Possibili domande Sistemi Operativi

Categoria vero/falso A

1. Se non vi sono percorsi chiusi in un grafo di allocazione allora non vi è situazione di stallo **V**
2. La politica di scheduling round robin minimizza il tempo medio di attesa dei processi **F**
3. Se un processo è in blocco da 10 ms significa che 10 ms fa ha eseguito una system call **V**
4. Ogni interrupt può essere associato ad un processo che ha richiesto una operazione di I/O **F**
5. Pwd è un comando GNU/Linux per modificare la password **F**
   Una system call generata da un processo utente viene gestita in modalità utente
6. Un interrupt viene gestito in modalità utente **F**
7. Il meccanismo dei semafori consente forme più generali di sincronizzazione tra processi rispetto alla mutua esclusione **V**
8. Un processo per creare un nuovo processo deve fare una system call **V**
9. l'attesa indefinita (starvation) è un caso particolare dello stallo **V**
10. Lo switch di un processo utente avviene sempre contestualmente a 2 mode switch (utente->kernel, kernel->utente) **V**
11. Un interrupt viene gestito in modalità kernel **V**
12. Il process switch può avvenire sia in modalità kernel che in modalità utente **V**
13. Ogni interrupt può essere associato ad un processo che ha richiesto una operazione di I/O **F**

Categoria risposte multiple B

1. Un semaforo contatore può:
   [A] assumere valori interi arbitrari
   [B] assicurare l'accesso in mutua esclusione ad una risorsa condivisa
   [C] servire alla realizzazione di una struttura monitor
   [D]consentire l'accesso simultaneo di più processi ad una risorsa condivisa, fino al limite fissato dal valore di inizializzazione del contatore

2. Quale tra le seguenti politiche di ordinamento, in generale minimizza il tempo medio di turn-around dei processi:
   [A] FCFS
   [B] Round-Robin con valutazione dell'attributo di priorità dei processi
   [C] Round-Robin senza valutazione dell'attributo di priorità dei processi
   [D] Shortest Job First

3. Un semaforo binario può:
   [A] assumere solo valori pari o discreti
   [B] gestire solo l'accesso a due risorse condivise
   [C] gestire solo le richieste di accesso provenienti da due processi
   [D] assumere solo i valori 0 e 1, con essi denotando "risorsa occupata" e "risorsa libera".

4. Quale tra le seguenti affermazioni, fatte osservando un grafo di allocazione delle risorse, è certamente vera in generale:
   [A] se vi sono percorsi chiusi allora vi è situazione di stallo
   [B] se non vi sono percorsi chiusi allora non vi è situazione di stallo

[C] se in un percorso chiuso rilevato si trovano solo risorse a molteplicità unaria, occorre analizzare il caso per decidere
[D] nessuna delle precedenti tre possibili risposte.

5. Quale tra le seguenti affermazioni è corretta in relazione alla politica di ordinamento processi "FCFS senza valutazione dell'attributo di priorità":
[A] il tempo di attesa è sempre maggiore del tempo di risposta
[B] il tempo di attesa è sempre minore del tempo di risposta
[C] il tempo di attesa è sempre uguale al tempo di risposta
[D] il tempo di attesa ed il tempo di risposta non hanno alcun legame prefissato.
(Si consideri il tempo risposta come il tempo che intercorre dall'attivazione del processo fino alla sua prima esecuzione sul processore)

6. Se un processo è in blocco (in coda di attesa) da 10 ms significa che 10 ms fa ha eseguito:
[A] nessuno dei seguenti
[B] una fork
[C] un context switch
[D] una system call

7. In quale tra i seguenti sistemi operativi è più conveniente l'utilizzo di Inverted Page Tables:
[A] nessuno dei seguenti, il vantaggio è pari per tutti
[B] sistemi a 16 bit
[C] sistemi a 32 bit
[D] sistemi a 64 bit

8. Una system call bloccante causa sempre un context switch:
[A] Sempre
[B] Mai
[C] Sì ma solo se la macchina ha più di un processore
[D] Sì ma solo se c'è qualche altro processo attivo

9. Quale tra i seguenti costituisce un criterio valido di valutazione di una politica di ordinamento di processi:
[A] la capacità di trattare anche processi di lunga durata
[**B] il numero di processi completati per unità di tempo**
[C] il numero di processi in esecuzione per unità di tempo
[D] il numero di processi in attesa di essere eseguiti.

10. Quale tra le seguenti affermazioni concernenti la politica di ordinamento Round-Robin è corretta:
[A] il tempo di attesa di un processo è sempre maggiore o uguale del suo tempo di risposta
[B] il tempo di attesa di un processo è sempre minore o uguale del suo tempo di risposta
[C] il tempo di attesa di un processo è sempre uguale al suo tempo di risposta
[D] il tempo di attesa di un processo e il suo tempo di risposta non hanno alcun legame prefissato.

11. In un confronto prestazionale tra hard link e simbolic link:

1. **gli HL sono da preferire perché velocizzano gli accessi ai file**
2. i SL sono da preferire perché assicurano la singolarità dell'associazione tra directory e i-node
3. i due sono sostanzialmente indistinguibili
4. i SL hanno prestazioni superiori perché impiegano meno spazio nella directory

12. Quale tra le seguenti politiche di ordinamento, in generale minimizza il tempo medio di attesa dei processi:

1. FCFS
2. Round Robin con valutazione dell'attributo di priorità dei processi
3. Round Robin senza valutazione dell'attributo di priorità dei processi
4. **SJF**

Categoria risposte aperte C

1. Elencare le varie politiche di scheduling dei processi. Per ognuna, si discuta in 2-3 righe l'algoritmo alla base del loro funzionamento. (Es. "Scheduling a Lotteria: il prossimo processo ad andare in esecuzione è scelto a caso, con un generatore di numeri random, tra quelli presenti in coda di pronti e una volta in esecuzione vi rimane fino alla sua terminazione, senza poter essere prerilasciato").

2. Considerando le politiche di ordinamento FCFS, RR e SRTN è possibile dire che qualcuna di esse tende a ridurre o addirittura minimizzare, rispetto alle altre politiche, qualcuna delle seguenti metriche: tempo medio di risposta, tempo medio di attesa, tempo medio di turn around? (Specificare e Commentare anche tenendo conto del fatto che "ridurre" e "minimizzare" sono termini con un significato ben preciso). RR minimizza il tempo di risposta se ha un quanto di tempo piccolo, fcfs non minimizza niene, SRTN minimizza tempo di attesa e tempo di turn_around

3. Si consideri la politica di scheduling Round Robin di quanto q e si supponga che in un sistema ci siano N processi interattivi tutti con lo stesso comportamento. Ciascuna interazione dà luogo ad un CPU burst che richiede la CPU per un tempo c.
[A] Se c < q quanto tempo aspetta al più un processo in coda ready prima di ottenere la CPU?
[B] Se c < q quanto tempo aspetta al più l'utente prima che la CPU finisca di elaborare l'interazione?
[C] Se c > q quanto tempo aspetta l'utente prima di iniziare l'ultimo quanto di interazione per l'ultimo processo rimasto ancora attivo? (Si consideri che c può essere espresso come c = aq + b; ovvero a quanti di tempo + b<c tempo nell'ultimo quanto)
[2.A]Un processo aspetta (N-1)c per ottenere la CPU.
[2.B]L'utente aspetta (N-1)c+c= Nc
[2.C]L'utente aspetta Nqa+ (N-1)b

4. E' noto che esistano 4 condizioni che concorrono all'insorgere di situazioni di stallo. Lo studente le elenchi illustrandole brevemente, ovvero in non più di due/tre righe ciascuna.

5. Illustrare con un diagramma come quello visto a lezione gli stati in cui può trovarsi un processo e le transizioni tra essi. (Presentare solo il diagramma/figura, con i nomi degli stati e delle transizioni; non sono necessarie ulteriori spiegazioni).

6. In quale tra i seguenti sistemi operativi è più conveniente l'utilizzo di Inverted Page Tables:
1. nessuno dei seguenti, il vantaggio è pari per tutti
2. sistemi a 16 bit

3. sistemi a 32 bit

4. sistemi a 64 bit

Motivare BREVEMENTE la risposta.

7. un file risenta della contiguità con cui i blocchi di tale file sono disposti sul disco fisso.

8. Si consideri un sistema con un solo processo attivo, il quale sta eseguendo la seguente porzione di codice:

pid_1 = fork ();
pid_2 = fork ();
pid_3 = fork ();

Quanti processi risulteranno attivi alla fine dell'esecuzione di queste tre line di codice? (Si motivi la risposta) 8 processi, la prima fork ne crea 2 la seconda 4 la terza 8

9. Si elenchino, senza spiegarle, le condizioni relative al verificarsi dello stallo (deadlock):
Lo stallo avviene in presenza di: accesso esclusivo ad una risorsa, accumulo di risorse, attesa circolare e affidamento ad un processo la decisione di prerilascio.

Categoria esercizi D

1.

Quesito 4: Cinque processi *batch*, identificati dalle lettere A, B, C, D, E, arrivano all'elaboratore agli istanti 0, 1, 2, 6, 7 rispettivamente. Essi hanno un tempo di esecuzione stimato di 3, 7, 2, 3, 1 unità di tempo rispettivamente e priorità 3, 5, 2, 4, 1 rispettivamente (dove 5 è la massima priorità e 0 è la minima). Per ognuna delle seguenti politiche di ordinamento:

A) *Round Robin* (divisione di tempo, con priorità, senza prerilascio per priorità, e con quanto di tempo di ampiezza 2)

B) *Fair Priority Scheduling*
Per evitare attesa infinita la politica di *Fair Priority Scheduling* prevede che, a seguito di due unità di tempo **consecutive** di esecuzione, la priorità del processo in esecuzione scenda di un punto. (*Esempio 1.* Se il processo è in esecuzione per 4 unità di tempo consecutive, la priorità di tale processo scende di 1 punto dopo le prime due unità temporali e di 1 altro punto dopo le ultime due unità temporali.) (*Esempio 2.* Se un processo è in esecuzione per 3 unità di tempo consecutivamente, la priorità di tale processo scende di 1 punto dopo le prime due unità temporali e basta; l'altra unità temporale di esecuzione non concorre in alcun modo, nemmeno successivamente, a far decrementare la priorità del processo in considerazione.) Infine, la priorità di un processo non risale mai e non può scendere sotto lo zero.

Determinare, trascurando i ritardi dovuti allo scambio di contesto: (i) il tempo medio di risposta; (ii) il tempo medio di attesa; (iii) il tempo medio di *turn around*.
Nel caso di arrivi simultanei di processi allo stato di pronto, fatta salva l'eventuale considerazione del rispettivo valore di priorità, si dia la precedenza ai processi usciti dallo stato di esecuzione rispetto a quelli appena arrivati.
Nel caso di due processi aventi la stessa priorità, di cui uno in esecuzione, si dia la precedenza a quello in esecuzione.

[4.A]: *RR* (divisione di tempo, con priorità, senza prerilascio per priorità, e con quanto di tempo di ampiezza 2)

Proc. A																								
Proc. B																								
Proc. C																								
Proc. D																								
Proc. E																								

CPU																								
Coda																								

processo	t. risposta	t. attesa	*turn-around*
A			
B			
C			
D			
E			
medie			

[4.B]: *Fair Priority Scheduling* (con priorità 3, 5, 2, 4, 1 per i processi A, B, C, D, E, dove 5 è la massima priorità)

Proc. A																								
Proc. B																								
Proc. C																								
Proc. D																								
Proc. E																								

CPU																								
Coda																								

processo	t. risposta	t. attesa	*turn-around*
A			
B			
C			
D			
E			
Medie			

2.

Quesito 6:
[6.A] La seguente soluzione del problema dei lettori-scrittori contiene alcuni errori e mancanze. Lo studente ne modifichi il codice tramite aggiunte, cancellazioni e correzioni. Il risultato dovrà rappresentare una versione corretta, realizzata apportando il minor numero possibile di modifiche all'originale qui di seguito.
(Per coloro che avessero studiato solo sul libro di testo: *P*, corrisponde a *down*, *V* corrisponde a *up*)

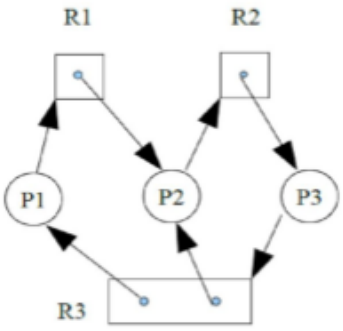
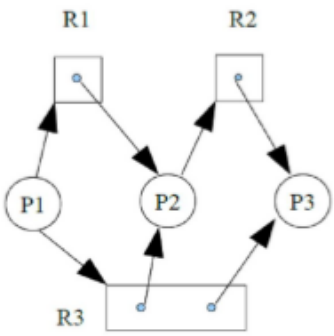
```
void Lettore (void) {

  while (true) {

    numeroLettori++;

    if (numeroLettori==1) P(mutex);

    // leggi il dato

    numeroLettori--;

    if (numeroLettori==0) V(mutex);

    // usa il dato letto

  }

}
```

```
void Scrittore (void) {

  while (true) {

    // prepara il dato da scrivere

    P(dati);

    // scrivi il dato

    V(dati);

  }

}
```

[6.B] Lo studente riporti qua sotto l'indicazione del tipo e del valore iniziale di ciascuna variabile.

3.

Quesito 2 – (6 punti):
Per ciascuno dei seguenti grafi di allocazione delle risorse, si discuta e commenti se i processi rappresentati si trovino o meno in una situazione di stallo.

Grafo A	Grafo B
	

4.

Quesito 6 – (5 punti):
Supponiamo di avere 3 processi che condividono una variabile x e che i loro pseudo-codici siano i seguenti:

P1:	P2:	P3:
P(SemA) P(Mutex) x=x-2 V(Mutex) V(SemC) P(SemA) P(Mutex) x=x-1 V(Mutex) V(SemC)	P(SemB) P(Mutex) x=x+2 V(Mutex) V(SemC) P(SemB)	P(SemC) P(Mutex) if (x<0) then V(SemB) V(Mutex) P(SemC) print(x)

Considerando lo pseudo-codice sopra riportato, determinare se `print(x)` sarà mai eseguita dal processo **P3** e in caso positivo dichiararne l'output. Si assuma che il valore iniziale di x sia 1 e che i semafori abbiano i seguenti valori iniziali: SemA = 1, SemB = 0, SemC = 0, Mutex = 1.

Si assuma che il valore iniziale di x sia 0 e che i tre semafori abbiano i seguenti valori iniziali: SemA = 1, SemB = 0, SemC = 0, Mutex = 1.

Si discuta l'ordine di esecuzione dei vari processi (e delle loro istruzioni) e si determini l'output print(x) del processo P3

5.

Supponiamo di avere 2 processi che condividono una variabile x e che i loro pseudo-codici siano i seguenti (i numeri a sinistra delle istruzioni non fanno parte del codice, servono solo a identificare le istruzioni nei commenti di chi risolve l'esercizio):

P1:

```
 1:  P(SemA)
 2:  P(Mutex)
 3:  x=x-2
 4:  V(Mutex)
 5:  V(SemB)
 6:  P(SemA)
 7:  P(Mutex)
 8:  x=x-1
 9:  V(Mutex)
10:  V(SemB)
11:  print(x)
```

P2:

```
12:  P(SemB)
13:  P(Mutex)
14:  if (x<0) then print(x)
15:  V(Mutex)
16:  V(SemA)
17:  P(SemB)
18:  print(x)
```

I due processi P1 e P2 tentano di eseguire in modo concorrente tra loro. Si assuma che il valore iniziale di x sia 1 e che i semafori abbiano i seguenti valori iniziali: `SemA` = 0, `SemB` = 1, `Mutex` = 1.

[A] Determinare se e quali istruzioni `print(x)` saranno mai eseguite (indicare il numero alla sinistra dell'istruzione corrispondente) e in caso positivo dichiararne l'output.

[B] Si elenchi inoltre un possibile ordine di esecuzione delle istruzioni appartenenti ai due processi (Es. *"1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18"*.

6.

Quesito 4:

```
Process A {

y= y + 2x ;

}
```

```
Process B {

x= x + 1;

Print(x);

}
```

```
Process C {

x= y * 3;

Print(y);

}
```

Sincronizzazione di 3 Processi con Semafori.

Per risolvere il seguente esercizio si faccia uso del meccanismo dei semafori.

Si considerino tre processi (A, B e C) i quali devono eseguire alcune operazioni sulle variabili x e y e poi stamparne il risultato finale.

A questo proposito, si consideri la seguente sequenza di avvenimenti:

- I processi A e C devono essere in attesa (ad un semaforo) di essere risvegliati
- Il processo B sveglia il processo A, che a sua volta sveglia il processo C.
- I tre processi accedono concorrentemente (ma in mutua esclusione) alle variabili condivise *x* e *y*, al fine di eseguire le operazioni riportate in figura.
- Nel caso in cui B e C eseguano le loro operazioni su *x* e *y* prima del processo A, i processi B e C si bloccano in attesa che A abbia terminato la computazione su x e y.
- Infine, B stampa il valore di *x* e C stampa il valore di *y* (senza un particolare ordine prestabilito).

Si utilizzino i semafori (es. *SemA*, *SemB*, *SemC* e *Mutex*) **dichiarandone i valori iniziali**.

Assumendo che inizialmente si abbia x = 2 e y = 1, si discutano brevemente (nel retro di questo foglio o del precedente) i possibili valori di x e y al termine dell'esecuzione concorrente di un'istanza dei processi A, B e C.

7. Un sistema ha 4 processi e 5 risorse da ripartire. L'attuale allocazione e i bisogni massimi sono i seguenti:

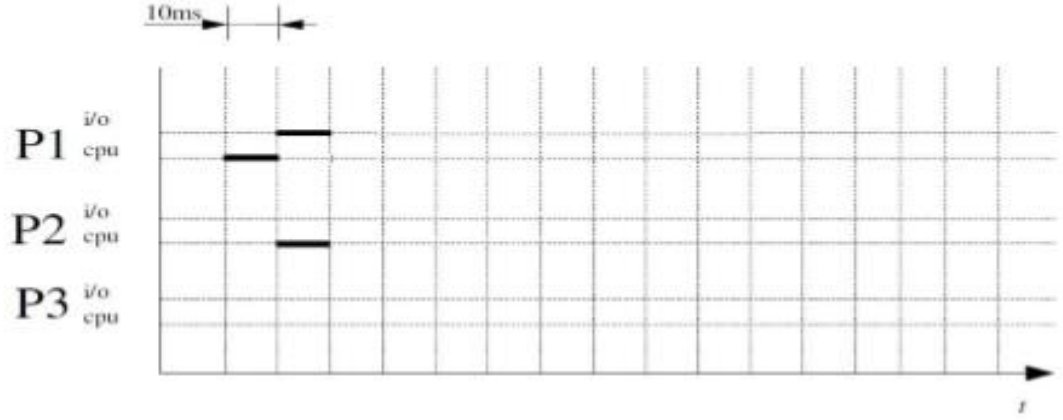
Processo	*Allocate*	*Massimo*
A	1 0 1 1 2	1 1 1 4 2
B	2 0 1 1 1	2 2 2 1 3
C	1 1 1 0 0	2 1 1 0 4
D	1 1 1 0 1	1 1 2 1 3

[A] Considerando il vettore delle risorse disponibili uguale a [0 0 1 2 x], si discuta per quale valore minimo di x questo sia uno stato sicuro e quando invece sia a rischio di deadlock.
[B] Per risolvere l'esercizio lo studente ha di fatto ripetutamente utilizzato una parte di un noto algoritmo. Tale algoritmo assegna risorse a processi solo se l'assegnazione fa rimanere il sistema in uno stato sicuro. Come si chiama questo algoritmo?

8. Si considerino tre processi che alternano fasi di uso di una CPU (job) con fasi di I/O burst. Si utilizzi una politica di CPU scheduling del tipo shortest remaining time next (SRTN) applicata sui vari job pronti per l'esecuzione sulla CPU. I tre processi, denominati P1, P2 e P3, sono caratterizzati dalle seguenti sequenze di CPU burst e I/O burst.

P1: cpu-10ms, i/o-10ms, cpu-30ms, i/o-10ms, cpu-10ms.
P2: cpu-20ms, i/o-10ms, cpu-10ms.
P3: cpu-50ms, i/o-10ms, cpu-10ms.

Ovviamente nessun processo può avanzare alla fase successiva senza aver prima completato le precedenti, nell'ordine. Si supponga che i tre processi facciano I/O su dispositivi distinti. Si mostri, in ciascun istante di tempo, quali processi risultano in stato di running sulla CPU e quali in blocco I/O, marcando, nel diagramma sottostante, le linee di CPU o di I/O corrispondenti a ciascun processo.



13. Il problema del "produttore/consumatore" è un classico problema di sincronizzazione tra più processi che accedono concorrentemente a risorse condivise.
[A] Lo studente utilizzi i monitor per scrivere due procedure chiamate Producer e Consumer che possano essere eseguite concorrentemente al fine di risolvere il problema evitando il deadlock del sistema. (Si consideri il caso in cui le risorse prodotte e non ancora consumate possano essere al massimo N).
[B] Lo studente utilizzi i semafori per risolvere lo stesso problema

9. La seguente soluzione del problema dei lettori-scrittori contiene cinque banali errori o mancanze. Lo studente ne modifichi il codice tramite aggiunte, cancellazioni e correzioni. Il risultato dovrà

rappresentare una versione corretta, realizzata apportando il minor numero possibile di modifiche all'originale qui di seguito. (Per coloro che avessero studiato solo sul libro di testo: P, corrisponde a down, V corrisponde a up) Inoltre, lo studente dichiari i valori iniziali delle variabili numeroLettori, mutex e database.

```
Void Lettore (void) {
  while (true) {
    P(mutex);
    numeroLettori++;
    if (numeroLettori==1) V(database);
    V(mutex);
    // leggi il dato
    numeroLettori--;
    if (numeroLettori==0) V(database);
    // usa il dato letto
  }
}
```

```
void Scrittore (void) {
  while (true) {
    // prepara il dato da scrivere
    V(database);
    // scrivi il dato
    P(database);
  }
}
```

10. I "filosofi a cena" è un classico problema di sincronizzazione tra più processi.
[A] Lo studente descriva brevemente di che problema si tratta e le analogie con scenari inerenti i sistemi operativi.

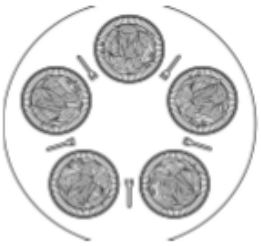
[B] Come visto in aula, lo studente utilizzi i monitor per scrivere una procedura Filosofo che possa essere eseguita concorrentemente ad altre (come fossero un gruppo di filosofi a cena, appunto) evitando deadlock e starvation. A questo proposito si consideri un tavolo con N filosofi ed N forchette.

11. In aula si è discusso di come l'algoritmo del banchiere faccia procedere il sistema attraverso una serie di stati sicuri (safe). Data la situazione schematizzata nelle tabelle seguenti, quante istanze della risorsa R1 devono essere disponibili affinché il sistema sia safe? (Commentare la scelta)

processi	risorse allocate			max ris. richieste		
	R1	R2	R3	R1	R2	R3
P1	2	1	1	4	2	2
P2	3	1	0	4	2	2
P3	1	3	0	1	6	1
P4	0	1	1	2	1	1

risorse disponibili		
R1	R2	R3
?	1	1

12.



I "filosofi a cena" è un classico problema di sincronizzazione tra più processi (i filosofi) che accedono concorrentemente a risorse condivise (le forchette).

Come visto in aula, lo studente utilizzi i ***semafori*** per scrivere una procedura `Filosofo` che cerchi a fasi alterne di pensare e mangiare. Tali procedure dovranno poter essere eseguite concorrentemente (come fossero un gruppo di filosofi a tavola) evitando *deadlock* del sistema o *starvation* di filosofi.

Si consideri un tavolo con `N` filosofi ed `N` forchette.

Nota: lo studente si ricordi di inizializzare i valori delle variabili semaforo usate nella sua soluzione.

Si ripeta l'esercizio sui "filosofi a cena" risolvendolo però con i monitor

13. Si consideri un sistema composto da quattro processi (P1, P2, P3, P4), e quattro tipologie di risorse (R1, R2, R3, R4) con disponibilità: 1 risorsa di tipo R1, 1 risorsa di tipo R2, 1 risorsa di tipo R3, 2 risorse di tipo R4. Si assuma che:

- ogni volta che un processo richieda una risorsa libera, questa venga assegnata al processo richiedente;
- ogni volta che un processo richieda una risorsa già occupata, il processo richiedente deve attendere che la risorsa si liberi prima di potersene impossessare (utilizzando una coda FIFO di processi in attesa di una determinata risorsa)

Si consideri la seguente successione cronologica di richieste e rilasci di risorse:

1) P2 richiede R1,R2,R3
2) P3 richiede R2,R4
3) P2 rilascia R2
4) P4 richiede R4
5) P1 richiede R1
6) P2 richiede R2
7) P3 richiede R3

Verificare se alla fine di questa serie di operazioni il sistema si trovi in condizioni di stallo (suggerimento: usare un grafo di allocazione delle risorse)

14. Il programma in linguaggio C riportato sotto, utilizzabile in ambiente GNU/Linux, mostra come un processo utente possa creare sia un processo figlio (tramite la chiamata fork() alla linea 8) che un thread (tramite la chiamata pthread create(...) alle linee 13 e 17). I flussi di controllo risultanti dall'esecuzione del programma condividono la variabile value inizializzata a 0 alla linea 2. Lo studente indichi quale valore tale variabile avrà quando sarà stampato alle linee 7, 12, 16, 20, 24 illustrando i meccanismi di livello di sistema operativo che intervengono per produrre tale effetto.

```
 0: #include <pthread.h>
 1: #include <stdio.h>
 2: int value = 0;
 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) del thread creato da pthread_create(...)
 7: printf("PARENT before: value = %d\n", value);
 8: pid = fork();
 9: if (pid == 0) { // ramo del processo figlio
10:   /* il processo figlio crea un thread nel suo proprio ambiente
11:   e gli fa eseguire la procedura "troublemaker" */
12:   printf("CHILD before: value = %d\n", value);
13:   pthread_create(&tid,NULL,troublemaker,NULL);
14:   // il processo figlio attende il completamento del thread
15:   pthread_join(tid,NULL);
16:   printf("CHILD after 1: value = %d\n", value);
17:   pthread_create(&tid,NULL,troublemaker,NULL);
18:   // il processo figlio attende il completamento del thread
19:   pthread_join(tid,NULL);
20:   printf("CHILD after 2: value = %d\n", value);}
21: else if (pid > 0) { // ramo del processo padre
22:   // il processo padre aspetta la fine del processo figlio
23:   waitpid(pid,0,0);
24:   printf("PARENT after: value = %d\n", value);}
25: }
26: void *troublemaker(void *param) {
27: value = value + 12;}
```

15. Il problema del "produttore/consumatore" è un classico problema di sincronizzazione tra più processi che accedono concorrentemente a risorse condivise. Lo studente descriva concisamente tale problema.

Inoltre, lo studente utilizzi i monitor per scrivere due procedure chiamate Producer e Consumer che possano essere eseguite concorrentemente al fine di risolvere il problema evitando il deadlock del sistema. (Si consideri il caso in cui le risorse prodotte e non ancora consumate possano essere al massimo N).

16.

Processo	*Allocate*	*Massimo*
A	1 0 2 1 1	1 1 2 1 4
B	2 0 1 1 1	2 2 3 2 1
C	1 1 0 1 0	2 1 4 1 0
D	1 1 1 1 0	1 1 3 2 1

[A] Considerando il vettore delle risorse disponibili uguale a [0 0 x 1 2], si discuta per quale valore minimo di x questo sia uno stato sicuro e quando invece sia a rischio di deadlock.

[B] Per risolvere l'esercizio lo studente ha di fatto ripetutamente utilizzato una parte di un noto algoritmo. Tale algoritmo assegna risorse a processi solo se l'assegnazione fa rimanere il sistema in uno stato sicuro. Come si chiama questo algoritmo?

17. La seguente soluzione del problema dei lettori-scrittori contiene alcuni errori e mancanze. Lo studente ne modifichi il codice tramite aggiunte, cancellazioni e correzioni. Il risultato dovrà rappresentare una versione corretta, realizzata apportando il minor numero possibile di modifiche all'originale qui di seguito. (Per coloro che avessero studiato solo sul libro di testo: P, corrisponde a

down, V corrisponde a up)

```
void Lettore (void) {
  while (true) {
    P(mutex);
    numeroLettori++;
    if (numeroLettori==1) V(database);
    V(mutex);
    // leggi il dato
    numeroLettori--;
    if (numeroLettori==0) V(database);
    // usa il dato letto
  }
}
```

```
void Scrittore (void) {
  while (true) {
    // prepara il dato da scrivere
    V(database);
    // scrivi il dato
    P(database);
  }
}
```

" *LO_FACCIO PER VOI* " cit. sua altissima papa CiccioGamer 89 capo branco dei suoi paguri **_()_**