1. (a) Che cos'è un'interruzione (interrupt)? Quali sono i passi fondamentali eseguiti dal sistema operativo per gestire un'interruzione?

   (b) Qual è la differenza tra un'interrupt e una trap?

   Risposta:

   (a) Un'interruzione è un segnale inviato da un controller alla CPU per informare il sistema del verificarsi di un evento (es.: terminazione di un'operazione di I/O). I passi fondamentali da eseguire per gestire un'interruzione sono: salvare i registri della CPU, impostare un contesto per la procedura di servizio, inviare un ack al controllore degli interrupt (per avere interrupt annidati), copiare la copia dei registri nel PCB, eseguire la procedura di servizio (determinata in base al tipo di interrupt), cambiare lo stato a un processo in attesa (e chiamare lo scheduler di breve termine), organizzare un contesto per il processo successivo, caricare i registri del nuovo processo dal suo PCB, continuare il processo selezionato.

   (b) Una trap (o interruzione software) è un interrupt generato da software, causato o da un errore o da una esplicita richiesta dell'utente (istruzioni TRAP, SVC).

2. (a) Può un processo a priorità bassa influire sull'esecuzione di un processo a più alta priorità? Se sì, si diano alcuni esempi.

   (b) Il problema descritto sopra può verificarsi con thread di livello utente?

   (c) Può verificarsi se l'algoritmo di scheduling è round robin?

   Risposta:

   (a) Sì, un processo a bassa priorità può influire sull'esecuzione di un processo a più alta priorità grazie al fenomeno noto come "inversione di priorità". In questo caso infatti se una data risorsa viene assegnata ad un processo a bassa priorità e quest'ultimo non la rilascia in tempo, può succedere che venga sospeso in favore di un processo a maggiore priorità. Se anche quest'ultimo dovesse necessitare della medesima risorsa per proseguire la propria esecuzione, si troverebbe ad essere bloccato dal processo a minore priorità detentore della risorsa contesa. In questo modo, pur vedendosi assegnare la CPU, non potrebbe proseguire, mentre il processo a bassa priorità non riuscirebbe mai ad andare in esecuzione, completando il proprio task e liberando la risorsa contesa.

   (b) Sì, il fenomeno descritto al punto precedente può verificarsi anche con i thread a livello utente con in più il rischio di bloccare tutti i thread utente relativi allo stesso processo.

   (c) Utilizzando l'algoritmo di scheduling RR invece non si corre il rischio che avvenga il problema descritto in precedenza in quanto tale algoritmo assegna ciclicamente lo stesso quanto di tempo di CPU a tutti i processi in coda.

3. 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 10 e 15 msec, rispettivamente; la coda C è FCFS.

Se un processo nella cosa A o B consuma il suo quanto di tempo, viene spostato in fondo alla coda B o C, rispettivamente.

(a) Di quale tipo di algoritmo di scheduling si tratta?

(b) Nelle code A, B, C entrano i seguenti processi:

- P_1 arriva nella coda A all'istante 0 e ha CPU burst di 30 msec;
- P_2 arriva nella coda C all'istante 5 e ha CPU burst di 20 msec;
- P_3 arriva nella coda B all'istante 15 e ha CPU burst di 15 msec;
- P_4 arriva nella coda A all'istante 20 e ha CPU burst di 15 msec.

Si determini:

i. il diagramma di GANTT relativo all'esecuzione dei quattro processi;

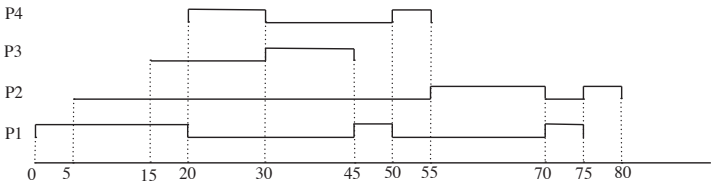
ii. il tempo di attesa medio;

iii. il tempo di reazione medio.

Risposta:

(a) L'algoritmo di scheduling descritto è del tipo a priorità con code multiple e possibilità di retroazione (feedback). Mentre le varie code hanno priorità fisse, i processi possono migrare da una coda all'altra.

(b) i. Nell'ipotesi che il processo prelazionato vada in fondo alla sua coda con il resto del quanto, il diagramma di GANTT relativo all'esecuzione dei quattro processi è il seguente:



ii. il tempo di attesa medio è dato da: $\Delta^1_{att} = 25 + 20 = 45$, $\Delta^2_{att} = 50 + 5 = 55$, $\Delta^3_{att} = 15$, $\Delta^4_{att} = 20$, $\Delta_{att} = \frac{135}{4} = 33,75\ ms$.

iii. il tempo di reazione medio è dato da: $\Delta^1_{reaz} = 0$, $\Delta^2_{reaz} = 50$, $\Delta^3_{reaz} = 15$, $\Delta^4_{reaz} = 0$, $\Delta_{reaz} = \frac{65}{4} = 16,25\ ms$.

4. (a) Si illustrino brevemente i passi eseguiti dai driver dei dispositivi.

(b) Cosa si intende dicendo che un driver deve essere *rientrante*?

(c) I driver possono eseguire delle system call durante il loro funzionamento? Perché?

Risposta:

(a) I passi eseguiti dai driver dei dispositivi sono i seguenti:

i. Controllare i parametri passati.

ii. Accodare le richieste in una coda di operazioni (soggette a scheduling).

iii. Eseguire le operazioni, accedendo al controller.

iv. Passare il processo in modo wait (I/O interrupt-driven), o attendere la fine dell'operazione in busy-wait.

v. Controllare lo stato dell'operazione nel controller.

vi. Restituire il risultato.

(b) Un driver si dice rientrante se, mentre viene eseguito, può essere lanciata una nuova esecuzione del suo codice.

(c) I driver non possono eseguire system call (sono in uno strato software del sistema operativo sottostante), ma possono accedere ad alcune funzionalità del kernel (es: allocazione memoria per buffer di I/O).

5. (a) Si spieghi brevemente come funziona il meccanismo di allocazione indicizzata nel filesystem.

   (b) Il meccanismo di allocazione indicizzata soffre di frammentazione interna o esterna?

   (c) Con il meccanismo di allocazione indicizzata è meglio avere un blocco indice grande o piccolo? Si discutano vantaggi e svantaggi dei due casi.

   Risposta:

   (a) Nel caso dell'allocazione indicizzata si mantengono tutti i puntatori ai blocchi di un file in una tabella indice, memorizzata a sua volta in un blocco su disco (blocco indice). In questo modo per accedere all'i-esimo blocco del file è sufficiente seguire l'i-esimo puntatore del blocco indice. Inizialmente (in fase di creazione del file) tutti gli indici sono inizializzati con un valore (-1) indicante che non vi è nessun blocco dati puntato. In questo modo, per allocare un nuovo blocco al file, è sufficiente assegnare l'indirizzo di un blocco dati disponibile alla prima entry del blocco indice marcata con -1. Così, sacrificando un blocco per la tabella indice, è facile implementare l'accesso random ai file ed evitare il problema della frammentazione esterna.

   (b) Organizzando lo spazio in blocchi di dimensione uguale, può soffrire unicamente di frammentazione interna (relativamente all'ultimo blocco di ogni file).

   (c) Per quanto riguarda il problema delle dimensioni del blocco indice, bisogna tener presente che è una struttura supplementare che implica un certo *overhead*. Quindi un blocco indice di piccole dimensioni riduce tale sovraccarico (in termini di spreco di spazio e di velocità di ricerca al suo interno), mentre un blocco indice di grandi dimensioni consente di supportare anche file di grandi dimensioni.

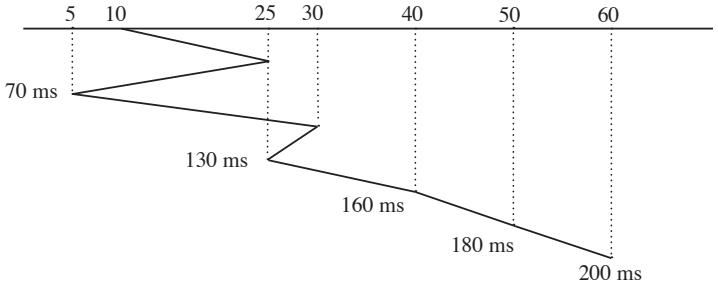
6. Si consideri un disco gestito con politica SSTF. Inizialmente la testina è posizionata sul cilindro 10; lo spostamento ad una traccia adiacente richiede 2 ms. Al driver di tale disco arrivano richieste per i cilindri 60, 5, 50, 40, 25, rispettivamente agli istanti 0 ms, 30 ms, 60 ms, 110 ms, 120 ms. Si trascuri il tempo di latenza.

   (a) In quale ordine vengono servite le 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?

Risposta:

(a) Le richieste vengono soddisfatte nell'ordine: 5, 25, 40, 50, 60, come risulta dal seguente diagramma:



(b) Il tempo di attesa medio per le cinque richieste in oggetto è $\frac{(70-30)+(130-120)+(160-110)+(180-60)+(200-0)}{5} = \frac{40+10+50+120+200}{5} = \frac{420}{5} = 84\ ms.$

Il punteggio attribuito ai quesiti è il seguente: 2, 2, 2, 2, 2, 1, 5, 2, 2, 2, 2, 2, 2, 3, 2 (totale: 33).