Sección Crítica y Exclusión Mutua

Sección Crítica

Parte de código que accede a un recurso compartido con otro proceso o thread. Hay que regular el acceso de ellos a la sección crítica para que los valores de dicho recurso no sean inconsistentes en el futuro, dado que se trata de una información compartida. Ej -> Al leer un valor compartido y modificarlo:

```
int i = 0; // Variable global, compartida entre threads
void* incrementar(void *arg) {
   int v; // Variable local, cada thread creará la suya
   v = i;     v++;      i = v; /*Sección crítica (modifica valor de i)*/ }
void ejecutar_thread() {
   pthread_t hilo;
   pthread_create(&hilo, NULL, incrementar, NULL); }
```

La sección critica es la parte de código donde se manipulan los recursos compartidos, en este caso es la variable i. Supongamos que nuestro programa crea 2 threads, su ejecución es la siguiente:

Tiempo	Thread1	Thread2
0	Se crean ambos threads, declarando su propia variable v en su propio stack	
1	Cada thread copia el valor de la variable global i (que vale 0) en v.	
2	Ejecuta v++;. Ahora su v = 1	Salio de CPU, se cumple v = i = 0
3	Ejecuta i = v; sobreescribiendo el valor de la var. Global i con el valor de su variable local v = 1, por lo que i ahora vale 1.	Sigue fuera de CPU
4	Termina su ejecución	Vuelve a CPU y ejecuta v++; por lo que v = 1.
5		Ejecura i = v; por lo que se sobreescribe el valor de la variable global i = 0, por la var. Local v = 1 -> i = 1.

Podemos ver que se han hecho 2 incrementos de la misma variable, pero su valor real solo ha sido incrementado 1 vez a causa de como funciona la memoria compartida entre threads. Este problema se denomina **actualización perdida**.

Exclusión Mutua (Mutex)

Consiste en garantizar que 2 procesos no pueden estar en su sección crítica simultáneamente. Hay que gestionar los accesos a las secciones críticas del programa

```
void* incrementar(void *arg) {
 int v; // Variable local, cada thread creará la suya
 lock; // Thread obtiene el control
 v = i;
 v++;
 i = v; // Sección crítica (modifica valor de i)
 unlock; // Thread libera el control
}
```

Con este mecanismo, el primer thread que llegue a ejecutar la instrucción lock pasa a bloquear al resto de hilos que traten de acceder a la sección crítica, haciendolos esperar a que dicho thread desbloquee la sección crítica llamando a unlock, haciendo que el resto de hilos sigan ejecutándose

Instrucciones Atómicas

Instrucciones del procesador que se ejecutan completamente y sin interrupciones. Sirven para implementar bloqueos dentro del código, lo cual es útil a la hora de controlar las secciones críticas

Instrucciones Atómicas – Test and set (…)

Librería Pthread

```
pthread_mutex_t *mutex; // Puntero a mutex (para poder compartirlo)
// Inicializar un mutex con unos atributos dados (NULL por defecto)
```

```c
int pthread_mutex_init(pthread_mutex_t* mutex, pthread_mutex_attr* atrib);
// Destruir un mutex
int pthread_mutex_destroy(pthread_mutex_t* mutex);
// Obtener el control sobre un mutex
int pthread_mutex_lock(pthread_mutex_t* mutex);
// Intentar obtener el control sobre un mutex que podría estar ocupado
int pthread_mutex_trylock(pthread_mutex_t* mutex);
// Liberar el control sobre un mutex
int pthread_mutex_unlock(pthread_mutex_t* mutex);
```