

Corso di Laurea in Informatica
Corso di Sistemi Operativi I
Compitino del 27/11/2002 (Fila A)

Cognome
Nome
Matricola
e-mail

Durante lo svolgimento dello scritto non si possono usare libri ne' appunti. Oltre ai fogli con le soluzioni degli esercizi da voi elaborate, dovete consegnare questo foglio, compilato con i vostri dati personali (l'indirizzo di e-mail serve per comunicare l'esito del compitino, quindi compilatelo solo se desiderate riceverlo via e-mail).

La tabella della prima pagina riporta i criteri che saranno utilizzati per valutare i diversi esercizi. Per superare il compitino, è necessario aver ottenuto un punteggio complessivo di almeno 18 e aver risposto, almeno in parte, a non meno di 3 domande. Chi totalizza un punteggio non inferiore a 16 può raggiungere la sufficienza con un recupero orale oppure scritto da svolgere in concomitanza del secondo compitino (l'argomento del recupero verrà comunicato dalla docente insieme all'esito del compito).

Es. no.	1	2	3	4	5
Punti	4	5	8	6	7

Esercizio A1 (punti 4)

Rispondere alle seguenti domande (non più di 3 righe per ciascuna risposta):

- 1.a) Cosa è il livello di multiprogrammazione ad un dato istante t ?
- 1.b) Quali dei seguenti aspetti sono vantaggi (miglioramento dell'efficienza nell'uso dei sistemi di calcolo) e quali svantaggi (complessità introdotte nel sistema operativo e nuovi potenziali rischi per i processi) legati all'introduzione della multiprogrammazione? (A) Incremento nell'utilizzo delle risorse; (B) Possibilità di sovrapporre operazioni di I/O e calcolo; (C) Necessità di proteggere ciascun processo e lo stesso sistema operativo dagli altri processi; (D) Aumento del throughput del sistema. (E) Necessità di gestire la competizione per l'uso delle risorse.
- 1.c) A cosa servono la modalità utente e *kernel* di esecuzione della CPU?
- 1.d) Come avviene il passaggio da modo utente a modo kernel al momento del richiamo di una system call?
- 1.e) Qual è la principale differenza fra una trap e un interrupt?
- 1.f) Perché sono stati introdotti i thread?
- 1.g) Perché è più costoso eseguire un context switch fra processi diversi rispetto al context switch fra thread di uno stesso processo?

Esercizio A2 (punti 5)

- 2.a) Spiegare brevemente cosa è una *sezione critica* (nel contesto dell'esecuzione di più processi che condividono qualche variabile) e
- 2.b) proporre una possibile soluzione (descritta con pseudo codice) al problema dell'esecuzione in mutua esclusione di sezioni critiche che eviti il busy waiting (cioè l'attesa attiva).
- 2.c) Perché è meglio evitare il busy waiting?

Esercizio A3 (punti 8)

3.a) Due processi, P1 e P2, eseguono rispettivamente i seguenti frammenti di codice:

P1:	P2:
down(sem1)	down(sem2)
...	...
down(sem2)	down(sem1)
....
....
up(sem2)	up(sem2)
up(sem1)	up(sem1)

Quale problema si può verificare durante l'esecuzione dei due processi se *sem1* e *sem2* sono due semafori inizializzati al valore 1?

Suggerimento: provare a simulare l'esecuzione dei due processi tenendo conto che le istruzioni di ciascun processo possono essere eseguite in modo intercalato con quelle dell'altro processo);

3.b) Fornire una soluzione al problema dei lettori e scrittori senza starvation, che utilizzi un semaforo privato per ciascuna delle due classi di processi e segua lo schema di procedure per l'acquisizione e il rilascio di risorse riportato sotto (occorrerà scrivere una *IniziaLettura* e una *IniziaScrittura* corrispondenti all'acquisizione della risorsa – il dato da leggere/scrivere – e una *FineLettura*, *FineScrittura*). Per evitare la starvation: (a) i lettori iniziano la lettura solo se non ci sono scrittori in attesa né in fase di scrittura, (b) quando un lettore termina il suo lavoro e non ci sono altri lettori attivi, risveglia uno scrittore, (c) uno scrittore al termine della scrittura sveglia tutti i lettori in attesa, o, in assenza di lettori, uno scrittore in attesa.

Schema di procedure Acquisisci/Rilascia (Tipo 2)

```
Acquisisci(int i)
{
    down(mutex);
    if (condizione di sincronizzazione NON soddisfatta)
        {<annota processo di classe i bloccato>
         up(mutex);
         down(priv[i]);
         <annota processo in classe i sbloccato>
        }
    <alloca risorsa>
    if ( condizione di sincronizzazione vera per qualche processo )
        { j = scegli_classe_di_processo_da_attivare;
          up(priv[j]);}
    else
        up(mutex);
}

Rilascia()
{
    int j;
    down(mutex);
    <annota risorsa rilasciata>
    if (condizione di sincronizzazione vera per qualche processo )
        { j = scegli_classe_di_processo_da_attivare;
          up(priv[j]);}
    else
        up(mutex);
}
```

Esercizio A4 (punti 6)

4.a) Quali system call utilizza l'interprete di comandi quando l'utente richiede l'esecuzione di un programma? (come quando per esempio un utente di un sistema UNIX sottopone alla shell il comando "helloworld", corrispondente ad un eseguibile che stampa "Hello world!" e poi termina).

Nota: la shell deve attendere la fine dell'esecuzione prima di chiedere all'utente un nuovo comando.

4.b) Quali strutture dati del sistema operativo vengono modificate dalle system call descritte al punto 4.a?

4.c) Indicare i cambiamenti di stato della shell durante la sequenza Inserimento del comando - Esecuzione del comando - Stampa del prompt per richiedere il prossimo comando.

Esercizio A5 (punti 7)

Nella ready queue sono presenti i seguenti processi (arrivati tutti insieme al tempo 0; sono elencati in ordine di inserimento nella ready queue, con P1 in testa alla lista, la priorità più alta è la 5, la più bassa è la 1):

Processo	P1	P2	P3	P4	P5
Cpu burst (sec)	9	6	1	4	7
Priorità	3	5	2	1	4

5.a) Disegnare il diagramma di Gantt e calcolare il tempo medio di attesa in coda (pura attesa in ready queue, al netto del tempo di esecuzione) per ciascuna delle seguenti politiche di scheduling: Round Robin con $q = 2$, SJF, Priorità.

5.b) Per ciascuna politica calcolare anche l'overhead complessivo dovuto al context switch, supponendo che il tempo necessario per effettuare un singolo context switch sia t_{cs} .