SISTEMI OPERATIVI
15 giugno 2011

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

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

ESERCIZI RELATIVI ALLA PARTE DI TEORIA DEL CORSO

ESERCIZIO 1 (5 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 i seguenti:

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**

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.

b) Riportate lo pseudocodice che descrive l'implementazione dell'operazione di Wait.

Si vedano i lucidi della sezione 6.5.2

c) Come è possibile implementare le sezioni critiche contenute nel codice della Wait? (spiegate perché la soluzione che avete indicato è accettabile)

1. Usando il busy waiting (perché, essendo le sezione critiche da implementare molto corte, l'uso del busy waiting produce uno spreco molto limitato di tempo di CPU)
2. Usando la disabilitazione degli interrupt (perché questa verrebbe comunque fatta sotto il controllo di codice del Sistema Operativo, e non di codice utente)

d) Riportate un semplice esempio in pseudo-codice 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;

ESERCIZIO 2 (5 punti)

Si consideri un sistema in cui in una tabella delle pagine di un processo l'indice più grande usabile nella tabella delle pagine di quel processo può essere 7FFF. Un indirizzo fisico del sistema è scritto su 25 bit, e la RAM è suddivisa in 4000 (esadecimale) frame.

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

4000(esadecimale) = 2^{14}, per cui un numero di frame è scritto su 14 bit, e la dimensione di un frame, e quindi di una pagina, è di $2^{\wedge 11}$ byte (25 - 14 = 11). Poiché il numero più grande di una pagina è 7FFF, ci possono essere al massimo $2^{\wedge 15}$ pagine, e lo spazio di indirizzamento logico è di $2^{\wedge 15}$ x $2^{\wedge 11}$ byte (pari a circa 64 megabyte).

(b) Per ciascuna entry di una tabella delle pagine di questo sistema, è necessario memorizzare anche il bit di validità della pagina corrispondente? (motivate la vostra risposta)

Si. Infatti lo spazio di indirizzamento logico è più grande di quello fisico, e il sistema deve implementare anche la memoria virtuale.

(c) In quale caso dobbiamo utilizzare il dirty bit?

Se vogliamo usare un algoritmo di rimpiazzamento delle pagine che ne faccia uso, come ad esempio l'algoritmo della seconda chance migliorato.

(d) Quale soluzione viene comunemente usata per limitare l'inefficienza dell'accesso in RAM tipica dei sistemi che adottano la paginazione della memoria?

Viene usato un TLB, una memoria associativa che limita il tempo necessario alla traduzione di un indirizzo logico in fisico.

(e) Che tipo di codice genera il compilatore di un moderno sistema operativo che implementa la memoria virtuale, e perché?

Codice dinamicamente rilocabile, in modo che il codice possa essere spostato, se necessario, da un punto all'altro della RAM senza dover ricalcolare gli indirizzi usati dalle istruzioni del codice stesso.

ESERCIZIO 3 (4 punti)

Un hard disk ha la capienza di 2^{38} byte, ed è formattato in blocchi da 1024 byte.

a) Quanti accessi al disco sono necessari per leggere l'ultimo blocco di un file A della dimensione di 8192 byte, assumendo che sia già in RAM il numero del primo blocco del file stesso e che venga adottata una allocazione concatenata dello spazio su disco? (motivate la vostra risposta)

9. Ogni blocco infatti memorizza 1020 byte di dati più 4 byte di puntatore al blocco successivo (infatti, $2^{38}/2^{10} = 2^{28}$), per cui sono necessari 9 blocchi per memorizzare l'intero file.

b) Qual è lo spreco di memoria dovuto alla frammentazione interna nella memorizzazione di A (motivate la risposta)?

L'hard disk è suddiviso in $2^{38}/2^{10} = 2^{28}$ blocchi, sono necessari 4 byte per memorizzare un puntatore al blocco successivo, e ogni blocco contiene 1020 byte di dati. Il nono blocco memorizzerà quindi 32 byte del file, e la frammentazione interna corrisponde a 1024 – 32 = 992 byte (988 se si considerano non sprecati i 4 byte del nono blocco che contengono il puntatore, non utilizzato, al blocco successivo)

c) Se si adottasse una allocazione indicizzata dello spazio su disco, quanti accessi al disco sarebbero necessari per leggere l'ultimo byte di un file B grande 300k byte (specificate quali assunzioni fate nel rispondere a questa domanda e motivate la vostra risposta)?

Poiché sono necessari 4 byte per scrivere il numero di un blocco, in un blocco indice possono essere memorizzati 256 puntatori a blocco, e con un blocco indice possiamo indirizzare in tutto $2^8 * 2^{10} = 2^{18} =$ 256K byte. Un solo blocco indice non è quindi sufficiente a memorizzare B. Assumendo una allocazione indicizzata concatenata (ma si ottengono gli stessi risultati con una allocazione indicizzata gerarchica) usando un secondo blocco indice è possibile memorizzare l'intero file. Se il numero del primo blocco indice è già in RAM, per leggere l'ultimo carattere del file sono necessari 3 accessi al disco: lettura del primo blocco indice, lettura del secondo blocco indice, lettura dell'ultimo blocco del file.