scritto-2022-01-17-soluzione-luizo

c1

Scrivere il monitor multibuf che implementi un buffer limitato (MAX elementi) di oggetti di tipo T che implementi le seguenti procedure entry:
`void add(int n, T objexts[]);`
`void get(int n, T objects[]);`
La funzione add deve aggiungere al buffer gli n oggetti passati col parametro objects. La funzione get deve prendere dal buffer in modalità FIFO i primi n elementi presenti nel buffer e copiarli negli elementi del vettore objects.
Entrambe le funzioni devono attendere che vi siano le condizioni per poter essere completate: che ci siano n elementi liberi per la add, che ci siano n elementi nel buffer per la get. Non sono ammesse esecuzioni parziali: mentre attendono le rispettive condizioni nessun elemento può essere aggiunto o rimosso dal buffer.
La definizione del problema C.1 presenta casi di possibile deadlock? quali?

🐰Soluzione

```
Monitor multibuf{

  define MAX = 10; //arbitrario

  condition ok2add;
  condition ok2get;

  T buffer[MAX];

  int current=0;

  void add(int n, T objexts[]) {
    if (current+n >= MAX)
      ok2add.wait();

    int count=0;
    while(count<=n)
      buffer[current]= objexts[count];
      current++;
      count++;

    if (current-n >= 0)
      ok2get.signal()
  }


  void get(int n, T objects[]){
      if (current-n < 0)
        ok2get.wait()

      int count=0;
      while(count =< n)
        objexts[count]= buffer[current];
        current--;
        count++;

      if (current+n < MAX)
      ok2add.signal();
  }

}
```

c'è deadlock nel caso in cui i processi riempiano completamente il buffer e facciano tutti una richiesta aggiuntiva, si bloccheranno tutti e nessuno riuscirà a rilasciare le risorse nel buffer, tutti bloccati.

se non arriva un processo che svuota il buffer, i processi rimarranno in deadlock.

c2

Un servizio di message passing sincrono senza selezione del mittente prevede una API con due funzioni:
`sasend(msg_t msg, pid dest);`

`msg_t sarecv(void);`

La funzione sarecv restituisce il primo messaggio ricevuto da qualsiasi mittente ed è bloccante se non ci sono messaggi pendenti. la funzione sasend si blocca fino a quando il messaggio msg non viene ricevuto dal processo dest.
Dato quindi un servizio di message passing sincrono senza selezione del mittente implementare un servizio di message passing sincrono (standard, quello definito nel corso) senza fare uso di processi server

---

🐰Soluzione

le strutture dati sono condivise con tutti i processi visto che siamo dal lato server tecnicamente no?

altrimenti non vedo come sia possibile una roba del genere

la soluzione di Alice mi sembra insensata e ChatGPT è rincoglionito

non si può fare una sega con le cose sincrone

OH CAZZO CE L'HO

UN'ORA CI HO MESSO MA CE L'HO!!

```
// assumo la coda messaggi non ricevuti sia gestita in FIFO
// db -> struttura dati che contiene i messaggi

define ANY = -1; // arbitrary

msg_t {
msg
pid
}

ssend(msg_t msg, pid dest) {
  sasend(<msg,getpid(), RECEIVED >, dest);
    // si sblocca automaticamente quando dest fa' sarecv()
  while (true)
    //appena si sblocca, rimane ad ascoltare finché
    // non gli ritorna l'ack del messaggio inviato
    <ack, pid> = sarecv();
    if (pid = dest, msg= ack)
      return
    else
      // ogni volta che fetcha un altro messaggio
      // lo aggiunge al db locale
      db.add(ack, pid);
}

msg_t srecv(dest) {
    // controllo nel db se ho il messaggio
    forall entry in db
```

```
      // se la trovo, non serve ricevere, ce l'ho già
      // mando l'ack e tolgo il msg dal database
      // se il destinatario è ANY prende il primo messaggio
      // contenuto nel db
      if(entry.second == dest || dest == ANY)
        sasend(entry.msg, dest)//sending ack
        db.remove(entry)
        return entry;
    // se non trovo il messaggio, aspetta di riceverlo
    // se ricevo altri messaggi mettili nel db
    while (true)
      mymsg = sarecv();
      if(mymsg.pid == dest || dest == ANY)
        return mymsg;
      db.add(mymsg)
}
```

g1

Considerare i seguenti processi gestiti mediante uno scheduler preemptive a priorità statica su una macchina biprocessore SMP:
P1: cpu 4 ms; I-O 4 ms; cpu 2 ms
P2: cpu 2 ms; I-O 4 ms; cpu 5 ms
P3: cpu 5 ms; I-O 3 ms; cpu 3 ms
P4: cpu 10 ms; I-O 1 ms
l'Input-Output avviene su un'unica unità. Per il processo P1 ha priorità massima seguito da P2, P3 e P4 in sequenza decrescente, le richieste di I/O sono gestite in ordine FIFO. Calcolare il tempo necessario a completare l'esecuzione dei 4 processi.

🐰 Soluzione

	0	1	2	3	4	5	6
CPU 1	1	1	1	1	4	4	2
CPU 2	2	2	3	3	3	3	3
I/O			2	2	2	2	1

g2

rispondere alle seguenti domande (motivando opportunamente le risposte):
a) Perché viene usata la paginazione per implementare la memoria virtuale?
b) L'algoritmo del banchiere dato uno stato di allocazione delle risorse restituisce un valore binario: safe o non safe.
In quali casi il sistema operativo esegue l'algoritmo del banchiere? Cosa succede se il risultato è safe e cosa se il risultato è non-safe?
c) Fornire esempi di file system con allocazione contigua, e spiegare perché sarebbe inefficiente usare altri metodi di allocazione nei casi d'uso tipici di questi file system.
d) perché l'invenzione degli interrupt ha reso i sistemi operativi più efficienti?

🐰 Soluzione

a) perché spostare i dati è più semplice con le pagine rispetto agli altri metodi di gestione memoria

b) l’algoritmo viene eseguito quando ci sono richieste multiple di risorse multiple. Se il risultato dell’algoritmo è safe, non c’è rischio che il sistema vada in deadlock. Al contrario, essere nello stato unsafe è condizione necessaria ma non sufficiente per avere un deadlock.

c) viene usata per cd e dvd di sola lettura, in quanto il problema della frammentazione esterna non viene mai a crearsi in questo caso, mentre tutte le altre politiche di gestione della memoria presentano altri svantaggi non presenti nell’allocazione contigua.

d) Prima dell’invenzione degli interrupt il ciclo FDE non poteva essere interrotto, i processi in attesa di I/O occupavano la CPU inutilmente (Spooling). Con l’avvento degli interrupt i processori hanno ridotto i tempi di busy waiting drasticamente.