SISTEMI OPERATIVI corso A – 18 giugno 2019

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)

a) Si consideri il problema dei produttori e consumatori, con buffer limitato ad *m* elementi, dove i codici del generico produttore e del generico consumatore sono riportati qui di seguito. Inserite le opportune operazioni di wait e signal necessarie per il corretto funzionamento del sistema, indicando anche i semafori necessari ed il loro valore di inizializzazione.

semafori necessari con relativo valore di inizializzazione:
semaphore mutex = 1;
semaphore full = 0;
semaphore empty = m;

"consumatore"	*"produttore"*
repeat	**repeat**
	<produci dato>
wait(full)	*wait(empty)*
wait(mutex)	*wait(mutex)*
<preleva dato dal buffer>	
	<inserisci dato nel buffer>
signal(mutex)	*signal(mutex)*
signal(empty)	*signal(full)*
<consuma dato>	
forever	**forever**

b) Come può essere semplificato il codice se si sa che c'è un solo processo produttore?

E' possibile rimuovere le operazioni di wait e signal su mutex nel codice del produttore.

c) Vogliamo essere certi che la procedura A eseguita all'interno del processo P_A e la procedura B eseguita all'interno del processo P_B siano eseguite (non importa in quale ordine) prima che venga eseguita la procedura C all'interno del processo P_C. Riportate una soluzione a questo problema usando uno o più semafori, il loro valore di inizializzazione, e le usuali operazioni di wait e signal.

Una possibile soluzione: Semaphor *syncA* = 0; *syncB = 0*

P_A: **P_B:** **P_C:**

A	B	wait(*syncA*) wait(*syncB*)
signal(*syncA*)	signal(*syncB*)	C

d) All'interno di un sistema operativo che implementa la memoria virtuale, un certo processo P è correntemente in stato di "*Waiting for page*", e si sa che, una volta ritornato in stato "*running*" 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, e non genererà più alcun page fault).
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)

Una volta servito il page fault il processo tornerà in coda di ready, dunque solo FCFS e round robin (che non soffrono di starvation) garantiranno la terminazione del processo.

e) Riportate lo pseudocodice che descrive la corretta implementazione dell'operazione di WAIT, e dite cosa fa la system call invocata nel codice.

Si vedano i lucidi della sezione 6.5.2.

f) In quali modi un sistema operativo mantiene sempre un certo controllo della macchina anche mentre non sta girando codice del sistema operativo stesso?

Uso di istruzioni privilegiate per eseguire operazioni delicate
Uso di un timer hardware
Uso di registri appositi per il controllo degli indirizzi usati nelle istruzioni dei programmi utente

ESERCIZIO 2 (7 punti)

In un sistema operativo un indirizzo fisico è scritto su 30 bit, e un frame è grande 16 Kbyte. E' noto che la tabella delle pagine più grande del sistema occupa esattamente un frame.

a) Qual è lo spazio di indirizzamento logico del sistema (esplicitate i calcoli che fate)?

Poiché 16 Kbyte = 2^{14} byte, il numero di un frame del sistema è scritto su 30 – 14 = 16 bit, ossia 2 byte, dunque in un singolo frame possiamo scrivere al massimo 8192 =2^{13} entry da due byte, e lo spazio di indirizzamento logico è pari a $2^{13} * 2^{14} = 2^{27}$ byte.

b) Il sistema deve usare il bit di validità nelle entry delle tabelle delle pagine? (motivate la risposta)

Solo se si vuole implementare la memoria virtuale (cosa non necessaria dato che lo spazio di indirizzamento fisico è maggiore di quello logico)

c) Se il sistema adottasse invece una Inverted Page Table, quanto sarebbe grande questa tabella? (nel rispondere a questa domanda esplicitate eventuali ipotesi che fate)

Per rispondere alla domanda è necessario ipotizzare il numero di bit usati per scrivere il PID di un processo. Assumiamo ad esempio di usare 10 bit. La IPT ha un numero di entry pari al numero di

frame del sistema, e ogni enty contiene il numero di una pagina (13 bit) e il numero di un PID (10 bit). Ne segue che la ITP è grande 2^{16} * 3 byte (assumendo di usare 3 byte per scrivere 13+10 bit)

d) Elencate tre svantaggi dell'uso delle librerie statiche

Non possono essere condivise tra più processi, per cui occupano più spazio in RAM di quelle dinamiche.
Vengono caricate in RAM anche se non sono chiamate, per cui i processi partono più lentamente.
Se vengono aggiornate occorre ricompilare i programmi che le usano

e) E' vera la seguente affermazione (motivate la vostra risposta)? *In un qualsiasi sistema operativo che implementi la memoria virtuale, l'effettivo tempo di esecuzione di un programma dipende molto dal numero di page fault generati dal programma durante la sua esecuzione.*

L'affermazione è vera: poiché il tempo di gestione di un page fault è comunque alto, un programma che generi molti page fault vede aumentare di molto il suo tempo di esecuzione.

f) In un sistema in funzione si osserva in un dato istante che la CPU ha una percentuale di utilizzo del 10%. Si decide allora di mandare in esecuzione un certo numero di nuovi processi CPU bound, ma la percentuale di utilizzo della CPU non aumenta. Come si può spiegare la cosa?

Il sistema è in thrashing (oppure i nuovi processi avviati sono subito andati in deadlock).

ESERCIZIO 3 (6 punti)

L'hard disk **di un sistema operativo Unix** ha la capacità di 256 gigabyte, ed è formattato in blocchi da 1000 (esadecimale) byte. Sull'hard disk è memorizzato un file A grande 3 Megabyte. Nel rispondere specificate le assunzioni che fate.

a) Quante operazioni di I/O su disco sono necessarie per portare in RAM l'ultimo blocco del file A, assumendo che inizialmente sia presente in RAM solo la copia del file directory che "contiene" il file?

L'hard disk contiene $2^{38}/2^{12} = 2^{26}$ blocchi, e sono quindi necessari 4 byte per scrivere in numero di un blocco. Un blocco indice può quindi contenere al massimo 4096/4 = 1024 numeri/puntatori a blocco.

Poiché si adotta l'allocazione indicizzata Unix è sufficiente usare il puntatore di singola in direzione dell'index-node del file, con il quale è possibile tenere traccia di un massimo di circa 4 Megabyte di dati (1024 blocchi da 4 Kbyte ciascuno), quindi, assumendo già in RAM il numero dell'index-node, sono necessarie 3 operazioni di I/O: lettura dell'index-node, lettura del blocco indice puntato dal puntatore singola indirezione, lettura del blocco del file.

b) Sempre facendo riferimento ai dati della domanda precedente, poiché l'hard disk è usato in un sistema Unix, quale sarebbe la dimensione massima che potrebbe avere un file di quel sistema? (è sufficiente riportare l'espressione che permette di calcolare la dimensione del file)

$10 * 4\text{Kbyte} + 1024 * 4\text{Kbyte} + 1024^2 * 4\text{Kbyte} + 1024^3 * 4\text{Kbyte}$

c) Quale/quali dei seguenti comandi modifica il valore del *link counter* dell'index-node associato al file di testo A? (si assuma che tutti i comandi vengono eseguiti correttamente)

1) ls A B; 2) ln –s A B; 3) rm A B; 4) ln A B

i comandi 3) e 4)

d) Sia data una cartella Unix associata all'index node X. Quali sono gli hard link permessi a X?

a) "." Dentro la cartella stessa
b) il nome con cui è stata creata la cartella, all'interno della cartella che la contiene
c) gli hard link ".." nelle eventuali sotto cartelle della cartella stessa.

e) perché è preferibile configurare un sistema RAID al livello 5 invece che al livello 4?

perché il livello 5 distribuisce gli strip di parità tra tutti i dischi del sistema, e in questo modo tutti i dischi vengono sollecitati in maniera uniforme.