

Fecha de Entrega: 15 de abril, 2022.

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<sup>1</sup>. Escriba un simulador que cree una cantidad de *threads* ajustable<sup>1</sup>. 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<sup>1</sup> 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;
}
```

<sup>1</sup>: 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.):

1	Iniciando programa	1	Iniciando programa
2	Creando threads	2	Creando threads
3	Iniciando thread 111	3	Iniciando thread 145
4	Iniciando thread 112	4	Iniciando iteracion 1
5	Iniciando thread 113	5	Se consumiran 21 recursos
6	Iniciando thread 114	6	Iniciando decrease_count
7	Iniciando thread 115	7	Mutex adquirido, entrando al monitor
8	Iniciando thread 116	8	Recursos suficientes disponibles, consumiendo...
9	Iniciando thread 117	9	Terminando decrease_count
10	Iniciando thread 118	10	145 - (!) Recurso tomado
11	Iniciando thread 119	11	Iniciando thread 146
12	Esperando threads	12	Iniciando iteracion 1
13	Iniciando thread 120	13	Se consumiran 21 recursos
14	112 - Semaforo abierto con exito.	14	Iniciando decrease_count
15	Iniciando iteracion 1	15	Mutex adquirido, entrando al monitor
16	112 - (!) Recurso tomado	16	NUAY! Recursos actuales: 9
17	114 - Semaforo abierto con exito.	17	Iniciando thread 147
18	Iniciando iteracion 1	18	Iniciando iteracion 1
19	113 - Semaforo abierto con exito.	19	Se consumiran 21 recursos
20	Iniciando iteracion 1	20	Iniciando decrease_count
21	116 - Semaforo abierto con exito.	21	Mutex adquirido, entrando al monitor
22	Iniciando iteracion 1	22	NUAY! Recursos actuales: 9
23	112 - Buenos dias! Recurso usado	23	Iniciando thread 148
24	112 - Recurso devuelto :)	24	Iniciando iteracion 1
25	Iniciando iteracion 2	25	Se consumiran 21 recursos
26	112 - (!) Recurso tomado	26	Iniciando decrease_count
27	115 - Semaforo abierto con exito.	27	Mutex adquirido, entrando al monitor
28	Iniciando iteracion 1	28	NUAY! Recursos actuales: 9
29	117 - Semaforo abierto con exito.	29	Iniciando thread 149
30	Iniciando iteracion 1	30	Iniciando iteracion 1
31	118 - Semaforo abierto con exito.	31	Se consumiran 21 recursos
32	Iniciando iteracion 1	32	Iniciando decrease_count
33	120 - Semaforo abierto con exito.	33	Mutex adquirido, entrando al monitor
34	Iniciando iteracion 1	34	NUAY! Recursos actuales: 9
35	112 - Buenos dias! Recurso usado	35	Iniciando thread 150
36	112 - Recurso devuelto :)	36	Iniciando iteracion 1
37	Iniciando iteracion 3	37	Se consumiran 21 recursos
38	112 - (!) Recurso tomado	38	Iniciando decrease_count
39	112 - Buenos dias! Recurso usado	39	Mutex adquirido, entrando al monitor
40	112 - Recurso devuelto :)	40	NUAY! Recursos actuales: 9
41	Iniciando iteracion 4	41	Iniciando thread 151
42	112 - (!) Recurso tomado	42	Iniciando iteracion 1
43	119 - Semaforo abierto con exito.	43	Se consumiran 21 recursos
44	Iniciando iteracion 1	44	Iniciando decrease_count
45	111 - Semaforo abierto con exito.	45	Mutex adquirido, entrando al monitor
46	Iniciando iteracion 1	46	NUAY! Recursos actuales: 9
47	112 - Buenos dias! Recurso usado	47	Iniciando thread 152
48	112 - Recurso devuelto :)		