Sistemi Operativi
19 settembre 2013
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.
Gli esercizi e le domande marcate con l'asterisco (*) devono essere svolti soltanto da chi ha in piano di studi l'esame di Sistemi Operativi da 9 o 12 CFU.

1. Si discuta ciascuna delle seguenti affermazioni.
   (a) Si può verificare starvation in un SO che utilizza uno scheduling basato su priorità, ma non in un SO che utilizza uno scheduling Round Robin.
   (b) (*) In un sistema lettori-scrittori che favorisce i lettori, i processi scrittori possono essere scavalcati da processi lettori che arrivano quando altri processi lettori stanno leggendo i dati condivisi.
   (c) (*) Si può verificare un deadlock nel problema dei filosofi a cena anche se un filosofo (solo uno) può mangiare con una forchetta.
   (d) Un processo può subire una starvation all'entrata di una sezione critica se l'implementazione della sezione critica non soddisfa la condizione di attesa limitata.

   Risposta:
   (a) Vero: per definizione lo scheduling RR garantisce che se ci sono n processi in coda ready e il quanto è q, allora nessun processo attende più di (n - 1)q.
   (b) Vero: il sistema fa sì che nessun processo scrittore possa accedere i dati fintanto che questi sono letti dai processi lettori. Questi ultimi invece possono tranquillamente accedere ai dati in presenza di altri lettori.
   (c) Falso: ci sono tante forchette quante i filosofi. Nel caso peggiore ogni filosofo riesce ad acquisire soltanto una forchetta, ma uno di essi riuscirà comunque a mangiare, deponendola alla fine (e lasciando così la possibilità ad un altro filosofo di mangiare). Se invece il filosofo che può mangiare con una sola forchetta non riesce ad acquisirne nessuna, allora ci sarà almeno un altro filosofo che riesce ad acquisire entrambe le forchette.
   (d) Vero: tale condizione infatti è proprio quella che garantisce un tempo di attesa limitato prima dell'entrata nella sezione critica.

2. Si descriva il funzionamento di un semaforo mutex e si mostri come può essere utilizzato per implementare una sezione critica.

   Risposta: Un mutex è un particolare semaforo che può assumere soltanto due valori: 0 e 1. In particolare, la up pone il mutex a 1 e risveglia eventualmente uno dei processi che erano in attesa sul mutex stesso. La down invece imposta il mutex a 0, se il valore era 1, mentre pone il chiamante in attesa se il valore era già 0. In questo modo l'implementazione di una sezione critica tramite mutex è molto facile:

```
while (TRUE) {
down(mutex);
// sezione critica
up(mutex);
// sezione non critica
}
```

3. Si consideri l'algoritmo di Peterson:

```
#define FALSE  0
#define TRUE   1
#define N      2                    /* number of processes */

int turn;                           /* whose turn is it? */
int interested[N];                  /* all values initially 0 (FALSE) */

void enter_region(int process);     /* process is 0 or 1 */
{
    int other;                      /* number of the other process */

    other = 1 - process;            /* the opposite of process */
    interested[process] = TRUE;     /* show that you are interested */
    turn = process;                 /* set flag */
    while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)      /* process: who is leaving */
{
    interested[process] = FALSE;    /* indicate departure from critical region */
}
```

All'algoritmo viene effettuata la seguente modifica: in entrambi i processi, le istruzioni `interested[process] = TRUE` della enter_region `interested[process] = FALSE` della leave_region vengono scambiate. Quali proprietà dell'implementazione delle sezioni critiche sono violate dal sistema risultante?

Risposta: Si perdono le seguenti proprietà:

- mutua esclusione: all'inizio sia `interested[0]` che `interested[1]` sono impostati a `FALSE`; quindi il primo processo che intende accedere alla sezione critica può farlo senza ostacoli. Tuttavia, se in seguito (mentre il primo processo è ancora in esecuzione nella sua sezione critica) anche l'altro processo vuole entrare nella sua sezione critica, può farlo senza problemi (violando così la mutua esclusione).
- progresso: se entrambi i processi sono usciti dalle rispettive sezioni critiche, sia `interested[0]` che `interested[1]` sono impostate a `TRUE`. Supponendo che, mentre P[1-i] è in esecuzione nella sua *remainder section*, P[i] voglia entrare nuovamente nella sua sezione critica, esso dovrà attendere che P[1-i] esca dalla sua remainder section ed esegua nuovamente la propria enter_region (violando la condizione di progresso).
- attesa limitata: se P[i] vuole entrare e si blocca sul `while` deve attendere che P[1-i] esegua di nuovo la propria enter_region, impostando `interested[1-i]` a `FALSE`. A questo punto però non è detto che P[i] possa immediatamente entrare nella propria sezione critica, dato che anche P[1-i] è sbloccato e può continuare la propria esecuzione se l'algoritmo di scheduling di breve termine lo consente, ritardando l'ingresso di P[i] in modo indefinito.

4. (*) Si supponga che in un sistema siano presenti 5 processi, P_0, P_1, P_2, P_3, P_4, un insieme di risorse di 4 tipi diversi, A, B, C, D, e di trovarsi nella seguente configurazione:

	Risorse allocate (**C**)				Risorse massime (**Max**)			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**
$\mathbf{P_0}$	4	$x-1$	3	2	6	4	5	6
$\mathbf{P_1}$	8	0	$y-2$	2	10	7	6	8
$\mathbf{P_2}$	4	0	0	0	6	2	0	8
$\mathbf{P_3}$	0	0	3	2	0	3	4	2
$\mathbf{P_4}$	2	1	$z+1$	4	9	1	6	9

Risorse disponibili (**A**)

A	**B**	**C**	**D**
2	2	10	4

Determinare gli intervalli dei valori interi di x, y, z per i quali:

(a) il sistema si trova in uno stato sicuro, elencando eventuali sequenze sicure;

(b) la richiesta attuale di P_2 $(2, 0, 0, 2)$ può essere soddisfatta.

Risposta:

(a) Dalla constatazione che il minimo valore per ogni risorsa allocata è 0 e dal fatto che il valore massimo non può essere superiore a quanto indicato dalla matrice **Max**, abbiamo i seguenti vincoli:

$$0 \leq x - 1 \leq 4$$
$$0 \leq y - 2 \leq 6$$
$$0 \leq z + 1 \leq 6$$

ovvero:

$$1 \leq x \leq 5$$
$$2 \leq y \leq 8$$
$$-1 \leq z \leq 5$$

Calcoliamo ora la matrice delle richieste:

	Richieste (**R**)		
A	**B**	**C**	**D**
2	$5 - x$	2	4
2	7	$8 - y$	6
2	2	0	8
0	3	1	0
7	0	$5 - z$	5

quindi l'unica riga di **R** a poter essere minore od uguale al vettore **A** è la prima, da cui l'ulteriore vincolo $5 - x \leq 2$, ovvero, $x \geq 3$. Quindi, complessivamente deve essere $3 \leq x \leq 5$. Mandando quindi in esecuzione $\mathbf{P_0}$, esso restituirà le risorse utilizzate ed il nuovo valore di **A** sarà $(6, x + 1, 13, 6)$. A questo punto possono essere eseguiti prima $\mathbf{P_3}$ e poi $\mathbf{P_2}$, ottenendo $\mathbf{A} = (10, x + 1, 16, 8)$. $\mathbf{P_1}$ non può essere eseguito perché dovrebbe essere $7 \leq x + 1$, ovvero, $x \geq 6$. Quindi può essere eseguito $\mathbf{P_4}$ a patto che $0 \leq x + 1$, ovvero, $x \geq -1$ (il che, per i vincoli precedenti, è banalmente vero) e $5 - z \leq 16$, ovvero, $z \geq -11$ (condizione soddisfatta dato che $-1 \leq z \leq 5$): il nuovo valore di **A** diventa $(12, x + 2, z + 17, 12)$. L'ultimo processo, $\mathbf{P_1}$, può essere eseguito se $8 - y \leq z + 17$, ovvero, $z \geq -9 - y$ (vincolo soddisfatto dato che $-1 \leq z \leq 5$ e $2 \leq y \leq 8$) e se $7 \leq x + 2$, ovvero se $x \geq 5$ (che, in base ai vincoli precedenti su x, equivale ad imporre $x = 5$).

Quindi gli intervalli di x, y e z affinché il sistema si trovi in uno stato sicuro sono i seguenti:

$$x = 5$$
$$2 \leq y \leq 8$$
$$-1 \leq z \leq 5$$

(b) Se la richiesta attuale di P_2 $(2, 0, 0, 2)$ venisse soddisfatta, il nuovo valore di **A** diverrebbe $(0, 2, 10, 2)$, mentre la matrice delle richieste sarebbe la seguente:

	Richieste (**R**)		
A	**B**	**C**	**D**
2	$5 - x$	2	4
2	7	$8 - y$	6
0	2	0	6
0	3	1	0
7	0	$5 - z$	5

Quindi, non essendoci nessuna riga di **R** minore od uguale al vettore **A**, il sistema si troverebbe in uno stato non sicuro. La richiesta va pertanto rifiutata.

5. Si illustrino brevemente i modi per far comunicare la CPU con i controller dei dispositivi.

Risposta: Ci sono essenzialmente due modi per far comunicare la CPU con i controller dei dispositivi:

- utilizzare un insieme di istruzioni di I/O dedicate: facili da controllare, ma impossibili da usare a livello utente (si deve passare sempre per il kernel);
- utilizzare il cosiddetto I/O mappato in memoria: una parte dello spazio indirizzi è collegato ai registri del controller (questa tecnica è più efficiente e flessibile anche se delega il controllo alle tecniche di gestione della memoria come, ad esempio, la paginazione).

Come variante della seconda modalità, esiste anche l'I/O separato in memoria: un segmento a parte distinto dallo spazio indirizzi è collegato ai registri del controller. Inoltre è possibile anche utilizzare approcci ibridi facendo uso, a seconda dei casi, di istruzioni dedicate oppure della mappatura in memoria.

6. (a) Per quanto riguarda i tipi dei file che assunzioni fa UNIX?
   (b) Cosa sono i *magic number* e l'utility file in UNIX?

   Risposta:

   (a) UNIX non fa nessuna assunzione sui tipi dei file, ovvero, non forza nessun tipo di file a livello di sistema operativo: non ci sono metadati che mantengano questa informazione. Sono le applicazioni a gestire i rispettivi tipi di file.

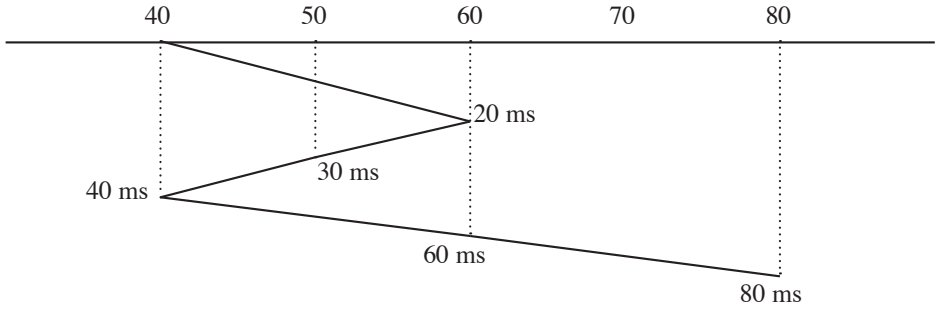
   (b) Tuttavia in UNIX esiste l'utility `file` che, grazie ad un opportuno database, permette di "indovinare" il tipo di un file, ispezionandone i primi byte (ovvero, i cosiddetti *magic numbers*) e cercando una corrispondenza nella base di dati suddetta.

7. Si consideri un disco gestito con politica SSTF. Inizialmente la testina è posizionata sul cilindro 40; lo spostamento ad una traccia adiacente richiede 1 ms. Al driver di tale disco arrivano richieste per i cilindri 80, 50, 40, 60, rispettivamente agli istanti 0 ms, 20 ms, 30 ms, 50 ms. Si trascuri il tempo di latenza.

   1. In quale ordine vengono servite le richieste?
   2. Il tempo di attesa di una richiesta è il tempo che intercorre dal momento in cui è sottoposta al driver a quando viene effettivamente servita. Qual è il tempo di attesa medio per le quattro richieste in oggetto?

   Risposta:

   1. Le richieste vengono servite nell'ordine 50, 40, 60, 80, come risulta dal seguente diagramma:



   2. Il tempo di attesa medio per le quattro richieste in oggetto è dato da $\frac{(30-20)+(40-30)+(60-50)+(80-0)}{4} = \frac{(10+10+10+80)}{4} = \frac{110}{4} = 27,5\ ms$

Punteggi:

- 9/12 CFU: 1a (2 pt.), 1b (2 pt.), 1c (2 pt.), 1d (2 pt.), 2 (4 pt.), 3 (4 pt.), 4a (3 pt.), 4b (3 pt.), 5 (2 pt.), 6a (2 pt.), 6b (2 pt.), 7.1 (2 pt.), 7.2 (1 pt.)
- 6 CFU: 1a (3 pt.), 1d (3 pt.), 2 (5 pt.), 3 (5 pt.), 5 (3 pt.), 6a (3 pt.), 6b (3 pt.), 7.1 (3 pt.), 7.2 (2 pt.)