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

   (b) Si descriva il funzionanmento di un server web basato su thread multipli. Conviene usare thread di livello utente o di livello kernel?

   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) Un server web basato su thread multipli mantiene un thread principale che resta in ascolto delle richieste di connessione da parte dei client. Quando arriva una di queste ultime, viene generato un nuovo thread con il compito di servire la richiesta in oggetto, lasciando libero il thread principale di rimanere in ascolto di ulteriori richieste. In tal modo si mantiene il massimo grado di parallelismo nel servire le varie richieste. Dato che un processo di questo tipo esegue molto I/O conviene utilizzare dei thread a livello kernel, in quanto il blocco di un singolo thread a livello utente, per l'attesa del compimento di un'operazione di I/O, comporta generalmente il blocco dell'intero processo (ovvero, di tutti i suoi thread).

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

   (b) 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 se 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):

i. il diagramma di GANTT relativo all'esecuzione dei quattro processi;

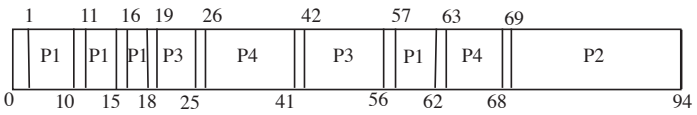
ii. il tempo di attesa medio;

iii. il tempo di reazione medio.

Risposta:

(a) 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.

(b) i. Diagramma di GANTT relativo all'esecuzione dei quattro processi:



ii. 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: $\frac{42+59+21+23}{4} = 36,25$.

iii. 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 Bathroom
        ...
        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 consideri la situazione in cui siano in esecuzione tre processi P_A, P_B e P_C con la matrice delle risorse allocate e la matrice del numero massimo di risorse di cui possono disporre come segue:

$$C = \begin{bmatrix} 3 & 0 & 1 \\ 0 & 2 & 4 \\ 1 & 5 & 2 \end{bmatrix}$$

$$Max = \begin{bmatrix} 3 & 2 & 2 \\ 1 & 2 & 5 \\ 2 & 6 & 3 \end{bmatrix}$$

Se il vettore delle risorse disponibili è $A = (2, 1, 1)$, lo stato attuale è sicuro (safe). Si supponga che arrivi la richiesta $(0, 1, 1)$ relativa al processo P_A: può essere soddisfatta subito? Ovvero, il nuovo stato generato soddisfando la richiesta è sicuro? Perché?

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

$$R = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Se la richiesta $(0, 1, 1)$ relativa al processo P_A venisse accettata, il nuovo valore di A sarebbe $(2, 0, 0)$ e quello di R sarebbe il seguente:

$$R = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Quindi, non essendoci nessuna riga di R minore od uguale a A, il sistema non sarebbe più in uno stato sicuro. Pertando la richiesta non può essere accettata.

Il punteggio attribuito ai quesiti è il seguente: 3, 3, 3, 7, 3, 6, 5 (totale: 30).