Sistemas Operativos
Mutex

2019

1. Mutex.

Exclusión Mutua: (Mutual Exclusion) es un mecanismo en el que se intenta dar solución al problema de la sección crítica en un programa. Recuerda que éste consiste en evitar que dos o más procesos o hilos ingresen en una sección crítica al mismo tiempo. En un sistema de un sólo procesador, la exclusión mutua se puede solucionar mediante la deshabilitación de las interrupciones y cuando se trata de sistemas multicore, se debe implementar mecanismos de software como semáforos, mutex, entre otros.

Mutex: son elementos del POSIX para el manejo de secciones críticas y evitar que múltiples hilos ingresen en las mismas. Las funciones más importantes para su uso son:

```
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                       const pthread_mutexattr_t *restrict attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);

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

Las primeras dos funciones servirán para crear y destruir las variables para control de la sección crítica. Las siguientes te permitirán controlar la entrada y la salida de la sección respectivamente. Es necesario tener en cuenta si las funciones requieren punteros o no.

El siguiente ejemplo muestra la forma en que se pueden lanzar múltiples hilos, declarando los descriptores para cada uno como arreglos. Lo anterior permite que se inicialicen las variables, se lancen los hilos y se sincronicen mediante ciclos for. En el ejemplo se define una función llamada *proceso* que recibe un puntero a una estructura, para mostrar un *dato* un número de veces p.

Listing 1: Ejemplo de múltiples hilos

```
1  /* Ejemplo de manejo de hilos
2  *  compilar con threads: -lpthread
3  */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <pthread.h>
7
8  #define MAX_PROCESOS 1
9  #define NUM_HILOS 8
10
11 struct datos_tipo
12 {
13         int dato;
14         int p;
15 }
16
17 * proceso(void *datos)
18 {
19         struct datos_tipo *datos_proceso;
20         datos_proceso = (struct datos_tipo *) datos;
21         int a, i, j, p;
22
23         a = datos_proceso -> dato;
24         p = datos_proceso -> p;
25 //--->
26         for(i=0; i <= p; i++)
27         {
28                 printf(" %i ",a);
29         }
30         fflush(stdout);
31         sleep(1);
32         for(i=0; i<=p; i++)
33         {
34                 printf("- ");
35         }
36         fflush(stdout);
37 //--->
38 }
39
40 int main()
41 {
42 int error, i;
43 char *valor_devuelto;
44
45     /* Variables para hilos*/
46     struct datos_tipo hilo_datos[NUM_HILOS];
47     pthread_t idhilo[NUM_HILOS];
48
49     for(i=0; i<NUM_HILOS; i++){
50         hilo_datos[i].dato = i;
```

```
51          hilo_datos[i].p = i+1;
52      }
53
54          /*se lanzan los hilos*/
55      for(i=0; i<NUM_HILOS; i++){
56          error=pthread_create(&idhilo[i], NULL, (void *)proceso, (void *)(&hilo_datos[i]));
57          if (error != 0)
58          {
59                  perror ("No puedo crear hilo");
60                  exit (-1);
61          }
62      }
63      /*Esperar a que terminen*/
64      for(i=0; i<NUM_HILOS; i++){
65          pthread_join(idhilo[i], (void **)&valor_devuelto);
66      }
67  return 0;
68  }
```

Para el taller se pide:

1. Transcribe el código, elabora un archivo Makefile, compila y ejecuta. Comenta lo que sucede. Cambia el número de hilos ejecutados y observa lo que sucede.

2. Tomando parte del proceso que muestra en pantalla como sección crítica, señalado por los comentarios $// - -- >$, añade las funciones para el manejo de secciones críticas con mutex. Comenta brevemente lo que sucede.

3. Evita copiar y pegar desde el pdf, algunos caracteres adicionales podrían insertarse generando errores durante la compilación.