Concurrencia y Sincronización

Sistemas Operativos

2017

Ejercicio 1 Implementar el siguiente programa C, crear un archivo number.txt conteniendo un valor (string) numérico y disparar varias instancias concurrentes de él.

```
#include <stdio.h>

void main(void) {
  int i;
  FILE * f = fopen("number.txt","r+");

  for (i=0; i < 10000; i++) {
    int n;
    fscanf(f,"%d",&n);
    rewind(f);

    fprintf(f,"%d",++n);
    rewind(f);

  }
  fclose(f);
}
```

- Analizar el valor final resultante en el archivo number.txt.
- Corregir el programa garantizando exlusión mutua (ver man 2 flock).

Ejercicio 2 Implementar el problema del productor consumidor en java, usando threads.

Ejercicio 3 Implementar las operaciones de un `Mutex`: `void lock(boolean * var)` y `void unlock(boolean * var)` usando las siguientes funciones asumiendo que son atómicas:

a)
```
int TestAndSet(int * v){
    int tmp = *v;
    if (*v==0) *v = 1;
    return tmp;
}
```

b)
```
int Swap(int * v1, int * v2){
    int tmp = *v1;
    *v1 = *v2; *v2 = tmp;
    return tmp;
}
```

Ejercicio 4 Compilar y ejecutar el programa `pthreads_example.c` que se encuentra la sección de programas de ejemplos. Eliminar la condición de carrera usando la implementación de locks el ejercicio anterior usando la versión de `swap()` implementada en el programa `xchg.c` disponible en el repositorio de Moodle.

Ejercicio 5 Demostrar que la solución de Peterson[1] conserva las siguientes propiedades:

a) exclusión mutua

b) progreso

c) espera limitada

Ejercicio 6 Implementar las funciones `int sem_init(semaphore * s, int init_value)`, `int sem_wait(semaphore * s)`, `int sem_signal(semaphore * s)` y `int sem_close(semaphore * s)` usando las siguientes llamadas al sistema de los sistemas POSIX compatibles:

- `int semctl(int semid, int semnum, int cmd, ...)`
- `int semget(key_t key, int nsems, int semflg)`
- `int semop(int semid,struct sembuf*sops, unsigned nsops)`

[1] Ingreso a la región crítica en forma alternada por dos procesos.

Nota: El sistema operativo garantiza que `semop` se ejecuta atómicamente.

Ejercicio 7 Compilar y ejecutar el programa `pthreads_example.c` que se encuentra la sección de programas de ejemplos. Eliminar la condición de carrera usando la implementación de semáforos realizada en el ejercicio anterior.

Ejercicio 8 Implementar el problema conocido como productor-consumidor

a) Con pthreads y la implementación de semáforos desarrollada.

b) Con procesos y un espacio de memoria compartida (ver `man shmget`, `man shmat`)

Ejercicio 9 Dar al menos tres soluciones posibles (sin deadlock) al problema de los filósofos comensales.