Universidad Nacional de Costa Rica

Escuela de informática

Proyecto de Sistemas Operativos: Puente

Estudiante: José David Flores Rodríguez

Profesor: Eddy Ramírez

Fecha May 17, 2021

1 Introducción

Para este proyecto se nos pidió realizar una simulación de un puente el cual tiene solo una vía, o sea solo un sentido. Esta vía o sentido puede ser de: este a oeste o de oeste a este, dependiendo de ciertos factores como lo son: semáforos vehiculares, policías de transito o el que "llega de primero". Dichas entidades serán las encargadas de controlar el paso por el puente para evitar choques entre los vehículos y aprovechar al máximo este recurso (el puente).

Para realizar esta simulación debíamos utilizar el lenguaje de programación C y el sistema operativo Linux. Además de usar los hilos de la librería POSIX para la implementación de las entidades (vehículos, semáforos, oficiales de tránsito, ambulancias...).

En este texto se explicarán las librerías de C utilizadas para la elaboración del proyecto, la descripción de la implementación del problema propuesto, descripción de la experiencia durante la realización del mismo y aprendizajes.

2 Marco Teórico

Sin duda alguna en un programa los hilos tienen gran importancia ya que permiten la ejecución de varias tareas de manera "simultanea". Sin embargo, aunque se perciba que las tareas se realizan al mismo tiempo, esto no es así, cada núcleo del procesador puede realizar una tarea a la vez, lo que sucede es que este trabaja a una velocidad tan rápida que nuestra percepción es que se ejecutan al mismo tiempo, más aún si el procesador es multi núcleo lo que si permitiría la ejecución de varios procesos de forma paralela (al mismo tiempo). En el lenguaje de programación C existe una librería muy popular para la administración de hilos la cual esta basada en POSIX. POSIX es un una interfaz "standart" para todos los sistemas operativos la cual permite que el programa sea portable y pueda ser ejecutado en diferentes sistemas operativos basados en POSIX, según [2].

La librería de la cual se habla en el párrafo anterior se llama Pthread y esta tiene las siguientes funciones que permiten la administración de hilos:

- **pthread_t:** Usado para identificar un hilo.
- **pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg):** Permite crear un nuevo hilo. Los atributos que se desean utilizar para el hilo se especifican en el parámetro "attr", el parámetro "start_routine" es la función que se va a ejecutar y el parámetro "arg" recibe los argumentos que se quieran pasar a la función que se va a ejecutar.
- **pthread_exit(void *value_ptr):** Esta función termina la ejecución del hilo que la llama, permite que el atributo referenciado en "value_ptr" sea accedido cuando se utilice la función "pthread_join".
- **pthread_join(pthread_t thread, void **value_ptr):** Permite suspender la ejecución del hilo que lo llama hasta que el hilo referenciado en el parámetro "thread" haya terminado su ejecución. El valor de "value_ptr" contiene el valor que pasó desde la función "pthread_exit".
- **int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr):** Función que permite que se inicie un MUTEX con los atributos especificados en el parámetro "atr".
- **int pthread_mutex_lock(pthread_mutex_t *mutex):** El MUTEX se bloquea cuando esta función es llamada.
- **int pthread_mutex_trylock(pthread_mutex_t *mutex):** Esta función intentará bloquear el MUTEX y retornará el entero 0 si la operación fue exitosa.
- **int pthread_mutex_unlock(pthread_mutex_t *mutex):** El MUTEX se desbloquea cuando se llama pthread_mutex_unlock.

Las funciones anteriores son solo algunas de disponibles en la librería Pthread, al igual que estas, la totalidad de funciones se pueden consultar en la documentación de "pthread.h" [1].

3 Descripción de la solución

Al estar ejecutándose diferentes procesos e hilos al mismo tiempo puede que uno de estos hilos intente modificar cierta variable de las estructuras de datos donde se guarda la información del puente, oficial de tránsito o semáforo, es por esta razón que se necesita un control sobre estas estructuras para que no ocurran deadlocks, osea que dos hilos intenten modificar una misma variable al "mismo tiempo". Aquí es donde se utilizan los MUTEX que ofrece la librería Pthread de Linux, que permite controlar que un hilo acceda o modifique el contenido de una variable a la vez, si alguien lo esta haciendo el otro proceso tendrá que esperar hasta que este MUTEX se encuentre desbloqueado.

Para la solución al problema planteado los MUTEX fueron indispensables, a continuación se explicará cómo se implementaron los 3 diferentes mecanismos para controlar el tránsito en el puente.

3.1 Semáforos

Para implementar el control del tránsito mediante semáforos se utilizaron las siguientes estructuras:

```
1   struct semaforo
2   {
3     int tiempo_verde_este;
4     int tiempo_verde_oeste;
5     int este; //este = 1 si luz verde de este esta encendido,
    este = 0 si no
6     int oeste; //oeste = 1 si luz verde de oeste esta encendido,
    oeste = 0 si no
7     int uso_timer; //Identifica si hay algun semaforo utilizando
    el timer para pasar de verde a rojo
8     pthread_mutex_t mutex;
9     struct puente *puente;
10  };
11
12  struct puente
13  {
14    int sentido;        //sentido = 1 si los automoviles van al
    este y sentido = 0 si van al oeste
15    int sentido_actual; //sentido = 1 si los automoviles van al
    este y sentido = 0 si van al oeste
16    int longitud_puente;
17    pthread_mutex_t *puente_lock; //Arreglo de MUTEX el cual
    representa el puente
18  };
19
20  struct info_autos
21  {
22    int velocidad_auto_este;
23    int velocidad_auto_oeste;
24    int media_este;
25    int media_oeste;
26    struct semaforo *semaforos;
27    struct puente *puente;
28  };
```

```

```

La estructura que se pasa como argumento en la mayoría de los hilos es "info_autos" la cual contiene toda la información que los automóviles necesitan para decidir si pasar o no.

Las funciones utilizadas en la implementación fueron las siguientes:

```
int iniciar_semaforo(); //Funcion principal que se encarga de iniciar las variables
int main_runner(struct info_autos *info); //Esta funcion se encarga de crear y ejecutar threads que iniciaran la creacion de los semaforos y automoviles
void *create_semaforos(void *arg);        //Esta funcion encapsula la creacion del semaforo ubicado al este y oeste del puente
void *cambiar_semaforo_este(void *arg);   //Enciende el semaforo del oeste
void *cambiar_semaforo_oeste(void *arg);  //Enciende el semaforo del oeste
void *create_automoviles(void *arg);      //Funcion que crea los automoviles
void *automovil_este(void *arg);          //Crea los automoviles por que entran por el este
void *automovil_oeste(void *arg);         //Crea los automoviles que entran por el oeste
void *crear_autos_este(void *arg);      //Automovil que entra por el este
void *crear_autos_oeste(void *arg);       //Automovil que entra por el oeste
double ejecutar_integral(struct info_autos *info, char eleccion_media[]); //Funcion que ejecuta la integral exponencial encargada de decidir la velocidad con que se crean los automoviles;
int revisar_puente_en_uso(struct puente *puente);
            //funcion que recorre el arreglo de mutex revisando si hay un automovil usado el puente o no

```

3.1.1 Semáforos

Los semáforos están siempre en un loop infinito verificando si el semáforo contrario cerro el paso y encendió la luz roja, cuando el semáforo contrario apagó la luz verde, el semáforo actual enciende la luz verde e inicia el timer mediante la función sleep(), este contador lleva el tiempo de luz verde del semáforo.

3.1.2 Puente

El puente mediante las variables sentido y sentido_actual permiten a los automóviles saber si tienen derecho de pasar o no. El "sentido" toma en cuenta el estado del semáforo, sin embargo, "sentido_actual" cambia hasta que el puente este desocupado de automóviles que ya estaban en el puente cuando el semáforo cambio de estado.

3.1.3 Automóviles

Estos toman la decisión de pasar si el "sentido" y "sentido_actual" son iguales, sino estos no pasaran por el puente.

Existe una función muy importante que se implementa tanto en los semáforos como en todos los otros mecanismos de control del tránsito la cual es revisar_puente_en_uso(). Esta función recorre el arreglo de MUTEX que representan el puente de izquierda a derecha y derecha izquierda al mismo tiempo lo cual verifica si el puente esta "vacío" o no.

3.2 Carnage

En el modo carnage, el programa permite que el automóvil que llega de primero al puente pueda pasar, siempre y cuando no exista otro automóvil en el sentido contrario. Si hay un carro circulando en el mismo sentido del nuevo automóvil, este nuevo automóvil podrá entrar al puente detrás de los autos que ya están usando este puente.

Este modo de administración del puente utiliza las siguientes estructuras:

```
1   struct puente_carnage
2   {
3     int sentido;        //sentido=1 si es este o sentido=0 si es
    oeste
4     int sentido_actual; //sentido=1 si es este o sentido=0 si es
    oeste
5     int longitud_puente;
6     pthread_mutex_t *puente_lock; //array de MUTEX que representa
     el puente
7   };
8
9   struct info_autos_carnage
10  {
11    int velocidad_auto_este;
12    int velocidad_auto_oeste;
13    int media_este;
14    int media_oeste;
15    struct puente_carnage *puente;
16  };
17
```

Las estructuras para almacenar los datos en este programa son muy similares a la anterior, la diferencia es que aquí no existe la estructura de semáforo ya que los que controlan el paso por el puente son los mismos automóviles.

A continuación se muestran las funciones utilizadas por este programa para la ejecución del mismo:

```
1   int iniciar_carnage(); //Funcion encargada de iniciar lar
    variables
2   int main_runner_carnage(struct info_autos_carnage *info); //
    Esta funcion se encarga de crear y ejecutar threads que
    iniciaran la creacion de los  automoviles
3   void *create_automoviles_carnage(void *arg); //Funcion que crea
     los automoviles
```

```
4     void *automovil_este_carnage(void *arg);      //Automovil que
      pasa por el este
5     void *automovil_oeste_carnage(void *arg);     //Automovil que
      pasa por el oeste
6     void *crear_autos_este_carnage(void *arg);    //Crea los
      automoviles por que entran por el este
7     void *crear_autos_oeste_carnage(void *arg);   //Crea los
      automoviles que entran por el oeste
8     double ejecutar_integral_carnage(struct info_autos_carnage *
      info, char eleccion_media[]); //Funcion que ejecuta la integral
       exponencial encargada de decidir la velocidad con que se crean
       los automoviles;
9     int revisar_sentido_carnage(struct puente_carnage *puente); //
      funcion que recorre el arreglo de mutex revisando si hay un
      automovil usando el puente o no
10
```

3.3 Oficiales de tránsito

Para la implementación de los oficiales de tránsito se utilizaron las siguientes estructuras para guardar los datos:

```
1     struct oficial
2     {
3       int k_este; //Cantidad de vehiculos que el oficial permite
      que pasen por el este
4       int k_oeste; //Cantidad de vehiculos que el oficial permite
      que pasen por el este
5       int este;    //este=1 si los automoviles van a pasar por el
      este
6       int oeste;   //oeste=1 si los automoviles van a pasar por el
      oeste
7       int contador_este; //Contador que lleva el control de los
      vehiculos que pasan por el este
8       int contador_oeste; //Contador que lleva el control de los
      vehiculos que pasan por el este
9       pthread_mutex_t lock_oficial;
10    };
11
12    struct puente_oficial
13    {
14      int sentido; //sentido=1 si es este o sentido=0 si es oeste
15      int sentido_actual;//sentido=1 si es este o sentido=0 si es
      oeste
16      int longitud_puente;
17      pthread_mutex_t puente_check;
18      pthread_mutex_t *puente_lock;
19    };
20    struct info_autos_oficial
21    {
22      int creando_automoviles_este;  //Esta variable es igual a 1
      si se estan creando automoviles o igual a 0 si no.
23      int creando_automoviles_oeste; //Esta variable es igual a 1
      si se estan creando automoviles o igual a 0 si no.
24      int velocidad_auto_este;
25      int velocidad_auto_oeste;
```

```
26      int media_este;
27      int media_oeste;
28      struct oficial *oficial;
29      struct puente_oficial *puente;
30    };
31
```

Los oficiales de tránsito mediante las variables contadoras permiten llevar el control de la cantidad de automóviles que pasan en cierto sentido. Cuando pasan la k cantidad de automóviles de un sentido, empiezan a pasar los automóviles en el sentido contrario.

Las funciones utilizadas son las siguientes:

```
1   void iniciar_oficial();
2   int main_runner_oficial(struct info_autos_oficial *info); //
    Esta funcion se encarga de crear y ejecutar threads que
    iniciaran la creacion de los semaforos y automoviles
3   void *create_oficial(void *arg);                          //
    Esta funcion encapsula la creacion del semaforo ubicado al este
     y oeste del puente
4   void *oficial_este(void *arg);                            //
    Enciende el semaforo del oeste
5   void *oficial_oeste(void *arg);                           //
    Enciende el semaforo del oeste
6   void *create_automoviles_oficial(void *arg);              //
    Funcion que crea los automoviles
7   void *automovil_este_oficial(void *arg);                  //
    Crea los automoviles por que entran por el este
8   void *automovil_oeste_oficial(void *arg);                 //
    Crea los automoviles que entran por el oeste
9   void *crear_autos_este_oficial(void *arg);
10  void *crear_autos_oeste_oficial(void *arg);
11  double ejecutar_integral_oficial(struct info_autos_oficial *
    info, char eleccion_media[]); //Funcion que ejecuta la integral
     exponencial encargada de decidir la velocidad con que se crean
     los automoviles;
12  int revisar_puente_en_uso_oficial(struct info_autos_oficial *
    info); //Revisa si el puente esta siendo usado por algun
    automovil
13
```

Al igual que en el semáforo una función bastante importante en esta implementación es la de revisar_puente_en_uso_oficial la cual permite verificar si existe algún automóvil usando el puente.

4 Conclusión

Para la realización de este proyecto el utilizar hilos fue de suma importancia ya que permitió que diferentes tareas se realizaran paralelamente (Tomando en cuenta el hardware del equipo). Los hilos permitieron que mientras se creaban automóviles de manera constante, las entidades de los semáforos o los oficiales de tránsito pudieran seguir operando sin que estos tuvieran que esperar a que la creación de automóviles terminara.

Es por esta razón que cuando nosotros utilizamos un software en nuestra computadora el usuario puede interactuar con la interfaz grafica mientras en el "fondo" el programa realiza otras tareas, o también de la misma manera que un servidor esta siempre a la espera de peticiones de diferentes usuarios sin dejar al usuario que se esta atendiendo sin la información requerida.

Tomando lo anterior señalado se entiende la importancia de los hilos en la programación y de igual manera se entiende de la excelente tarea que realiza el sistema operativo sincronizando los procesos y el scheduling de los mismos los cuales permiten un correcto funcionamiento del sistema en general.

5 Descripción del la experiencia

La experiencia fue bastante interesante ya que hubieron varios tipos de situaciones a lo largo de la implementación de la solución al problema.Se tuvo que entender y obtener bastante conocimiento del funcionamiento de los hilos en conjunto con los MUTEX para la implementación de la misma.

El proyecto se inicio con la implementación del mecanismo de los semáforos, esta se dio de manera fluida sin encontrar mayor complicación, se pudieron iniciar los semáforos cambiando el tiempo que estos estaban en verde y de la misma manera los automóviles empezaron a pasar por el puente sin provocar choques y aprovechando al máximo este recurso.

Lo mismo ocurrió con la implementación del método carnage, donde se tenía que tener en cuenta que el automóvil que llegara al puente debía verificar si existía algún automóvil en el sentido contrario, en ese caso debía esperar que el puente estuviera libre. Este mecanismo tampoco tuvo una gran complicación fuera de lo normal.

Donde si existió una mayor complicación fue en la creación del mecanismo del oficial de tránsito y en la implementación de las ambulancias las cuales tenían prioridad sobre cualquier vehículo esperando en el puente. Este ultimo no fue implementado en su totalidad en ninguno de los mecanismos.

Lo interesante es que la implementación del método del oficial de tránsito no era muy diferente que el método del semáforo, si embargo, con el oficial de tránsito existieron grandes desafíos al controlar que los automóviles ingresaran de manera ordenada al puente cuando el oficial de tránsito decidiera cambiar de sentido. Cuando el oficial cambiaba el sentido en que los automóviles debían ingresar al puente, los vehículos del nuevo sentido empezaban a ingresar al puente sin esperar a que los otros terminaran, lo que produciría un choque entre automóviles. Dichosamente este requerimiento si fue implementado con éxito, lo que hizo falta en los oficiales de transito fue que estos dieran paso a automóviles que estuvieran esperando si no había ningún automóvil recorriendo el sentido contrario.

6 Aprendizajes

Se puede decir que después de tanto problema provocado por los oficiales de tránsito el principal aprendizaje ha sido lo interesante y difícil que puede llegar a ser controlar los hilos de manera correcta cuando estos se manejan en grandes cantidades. Si se llegara a necesitar una gran cantidad de estos en un programa, se debe ser suficientemente cuidadoso para que no ocurran deadlocks y errores en la sincronizan de los mismos.

Después de esto, otro gran aprendizaje fue el de la utilización de los MUTEX, los cuales son una herramienta para precisamente evitar lo anterior mencionado en cuanto a la sincronización. C y Linux siguen dando sorpresas conforme se avanza en el curso y se aprenden funcionalidades nuevas de los sistemas operativos y el lenguaje de programación C.

References

[1] *<pthread.h>*. 2021. URL: https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html.

[2] *The Open Group Base Specifications Issue 7, 2018 edition*. 2021. URL: https://pubs.opengroup.org/onlinepubs/9699919799/.