file*.tar.gz`** [*Invio*]

I primi due file vanno combinati assieme, in modo da formare un solo file organizzato nel modo seguente:

‘`mtboot`’	512 byte
‘`mtboot`’	512 byte
‘`boot`’	35328 byte

In pratica, essendo i file più corti delle dimensioni indicate, vanno completati con byte a zero; inoltre, come si vede, il settore di avvio va duplicato. Si procede nel modo seguente per aggiustare la

dimensione del primo file:

```
GNU/Linux $ dd if=mkboot count=1 bs=512 conv=sync ↩
↪     of=mtboot.sync [Invio]
```

Con il comando appena mostrato si ottiene il file ‘`mkboot.sync`’ che completa la dimensione di 512 byte, come richiesto per il concatenamento. Il comando successivo predispone il file di avvio da collocare nella prima posizione del nastro:

```
GNU/Linux $ cat mkboot.sync mkboot.sync boot > avvio [Invio]
```

Si può quindi procedere alla preparazione del nastro virtuale, tenendo conto della dimensione dei settori prevista dalla documentazione originale, utilizzando i programmi descritti nella sezione u0.2:

```
GNU/Linux $ convert_file_to_simh_tape 512 < avvio > tm11_0.tap [Invio]

GNU/Linux $ convert_file_to_simh_tape 1024 < disklabel ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux $ convert_file_to_simh_tape 1024 < mkfs ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux $ convert_file_to_simh_tape 1024 < restor ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux $ convert_file_to_simh_tape 1024 < icheck ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux $ convert_file_to_simh_tape 10240 < root.dump ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux $ convert_file_to_simh_tape 10240 < file6.tar ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux $ convert_file_to_simh_tape 10240 < file7.tar ↩
```

↪ **>> tm11_0.tap** [*Invio*]

GNU/Linux $ **convert_file_to_simh_tape 10240 < file8.tar** ↩
↪ **>> tm11_0.tap** [*Invio*][2]

Con gli strumenti di SIMH è possibile controllare il contenuto del nastro virtuale generato:

GNU/Linux $ **mtdump tm11_0.tap** [*Invio*]

Configurazione iniziale del simulatore e avvio

Si decide di simulare un PDP-11/44 del 1979, con solo 1 Mibyte di memoria centrale (il minimo per poter utilizzare 2.11BSD), con due unità a nastro connesse a un'unità di controllo TM11 (se si volesse usare l'unità TS11 si potrebbe gestire un solo nastro) e con un disco MSCP RA82 di dimensione inusuale: 200000000 byte. Un disco così capiente consente di installare tutto in una sola partizione, senza bisogno di innestarne altri.

Il file-immagine del disco deve essere creato prima di avviare il simulatore:

GNU/Linux $ **dd if=/dev/zero of=ra82_0.dsk bs=1000000 count=200** [*Invio*]

Così facendo viene creato il file '`ra82_0.dsk`'. Nella simulazione vengono usati inoltre i file '`tm11_0.tap`', creato precedentemente con il necessario per procedere all'installazione del sistema, e '`tm11_1.tap`', il cui scopo è quello di disporre di un'unità ulteriore per archiviare dati mentre si usa 2.11BSD nel simulatore. Il file-immagine del secondo nastro non va predisposto, perché viene creato contestualmente al suo utilizzo.

L'ultima fase prima dell'avvio del simulatore consiste nel predisporre uno script con la configurazione desiderata della simulazione:

```
;
; PDP-11/44 (1979) with only 1 Mibyte RAM memory.
;
SET    CPU  11/44
SET    CPU  1024K
SHOW   CPU
;
; Devices that might be disabled.
;
;SET    RK   DISABLE
;SET    HK   DISABLE
;SET    TC   DISABLE
;SET    TS   DISABLE
;
; TM11 tape simulator.
;
SET    TM   ENABLED
SET    TM0  LOCKED
ATTACH TM0  tm11_0.tap
SHOW   TM0
;
SET    TM1  WRITEENABLED
ATTACH TM1  tm11_1.tap
SHOW   TM1
;
; MSCP disk.
; The actual disk has an unusual size: 200000000 byte
;
SET    RQ   ENABLED
SET    RQ0  RAUSER=200
ATTACH RQ0  ra82_0.dsk
```

```
SHOW   RQ0
;
; Should boot manually.
;
```

Supponendo che questo file si chiami '**simh.ini**', si può avviare il simulatore nel modo seguente:

GNU/Linux `$ pdp11 simh.ini` [*Invio*]

```
PDP-11 simulator V3.5-1
Disabling XQ
CPU, 11/44, FPP, NOCIS, autoconfiguration on, 1024KB
TM0, attached to tm11_0.tap, write locked, SIMH format
TM: creating new file
TM1, attached to tm11_1.tap, write enabled, SIMH format
RQ0, 200MB, attached to ra82_0.dsk, write enabled, RAUSER
```

Lo script non contiene l'istruzione di avvio ('**BOOT**') che così deve essere data a mano. Ciò consente di verificare la correttezza della configurazione dai messaggi che si ottengono. Dall'invito del simulatore si può dare il comando di avvio attraverso il nastro contenente la distribuzione:

SIMH `sim> BOOT TM0` [*Invio*]

```
44Boot from tm(0,0,0) at 0172522
```

tape boot `:`

Il programma di avvio contenuto nel primo file del nastro viene eseguito e si presenta così un altro invito (i due punti), dove va scritto quale programma eseguire (quale file eseguire) all'interno del nastro.

«

Dall'invito del programma di avvio della distribuzione occorre iniziare selezionando il programma che consente di predisporre le partizioni all'interno del disco virtuale. Questo programma ('**disklabel**') si trova nel secondo file del nastro (è il secondo in base alla procedura descritta per la preparazione di tale nastro); pertanto si avvia così:

tape boot : **tm(0,1)** [*Invio*][3]

```
Boot: bootdev=0401 bootscr=0172522
disklabel
```

disklabel Disk?

A questo punto appare l'invito di '**disklabel**', dal quale è necessario inserire le coordinate del disco che si vuole suddividere in partizioni. In questo caso, si tratta di un'unità «ra», pertanto si usa la sigla '**ra(0,0)**':

disklabel Disk? **ra(0,0)** [*Invio*]

```
'ra(0,0)' is unlabeled or the label is corrupt.
```

disklabel Proceed? [y/n] **y**

disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? **y**

Si comincia visualizzando la situazione, per poter calcolare la posizione delle partizioni:

disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? **d**

```
type: MSCP
disk: RA82
label: DEFAULT
flags:
bytes/sector: 512
sectors/track: 57
tracks/cylinder: 15
sectors/cylinder: 855
cylinders: 457
rpm: 3600
drivedata: 0 0 0 0 0

1 partitions:
#  size    offset  fstype   [fsize bsize]
a: 390800  0       2.11BSD   1024  1024    # (Cyl. 0 - 457*)
```

La partizione '**a:**' viene creata automaticamente, ma va modificata perché le dimensioni non sono corrette e perché occorre comunque una partizione per lo scambio della memoria virtuale.

In base alla geometria del disco, sono disponibili 457 cilindri contenenti 855 settori da 512 byte; pertanto, ogni cilindro ha una capacità di 437760 byte. 2.11BSD può utilizzare una sola partizione per lo scambio della memoria virtuale, al massimo da 32 Mibyte; pertanto, si possono utilizzare al massimo 76 cilindri per questo fine, pari a 64980 settori, mentre i restanti 381 cilindri, pari a 325755 settori, vanno usati per il file system:

```
disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? m

  modify

disklabel d(isplay) g(eometry) m(isc) p(artitions) q(uit)? p
```

```
  modify partitions
disklabel d(isplay) n(umber) s(elect) q(uit)? s

disklabel a b c d e f g h q(uit)? a

  sizes and offsets may be given as sectors, cylinders
  or cylinders plus sectors:  6200, 32c, 19c10s respectively
  modify partition 'a'
disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? t

disklabel 'a' fstype [2.11BSD]: 2.11BSD [Invio]

  modify partition 'a'
disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? o

disklabel 'a' offset [0]: 0 [Invio][4]

  modify partition 'a'
disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? s

disklabel 'a' size [390800]: 325755 [Invio][5]

  modify partition 'a'
disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? f

disklabel 'a' frags/fs-block [1]: 1 [Invio]

  modify partition 'a'
```

```
disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? F

disklabel 'a' frag size [1024]: 1024 [Invio]

  modify partition 'a'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? q

  modify partitions
```

Termina così la configurazione della partizione '**a:**'. Si può passare a '**b:**', che **deve** essere usata per lo scambio della memoria virtuale.

```
disklabel d(isplay) n(umber) s(elect) q(uit)? s

disklabel a b c d e f g h q(uit)? b

  sizes and offsets may be given as sectors, cylinders
  or cylinders plus sectors:  6200, 32c, 19c10s respectively
  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? t

disklabel 'b' fstype [unused]: swap [Invio][6]

  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? o

disklabel 'b' offset [0]: 325755 [Invio][7]

  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? s
```

disklabel'b' size [0]: **64980**[*Invio*]

```
  modify partition 'b'
```

disklabeld(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? **f**

disklabel'b' frags/fs-block [1]: **1**[*Invio*]

```
  modify partition 'b'
```

disklabeld(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? **F**

disklabel'b' frag size [1024]: **1024**[*Invio*]

```
  modify partition 'b'
```

disklabeld(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size) q(uit)? **q**

```
  modify partitions
```

Termina anche la configurazione della partizione '**b:**' e si deve controllare che i dati inseriti siano coerenti, soprattutto che non ci siano accavallamenti tra le due partizioni.

disklabeld(isplay) n(umber) s(elect) q(uit)? **d**

```
type: MSCP
disk: RA82
label: DEFAULT
flags:
bytes/sector: 512
sectors/track: 57
tracks/cylinder: 15
sectors/cylinder: 855
cylinders: 457
rpm: 3600
drivedata: 0 0 0 0 0

2 partitions:
#  size    offset  fstype   [fsize bsize]
a: 325755  0       2.11BSD   1024 1024     # (Cyl. 0 - 380)
b: 64980   325755  swap                    # (Cyl. 381 - 456)

modify partitions
```

Si può procedere quindi con la partizione '**c:**', il cui scopo è solo quello di descrivere lo spazio usato complessivamente dalle altre due partizioni.

```
disklabel d(isplay) n(umber) s(elect) q(uit)? s

disklabel a b c d e f g h q(uit)? c

  sizes and offsets may be given as sectors, cylinders
  or cylinders plus sectors:  6200, 32c, 19c10s respectively
  modify partition 'c'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? o

disklabel 'c' offset [0]: 0 [Invio]
```

```
  modify partition 'c'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? s

disklabel 'c' size [0]: 390735[Invio]

  modify partition 'c'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? q

  modify partitions
```

Si controlla ulteriormente la situazione:

```
disklabel d(isplay) n(umber) s(elect) q(uit)? d
```

```
type: MSCP
disk: RA82
label: DEFAULT
flags:
bytes/sector: 512
sectors/track: 57
tracks/cylinder: 15
sectors/cylinder: 855
cylinders: 457
rpm: 3600
drivedata: 0 0 0 0 0

2 partitions:
#  size    offset  fstype   [fsize bsize]
a: 325755  0       2.11BSD   1024 1024     # (Cyl. 0 - 380)
b: 64980   325755  swap                    # (Cyl. 381 - 456)
c: 390735  0       unused    1024 1024     # (Cyl. 0 - 456)

modify partitions
```

A questo punto si può concludere confermando la suddivisione stabilita:

```
disklabel d(isplay) n(umber) s(elect) q(uit)? q

disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? w

disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? q
```

Si ritorna così sotto il controllo del programma di gestione del nastro:

```
44Boot from tm(0,0,1) at 0172522
```

tape boot :

«

Il programma che serve a inizializzare il file system nella prima partizione del disco si trova nel terzo file del nastro. Il programma in question è '**mkfs**':

```
tape boot : tm(0,2) [Invio]

  Boot: bootdev=0402 bootcsr=0172522
  Mkfs

mkfs file system: ra(0,0) [Invio]
```

Si osservi che le coordinate '**ra(0,0)**' rappresentano precisamente la prima partizione del disco. Viene proposta la dimensione del file system, che è corretta, perché si riferisce a unità da 1024 byte (si perde un settore, perché la partizione ne è composta da una quantità dispari).

```
mkfs file sys size [162877]: [Invio]
```

Si conferma anche la dimensione degli inode:

```
mkfs bytes per inode [4096]: [Invio]
```

Per quanto riguarda la sequenza dei settori nel disco, trattandosi di una simulazione in un file, non serve a nulla che questi siano alternati, pertanto si evita tale accorgimento:

```
mkfs interleaving factor (m; 2 default): 1 [Invio]

mkfs interleaving modulus (n; 427 default): 1 [Invio]

  m/n = 1 1
  Exit called
```

```
44Boot from tm(0,0,2) at 0172522
```

tape boot :

Terminata l'inizializzazione, si può fare la verifica del file system con il programma '**icheck**' che si trova nel quinto file del nastro:

tape boot : **tm(0,4)** [*Invio*]

```
Boot: bootdev=0404 bootcsr=0172522
Icheck
```

icheck File: **ra(0,0)** [*Invio*]

```
ra(0,0):
Not enough core; duplicates unchecked
files 3 (r=1,d=2,b=0,c=0,l=0,s=0)
used 2 (i=0,ii=0,iii=0,d=2)
free 160328

44Boot from tm(0,0,4) at 0172522
```

tape boot :

Copia dei file principali e primo avvio del sistema

Il nastro contiene quattro file separati da cui estrarre il contenuto del sistema operativo. Il primo, collocato nella sesta posizione del nastro, è un archivio ottenuto con il programma '**dump**' e contiene i file principali indispensabili per l'avvio di un sistema minimo; il secondo, collocato nella settima posizione, contiene ciò che va installato a partire dalla directory '`/usr/`'; il terzo, collocato nell'ottava posizione, contiene i sorgenti del kernel da installare a partire da '`/usr/src/`'; infine, il quarto, collocato nella nona posizione, contiene gli

altri sorgenti disponibili e va installato sempre a partire da '`/usr/src/`'.

Si comincia con il ripristino dei file principali; poi, le operazioni successive si devono svolgere con il sistema avviato regolarmente.

tape boot : **tm(0,3)** [*Invio*]

Viene caricato il programma per il ripristino dei dati archiviati: '**restore**'.

```
Boot: bootdev=0403 bootcsr=0172522
Restor
```

restor Tape?: **tm(0,5)** [*Invio*]

restor Disk?: **ra(0,0)** [*Invio*]

restor Last chance before scribbling on disk. [*Invio*]

```
End of tape

44Boot from tm(0,0,3) at 0172522
```

tape boot :

A questo punto si può avviare il sistema minimo appena installato, per poi proseguire con le altre fasi di copia della distribuzione.

tape boot : **ra(0,0)unix** [*Invio*]

Quanto appena scritto indica di avviare il file '`unix`' che si trova nella directory radice del file system collocato nella prima partizione del primo disco. Il file '`unix`' è quindi il kernel del sistema.

```
Boot: bootdev=02400 bootcsr=0172150

2.11 BSD UNIX #115: Sat Apr 22 19:07:25 PDT 2000
```

```
sms1@curly.2bsd.com:/usr/src/sys/GENERIC

ra0: Ver 3 mod 6
ra0: RA82  size=390800

phys mem  = 1048576
avail mem = 824640
user mem  = 307200

June  8 21:21:24 init: configure system

hk 0 csr 177440 vector 210 attached
ht ? csr 172440 vector 224 skipped:  No CSR.
ra 0 csr 172150 vector 154 vectorset attached
rl 0 csr 174400 vector 160 attached
tm 0 csr 172520 vector 224 attached
tms 0 csr 174500 vector 260 vectorset attached
ts ? csr 172520 vector 224 interrupt vector already in use.
xp 0 csr 176700 vector 254 attached
erase, kill ^U, intr ^C
```

A questo punto appare l'invito del sistema operativo e si deve procedere con l'installazione degli altri archivi. È da osservare che il primo nastro magnetico viene individuato dal file di dispositivo '`/dev/rmt12`' (ed eventualmente il secondo corrisponde a '`/dev/rmt13`'), mentre le partizioni '***x*:**' corrispondono ai file di dispositivo '`/dev/rra0`***x*'. Ma occorre prima accertarsi che i file di dispositivo siano quelli adatti all'hardware scelto.

«

Per il momento, il sistema è stato avviato con l'aiuto del programma di gestione del nastro. In un secondo momento, il nastro può essere usato ancora per avviare il disco, ma se è possibile, è meglio sistemare il programma di avvio all'inizio del disco:

2.11BSD # **cd /mdec** [*Invio*]

2.11BSD # **ls -l** [*Invio*]

```
total 14
-r--r--r--  1 root          512 Dec  5  1995 bruboot
-r--r--r--  1 root          512 Dec  5  1995 dvhpuboot
-r--r--r--  1 root          512 Dec  5  1995 hkuboot
-r--r--r--  1 root          512 Dec  5  1995 hpuboot
-r--r--r--  1 root          512 Dec  5  1995 rauboot
-r--r--r--  1 root          512 Dec  5  1995 rkuboot
-r--r--r--  1 root          512 Dec  5  1995 rluboot
-r--r--r--  1 root          512 Dec  5  1995 rm03uboot
-r--r--r--  1 root          512 Dec  5  1995 rm05uboot
-r--r--r--  1 root          512 Dec  5  1995 rx01uboot
-r--r--r--  1 root          512 Dec  5  1995 rx02uboot
-r--r--r--  1 root          512 Dec  5  1995 si51uboot
-r--r--r--  1 root          512 Dec  5  1995 si94uboot
-r--r--r--  1 root          512 Dec  5  1995 si95uboot
```

Dei programmi contenuti nella directory '`/mdec/`' occorre scegliere quello adatto al tipo di disco che si utilizza. In questo caso, si deve scegliere il file '`rauboot`'. Il file, della dimensione di un solo settore, va copiato con l'aiuto di '**dd**', all'inizio della prima partizione:

2.11BSD # **dd if=/mdec/rauboot of=/dev/rra0a count=1** [*Invio*]

```
1+0 records in
1+0 records out
```

A questo punto conviene verificare che l'operazione abbia avuto successo, arrestando il sistema (quello installato nell'hardware simulato con SIMH):

2.11BSD # **shutdown -h now** [*Invio*]

```
Shutdown at 18:02 (in 0 minutes) [pid 16]
#
System shutdown time has arrived
syncing disks... done
halting

HALT instruction, PC: 000014 (MOV #1,17406)
```

A questo punto si ritorna sotto il controllo di SIMH e si può tentare di avviare il sistema operativo direttamente dal disco:

SIMH sim> **BOOT RQ0** [*Invio*]

Si osservi che per SIMH, la sigla '**RQ0**' rappresenta il primo disco simulato, in base alla scelta dell'hardware fatta in precedenza.

```
44Boot from ra(0,0,0) at 0172150
```

disk boot : **ra(0,0)unix** [*Invio*]

```
Boot: bootdev=02400 bootcsr=0172150

2.11 BSD UNIX #115: Sat Apr 22 19:07:25 PDT 2000
sms1@curly.2bsd.com:/usr/src/sys/GENERIC

ra0: Ver 3 mod 6
ra0: RA82  size=390800
```

```
phys mem  = 1048576
avail mem = 824640
user mem  = 307200

hk 0 csr 177440 vector 210 attached
ht ? csr 172440 vector 224 skipped:  No CSR.
ra 0 csr 172150 vector 154 vectorset attached
rl 0 csr 174400 vector 160 attached
tm 0 csr 172520 vector 224 attached
tms 0 csr 174500 vector 260 vectorset attached
ts ? csr 172520 vector 224 interrupt vector already in use.
xp 0 csr 176700 vector 254 attached
erase, kill ^U, intr ^C
```

2.11BSD #

Predisposizione dei file di dispositivo e conclusione dell'installazione

«

Nella directory '`/dev/`' occorre eliminare i file di dispositivo riferiti alle unità a nastro, per ricrearli in base alle caratteristiche del tipo di nastro simulato effettivamente:

2.11BSD # **`cd /dev`** [*Invio*]

2.11BSD # **`rm *mt*`** [*Invio*]

2.11BSD # **`./MAKEDEV tm0`** [*Invio*][8]

2.11BSD # **`sync`** [*Invio*]

A questo punto, si può procedere con il recupero degli archivi rimasti.

2.11BSD # **cd /usr** [*Invio*]

Si riavvolge il nastro:

2.11BSD # **mt -f /dev/rmt12 rew** [*Invio*]

Si posiziona il nastro all'inizio della settima posizione:

2.11BSD # **mt -f /dev/rmt12 fsf 6** [*Invio*]

Si estrae l'archivio a partire dalla directory corrente:

2.11BSD # **tar xpbf 20 /dev/rmt12** [*Invio*]

Si procede in modo analogo per gli altri archivi.

2.11BSD # **mkdir /usr/src** [*Invio*]

2.11BSD # **cd /usr/src** [*Invio*]

In questo caso basta portare il nastro all'inizio del file successivo:

2.11BSD # **mt -f /dev/rmt12 fsf** [*Invio*]

2.11BSD # **tar xpbf 20 /dev/rmt12** [*Invio*]

Dopo aver estratto i sorgenti del kernel, occorre sistemare un collegamento simbolico:

2.11BSD # **cd /** [*Invio*]

2.11BSD # **rm -f /sys** [*Invio*]

2.11BSD # **ln -s usr/src/sys /sys** [*Invio*]

L'ultimo archivio da estrarre nella stessa directory '`/usr/src/`':

2.11BSD # **cd /usr/src** [*Invio*]

2.11BSD # **mt -f /dev/rmt12 fsf** [*Invio*]

2.11BSD # **tar xpbf 20 /dev/rmt12** [*Invio*]

Sistemare la data

«

Se si verifica la data, si può osservare che questa riporta l'anno 1995 e se si usa il comando '**date**', è possibile indicare solo le ultime due cifre dell'anno. Per risolvere il problema occorre un piccolo raggiro: prima si regola la data un secondo prima della mezzanotte del 1999, poi, passati al 2000, si può regolare l'ora in modo corretto:

2.11BSD # **date** [*Invio*]

```
Fri Jun  9 11:52:39 PDT 1995
```

2.11BSD # **date 9912312359.59** [*Invio*]

```
Fri Dec 31 23:59:59 PST 1999
```

2.11BSD # **date** [*Invio*]

```
Sat Jan  1 00:00:44 PST 2000
```

2.11BSD # **date 0702041821** [*Invio*]

```
Sun Feb  4 18:21:00 PST 2007
```

Si osservi che quando il sistema operativo (nell'hardware simulato) viene arrestato, l'orologio viene salvato e al riavvio successivo riprende da quel orario. Pertanto, a ogni riavvio occorre sistemare l'orologio.

2.11BSD # **shutdown -h now** [*Invio*]

Installazione di file-immagine pronti

Alcune edizioni dello UNIX di ricerca sono disponibili in file-immagine già pronti per questa o quella unità a disco. l'utilizzo di tali file con i simulatori è molto più semplice rispetto a una distribuzione su «nastro». Tuttavia, rimane il fatto che si tratta di versioni di UNIX prive di tanti accorgimenti a cui si è abituati se si conosce un sistema GNU e anche cose semplici come la correzione di quanto digitato sulla riga di comando possono essere impossibili.

UNIX versione 5 (RK05)

Si può trovare il file-immagine della versione 5 dello UNIX di ricerca, per un disco RK05, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/research/Dennis_v5/v5root.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux $ **gunzip < v5root.gz > unix_v5_root_rk05.dsk** [*Invio*]

Il nome scelto per il file estratto serve a sintetizzare le caratteristiche dell'immagine. Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RK05 cartridge disk.
;
SET     RK   ENABLE
ATTACH  RK0  unix_v5_root_rk05.dsk
SHOW    RK0
;
```

```
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`unix_v5.ini`’, si avvia la simulazione così:

GNU/Linux `$` **`pdp11 unix_v5.ini`** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RK0, 1247KW, attached to unix_v5_root_rk05.dsk, write enabled
```

Quindi, dall’invito di SIMH si dà il comando di avvio:

SIMH `sim>` **`BOOT RK0`** [*Invio*]

Se funziona, appare un altro invito, generato dal settore di avvio. Questo invito è rappresentato da una chiocciolina (‘`@`’), dopo la quale va scritto il nome del file del kernel da eseguire:

disk boot `@`**`unix`** [*Invio*]

Praticamente non c’è alcuna procedura di avvio, quindi si ottiene immediatamente la richiesta di identificazione dell’utente:

UNIX `login:` **`root`** [*Invio*]

UNIX `#`

Si annotano i file di dispositivo presenti:

UNIX `#` **`chdir /dev`** [*Invio*]

UNIX `#` **`ls -l`** [*Invio*]

```
total 0
cr--r--r--  1 bin      1,  0 Nov 26 18:13 mem
crw-rw-rw-  1 bin      1,  2 Nov 26 18:13 null
crw--w--w-  1 root     0,  0 Mar 21 12:10 tty8
#
```

Non è prevista la procedura di arresto del sistema ed è disponibile solo '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 014150 (INC R4)
```

SIMH sim> **quit** [*Invio*]

UNIX versione 6 (RK05)

Si può trovare il file-immagine della versione 6 dello UNIX di ricerca, per un disco RK05, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/research/Dennis_v6/v6root.gz* . Il file va estratto e quindi va preparato uno script per SIMH. Tuttavia, il settore di avvio contenuto nel file-immagine non funziona con il simulatore e va sostituito con una copia della versione 5.

Per prelevare il settore di avvio dal file-immagine della versione 5 si procede come nell'esempio seguente, dove si ottiene il file '`avvio`':

GNU/Linux $ **dd if=unix_v5_root_rk05.dsk of=avvio bs=512 count=1** [*Invio*]

Si estrae il file che contiene l'immagine principale della versione 6:

GNU/Linux $ **gunzip < v6root.gz > unix_v6_root_rk05_orig.dsk** [*Invio*]

Si separa la porzione successiva al primo settore, generando un file temporaneo:

GNU/Linux $ **dd if=unix_v6_root_rk05_orig.dsk of=tmp bs=512 skip=1** [*Invio*]

Si produce un nuovo file-immagine:

GNU/Linux $ **cat avvio tmp > unix_v6_root_rk05_fixed.dsk** [*Invio*]

Si prepara anche il secondo file-immagine previsto, contenente i sorgenti dei programmi:

GNU/Linux $ **gunzip < v6src.gz > unix_v6_root_rk05_src.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RK05 cartridge disks.
;
SET     RK   ENABLE
;
ATTACH  RK0  unix_v6_root_rk05_fixed.dsk
SHOW    RK0
;
ATTACH  RK1  unix_v6_src_rk05.dsk
SHOW    RK1
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`unix_v6.ini`’, si avvia la

simulazione così:

GNU/Linux `$ pdp11 unix_v6.ini` [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RK0, 1247KW, attached to unix_v6_root_rk05_fixed.dsk, write enabled
RK1, 1247KW, attached to unix_v6_src_rk05.dsk, write enabled
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

SIMH `sim> BOOT RK0` [*Invio*]

Se funziona appare l'invito del settore di avvio ('`@`'), dal quale va scritto il nome del file del kernel da eseguire: in questo caso si tratta del kernel '`rkunix`'.

disk boot `@rkunix` [*Invio*]

UNIX `login: root` [*Invio*]

UNIX `#`

Si annotano i file di dispositivo presenti:

UNIX `# chdir /dev` [*Invio*]

UNIX `# ls -l` [*Invio*]

```
total 0
crw-rw-r--  1 bin     8,  1 May 13 20:01 kmem
crw-rw-r--  1 bin     8,  0 May 13 20:01 mem
crw-rw-rw-  1 bin     8,  2 May 13 20:01 null
crw--w--w-  1 root    0,  0 Aug 14 22:06 tty8
```

Non è prevista la procedura di arresto del sistema ed è disponibile solo '**`sync`**', dopo il quale è possibile interrompere il funzionamento

del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 015670 (BNE 15722)
```

SIMH `sim>` **quit** [*Invio*]

UNIX versione 6 (RL02)

Si può trovare il file-immagine della versione 6 dello UNIX di ricerca, modificato per utilizzare un disco RL02, presso *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/v6_rl02_unknown.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux $ **gunzip < v6_rl02_unknown.gz > unix_v6_root_rl02.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  unix_v6_root_rl02.dsk
SHOW    RL0
;
; Should boot manually.
```

```
;
```

Se lo script è contenuto nel file ‘`unix_v6.ini`’, si avvia la simulazione così:

GNU/Linux `$` **`pdp11 unix_v6.ini`** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to unix_v6_root_rl02.dsk, write enabled, RL02
```

Quindi, dall’invito di SIMH si dà il comando di avvio:

SIMH `sim>` **`BOOT RL0`** [*Invio*]

Se funziona appare l’invito del programma di avvio (‘**`!`**’), dal quale va scritto il nome del file del kernel da eseguire: ‘`unix`’.

disk boot `!`**`unix`** [*Invio*]

```
unix v6 11/23
mem = 99 KW max = 63
```

UNIX `#`

Da questo punto, il terminale potrebbe funzionare solo con lettere maiuscole, perciò conviene dare il comando successivo, in modo da ottenere le lettere minuscole consuete:

UNIX `#` **`STTY -LCASE`** [*Invio*]

Si annotano qui i file di dispositivo presenti, ma è comunque disponibile, nella directory ‘`/dev/`’ un file ‘`Makefile`’:

UNIX `#` **`cd /dev`** [*Invio*]

UNIX # **ls -l** [*Invio*]

```
crw-rw-r--  1 root      23,   0 Jun 19  1984 ad0
crw-rw-rw-  1 root      23,   1 Jul 25  1984 ad1
crw-rw-rw-  1 root      23,   2 Jul 25  1984 ad2
crw-rw-rw-  1 root      23,   3 Jul 25  1984 ad3
crw-rw-rw-  1 root      23,   4 Jul 25  1984 ad4
crw-rw-rw-  1 root      23,   5 Jul 25  1984 ad5
crw-rw-rw-  1 root      23,   6 Jul 25  1984 ad6
crw-rw-rw-  1 root      23,   7 Jul 25  1984 ad7
brw-rw-r--  1 root       6,   0 Aug  4  1982 hm0
brw-rw-r--  1 root       6,   1 Aug  4  1982 hm1
brw-rw-r--  1 root       6,   2 Aug  4  1982 hm2
brw-rw-r--  1 root       6,   3 Aug  4  1982 hm3
brw-rw-r--  1 root       6,   4 Aug  4  1982 hm4
brw-rw-r--  1 root       6,   5 Aug  4  1982 hm5
brw-rw-r--  1 root       6,   6 Aug  4  1982 hm6
brw-rw-r--  1 root       6,   7 Aug  4  1982 hm7
brw-rw-r--  1 root       6,   8 Aug  4  1982 hm8
crw-rw-r--  1 root       8,   3 Aug  4  1982 imem
crw-rw-r--  1 root       8,   1 Aug  4  1982 kmem
c-w--w--w-  1 root       2,   0 Aug  4  1982 lp
crw-rw-r--  1 root       8,   0 Aug  4  1982 mem
brw-rw-r--  1 root       3,   0 Aug  4  1982 mt0
crw-rw-r--  1 root       8,   2 Dec 16  1983 null
brw-rw-r--  1 root      15,   0 Aug  4  1982 rhm0
brw-rw-r--  1 root      15,   1 Aug  4  1982 rhm1
brw-rw-r--  1 root      15,   2 Aug  4  1982 rhm2
brw-rw-r--  1 root      15,   3 Aug  4  1982 rhm3
brw-rw-r--  1 root      15,   4 Aug  4  1982 rhm4
brw-rw-r--  1 root      15,   5 Aug  4  1982 rhm5
brw-rw-r--  1 root      15,   6 Aug  4  1982 rhm6
brw-rw-r--  1 root      15,   7 Aug  4  1982 rhm7
brw-rw-r--  1 root      15,   8 Aug  4  1982 rhm8
brw-rw-r--  1 root       0,   0 Aug  4  1982 rk0
```

```
brw-rw-r--  1 root       0,   1 Aug  4  1982 rk1
brw-rw-r--  2 root       2,   0 May  2  1983 rl0
brw-rw-r--  1 root       2,   1 May 12  1983 rl1
c-w--w--w-  1 root       2,   1 Aug  4  1982 rlp
brw-rw-r--  1 root       1,   0 Aug  4  1982 rp0
brw-rw-r--  1 root       1,   1 Aug  4  1982 rp1
brw-rw-r--  1 root       1,   2 Aug  4  1982 rp2
brw-rw-r--  1 root       1,   3 Aug  4  1982 rp3
crw-rw-r--  1 root      10,   0 Aug  4  1982 rrk0
crw-rw-r--  1 root      10,   1 Aug  4  1982 rrk1
crw-rw-r--  1 root      18,   0 May 26  1986 rrl0
crw-rw-r--  1 root      18,   1 Sep 12 12:35 rrl1
crw-rw-r--  1 root      12,   0 Aug  4  1982 rrp0
crw-rw-r--  1 root      12,   1 Aug  4  1982 rrp1
crw-rw-r--  1 root      12,   2 Aug  4  1982 rrp2
crw-rw-r--  1 root      12,   3 Aug  4  1982 rrp3
crw-rw-rw-  1 root      11,   0 Feb 25  1983 rrx0
crw-rw-rw-  1 root      11,   1 Feb 25  1983 rrx1
crw-rw-rw-  1 root      11,   2 Apr 26  1983 rrx2
crw-rw-rw-  1 root      11,   3 Feb 25  1983 rrx3
brw-rw-rw-  1 root       5,   0 Feb 25  1983 rx0
brw-rw-rw-  1 root       5,   1 Mar  2  1983 rx1
brw-rw-rw-  1 root       5,   2 Feb 25  1983 rx2
brw-rw-rw-  1 root       5,   3 Apr 26  1983 rx3
crw-rw-r--  1 root      21,   0 Aug  4  1982 stat0
crw-rw-r--  1 root      21,   1 Aug  4  1982 stat1
brw-rw-r--  2 root       2,   0 May  2  1983 swap
crw-rw-rw-  1 root       9,   0 Feb 14  1984 tty
crw--w--w-  1 root       0,   1 Feb  2 15:16 tty1
crw--w--w-  1 root       0,   2 Feb  2 09:51 tty2
crw--w--w-  1 root       0,   3 Feb  2 15:16 tty3
crw--w--w-  1 root       0,   4 Feb  2 15:16 tty4
crw--w--w-  1 root       0,   5 Feb  2 15:16 tty5
crw--w--w-  1 root       0,   0 Feb  2 15:25 tty8
crw-rw-rw-  1 root      20,   0 May  7  1983 ttya
```

```
crw-rw-rw-  1 root      20,   1 May  7  1983 ttyb
crw-rw-rw-  1 root      20,   2 May  7  1983 ttyc
crw-rw-rw-  1 root      20,   3 May  7  1983 ttyd
crw-rw-rw-  1 root      20,   4 May  7  1983 ttye
crw-rw-rw-  1 root      20,   5 May  7  1983 ttyf
crw-rw-rw-  1 root      20,   6 May  7  1983 ttyg
crw-rw-rw-  1 root      20,   7 May  7  1983 ttyh
```

Non essendo prevista la procedura di arresto del sistema, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 017124 (CMPB #3,(R2))
```

SIMH sim> **quit** [*Invio*]

UNIX versione 7 (RL02)

Si può trovare il file-immagine della versione 7 dello UNIX di ricerca, per un disco RL02, presso *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/v7_rl02_1145.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux $ **gunzip < v7_rl02_1145.gz > unix_v7_root_rl02.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
```

```
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  unix_v7_root_rl02.dk
SHOW    RL0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`unix_v7.ini`’, si avvia la simulazione così:

GNU/Linux `$` **`pdp11 unix_v7.ini`** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to unix_v7_root_rl02.dsk, write enabled, RL02
```

Quindi, dall’invito di SIMH si dà il comando di avvio:

SIMH `sim>` **`BOOT RL0`** [*Invio*]

Se funziona appare l’invito del settore di avvio (‘`@`’), dal quale va scritto il nome del programma di avvio: ‘`boot`’.

disk boot `@`**`boot`** [*Invio*]

Quindi si inseriscono le coordinate del file del kernel da avviare:

```
New Boot, known devices are hp ht rk rl rp tm vt
```

disk boot **`: rl(0,0)rl2unix`** [*Invio*]

```
mem = 177856
```

UNIX #

Si annota il contenuto del file '`/dev/makefile`', con il quale si possono creare i file di dispositivo mancanti:

UNIX # **cd /dev**[*Invio*]

UNIX # **cat makefile**[*Invio*]

```
# You will want to do at least a make std (the default), followed
# the make on the types of disks you have
#

std:
/etc/mknod console  c 0 0
/etc/mknod tty     c 17 0
/etc/mknod tty1     c 0 1
/etc/mknod tty2     c 0 2
/etc/mknod mem      c 8 0
/etc/mknod kmem     c 8 1
/etc/mknod null     c 8 2

rk:
/etc/mknod rk0  b 0 0
/etc/mknod rk1  b 0 1
/etc/mknod rrk0 c 9 0
/etc/mknod rrk1 c 9 1
chmod go-rw rk0 rk1 rrk0 rrk1

rl:
/etc/mknod rl0  b 8 0
/etc/mknod rl1  b 8 1
/etc/mknod rrl0 c 18 0
/etc/mknod rrl1 c 18 1
```

```
chmod go-rw rl0 rl1 rrl0 rrl1

rp03:
/etc/mknod rp0 b 1 1
/etc/mknod swap b 1 2
/etc/mknod rp3 b 1 3
/etc/mknod rrp0 c 11 1
/etc/mknod rrp3 c 11 3
chmod go-rw rp0 swap rp3 rrp0 rrp3

rp04 rp05:
/etc/mknod rp0 b 6 0
/etc/mknod swap b 6 1
/etc/mknod rp3 b 6 6
/etc/mknod rrp0 c 14 0
/etc/mknod rrp3 c 14 6
chmod go-rw rp0 swap rp3 rrp0 rrp3

rp06:
/etc/mknod rp0 b 6 0
/etc/mknod swap b 6 1
/etc/mknod rp3 b 6 7
/etc/mknod rrp0 c 14 0
/etc/mknod rrp3 c 14 7
chmod go-rw rp0 swap rp3 rrp0 rrp3

tm:
/etc/mknod mt0 b 3 0
/etc/mknod rmt0 c 12 0
/etc/mknod nrmt0 c 12 128
chmod go+w mt0 rmt0 nrmt0

ht:
/etc/mknod mt0 b 7 64
/etc/mknod mt1 b 7 0
```

```
/etc/mknod rmt0 c 15 64
/etc/mknod rmt1 c 15 0
/etc/mknod nrmt0 c 15 192
/etc/mknod nrmt1 c 15 128
chmod go+w mt0 mt1 rmt0 rmt1 nrmt0 nrmt1
```

Non essendo prevista la procedura di arresto del sistema, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 002312 (RTS PC)
```

SIMH sim> **quit** [*Invio*]

UNIX versione 7 (RL02) «Torsten»

«

Si può trovare il file-immagine della versione 7 dello UNIX di ricerca, per un disco RL02, modificato da Torsten Hippe, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/other/Torsten_Hippe_v7/v7.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux $ **gunzip < v7.gz > unix_v7_root_rl02_torsten.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
```

```
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  unix_v7_root_rl02_torsten.dsk
SHOW    RL0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`unix_v7.ini`’, si avvia la simulazione così:

GNU/Linux `$` **`pdp11 unix_v7.ini`** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to unix_v7_root_rl02_torsten.dsk, write enabled, RL02
```

Quindi, dall’invito di SIMH si dà il comando di avvio:

SIMH `sim>` **`BOOT RL0`** [*Invio*]

Se funziona appare l’invito del settore di avvio (‘`@`’), dal quale va scritto il nome del programma di avvio: ‘`boot`’.

disk boot **`@boot`** [*Invio*]

Quindi si inseriscono le coordinate del file del kernel da avviare:

```
Boot
```

disk boot **`: rl(0,0)rl1unix`** [*Invio*]

```
mem = 205376
```

Viene chiesto di eseguire un accesso normale. La parola d'ordine per l'utente '**root**' è «pdp». Inizialmente il terminale mostra solo lettere maiuscole:

UNIX `SINGLE USER LOGIN:` **ROOT** [*Invio*]

UNIX `PASSWORD:` ***PDP*** [*Invio*]

Con il comando successivo si riporta il terminale a funzionare con le lettere minuscole:

UNIX # **STTY -LCASE** [*Invio*]

Si annota il contenuto del file '`/dev/makefile`', con il quale si possono creare i file di dispositivo mancanti:

UNIX # **cd /dev** [*Invio*]

UNIX # **cat makefile** [*Invio*]

```
basic:
/etc/mknod console c 0 0
/etc/mknod tty c 1 0
/etc/mknod mem c 2 0
/etc/mknod kmem c 2 1
/etc/mknod null c 2 2
chmod go-w+r console
chmod go-w mem kmem
chmod go+rw null tty
chown bin mem kmem null tty
chgrp bin mem kmem null tty

rp:
make TYPE=rp bigdisk

hp:
```

```
make TYPE=hp bigdisk

rm:
make TYPE=rm bigdisk

hk:
make TYPE=hk bigdisk

si:
make TYPE=si bigdisk

bigdisk:
/etc/mknod $(TYPE)0 b 2 0
#       /etc/mknod $(TYPE)01 b 2 1
#       /etc/mknod $(TYPE)02 b 2 2
#       /etc/mknod $(TYPE)03 b 2 3
#       /etc/mknod $(TYPE)04 b 2 4
#       /etc/mknod $(TYPE)05 b 2 5
#       /etc/mknod $(TYPE)06 b 2 6
#       /etc/mknod $(TYPE)07 b 2 7
/etc/mknod $(TYPE)1 b 2 8
#       /etc/mknod $(TYPE)11 b 2 9
#       /etc/mknod $(TYPE)12 b 2 10
#       /etc/mknod $(TYPE)13 b 2 11
#       /etc/mknod $(TYPE)14 b 2 12
#       /etc/mknod $(TYPE)15 b 2 13
#       /etc/mknod $(TYPE)16 b 2 14
#       /etc/mknod $(TYPE)17 b 2 15
/etc/mknod r$(TYPE)0 c 11 0
#       /etc/mknod r$(TYPE)01 c 11 1
#       /etc/mknod r$(TYPE)02 c 11 2
#       /etc/mknod r$(TYPE)03 c 11 3
#       /etc/mknod r$(TYPE)04 c 11 4
#       /etc/mknod r$(TYPE)05 c 11 5
#       /etc/mknod r$(TYPE)06 c 11 6
```

```
#       /etc/mknod r$(TYPE)07 c 11 7
/etc/mknod r$(TYPE)1 c 11 8
#       /etc/mknod r$(TYPE)11 c 11 9
#       /etc/mknod r$(TYPE)12 c 11 10
#       /etc/mknod r$(TYPE)13 c 11 11
#       /etc/mknod r$(TYPE)14 c 11 12
#       /etc/mknod r$(TYPE)15 c 11 13
#       /etc/mknod r$(TYPE)16 c 11 14
#       /etc/mknod r$(TYPE)17 c 11 15
chmod go-w $(TYPE)[0-7] $(TYPE)[0-7]? r$(TYPE)[0-7] r$(TYPE)[0-7]

rl:
/etc/mknod rl0 b 3 0
/etc/mknod rl1 b 3 1
/etc/mknod rrl0 c 12 0
/etc/mknod rrl1 c 12 1
chmod go-w rl? rrl?

rk:
/etc/mknod rk0 b 4 0
/etc/mknod rk1 b 4 1
/etc/mknod rrk0 c 13 0
/etc/mknod rrk1 c 13 1
chmod go-w rk? rrk?

dl:
/etc/mknod tty1 c 0 1
/etc/mknod tty2 c 0 2
chmod go-r+w tty[1-2]
< more 64% >dl:
/etc/mknod tty1 c 0 1
/etc/mknod tty2 c 0 2
chmod go-r+w tty[1-2]

pc:
```

```
/etc/mknod pc c 4 0
chown bin pc
chgrp bin pc
chmod go+w pc

lp:
/etc/mknod lp c 5 0
chown bin lp
chgrp bin lp
chmod go-rw lp

dc:
/etc/mknod dc0 c 6 0
/etc/mknod dc1 c 6 1
chown bin dc?
chgrp bin dc?
chmod go-rw dc?

dz:
/etc/mknod ttya c 7 0
/etc/mknod ttyb c 7 1
/etc/mknod ttyc c 7 2
/etc/mknod ttyd c 7 3
/etc/mknod ttye c 7 4
/etc/mknod ttyf c 7 5
/etc/mknod ttyg c 7 6
/etc/mknod ttyh c 7 7
chmod go-r+w tty[a-h]

tm:
/etc/mknod mt0 b 0 0
/etc/mknod mt1 b 0 1
/etc/mknod hmt0 b 0 64
/etc/mknod hmt1 b 0 65
/etc/mknod nmt0 b 0 128
```

```
/etc/mknod nmt1 b 0 129
/etc/mknod nhmt0 b 0 192
/etc/mknod nhmt1 b 0 193
/etc/mknod rmt0 c 10 0
/etc/mknod rmt1 c 10 1
/etc/mknod hrmt0 c 10 64
/etc/mknod hrmt1 c 10 65
/etc/mknod nrmt0 c 10 128
/etc/mknod nrmt1 c 10 129
/etc/mknod nhrmt0 c 10 192
/etc/mknod nhrmt1 c 10 193
chmod go-w *mt[01]

tc:
/etc/mknod tap0 b 1 0
/etc/mknod tap1 b 1 1
chmod go-w tap[01]

swap:
@echo ln ??? swap
```

Non essendo prevista la procedura di arresto del sistema, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 002312 (RTS PC)
```

SIMH sim> **quit** [*Invio*]

BSD versione 2.9 (RL02)

Si può trovare il file-immagine della versione 2.9 di BSD, per un disco RL02, presso *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/2.9BSD_rl02_1145.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux $ **gunzip < 2.9BSD_rl02_1145.gz > bsd_2.9_root_rl02.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  bsd_2.9_root_rl02.dsk
SHOW    RL0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`bsd_6.9.ini`’, si avvia la simulazione così:

GNU/Linux $ **pdp11 bsd_6.9.ini** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to bsd_2.9_root_rl02.dsk, write enabled, RL02
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

SIMH `sim>` **`BOOT RL0`** [*Invio*]

```
:boot

45Boot
```

Se funziona appare l'invito del programma di avvio ('**`:`**'), dal quale vanno scritte le coordinate per raggiungere il kernel da avviare:

disk boot **`: rl(0,0)rlunix`** [*Invio*]

```
Berkeley UNIX (Rev. 2.9.1) Sun Nov 20 14:55:50 PST 1983
mem = 135872

CONFIGURE SYSTEM:
xp 0 csr 176700 vector 254 attached
rk 0 csr 177400 vector 220 attached
hk 0 csr 177440 vector 210 attached
rl 0 csr 174400 vector 160 attached
rp ? csr 176700 vector 254 interrupt vector already in use
ht 0 csr 172440 vector 224 skipped:  No CSR
tm 0 csr 172520 vector 224 attached
ts 0 csr 172520 vector 224 interrupt vector already in use
dh ? csr 160020 vector 370 skipped:  No CSR
dm ? csr 170500 vector 360 skipped:  No autoconfig routines
dz ? csr 160110 vector 320 interrupt vector wrong
dz ? csr 160110 vector 320 interrupt vector wrong
dn 0 csr 175200 vector 300 skipped:  No autoconfig routines
vp ? csr 177500 vector 174 skipped:  No autoconfig routines
lp 0 csr 177514 vector 200 attached
Erase=^?, kill=^U, intr=^C
```

Si accede immediatamente alla shell. Si annota il contenuto della directory '`/dev/`'; tuttavia, è presente anche lo script '**`MAKE`**', con il quale si possono creare i file di dispositivo mancanti:

UNIX # **`cd /dev`** [*Invio*]

UNIX # **`ls -l`** [*Invio*]

```
total 5
-rwxrwxr-x 1 root     daemon      4375 Sep 19 01:21 MAKE
crw--w--w- 1 root     superuse    0,  0 Dec 31 16:03 console
brw------- 1 root     superuse    4,  0 Mar 29 15:42 hk0a
brw------- 1 root     superuse    4,  1 Mar 29 15:42 hk0b
brw------- 1 root     superuse    4,  2 Mar 29 15:42 hk0c
brw------- 1 root     superuse    4,  3 Mar 29 15:42 hk0d
brw------- 1 root     superuse    4,  6 Mar 29 15:42 hk0g
brw------- 1 root     superuse    4,  7 Mar 29 15:42 hk0h
crw------- 1 root     superuse    8,  1 Dec 31 16:02 kmem
crw------- 1 root     superuse    8,  0 Mar 29 15:40 mem
crw-rw-rw- 1 root     superuse    8,  2 Jul 27 17:58 null
crw------- 1 root     superuse   19,  0 Mar 29 15:42 rhk0a
crw------- 1 root     superuse   19,  1 Mar 29 15:42 rhk0b
crw------- 1 root     superuse   19,  2 Mar 29 15:42 rhk0c
crw------- 1 root     superuse   19,  3 Mar 29 15:42 rhk0d
crw------- 1 root     superuse   19,  6 Mar 29 15:42 rhk0g
crw------- 1 root     superuse   19,  7 Mar 29 15:42 rhk0h
brw------- 1 root     superuse    0,  0 Mar 29 15:40 rk0
brw------- 1 root     superuse    0,  1 Mar 30 00:14 rk1
brw------- 1 root     superuse    8,  0 Mar 29 15:40 rl0
brw------- 1 root     superuse    8,  1 Mar 30 00:14 rl1
brw-rw---- 1 root     daemon      6,  0 Mar 10 12:25 rm0a
brw-rw---- 1 root     daemon      6,  1 Mar 29 08:21 rm0b
brw------- 1 root     superuse    6,  2 Jun  2 09:41 rm0c
brw-rw---- 1 root     daemon      6,  3 Sep 29 18:13 rm0d
brw-rw---- 1 root     daemon      6,  4 Feb 25 04:58 rm0e
```

```
brw------- 1 root     superuse   1,  0 Mar 29 15:42 rp0a
brw------- 1 root     superuse   1,  1 Mar 29 15:43 rp0b
brw------- 1 root     superuse   1,  2 Mar 29 15:43 rp0c
brw------- 1 root     superuse   1,  7 Mar 29 15:43 rp0h
crw------- 1 root     superuse   9,  0 Jul 27 17:59 rrk0
crw------- 1 root     superuse   9,  1 Mar 30 00:14 rrk1
crw------- 1 root     superuse  18,  0 Dec 31 16:02 rrl0
crw------- 1 root     superuse  18,  1 Mar 30 00:14 rrl1
crw-rw---- 1 root     daemon    14,  0 Mar 15 18:30 rrm0a
crw-rw---- 1 root     daemon    14,  1 Jul 26 17:42 rrm0b
crw-rw-r-- 1 root     daemon    14,  2 Jan 11 21:19 rrm0c
crw-rw---- 1 root     daemon    14,  3 Mar 26 01:44 rrm0d
crw-rw---- 1 root     daemon    14,  4 Jan 18 08:36 rrm0e
crw------- 1 root     superuse  11,  0 Mar 29 15:42 rrp0a
crw------- 1 root     superuse  11,  1 Mar 29 15:43 rrp0b
crw------- 1 root     superuse  11,  2 Mar 29 15:43 rrp0c
crw------- 1 root     superuse  11,  7 Mar 29 15:43 rrp0h
crw------- 1 root     superuse  14,  0 Mar 29 15:40 rxp0a
crw------- 1 root     superuse  14,  1 Mar 29 15:41 rxp0b
crw------- 1 root     superuse  14,  2 Mar 29 15:41 rxp0c
crw------- 1 root     superuse  14,  3 Mar 29 15:41 rxp0d
crw------- 1 root     superuse  14,  4 Mar 29 15:41 rxp0e
crw------- 1 root     superuse  14,  5 Mar 29 15:41 rxp0f
crw------- 1 root     superuse  14,  6 Mar 29 15:41 rxp0g
crw------- 1 root     superuse  14,  7 Mar 29 15:41 rxp0h
crw-rw-rw- 1 root     superuse  17,  0 Mar 29 15:40 tty
crw--w--w- 1 root     sys       21,  0 Mar  7 09:13 tty00
crw--w--w- 1 root     superuse  21,  1 Mar  7 09:04 tty01
crw--w--w- 1 root     superuse  21,  2 Mar  7 09:04 tty02
crw--w--w- 1 root     superuse  21,  3 Mar  7 09:04 tty03
crw--w--w- 1 root     superuse  21,  4 Jul 26 17:15 tty04
crw--w--w- 1 root     superuse  21,  5 Aug  6 23:09 tty05
crw--w--w- 1 root     superuse  21,  6 Mar  7 09:04 tty06
crw--w--w- 1 root     superuse  21,  7 Mar  7 09:04 tty07
crw------- 1 root     superuse   4,  0 Mar 30 00:14 ttyh0
```

```
crw------- 1 root     superuse   4,  1 Mar 30 00:14 ttyh1
crw------- 1 root     superuse   4,  2 Mar 30 00:14 ttyh2
crw------- 1 root     superuse   4,  3 Mar 30 00:14 ttyh3
crw------- 1 root     superuse   4,  4 Mar 30 00:14 ttyh4
crw------- 1 root     superuse   4,  5 Mar 30 00:14 ttyh5
crw------- 1 root     superuse   4,  6 Mar 30 00:14 ttyh6
crw------- 1 root     superuse   4,  7 Mar 30 00:14 ttyh7
crw------- 1 root     superuse   4,  8 Mar 30 00:14 ttyh8
crw------- 1 root     superuse   4,  9 Mar 30 00:14 ttyh9
crw------- 1 root     superuse   4, 10 Mar 30 00:14 ttyha
crw------- 1 root     superuse   4, 11 Mar 30 00:14 ttyhb
crw------- 1 root     superuse   4, 12 Mar 30 00:14 ttyhc
crw------- 1 root     superuse   4, 13 Mar 30 00:14 ttyhd
crw------- 1 root     superuse   4, 14 Mar 30 00:14 ttyhe
crw------- 1 root     superuse   4, 15 Mar 30 00:14 ttyhf
brw------- 1 root     superuse   6,  0 Mar 29 15:40 xp0a
brw------- 1 root     superuse   6,  1 Mar 29 15:40 xp0b
brw------- 1 root     superuse   6,  2 Mar 29 15:41 xp0c
brw------- 1 root     superuse   6,  3 Mar 29 15:41 xp0d
brw------- 1 root     superuse   6,  4 Mar 29 15:41 xp0e
brw------- 1 root     superuse   6,  5 Mar 29 15:41 xp0f
brw------- 1 root     superuse   6,  6 Mar 29 15:41 xp0g
brw------- 1 root     superuse   6,  7 Mar 29 15:41 xp0h
```

Sarebbe disponibile il comando '**shutdown**', ma non sembra funzionare come di consueto. Pertanto, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 016662 (MOV #200,R4)
```

SIMH `sim>` **`quit`** [*Invio*]

Derivazioni di UNIX per hardware ridotto

«

Dalle versioni dello UNIX di ricerca sono derivate, a suo tempo, delle varianti per sistemi molto poveri di risorse. In particolare, Mini-UNIX e LSX. Si tratta di sistemi in grado di lavorare con una memoria centrale da 64 Kibyte.

Mini-UNIX

«

Mini-UNIX è un sistema derivato da UNIX versione 6, del quale utilizza lo stesso file system. È importante osservare che, anche se nel file system sono presenti le informazioni sui gruppi di utenti, questi non sono considerati.

Si possono trovare i file-immagine di Mini-UNIX, per dischi RK05, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/usdl/Mini-Unix/*. Servono precisamente i file ‘`tape1.bin.gz`’, ‘`tape2.bin.gz`’ e ‘`tape3.bin.gz`’.

I file vanno estratti e quindi va preparato uno script per SIMH. Nell’estrarre i file gli si attribuisce un nome che sintetizzi il loro contenuto:

GNU/Linux `$` **`gunzip < tape1.bin.gz > mx_root_rk05.dsk`** [*Invio*]

GNU/Linux `$` **`gunzip < tape2.bin.gz > mx_src_rk05.dsk`** [*Invio*]

GNU/Linux `$` **`gunzip < tape3.bin.gz > mx_man_rk05.dsk`** [*Invio*]

Lo script per SIMH può avere il contenuto seguente, nel quale si prevede l’uso di tre dischi:

```
;
; PDP-11/20 (1970)
;
SET     CPU  11/20
SET     CPU  64K
SHOW    CPU
;
; RK05 cartridge disks.
;
SET     RK   ENABLE
;
ATTACH  RK0  mx_root_rk05.dsk
SHOW    RK0
;
ATTACH  RK1  mx_src_rk05.dsk
SHOW    RK1
;
ATTACH  RK2  mx_man_rk05.dsk
SHOW    RK2
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`mx.ini`’, si avvia la simulazione così:

GNU/Linux `$` **`pdp11 mx.ini`** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling CR
Disabling XQ
CPU, 11/20, autoconfiguration on, 64KB
RK0, 1247KW, attached to mx_root_rk05.dsk, write enabled
RK1, 1247KW, attached to mx_src_rk05.dsk, write enabled
RK2, 1247KW, attached to mx_man_rk05.dsk, write enabled
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

SIMH `sim>` **`BOOT RK0`** [*Invio*]

Se funziona appare l'invito del settore di avvio ('`@`'), dal quale va scritto il nome del file del kernel da eseguire: in questo caso si tratta di '`rkmx`'.

disk boot `@`**`rkmx`** [*Invio*]

```
RESTRICTED RIGHTS

USE, DUPLICATION OR DISCLOSURE IS SUBJECT TO
RESTRICTIONS STATED IN YOUR CONTRACT WITH
WESTERN ELECTRIC COMPANY, INC.
```

MX `login:` **`root`** [*Invio*]

MX `#`

Si annotano i file di dispositivo presenti:

MX `#` **`chdir /dev`** [*Invio*]

MX `#` **`ls -l`** [*Invio*]

```
total 0
crw-rw-rw-  1 root    1,  1 Jan 26  1976 kmem
crw-rw-rw-  1 root    1,  0 Jan 26  1976 mem
crw-rw-rw-  1 root    1,  2 Jan 26  1976 null
brw-rw-rw-  1 root    0,  0 Sep 17 17:30 rk0
brw-rw-rw-  1 root    0,  1 Sep 18 01:53 rk1
crw--w--w-  1 root    0,  0 Sep 18 01:57 tty8
```

Come si può vedere, sono disponibili i file di dispositivo per due soli dischi, mentre nel simulatore ne sono stati previsti tre. Per aggiungere il file di dispositivo del terzo disco si può procedere così:

MX # **/etc/mknod rk2 b 0 2** [*Invio*]

È possibile innestare un solo disco alla volta. Per esempio, volendo aggiungere quello dei sorgenti corrispondente al secondo, si può procedere così:

MX # **/etc/mount /dev/rk1 /mnt** [*Invio*]

Poi, per il distacco del disco si procede come di consueto, specificando il file di dispositivo:

MX # **/etc/umount /dev/rk1** [*Invio*]

Non è prevista la procedura di arresto del sistema ed è disponibile solo '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

MX # **sync** [*Invio*]

MX # [*Ctrl e*]

```
Simulation stopped, PC: 016662 (BNE 16674)
```

SIMH sim> **quit** [*Invio*]

«

LSX è un sistema derivato da UNIX versione 6; per microprocessore LSI-11, ridotto al punto di poter funzionare con soli 48 Kibyte di memoria centrale.

Si possono trovare i file-immagine di LSX, per dischetti RX01, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/usdl/LSX/* ; precisamente serve il file *http://minnie.tuhs.org/Archive/PDP-11/Distributions/usdl/LSX/lsximgs.tar.bz2* che contiene tutto il necessario per questi esempi.

I file-immagine vanno estratti e quindi va preparato uno script per SIMH.

GNU/Linux $ **tar xjvf lsximgs.tar.bz2** [*Invio*]

Si ottengono diversi file, tra cui, in particolare, '`root.dsk`' e '`usr.dsk`'. Lo script per SIMH può avere il contenuto seguente, nel quale si prevede l'uso di due dischi:

```
;
; LSI-11 with only 48 Kibyte RAM memory.
;
SET     CPU  48K
SHOW    CPU
;
; RX01 floppy disk.
;
SET     RX   ENABLE
;
ATTACH  RX0  root.dsk
SHOW    RX0
;
ATTACH  RX1  usr.dsk
```

```
SHOW    RX1
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`lsx.ini`’, si avvia la simulazione così:

GNU/Linux `$` **`pdp11 lsx.ini`** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling CR
CPU, 11/73, NOCIS, autoconfiguration on, 48KB
RX: buffering file in memory
RX0, 256KB, attached to root.dsk, write enabled
RX: buffering file in memory
RX1, 256KB, attached to usr.dsk, write enabled
```

Quindi, dall’invito di SIMH si dà il comando di avvio:

SIMH `sim>` **`BOOT RX0`** [*Invio*]

Se funziona appare l’invito del settore di avvio (‘**`rx boot:`**’), dal quale va scritto il nome del file del kernel da eseguire: in questo caso si tratta di ‘`lsx`’.

disk boot `rx boot:`**`lsx`** [*Invio*]

Si ottiene subito l’invito della shell. Dal momento che il terminale si presenta configurato per le lettere maiuscole, conviene regolare subito questa cosa:

LSX `#` **`STTY -LCASE`** [*Invio*]

Si annotano i file di dispositivo presenti:

LSX # **chdir /dev** [*Invio*]

LSX # **ls -l** [*Invio*]

```
total 0
brw-rw-rw- 1 0        0,   0 Jun   8 15:00 fd0
brw-rw-rw- 1 0        0,   1 Oct 29 03:10 fd1
crw-rw-rw- 1 0        0,   0 Jul   1  1977 tty8
```

Nel sistema che si ottiene mancano programmi importanti e anche l'innesto del secondo dischetto può essere impossibile.

LSX # **sync** [*Invio*]

LSX # [*Ctrl e*]

```
Simulation stopped, PC: 015650 (MOV R3,(SP))
```

SIMH sim> **quit** [*Invio*]

Programmi di servizio

«

Le varie versioni dello UNIX di ricerca utilizzano dei file system inaccessibili con i sistemi attuali. Ciò rende difficile il trasferimento di dati con un file-immagine contenente uno dei vecchi UNIX. Probabilmente, l'unico programma che venga in aiuto per questo è V7fs, che comunque occorre compilare in proprio, ma almeno funziona in un sistema GNU/Linux comune.

V7fs

«

V7fs è un programma in grado di leggere un file-immagine contenente un file system di UNIX versione 7, di attraversare il suo contenuto e di estrapolare i file. Il programma va raccolto in forma sorgente da *http://minnie.tuhs.org/Archive/PDP-11/Tools/Filesys/v7fs-*

0.1.tar.gz . Dopo l'estrazione si ottiene in particolare il file '`v7fs.c`' e il file-make; pertanto si può compilare così:

GNU/Linux $ **make v7fs** [*Invio*]

Si ottiene il file eseguibile '`v7fs`' nella directory corrente. Supponendo di disporre del file '`unix_v7_root_rl02.dsk`', contenente un file system da scorrere con V7fs, si può procedere nel modo seguente:

GNU/Linux $ **./v7fs unix_v7_rl02.dsk** [*Invio*]

V7fs funziona in modo interattivo e mostra un invito, dal quale si possono dare comandi simili a quelli di un vecchio sistema UNIX:

v7fs * **?** [*Invio*]

```
commands:
        ls [-i] [dir]: list directory contents, current dir default
        cd name: change to directory 'name'
        cat name1: print file 'name1' on terminal
        cp name1 [name2]: copy internal file 'name1' to external 'name2'
               name2 defaults to name1.
               (An i-number can be used instead of name1 for cp or cat.)
        cpdir: copy all files in current internal directory
               to current external directory
        lcd name: change to local directory 'name'
        printi ino ...: print contents of inode 'ino'
        printblk blk ...: print contents of block 'blk'
        printsb: print contents of the super block
        dumpblk blk ...: hex dump of block 'blk'
        dumpboot: hex dump of the boot block
        cpblk file blk ...: copy contents of 'blk' to external file 'file'
               (append to file if it exists)
        rootino ino: read directory with inode 'ino', making it
               the root directory
        ! : shell escape; the rest of the line is passed to the shell
        q or ^d: quit
```

v7fs * **ls** [*Invio*]

```
.               ..              bin             boot
dev             etc             hphtunix        hptmunix
lib             mdec            rkunix          rl2unix
rphtunix        rptmunix        usr
```

v7fs * **cd etc** [*Invio*]

v7fs * **ls** [*Invio*]

```
.               ..              accton          cron
ddate           dmesg           fsck            getty
group           init            mkfs            mknod
mount           mtab            passwd          rc
ttys            umount          update          utmp
wall
```

v7fs * **cat group** [*Invio*]

```
other::1:
sys::2:bin,sys
bin::3:sys,bin
uucp::4:
```

v7fs * **cp group** [*Invio*]

L'ultimo comando mostra la copia del file '`group`' all'esterno, in un file con lo stesso nome, nella directory corrente nel momento dell'avvio del programma.

v7fs * **q** [*Invio*]

Riferimenti

- Bob Supnik, *The Computer History Simulation Project*
  http://simh.trailing-edge.com/
- Robert M. Supnik, *SIMH user's guide*
  http://simh.trailing-edge.com/pdf/simh_doc.pdf
- Robert M. Supnik, *PDP-11 simulator usage*
  http://simh.trailing-edge.com/pdf/pdp11_doc.pdf
- Phil Budne, *Article 1995 of alt.sys.pdp10, How big were they?*
  http://www.inwap.com/pdp10/usenet/disks
- John Holden, *YAPP - Yet another PDP-11 Page*
  http://www.psych.usyd.edu.au/pdp-11/
- Steven Schulz, *Installing and operating 2.11BSD on the PDP-11*, 1995
  http://minnie.tuhs.org/PUPS/Setup/2.11bs11d_setup.pdf
  http://minnie.tuhs.org/PUPS/Setup/2.11bs11d_setup.html
- *2.11BSD*
  http://minnie.tuhs.org/Archive/PDP-11/Distributions/ucb/2.11BS11D/
  http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/2.11_on_Simh/
- Warren Toomey, *Details of the PUPS Archive*, 1996; *FAQ on the Unix Archive and Unix on the PDP-11*, 2001; *What Unixes run on What PDPs?*
  http://minnie.tuhs.org/PUPS/archive_details.html

http://minnie.tuhs.org/PUPS/pupsfaq.html

http://minnie.tuhs.org/PUPS/node6.html

- John Lions, *Lions' Commentary on Unix 6th Edition with Source Code*, Peer-To-Peer Communications, sesta edizione, 1996, ISBN 1573980137

  http://www.amazon.ca/Lions-Commentary-Unix-Source-Code/dp/1573980137

- *Mini Unix*

  http://minnie.tuhs.org/UnixTree/MiniUnix/

[1] **SIMH** software libero con licenza speciale

[2] Rispetto alla documentazione originale, il file '`file8.tar`' viene inserito in coda al nastro principale, senza bisogno di creare un secondo nastro apposito. Nella realtà ciò non sarebbe possibile, per via della capacità limitata del nastro stesso.

[3] La sigla '**`tm`**' va utilizzata in quanto si tratta di un nastro di un'unità a nastro di tipo TM11; se fosse un nastro TS11, va usata probabilmente la sigla '**`ts`**', come descritto nella documentazione di 2.11BSD.

[4] La partizione '**`a:`**' deve iniziare a partire dal primo settore disponibile del disco, altrimenti non è possibile avviare poi il sistema operativo.

[5] La dimensione viene data in settori e la si cambia in base ai calcoli effettuati precedentemente.

[6] Si osservi che è obbligatorio dare il nome '**`swap`**' alla partizione usata per lo scambio della memoria virtuale.

[7] La partizione ‘**b:**’ deve cominciare a partire dal settore successivo a quello della partizione ‘**a:**’. Dal momento che la partizione ‘**a:**’ è composta da 325755 settori, contando a partire da zero, l’ultimo settore della prima partizione è il numero 325754, pertanto il successivo, che inizia la partizione ‘**b:**’ è il numero 325755.

[8] La sigla ‘**tm0**’ fa riferimento alle unità TM11.