Universidad del Valle de Guatemala
Sistemas Operativos
Docentes: Erick Pineda; Tomás Gálvez P.
Semestre I, 2021

Laboratorio #6

Fecha de Entrega: 26 de abril, 2021.

Descripción: este laboratorio aplicará los conceptos de *mutex lock*, semáforos y monitores para sincronizar *threads*. Se usarán las herramientas disponibles en Pthreads. Deberá entregar todo el código que escriba junto con archivos de texto que muestren una bitácora del consumo de recursos regulado por los semáforos y monitores, producida por sus programas.

Materiales: una máquina con Glibc.

Contenido:

Usted está encargad@ de gestionar el consumo de un recurso digital (*e.g.*, identificadores para *threads* en una *thread pool*), representado por una cantidad (`int`) ajustable[1]. Escriba un simulador que cree una cantidad de *threads* ajustable[1]. Todos los *threads* ejecutarán el mismo método. Mientras los *threads* se ejecutan, el programa principal (`main`) sólo esperará.

El método que los *threads* ejecutarán debe poseer un ciclo cuyo número de iteraciones es también ajustable[1] pero igual para todos los *threads*. En el ciclo se deberá consumir uno de los recursos (*i.e.*, decrementar la cantidad de recursos disponible) y luego esperar por una cantidad de tiempo aleatoria para simular que el *thread* está haciendo algo con ese recurso. Al concluir la espera deberá devolver el recurso consumido antes de iniciar una nueva iteración.

Su primer programa debe usar semáforos de Pthreads para regular el consumo y devolución de recursos sin incurrir en *race conditions* sobre la cantidad de recursos disponible.
Su segundo programa mejorará este esquema incluyendo funciones que permitan consumir una cantidad dada de recursos en lugar de solo uno. A continuación, se ilustra la estructura básica de estas funciones:

```
/* decrease available resources by count resources */
/* return 0 if sufficient resources available, */
/* otherwise return -1 */
int decrease_count(int count) {
     if (available_resources < count)
          return -1;
     else {
          available_resources -= count;
          return 0;
     }
}

/* increase available resources by count */
int increase_count(int count) {
     available_resources += count;
     return 0;
}
```

[1]: ajustable no necesariamente significa ajustable por el usuario. Sólo debe ser algo que se pueda modificar, como con un `#define`, no valores *hard-coded* en las funciones.

El uso de estas funciones también manifiesta problemas de sincronización, por lo que su uso se debe regular con un monitor. C no tiene objetos de tipo monitor por defecto, entonces puede desarrollar su propia clase monitor (con C++) o implementar el comportamiento de un monitor mediante semáforos y variables de condición de Pthreads (como discutido durante la presentación; ver libro de los dinosaurios de Silberschatz).

Ambos de sus programas deben producir una bitácora (archivo de texto) que registre cómo los semáforos y el monitor administran el uso de los recursos. Sea tan detallado como pueda para que esta bitácora sea fácil de entender (*e.g.*, identifique los *threads*, avise cuando un *thread* consume o espera recursos, etc.). En Canvas se incluyen ejemplos de cómo podrían verse las bitácoras producidas por sus programas. A continuación, capturas de ejemplo de la bitácora con semáforos (izq.) y monitor (der.):

```
Iniciando programa
Creando threads
Iniciando thread 111
Iniciando thread 112
Iniciando thread 113
Iniciando thread 114
Iniciando thread 115
Iniciando thread 116
Iniciando thread 117
Iniciando thread 118
Iniciando thread 119
Esperando threads
Iniciando thread 120
112 - Semaforo abierto con exito.
Iniciando iteracion 1
112 - (!) Recurso tomado
114 - Semaforo abierto con exito.
Iniciando iteracion 1
113 - Semaforo abierto con exito.
Iniciando iteracion 1
116 - Semaforo abierto con exito.
Iniciando iteracion 1
112 - Buenos dias! Recurso usado
112 - Recurso devuelto :)
Iniciando iteracion 2
112 - (!) Recurso tomado
115 - Semaforo abierto con exito.
Iniciando iteracion 1
117 - Semaforo abierto con exito.
Iniciando iteracion 1
118 - Semaforo abierto con exito.
Iniciando iteracion 1
120 - Semaforo abierto con exito.
Iniciando iteracion 1
112 - Buenos dias! Recurso usado
112 - Recurso devuelto :)
Iniciando iteracion 3
112 - (!) Recurso tomado
112 - Buenos dias! Recurso usado
112 - Recurso devuelto :)
Iniciando iteracion 4
112 - (!) Recurso tomado
119 - Semaforo abierto con exito.
Iniciando iteracion 1
111 - Semaforo abierto con exito.
Iniciando iteracion 1
112 - Buenos dias! Recurso usado
112 - Recurso devuelto :)
```

```
Iniciando programa
Creando threads
Iniciando thread 145
Iniciando iteracion 1
Se consumiran 21 recursos
Iniciando decrease_count
Mutex adquirido, entrando al monitor
Recursos suficientes disponibles, consumiendo...
Terminando decrease_count
145 - (!) Recurso tomado
Iniciando thread 146
Iniciando iteracion 1
Se consumiran 21 recursos
Iniciando decrease_count
Mutex adquirido, entrando al monitor
NUAY! Recursos actuales: 9
Iniciando thread 147
Iniciando iteracion 1
Se consumiran 21 recursos
Iniciando decrease_count
Mutex adquirido, entrando al monitor
NUAY! Recursos actuales: 9
Iniciando thread 148
Iniciando iteracion 1
Se consumiran 21 recursos
Iniciando decrease_count
Mutex adquirido, entrando al monitor
NUAY! Recursos actuales: 9
Iniciando thread 149
Iniciando iteracion 1
Se consumiran 21 recursos
Iniciando decrease_count
Mutex adquirido, entrando al monitor
NUAY! Recursos actuales: 9
Iniciando thread 150
Iniciando iteracion 1
Se consumiran 21 recursos
Iniciando decrease_count
Mutex adquirido, entrando al monitor
NUAY! Recursos actuales: 9
Iniciando thread 151
Iniciando iteracion 1
Se consumiran 21 recursos
Iniciando decrease_count
Mutex adquirido, entrando al monitor
NUAY! Recursos actuales: 9
Iniciando thread 152
```