Esercitazione SO/LAB

Sistemi Operativi

Antonino Staiano, Aniello Castiglione, Gianmaria Perillo
Email: antonino.staiano@uniparthenope.it

Esercizio 1

- Costruire l'albero dei processi ed indicare per ciascun processo il valore finale di glob

```
int glob=5;
int pid=0;
pid=fork();
glob--;
fork();
pid=1;
glob--;
if (!pid) {
   pid=fork();
   glob--;
}
printf("Valore di glob=%d\n",glob);
```


Esercizio 2

- Costruire l'albero dei processi ed indicare per ciascun processo il valore finale di glob

```
int glob=5;
int pid=0;
pid=fork();
glob--;
pid=fork();
glob--;
if (pid!=0) {
    pid=fork();
    glob--;
}
printf("Valore di glob=%d\n",glob);
```


Esercizio 3

```
int glob=5;
int pid=0;
int main() {
     for (i=1;i<=glob;i++) {
          pid=fork();
          if (pid==0)
                  glob=glob-1;
          glob=glob-1;
     }
printf("Valore di glob=%d\n",glob);
}
```

Esercizio 4 (incompleto)

- In una fabbrica, N operai preparano piastrelle da far cuocere in un forno, capace di cuocerne M contemporaneamente.
- All'uscita dal forno K operai visionano le piastrelle per decorarle secondo tale sequenza di passi:
  - se trova una piastrella difettata inizia a prenderne dal forno 2 alla volta
  - altrimenti ne prende 1 alla volta


```
semaforo contatore sem_inf=N;
semaforo contatore sem_dec=K;
semaforo contatore forno_vuoto=M;
semaforo contatore forno_pieno=0;
semaforo binario mutex=1;
int piastrelle=0;
```

OperaioInf

```
parbegin
repeat
  wait(sem_inf);
  wait(forno_vuoto);
  wait(mutex);
  piastrelle++;
  signal(mutex);
  cuociPiastrelle;
  signal(forno_pieno);
  signal(sem_inf);
forever
end
```

OperaioDec

```
parbegin
repeat
wait(sem_dec);
  wait(forno_pieno);
  wait(mutex);
  piastrelle--;
  if(piastre_difettate)
    piastrelle=piastrelle-2;
  signal(mutex);
  decoraPiastrelle();
  signal(forno_vuoto);
  signal(sem_dec);
forever
end
```


Esercizio 5

- Un gruppo di N thread condivide un array di M elementi (con M = N/2). Ciascun indice dell'array condiviso può essere occupato al più da un thread alla volta.
- Un coordinatore indica ai thread il momento in cui possono contendersi l'acceso ad una posizione dell'array. Appena il coordinatore li sblocca, tutti i thread tentano di accedere ad una posizione, occupandola attraverso il proprio TID.
- Per i rimanenti N/2 thread rimasti fuori inizierà un processo di voto. Ognuno di essi voterà una sola volta per uno solo dei thread disposti nell'array. Il TID del processo più votato (a parità di voto vince il TID maggiore) viene salvato dal coordinatore che reimposta tutto allo stato originale, ossia l'array viene liberato e invia un nuovo segnale di inizio contesa di accesso all'array stesso.
- Tutti i thread possono partecipare alla contesa ad esclusione del thread che nel ciclo precedente era risultato vincitore. Il processo termina quando tutti gli N thread sono stati almeno una volta vincitori del processo di voto.

Esercizio

- In una fabbrica, N operai preparano piastrelle da far cuocere in un forno, capace di cuocerne M contemporaneamente.
- All'uscita dal forno K operai visionano le piastrelle per decorarle secondo tale sequenza di passi:
  - se trova una piastrella difettata inizia a prenderne dal forno 2 alla volta
  - altrimenti ne prende 1 alla volta

Strutture dati e semafori

```
#define true 1
#define false 0
#define qualita_minima 1

sem_t sem_inf,sem_dec,sem_forno_vuoto,sem_forno_pieno;
pthread_mutex_t mutex;
int piastrelle = -1, qualita_mattonella = 0;
int aPiastrelle[1000];

int flagPiastrellaDifettata = false;

void *operaioInf();
void *operaioDec();

int main() {
```


main()

```
 srand(time(NULL));
    sem_init(&sem_inf,0,10);
    sem_init(&sem_dec,0,10);
    sem_init(&sem_forno_vuoto,0,10);
    sem_init(&sem_forno_pieno,0,0);
    pthread_mutex_init(&mutex,NULL);
    pthread_t thread1,thread2;

   // creiamo 10 thread operai che infornano
    pthread_create(&thread1,NULL,operaioInf,NULL);
   // creiamo 10 thread operai che decorano
    pthread_create(&thread2,NULL,operaioDec,NULL);

    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);
    return 0;
}
```


Operai che Infornano

```
void *operaioInf(){
    while(1){
        sem_wait(&sem_inf);
        sem_wait(&sem_forno_vuoto);
        pthread_mutex_lock(&mutex);
        piastrelle++;
        printf("Inforno Mattonella %d\n\n",piastrelle);
        qualita_mattonella = rand()%10;
        aPiastrelle[piastrelle] = qualita_mattonella;
        pthread_mutex_unlock(&mutex);
        sem_post(&sem_forno_pieno);
        //CuociPiastrella();
        sem_post(&sem_inf);
    }
    pthread_exit((void*)0);
}
```


Operai che decorano

```
void *operaioDec(){
    while(1){
        sem_wait(&sem_dec);
        sem_wait(&sem_forno_pieno);
        if (flagPiastrellaDifettata == true){
            printf("Prendo due mattonelle dal forno\n\n");
            sem_wait(&sem_forno_pieno);
            pthread_mutex_lock(&mutex);
            piastrelle -=2;
            sem_post(&sem_forno_vuoto);
            sem_post(&sem_forno_vuoto);
        }else if(aPiastrelle[piastrelle] < qualita_minima){
            pthread_mutex_lock(&mutex);
            printf("Prendo Mattonella dal forno\nTrovata prima
mattonella difettata\n\n");
            flagPiastrellaDifettata = true;
            piastrelle--;
            sem_post(&sem_forno_vuoto);
```

Operai che decorano (cont.)

```
}else{
            pthread_mutex_lock(&mutex);
            printf("Prendo Mattonella dal forno\n\n");
            piastrelle--;
            sem_post(&sem_forno_vuoto);
        }
        pthread_mutex_unlock(&mutex);
        //DecoraPiastrelle();
        sem_post(&sem_dec);
    }
    pthread_exit((void*)0);
}
```


Produttori-Consumatori

Globali

```
#include …

#define   NFORNO         10
#define   MAXNOPERAI     100

int  npiastrelle, nInformatori, nDecoratori ;
/* sola lettura */

struct {   /* dati condivisi tra prods. e cons. */
  int     forno[NFORNO];
  int     nput;
  int     nputval;
  int     nget;
  int     nval;
  sem_t mutex, nempty, nstored;
} shared;

void *Informatori(void *), *Consumatori(void *);
```


main …

```
int
main(int argc, char **argv)
{
 int       i, Infcount[MAXNOPERAI], Decscount[MAXNOPERAI];
 pthread_t  tid_Inf[MAXNTHREADS], tid_Dec[MAXNTHREADS];

 npiastrelle = atoi(argv[1]);
 nInformatori = MIN(atoi(argv[2]), MAXNOPERAI);
 nDecoratori = MIN(atoi(argv[3]), MAXNOPERAI);
     /* inizializza tre semafori */
 sem_init(&shared.mutex, 0, 1);
 sem_init(&shared.nempty, 0, NFORNO);
 sem_init(&shared.nstored, 0, 0);
     /* crea tutti gli operai */
 for (i = 0; i < nInformatori; i++) {
     Infcount[i] = 0;
     pthread_create(&tid_Inf[i], NULL, Informatori, &Infcount[i]);
 }
 // stesso per i decoratori
```

… main

```
/* aspetta tutti i gli operai */
 for (i = 0; i < nInfornatori; i++) {
     pthread_join(tid_Inf[i], NULL);
     printf("Infcount[%d] = %d\n", i, Infcount[i]);
 }
 // stesso per i decoratori

 sem_destroy(&shared.mutex);
 sem_destroy(&shared.nempty);
 sem_destroy(&shared.nstored);
 exit(0);
}
```


Infornatori

```
void *Infornatori(void *arg)
{
  for ( ; ; ) {
        sem_wait(&shared.nempty);
  /* aspetta almeno una locazione libera */
        sem_wait(&shared.mutex);

        if (shared.nput >= npiastrelle) {
               sem_post(&shared.nstored); /* consente ai decoratori di finire */
               sem_post(&shared.nempty);
               sem_post(&shared.mutex);
               return(NULL);          /* tutto infornato */
        }

        shared.forno[shared.nput % NFORNO] = shared.nputval;
        shared.nput++;
        shared.nputval++;

        sem_post(&shared.mutex);
        sem_post(&shared.nstored);
/* un altro elemento memorizzato */
        *((int *) arg) += 1;
  }
}
```


Decoratori

```
void *Decoratori(void *arg)
{
  int       i;
  for ( ; ; ) {
   sem_wait(&shared.nstored);
/* attende almeno un elemento memorizzato */
   sem_wait(&shared.mutex);
  if (shared.nget >= npiastrelle){
      sem_post(&shared.nstored);
      sem_post(&shared.mutex);
      return (NULL); /* tutto decorato */
   }
   i = shared.nget % NFORNO;
  if (shared.forno[i] != shared.ngetval)
            printf('Errore\n');
  shared.nget++;
   shared.ngetval++;
   sem_post(&shared.mutex);
   sem_post(&shared.nempty);
*((int *) arg) += 1;
}
}
```