Sistemi Operativi
Esercizi File System

Docente: Claudio E. Palazzi
cpalazzi@math.unipd.it

Appello AE-2 del 14/9/2005

Quesito 1 (punti 8). Sia data una memoria secondaria di ampiezza 64 GB, organizzata in blocchi di ampiezza 1 kB. Dopo aver calcolato la dimensione minima di un indice di blocco per tale memoria, sotto il vincolo che essa debba essere un multiplo di 8 (*bit*), si determini la dimensione massima di *file* ottenibile nel caso pessimo di contiguità nulla sotto le seguenti ipotesi:

1. *file system* di tipo NTFS, con *record* ampi 1 kB, 408 B riservati all'attributo dati nel *record* principale ed 800 B nei *record* di estensione, utilizzando esattamente 2 *record*;
2. *file system* di tipo Extfs, con *i-node* ampi 1 kB, nodo principale contenente 16 indici di blocco ed 1 indice di I e di II indirezione, utilizzando l'intero *i-node* principale.

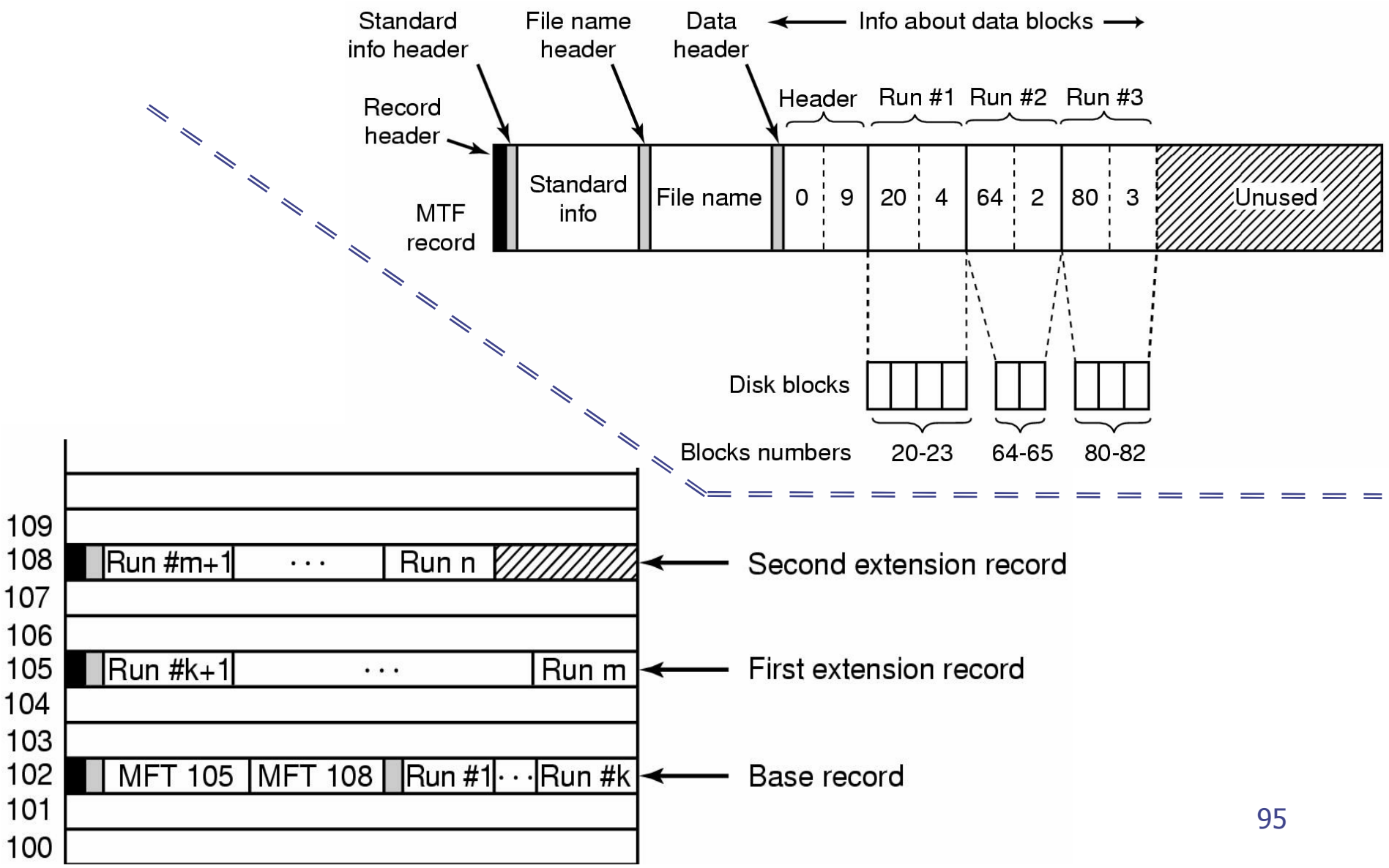
Calcolate le dimensioni richieste, si determini per ciascun tipo di *file system*, il rapporto inflattivo determinato dalla sua organizzazione strutturale, ossia l'onere proporzionale dovuto alla memorizzazione delle strutture di rappresentazione rispetto a quella dei dati veri e propri.

Soluzione

Soluzione 1 (punti 8). Essendo la memoria secondaria ampia 64 GB e i blocchi ampi 1 kB, è immediato calcolare che siano necessari $\lceil \frac{64\ GB}{1\ kB} \rceil = 64\ M = 2^6 \times 2^{20} = 2^{26}$ indici, la cui rappresentazione binaria banalmente richiede 26 *bit*. Stante il vincolo che la dimensione dell'indice debba essere un multiplo di 8 *bit*, la dimensione dell'indice deve salire a 32 *bit* (4 B). Vediamo ora quale possa essere la dimensione massima di *file* ottenibile sotto le ipotesi fissate dal quesito.

***File system* di tipo** NTFS : Dei 408 B riservati all'attributo dati nel *record* principale, $2 \times 4 = 8$ B saranno riservati alla coppia {base, indice}, mentre i rimanenti $408 - 8 = 400$ B potranno essere utilizzati per indicare le sequenze contigue di caso peggiore (dunque tutte ampie 1 blocco). Poiché ciascuna sequenza richiede una coppia di indici {inizio, fine}, pari a $2 \times 4 = 8$ B, il *record* principale potrà ospitare $\lfloor \frac{400\ B}{8\ B} \rfloor = 50$ sequenze ampie 1 blocco. Il *record* di estensione dispone invece di 800 B per la memorizzazione di $\lfloor \frac{800\ B}{8\ B} \rfloor = 100$ ulteriori sequenze. Ne segue che, sotto le ipotesi del quesito, la dimensione massima di *file* consentita da NTFS è pari a: $(50 + 100)$ blocchi $\times$ 1kB/blocco = 150 kB, al costo di 2 *record*, ciascuno ampio 1 kB. Il rapporto inflattivo in questo caso è dunque pari a: $\frac{2\ kB}{150\ kB} = 1.33\%$.

Soluzione (descrizione *record* MFT)



Soluzione

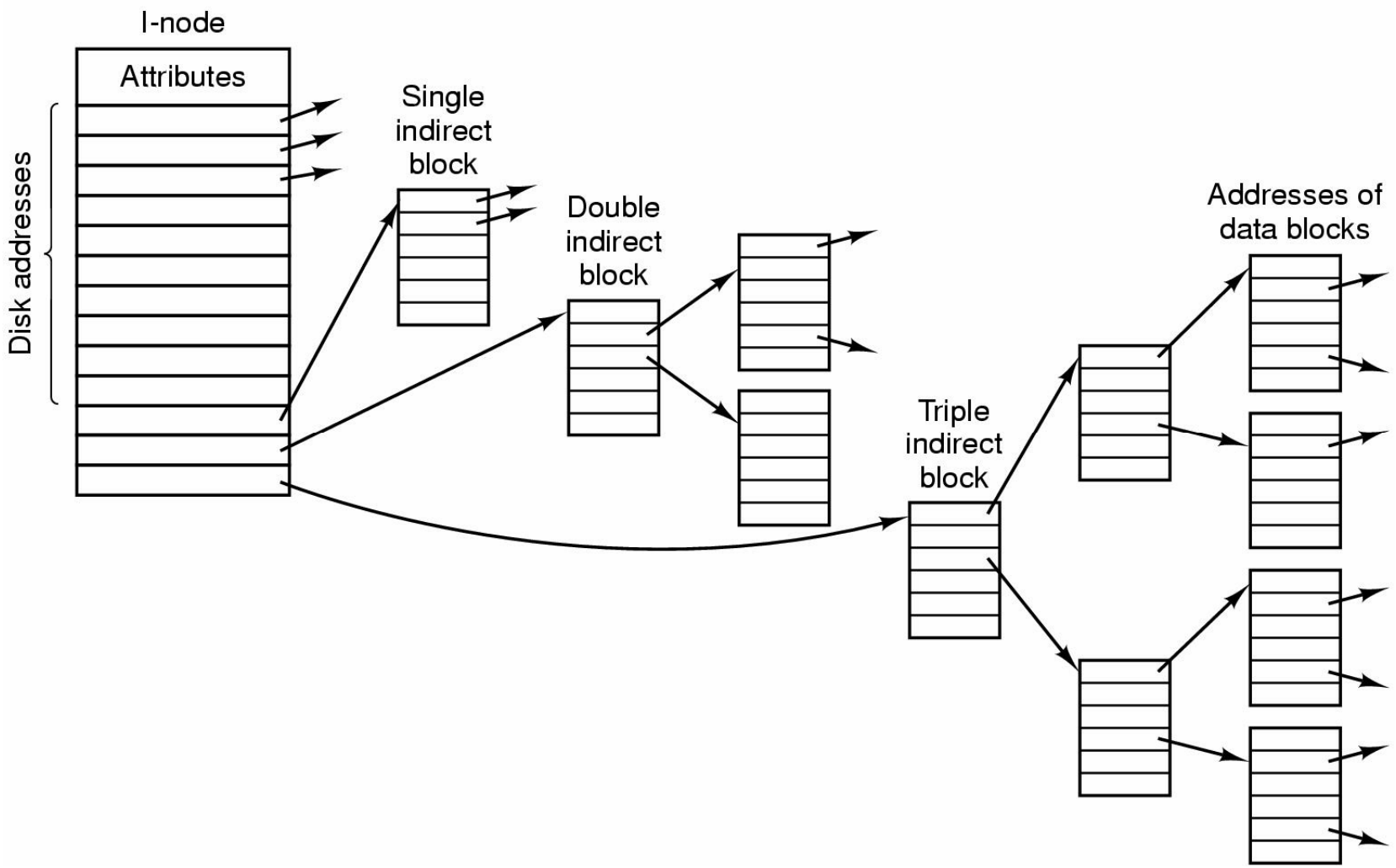
***file system* di tipo** Extfs : In questo caso, utilizzando tutti i campi dell'*i-node* principale, abbiamo a disposizione:

- 16 indici diretti di blocco, al costo di 1 blocco poiché un *i-node* occupa lo stesso spazio di un blocco;
- 1 indice di I indirezione, il quale punta ad un *i-node* interamente utilizzato per contenere indici diretti di blocco, che consente di esprimere fino a: $\lfloor \frac{1\,kB}{4\,B} \rfloor = \lfloor \frac{2^{10}}{2^2} \rfloor = 2^8 = 256$ indici di blocco, al costo di 1 ulteriore blocco;
- 1 indice di II indirezione, il quale punta ad un *i-node* speciale, interamente utilizzato per contenere puntatori ad *i-node* di I livello, che dunque consente di esprimere 256 puntatori a strutture ciascuna contenente fino a 256 indici diretti di blocco, per un totale di $256^2 = (2^8)^2 = 2^{16} = 65.536$ blocchi, al costo di $1 + 256 = 257$ ulteriori blocchi.

Conseguentemente, Extfs consente di rappresentare *file* di dimensione massima pari a: $(16 + 256 + 65.536) \times 1\,kB = 65.808\,kB = 64\,MB + 272\,kB$ (438,72 volte maggiore di quanto ottenuto con NTFS) per un rapporto inflattivo pari a: $\frac{(1+1+257)\times 1\,kB}{65.808\,kB} = 0,39\%$ (3,4 volte inferiore a quanto ottenuto con NTFS).

La considerazione ovvia da trarre da queste considerazioni è che NTFS) è particolarmente penalizzato dalle condizioni di scarsa o nulla contiguità. (Per contro, ad Extfs la contiguità non giova in alcun modo.)

Soluzione (descrizione *i-node*)



Esercizio "*Keeping Track of Free Blocks*"

Sia dato un disco di 16 GB diviso in blocchi ampi 1 KB. Si considerino due possibili strutture per tener traccia dei blocchi liberi: lista concatenata e *bitmap*. Nel primo caso, ogni elemento della lista è costituito a sua volta da un blocco, il quale contiene indici di blocco (di 32 *bit* ciascuno), dei quali l'ultimo è riservato per l'indicazione del prossimo blocco di lista libera. Nel secondo caso l'uso di un *bit* 1 o 0 definisce se il corrispondente blocco sia libero o utilizzato.
Si calcoli l'occupazione di memoria delle due strutture.

Soluzione

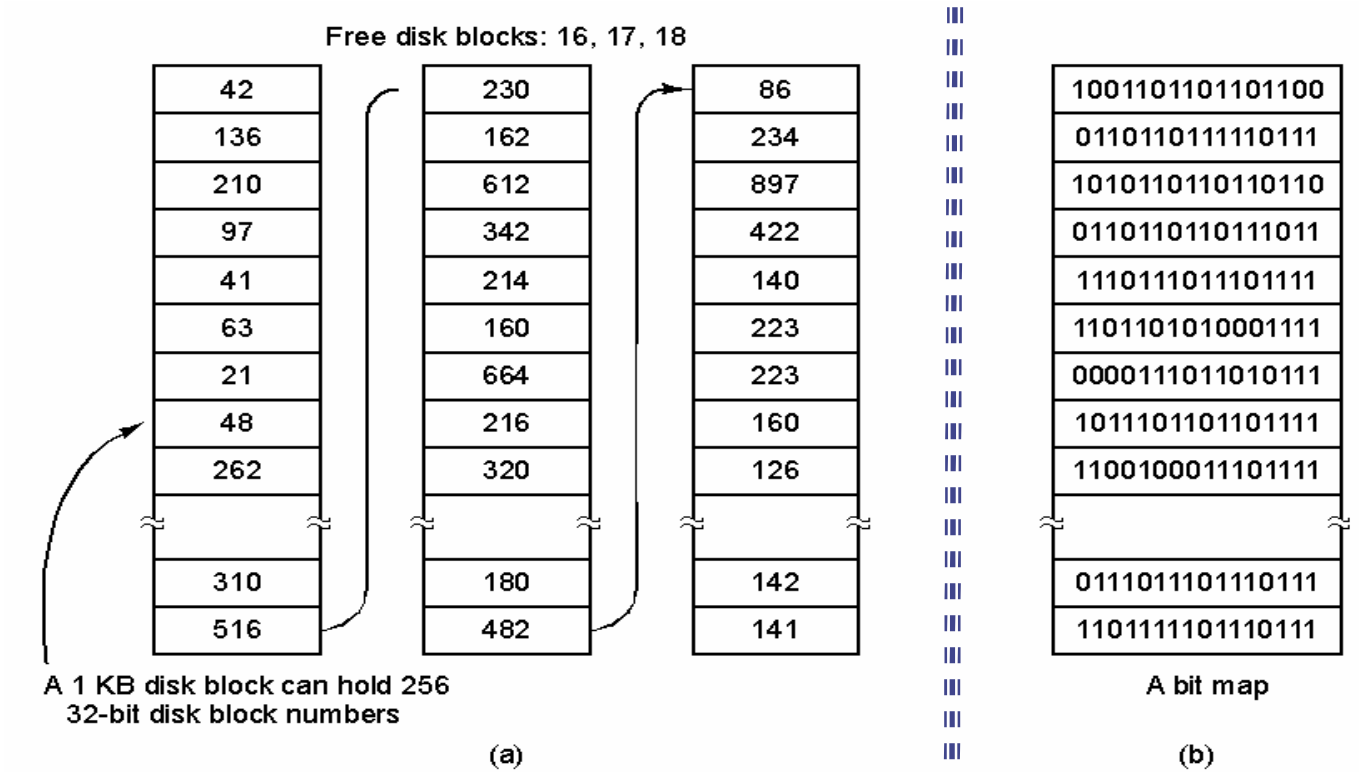
16 GB = 2^{34} B diviso in blocchi da 1 KB = 2^{10} B/blocco
ovvero 2^{34} B / 2^{10} B/blocco = 2^{24} blocchi = 16 M blocchi.

Ogni blocco può contenere 1 KB / 4 B/indice = 256 indici di blocco di cui 1 viene usato come collegamento al "blocco di indici" successivo nella lista. Ne rimangono dunque 255 utilizzabili per rappresentare i blocchi liberi.

Per rappresentare una lista di **massima ampiezza** servono dunque:
16 M indici / 255 indici/blocco = 65793,0039.. ≈ **65794** blocchi
cioè poco più di 64 K × 1 KB = **64 MB**

Con la struttura a *bitmap* sono invece **sempre** necessari 2^{24} *bit* = 2^{21} B =
= 2^{11} KB = **2 MB**

Soluzione



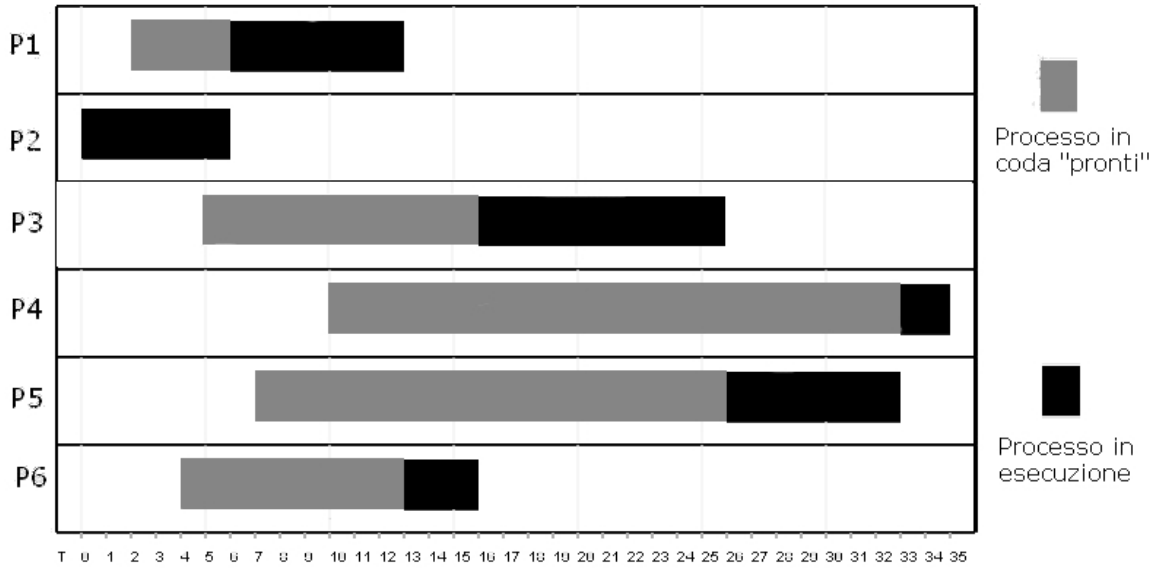
Riformulare la soluzione:

- variando la dimensione del blocco
- senza conoscere a priori la dimensione di ogni indice (32 *bit*)

Esercizio - Simulatore

Considerando i processi P1, P2, P3, P4, P5 e P6, aventi un ordine di arrivo e di esecuzione su una macchina monoprocessore così come in figura si determini quale/i tra le seguenti politiche di scheduling senza priorità esplicite possa essere stata utilizzata:

1. FIRST IN FIRST OUT (Sì o No)
2. ROUND ROBIN (in caso di risposta positiva, indicare un esempio di ampiezza del quanto temporale)
3. SHORTEST JOB FIRST nella versione **con** prerilascio (Sì o No)



Soluzione

1. FIFO: Si'

2. RR: Si', con qualsiasi quanto temporale di ampiezza maggiore o uguale al massimo tempo di esecuzione fra i processi considerati, ovvero 10 u.t. (per P3)

3. SHORTEST JOB FIRST con prerilascio: No

Esercizio – processi e thread

Si discutano i meccanismi di livello di S.O. che intervengono nell'esecuzione del seguente codice

```
1:  #include <pthread.h>
2:  #include <stdio.h>
3:  void *troublemaker(void *param);
4:  int main(int argc, char *argv[]) {
5:  int pid; // id (intero) del processo creato da fork()
6:  pthread_t tid; // id (strutturato) di un thread
7:  pid = fork();
8:  if (pid == 0) { // ramo del processo figlio
9:     pthread_create(&tid,NULL,troublemaker,NULL);
10:   pthread_join(tid,NULL);}
11: else if (pid > 0) { // ramo del processo padre
12:    waitpid(pid,0,0);}
13: }
```

Soluzione

Il processo avra' inizio con l'invocazione dell'eseguibile prodotto da:
gcc –g thread.c –o thread –lpthread

Linea 4 – con l'invocazione dell'eseguibile viene creato dal S.O. un singolo processo *P* con assegnata la sua memoria virtuale in modalita' *paging-on-demand*. Essendo il programma di dimensione molto piccola e' plausibile assumere che stara' tutto contenuto in una singola pagina. Dato l'ambiente GNU/Linux, al processo *P* il S.O. assegna implicitamente un *thread* *Tp* che ne rappresenta il flusso di controllo.

Linea 7 – l'invocazione di `fork()` duplica il processo *P* creando un clone identico *C*, ma non comprensivo dei thread interni, e in modalita' *copy-on-write*; come per *P* viene creato implicitamente un thread *Tc1* il cui PC (*program counter*) viene posizionato alla stessa linea 7 del programma, al punto in cui la variabile pid viene assegnata; *Tc1* legge una copia diversa della variabile `pid`, che vale 0 (un valore fittizio) per *Tc1* e un valore positivo (vero) per *Tp*

Soluzione

Linea 9 – *Tc1* crea un nuovo *thread Tc2* con il quale condivide tutte le risorse fisiche e logiche, e lo incarica di eseguire la procedura `troublemaker`

Linea 10 – *Tc1* si mete in attesa del completamento di *Tc2*. In pratica si mette in attesa che `troublemaker` sia eseguita; le azioni effettuate entro `troublemaker` da parte di *Tc2* possono operare sulle stesse variabili di *Tc1* e modificarne il valore (anche per *Tc1*). Tali azioni non hanno effetto sulla copia di *P* di tali variabili.

Linea 12 – La terminazione di *Tc1* comporta la terminazione di tutto *C* che non ha altri *thread* in esecuzione; a questo punto *Tp* puo' riprendere l'esecuzione (ritrovando gli stessi valori delle proprie variabili cosi' come li aveva lasciati).