- **(a)** Quali sono i vantaggi e gli svantaggi dell'implementazione dei thread a livello utente e a livello kernel, rispettivamente?

  (b) In un sistema in cui i thread sono implementati a livello utente, perché un thread dovrebbe rilasciare la CPU chiamando *thread_yield*?

  Risposta:

  (a) I thread a livello utente sono "invisibili" al sistema operativo che vede un unico processo. Quindi esiste una tabella dei thread separata per ogni processo e questo consente una maggiore flessibilità nel senso che, ad esempio, ogni processo può stabilire una politica di scheduling personalizzata per i propri thread. Lo svantaggio è che il blocco di un thread od il mancato rilascio della CPU da parte di quest'ultimo, porta al blocco dell'intero processo. Quando si parla di thread a livello kernel invece esiste un'unica tabella a livello di sistema per tutti i thread di tutti i processi. Quindi la politica di scheduling viene decisa a livello di sistema ed il blocco di un thread non è fatale per gli altri thread che vengono visti come entità autonome dal sistema operativo.

  (b) In un sistema in cui i thread sono implementati a livello utente, un thread rilascia la CPU chiamando *thread_yield* quando, ad esempio, si mette in attesa di un risultato che deve essere prodotto da un altro thread, in quanto i thread stessi non sono visibili al sistema operativo. Quindi, senza richiamare *thread_yield* non vi potrebbe essere il rilascio della CPU a favore degli altri thread del processo.

- Che cos'è la starvation? Si diano esempi di algoritmi che prevengono la starvation.

  Risposta: 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. Esempi di algoritmi che prevengono la starvation sono FCFS, RR.

- Si consideri un sistema con scheduling SJF con prelazione (cioè SRTF), ove $\alpha = 0,5$ e $\tau_0 = 30$ msec. All'istante 0 il processore si libera e tre processi, P_1, P_2, P_3, sono in coda ready. Finora i processi P_1, P_2 sono andati in esecuzione due volte con CPU burst 30, 20 msec per P_1 e 25, 40 msec per P_2; mentre P_3 è andato in esecuzione una volta con CPU burst di 50 msec.

  Si determini:

  (a) Quale processo viene selezionato dallo scheduler all'istante 0?

  (b) All'istante 10 msec entra nella coda ready un nuovo processo P_4 con CPU burst previsto di 20 msec. Il processo selezionato precedentemente è ancora in esecuzione. Che cosa succede?

  (c) Che cosa succede quando il processo in esecuzione termina il suo burst?

  Risposta:

(a) Viene selezionato P_1 perché il suo prossimo CPU burst previsto (25 msec) è più basso di quelli di P_2 (33,75 msec) e P_3 (40 msec). La formula per calcolare i CPU burst previsti è $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$; quindi, quando $\alpha = 0,5$, si ha $\tau_{n+1} = \frac{t_n+\tau_n}{2}$.

(b) Continua l'esecuzione P_1 in quanto il suo CPU burst previsto è 25 msec ed ha eseguito per 10 msec; quindi il tempo rimanente stimato è 15 msec che è inferiore al CPU burst previsto per P_4 (20 msec).

(c) Viene selezionato di nuovo P_1 se il suo prossimo CPU burst previsto è minore o uguale a 20 msec (burst previsto per P_4, che è quello minimo fra gli altri processi), altrimenti viene selezionato P_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

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, $(4, 11, 2)$.

- Elencare i passi eseguiti dal *driver delle interruzioni*.

  Risposta: I passi eseguiti dal driver delle interruzioni sono i seguenti:

  - salvare i registri della CPU,
  - impostare un contesto per la procedura di servizio (inizializzare TLB, MMU, stack ecc.),
  - inviare un segnale di *acknowledge* al controllore degli interrupt (per avere interrupt annidati),
  - copiare la copia dei registri nel PCB,
  - eseguire la procedura di servizio che accede al dispositivo,
  - eventualmente, cambiare lo stato a un processo in attesa (e chiamare lo scheduler di breve termine),
  - organizzare un contesto (TLB, MMU ecc.) per il processo successivo,
  - caricare i registri del nuovo processo dal suo PCB,
  - continuare l'esecuzione del processo selezionato.

- Supponendo di avere un sistema con tre frame e sette pagine, addottando una politica di rimpiazzamento basata sul working set model, quanti page fault si verificheranno con la reference string seguente, assumendo $\Delta = 3$ e di mantenere in memoria esattamente il solo working set?

  3 4 0 1 0 2 4 5 1 2 6

  (Si assuma che i tre frame siano inizialmente vuoti.)

  Risposta: Simuliamo il funzionamento dell'algoritmo basato sul working set model:

3	4	0	1	0	2	4	5	1	2	6
3	4	0	1	0	2	4	5	1	2	6
	3	4	0	1	0	2	4	5	1	2
		3	4		1	0	2	4	5	1
P	P	P	P		P	P	P	P	P	P

  Si verificano quindi dieci page fault.

- Qual è la differenza fondamentale fra un'architettura UMA ed un'architettura NUMA?

  Risposta: Un sistema multiprocessore UMA (Uniform Memory Access) è un sistema strettamente accoppiato che gode della proprietà che il tempo di accesso a qualunque locazione di memoria è sempre lo stesso (indipendentemente dal fatto che si faccia riferimento a memoria locale o di un altro nodo del sistema). Invece un sistema NUMA (Non Uniform Memory Access), pur rimanendo un sistema strettamente accoppiato, è caratterizzato dal fatto che l'accesso alla memoria locale è più veloce (da 2 a 15 volte) dell'accesso alla memoria remota. Ciò consente di ottenere sistemi scalabili con un gran numero di CPU.

Il punteggio attribuito ai quesiti è il seguente: 2, 3, 3, 3, 3, 3, 2, 2, 3, 4, 3 (totale: 31).