- 1. Quali sono le componenti principali dei sistemi operativi?
  2. Si descriva il meccanismo attraverso cui i programmi in esecuzione richiamano i servizi dei sistemi operativi.

  Risposta:

  1. Le componenti principali di un sistema operativo sono:
     - interfaccia con l'utente (interprete dei comandi e/o interfaccia grafica);
     - gestore dei processi;
     - gestore della memoria (fisica e virtuale);
     - gestore della memoria secondaria;
     - sottosistema di I/O;
     - sottosistema per la protezione del sistema operativo.
  2. I programmi richiamano i servizi del sistema operativo per mezzo delle chiamate di sistema (system call): sono solitamente disponibili come speciali istruzioni assembler o come delle funzioni nei linguaggi che supportano direttamente la programmazione di sistema (ad esempio, il C). Esistono vari tipi di chiamate di sistema relative al controllo di processi, alla gestione dei file e dei dispositivi, all'ottenimento di informazioni di sistema, alle comunicazioni. L'invocazione di una chiamata di sistema serve ad ottenere un servizio dal sistema operativo; ciò viene fatto passando dal cosiddetto *user mode* al *kernel mode* per mezzo dell'istruzione speciale `TRAP`. Infatti, il codice relativo ai servizi del sistema operativo è eseguibile soltanto in kernel mode per ragioni di sicurezza. Una volta terminato il compito relativo alla particolare chiamata di sistema invocata, il controllo ritorna al processo chiamante passando dal kernel mode allo user mode.

- 1. Nell'ambito dei sistemi operativi, che cosa si intende per separazione tra meccanismi e politiche?
  2. Si diano esempi di meccanismi e politiche.

  Risposta:

  1. Con l'espressione separazione tra meccanismi e politiche nell'ambito dei sistemi operativi si intende la distinzione fra come eseguire qualcosa (meccanismo) e che cosa si debba fare, ovvero quali scelte operare, in risposta ad un certo evento (politica).
  2. L'adozione di un timer per proteggere la CPU è un esempio di meccanismo, mentre la decisione riguardante la quantità di tempo da impostare nel timer per un certo utente/processo è un esempio di politica. Nell'ambito della gestione dei processi un meccanismo è l'assegnamento delle priorità ai processi, mentre un esempio di politica è il criterio secondo cui si decide di assegnare priorità maggiori ai processi interattivi a discapito dei processi CPU-bound.

- 1. Si descriva l'algoritmo di scheduling in Unix moderno e in particolare in Solaris.

2. I processi P_1, P_2, P_3 sono in coda ready nell'ordine specificato all'istante 0. P_1 è un processo ciclico che alterna 4ms di esecuzione a 1ms di operazioni di I/O. Il tempo totale di esecuzione per P_1 è pari a 15ms. I processi P_2 e P_3 hanno una richiesta di CPU pari a 17ms e 20ms, rispettivamente e non effettuano I/O. Si calcolino i tempi di attesa per i 3 processi, sapendo che l'algoritmo di scheduling della CPU è RR con q= 5ms e che un processo che rilascia la CPU prima dello scadere del suo quanto, torna all'inizio della coda ready con il resto del quanto.

Risposta:

1. In Unix moderno l'attività di scheduling si basa su un meccanismo generale con 160 livelli di priorità (a numero maggiore corrisponde priorità maggiore) in cui ogni livello viene gestito separatamente con politiche differenti. In particolare è possibile definire delle *classi di scheduling* a cui associare delle politiche di gestione differenti. Ogni classe è caratterizzata da un intervallo di priorità, da un algoritmo per il calcolo di quest'ultima, dall'assegnazione dei quanti di tempo ai vari livelli e dalla migrazione dei processi da un livello all'altro. Infine, per supportare processi real-time i tempi di latenza sono stati ridotti, grazie all'introduzione di punti di prelazionabilità del kernel.

   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:

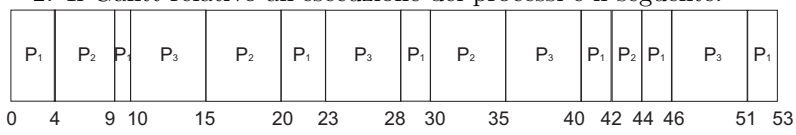
   - priorità: valore che dipende dalla classe (più è alto, maggiore è la priorità),
   - quanto di tempo: è inversamente correlato alla priorità (a priorità alte corrispondono quanti piccoli e viceversa),
   - quanto di tempo esaurito: nuova priorità di un thread che esaurisce il suo quanto di tempo senza 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 è 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.

2. Il Gantt relativo all'esecuzione dei processi è il seguente:



Tempi di attesa:
$\Delta_{P_1} = 5 + 10 + 5 + 10 + 2 + 5 = 37$ ms
$\Delta_{P_2} = 4 + 6 + 10 + 7 = 27$ ms
$\Delta_{P_3} = 10 + 8 + 7 + 6 = 31$ ms

- 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	3	2	6	4	2
P_1	4	0	1	4	1	2
P_2	0	0	2	1	3	6

Available (A)		
A	B	C
3	2	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 $(3, 1, 0)$ per il processo P_0 può essere accettata oppure no (spiegandone il motivo).

Risposta:

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

R		
A	B	C
5	1	0
0	1	1
1	3	4

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. Supponiamo quindi che

sia $x = 1$, allora l'unica riga di R minore o uguale a A è la seconda. Quindi possiamo eseguire il processo corrispondente P_1 che, una volta terminato, restituisce le risorse ad esso allocate aggiornando A al valore $(7, 2, 2)$. A questo punto si può eseguire P_0 dato che $R_0 \leq A$, generando il nuovo valore di A: $(8, 5, 4)$. Infine si può eseguire P_2 dato che $R_2 \leq A$, ottenendo il valore finale di quest'ultimo, ovvero, $(8, 5, 6)$.

Quindi il valore minimo di x per cui lo stato risulta sicuro è 1; infatti in questo caso esiste la sequenza sicura $\langle P_1, P_0, P_2 \rangle$.

3. Per verificare se la richiesta $(3, 1, 0)$ per il processo P_0 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:

	C	
A	B	C
4	4	2
4	0	1
0	0	2

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

ed $A = (0, 1, 1)$. L'unica riga di R minore od uguale al vettore A è la seconda; quindi eseguiamo P_1, ottenendo come nuovo valore di A il vettore $(4, 1, 2)$. A questo punto, dato che $R_0 \leq A$, mandiamo in esecuzione P_0 e calcoliamo il nuovo valore di A: $(8, 5, 4)$. Infine viene eseguito P_2 (dato che $R_2 \leq A$), generando il valore finale di A: $(8, 5, 6)$.

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

- Si illustrino le condizioni per una buona soluzione del problema della sezione critica.

  Risposta: Le condizioni per una buona soluzione del problema della sezione critica sono le seguenti:

  1. **Mutua esclusione:** se il processo P_i sta eseguendo la sua sezione critica, allora nessun altro processo può eseguire la propria sezione critica.
  2. **Progresso:** nessun processo in esecuzione fuori dalla sua sezione critica può bloccare processi che desiderano entrare nella propria sezione critica.
  3. **Attesa limitata:** se un processo P ha richiesto di entrare nella propria sezione critica, allora il numero di volte che si concede agli altri processi di accedere alla propria sezione critica prima del processo P deve essere limitato.

Si suppone che ogni processo venga eseguito ad una velocità non nulla, mentre non si suppone niente sulla velocità relativa dei processi (e quindi sul numero e tipo di CPU).

- Spiegare la differenza fra elusione e prevenzione dei deadlock. L'algoritmo del banchiere è un algoritmo di elusione o di prevenzione dei deadlock? Motivare la risposta.

  Risposta: Eludere i deadlock significa allocare in modo intelligente le risorse in modo da garantire che il sistema si evolva attraversando degli stati sicuri. In altre parole in ogni momento viene garantita una sequenza di terminazione dei processi anche in presenza della massima richiesta possibile di risorse da parte di tutti i processi in esecuzione. Prevenire i deadlock invece significa escludere a priori la possibilità che si verifichi uno stallo negando almeno una delle condizioni di Coffman.

  Da quanto detto segue che l'algoritmo del banchiere è un algoritmo di elusione dei deadlock in quanto soddisfa o meno ogni richiesta in base al fatto che ciò garantisca o meno la transizione in uno stato sicuro.

- Illustrare una soluzione per evitare il verificarsi della condizione *Hold & Wait*. Un sistema che adotti tale soluzione è esente da deadlock? Perché?

  Risposta: Per evitare il verificarsi della condizione *Hold & Wait* è sufficiente garantire che quando un processo richiede un insieme di risorse, non ne richieda nessun'altra prima di rilasciare quelle che ha. Ciò comporta quanto segue:

  - è necessario che i processi richiedano e ricevano tutte le risorse necessarie all'inizio, o che rilascino tutte le risorse prima di chiederne altre;
  - se l'insieme di risorse non può essere allocato in toto, il processo aspetta (metodo transazionale).

  Gli aspetti negativi di tale soluzione sono che vi può essere un basso utilizzo delle risorse e la possibilità di starvation per alcuni processi.

  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).