Esercitazione finale per il corso di Sistemi Operativi (A.A. 2002/2003)

1

Descrizione dell'esercitazione

- L'esercitazione consiste nell'implementare la soluzione corretta di una versione generalizzata del problema dei cinque filosofi, in cui cioé il numero di filosofi presenti è un parametro passato in input al sistema. (fate riferimento alla soluzione per il caso particolare di cinque filosofi vista a lezione)
- Nel seguito, chiameremo *accademia* il programma che inizializza, mette a disposizione le risorse necessarie, e fa partire i processi *filosofi*.

2

Il processo *accademia*

- L'*accademia* viene lanciata così:

  $> accademia N

- dove N è il numero di filosofi da creare ($3 \leq N \leq 10$)
- Ricordate, la soluzione vista per il problema dei cinque filosofi è immediatamente generalizzabile ad un numero arbitrario di filosofi

3

Il processo *accademia*

- Quando viene lanciato, il processo *accademia* deve innanzitutto allocare le risorse necessarie a gestire i vari filosofi, ed in particolare:
  - una coda di messaggi per gestire le comunicazioni tra i filosofi e l'accademia
  - un'area di memoria condivisa di dimensione adeguata a contenere l'array STATE usato per registrare in ogni istante lo stato di ogni filosofo
  - Un numero sufficiente di semafori necessari ad implementare l'intero sistema (ricordatevi che con una semget potete prelevare un intero array di semafori)

4

Il processo *accademia*

- *accademia* lancia poi i vari filosofi forkandosi N volte, in modo che ogni figlio esegua (mediante una execl) il codice del programma *filosofo*.
- dopo aver generato i filosofi, *accademia* esegue un loop infinito, in cui si mette in attesa sulla coda della richiesta di un pasto da parte di ogni *filosofo* che è entrato nello stato **mangia**

5

Il processo *accademia*

- Ogni volta che *accademia* riceve un messaggio (che contiene il nome di un piatto richiesto da un *filosofo*),
- accademia si forka, e mentre il figlio (che chiameremo *servo*) gestisce la richiesta del piatto, *accademia* ritorna in attesa della prossima richiesta in arrivo sulla coda.
- il *servo* attende un tempo random di 1-5 secondi (che simula il tempo necessario a preparare il piatto richiesto), poi invia sulla coda, al *filosofo* che ne ha fatto richiesta, il piatto preparato, e termina.

6

I processi *filosofo*

- Ogni *filosofo* lanciato, esegue all'infinito un loop in cui:
  - si mette a **pensare** (attende un tempo random da 1 a 10 secondi)
  - dopo aver pensato diviene **affamato**, e cerca di prelevare le due forchette che ha a destra e a sinistra
  - quando è riuscito ad entrare in possesso delle due forchette, il *filosofo* può **mangiare**. Ordina all'accademia un piatto (scelto a caso tra cinque possibili, ad esempio sushi, sashimi, sukiyaki, futomaki e makizushi... ok, ok, potete sceglierne altri più comuni) usando la coda di messaggi, e attende sulla stessa coda che il piatto gli venga consegnato. Ricevuto il piatto lo mangia (attende un tempo random da 1 a 5 secondi), rilascia le forchette e si rimette a pensare.

7

Terminazione del sistema

- l'*accademia* deve poter essere chiusa inviando al processo un segnale opportuno (ad esempio SIGUSR1).
- Alla ricezione del segnale, *accademia* deve:
  - inviare un segnale di terminazione ad ogni *filosofo,* e attenderne la terminazione
  - inviare un segnale di terminazione immediata agli eventuali *servi* che stanno preparando piatti
  - rimuovere tutte le risorse in uso, e terminare.
- Quando un *filosofo* riceve il segnale di terminazione, stampa in output un messaggio adeguato, e termina.

8

Monitoraggio del sistema

- Il funzionamento del sistema deve poter essere seguito su terminale mediante opportune stampe dei vari processi (scegliete voi qualcosa di adeguato).
- Inoltre, *accademia* e ogni *filosofo* devono mantenere un file di log (un file distinto per ogni *filosofo* e per *accademia*) in cui vengono registrate tutte le fasi della vita del relativo processo.

9

Esempio di log file generato da *accademia*

- prelevo le risorse
- faccio partire i N filosofi
- mi metto in attesa delle richieste dei filosofi
- 547 (oppure, filosofo 5) richiede un futomaki
- (servo 654) preparo un futomaki (tempo necessario 3 secondi)
- 568 (oppure, filosofo 7) richiede un sushi
- (servo 670) preparo un sushi (tempo necessario 4 secondi)
- (servo 654) invio un futomaki al 547 (o al filosofo 5) e termino
- ...
- ...
- ...

10

Esempio di log file generato da *accademia*

- ...
- ricevuto messaggio di terminazione
- invio messaggi di terminazione ai filosofi
- filosofo 5 (oppure, 547) terminato
- filosofo 1 (oppure, 512) terminato
- ...
- ...
- filosofo 3 (oppure, 520) terminato
- termino i servi
- rimuovo le risorse e termino

11

Esempio di log file generato dal *filosofo i-esimo*

- penso per 7 secondi
- sono affamato: cerco di prelevare le forchette
- forchette prelevate: ordino un sushi
- sushi arrivato, mangio per 2 secondi
- rilascio le forchette e torno a pensare
- penso per 3 secondi
- sono affamato: cerco di prelevare le forchette
- forchette prelevate: ordino un sukiyaki
- sukiyaki arrivato, mangio per 5 secondi
- rilascio le forchette e torno a pensare
- penso per 4 secondi
- ...
- ho ricevuto l'ordine di terminare. Adieu...

12

Monitoraggio del sistema

- Infine, deve essere previsto un programma separato (chiamiamolo *stato*) che, lanciato da terminale come un normale comando, stampa su video lo stato corrente di ogni filosofo.
- Ovviamente, *stato* deve accedere alla memoria condivisa per conoscere la situazione dei vari filosofi, e lo deve fare in modo mutuamente esclusivo.
- **Attenzione**: se *stato* viene lanciato quando l'accademia non sta funzionando, il comando deve restituire un messaggio del tipo: L'ACCADEMIA E' CHIUSA

13

Monitoraggio del sistema

- Un possibile output di *stato* sarà quindi del tipo:

```
$> stato
filosofo 0: pensa
filosofo 1: pensa
filosofo 2: mangia
filosofo 3: affamato
filosofo 4: mangia
```

- Ovviamente, non deve mai accadere che due filosofi adiacenti stiano mangiando

14

Osservazioni e consigli

- Naturalmente, dovete fare riferimento all'algoritmo che abbiamo visto a lezione per implementare una soluzione corretta del problema.
- Potete usare delle costanti intere per rappresentare i tre stati in cui si può trovare un filosofo, e anche per indicare i vari piatti.
- Nella funzione *prendi_le_due_forchette_e_mangia();* il generico filosofo:
  - sceglie a caso un piatto fra cinque possibili
  - invia la richiesta del piatto all'accademia e attende di ricevere il piatto
  - mangia (attende per un tempo di 1—5 secondi), e torna a pensare

15

Osservazioni e consigli

- l'array STATE deve ovviamente essere contenuto in memoria condivisa. come fare ad allocarlo, visto che la sua dimensione dipende da N?

preleva lo spazio necessario a memorizzare n interi

```
int n, *punt; // n conterrà il valore passato in input ad accademia
shmid = shmget(KEY, n*sizeof(n),0666);
punt = (int *)shmat(shmid, 0, SHM_RND); // e a questo punto...
punt[1] = 5;     // ora posso usare punt come un normale array
punt[n-1] = 47;          // ma ecco anche come si può usare punt:
printf("secondo elemento dell'array = %d",*(punt+1));
printf("ultimo elemento dell'array = %d",*(punt+n-1));
```

16

Osservazioni e consigli

- L'implementazione dell'esercitazione richiede un frequente uso di operazioni sui semafori.
- E' consigliabile implementare queste operazioni come due funzioni, chiamate ogni volta che si deve incrementare o decrementare un semaforo.
- Ovviamente non potete chiamare le due funzioni *signal* e *wait*, poiché c'è conflitto con le corrispondenti system call Unix (potete usare, ad esempio *V* e *P*, oppure *UP* e *DOWN*, o altri termini sufficientemente significativi)

17

Osservazioni e consigli

- Se un filosofo riceve un piatto diverso da quello ordinato, può stampare un messaggio di allarme: L'ACCADEMIA NON FUNZIONA! (non dovrebbe mai succedere...)
- **IMPORTANTISSIMO**: nell'implementazione finale (cioè quella che presentate per l'esame) non devono essere presenti delle sleep, oltre a quelle usate per simulare lo stato di **pensa** e di **mangia** dei filosofi, e per la preparazione dei piatti.
- Come fate a sapere se l'accademia funziona? Beh, se la fate girare per qualche decina di minuti, e non si verificano deadlock o starvation, probabilmente...

18

Parte opzionale
(ovviamente viene valutata per l'esame)

- Prevedete la possibilità di lanciare il sistema in questo modo:

  $> accademia N *max*
- dove *max* è il numero di secondi massimi che un filosofo può stare senza mangiare. Trascorsi *max* secondi, il filosofo emette un messaggio del tipo: HO FAME!!!
- dopo altri max secondi senza cibo, viene emesso un altro messaggio "HO FAME!!!", e così via...

19

Che cosa dovete fare:

- Dovete scrivere un programma principale *accademia*, un programma *filosofo* (di fatto, tutti i processi filosofo usano lo tesso codice, e i filosofi sono contraddistinti da un numero consecutivo 0, 1, 2 ... N-1), e un programma *stato*, usabile come un comando da terminale.
- Ovviamente potete far uso di qualsiasi system call, programma d'esempio, comandi shell visti a lezione, e di qualsiasi strumento software messo a disposizione dall'ambiente Unix. Potete ovviamente sfruttare gli esercizi su code, semafori e memoria condivisa già svolti.

20

Informazioni generali sull'esercitazione e sull'esame:

- Potete svolgere l'esercitazione in gruppo, di al massimo 4 persone (personalmente consiglierei gruppi di 2 o 3 persone)
- Scadenza per la consegna dell'esercitazione: giugno 2004
- Cosa si consegna: listati del codice stampati, commentati in maniera adeguata. Sulla prima pagina, indicate chiaramente il nome, la matricola e l'e-mail di tutti gli studenti del gruppo

21

Informazioni generali sull'esercitazione e sull'esame:

- Quando avete terminato l'esercitazione, fissate un appuntamento con me, via e-mail o passando nel mio ufficio, per quando mi farete vedere girare i programmi.
- Se avete sviluppato i programmi su Linux (o comunque non sulle macchine del laboratorio) e avete a disposizione un portatile, è possibile dimostrare il funzionamento dei programmi sul portatile.
- Se non avete un portatile, dovete portare il codice sulle macchine Solaris, e verificarne il corretto funzionamento (ricordatevi che ci sono leggere differenze tra diverse versioni di unix)

22

Informazioni generali sull'esercitazione e sull'esame:

- Se il codice sviluppato gira correttamente secondo le specifiche che abbiamo visto, allora potete consegnarmi gli stampati del codice.
- Da questo momento in poi, potete dare l'esame del corso.
- L'esame è orale, e può essere dato su appuntamento in qualsiasi periodo dell'anno (beh, non a ferragosto, Natale...)

23

Contenuti dell'esame :

- Parte di teoria (per chi non ha dato o passato gli scritti, o vuole migliorare il voto preso)
- Implementazione del kernel Unix
- Conoscenza del funzionamento delle principali system call Unix (in particolare, fork, wait, signal, kill)
- Conoscenza di base della shell, e capacita' di risolvere semplici problemi che usano i comandi della shell (se sapete risolvere gli esercizi dati a lezione, non avrete problemi).
- Conoscenza dei programmi sviluppati per l'esercitazione finale del corso (per cui, fate attenzione, se lavorate in gruppo, a non far fare tutto il lavoro ad uno solo)

24