# Elementi di Sistemi Operativi

Bioinformatica - Tiziano Villa

9 Gennaio 2009

Nome e Cognome:
Matricola:
Posta elettronica:

| problema | punti massimi | i tuoi punti |
|---|---|---|
| problema 1 | 8 | |
| problema 2 | 8 | |
| problema 3 | 7 | |
| problema 4 | 7 | |
| totale | 30 | |

1. Rispondere in modo preciso ma conciso alle seguenti domande.

   (a) Si descrivano succintamente i servizi di un sistema operativo relativamente alle comunicazioni e al rilevamento d'errori.

   Traccia di soluzione.

   Vedi Sez. 2.1 del libro di testo.

(b) Qual e' la funzione dell'interprete dei comandi ? E' meglio che sia incluso o separato dal nucleo ("kernel") del sistema operativo ? Si descriva una o piu' modalita' d'implementazione dei comandi, e si commenti rispetto alla facilita' d'introdurre nuovi comandi. Un utente potrebbe sviluppare un nuovo interprete dei comandi a partire dall'interfaccia di chiamate a sistema messe a disposizione dal sistema operativo ?

Traccia di soluzione.

L'interprete dei comandi legge dei comandi dall'utente o da un file di comandi e li esegue, di solito trasformandoli in una o piu' chiamate a sistema. E' meglio tenerlo separato dal nucleo del sistema operativo, per modificarlo piu' agevolmente. Vedi Sez. 2.2.1 del libro di testo. Un interprete dei comandi permette all'utente di creare e manipolare processi e di farli comunicare (attraverso "pipes" e "files"). Poiche' tutte queste funzioni sarebbero accessibili da un applicativo dell'utente mediante l'uso di chiamate a sistema, l'utente potrebbe sviluppare un nuovo interprete dei comandi.

(c) Si descriva il ruolo delle chiamate di sistema.
Traccia di soluzione.
Vedi Sez. 2.3, p. 44-46 del libro di testo.

(d) Quali chiamate di sistema devono essere eseguite dall'interprete dei comandi per far partire un nuovo processo in Unix ?

Traccia di soluzione.

In Unix, una chiamata a *fork* seguita da una chiamata ad *exec* fanno partire un nuovo processo. La chiamata a *fork* produce un clone del processo corrente, mentre la chiamata ad *exec* sovrappone (sovraimprime) al processo chiamante un nuovo processo con un eseguibile diverso.

2. Si consideri il seguente codice per risolvere il problema dei lettori e scrittori mediante semafori ($OKToRead$, $OKToWrite$, $Mutex$) e variabili di stato ($AR$ numero lettori attivi, $WR$ numero lettori in attesa, $AW$ numero scrittori attivi, $WW$ numero scrittori in attesa).

```
Semaphore OKToRead = 0; OKToWrite = 0; Mutex = 1;
AR = WR = AW = WW = 0;

Lettore {
  P(Mutex);
  if ((AW + WW) == 0) {
    V(OKToRead);
    AR = AR + 1;
  } else WR = WR + 1;
  V(Mutex);
  P(OKToRead);

  leggere i dati;

  P(Mutex);
  AR = AR -1;
  if (AR == 0 && WW > 0) {
    V(OKToWrite);
    AW = AW + 1;
    WW = WW - 1;
  }
  V(Mutex);
}
```

```
Scrittore {
  P(Mutex);
  if ((AW + AR + WW) == 0) {
    V(OKToWrite);
    AW = AW + 1;
  } else WW = WW + 1;
  V(Mutex);
  P(OKToWrite);

  scrivere i dati;

  P(Mutex);
  AW = AW -1;
  if (WW > 0) {
    V(OKToWrite);
    AW = AW + 1;
    WW = WW - 1;
  } else while (WR > 0) {
    V(OKToRead);
    AR = AR + 1;
    WR = WR - 1;
  }
  V(Mutex);
}
```

(a) Si analizzi il codice precedente spiegandone il funzionamento.

Traccia di soluzione.

- Piu' lettori possono accedere al codice contemporaneamente, ma gli scrittori devono avere accesso esclusivo.
- I lettori possono procedere solo se non ci sono scrittori attivi o in attesa.
- Gli scrittori possono procedere solo se non ci sono lettori attivi o scrittori attivi o in attesa.
- Solo un processo alla volta manipola le variabili di stato con il semaforo $Mutex$.

(b) In caso di conflitto tra lettori e scrittori chi ha la priorita' ? Perche' ?

Traccia di soluzione.

Gli scrittori hanno la priorita'. Ai lettori puo' essere negata la risorsa indefinitamente.

(c) E' necessario $WW$ nella condizione del primo $if$ dello $Scrittore$ ? Perche' ?

Traccia di soluzione.

No, se $WW \neq 0$ si deve avere $AW \neq 0$ oppure $AR \neq 0$.

(d) Puo' essere $OKToRead > 1$ ? Puo' essere $OKToWrite > 1$ ? Perche' ?

Traccia di soluzione.

Si alla prima, No alla seconda.

(e) Si puo' garantire che il primo scrittore che esegue $P(Mutex)$ sia anche il primo scrittore a scrivere ? Si risponda alla precedente domanda nel caso che non ci siano lettori attivi. Si argomentino le risposte in dettaglio.

Traccia di soluzione.

La risposta e' NO.

Se ci sono lettori attivi, tutti gli scrittori che arrivano sono messi in attesa in una lista e chi sara' il primo dipendera' dal meccanismo di selezione dei processi dalla lista degli scrittori in attesa (quando i lettori attivi avranno finito).

La risposta e' NO anche nel caso che non ci siano lettori attivi e la spiegazione e' piu sottile, come segue. Il primo scrittore puo' essere fermato dopo $V(Mutex)$, e un secondo scrittore puo' arrivare ad eseguire $P(OKToWrite)$ e proseguire senza problemi poiche' la variabile $OKToWrite$ e' stata abilitata con l'operazione $V(OKToWrite)$ dal primo scrittore, come mostrato in dettaglio di seguito.

Il primo scrittore esegue il seguente codice e poi supponiamo che sia sospeso esattamente dopo $V(Mutex)$:

```
Scrittore {
  P(Mutex);
  if ((AW + AR + WW) == 0) {
    V(OKToWrite);
    AW = AW + 1;
  } else WW = WW + 1;
  V(Mutex);
```

A questo punto si ha $AW = 1, WW = 0$.

Il secondo scrittore esegue il suo codice

```
Scrittore {
  P(Mutex);
  if ((AW + AR + WW) == 0) {
    V(OKToWrite);
    AW = AW + 1;
  } else WW = WW + 1;
  V(Mutex);
  P(OKToWrite);
```

vede che $OKToWrite = 1$ (grazie al primo scrittore), e prosegue accedendo in scrittura

```
  P(OKToWrite);

  scrivere i dati;
```

poi continua, decrementando $AW$ (incrementato dal primo scrittore), verificando se $WW > 0$ - il che e' vero perche' lo stesso secondo scrittore ha incrementato la variable $WW$ - e quindi incrementando $OKToWrite$ con $V(OKToWrite)$ e $AW$ (e decrementando $WW$).

```
  P(Mutex);
  AW = AW -1;
  if (WW > 0) {
    V(OKToWrite);
    AW = AW + 1;
    WW = WW - 1;
  } else while (WR > 0) {
    V(OKToRead);
    AR = AR + 1;
    WR = WR - 1;
  }
  V(Mutex);
}
```

Adesso con $OKToWrite = 1$ il primo scrittore (quando riattivato) puo' proseguire come scrittore attivo con il resto del suo codice.

```
  P(OKToWrite);

  scrivere i dati;

  P(Mutex);
  AW = AW -1;
  if (WW > 0) {
    V(OKToWrite);
    AW = AW + 1;
    WW = WW - 1;
```

```
  } else while (WR > 0) {
    V(OKToRead);
    AR = AR + 1;
    WR = WR - 1;
  }
  V(Mutex);
}
```

Si noti che il primo scrittore ha incrementato la variabile $AW$ a cui poi ”si riferisce” il secondo scrittore quando scrive; a sua volta il secondo scrittore ha incrementato la variabile $WW$ a cui poi ”si riferisce” il primo scrittore quando e' in attesa di scrivere (poi il secondo scrittore quando ha finito incrementa $AW$ e decrementa $WW$, il che permette al primo scrittore di scrivere).

3. Siano dati 5 processi con la durata della sequenza di operazioni della CPU espressa in millesecondi.

| Processo | Durata |
|---|---|
| P1 | 5 |
| P2 | 1 |
| P3 | 2 |
| P4 | 1 |
| P5 | 10 |

Si supponga che i processi siano arrivati nell'ordine $P1, P2, P3, P4, P5$ e che siano tutti presenti al tempo 0.

(a) i. Si disegni lo schema GANTT (come nel libro di testo) che illustri l'esecuzione di questi processi con l'algoritmo di schedulazione FCFS (First-Come,First-Served).
Traccia di soluzione.

```
P1 P1 P1 P1 P1 P2 P3 P3 P4 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5
X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
```

ii. Si calcoli il tempo di completamento di ciascun processo.
I tempi di completamento sono:
P1 5,
P2 6,
P3 8,
P4 9,
P5 19

iii. Si calcoli il tempo di attesa di ciascun processo.
Traccia di soluzione.
I tempi di attesa sono (tempo di completamento - durata):
P1 0,
P2 5,
P3 6,
P4 8,
P5 9

iv. Si calcoli il tempo di attesa medio tra tutti i processi.
Traccia di soluzione.
Tempo di attesa medio: (0+5+6+8+9)/5 = 5,6

(b) i. Si disegni lo schema GANTT (come nel libro di testo) che illustri l'esecuzione di questi processi con l'algoritmo di schedulazione SJF (Shortest Job First).
Traccia di soluzione.

```
P2 P4 P3 P3 P1 P1 P1 P1 P1 P5 P5 P5 P5 P5 P5 P5 P5 P5 P5
X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
```

ii. Si calcoli il tempo di completamento di ciascun processo.
I tempi di completamento sono:
P1 9,
P2 1,
P3 4,
P4 2,
P5 19

iii. Si calcoli il tempo di attesa di ciascun processo.
Traccia di soluzione.
I tempi di attesa sono (tempo di completamento - durata):
P1 4,
P2 0,
P3 2,
P4 1,
P5 9

iv. Si calcoli il tempo di attesa medio tra tutti i processi.
Traccia di soluzione.
Tempo di attesa medio: (4+0+2+1+9)/5 = 3,2

4. L'interazione con le unita' d'ingresso/uscita puo' avvenire tramite interrogazione ciclica ("polling") o interruzioni ("interrupt"). Si vuole studiare la convenienza di usare il sistema dell'interruzione per gestire l'interazione con il disco rigido.

   Si supponga che il costo aggiuntivo di ogni trasferimento dovuto all'interruzione sia pari a 500 cicli di orologio. Inoltre si supponga che il processore lavori con una frequenza di 500 MHz.

   Nell'ipotesi che il disco trasferisca dati in blocchi di 4 parole alla velocita' di 4 MB/s, e che il disco rigido trasferisca dati solo per il $5\%$ del tempo, si determini la frazione di tempo del processore consumato dal meccanismo dell'interruzione.

   Che cosa si puo' concludere sulla convenienza del meccanismo dell'interruzione nell'interagire con il disco rigido ?

   Traccia di soluzione.

   Sono necessarie tante interruzioni al secondo quanti sono i trasferimenti di blocchi di 4 parole al secondo, che sono uguali a 250 K al secondo (4 MB/secondo / 16 B per trasferimento = 250 K trasferimenti al secondo - si note che 4 parole = 16 B).

   I cicli di orologio per le interruzioni al secondo sono $250K \times 500 = 125 \times 10^3 K$ cicli al secondo.

   Ignorando la discrepanza tra le unita' di misura relativa alle basi e quindi assumendo che sia $1K = 1000$ (in realta' $1K = 1024$), la frazione di cicli del processore utilizzati dalle interruzioni e' data da

   $$125 \times 10^6 / 500 \times 10^6 = 25\%.$$

   Data l'ipotesi che il disco trasferisca dati solo per il $5\%$ del tempo, la frazione di cicli del processore utilizzati dalle interruzioni e' data da

   $$25\% \times 5\% = 0,0125 = 1,25\%.$$

   Si conclude che sarebbe conveniente usare il meccanismo delle interruzioni per interagire con il disco. Si noti che il vantaggio si deve all'assenza di costo aggiuntivo quando il disco non e' effettivamente impegnato in trasferimenti.