- 1. Che cos'è la multiprogrammazione? Si può realizzare su un sistema monoprocessore?
  2. Quali sono i servizi offerti dai sistemi operativi?

  Risposta:

  1. La nozione di multiprogrammazione prevede la possibilità di mantenere in memoria più processi contemporaneamente in modo da mantenere la CPU occupata il maggior tempo possibile. Infatti, non appena un processo richiede al sistema operativo di eseguire un'operazione di I/O, quest'ultimo può dedicare l'utilizzo della CPU ad un altro processo, invece di attendere passivamente il completamento della suddetta operazione. Questa tecnica è realizzabile anche su un sistema monoprocessore.
  2. I servizi principali messi a disposizione da un sistema operativo sono i seguenti:
     - esecuzione dei programmi: caricamento dei programmi in memoria e relativa esecuzione;
     - operazioni di I/O: il sistema operativo deve fornire un modo per condurre le operazioni di I/O, dato che gli utenti non possono eseguirle direttamente;
     - manipolazione del file system: capacità di creare, cancellare, leggere, scrivere file e directory;
     - comunicazioni: scambio di informazioni tra processi in esecuzione sullo stesso computer o su sistemi diversi collegati da una rete. Implementati attraverso *memoria condivisa* o *passaggio di messaggi*;
     - individuazione di errori: garantire una computazione corretta individuando errori nell'hardware della CPU o della memoria, nei dispositivi di I/O, o nei programmi degli utenti.

- 1. Si descriva il funzionamento di base dei sistemi operativi basati su microkernel.
  2. Si diano esempi di sistemi a microkernel.

  Risposta:

  1. Nei sistemi operativi basati su microkernel il kernel è ridotto all'osso, fornendo soltanto i meccanismi essenziali:
     - un meccanismo di comunicazione tra processi;
     - una minima gestione della memoria e dei processi/thread;
     - una gestione dell'hardware di basso livello (tramite i driver).

     Tutto il resto viene gestito da processi in spazio utente: ad esempio, tutte le politiche di gestione del file system, dello scheduling, della memoria sono implementate come processi. Dal punto di vista delle performance è meno efficiente del kernel monolitico, ma, in compenso, offre una maggiore flessibilità ed immediata scalabilità in ambiente di rete

2. Molti sistemi operativi recenti sono basati, in diversa misura, su microkernel (AIX4, BeOS, GNU HURD, MacOS X, QNX, Tru64, Windows NT ...).

- 1. Si descriva l'algoritmo di scheduling della CPU in Unix tradizionale.
  2. Supponendo che un quanto = 5 tic = 100 msec e che all'istante 0 si trovino in coda ready 3 processi nuovi, nell'ordine A, B, C, tutti con priorità uguale a 0 e CPU burst di 150, 100 e 200 tic rispettivamente, simulare l'esecuzione dei 3 processi nell'intervallo di tempo 0 – 2 secondi.

Risposta:

1. L'algoritmo di scheduling del sistema operativo Unix tradizionale è di tipo RR con code multiple; ogni processo ha una priorità di scheduling e numeri più grandi indicano priorità minore. Per favorire i processi interattivi è previsto un feedback negativo sul tempo di CPU impiegato ed un meccanismo di invecchiamento per prevenire la starvation.

   Un processo esegue per il quanto di tempo assegnato; alla fine di quest'ultimo viene prelazionato. Quando il processo j rilascia la CPU:

   – viene incrementato il suo contatore CPU_j di uso CPU,
   – viene messo in fondo alla stessa coda di priorità,
   – riparte lo scheduler su tutte le code.

   Una volta al secondo, vengono ricalcolate tutte le priorità dei processi in user mode (dove $nice_j$ è un parametro fornito dall'utente):
   $CPU_j = CPU_j/2$ (*fading esponenziale*)
   $P_j = CPU_j + nice_j$
   I processi in kernel mode non cambiano priorità.

2. la simulazione dell'esecuzione è data dalla seguente tabella (si ricordi che l'algoritmo di scheduling nel sistema Unix tradizionale ricalcola i tempi di CPU e le priorità ogni secondo secondo le equazioni CPU_j=$\mathrm{CPU}_j/2$ e Pr_j=CPU_j+nice_j, dove nice_j in questo caso vale 0):

	Pr_A	CPU_A	Pr_B	CPU_B	Pr_C	CPU_C
0 sec	0	0	0	0	0	0
	0	5	0	0	0	0
	0	5	0	5	0	0
	0	5	0	5	0	5
	0	10	0	5	0	5
	0	10	0	10	0	5
	0	10	0	10	0	10
	0	15	0	10	0	10
	0	15	0	15	0	10
	0	15	0	15	0	15
	0	20	0	15	0	15
1 sec	10	10	7	7	7	7
	10	10	7	12	7	7
	10	10	7	12	7	12
	10	10	7	17	7	12
	10	10	7	17	7	17
	10	10	7	22	7	17
	10	10	7	22	7	22
	10	10	7	27	7	22
	10	10	7	27	7	27
	10	10	7	32	7	27
	10	10	7	32	7	32
2 sec	5	5	16	16	16	16

- 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	0	4	3	2	4	5
P_1	2	5	1	3	9	4
P_2	3	6	9	3	7	11

Available (A)

A	B	C
1	4	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.
3. Dopo aver determinato il valore di x con le caratteristiche descritte al punto precedente, dire se la richiesta $(0, 1, 0)$ per il processo P_2 può essere accettata oppure no (spiegandone il motivo).

Risposta:

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

$$\begin{array}{ccc} & \underline{\text{R}} & \\ A & B & C \\ 2 & 0 & 2 \\ 1 & 4 & 3 \\ 0 & 1 & 2 \end{array}$$

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. Lo stesso accade se $x = 1$. Supponiamo quindi che sia $x = 2$, allora l'unica riga di R minore o uguale a A è la terza. Quindi possiamo eseguire il processo corrispondente P_2 che, una volta terminato, restituisce le risorse ad esso allocate aggiornando A al valore $(4, 10, 11)$. A questo punto si può eseguire P_0 dato che $R_0 \leq A$, generando il nuovo valore di A: $(4, 14, 14)$. Infine si può eseguire P_1 dato che $R_1 \leq A$, ottenendo il valore finale di quest'ultimo, ovvero, $(6, 19, 15)$.

   Quindi il valore minimo di x per cui lo stato risulta sicuro è 2; infatti in questo caso esiste la sequenza sicura $\langle P_2, P_0, P_1 \rangle$.

3. Per verificare se la richiesta $(0, 1, 0)$ per il processo P_2 possa essere accettata o meno, simuliamo di soddisfarla e verifichiamo se lo stato risultante è ancora uno stato sicuro. I nuovi valori di C, R ed A sono i seguenti:

$$\begin{array}{ccc} & \underline{\text{C}} & \\ A & B & C \\ 0 & 4 & 3 \\ 2 & 5 & 1 \\ 3 & 7 & 9 \end{array}$$

$$\begin{array}{ccc} & \underline{\text{R}} & \\ A & B & C \\ 2 & 0 & 2 \\ 1 & 4 & 3 \\ 0 & 0 & 2 \end{array}$$

   ed $A = (1, 3, 2)$. L'unica riga di R minore od uguale al vettore A è la terza; quindi eseguiamo P_2, ottenendo come nuovo valore di A il vettore $(4, 10, 11)$. A questo punto, dato che $R_0 \leq A$, mandiamo in esecuzione P_0 e calcoliamo il nuovo valore di A: $(4, 14, 14)$. Infine viene eseguito P_1 (dato che $R_1 \leq A$), generando il valore finale di A: $(6, 19, 15)$.

   Quindi la nuova richiesta per il processo P_2 può essere accettata, dato che il nuovo stato risulta ancora sicuro; infatti anche in questo caso esiste la sequenza sicura $\langle P_2, P_0, P_1 \rangle$.

- Si elenchino (spiegandole) le quattro condizioni di Coffman. Sono condizioni necessarie o sufficienti perché si possa verificare un deadlock?

  Risposta: Le quattro condizioni di Coffman sono:

  1. Mutua esclusione: ogni risorsa è assegnata ad un solo processo, oppure è disponibile.

2. Hold&Wait: i processi che hanno richiesto ed ottenuto delle risorse, ne possono richiedere altre.
3. Mancanza di prerilascio: le risorse che un processo detiene possono essere rilasciate dal processo solo volontariamente.
4. Catena di attesa circolare di processi: esiste un sottoinsieme di processi $\{P_0, P_1, ..., P_n\}$ tali che P_i è in attesa di una risorsa che è assegnata a $P_{i+1 \bmod n}$.

Le condizioni sono necessarie, ma non sufficienti per il verificarsi di uno stallo.

- Spiegare la differenza fra i monitor di Hoare e di Brinch-Hansen.

  Risposta: Nei monitor di Hoare il processo/thread che esegue una signal si sospende, rilasciando il controllo del monitor, per permettere al processo/thread risvegliato di acquisirne il controllo. Invece nei monitor di Brinch-Hansen la signal deve essere l'ultima istruzione nelle procedure del monitor in modo che il processo/thread che la esegue esca automaticamente dal monitor stesso.

- Illustrare una soluzione che permetta di evitare la formazione di catene di attesa circolari di processi. Un sistema che adotti tale soluzione è esente da deadlock? Perché?

  Risposta: Per evitare la formazione di catene di attesa circolari di processi, basta permettere che un processo allochi al più 1 risorsa: tuttavia ciò è molto restrittivo. Una soluzione migliore consiste nello stabilire un ordinamento delle risorse in modo che:

  - vi sia un ordine totale su tutte le classi di risorse,
  - ogni processo richieda le risorse nell'ordine fissato,
  - un processo che detiene la risorsa j non può mai chiedere una risorsa $i < j$, e quindi non si possono creare dei cicli.

  Tale soluzione è teoricamente fattibile, ma difficile da implementare visto che l'ordinamento può non andare bene per tutti ed ogni volta che le risorse cambiano, l'ordinamento deve essere aggiornato.

  Naturalmente un sistema che adotti tale soluzione è esente da deadlock perché nega una delle quattro condizioni di Coffman.

Il punteggio attribuito ai quesiti è il seguente: 2, 2, 2, 2, 3, 4, 2, 2, 2, 4, 3, 4 (totale: 32).