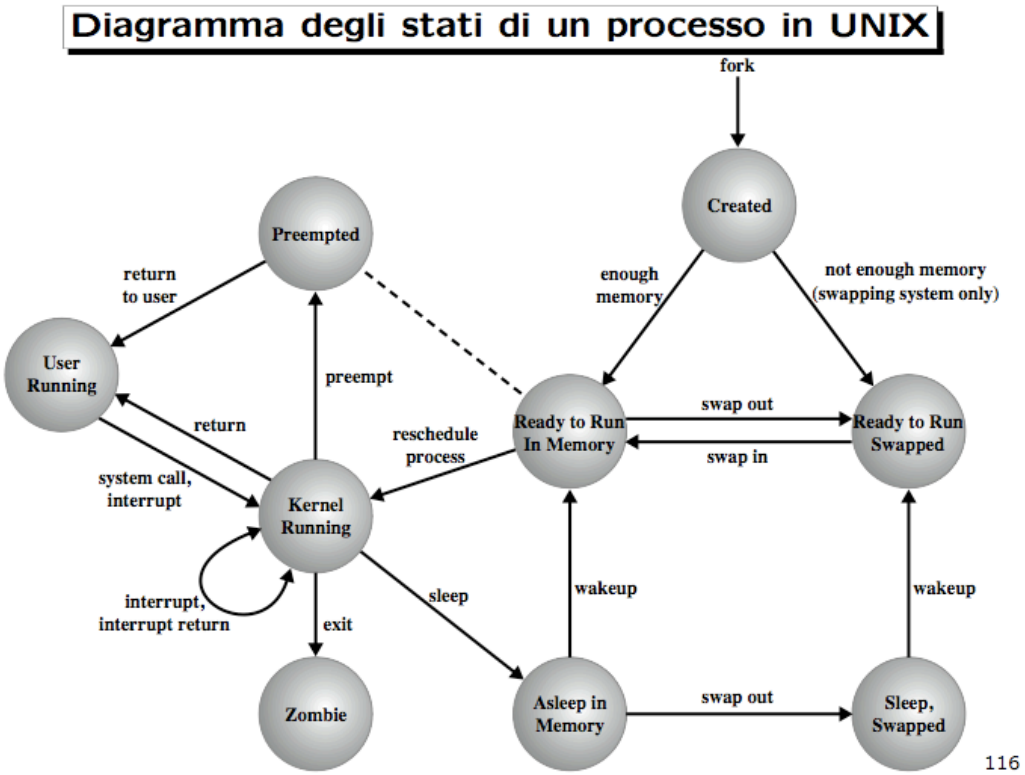
Sistemi Operativi
20 giugno 2013
Compito B

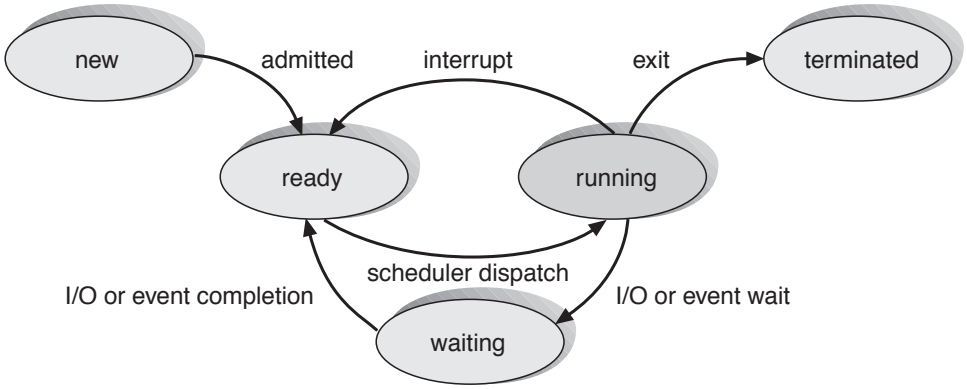
Si risponda ai seguenti quesiti, giustificando le risposte.
Gli esercizi e le domande marcate con l'asterisco (*) devono essere svolti soltanto da chi ha in piano di studi l'esame di Sistemi Operativi da 9 o 12 CFU.

1. (a) Si descriva il diagramma degli stati di un processo in un sistema operativo generico.
   (b) Si diano esempi di eventi che provocano il passaggio da uno stato ad un altro stato.
   (c) * La seguente figura mostra il diagramma degli stati di un processo in Unix. Si spieghino le differenze con il diagramma generale discusso ai punti precedenti.



Risposta:

(a) Diagramma degli stati:



(b) Esempi di eventi che provocano il passaggio da uno stato ad un altro stato:

- Da new a ready: un nuovo processo viene allocato in coda ready.
- Da running a ready: in caso di scheduling della CPU con prelazione, un processo che passa da stato new in stato ready oppure da stato waiting a stato ready (per es. perché ha terminato un'operazione di I/O), può provocare il passaggio di un processo a minor priorità da stato running a stato ready.
- Da running a waiting: richiesta di esecuzione di un'operazione di I/O.
- Da waiting a ready: l'operazione di I/O è terminata.

- Da ready a running: un processo che passa da stato running a stato terminated o a stato waiting (in attesa di un evento, per es. il completamento di un'operazione di I/O), provoca il passaggio di un processo da ready a running. Questo tipo di passaggio di stato può avvenire anche per intervento dello scheduler di breve termine (per esempio in caso di schedulazione RR su interrupt del timer di sistema, qualora un processo abbia terminato il proprio quanto e rilasci la CPU ad un altro processo in coda ready).
- Da running a terminated: un processo termina la propria esecuzione.

(c) Nel caso del diagramma degli stati di un processo in Unix, le differenze con il diagramma generale discusso ai punti precedenti emergono dall'analisi degli stati:

- User running: esecuzione in modo utente,
- Kernel running: esecuzione in modo kernel,
- Ready to run, in memory: pronto per andare in esecuzione,
- Asleep in memory: in attesa di un evento; processo in memoria,
- Ready to run, swapped: eseguibile, ma swappato su disco,
- Sleeping, swapped: in attesa di un evento; processo swappato,
- Preempted: il kernel lo blocca per mandare un altro processo,
- Zombie: il processo non esiste più, si attende che il padre riceva l'informazione dello stato di ritorno.

In particolare il diagramma evidenzia l'esecuzione di codice in due modalità (utente e kernel) ed il fatto che i processi possono essere soggetti a swapping (per ridurre il grado di multiprogrammazione) ad opera dello scheduler di medio termine.

2. (a) Cinque processi sono in coda ready. I loro tempi di esecuzione previsti sono 10, 7, 4, 6 e x. In quale ordine dovrebbero essere eseguiti i processi per minimizzare il tempo di risposta medio? Perché?

(b) * Si consideri un sistema con scheduling SJF. Stimare la durata del prossimo CPU burst di un processo con burst precedenti, dal più vecchio al più recente, di 30, 10, 30, 5 ms. Si consideri $\alpha = 0,5$.

Risposta:

(a) Dato che il tempo di risposta è quello che intercorre fra il momento in cui un processo arriva in coda ready ed il momento in cui viene schedulato per l'esecuzione per la prima volta, per minimizzare il tempo di risposta medio, i cinque processi devono essere eseguiti in ordine crescente rispetto ai loro CPU burst, ovvero:

- x, 4, 6, 7, 10 se x$\leq$ 4;
- 4, x, 6, 7, 10 se x=5;
- 4, 6, 7, x, 10 se 7 $\leq$x$\leq$ 10;
- 4, 6, 7, 10, x se x$>$ 10.

(b) Supponendo di partire con $\tau_0 = 0$, abbiamo: $\tau_1 = 0,5 \cdot 30 + 0,5 \cdot \tau_0 = 15$, $\tau_2 = 0,5 \cdot 10 + 0,5 \cdot \tau_1 = 5 + 7,5 = 12,5$, $\tau_3 = 0,5 \cdot 30 + 0,5 \cdot \tau_2 = 15 + 6,25 = 21,25$ e $\tau_4 = 0,5 \cdot 5 + 0,5 \cdot \tau_3 = 2,5 + 10,625 = 13,125$ ms che rappresenta la stima del prossimo burst.

3. Si consideri la soluzione al problema della sezione critica per due processi realizzata da Dekker. I due processi condividono le seguenti variabili:

```
int flag[2]={0, 0}; int turn=0; /* oppure 1 */
```

La struttura di P_i ($i = 0$ o 1) con P_j ($j = 1 - i$) è definita come segue:

```
do {
  /* entry section begins */
  flag[i]=1;

  while( flag[j] ) {
```

```
    if(turn == j) {
      flag[i]=0;
      while( turn == j );
      flag[i]=1;
    }

  }
  /* entry section ends */

  /* critical section */

  /* exit section begins */
  turn=j;
  flag[i]=0;
  /* exit section ends */

  /* remainder section */
} while true;
```

Si provi che l'algoritmo soddisfa i tre requisiti relativi al problema della sezione critica.

Risposta: bisogna verificare che l'algoritmo soddisfa i tre requisiti relativi al problema della sezione critica.

1. Come prima cosa verifichiamo che i due processi P_i e P_j non possano accedere simultaneamente alla regione critica. Supponiamo che entrambi i processi abbiano dichiarato il loro interesse ad entrare nella sezione critica impostando il proprio flag a 1 (`flag[i]` per P_i e `flag[j]` per P_j); entrambi quindi eseguiranno le istruzioni all'interno dei cicli while esterni. A questo punto il valore della variabile `turn` consente di sbloccare uno dei due. Supponiamo che valga `0` (quindi sia a favore di P_i): P_j entrerà nel corpo dell'if, azzerando il proprio flag e ponendosi in attesa attiva nel ciclo while interno. P_i accorgendosi che il flag di P_j è stato azzerato entra nella regione critica (uscendo dal while esterno). A questo punto soltanto P_i potrà trovarsi nella regione critica e vi rimarrà (mantenendo bloccato in attesa attiva P_j) fintanto che non eseguirà il codice dell'*exit section* impostando il valore di `turn` a `1` ed azzerando il proprio flag. La prima operazione fa uscire P_j dalla situazione di attesa attiva (while interno), riattivando il proprio flag, mentre la seconda fa effettivamente entrare P_j nella sezione critica. Se a questo punto P_i tentasse di rientrare nella sezione critica si bloccherebbe sul while interno (trovando attivato il flag di P_j e la variabile `turn` impostata a `1`).
2. La seconda condizione è banalmente verificata in quanto le decisioni su quale processo possa entrare nella sezione critica vengono prese soltanto nella *entry section* e nella *exit section* (tramite la modifica dei flag e della variabile `turn`).
3. Un processo al di fuori della sezione critica non deve attendere indefinitamente per entrarvi, dato che:
   - se l'altro processo non è interessato ad entrare nella sezione critica il suo flag sarà azzerato e l'accesso immediato;
   - se l'altro processo sta eseguendo nella sezione critica, al momento della sua uscita la variabile `turn` verrà aggiornata per dare la precedenza al processo attualmente all'esterno della sezione critica.

4. * Si consideri la seguente situazione, dove P_0, P_1, P_2 sono tre processi in esecuzione, C è la matrice delle risorse correntemente allocate, Max è la matrice del numero massimo di risorse assegnabili ad ogni processo e A è il vettore delle risorse disponibili:

	C			Max		
	A	B	C	A	B	C
P_0	1	2	0	1	5	1
P_1	0	1	0	2	5	2
P_2	2	3	0	2	4	2

Available (A)		
A	B	C
1	5	x

(a) Calcolare la matrice R delle richieste.

(b) Determinare il minimo valore di x tale che il sistema si trovi in uno stato sicuro.

Risposta:

(a) La matrice R delle richieste è data dalla differenza $Max - C$:

	R	
A	B	C
0	3	1
2	4	2
0	1	2

(b) Se $x = 0$, allora non esiste nessuna riga R_i tale che $R_i \leq A$; quindi il sistema si trova in uno stato di deadlock. Se $x = 1$, allora l'unica riga di R minore o uguale a A è la prima. Quindi possiamo eseguire P_0 che, una volta terminato, restituisce le risorse ad esso allocate aggiornando A al valore $(2, 7, 1)$. A questo punto non esiste alcuna riga di R minore o uguale al vettore A e quindi il sistema è in stato di deadlock.

Il valore minimo di x per cui lo stato risulta sicuro è 2; infatti in questo caso esiste la sequenza sicura $\langle P_0, P_1, P_2 \rangle$. Dapprima si esegue P_0, generando il valore $(2, 7, 2)$ di A, poi si esegue P_1 portando A al valore $(2, 8, 2)$. A questo punto si conclude la sequenza eseguendo P_2 e generando il valore finale di A, ovvero, $(4, 11, 2)$.

5. * Si spieghi come funziona l'allocazione contigua, specificando lo schema di traduzione dell'indirizzo logico in indirizzo fisico. Questa soluzione soffre di frammentazione esterna? (in caso di risposta affermativa spiegare perché).

   Risposta: lo schema di allocazione contigua prevede che ogni file occupi un insieme di blocchi contigui sul disco. È quindi molto semplice da implementare in quanto basta conoscere l'indirizzo del blocco iniziale e la lunghezza per accedere ai blocchi del file. La traduzione da indirizzo logico a fisico (indicando con B la dimensione del blocco e LA l'indirizzo logico) avviene eseguendo la divisione intera di LA per B: il quoziente va sommato all'indirizzo del blocco iniziale per determinare l'indirizzo del blocco da accedere, mentre il resto indicherà l'offset all'interno di quest'ultimo.

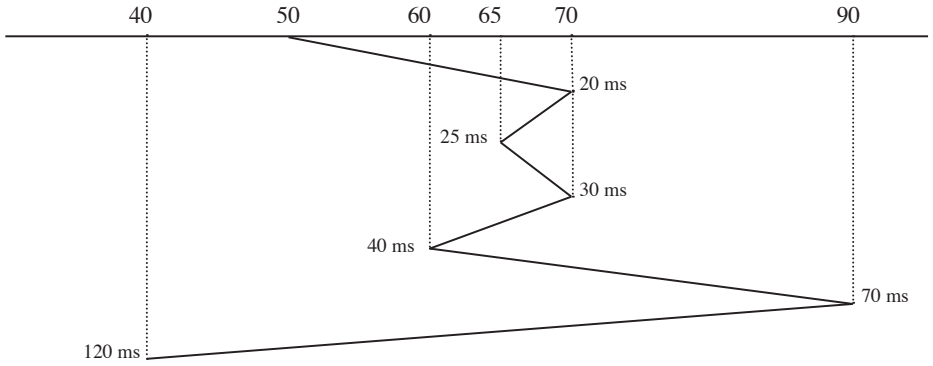
   L'allocazione contigua soffre di frammentazione esterna in quanto, dovendo i file occupare una sequenza di blocchi contigui, in seguito a creazioni e cancellazioni di file si può avere una situazione in cui lo spazio libero totale su disco è sufficiente a soddisfare una nuova richiesta, ma non è contiguo, ovvero, i "buchi" di blocchi liberi contigui sono troppo piccoli per la richiesta in oggetto.

6. Si consideri un disco gestito con politica SSTF. Inizialmente la testina è posizionata sul cilindro 50; lo spostamento ad una traccia adiacente richiede 1 ms. Al driver di tale disco arrivano richieste per i cilindri 90, 65, 60, 40, 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 soddisfatte nell'ordine: 65, 60, 90, 40, come risulta dal seguente diagramma:



2. Il tempo di attesa medio per le quattro richieste in oggetto è $\frac{(25-20)+(40-30)+(70-0)+(120-50)}{4} = \frac{5+10+70+70}{4} = \frac{155}{4} = 38,75\ ms$.

Punteggi:

- 9/12 CFU: 1a (2 pt.), 1b (2 pt.), 1c (2 pt.), 2a (2 pt.), 2b (2 pt.), 3 (4 pt.), 4a (3 pt.), 4b (5 pt.), 5 (4 pt.), 6.1 (4 pt.), 6.2 (2 pt.)
- 6 CFU: 1a (4 pt.), 1b (4 pt.), 2a (5 pt.), 3 (8 pt.), 6.1 (4 pt.), 6.2 (2 pt.)[1]

[1] La valutazione parte da un punteggio minimo (base) di 3 punti.