Sistemi Operativi
Compito Advanced 7 luglio 2005

1. (a) Che cos'è la multiprogrammazione? Si può realizzare su sistemi uniprocessore?

   (b) In un sistema basato su thread di livello utente, c'è uno stack per ogni thread oppure uno stack per processo? Che cosa succede se i thread sono di livello kernel?

2. Si consideri un sistema con scheduling a priorità con tre code, A, B, C, di priorità decrescente, con prelazione tra code. Le code A e B sono round robin con quanto di 15 e 20 ms, rispettivamente; la coda C è FCFS. Se un processo nella coda A o B consuma il suo quanto di tempo, viene spostato in fondo alla coda B o C, rispettivamente.

   Nelle code A, B, C entrano i seguenti processi:

	coda	arrivo	burst
P_1	A	0	40ms
P_2	C	10	25ms
P_3	B	15	20ms
P_4	A	25	20ms

   Si determini:

   (a) il diagramma di GANTT relativo all'esecuzione dei quattro processi;

   (b) il tempo di attesa medio;

   (c) il tempo di reazione medio.

3. Si consideri la segreteria di uno studio dentistico che si deve occupare di gestire le prenotazioni degli appuntamenti stabilendo a priori quanto tempo allocare per ogni appuntamento in base al tipo di intervento da eseguire. Si vuole cercare di massimizzare il carico di lavoro del dentista (evitando quindi di lasciare tempi morti fra gli appuntamenti). Si deve comunque garantire la possibilità di gestire delle urgenze. Per risolvere situazioni di urgenza è ammesso riorganizzare gli appuntamenti. Si studi un algoritmo di scheduling degli appuntamenti e lo si inquadri in una delle classi di algoritmi di scheduling visti a lezione. Si provi a considerare la situazione in cui le urgenze vanno evase entro determinate deadlines e gli appuntamenti vengano spostati, in prima istanza, solo se le persone coinvolte sono daccordo allo spostamento (tramite accordi telefonici) e, come estrema risorsa, si faccia aspettare un utente che oramai si trova già nello studio dentistico. Alternativamente si semplifichi considerando il caso in cui tutti gli utenti scoprono dello slittamento dell'appuntamento una volta in loco.

4. Si consideri una situazione in cui si hanno 2 processi produttori A e B e 1 processo consumatore che deve consumare gli elementi prodotti dai produttori in alternanza stretta, cioè prima un elemento di A poi uno di B e quindi da capo. Ovviamente non ci si preoccupi di questioni di starvation.

   Si completi il seguente codice concorrente usando opportunamente le primitive `up(&semaph)` e `down(&semaph)`.

Sistemi Operativi
Compito Advanced 7 luglio 2005

```
[dich. e inizializz. semafori/o]

void producer(semaph full, empty) {
    int item;

    while (TRUE) {
        item = produce_item();
        [ sez1 ]
        insert_item(item);
        [ sez2 ]
    }
}

void consumer(void) {
    int item;

    while (TRUE) {
        [ sez3 ]
        item = remove_item();
        [ sez4 ]
        consume1_item(item);
        [ sez5 ]
        item = remove_item();
        [ sez6 ]
        consume2_item(item);
    }
}
```

```
semaph s1full=0, s1empty=1, s2full=0, s2empty=1;

void producer(semaph full, empty) {
    int item;

    while (TRUE) {
        item = produce_item();
        down(&empty);
        insert_item(item);
        up(&full);
    }
}

void consumer(void) {
    int item;

    while (TRUE) {
        down(&s1full);
        item = remove_item();
        up(&s1empty);
        consume1_item(item);
        down(&s2full);
        item = remove_item();
        up(&s2empty);
        consume2_item(item);
    }
}
```

Sistemi Operativi
Compito Advanced 7 luglio 2005

5. Si illustrino le differenze in termini di modalità di connessione e funzionamento, nonché di performance, fra un File Server Stateful (a stato) e uno Stateless (senza stato).

6. Si consideri un sistema di paginazione a 4 livelli con pagine da 4K e 10 bit per ogni livello (quindi indirizzamento logico a 52bit) dove ogni entry nella page table occupi 32 bit, tanti quanti i bits dell'indirizzamento fisico.
   (a) Supponiamo che un processo acceda alle seguenti pagine del suo spazio virtuale: 0x780E20, 0x780F22, 0xFFFFF03FF e 0xFFFFFFFFF. Quanti frame devono essere allocati al processo per contenere queste pagine e quanto serve al sistema di paginazione?

   Le pagine della tabella hanno una dimensione di $32b * 2^{10} = 4KB = 1$ frame.
   Abbiamo sicuramente la pagina del primo livello. Poi visto che i primi 10 bits dei 4 indirizzi sono tutti uguali (a 0) avrò una sola pagina di secondo livello. Inoltre, visto che i primi 2 indirizzi e gli altri 2 hanno i secondi 10 bits rispettivamente uguali fra loro, avrò 2 pagine di terzo livello.
   Per i primi 2 indirizzi, che hanno i terzi e i quarti 10 bits uguali, avrò quindi 1 pagina di quarto livello e 1 frame per contenere i dati indirizzati.
   Per gli altri due, che hanno i terzi 10 bits uguali fra loro, avrò 1 pagina di quarto livello. Poi, visto che i quarti 10 bits sono diversi, avrò altri 2 frames per contenere i dati indirizzati.
   In totale 9 frames.

   (b) Si valuti la differenza di performance sull'accesso medio in memoria (EAT) fra questo sistema e uno con tabella delle pagine invertita. Si assuma un tempo di accesso in cache t_c, hit ratio α, accesso in RAM t_m, un'hash per la tabella invertita che mediamente trova il riferimento in 2 passi, page fault rate p e tempo di recupero pagina da disco t_d. Si assuma inoltre nel caso a livelli di avere una località da far mantenere in memoria principale tutti i primi 3 livelli della tabella che servono.

   pagine invertite Assumiamo (come accade di solito nel caso della tabella invertita) di avere la tabella tutta in memoria, quindi:

   - ad ogni accesso attendiamo che la cache tenti di risolvere l'indirizzo;
   - se va male, con frequenza $1 - \alpha$, dobbiamo accedere alla tabella per la ricerca e il tempo medio di ricerca in tabella è $t_r = 4t_m$ (2 accessi per passo);
   - ora, se va male, con frequenza p dobbiamo accedere al disco;
   - poi (in ogni caso) facciamo l'accesso in RAM.

   Quindi $t_{EAT} = t_c + (1 - \alpha)(t_r + pt_d) + t_m = t_c + (1 + 4(1 - \alpha))t_m + p(1 - \alpha)t_d$.

   4 livelli Viste le dimensioni della tabella *non* possiamo assumere venga mantenuta *tutta* in memoria e le ipotesi dell'esercizio specificano che troverò direttamente in RAM le sole pagine che servono per i primi 3 livelli, quindi:

   - ad ogni accesso attendiamo che la cache tenti di risolvere l'indirizzo;
   - se va male, con frequenza $1 - \alpha$, dobbiamo accedere alla tabella di primo livello per la ricerca, che sta in RAM;
   - analogamente per gli altri 2 livelli;
   - per il quarto livello non abbiamo la pagina necessariamente in RAM per cui con frequenza p dobbiamo prima fare l'accesso a disco per la pagina e, poi, in ogni caso andiamo a fare l'accesso in RAM.

- a questo punto con frequenza p dobbiamo fare l'accesso a disco per il frame coi dati
- infine (in ogni caso) facciamo l'accesso in RAM.

Quindi $t_{EAT} = t_c+(1-\alpha)(3t_m+pt_d+t_m+pt_d)+t_m = t_c+(1+4(1-\alpha))t_m+2p(1-\alpha)t_d$.
La differenza è quindi $p(1-\alpha)t_d$.

7. Si consideri un disco gestito con politica LOOK. Inizialmente, la testina è posizionata sul cilindro 30, ascendente; lo spostamento ad una traccia adiacente richiede 1ms. Al driver di tale disco arrivano richieste per i cilindri 50, 35, 10, 40, rispettivamente agli istanti 0ms, 7ms, 10ms, 33ms. Si trascuri il tempo di latenza.
   (a) In quale ordine vengono servite le richieste?

   All'istante 0, la testina inizia a muoversi alla velocità di 1 traccia/ms verso il cilindro 50. Dopo 7ms, quando arriva la richiesta per il cilindro 35, la testina si trova già oltre, sul cilindro 37, la direzione è ascendente e quindi non viene servita. Stesso discorso per l'istante 10, quando arriva la richiesta per il cilindro 10. La testina continua il movimento verso il cilindro 50, ove vi giunge all'istante $50-30 = 20$ms. Dopo aver servito questa richiesta, ci sono le due richieste a 10 e 35 in sospeso; la testina inverte quindi la direzione, verso la traccia 10. Quando, dopo 13ms, arriva la richiesta per il cilindro 40, è troppo tardi: la testina continua a scendere fino al cilindro 35 (che viene raggiunto all'istante $20+(50-35) = 35$ms) e poi al cilindro 10 (che viene raggiunto all'istante $35+(35-10) = 60$ms). Infine, viene invertita nuovamente la direzione per raggiungere il cilindro 40 all'istante $60+(40-10) = 90$ms. Quindi l'ordine è: 50, 35, 10, 40 (casualmente uguale a quello delle richieste).

   (b) Il *tempo di attesa* di una richiesta è il tempo che intercorre dal momento in cui è sottoposta al driver, a quando viene effettivamente servita. Qual è il tempo di attesa medio per le quattro richieste in oggetto?

   La media è $\frac{(20-0)+(35-7)+(60-10)+(90-33)}{4} = 38.75$ ms.

8. Si consideri un disco con velocità di rotazione pari a 7200 rpm, tempo medio di seek t_s pari a 8 msec, blocchi da 4K. Ogni traccia contiene 720 blocchi. Il file system sia NTFS. Quant'è il tempo per la lettura sequenziale completa di un file di 4.2MB allocato in tre blocchi contigui da 1MB, 2MB e 1.2MB?

   Il primo run è composto da $\lceil \frac{2^{20}B}{4\cdot 2^{10}B} \rceil = 2^8 = 256$ blocchi, in secondo da 512 e il terzo 308. Quindi possiamo assumere che ogni run stia tutto su una traccia per cui, detto t_l il tempo di latenza medio, abbiamo: lettura MFT $t_s+t_l+t_b$, lettura primo run $t_s+t_l+256t_b$, lettura secondo run $t_s+t_l+512t_b$ e lettura terzo run $t_s+t_l+308t_b$ Totale $t = 4t_s+4t_l+1077t_b$.

9. (a) Si illustrino le problematiche che si dovrebbe affrontare per sviluppare un sistema operativo su un multiprocessore NUMA. Si pensi in particolare a dove e come mantenere le strutture dati del kernel.

   (b) Che differenza in scalabilità ci si può aspettare (sempre su questa architettura) fra un Sistema Operativo a microkernel e uno monolitico?

(c) Si possono avere deadlock "locali" dovuti all'uso di risorse del kernel tutte situate in una memoria locale? E ci possono essere deadlock "globali" che coinvolgono risorse del kernel che si trovano in differenti memorie locali? Quale soluzione per evitarli, in caso, appare più adeguata?

Sì è possibile avere deadlocks di entrambe i tipi. Visto che le risorse del kernel sono ben note, e non cambiano facilmente nelle varie release del sistema operativo, la soluzione più adeguata (per evitare deadlocks) è l'ordinamento preventivo delle risorse del kernel: ogni processo può allocare tali risorse solo in ordine crescente.