Universidad Nacional de Costa Rica

Escuela de informática

Proyecto de Sistemas Operativos: Puente

Estudiante: José David Flores Rodríguez

Profesor: Eddy Rampirez

Fecha May 17, 2021

1 Introducción

Para este proyecto se nos pide 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 transito, 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 son de gran importancia ya que permite 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

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  };
29
```

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:

```
1   int iniciar_semaforo(); //Funcion principal que se encarga de
    iniciar las variables
2   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
3   void *create_semaforos(void *arg);        //Esta funcion
    encapsula la creacion del semaforo ubicado al este y oeste del
    puente
```

```
4     void *cambiar_semaforo_este(void *arg);    //Enciende el
      semaforo del oeste
5     void *cambiar_semaforo_oeste(void *arg);  //Enciende el
      semaforo del oeste
6     void *create_automoviles(void *arg);       //Funcion que crea
      los automoviles
7     void *automovil_este(void *arg);           //Crea los
      automoviles por que entran por el este
8     void *automovil_oeste(void *arg);          //Crea los
      automoviles que entran por el oeste
9     void *crear_autos_este(void *arg);       //Automovil que entra
      por el este
10    void *crear_autos_oeste(void *arg);        //Automovil que entra
       por el oeste
11    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;
12    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
13
```

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().

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 tomando 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 cunado 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.

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/.