Es semafori

Implementare tramite semafori una barriera di sincronizzazione. La barriera blocca tutti i processi che la raggiungono fino all'arrivo dell'ultimo, che li sblocca.

Quindi supponendo n processi P1..Pn, i primi n-1 fanno del lavoro locale e poi si mettono in attesa, fino all'arrivo dell'ultimo che sblocca gli n-1 precedenti.

Nei primi computer tutte le istruzione di lettura e scrittura erano gestite direttamente dalla CPU (non esisteva il DMA).
Quali implicazioni aveva tutto ciò sulla multiprogrammazione?

- Quali delle seguenti istruzioni dovrebbero essere permesse solo in *modalità kernel* e per quale motivo:

a) disabilitazione di tutti gli interrupt

b) lettura del tempo di sistema

c) modifica del tempo di sistema

d) modifica dell'immagine di memoria di un processo

Considerate un sistema con due CPU, ognuna processa due thread (in hyperthreading). Tre programmi P1, P2 e P3 partano assieme e hanno tempi di computazione di 5, 10, e 20 msec, rispettivamente. Quanto sarà il tempo necessario al completamento di tutti i programmi?

Assumete che tutti programmi sono 100% CPU bound, non si bloccano durante l'esecuzione e non cambiano la CPU a loro inizialmente assegnata

Quale tipo di multiplexing (tempo, spazio o entrambi) può essere usato per condividere le seguenti risorse:

- CPU
- Memoria
- Disco
- Scheda di rete
- Stampante
- Tastiera
- Display

Considerate un sistema capace di mantenere in memoria centrale al massimo cinque programmi. Questi programmi in media sono in attesa di I/O metà del tempo. Determinate la percentuale di tempo di CPU sprecato.

Un computer ha 4 GB di RAM di cui il SO occupa 512 MB. Per semplicità ciascun processo occupa 256 MB e ha le stesse caratteristiche degli altri. Se si vuole ottenere un utilizzo della CPU del 99%, quale sarà la percentuale massima di tempo di attesa I/O tollerabile?

SOLUZIONI

Es semafori

Implementare tramite semafori una barriera di sincronizzazione. La barriera blocca tutti i processi che la raggiungono fino all'arrivo dell'ultimo, che li sblocca.

Quindi supponendo n processi P1..Pn, i primi n-1 fanno del lavoro locale e poi si mettono in attesa, fino all'arrivo dell'ultimo che sblocca gli n-1 precedenti.

Es semafori

```
Sem S;
mutex m;

SemInit(S, 0)     // Inizializzo semaforo a 0
SemInit(m, 0)     // Inizializzo semaforo a 1
shared int arrive = 0  // Contatore condiviso dei processi arrivati alla barriera,
inizialmente 0

// funzione eseguita da ciascun processo/thread
Synch_wall(N){       // N: numero componenti della barriera
      down(m);       // aggiornamento var. condivise va protetto
      arrive ++;     // ulteriore processo arrivato
      if(arrive<N){        // se non sono l'ultimo
        up(m);      // DEVO liberare mutex, altrimenti altri bloccati; prima di….
        down(S);   }     // …bloccarmi sulla barriera
      else {               // sono l'ultimo …
        for(int i=1; i<N; i++)    // … sveglio tutti gli N-1 precedenti
           up(S);
        arrive =0;
        up(m)  }
}
```

Nei primi computer tutte le istruzione di lettura e scrittura erano gestite direttamente dalla CPU (non esisteva il DMA). Quali implicazioni aveva tutto ciò sulla multiprogrammazione?

Il motivo principale per l'uso della multiprogrammazione è impegnare la CPU in lavoro utile quando è in attesa del completamento dell'I/O. Senza DMA, la CPU sarebbe occupata a fare I/O e quindi non potrebbe passare a fare altro.

- Quali delle seguenti istruzioni dovrebbero essere permesse solo in *modalità kernel* e per quale motivo:

a) X disabilitazione di tutti gli interrupt

b) lettura del tempo di sistema

c) X modifica del tempo di sistema

d) X modifica dell'immagine di memoria di un processo

Considerate un sistema con due CPU, ognuna processa due thread (in hyperthreading). Tre programmi P1, P2 e P3 … Quanto sarà il tempo necessario al completamento di tutti i programmi?

Dipende: se P1 e P2 sono schedulati sulla stessa CPU e P3 sull'altra, si avrà max(5+10, 20)=20. Se P1 e P3 sono schedulati sulla stessa CPU e P2 sull'altra, si avrà max(5+20, 10)=25, ecc… Da notare che il tempo di processamento di programmi sulla stessa CPU in hyperthreading si somma in quanto si ha uno pseudoparallelismo (vedere sezione 1.3.1 "Processors" del libro), mentre con CPU diverse si ha un reale parallelismo e perciò domina l'esecuzione più lenta (max).

Quale tipo di multiplexing (tempo, spazio o entrambi) può essere usato per condividere le seguenti risorse:

- CPU t
- Memoria s
- Disco s
- Scheda di rete t
- Stampante t
- Tastiera t
- Display ts

Considerate un sistema capace di mantenere in memoria centrale al massimo cinque programmi. Questi programmi in media sono in attesa di I/O metà del tempo. Determinate la percentuale di tempo di CPU sprecato.

La CPU non può fare nulla quando tutti i programmi sono in attesa di I/O, ciò succede con probabilità $(½)^5=1/32 \sim 0.031$

Chp 2 es 6

Un computer ha 4 GB di RAM di cui il SO occupa 512 MB. Per semplicità tutti i processi occupano 256 MB e hanno le stesse caratteristiche. Se si vuole ottenere un utilizzo della CPU del 99%, quale sarà la percentuale massima di tempo di attesa I/O tollerabile?

In memoria c'è spazio al massimo per 14 processi. Se ciascun processo fa I/O con probabilità p, la probabilità che tutti aspettino I/O è p^{14}. Uguagliando a 0.01, si ottiene l'equazione $p^{14} = 0.01$. Risolvendola si ha p = 0.72, perciò è possibile tollerare fino al 72% di attesa per I/O.

Vale la seguente proprietà

U = 1- p^n

da cui:

1-U= p ^ n

Sostituendo i dati del problema:

$1-0.99=p^{14}$

$p^{14} = 0.\ 01$

$(p^{14})^{1/14} = (0.01)^{1/14}$

$p \sim 0.72$