- 1. In un sistema operativo basato su processi tradizionali, quali sono le informazioni relative a ciascun processo necessarie per gestire il context switch? Dove sono mantenute?
  2. In un sistema operativo basato su thread multipli, in quali situazioni è conveniente usare thread di livello kernel? Si faccia qualche esempio.

  Risposta:

  1. Le informazioni relative a ciascun processo per il context switch sono: il valore del program counter e della program status word, i registri, lo stack pointer, lo stato di esecuzione. Tali informazioni vengono salvate nell'entry (process control block) relativa al processo della process table mantenuta dal sistema operativo.
  2. È conveniente usare thread di livello kernel quando si prevede di compiere parecchie operazioni di I/O; infatti in questo caso l'utilizzo di thread di livello utente porterebbe al blocco dell'intero processo al momento di compiere la system call. Un tipico esempio è costituito da un server web che deve fornire delle risorse (tipicamente pagine HTML ed altri file multimediali), recuperandole dal filesystem, in risposta alle richieste dei client.

- 1. Processi real-time e processi time-sharing possono convivere in uno stesso sistema?
  2. Quali algoritmi di scheduling della CPU sono adatti a sistemi in cui sono presenti sia processi time-sharing che processi real-time?

  Risposta:

  1. Sì, in un sistema possono convivere sia processi real-time che processi time-sharing. Ovviamente, si tratta di sistemi che trattano processi *soft real-time* e bisogna fare in modo che questi ultimi abbiano la precedenza (e quindi la possibilità di prelazione) sui processi time-sharing.
  2. Nei sistemi in cui convivono processi time-sharing e processi real-time gli algoritmi di scheduling della CPU che risultano più efficaci sono quelli con code multiple in cui la coda relativa ai processi real-time è quella con maggiore priorità, mentre i processi time-sharing sono organizzati in code con priorità minori. All'interno delle varie code poi gli algoritmi di scheduling sono solitamente FCFS o RR per i processi real-time e RR per i processi time-sharing. Eventualmente è possibile implementare una tecnica di aging per evitare la starvation dei processi time-sharing.

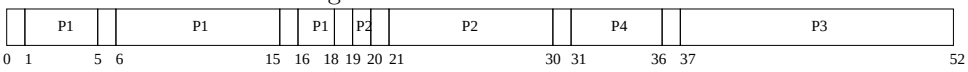
- In coda ready arrivano i processi P_1, P_2, P_3, P_4, con CPU burst e istanti di arrivo specificati in tabella:

	arrivo	burst
P_1	0	15ms
P_2	5	10ms
P_3	15	15ms
P_4	20	5ms

1. Se i processi terminano nell'ordine P_2, P_1, P_4, P_3, quale può essere l'algoritmo di scheduling? (Si trascuri il tempo di latenza del kernel e, nel caso di processi che arrivano in coda ready nello stesso istante, si dia priorità maggiore al processo con indice inferiore.)
   (a) SRTF
   (b) Scheduling non-preemptive
   (c) RR q=5ms
   (d) RR q=10ms
   (e) Nessuno dei precedenti
2. (*) Si consideri ora tempo di latenza pari a 1ms e un algoritmo di scheduling SJF. 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. c), d).
2. Considerando un tempo di latenza pari a 1ms e un algoritmo di scheduling SJF, abbiamo quanto segue:
   i) il diagramma di GANTT relativo all'esecuzione dei quattro processi è il seguente:



   ii) il tempo di attesa medio è $\frac{3+15+22+11}{4} = \frac{51}{4} = 12,75ms$,
   iii) il tempo di turnaround medio è: $\frac{18+25+37+16}{4} = \frac{96}{4} = 24ms$.

- Si illustri un algoritmo di prevenzione dei deadlock che impedisca il verificarsi dell'attesa circolare.

  Risposta: Un modo per evitare l'attesa circolare è quello di imporre un ordinamento totale all'insieme di tutti i tipi di risorse e un ordine crescente per quanto riguarda le richieste effettuate da ciascun processo. In dettaglio, il protocollo adottato è il seguente (nel seguito f indica la funzione di numerazione):

  – ogni processo può richiedere risorse seguendo rigorosamente un ordine crescente di numerazione,
  – inizialmente si può quindi richiedere istanze di una qualunque risorsa R_j,
  – successivamente si possono richiedere istanze di una risorsa R_i se e solo se $f(R_i) > f(R_j)$ (alternativamente si può imporre che un processo prima di richiedere una risorsa R_i debba rilasciare tutte le istanze in suo possesso delle risorse R_j tale che $f(R_j) < f(R_i)$).

In questo modo si può verificare l'impossibilità del verificarsi di un'attesa circolare; infatti, se per assurdo avessimo una catena circolare P_0, P_1, $\ldots$, P_n, dove ogni P_i fosse in attesa di una risorsa R_i posseduta da P_{i+1} (usando l'aritmetica circolare), avremmo l'assurdo che $f(R_0) < f(R_1) < \ldots < f(R_n) < f(R_0)$, ovvero, $f(R_0) < f(R_0)$.

- La programmazione con i semafori può portare a situazioni di deadlock? In caso di risposta positiva si illustri un esempio. In caso di risposta negativa si spieghi il motivo.

  Risposta: Sì, la programmazione con i semafori può portare a deadlock; un esempio è rappresentato dai due frammenti di pseudo-codice seguenti (in cui S1 e S2 sono due semafori inizializzati a 1):

P_0	P_1
down(&S1)	down(&S2)
down(&S2)	down(&S1)
$\vdots$	$\vdots$
up(&S1)	up(&S2)
up(&S2)	up(&S1)

  In questo esempio, se P_0 viene sospeso dopo aver eseguito l'operazione down(&S1) ed aver quindi acquisito l'accesso esclusivo al semaforo S1, P_1 può essere eseguito ed acquisire S2, eseguendo l'operazione down(&S2). A questo punto, qualunque sia l'ordine di scheduling di P_0 e P_1, entrambi si metteranno in attesa di ottenere l'accesso al semaforo bloccato dall'altro processo e quindi si produrrà una situazione di deadlock.

- 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

  1. Calcolare la matrice R delle richieste.
  2. Determinare il minimo valore di x tale che il sistema si trovi in uno stato sicuro.

  Risposta:

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

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

2. 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, $(4, 11, 2)$.

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