Sección Crítica y Exclusión Mutua

Concurrencia y Paralelismo

Juan Quintela
quintela@udc.es

Javier París
javier.paris@udc.es

Sección Crítica

- Parte de código que accede a un recurso compartido con otro proceso/thread.
- Hay que regular el acceso de los procesos a la sección crítica para que no se produzcan inconsistencias en la información compartida.

Sección Crítica : Ejemplo

Leer un valor compartido e incrementarlo

```
int i=0; // Variable global, compartida entre threads

void *incrementar(void *arg) {
  int v; // Variable local, una por thread
  v=i;
  v++;
  i=v;
}

void lanzar_thread() {
  pthread_t thr;
  pthread_create(&thr, NULL, incrementar, NULL);
}
```

Sección Crítica: Ejemplo

Sección Crítica

```
void *incrementar(void *arg) {
  int v;
  v=i;  //
  v++;  // Seccion critica
  i=v;  //
}
```

La sección crítica es la parte de código donde estamos tocando recursos compartidos, en este caso la variable i.

Sección Crítica: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

Los dos threads comienzan a ejecutarse entrando en la función incrementar.

Sección Crítica: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

Los dos threads copian el valor de la variable global i(=0). Las variables v son locales, por lo que cada thread tiene su propia variable en su stack y no se comparten.

Sección Crítica: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  v = i;
  v++;
  i=v;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

El thread 1 incrementa el valor de su variable local v (=1). El thread 2 ha salido del procesador y sigue en el mismo punto

Sección Crítica: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  v = i;
  v++;
  i=v;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

El thread 1 sobreescribe la variable global con el valor de su copia de v (i pasa a valer 1).

Sección Crítica: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  v = i;
  v++;
  i=v;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

El thread 1 termina. El thread 2 vuelve a ejecutarse e incrementa el valor de su variable local v (pasa a valer 1)

Sección Crítica: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  v = i;
  v++;
  i=v;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  v=i;
  v++;
  i=v;
}
```

El thread 2 sobreescribe la variable i con el valor de v (i pasa a valer 1) y termina

Sección Crítica: Ejemplo

- Hemos ejecutado la función de incremento 2 veces, pero sólo se ha incrementado una vez debido a las interacciones entre los threads.
- Este problema se llama actualización perdida (lost update).

Exclusión Mutua

- Consiste en garantizar que dos procesos no pueden estar en su sección crítica simultáneamente.
- Hay que gestionar los accesos a las secciones críticas del programa.

Exclusión Mútua: Ejemplo

Vamos a añadir al ejemplo anterior un mecanismo que solo permita un thread a la vez en la sección crítica:

Ejemplo lock

```
void *incrementar(void *arg) {
int v;
lock;
v = i;
v++;
i=v;
unlock;
}
```

El primer thread que llegue a lock bloquea el acceso a los demás. Cuando ese proceso haga unlock se permite el paso al siguiente. Este mecanismo de bloqueo se denomina Mutex.

Exclusión Mútua: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

Exclusión Mútua: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

El primer thread llega al mutex. Como no hay otro thread que lo tenga bloqueado, lo bloquea y continúa.

Exclusión Mútua: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

El segundo thread llega al mutex. Como está bloqueado queda esperando hasta que se libere.

Exclusión Mútua: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

El primer thread ejecuta toda la sección crítica. Lee el valor de la variable i (0), lo incrementa en la variable v(1), y lo vuelve a copiar a i (1).

Exclusión Mútua: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

El primer thread libera el mutex. El segundo thread puede continuar y entrar en la sección crítica.

Exclusión Mútua: Ejemplo

Thread 1

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

Thread 2

```
void *incrementar(void *arg){
  int v;
  lock;
  v=i;
  v++;
  i=v;
  unlock;
}
```

El segundo thread lee el valor de i(1) en v, lo incrementa(2) y lo escribe en i(2). No se pierde ninguna de las dos actualizaciones.

Instrucciones Atómicas

- Hay procesadores que proporcionan instrucciones que se ejecutan de forma atómica.
- Estas instrucciones se usan para implementar bloqueos.

Instrucciones Atómicas: Test and set

- `test_and_set` es una instrucción atómica que permite cambiar el valor de una posición de memoria y devolver su valor antiguo en un único paso. Normalmente `test_and_set` escribe el valor 1.
- Se puede implementar un mutex usando `test_and_set`:

Implementación Lock y Unlock

```
void lock(int *v) {
  while(test_and_set(v)==1);
}

void unlock(int *v) {
  *v=0;
}
```

Mutex Pthread

La librería pthread implementa mutex:

Pthread Mutex

```
pthread_mutex_t *mutex;

int pthread_mutex_init(pthread_mutex_t *,
pthread_mutex_attr *);
int pthread_mutex_destroy(pthread_mutex_t *);

int pthread_mutex_lock(pthread_mutex_t *);
int pthread_mutex_trylock(pthread_mutex_t *);
int pthread_mutex_unlock(pthread_mutex_t *);
```

Mutex Pthread

Ejemplo

```
struct thr_args {
  pthread_mutex_t mutex;
};

int main() {
  struct thr_args arg;
  pthread_t thr1,thr2;
  pthread_mutex_init(&args.mutex, NULL);

  pthread_create(&thr1, NULL, thr_fun, &arg);
  pthread_create(&thr2, NULL, thr_fun, &arg);

  pthread_join(thr1, NULL);
  pthread_join(thr2, NULL);
  pthread_mutex_destroy(&arg.mutex);
}
```

Mutex Pthread

Ejemplo

```
int i=0;

void *thr_fun(void *p) {
  struct thr_args *arg=p;
  int v;

  pthread_mutex_lock(&arg->mutex);
  v=i;
  v++;
  i=v;
  pthread_mutex_unlock(&arg->mutex);
}
```

Semáforos

- Los semáforos son otro mecanismo de control de acceso a la sección crítica.
- Un semáforo tiene un cierto valor numérico que se le asigna al crearlo.
- Intentar bloquearlo decrementa su valor, liberarlo lo incrementa.
- Cuando el valor del semáforo es 0 el proceso bloquea.
- P(bloquear) y V(liberar) son atómicas.

Semáforos

P y V

```
V(semaphore S) {
  S=S+1;
}

P(semaphore S) {
  while(1) {
    if(S > 0) { S=S-1; break; }
  }
}
```

Un Mutex es un caso especial de semáforo con valor 1.

Semáforos Posix

```
sem_t *sem;
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);

int sem_wait(sem_t *sem); // P
int sem_trywait(sem_t *sem);
int sem_timedwait(sem_t *sem,
                  const struct timespec *abs_timeoout);
int sem_post(sem_t *sem); // V
```

La estructura sem tiene que estar en memoria compartida entre los procesos

Semáforos Posix: Ejemplo

Ejemplo

```
int main() {
  sem_t *sem; int *shr_i;

  shr_i=mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
             MAP_SHARED | MAP_ANONYMOUS, 0, 0);
  sem=mmap(NULL, sizeof(sem_t), PROT_READ | PROT_WRITE,
           MAP_SHARED | MAP_ANONYMOUS, 0, 0);
  sem_init(sem, 1, 1);

  if(fork()==0) { proc_fun(sem,shr_i); exit(0); }
  if(fork()==0) { proc_fun(sem,shr_i); exit(0); }

  sem_destroy(sem);
  munmap(sem);
}
```

Semáforos Posix: Ejemplo

Ejemplo

```
void proc_fun(sem_t *sem, int *shr_i) {
  int v;

  sem_wait(sem);
  v=*shr_i;
  v++;
  *shr_i=v;
  sem_post(sem);
}
```