Gestione della memoria

Dal punto di vista del kernel di os16, l'allocazione della memoria riguarda la collocazione dei processi elaborativi nella stessa. Disponendo di una quantità di memoria esigua, si utilizza una mappa di bit per indicare lo stato dei blocchi di memoria, dove un bit a uno indica un blocco di memoria occupato.

Nel file 'memory.h' viene definita la dimensione di un blocco di memoria e, di conseguenza, la quantità massima che possa essere gestita. Attualmente i blocchi sono da 256 byte, pertanto, sapendo che la memoria può arrivare solo fino a 640 Kibyte, si gestiscono al massimo 2560 blocchi.

Per la scansione della mappa si utilizzano interi da 16 bit, pertanto tutta la mappa si riduce a 40 di questi interi, ovvero 80 byte. Nell'ambito di ogni intero da 16 bit, il bit più significativo rappresenta il primo blocco di memoria di sua competenza. Per esempio, per indicare che si stanno utilizzando i primi 1280 byte, pari ai primi cinque blocchi di memoria, si rappresenta la mappa della memoria come «F80000000...».

Il fatto che la mappa della memoria vada scandito a ranghi di 16 bit va tenuto in considerazione, perché se invece si andasse con ranghi differenti, si incapperebbe nel problema dell'inversione dei byte.

Quando possibile, si fa riferimento a indirizzi di memoria efficaci, nel senso che, con un solo valore, si rappresentano le posizioni da 00000_{16} a $FFFFF_{16}$. Per questo viene predisposto il tipo derivato '**`addr_t`**' nel file '`memory.h`'.

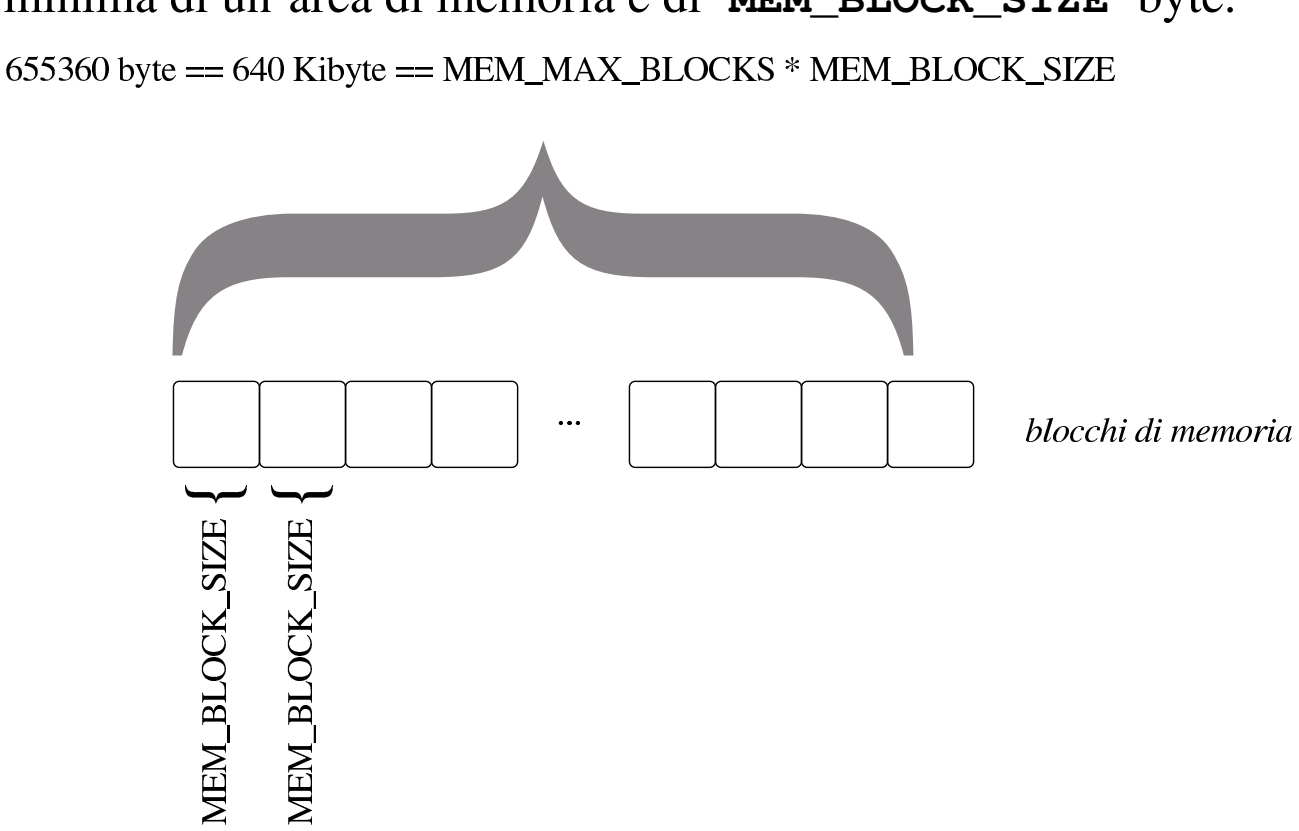
File «kernel/memory.h» e «kernel/memory/...»

«

Listato u0.8 e successivi.

Il file '`kernel/memory.h`', oltre ai prototipi delle funzioni usate per la gestione della memoria, definisce la dimensione del blocco minimo di memoria e la quantità massima di questi, rispettivamente con le macro-variabili ***MEM_BLOCK_SIZE*** e ***MEM_MAX_BLOCKS***; inoltre predispone i tipi derivati '**`memory_t`**', '**`segment_t`**', '**`offset_t`**' e '**`addr_t`**', corrispondenti, rispettivamente, a una variabile strutturata che consente di rappresentare un'area di memoria in modo relativamente comodo (indirizzo efficace, segmento e dimensione), un indirizzo di segmento, uno scostamento dall'inizio di un segmento e un indirizzo efficace.

Figura u145.1. Mappa della memoria in blocchi: la dimensione minima di un'area di memoria è di '**`MEM_BLOCK_SIZE`**' byte.



Nei file della directory '`kernel/memory/`' viene dichiarata la mappa della memoria, corrispondente a un array di interi a 16 bit, denominato ***mb_table[]***. L'array è pubblico, tuttavia è disponibile anche una funzione che ne restituisce il puntatore: ***mb_reference()***. Tale funzione sarebbe perfettamente inutile, ma rimane per uniformità rispetto alla gestione delle altre tabelle.

Nelle funzioni che riguardano l'allocazione della memoria, quando si indica la dimensione di questa, spesso si considera il valore zero equivalente a 10000_{16}, ovvero la dimensione massima di un segmento secondo l'architettura.

Tabella u145.2. Funzioni per la gestione della mappa della memoria, dichiarate nel file di intestazione ‘`kernel/memory.h`’ e realizzate nella directory ‘`kernel/memory/`’.

Funzione	Descrizione
`uint16_t *mb_reference (void);`	Restituisce il puntatore alla tabella dei blocchi di memoria, per uniformare l’accesso alla tabella dalle funzioni che non fanno parte del gruppo contenuto nella directory ‘`kernel/memory/`’.
`ssize_t mb_alloc (addr_t` ***address*****`,`** `size_t` ***size*****`);`**	Alloca la memoria a partire dall’indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). L’allocazione ha termine anticipatamente se si incontra un blocco già utilizzato. La funzione restituisce la dimensione allocata effettivamente.

Funzione	Descrizione
`ssize_t mb_free (addr_t` ***address*****`,`** `size_t` ***size*****`);`	Libera la memoria a partire dall'indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). Lo spazio viene liberato in ogni caso, anche se risulta già libero; tuttavia viene prodotto un avvertimento a video se si verifica tale ipotesi.

`int mb_alloc_size`
`(size_t` ***size*****`,`**
`memory_t *`***allocated***`);`
	Cerca e alloca un'area di memoria della dimensione richiesta, modificando la variabile strutturata di cui viene fornito il puntatore come secondo parametro. In pratica, l'indirizzo e l'estensione della memoria allocata effettivamente si trovano nella variabile strutturata in questione, mentre la funzione restituisce zero (se va tutto bene) o -1 se non è disponibile la memoria libera richiesta.

Tabella u145.3. Funzioni per le operazioni di lettura e scrittura in memoria, dichiarate nel file di intestazione ‘`kernel/memory.h`’ e realizzate nella directory ‘`kernel/memory/`’.

Funzione	Descrizione

`void mem_copy (addr_t` ***orig*`,`
`addr_t` ***dest*`,`
`size_t` ***size*`);`
	Copia la quantità richiesta di byte, dall’indirizzo di origine a quello di destinazione, espressi in modo efficace.

`size_t mem_read (addr_t` ***start*`,`
`void *`***buffer*`,`
`size_t` ***size*`);`
	Legge dalla memoria, a partire dall’indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene letto va poi copiato nella memoria tampone corrispondente al puntatore generico indicato come secondo parametro.

`size_t mem_write (addr_t` ***start*`,`
`void *`***buffer*`,`
`size_t` ***size*`);`
	Scrive, in memoria, a partire dall’indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene scritto proviene dalla memoria tampone corrispondente al puntatore generico indicato come secondo parametro.

Scansione della mappa di memoria

«

Listato u0.8 e successivi.

La mappa della memoria si rappresenta (a sua volta in memoria), con un array di interi a 16 bit, dove ogni bit individua un blocco di

memoria. Pertanto, l'array si compone di una quantità di elementi pari al valore di '**`MEM_MAX_BLOCKS`**' diviso 16.

Il primo elemento di questo array, ovvero ***mb_table[0]***, individua i primi 16 blocchi di memoria, dove il bit più significativo si riferisce precisamente al primo blocco. Per esempio, se ***mb_table[0]*** contiene il valore $F800_{16}$, ovvero 1111100000000000_2, significa che i primi cinque blocchi di memoria sono occupati, mentre i blocchi dal sesto al sedicesimo sono liberi.

Dal momento che i calcoli per individuare i blocchi di memoria e per intervenire nella mappa relativa, possono creare confusione, queste operazioni sono raccolte in funzioni statiche separate, anche se sono utili esclusivamente all'interno del file in cui si trovano. Tali funzioni statiche hanno una sintassi comune:

```
int mb_block_set1   (int block)
int mb_block_set0   (int block)
int mb_block_status (int block)
```

Le funzioni ***mb_block_set1()*** e ***mb_block_set0()*** servono rispettivamente a impegnare o liberare un certo blocco di memoria, individuato dal valore dell'argomento. La funzione ***mb_block_status()*** restituisce uno nel caso il blocco indicato risulti allocato, oppure zero in caso contrario.

Queste tre funzioni usano un metodo comune per scandire la mappa della memoria: il valore che rappresenta il blocco a cui si vuole fare riferimento, viene diviso per 16, ovvero il rango degli elementi dell'array che rappresenta la mappa della memoria. Il risultato intero della divisione serve per trovare quale elemento dell'array conside-

rare, mentre il resto della divisione serve per determinare quale bit dell'elemento trovato rappresenta il blocco desiderato. Trovato ciò, si deve costruire una maschera, nella quale si mette a uno il bit che rappresenta il blocco; per farlo, si pone inizialmente a uno il bit più significativo della maschera, quindi lo si fa scorrere verso destra di un valore pari al resto della divisione.

Per esempio, volendo individuare il terzo blocco di memoria, pari al numero 2 (il primo blocco corrisponderebbe allo zero), si avrebbe che questo è descritto dal primo elemento dell'array (in quanto 2/16 dà zero, come risultato intero), mentre la maschera necessaria a trovare il bit corrispondente è 0010000000000000_2, la quale si ottiene spostando per due volte verso destra il bit più significativo (due volte, pari al resto della divisione).

Una volta determinata la maschera, per segnare come occupato un blocco di memoria, basta utilizzare l'operatore OR binario:

```
        mb_table[i] = mb_table[i] | mask;
```

Se invece si vuole liberare un blocco di memoria, si utilizza un AND binario, invertendo però il contenuto della maschera:

```
        mb_table[i] = mb_table[i] & ~mask;
```

Va osservato che la rappresentazione dei blocchi nella mappa è invertita rispetto ad altri sistemi operativi, in quanto non sarebbe tanto logico il fatto che il bit più significativo si riferisca invece alla parte più bassa del proprio insieme di blocchi di memoria. La scelta è dovuta al fatto che, volendo rappresentare la mappa numericamente, la lettura di questa sarebbe più vicina a quella che è la percezione umana del problema.