SISTEMI OPERATIVI – 6 settembre 2018 corso A

Cognome:_____**Nome:**_____
Matricola:_____

Ricordate che non potete usare calcolatrici o materiale didattico, e che potete consegnare al massimo tre prove scritte per anno accademico.

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 = –1; (valore iniziale)

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

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 la/le sequenza/e che scegliete con una croce nello spazio apposito, e assumete che nella coda di waiting di "done" sia inizialmente addormentato un processo)

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

b) Riportate il diagramma di stato che descrive come, durante la vita di un processo di un modermo sistema time sharing, questo si muova tra diversi stati.

Si vedano i lucidi della sezione 3.1.2

c) In un moderno sistema time-sharing con uno spazio di indirizzamento logico maggiore di quello fisico, per quali ragioni un processo può dover transire dallo stato "running" allo stato "waiting"?

Perché deve compiere una operazione di I/O
Perché si è addormentato su un semaforo
Perché ha indirizzato una pagina non presente in RAM (page fault)

d) Come si può modificare il diagramma di stato per descrivere un sistema multi-tasking (ma non time-sharing)?

Basta rimuovere l'arco che va dallo stato "running" allo stato "ready to run"

e) Che vantaggio da l'uso dei thread al posto dei processi?

Un insieme di peer thread condivide lo spazio di indirizzamento. Per questa ragione, il context switch tra peer thread e la creazione di un nuovo peer thread richiedono molto meno tempo delle corrispondenti operazioni sui processi.

ESERCIZIO 2 (7 punti)

In un sistema vengono usati 16 bit per scrivere il numero del frame di un indirizzo fisico, un indirizzo logico è scritto su 35 bit, e all'interno di una pagina, l'offset massimo è 1FFF.

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

Un frame/pagina è grande 2^{13} = 8192 byte, e quindi la page table più grande può avere $2^{(35-13)} = 2^{22}$ entry. Nel sistema vi sono 2^{16} frame, per cui sono necessari 2 byte per scrivere il numero di un frame, e quindi la page table più grande occupa ($2^{22} \cdot 2$) byte = 8 Mbyte

b) Assumendo che un indirizzo logico sia sempre scritto su 35 bit e che la memoria fisica del sistema sia sempre divisa in 2^{16} frame, quale dimensione minima dovrebbero avere le pagine di questo sistema per essere certi di non dover ricorrere ad una paginazione a più livelli?

Poniamo 35 = 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 (cioè 2^m entry), moltiplicato per la dimensione di una entry (cioè 16 bit, o due byte) deve poter essere contenuto in una pagina/frame, ossia: $2^m \cdot 2^1 \leq 2^n$
Da cui: $m + 1 \leq n$. Poiché $m = 35 - n$, risolvendo il semplice sistema si ha $n \geq 18$, ossia le pagine devono almeno essere grandi 2^{18} = 256 Kbyte.

c) assumendo che nel sistema possano essere presenti al massimo 64K processi, e facendo riferimento ai dati del problema indicati nel punto a), quanto sarebbe grande la IPT del sistema? (Motivate numericamente la vostra risposta)

La IPT ha un numero di entry pari al numero di frame del sistema, e ogni entry della IPT deve contenere il PID di processo (2 byte, dato che 2^{16} = 64K) e il numero di una pagina (3 byte).
Dunque la IPT è grande $2^{16} \cdot$ (2+3) byte = 320 Kbyte.

d) perché i sistemi operativi moderni non usano l'allocazione contigua dello spazio in RAM a partizioni fisse o a partizioni variabili?

Non usano l'allocazione a partizioni fisse perché limita a priori il grado di programmazione e soffre in modo eccessivo del problema della frammentazione interna. Non usano l'allocazione a partizioni variabili perché soffre del problema della frammentazione esterna e costringe periodicamente al ricompattamento dello spazio in RAM.

e) Quali sono gli **svantaggi** della paginazione della memoria? (si consideri il caso di un sistema che implementa la memoria virtuale)

Costo della traduzione degli indirizzi la logici a fisici (con conseguente necessità di un supporto hardware, il TLB, per limitare questi costi
Maggior overhead della memoria principale per ospitare le page table dei processi
Maggior lavoro del SO per amministrare le page table dei processi
Necessità di implementare un meccanismo di rimpiazzamento delle pagine e rischio di thrashing

ESERCIZIO 3 (6 punti)

a) In Unix occupa più spazio un link fisico o un link simbolico? (motivate la vostra risposta)

un link simbolico, perché alloca un index-node in più.

b) perché sono necessari i link simbolici?

Per permettere di costruire collegamenti fra le directory, tra le quali non sono ammessi i link fisici (in Unix, i link simbolici permettono anche link fra file che stanno su partizioni diverse degli hard disk).

c) In quale caso, e perché, l'allocazione concatenata (senza FAT) dello spazio in memoria secondaria è particolarmente **svantaggiosa**?

Nel caso di file molto grandi, perché per leggere i dati contenuti nell'ultima parte del file occorre seguire la catena di blocchi sull'hard disk, ossia effettuare molte operazioni di I/O su un dispositivo lento.

d) In quali casi, rispettivamente, è meglio usare un sistema RAID nella configurazione 0, 1 o 5?

Usiamo il RAID 0 se abbiamo bisogno di massimizzare lo spazio di memorizzazione disponibile e la velocità di accesso ai dati, mentre l'affidabilità non è un requisito fondamentale.
Usiamo il RAID 1 se abbiamo bisogno della massima affidabilità e velocità di accesso ai dati.
Usiamo il RAID 5 se vogliamo un ragionevole compromesso tra affidabilità, velocità di accesso ai dati, e spazio di memorizzazione disponibile.

e) Considerate la seguente sequenza di comandi Unix (assumete che tutti i comandi lanciati possano essere correttamente eseguiti):

```
1:   cd /tmp
2:   mkdir mynewdir
3:   cd mynewdir
4:   echo "ciao" > pippo          // crea un nuovo file di nome pippo contenente la stringa ciao
5:   ln pippo paperino
6:   ln –s pippo pluto
7:   ln topolino paperino
8:   rm pippo
9:   cat pluto                   // cat stampa il contenuto del file passato come argomento
10:  cd ..
11:  mkdir aseconddir
```

Dopo l'esecuzione di tutti i comandi:
qual è il valore del link counter nell'index-node associato al link fisico *paperino*? 1 (perché il comando 7 fallisce)
qual è il valore del link counter nell'index-node associato al link fisico *mynewdir*? 2

cosa possiamo dire del link counter dell'index-node associato al link fisico *tmp*? Che è aumentato di 2, a causa delle entry ".." inserite dentro le sottocartelle mynewdir e aseconddir.
Qual è l'output del comando numero 9? "no such file or directory"