SISTEMI OPERATIVI E LABORATORIO
Indirizzo Sistemi e Reti – 21 settembre 2010

Cognome:_____Nome:_____
Matricola:_____

1. Ricordate che non potete usare calcolatrici o materiale didattico, e che se consegnate annullate automaticamente qualsiasi voto conseguito in una prova precedente.
2. Ricordate che se prendete un voto inferiore a 10 dovete saltare il primo scritto della sessione di febbraio/marzo 2011.
3. Si ricorda che in questo appello è necessario sostenere l'intera prova scritta (teoria più laboratorio)

ESERCIZI RELATIVI ALLA PARTE DI TEORIA DEL CORSO

ESERCIZIO 1 (7 punti)

Tre processi P_A, P_B e P_C eseguono il seguente codice:

Shared **Var** semaphore mutex = 1; (valore iniziale)
semaphore done = 0; (valore iniziale)

P_A:	**P_B:**	**P_C:**
repeat forever:	**repeat forever**:	**repeat forever**:
wait(mutex)	**wait**(done)	**wait**(done)
<A>	**wait**(mutex)	**wait**(done)
signal(mutex)	<B>	**wait**(mutex)
signal(done)	**signal**(mutex)	<C>
	signal(done)	**signal**(mutex)

a1)
L'esecuzione concorrente di P_A, P_B e P_C produce una sequenza (di lunghezza indefinita) di chiamate alle procedure A, B e C. Quali delle sequenze qui sotto riportate possono essere la porzione iniziale di sequenze prodotte dall'esecuzione concorrente di P_A, P_B e P_C? (marcate le sequenze che scegliete con una croce nello spazio apposito)
A2)
In ciascuna delle sequenze restanti, che non possono essere prodotte dall'esecuzione concorrente dei tre processi, cerchiate la lettera che rappresenta l'ultima procedura che può effettivamente essere eseguita nell'esecuzione concorrente di P_A, P_B e P_C

1. [X] A,A,B,C,A,A,C,A,A …
2. [ ] A,B,B,C,A,A,C,B,A…
3. [ ] A,B,A,C,A,A,C,B,A …
4. [X] A,B,A,A,C,A,B,B,C …

b)
Riportate un semplice esempio in pseudo-codice (simile a quello usato per la prima parte di questa domanda) di due processi concorrenti che usano uno o più semafori per sincronizzarsi e che, *a seconda dell'ordine relativo in cui vengono eseguite le istruzioni dei due processi, può sia funzionare*

correttamente che portare in una situazione di deadlock. Indicate anche come devono essere inizializzati i semafori che usate.

P1	P2
wait(mutex1)	wait(mutex2)
wait(mutex2)	wait(mutex1)
sez. critica	sez. critica
signal(mutex2)	signal(mutex1)
signal(mutex1)	signal(mutex2)

semaphore mutex1 = 1; semaphore mutex2 = 1;

c)
All'interno di un sistema operativo, un certo processo P è correntemente in stato di "Running", e si sa che non dovrà più rilasciare la CPU volontariamente prima di aver terminato la propria esecuzione (in altre parole, non deve più eseguire operazioni di I/O, di sincronizzazione o di comunicazione con altri processi).
Quale/quali, tra gli algoritmi di scheduling **FCFS**, **SJF preemptive, SJF non-preemptive**, **round robin** garantisce/garantiscono che il processo P riuscirà a portare a termine la propria computazione? (motivate la vostra risposta, assumendo che SJF possa effettivamente essere implementato)

FCFS, SJF non-preemptive e round robin. Infatti, nel caso di SJF preemptive potrebbe sempre arrivare in coda di ready un processo che deve usare la CPU per un tempo minore di quanto rimane da eseguire a P.

d) in che modo potremmo assicurarci che P possa terminare la propria esecuzione indipendentemente da quale degli algoritmi di scheduling sopra indicati viene usato?

Usando un meccanismo di aging, cosicché la priorità di P aumenta quanto maggiore è il tempo che passa in coda di ready.

e) in quale caso P potrebbe uscire volontariamente dallo stato di "Ready to Run"?

mai.

f) Come deve essere modificato il diagramma di stato della vita di un processo per rappresentare il funzionamento di un sistema multi-tasking?

Basta eliminare l'arco di transizione che va dallo stato *Running* allo stato *Ready (to Run)* e marcato *interrupt*

ESERCIZIO 2 (7 punti)

In un sistema la memoria fisica è divisa in 2^{20} frame, un indirizzo logico è scritto su 31 bit, e all'interno di una pagina, l'offset massimo è 1FF.

a) Quanti frame occupa la la page table più grande del sistema? (motivate numericamente la vostra risposta)

Un frame/pagina è grande 2^9 = 512 byte, e quindi la page table più grande può avere $2^{(31-9)} = 2^{22}$ entry. Nel sistema vi sono 2^{20} frame, per cui sono necessari tre byte per scrivere il numero di un frame, e quindi la page table più grande occupa $(2^{22} \cdot 3) / 2^9$ frame = $2^{13} \cdot 3$ frame = 24K frame

b) Il sistema può adottare una paginazione semplice, cioè ad un solo livello (motivate la vostra risposta)?

No perché la page table più grande non può essere memorizzata in un unico frame, e neanche in un numero ragionevolmente piccolo di frame adiacenti.

c) Quale dimensione minima dovrebbero avere le pagine di questo sistema per essere certi di non dover ricorrere ad una paginazione a più livelli? (motivate la vostra risposta, e per questa domanda assumete di usare 8 byte per scrivere il numero di un frame all'interno di una page table)

Poniamo 31 = m + n (m = bit usati per scrivere un numero di pagina, n = bit usati per scrivere l'offset).
Allora il numero di entry della PT più grande, moltiplicato per la dimensione di una entry deve poter essere contenuto in una pagina/frame, ossia: $2^m \cdot 2^3 \leq 2^n$
Da cui: $m + 3 \leq n$. Poiché $m = 31 - n$; risolvendo il semplice sistema si ha $n = 17$, ossia le pagine devono almeno essere grandi 2^{17} = 128 Kbyte.

d) Se in un sistema non si vuole usare una paginazione a più livelli, quale soluzione alternativa si può adottare?

Una inverted page table

e) In che senso la segmentazione della memoria è una tecnica più "naturale" della paginazione?

Nel senso che i segmenti, al contrario delle pagine, contengono porzioni "omogenee" di un programma (una subroutine, una struttura dati, eccetera), ed è qunidi più facile gestire la protezione dei segmenti (ad esempio la protezione in scrittura o l'uso di indici errati per gli array)