**Práctica 5. Laboratorio Sistemas Operativos**

**ANA MARÍA PALACIO LÓPEZ C.C.**

**LEIDY ARRUBLA CASTRILLÓN C.C. 1033650045**

1. **Estrategia de solución inicial**

Inicialmente se planea realizar dos programas, uno en el que se realizan las transferencias entre cuentas y otro que ejecuta automáticamente el primero.

En el primer programa se creará una función llamada transferencias que será ejecutada por cada uno de los hilos. Esta función permitirá a un hilo transferir un monto aleatorio de una cuenta a otra.

Dado que el hilo principal debe crear tanto la cantidad de hilos como el numero de cuentas pasados como parámetros, se evidencia la necesidad de definir dos arreglos: uno de cuentas (con el tamaño según el parámetro numero\_cuentas) y otro de hilos (con el tamaño indicado por el parámetro numero de hilos).

Para controlar las condiciones de carrera y posibles interbloqueos entre hilos, se planea hacer uso de la clase **semaphore.**h, para definir una variable de tipo semáforo que permita controlar el acceso de los hilos a las cuentas, que son las regiones críticas del programa.

La inicialización de la variable de tipo semáforo se realizará en la función **main()**. Cada vez que un hilo ejecute la función transferencia, se bloqueará el semáforo y una vez dicho hilo finalice su tarea, el semáforo será desbloqueado y con esto otros hilos podrán hacer uso de las respectivas cuentas.

Para inicializar el semáforo se hará uso de la función:

**int sem\_init(sem\_t \*sem, int pshared, unsigned int value**), en la que sem será un apuntador al semáforo que se va a inicializar, **pshared** será cero puesto que el semáforo no será compartido con procesos fork() y **value** será el valor inicial para el semáforo.

Cuando todos los hilos terminen de ejecutarse, se utilizará la función **sem\_destroy()** para destruir el semáforo.

Cada hilo será creado la cantidad de veces que especifique el parámetro cantidad\_de\_tiempo, con esto se dará cumplimiento a la necesidad de cada hilo realice transferencias entre diferentes cuenta durante la cantidad de tiempo que se especifique como parámetro.

Para la selección aleatoria de cuentas por parte de cada hilo, al igual que para la definición del monto a transferir, se hará uso de la función **rand()** parametrizada de manera que no exceda la cantidad de cuentas existentes o el monto disponible.

Una vez seleccionadas las cuentas de destino y de origen, es necesario consultar el saldo de ambas cuentas antes de realizar la transferencia.

En caso que el saldo de la cuenta de origen sea cero, y que la cuenta de destino tenga saldo, la transferencia se realizará desde la cuenta de destino. En caso de que ambas cuentas tengan saldo igual a cero, el hilo no ejecutará ninguna transacción y liberará ambas cuentas para usos futuros.

1. **Problemas encontrados durante la solución de la práctica**

**Problema:** Durante la solución de la práctica hubo problemas con el uso de semáforos, porque inicialmente se planteó mal la estrategia. Para controlar las condiciones de carrera y los interbloqueos estábamos usando una sola variable de tipo semáforo, pero aún así los hilos seguían interfiriendo entre sí y los resultados obtenidos no eran los esperados.

Este error fue detectado porque en cada ejecución los resultados se presentaban de manera desordenada y nunca se realizaba la cantidad de transferencias que esperábamos.

**Solución:** Después de leer sobre el funcionamiento de los semáforos, entendimos que debíamos crear un semáforo por cada cuenta existente, porque éstas son las regiones críticas del programa y un solo semáforo no podía controlar adecuadamente el acceso de todos los hilos a las cuentas. Definimos entonces un arreglo global de semáforos (del tamaño de la cantidad de cuentas). Con esto, cada vez que un hilo ejecutaba la función transferencia, estábamos haciendo un down a los semáforos correspondientes a las cuentas implicadas. De esta manera logramos controlar efectivamente la concurrencia de hilos en las regiones críticas.

**Problema:** La generación de las cuentas de origen y de destino al igual que del monto a transferir, arrojaba siempre valores muy similares a pesar de que estábamos usando la función **rand()** para generar estos valores de manera aleatoria.

Este error fue detectado porque siempre al ejecutar el programa notábamos que en cada ejecución se quedaban varias cuentas sin realizar ningún movimiento y muchos de los montos a transferir eran cero o eran iguales entre sí.

**Solución:** Se incluyeron las siguientes líneas de código:

unsigned int seed = (unsigned) time(NULL) \* rand();

seed = seed \* (unsigned) time(NULL) + getpid();

Con estas líneas se inicializan los números aleatorios y se logra dar una mayor precisión a su generación.

1. **Solución final**

La estructura quedó compuesta por un programa principal en el que se realizan las transferencias y por otro de prueba que ejecuta automáticamente el principal.

En la solución final se implementaron las modificaciones mencionadas en el literal anterior, con esto se logró dar solución a los problemas encontrados.

Al ser creados los hilos, ejecutan la función **transferencia()** para realizar depósitos y retiros entre cuentas.

Para controlar las condiciones de carrera e interbloqueos se hizo uso de la clase **semaphore.h** con sus funciones sem\_init(), sem\_trywait() y sem\_destroy y se definió un arreglo de tipo semáforo del mismo tamaño de la cantidad de cuentas.

La generación de números aleatorios se hace con la función **rand().** Además se inicializan los números aleatorios antes de cada nueva generación.

Se hace uso de la variable de tipo booleano **desocupados** para indicar si un hilo se encuentra o no realizando una transferencia.

Se hace uso de la función **malloc()** para reservar espacio en memoria tanto para el arreglo de cuentas como para el de semáforos.

1. **Comparación solución inicial y solución final**

Inicialmente se estaba utilizando un solo semáforo para controlar las condiciones de carrera y los interbloqueos entre hilos, sin embargo, finalmente se hizo uso de un arreglo de semáforos que permitiera tener una variable de este tipo por cada cuenta existente.

Ennuestra estrategia inicial, para cumplir la condición de que un mismo hilo realizara transferencias durante el tiempo especificado en el parámetro cantidad\_tiempo\_a\_correr, estábamos generando en la función **main()** la cuenta de origen y la de destino de manera aleatoria, después de esto estábamos creando un mismo hilo cada vez que éste iba a ejecutar la función transferencia, es decir si un hilo debía realizar por ejemplo 5 transferencias, dicho hilo era creado 5 veces. Esto fue modificado, debido a que no era óptimo.

Con las modificaciones realizadas, cada hilo es creado una sola vez y es en la función transferencia donde cada uno de los hilos elige de manera aleatoria las cuentas de origen y de destino y realiza la transacción correspondiente.