Cognome e nome: _____ Matricola: _____ Posto: ____ ____

Università degli Studi di Padova - Facoltà di Scienze MM.FF.NN. - Corso di Laurea in Informatica

Regole dell'esame

Il presente esame scritto deve essere svolto in forma individuale in un tempo massimo di 90 minuti dalla sua presentazione.
Non è consentita la consultazione di libri o appunti in forma cartacea o elettronica, né l'uso di palmari e telefoni cellulari.
La correzione e la sessione orale avverrà in data e ora comunicate dal docente durante la prova scritta; i risultati saranno esposti sul sito del docente entro il giorno precedente gli orali.
Per superare l'esame il candidato deve acquisire almeno 1.5 punti nel Quesito 1 e un totale di almeno 18 punti su tutti i quesiti, inserendo le proprie risposte interamente su questi fogli. Riportare generalità e matricola negli spazi indicati.
Per la convalida e registrazione del voto finale il docente si riserva di proporre al singolo candidato una prova orale.

Quesito 1 (punti 4): ***1 punto per risposta giusta, diminuzione di 0,33 punti per risposta sbagliata, 0 punti per risposta vuota***

[1.A]: Quale tra le seguenti affermazioni è corretta in relazione alla politica di ordinamento processi "FCFS senza valutazione dell'attributo di priorità":

1. il tempo di attesa è sempre maggiore del tempo di risposta
2. il tempo di attesa è sempre minore del tempo di risposta
3. il tempo di attesa è sempre uguale al tempo di risposta
4. il tempo di attesa ed il tempo di risposta non hanno alcun legame prefissato.

[1.B]: Sia dato un sistema di memoria con indirizzi virtuali suddivisi in 4 campi: *a, b, c, d,* i primi 3 dei quali siano utilizzati per indirizzare tre livelli gerarchici di tabelle delle pagine e il quarto campo rappresenti l'*offset* entro la pagina selezionata. Indicare dall'ampiezza di quali campi dipende il numero di pagine indirizzate nel sistema:

1. da quella di tutti e quattro i campi
2. da quella del campo *d*
3. da quella del campo *a* e *d*
4. da quelle dei campi *a, b, c*.

[1.C]: Un semaforo binario può:

1. assumere solo valori discreti
2. gestire solo l'accesso a due risorse condivise
3. gestire solo le richieste di accesso provenienti da due processi
4. assumere solo i valori 0 e 1, con essi denotando "risorsa occupata" e "risorsa libera".

[1.D] In quale tra i seguenti sistemi operativi è più conveniente l'utilizzo di *Inverted Page Tables*:

1. nessuno dei seguenti, il vantaggio è pari per tutti
2. sistemi a 16 bit
3. sistemi a 32 bit
4. sistemi a 64 bit

RISPOSTE AL QUESITO 1: **A ____** **B ____** **C ____** **D ____**

Quesito 2 – (6 punti):
Si consideri un sistema che utilizza la paginazione per gestire la memoria.
Si discutano brevemente vantaggi e svantaggi nell'adottare pagine di dimensione ampia oppure di piccola.

A) Pagine di dimensione ampia (vantaggi e svantaggi):

B) Pagine di dimensione piccola (vantaggi e svantaggi):

Cognome e nome: _____ Matricola: _____ Posto: ____ ____

Come visto in classe, il valore ottimo di dimensione di una pagina può essere definito matematicamente.
Utilizzando i seguenti parametri:

- σ byte dimensione media di un processo
- π byte dimensione media di una pagina
- ε byte per riga in tabella delle pagine

C) si scriva innanzitutto una funzione $f(\pi)$ che definisca matematicamente lo spreco di memoria in maniera dipendente dalla dimensione π di una pagina.

- **$f(\pi)$ =**

D) si determini quindi il valore ottimo di π che minimizza lo spreco di memoria.

Quesito 3 – (6 punti):
Si consideri un sistema composto da quattro processi (P1, P2, P3, P4), e quattro tipologie di risorse
(R1, R2, R3, R4) con disponibilità: 1 risorsa di tipo R1, 1 risorsa di tipo R2, 1 risorsa di tipo R3, 2 risorse di tipo R4.
Si assuma che:
- ogni volta che un processo richieda una risorsa libera, questa venga assegnata al processo richiedente;
- ogni volta che un processo richieda una risorsa già occupata, il processo richiedente deve attendere che la risorsa si liberi prima di potersene impossessare (utilizzando una coda FIFO di processi in attesa di una determinata risorsa)
Si consideri la seguente successione cronologica di richieste e rilasci di risorse:
1) P2 richiede R1,R2,R3
2) P3 richiede R2,R4
3) P2 rilascia R2
4) P4 richiede R4
5) P1 richiede R1
6) P2 richiede R2
7) P3 richiede R3
Verificare se alla fine di questa serie di operazioni il sistema si trovi in condizioni di stallo (suggerimento: usare un grafo di allocazione delle risorse).

Cognome e nome: _____ Matricola: _____ Posto: ____ ____

Quesito 4 – (8 punti):

Il programma in linguaggio C riportato sotto, utilizzabile in ambiente GNU/Linux, mostra come un processo utente possa creare sia un processo figlio (tramite la chiamata fork() alla linea 8) che un thread (tramite la chiamata pthread create(...) alle linee 13 e 17). I flussi di controllo risultanti dall'esecuzione del programma condividono la variabile value inizializzata a 0 alla linea 2. Lo studente indichi quale valore tale variabile avrà quando sarà stampato alle linee 7, 12, 16, 20, 24 illustrando i meccanismi di livello di sistema operativo che intervengono per produrre tale effetto.

```
 0: #include <pthread.h>
 1: #include <stdio.h>
 2: int value = 0;
 3: void *troublemaker(void *param);
 4: int main(int argc, char *argv[]) {
 5: int pid; // id (intero) del processo creato da fork()
 6: pthread_t tid; // id (strutturato) del thread creato da pthread_create(...)
 7: printf("PARENT before: value = %d\n", value);
 8: pid = fork();
 9: if (pid == 0) { // ramo del processo figlio
10:   /* il processo figlio crea un thread nel suo proprio ambiente
11:   e gli fa eseguire la procedura "troublemaker" */
12:   printf("CHILD before: value = %d\n", value);
13:   pthread_create(&tid,NULL,troublemaker,NULL);
14:   // il processo figlio attende il completamento del thread
15:   pthread_join(tid,NULL);
16:   printf("CHILD after 1: value = %d\n", value);
17:   pthread_create(&tid,NULL,troublemaker,NULL);
18:   // il processo figlio attende il completamento del thread
19:   pthread_join(tid,NULL);
20:   printf("CHILD after 2: value = %d\n", value);}
21: else if (pid > 0) { // ramo del processo padre
22:   // il processo padre aspetta la fine del processo figlio
23:   waitpid(pid,0,0);
24:   printf("PARENT after: value = %d\n", value);}
25: }
26: void *troublemaker(void *param) {
27: value = value + 12;}
```

Cognome e nome: _____ Matricola: _____ Posto: ____ ____

Quesito 5 – (8 punti):

Il problema del "produttore/consumatore" è un classico problema di sincronizzazione tra più processi che accedono concorrentemente a risorse condivise.
Lo studente descriva concisamente tale problema.

Inoltre, lo studente utilizzi i monitor per scrivere due procedure chiamate `Producer` e `Consumer` che possano essere eseguite concorrentemente al fine di risolvere il problema evitando il *deadlock* del sistema.
(Si consideri il caso in cui le risorse prodotte e non ancora consumate possano essere al massimo `N`).

Cognome e nome: _____ **Matricola:** _____ **Posto:** ____ ____

Soluzione

Soluzione al Quesito 1

[1.A]: risposta 3
[1.B]: risposta 4
[1.C]: risposta 4
[1.D]: risposta 4

Soluzione al Quesito 2

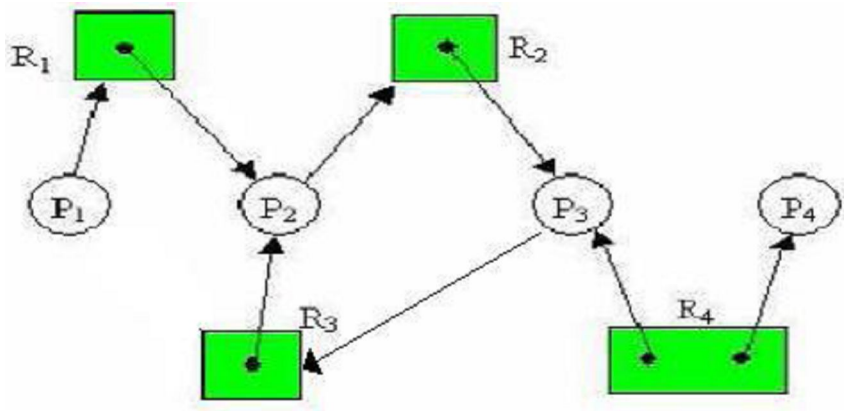
A) Pagine **ampie**
- Maggiore rischio di **frammentazione interna** ma tabella delle pagine più piccola
  - In media ogni processo lascia inutilizzata metà del suo ultimo *page frame*

B) Pagine **piccole**
- Maggiore ampiezza della tabella delle pagine ma minor frammentazione interna

C) Spreco per processo come $f(\pi) = (\sigma / \pi) \times \varepsilon + \pi / 2$
- Parte di tabella delle pagine + frammentazione interna

D) Derivata prima è $- \sigma \varepsilon / \pi 2 + ½$
- Ponendo uguale a zero si ha che il minimo di $f(\pi)$ si ha per $\pi = \sqrt{(2 \sigma \varepsilon)}$

Soluzione al Quesito 3



Alla fine delle operazioni descritte, il grafo di allocazione delle risorse appare come in figura. Come è evidente, esiste un ciclo di richieste/assegnazioni che coinvolge P2, R2, P3,R3: pertanto, il sistema è in stallo.

Soluzione al Quesito 4

L'esecuzione del programma mostrato richiede l'intervento del sistema operativo a vari livelli. Esaminiamoli seguendo il numero d'ordine della linea di programma che li chiama in gioco, assumendo una memoria virtuale paginata.

linee 4 : questo è il punto di inizio del processo P risultante dall'invocazione dell'eseguibile prodotto dal comando di compilazione del programma (`gcc -g thread.c -o thread -lpthread`, dove "`thread.c`" è il nome arbitrario del *file* contenente il sorgente); in questo momento il S/O ha creato un singolo processo, assegnandogli la sua memoria virtuale in modalit`a "*paging-on-demand*"; naturalmente, trattandosi di un programma estremamente ridotto, tutti i segmenti del suo eseguibile sono agevolmente contenuti, separatamente, in una singola pagina; poiché siamo in ambiente GNU/Linux, al processo P è stato implicitamente assegnato un *thread* TP che ne rappresenta il flusso di controllo

linee 5-7 : in questa fase è in esecuzione il solo *thread* TP per conto del processo P; alla linea 8 TP stampa su `stout` (il terminale di invocazione del programma) la stringa `PARENT before: value = 0`

linea 8 : l'invocazione di `fork()` duplica il processo P creando un clone identico C, ma non comprensivo dei *thread* interni, e in modalità *copy-on-write*; come per P viene creato implicitamente un *thread* TC0 il cui PC (*program counter* viene posizionato alla stessa linea 8 del programma, al punto in cui la variabile `pid` viene assegnata; in virtù della condivisione

Cognome e nome: _____ Matricola: _____ Posto: ____ ____

copy-on-write, TC0 legge una copia diversa della variabile `pid`, che vale 0 (un valore fittizio) per TC0 e un valore positivo (vero) per TP

linea 12 : qui è in esecuzione TC0 , che legge la variabile `value` dalla copia di TP e dunque stampa sullo stesso `stout` di TP la stringa `CHILD before: value = 0`

linee 13-15 : TC0 prima crea un nuovo *thread* TC1, con il quale condivide tutte le risorse fisiche e logiche, e lo incarica di eseguire la procedura `troublemaker` e poi si pone in attesa del suo completamento

linea 27 : questa è la sola istruzione eseguita da TC1, che opera sulla stessa variabile `value` di TC0 , cui dunque viene incrementato il valore da 0 a 12; in virtù della modalità di lavoro *copy-on-write* questa modifica non ha effetto sulla copia in possesso del processo P; TC1 termina subito dopo l'assegnamento a `value`

linea 16 : quando TC0 riprende l'esecuzione, TC1 è terminato e dunque `value` vale 12, ragion per cui TC0 stampa su `stout` la stringa `CHILD after 1: value = 12`

linee 17-19 : TC0 crea un nuovo *thread* TC2, con il quale condivide tutte le risorse fisiche e logiche, e lo incarica di eseguire la procedura `troublemaker` e poi si pone in attesa del suo completamento

linea 27 : questa è la sola istruzione eseguita da TC2, che opera sulla stessa variabile `value` di TC0 , cui dunque viene incrementato il valore da 12 a 24; in virtù della modalità di lavoro *copy-on-write* questa modifica non ha effetto sulla copia in possesso del processo P; TC2 termina subito dopo l'assegnamento a `value`

linea 20 : quando TC0 riprende l'esecuzione, TC2 è terminato e dunque `value` vale 24, ragion per cui TC0 stampa su `stout` la stringa `CHILD after 2: value = 24`

linee 21-24 : la terminazione di TC0 comporta anche la terminazione del processo C che non ha altri *thread* in esecuzione; a questo punto TP può riprendere l'esecuzione, non trovando alcuna modifica nella propria copia di `value`, stampando dunque su `stout` la stringa `PARENT after: value = 0`

Soluzione al Quesito 5

Il problema è chiaramente spiegato nel libro di testo e nei lucidi. Varie soluzioni possibili, ad esempio:

```
monitor ProducerConsumer
        condition full, empty;
        integer count;
        procedure insert(item: integer);
        begin
                if count = N then wait(full);
                insert_item(item);
                count := count + 1;
                if count = 1 then signal(empty)
        end;
        function remove: integer;
        begin
                if count = 0 then wait(empty);
                remove = remove_item;
                count := count – 1;
                if count = N – 1 then signal(full)
        end;
        count := 0;
end monitor;
```

```
procedure producer;
begin
        while true do
        begin
                item = produce_item;
                ProducerConsumer.insert(item)
        end
end;
procedure consumer;
begin
        while true do
        begin
                item = ProducerConsumer.remove;
                consume_item(item)
        end
end;
```