Nome e Cognome:
Matricola:

Compitino di Sistemi Operativi

2 Maggio 2006

1. Definire *brevemente* processi e threads. Si descrivano le loro differenze e in quali situazioni l'impiego dei threads può portare dei vantaggi.

   [Punti 3]

2. Descrivere con un opportuno grafo diretto gli stati e le transizioni che possono coinvolgere il ciclo di vita di un processo. In particolare, dettagliare le situazioni in cui può avvenire ciascuna transizione.

   [Punti 3]

3. In un sistema batch, all'istante $t = 0$ ci sono quattro jobs in coda:

Job	CPU
A	5
B	3
C	13
D	X

   dove la colonna CPU indica la lunghezza del picco di CPU in millisecondi.

   Discutere, per i possibili valori di $X \in \mathbb{N}$, con quale strategia e in quale ordine i jobs devono passare in esecuzione in modo da minimizzare il tempo medio di turnaround.

   [Punti 5]

   [**Opzionale**] Confrontare i tempi medi di turnaround ottenuti dalla vostra soluzione con quelli prodotti dalla strategia FCFS con ordine (A,B,C,D), al variare di X.

   [Punti 3]

4. Si consideri un sistema dotato di 5 risorse (R,S,T,U,V) seriali, non rilasciabili, di tipo diverso e contese da cinque processi (A,B,C,D,E). Il sistema alloca le risorse disponibili al primo processo che le richiede. La sequenza di richieste è la seguente:

0. A richiede R	5. E richiede U
1. A richiede S	6. D richiede V
2. C richiede T	7. C richiede U
3. A richiede T	8. E richiede R
4. B richiede R	9. D richiede T

   Rappresentare lo stato del sistema dopo l'ultima richiesta con il grafo di Holt.

   Il sistema è in deadlock? In caso affermativo, quali sono i processi la cui terminazione forzata permetterebbe di far terminare correttamente il maggior numero di processi? In caso negativo, quale ulteriore richiesta porterebbe il sistema in deadlock? Motivare la risposta.

   [Punti 2]

5. Considerare il seguente codice parallelo:

P0:

```
while(TRUE) {
    while(turn != 0);
    sez_critica();
    turn = 1;
    sez_non_critica();
 }
```

P1:

```
while(TRUE) {
    while(turn != 1);
    sez_critica();
    turn = 0;
    sez_non_critica();
}
```

dove `turn` è una variabile condivisa, inizialmente uguale a 0.

Quale delle sequenti proprietà non è soddisfatta? Motivare la risposta.

A. Mutua esclusione
B. Progresso
C. Attesa limitata
D. Nessuna delle precedenti

[Punti 3]

6. Considerare il seguente programma:

```
int readcount=0;        // condivisa
semaphore mutex=1, wrt=1; // condivise
-----------------------------------------------------
 // Scrittore          |    // Lettore
                       |    Down(mutex);
                       |    readcount++;
 Down(wrt);            |    if (readcount == 1) Down(wrt);
 /*  Scrittura */      |    Up(mutex);
 Up(wrt);              |    /* Lettura */
                       |    Down(mutex);
                       |    readcount--;
                       |    if (readcount == 0) Up(wrt);
                       |    Up(mutex);
```

Quale dei seguenti casi si può verificare? Motivare la risposta.

A. Non c'è progresso
B. Non si verifica l'attesa limitata
C. Starvation
D. Non c'è mutua esclusione
E. Lettori e scrittori operano contemporaneamente

[Punti 3]

7. Si assuma che il sistema operativo fornisca le chiamate `up()` e `down()` per implementare un semaforo binario (`sem_binario`). I possibili valori di `sem_binario` sono solamente 0 o 1.

Si implementino (in pseudo codice) le chiamate `sem_up()` e `sem_down()` per fornire un semaforo con contatore secondo la definizione di Dijkstra. Si utilizzi la variabile condivisa `counter` per memorizzare il contatore. Nelle chiamate da implementare si devono utilizzare `up()` e `down()`. La soluzione *non* deve contenere busy waiting. Hint: sfruttare le proprietà del semaforo binario offerte da `up()` e `down()` per risolvere il problema dell'attesa.

[Punti 8]