Sistemi Operativi
Compito Advanced 2 settembre 2005

1. (a) Che cos'è lo spooling? È una caratteristica dei sistemi moderni?

   La tecnica dello spooling consente di effettuare operazioni sulle periferiche di I/O contemporaneamente all'esecuzione di programmi, in modo asincrono. Inizialmente (anni '60) si riferiva sia a dispositivi di input che di output, ora essenzialmente a dispositivi di output. Un'aspetto importante è che lo spooling prevede una coda per ciascun dispositivo su cui accodare i processi in attesa di utilizzo del dispositivo, dando così l'illusione all'utente che ha lanciato il comando (per es. di stampa) che questo sia stato immediatamente completato (anche se in realtà il processo di stampa è stato solo accodato). Lo spooling è utilizzato nei sistemi moderni per per le stampanti.

   (b) Si tracci il classico diagramma a stati di un processo. Quali delle transizioni mancanti tra stati potrebbero verificarsi? In quali circostanze?

   Per il diagramma a stati si vedano i lucidi delle lezioni. Si potrebbero aggiungere transizioni da new a running e da waiting a running, in caso di processi con elevata priorità.

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	B	10	25ms
P_3	C	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;

   55 ms

   (c) il tempo di reazione medio.

   16,25 ms

3. Si consideri una società sportiva che offre i seguenti tipi di corsi: ginnastica artistica, ginnastica ritmica, danza classica, danza moderna. Le richieste di iscrizione sono gestite nel modo seguente. Ci sono quattro operatori con quattro code distinte, ciascuno gestisce un tipo di corso. Le richieste di iscrizione sono gestite in ordine di arrivo sulla coda, però le persone che desiderano iscriversi a più di un corso, dopo essersi iscritte al primo, non serve che facciano di nuovo la coda per iscriversi al corso successivoe. Considerando un calcolatore dove gli operatori sono processori e le persone che desiderano iscriversi sono processi in attesa di una CPU, questo modo di gestione in che tipologia di scheduling rientra?

   A ciascun processore si associano due code di processi, entrambe gestite FCFS, ma con diversa priorità. I processi sono inzialmente messi sulla coda a minor priorità relativa al corso a cui vogliono iscriversi. Nela caso di iscrizione ad un ulteriore corso, i processi vengono messi sulla coda a maggior priorità relativa a questo corso. Non c'é prelazione tra code.

Sistemi Operativi
Compito Advanced 2 settembre 2005

4. Un ascensore viene realizzato (oltre alle parti meccaniche) mediante una scheda a processore sulla quale viene installato un semplice sistema operativo che fornisce multiprogrammazione e semafori.

   Il sistema di funzionamento è strutturato a processi. I motori vengono comandati dal processo `motors_controller` mentre la gestione dei tasti nell'ascensore è affidata al processo `elevator_keypad`. Inoltre ad ogni piano ci sono dei sensori di posizione che vanno ad attivare una apposita routine di interrupt `floor_sensor_int_service` che aggiorna la variabile globale `floor`.

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

```
int floor = 0;

[dich. e inizializz. semafori/o]

void motors_controller(void) {
while (TRUE) {
    [ sez1 ]  // waits for some action
    newfloor = get_action();
    while( newfloor != floor ) {
        if ( newfloor > floor ) activate_motors(UP);
        else activate_motors(DOWN);
        [ sez2 ]  // waits for next floor
        }
    stop_motors();
    [ sez3 ]
    }
}

void floor_sensor_int_service(void) {
    floor = get_floor();
    [ sez4 ]  // signals new floor
}

void elevator_keypad(void) {
while (TRUE) {
    [ sez5 ]  // waits motors controller is idle
    put_action(read_floor_key());  // read_floor_key is blocking
    [ sez6 ]
    }
}
```

```
int floor = 0;

semaph wake = 0, sleep = 1;

void motors_controller(void) {
while (TRUE) {
    down(&wake); // waits for some action
    newfloor = get_action();
    while( newfloor != floor ) {
        if ( newfloor > floor ) activate_motors(UP);
        else activate_motors(DOWN);
        down(&floor_sensor);  // waits for next floor
        }
    stop_motors();
    up(&sleep);
    }
}
```

```
void floor_sensor_int_service(void) {
    floor = get_floor();
    up(&floor_sensor);  // signals new floor
}

void elevator_keypad(void) {
while (TRUE) {
    down(&sleep);  // waits motors controller is idle
    put_action(read_floor_key());  // read_floor_key is blocking
    up(&wake);
    }
}
```

5. Si consideri un sistema ove sia in funzione un algoritmo di deadlock detection munito anche di un sistema automatico di "soluzione" che rimanda in esecuzione un opportuno processo coinvolto nel deadlock dopo avergli tolto tutte le risorse. Il meccanismo di gestione del sistema è in grado di fornire statistiche sull'overhead medio di sistema p_O (espresso in percentuale al giorno) dovuto al suo intervento, nonché del tempo pagato in termini di riesecuzione del processo sacrificato p_R e della frequenza di deadlocks p_D.

   Si vuole valutare se sia il caso di rimuovere il sistema di deadlock solution (per non pagarne l'overhead) stabilendo in funzione dei precedenti parametri quanto debba essere (mediamente) il limite del tempo α (espresso in percentuale di giorno) affinché i tecnici si accorgano della presenza di un deadlock, lo risolvano manualmente e il sistema riesca poi a recuperare il tempo perso a causa di detto intervento.

   Il sistema produce un lavoro utile di $1 - p_O - p_R$ ogni giorno. La percentuale di tempo sprecata senza di lui sarebbe αp_D, quindi il lavoro utile sarebbe $1 - \alpha p_D$ da cui si ricava $\alpha < \frac{p_O + p_R}{p_D}$

6. Il meccanismo di file servicing NFS della Sun è Stateful (a stato) o Stateless (senza stato)?

   Tale scelta compiuta dai progettisti è giustificata in termini di performance o dovuta ad altri fattori?

   Se è stata compiuta la scelta più performante, a quale altre caratteristiche si è rinunciato? Se è stata compiuta la scelta meno performante, quali accorgimenti sono stati adottati in modo da alleviare il problema?

   Il protocollo NFS fornisce controlli di coerenza? In caso non lo faccia come vengono fornite queste funzionalità?

   ☐

7. Si consideri un sistema di paginazione a 3 livelli con pagine da 4K e 14 bit per ogni livello (quindi indirizzamento logico a 54bit) 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 $32b * 2^{14} = 64KB = 16$ frames. In ogni frame ci sono 1024 entries della page table.
   Visto che i primi 14 bits dei 4 indirizzi sono tutti uguali (a 0) avrò 1 solo frame della pagina di primo livello (quello che contiene lo 0).
   Poi visto che i secondi 14 bits di tutti e 4 gli indirizzi sono nel range 0-3FF mi basterà 1 solo frame della pagina di secondo livello. Inoltre, i primi 2 indirizzi e gli altri 2 hanno i

secondi 14 bits rispettivamente uguali fra loro, quindi avrò 2 pagine di terzo livello. Per entrambe queste 2 pagine mi basta 1 frame ciascuna (frame 0 per l'entry 780 in un caso e frame F per le entries 3FF0 e 3FFF nell'altro)

Per i primi 2 indirizzi, che hanno i terzi 14 bits uguali, avrò 1 frame per contenere i dati indirizzati.

Per gli altri due, che hanno i terzi 14 bits diversi, avrò 2 frames per contenere i dati indirizzati.

In totale 7 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 1.5 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 2 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 $t_r = 3t_m$ (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 $t_{EAT} = t_c + (1-\alpha)(t_r + pt_d) + t_m = t_c + (1 + 3(1-\alpha))t_m + p(1-\alpha)t_d$.

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 2 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 il secondo livello;
- per il terzo 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 $t_{EAT} = t_c+(1-\alpha)(2t_m+pt_d+t_m+pt_d)+t_m = t_c+(1+3(1-\alpha))t_m+2p(1-\alpha)t_d$.

La differenza è quindi $p(1-\alpha)t_d$.

8. Si consideri un disco gestito con politica SSTF. Inizialmente, la testina è posizionata sul cilindro 30; 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 sul cilindro 37 e quindi questa richiesta prende il soppravvento sull'altra, che viene accodata. All'istante 9

la 35 viene evasa e si riprende la 50. All'istante 10, quando arriva la richiesta per il cilindro 10 servono 14 cilindri per la 50 e 26 per la 10, quindi continua la 50, che viene poi evasa all'istante $10 + 14 = 24$ms. A questo punto si riprende la 10 fino all'istante 33 in cui arriva la rihiesta del cilindro 40. Ci troviamo in questo istante sul cilindro 41 quindi continuiamo e evadiamo la 40 dopo 1ms. Arriviamo infine alla 10 dopo altri 30ms, istante 64ms. Quindi l'ordine è: 35, 50, 40, 10.

(b) Qual è il tempo di attesa medio per le quattro richieste in oggetto?

La media è $\frac{(24-0)+(9-7)+(34-33)+(64-10)}{4} = 20.25$ ms.

9. Si consideri un disco con velocità di rotazione pari a v_r rpm, tempo medio di seek t_s, con 3MB in ogni traccia, sul quale si impiega un File System EXT2 con blocchi da 4KB e puntatori a 4B.

   Quant'è il tempo per la lettura sequenziale completa di un file di 4MB allocato in modo contiguo per quanto possibile?

Il file sarà messo su 2 traccie, una parte da 3MB (quindi $\frac{3 \cdot 2^{20} B}{4 \cdot 2^{10}} = 768$ blocchi) e l'altra da 1MB (256 blocchi).

Sia t_l il tempo di latenza medio e t_b il tempo di lettura di un blocco.

Iniziamo a leggere l'inode (che entra in cache) impiegando $t_s + t_l + t_b$.

Quindi andiamo a leggere i primi 10 blocchi diretti (che sono allocati sequenzialmente) impiegando $10t_b$. Ora dobbiamo andare a leggere il blocco del primo livello indiretto (che entra in cache) impiegando $t_s + t_l + t_b$. Visto che questo blocco contiene 1024 puntatori è abbastanza grande per tutto il resto del file. Andiamo quindi a leggere gli altri 758 blocchi della prima traccia contigua impiegando $t_s + t_l + 758t_b$. Ci portiamo quindi sull'altro troncone del file e lo finiamo di leggere impiegando $t_s + t_l + 256t_b$.

Totale $t = 4t_s + 4t_l + 1026t_b$.