- 1. Che cos'è un sistema multiprogrammato? Si può realizzare la multiprogrammazione su un sistema con una sola CPU?
  2. Qual è la differenza tra un'interruzione e una trap? Si faccia qualche esempio.

  Risposta:

  1. Un sistema multiprogrammato suddivide la memoria in partizioni, dedicando ognuna di queste ultime a contenere l'immagine di un job. In questo modo se il job correntemente in esecuzione si mette in attesa del completamento di un'operazione di I/O, il sistema operativo può far eseguire un altro job tra quelli presenti in memoria. In questo modo non si spreca tempo di CPU per l'attesa del completamento di operazioni di I/O. La multiprogrammazione può essere realizzata anche su un sistema con una singola CPU.
  2. Un'interruzione è un segnale inviato da un controller alla CPU per informare il sistema del verificarsi di un evento (es.: terminazione di un'operazione di I/O). Un numero, che contraddistingue il controller che ha sollevato l'interrupt, accompagna quest'ultimo e, usato come indice nell'interrupt vector, consente al sistema operativo di caricare ed eseguire la routine corretta per la gestione dell'interruzione. Una trap (o interruzione software) è un'istruzione speciale che serve a far passare il sistema da user mode a kernel mode e viene solitamente usata per implementare il meccanismo delle system call. L'argomento della trap è un numero che identifica la system call; tale numero viene salvato in un posto noto al sistema operativo che così può far partire la routine corretta per l'adempimento della system call.

- 1. Quali sono i vantaggi di un sistema operativo basato su thread rispetto ad uno basato su processi?
  2. Si descriva l'algoritmo di scheduling del sistema Solaris.

  Risposta:

  1. I vantaggi di un sistema operativo basato su thread rispetto ad uno basato sui processi sono i seguenti:
     - creare e cancellare thread è circa 100-1000 volte più veloce che creare e cancellare processi in quanto c'è meno informazione da duplicare/creare/cancellare;
     - lo scheduling fra thread di uno stesso processo è molto più veloce di quello tra processi;
     - la cooperazione di più thread nello stesso task ha come conseguenza un maggior throughput e performance.
  2. L'algoritmo di scheduling di Solaris prevede quattro classi in ordine decrescente di priorità:
     - Real-time,
     - Sistema,
     - Tempo ripartito,
     - Interattiva.

Ognuna di esse ha scale di priorità e algoritmi di scheduling differenti. Per i processi utente la classe predefinita è quella a tempo ripartito che modifica dinamicamente le priorità, assegnando quanti di tempo variabili grazie ad una coda multipla con retroazione. Dato che si vuole privilegiare i processi interattivi ed assicurare buoni tempi di risposta, maggiore è la priorità minore sarà il quanto associato; viceversa a priorità minori corrisponderanno quanti più lunghi. La classe interattiva differisce per il fatto che privilegia le applicazioni che utilizzano un'interfaccia a finestre, assegnando loro priorità più elevate. I livelli di priorità usati in queste due classi sono in tutto sessanta e la tabella di dispatch tiene conto delle seguenti grandezze:

- priorità: valore che dipende dalla classe (più è alto, maggiore è la priorità),
- quanto di tempo: è inversamente correlato alla priorità (a priorità alte corrispondono quanti piccoli e viceversa),
- quanto di tempo esaurito: nuova priorità di un thread che esaurisce il suo quanto di tempo senza sospendersi (la priorità diminuisce perché il thread viene considerato di elaborazione),
- ripresa dell'attività: nuova priorità di un thread che riprende l'esecuzione dopo un periodo d'attesa; di solito il valore à alto per la politica di garantire risposte sollecite ai processi interattivi (che si sospendono spesso).

La classe di sistema viene utilizzata esclusivamente per eseguire i processi del kernel (la priorità di un processo di sistema è fissata a priori, ovvero, non può cambiare in corso di esecuzione).

I thread della classe real-time infine godono di priorità massima e vengono quindi eseguiti prima di quelli di qualunque altra classe.

- In coda ready arrivano i processi P_1, P_2, P_3, P_4, con CPU burst e istanti di arrivo specificati in tabella:

	arrivo	burst
P_1	0	15ms
P_2	10	15ms
P_3	15	10ms
P_4	30	10ms

1. Se i processi terminano nell'ordine P_1, P_3, P_2, P_4, quale può essere l'algoritmo di scheduling? (Si trascuri il tempo di latenza del kernel.)
   (a) RR q=10ms
   (b) Scheduling non-preemptive
   (c) SJF
   (d) RR q=15ms
   (e) Nessuno dei precedenti
2. (*) Si consideri ora tempo di latenza pari a 3ms e un algoritmo di scheduling SRTF. Si determini per i processi P_1, P_2, P_3, P_4 della tabella sopra:

i) il diagramma di GANTT relativo all'esecuzione dei quattro processi;

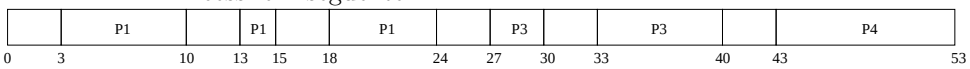
ii) il tempo di attesa medio;

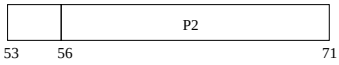
iii) il tempo di turnaround medio.

Risposta:

1. a), b), c).
2. Considerando un tempo di latenza pari a 3ms e un algoritmo di scheduling SRTF, abbiamo quanto segue:

   i) il diagramma di GANTT relativo all'esecuzione dei quattro processi è il seguente:





   ii) il tempo di attesa medio è $\frac{9+(56-10)+(27-15+3)+(43-30)}{4} = 20,75ms$,

   iii) il tempo di turnaround medio è: $\frac{24+(71-10)+(40-15)+(53-30)}{4} = 33,25ms$.

- Si illustri la differenza fra *deadlock* e *starvation*.

  Risposta: nonostante il problema della starvation sia strettamente correlato a quello del deadlock, si tratta di due cose differenti. Infatti, con il termine starvation, si indica un'assenza di progresso per cui un programma in esecuzione non riesce ad ottenere una risorsa od un servizio (a causa ad esempio della politica di allocazione della risorsa/servizio in questione adottata nel sistema), nonostante non venga mai bloccato. Invece con il termine deadlock si intende una situazione in cui un insieme di processi è bloccato in quanto ogni processo è in attesa di un evento che soltanto un altro processo dell'insieme può provocare. Dato che tutti i processi sono in attesa, nessuno di essi potrà mai produrre uno degli eventi che sbloccherebbero un altro elemento dell'insieme.

- Si illustrino le differenze fra semafori e mutex.

  Risposta: un semaforo è sostanzialmente una variabile il cui valore "registra" il numero di *wakeup* eseguiti. La modifica di tale valore è possibile solo attraverso due operazioni atomiche: la *down* che controlla il valore, decrementandolo di un'unità se maggiore di zero oppure sospendendo il processo in caso contrario, e la *up* che incrementa il valore, risvegliando uno dei processi eventualmente in attesa. Un mutex invece è sostanzialmente una versione semplificata del semaforo quando non serve la capacità di "contare" il numero di wakeup. Quindi un mutex è una variabile che può assumere solo due valori: bloccato o sbloccato. In questo caso infatti le operazioni down e up vengono chiamate *mutex_lock* e *mutex_unlock*.

- Si consideri la seguente situazione, dove P_0, P_1, P_2, P_3, P_4 sono cinque 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	D	A	B	C	D
P_0	0	2	0	2	0	3	1	2
P_1	0	0	0	0	2	7	5	0
P_2	2	3	5	4	2	3	5	6
P_3	0	4	3	2	0	4	5	2
P_4	0	0	1	5	0	6	5	5

Available (A)			
A	B	C	D
1	5	3	0

1. Calcolare la matrice R delle richieste.
2. Il sistema è in uno stato sicuro (safe)?
3. Nel caso arrivi la richiesta di allocazione $(1, 4, 2, 0)$ per il processo P_1, quest'ultima può essere soddisfatta, ovvero, il sistema rimane in uno stato sicuro?

Risposta:

1. La matrice R delle richieste è data dalla differenza $Max - C$:

R			
A	B	C	D
0	1	1	0
2	7	5	0
0	0	0	2
0	0	2	0
0	6	4	0

2. Sì il sistema è in uno stato sicuro in quanto esiste la sequenza sicura $\langle P_0, P_2, P_1, P_3, P_4 \rangle$. Infatti, dapprima si esegue P_0 in quanto $R_0 \leq A$ ed A diventa quindi $(1, 7, 3, 2)$. A questo punto $R_2 \leq A$ e quindi si esegue P_2 generando il nuovo valore di A: $(3, 10, 8, 6)$. Quindi si può mandare in esecuzione P_1 dato che $R_1 \leq A$ lasciando inalterato A, dato che $C_1 = (0, 0, 0, 0)$. In seguito può essere eseguito P_3 ($R_3 \leq A$) aggiornando A al valore $(3, 14, 11, 8)$. Infine viene eseguito P_4 ($R_4 \leq A$) generando il valore finale di $A = (3, 14, 12, 13)$.
3. Nel caso venga soddisfatta la richiesta $(1, 4, 2, 0)$ per il processo P_1 i nuovi valori di C, R e A sono i seguenti:

	C				R			
	A	B	C	D	A	B	C	D
P_0	0	2	0	2	0	1	1	0
P_1	1	4	2	0	1	3	3	0
P_2	2	3	5	4	0	0	0	2
P_3	0	4	3	2	0	0	2	0
P_4	0	0	1	5	0	6	4	0

Available (A)			
A	B	C	D
0	1	1	0

Il sistema si trova ancora in uno stato sicuro; infatti esiste la sequenza sicura $\langle P_0, P_2, P_1, P_3, P_4 \rangle$. Anche stavolta si manda in esecuzione dapprima P_0 ($R_0 \leq A$), generando il nuovo valore di A: $(0, 3, 1, 2)$. Dopodiché si può eseguire P_2 ($R_2 \leq A$) aggiornando A al valore $(2, 6, 6, 6)$. Quindi si esegue P_1 ($R_1 \leq A$) portando A al valore

$(3, 10, 8, 6)$. In seguito può essere eseguito P_3 ($R_3 \leq A$) aggiornando A al valore $(3, 14, 11, 8)$; infine l'esecuzione di P_4 ($R_4 \leq A$) genera il valore finale di $A = (3, 14, 12, 13)$.

Il punteggio attribuito ai quesiti è il seguente: 3, 3, 3, 3, 3, 6, 3, 3, 2, 2, 2 (totale: 33).