1. (a) Si descriva il meccanismo attraverso cui i programmi richiamano i servizi del Sistema Operativo. Si faccia qualche esempio.

   (b) Si descriva l'algoritmo di scheduling nel sistema Windows Vista (e Windows 2000).

   Risposta:

   (a) 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. Esempi di chiamate di sistema sono quelle relative alle operazioni fondamentali sui file (apertura, chiusura, lettura, scrittura ecc.).

   (b) In Windows Vista (Windows 2000) i thread rappresentano l'unità di esecuzione gestita dal dispatcher del kernel. Ogni thread ha un proprio stato che include una priorità, un grado di affinità del processore e dell'informazione di accounting. Un thread può trovarsi in uno fra sei stati: ready, standby, in esecuzione, in attesa, in transizione e terminato. Il dispatcher utilizza uno schema a 32 livelli di priorità per determinare l'ordine di esecuzione dei thread. Le priorità sono suddivise in due classi. La classe real-time contiene i thread con intervallo di priorità da 16 a 31. La classe a priorità variabile contiene i thread con intervallo di priorità da 0 a 15. L'algoritmo di scheduling tende a favorire i thread interattivi (decrementando la priorità dei thread che terminano il loro quanto ed aumentando invece la priorità dei thread in attesa di un evento) con un tempo di risposta molto buono; permette inoltre ai thread I/O-bound di mantenere i dispositivi di I/O occupati. I thread CPU-bound utilizzano in background i cicli di CPU disponibili. Lo scheduling può avvenire quando un thread si trova negli stati ready o in attesa, quando termina, oppure quando un'applicazione cambia la priorità oppure il grado di affinità rispetto ad un processore di un thread. I thread real-time hanno un accesso preferenziale alla CPU, ma il sistema non garantisce che un thread real-time verrà eseguito entro dei limiti temporali precisi.

2. Si consideri un sistema con scheduling a priorità con tre code, A, B, C, di priorità decrescente, con prelazione tra code. Un processo prelazionato viene rimesso all'inizio della sua coda e quando riselezionato riceve un nuovo quanto intero. Le code A e B sono round robin con quanto di 15 e 20 ms, rispettivamente; la coda C è FCFS. Se un processo nella coda A o B consuma il suo quanto di tempo, viene spostato in fondo alla coda B o

C, rispettivamente.
Si supponga che i processi P_1, P_2, P_3, P_4 arrivino rispettivamente nelle code A, C, B, A, con CPU burst e tempi indicati nella seguente tabella:

	coda	arrivo	burst
P_1	A	0	20ms
P_2	C	10	25ms
P_3	B	15	20ms
P_4	A	25	20ms

Si determini, approssimando a 1ms il tempo di latenza del kernel (per qualsiasi tipo di operazione):

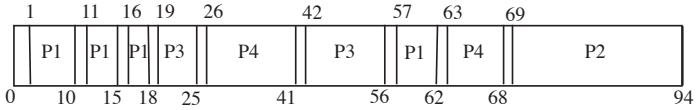
(a) il diagramma di GANTT relativo all'esecuzione dei quattro processi;

(b) il tempo di attesa medio;

(c) il tempo di reazione medio.

Risposta:

(a) Diagramma di GANTT relativo all'esecuzione dei quattro processi:



(b) Tempi di attesa: $\Delta^1_{att} = 3 + (57 - 18) = 42$, $\Delta^2_{att} = 59$, $\Delta^3_{att} = 4 + (42 - 25) = 21$, $\Delta^4_{att} = 1 + (63 - 41) = 23$. Tempo di attesa medio: $\frac{42+59+21+23}{4} = 36,25$.

(c) Tempi di reazione: $\Delta^1_{reaz} = 1$, $\Delta^2_{reaz} = 59$, $\Delta^3_{reaz} = 4$, $\Delta^4_{reaz} = 1$. Tempo di reazione medio: $\frac{1+59+4+1}{4}\frac{70}{4} = 16,25$.

3. (a) Disabilitare tutti gli interrupt prima dell'accesso ad una sezione critica per poi riabilitarli all'uscita è una possibile soluzione per garantire la mutua esclusione. Si elenchino aspetti positivi e negativi.

(b) Si supponga che vi sia un unico cortile condiviso tra due diverse scuole, A e B, e che sia necessario disciplinare l'accesso al cortile tenendo presente quanto segue:

- non deve mai capitare che nel cortile siano presenti contemporaneamente studenti delle scuole A e B;
- uno studente della scuola A (risp. B) prima di accedere al cortile deve controllare che non ci siano già studenti della scuola B (risp. A): se la condizione non è verificata deve attendere.

Si completi la specifica del seguente monitor in modo da soddisfare i requisiti menzionati.

```
monitor SchoolYard
        ...
        integer nA, nB;

        procedure A_enter();
                begin
                        while (nB>0) do ...
                        ...
                end;

        procedure A_exit();
                begin
                        nA:= nA-1;
                        if (...) then ...
                end;

        procedure B_enter();
                begin
                        ...
                end;

        procedure B_exit();
                begin
                        ...
                end;

        nA:=0;
        nB:=0;
end monitor;
```

Risposta:

(a) Se un processo disabilita tutti gli interrupt all'ingresso della sezione critica, per poi riabilitarli all'uscita garantisce la mutua esclusione, ma c'è il rischio che non li riabiliti più, acquisendo il controllo della macchina. Inoltre questa soluzione può allungare di molto i tempi di latenza e non si estende a macchine multiprocessore (a meno di non bloccare tutte le altre CPU). In sostanza si tratta di un meccanismo di mutua esclusione inadatto per i processi utente. Si può tuttavia utilizzare per brevi(ssimi) segmenti di codice affidabile (es: in kernel space, quando si accede a strutture condivise).

(b)
```
monitor SchoolYard
        condition SchoolYardFree;
        integer nA, nB;

        procedure A_enter();
                begin
                        while (nB>0) do wait(SchoolYardFree);
                        nA:=nA+1;
```

```
                end;

        procedure A_exit();
                begin
                        nA:= nA-1;
                        if (nA=0) then signal(SchoolYardFree);
                end;

        procedure B_enter();
                begin
                        while (nA>0) do wait(SchoolYardFree);
                        nB:=nB+1;
                end;

        procedure B_exit();
                begin
                        nB:= nB-1;
                        if (nB=0) then signal(SchoolYardFree);
                end;

        nA:=0;
        nB:=0;
end monitor;
```

4. Si illustrino le problematiche relative alle seguenti possibili implementazioni della primitiva send nel paradigma delle comunicazioni con passaggio di messaggi:

   (a) send bloccante;

   (b) send non bloccante con copia su un buffer di sistema;

   (c) send con meccanismo di copia su scrittura.

   Risposta:

   (a) Nel caso di send bloccante si rischia che la CPU rimanga inattiva durante la trasmissione del messaggio (a meno che ovviamente non ci siano altri processi da eseguire).

   (b) Nel caso di send non bloccante con copia su un buffer di sistema, si "spreca" tempo di CPU per effettuare la copia dal buffer in spazio utente a quello in spazio kernel.

   (c) Infine nel caso di send con meccanismo di copia su scrittura c'è il vantaggio che il buffer viene copiato quando viene modicato solamente in caso di modifica della pagina che lo contiene (infatti la pagina viene marcata come se fosse di sola lettura: in questo modo un tentativo di modifica provoca una trap al sistema operativo che può così gestire la copia). Tuttavia c'è il rischio che venga effettuata una copia "inutile" del buffer quando ad essere modificata è (per esempio) una variabile memorizzata nella stessa pagina del buffer (in questo caso infatti non ci sarebbe il bisogno di effettuare la copia di quest'ultimo, ma il sistema operativo non è in grado di discernere fra i due casi).

5. Si spieghi cosa sono e come funzionano le RPC.

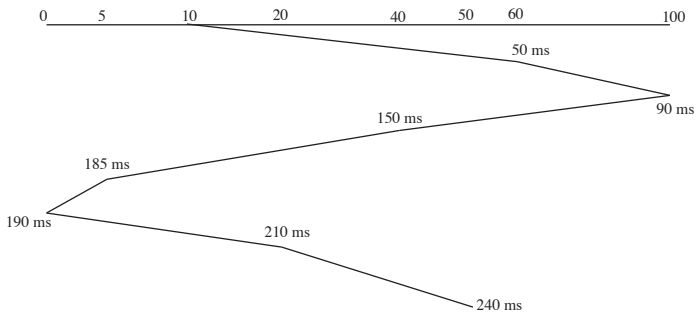
   Risposta: Una Remote Procedure Call (RPC) consente ad un processo in esecuzione su un host di effettuare una chiamata ad una procedura che viene eseguita su un host remoto come se questa fosse una normale chiamata ad una funzione locale. Lo scambio di messaggi necessario per il funzionamento di una RPC è completamente invisibile al programmatore. Sostanzialmente quando un processo su un host A chiama una procedura su un host B, il processo chiamante su A viene sospeso, l'informazione necessaria per la computazione (i parametri della procedura) vengono comunicati via rete e l'esecuzione prosegue su B. Per effettuare una chiamata RPC il processo chiamante (client) utilizza una piccola procedura (client stub) che rappresenta la procedura remota nello spazio di indirizzamento del client. Il client stub organizza i parametri in un messaggio (marshaling) che viene spedito all'host remoto. Giunto a destinazione il messaggio, esso viene elaborato dal server stub che chiama la procedura responsabile della computazione. Il risultato viene poi rispedito via rete dal server stub al client stub che lo restituisce al processo chiamante. Quest'ultimo può quindi riprendere la sua esecuzione come in seguito ad una normale chiamata di procedura locale.

6. Si consideri un disco gestito con politica SCAN. Inizialmente la testina è posizionata sul cilindro 10, ascendente; lo spostamento ad una traccia adiacente richiede 1 ms. Le tracce del disco variano da 0 (prima traccia) a 100 (ultima traccia). Al driver di tale disco arrivano richieste per i cilindri 60, 5, 40, 50, 20, rispettivamente agli istanti 0 ms, 30 ms, 100 ms, 145 ms, 180 ms. Si trascuri il tempo di latenza.

   (a) In quale ordine vengono servite le richieste?

   (b) Il tempo di attesa di una richiesta è il tempo che intercorre dal momento in cui è sottoposta al driver a quando viene effettivamente servita. Qual è il tempo di attesa medio per le quattro richieste in oggetto?

   (a) Le richieste vengono soddisfatte nell'ordine: 60, 40, 5, 20, 50, come risulta dal seguente diagramma:



   (b) Il tempo di attesa medio per le cinque richieste in oggetto è $\frac{(50-0)+(150-100)+(185-30)+(210-180)+(240-145)}{5} = \frac{50+50+155+30+95}{5} = \frac{380}{5} = 76\ ms.$

Il punteggio attribuito ai quesiti è il seguente: 3, 3, 7, 2, 5, 2, 2, 2, 2, 2, 2 (totale: 32).