



Tarea N°2

Sistemas Operativos

CIT2010

Fecha de entrega: 03/11/2024

Tomás León
Hugo Rojas

Índice

1	Introducción	1
2	Desarrollo	1
2.1	Solución Principal	1
2.2	Deadlock y Livelock	3
2.2.1	Implementación de Deadlock	4
2.2.2	Implementación del Livelock	6
3	Resultados	8
3.1	Solución Principal	8
3.2	Deadlock y Livelock	10
3.2.1	Deadlock	11
3.2.2	Livelock	11
4	Análisis	12
4.1	Análisis de la solución principal	12
4.2	Análisis de las Condiciones del Deadlock	13
4.3	Análisis del Livelock	13
4.3.1	Importancia de las Variables volatile	13
5	Conclusión	13
6	Anexos	14

1 Introducción

La presente experiencia busca la implementación de threads y herramientas de sincronización, a manera de resolver el problema planteado, el cual consiste en una colaboración entre profesores de la EIT para optimizar el acceso a contenido en plataformas de streaming, dada la limitación de recursos para suscribirse a múltiples servicios. A su vez, se realizaran cambios a la solución para la generación de un tipo de deadlock y livelock.

2 Desarrollo

En esta sección se detalla la implementación y explicación del código para resolver el problema de tipo productor consumidor. Primero, se presenta la solución correcta con una explicación de los mecanismos de sincronización empleados, acompañado con imágenes del código. Posteriormente, se realiza una descripción individual de las condiciones de *livelock* y *deadlock*, explicando los cambios introducidos en el código y cómo estos afectan el comportamiento del sistema.

2.1 Solución Principal

Inicialmente, se procederán a establecer las variables globales a utilizar, la figura 1 muestra la forma en la que se realizó

```

1 float TseriesD;
2 float TseriesB;
3 float StseriesD = 0;
4 float StseriesB = 0;
5
6 float t[] = {0.5, 1.0, 1.5, 2.0};
7 float tiempoProfesorD[6];
8 float tiempoProfesorB[6];
9 float sumatoriaD = 0;
10 float sumatoriaB = 0;
11 int Bool = 0;
12 int Nsemana = 1;
13 float auxD = 0;
14 float auxB = 0;
```

Figura 1: Variables Globales.

Las variables `TseriesD` y `TseriesB` de la imagen representan las series disponibles de las plataformas '*Dasney*' y '*Betflix*', mientras que las líneas 3 y 4 servirán como memoria para acumular las series restantes de cada semana.

La línea 6 del código corresponde a un arreglo de las cantidades de series que los profesores ven por semana, mientras que las variables `tiempoProfesorD` y `tiempoProfesorB` servirán para guardar dichos tiempos y de esa manera realizar los cálculos de las series vistas a la semana. Por otro lado, `sumatoriaD` y `sumatoriaB` guardará el total de series vistas por ambos grupos de profesores.

Finalmente, las variables `Bool` y `Nsemana` ayudan al código a llevar una cuenta de los hilos que terminan en cada ciclo, y el número de la semana actual respectivamente. Además, las últimas dos variables son auxiliares que guardarán los valores de las series restantes por semana que no se pudieron ver.

Ya establecidas las variables globales, es posible tratar con ellas para cumplir los objetivos planteados para la actividad. Se reconoce que el problema a resolver es un tipo de problema productor-consumidor, lo que facilita la elección de los métodos de sincronización a utilizar. Para este tipo de problemas, se decidió utilizar mutex locks y semáforos debido a que permiten proteger el recurso, que corresponde a las series, de ser modificadas simultáneamente por los diferentes hilos, correspondiente a los profesores, en cada ciclo. Esto significa que cada profesor registrará las series que ha visualizado exclusivamente, evitando el registro de datos incorrectos.

```

1 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
2 pthread_mutex_t mutexdebool = PTHREAD_MUTEX_INITIALIZER;

```

Figura 2: Definición de mutex locks.

La figura 2 muestra la manera en la que se definen los mutex lock a utilizar, se definieron dos mutex los cuales protegerán el acceso a las diferentes zonas críticas y sus variables compartidas. Posteriormente, se definieron dos funciones las cuales ayudarán a definir la cantidad de series que ven los profesores, y por otro lado la cantidad de series que las plataformas producen por semana.

```

1 float t_serie(void){
2     int indice = rand() % 4;
3     return t[indice];
4 }
5
6 int c_series(void){
7     int series = rand() % 6 + 10;
8     return series;
9 }

```

Figura 3: Funciones de definición de cantidad y tiempo de series.

La función llamada **t_serie** retorna el valor dentro del arreglo **t[]** correspondiente a un índice obtenido aleatoriamente, mientras que la función **c_series** es la que busca aleatoriamente números entre el 10 y el 15 para obtener la cantidad de series que se ofrecen semanalmente.

Para la función **main**, se definen la cantidad de threads a utilizar, en el caso de el problema se especifica que seran 6 profesores para Disney y 6 para Netflix. En esta sección del código se establece el menú para que el usuario pueda elegir el tiempo de la ejecución (1 mes, 6 meses o 1 año). Es importante destacar que los hilos se sincronizan con el proceso principal mediante la función **pthread_join** (ver Figura 4b), que asegura que cada hilo finalice antes de que el programa principal continúe. Para luego crear nuevos que cumplirán la misma función para las siguientes semanas correspondientes al tiempos de la simulación seleccionado. La estructura de bucles en **main** controla el avance de las semanas y asegura que todos los hilos completen sus tareas antes de que la simulación avance.

```

1 int main(void){
2
3     srand((unsigned int)time(NULL));
4     Pd = 6;
5     Ps = 6;
6     pthread_t P0[Pd];
7     pthread_t PB[Ps];
8     pthread_t generador;
9     sem_init(&semaforoB, 0, Ps);
10    sem_init(&semaforoD, 0, Pd);
11    N = 4;
12    int Ctempo = 0;
13    printf("Imprese el tiempo:\n[1] 1 Mes\n[2] 6 Mes\n[3] 1 Año\n");
14    scanf("%d", &N);
15    if(N <= 0 || N > 3){
16        while(1){
17            printf("El numero debe ser mayor a 0 y menor a 3\n");
18            printf("Imprese el tiempo:\n");
19            scanf("%d", &N);
20            if(N > 0 && N < 4){
21                break;
22            }
23        }
24    }
25    if(N == 1){
26        Ctempo = 4;
27    }else if(N == 2){
28        Ctempo = 24;
29    }else if(N == 3){
30        Ctempo = 48;
31    }
32
33    pthread_mutex_init(&mutex, NULL);
34    pthread_mutex_init(&mutexdebool, NULL);

```



```

1     while(Ctempo > 0){
2         Bool = 0;
3         sumatoriaB = 0;
4         sumatoriaD = 0;
5         TseriesB = C.series();
6         TseriesD = C.series();
7         printf("-----\n");
8         printf("Semana %d\n", Nsemana);
9         printf("-----\n");
10        printf("Esta semana salieron %.1f series de Disney\n", TseriesB);
11        printf("Esta semana salieron %.1f series de Netflix\n", TseriesD);
12        auxB = TseriesB;
13        auxD = TseriesD;
14
15        for(int i = 0; i < Ps; i++){
16            idB = i + 1;
17            pthread_create(&P0[i], NULL, verserie, (void *)idB);
18        }
19        for(int i = 0; i < Ps; i++){
20            idB = i + Ps + 1;
21            pthread_create(&PB[i], NULL, verserie, (void *)idB);
22            *idB = i + Ps + 1;
23        }
24        for(int i = 0; i < Ctempo; i++){
25            pthread_join(P0[i], NULL);
26            pthread_join(PB[i], NULL);
27        }
28        Ctempo--;
29        Nsemana++;
30        TseriesD = 0;
31        TseriesB = 0;
32    }
33
34 }

```

(a) Función main 1.

(b) Función main 2.

Figura 4: Funcion main.

En la figura 5 se puede ver la declaración de la función `verserie`, en la cual se identificaran los profesores para cada plataforma, seguido del tiempo que verán cada serie. Luego de esto, se activarán los semáforos para ambas plataformas, seguido del bloqueo del mutex. La zona crítica de la función se encarga de verificar si la plataforma tiene suficiente contenido disponible para satisfacer el tiempo de visualización del profesor. Si no es así, el profesor ve el contenido restante para luego acumular el tiempo de visualización y actualizar las variables que rastrean el progreso de cada hilo. Además, se incrementará el valor de la variable `Bool` hasta llegar a 12, donde el ultimo thread podrá imprimir los datos de la ejecución por semana.

```

1 void *verserie(void *arg){
2
3 int id = *(int *)arg;
4 float Tvisualizacion = t_serie();
5 sem_wait(&semaforoD);
6 sem_wait(&semaforoB);
7 pthread_mutex_lock(&mutex);
8
9 if(id < (Pd + 1)){
10     if(TseriesD >= Tvisualizacion){
11         TseriesD = TseriesD - Tvisualizacion;
12         TdeCadaProfesorD[id - 1] = Tvisualizacion;
13     }
14     else{
15         Tvisualizacion = TseriesD;
16         TdeCadaProfesorD[id - 1] = Tvisualizacion;
17     }
18 }else {
19     if (TseriesB > Tvisualizacion){
20         TseriesB = TseriesB - Tvisualizacion;
21         TdeCadaProfesorB[id - (Pd + 1)] = Tvisualizacion;
22     }
23     else{
24         Tvisualizacion = TseriesB;
25         TdeCadaProfesorB[id - (Pd + 1)] = Tvisualizacion;
26     }
27 }
28 pthread_mutex_lock(&mutexdebool);
29 Bool++;
30 pthread_mutex_unlock(&mutexdebool);
31
32 [...]

```

Figura 5: Función para visualizar series.

2.2 Deadlock y Livelock

Tras resolver el problema planteado, se requiere modificar el código para generar situaciones de tipo Deadlock y Livelock. Este capítulo describirá dichas modificaciones. Cabe destacar que tanto el Deadlock como el Livelock están en el mismo código; por ello, primero se analizará la función principal `main`, y luego se examinarán las funciones de Deadlock y Livelock de forma independiente.

```

1 srand((unsigned int)time(NULL));
2     pthread_t PD[Pd]; // Profesores que van Dasney
3     pthread_t PB[Pb]; // Profesores que van Netflix
4
5     int N = 0;
6     int Ctiempo = 0;
7     int choice = 0;
8
9     printf("Seleccione el modo:\n");
10    printf("1. Demostración de deadlock\n");
11    printf("2. Demostración de livelock\n");
12    scanf("%d", &choice);
13
14    void *(*selected_function)(void *);
15    if (choice == 1) {
16        selected_function = deadlock_function;
17    } else if (choice == 2) {
18        selected_function = livelock_function;
19    }
20
21    printf("Ingrese el tiempo:\n[1] 1 Mes\n[2] 6 Mes\n[3] 1 Año\n");
22    scanf("%d", &N);
23    if (N == 1) {
24        Ctiempo = 4;
25    } else if (N == 2) {
26        Ctiempo = 24;
27    } else if (N == 3) {
28        Ctiempo = 48;
29    }
30
31
32    //While de semanas..
33
34    return 0;
35 }
36

```

Figura 6: Función principal `main` del código de Deadlock y Livelock.

En la figura 6, se observa que el programa primero solicita seleccionar el modo de ejecución. Luego, se define la variable `void *(*selected_function)(void *)`, que permite almacenar la función elegida (Deadlock o Livelock). Esta variable es un puntero a una función, lo que permite que la función seleccionada se ejecute. La selección se realiza a través de una entrada por consola. Además, se ha removido el uso de semáforos ya que no son necesarios para verificar la ocurrencia de Deadlock o Livelock en el contexto del problema, debido a que el número de profesores se mantiene en seis por plataforma. Con esto explicado, se describen las funciones de Deadlock y Livelock por separado.

2.2.1 Implementación de Deadlock

La función encargada de introducir el deadlock en el sistema es `deadlock_function` (Figura 7). Su objetivo es simular una situación donde dos o más hilos (profesores) quedan bloqueados indefinidamente al intentar adquirir recursos (series) que están siendo utilizados por otros hilos. A continuación, se describe paso a paso cómo trabaja esta función:

```

1 void *deadlock_function(void *arg) {
2     int id = *(int *)arg;
3     float Tvisualizacion = t_serie();
4     printf("Profesor %d intentando adquirir recursos para deadlock...\n", id);
5
6     if (id < (Pd + 1)) {
7         pthread_mutex_lock(&mutexD);
8         printf("Profesor Dasney %d adquirió mutexD\n", id);
9         sleep(1);
10        if(TseriesD >= Tvisualizacion){
11            TseriesD = TseriesD - Tvisualizacion;
12            TdeCadaProfesorD[id - 1] = Tvisualizacion;
13        }
14        else{
15            Tvisualizacion = TseriesD;
16            TdeCadaProfesorD[id - 1] = Tvisualizacion;
17        }
18        pthread_mutex_lock(&mutexB);
19        printf("Profesor Betflix %d adquirió mutexB\n", id);
20    } else {
21        pthread_mutex_lock(&mutexB);
22        printf("Profesor Betflix %d adquirió mutexB\n", id);
23        sleep(1);
24        if (TseriesB > Tvisualizacion){
25            TseriesB = TseriesB - Tvisualizacion;
26            TdeCadaProfesorB[id - (Pd + 1)] = Tvisualizacion;
27        }
28        else{
29            Tvisualizacion = TseriesB;
30            TdeCadaProfesorB[id - (Pd + 1)] = Tvisualizacion;
31        }
32        pthread_mutex_lock(&mutexD);
33        printf("Profesor Betflix %d adquirió mutexD\n", id);
34    }
35    pthread_mutex_unlock(&mutexD);
36    pthread_mutex_unlock(&mutexB);
37    pthread_mutex_lock(&mutexdebool);
38    Bool++;
39    pthread_mutex_unlock(&mutexdebool);
40
41    if(Bool == (Pd + Pb)){
42        printf("-----\n");
43        printf("Print de datos, que no se verá porque esta en deadlock\n");
44        printf("-----\n");
45    }
46    free(arg);
47    return NULL;
48 }
49

```

Figura 7: Código Deadlock.

Descripción del Funcionamiento

- Primero, se analizará los profesores al igual que en la solución principal; profesor de Dasney (`id < 7`) o de Betflix (`id >= 7`).
- **Adquisición de Recursos:**
 - *Profesores de Dasney:*
 1. Adquieren el `mutexD`.
 2. Esperan 1 segundo (`sleep(1)`) para simular tiempo de procesamiento.
 3. Intentan adquirir el `mutexB`.
 - *Profesores de Betflix:*
 1. Adquieren el `mutexB`.
 2. Esperan 1 segundo (`sleep(1)`).
 3. Intentan adquirir el `mutexD`.
- **Possible Situación de Deadlock:** Si un profesor de Dasney ha adquirido `mutexD` y un profesor de Betflix ha adquirido `mutexB`, ambos hilos intentarán adquirir el mutex que posee el otro después de esperar un segundo gracias a la función `sleep`. Esto provoca que ambos hilos queden bloqueados indefinidamente, ya que ninguno puede avanzar sin que el otro libere el recurso que posee.

- **Liberación de Recursos:** Si un hilo logra adquirir ambos mutexes, procede a liberarlos y termina su ejecución. Sin embargo, en una situación de deadlock, este punto nunca se alcanza.
- **Eliminación de lógica de series:** Para observar la condición de Deadlock de manera más clara, se ha eliminado la lógica relacionada con las visualizaciones de series. Este cambio tiene como objetivo simplificar el código y facilitar la comprensión del lector respecto a las condiciones que provocan el Deadlock. La figura 8 muestra el código sin la lógica de series:

```

1 void *deadlock_function(void *arg) {
2     int id = *(int *)arg;
3     float Tvisualizacion = t_serie();
4     printf("Profesor %d intentando adquirir recursos para deadlock...\n", id);
5
6     if (id < (Pd + 1)) {
7         pthread_mutex_lock(&mutexD);
8         printf("Profesor Dasney %d adquirió mutexD\n", id);
9         sleep(1);
10        // Logica de series
11        pthread_mutex_lock(&mutexB);
12        printf("Profesor Dasney %d adquirió mutexB\n", id);
13    } else {
14        pthread_mutex_lock(&mutexB);
15        printf("Profesor Betflix %d adquirió mutexB\n", id);
16        sleep(1);
17        // Logica de series
18        pthread_mutex_lock(&mutexD);
19        printf("Profesor Betflix %d adquirió mutexD\n", id);
20    }
21    pthread_mutex_unlock(&mutexD);
22    pthread_mutex_unlock(&mutexB);
23
24
25    free(arg);
26    return NULL;
27 }
```

Figura 8: Código de Deadlock sin lógica de visualización de series.

2.2.2 Implementación del Livelock

La función que introduce el livelock es `livelock_function`(Figura 9). En este caso, los threads siguen activos y cambiando de estado en respuesta a las acciones de otros hilos, pero sin hacer progreso real en sus tareas. A continuación, se explica detalladamente su funcionamiento:

```

1 volatile int trying_dasney = 1;
2 volatile int trying_betflix = 1;
3 void *livelock_function(void *arg) {
4     int id = *(int *)arg;
5     float Tvisualizacion = t_serie();
6
7     while (1) {
8         if (id < (Pd + 1)) {
9             trying_dasney = 1;
10            if (trying_betflix) {
11                printf("Profesor Dasney %d cediendo a profesores de Netflix...\n", id);
12                sleep(1);
13                continue;
14            }
15            pthread_mutex_lock(&mutexD);
16            trying_dasney = 0;
17            printf("Profesor Dasney %d en región crítica\n", id);
18            if (TseriesD >= Tvisualizacion) {
19                TseriesD -= Tvisualizacion;
20                TdeCadaProfesorD[id - 1] = Tvisualizacion;
21            } else {
22                Tvisualizacion = TseriesD;
23                TdeCadaProfesorD[id - 1] = Tvisualizacion;
24                TseriesD = 0;
25            }
26            pthread_mutex_unlock(&mutexD);
27            break;
28        } else {
29            trying_betflix = 1;
30            if (trying_dasney) {
31                printf("Profesor Netflix %d cediendo a profesores de Dasney...\n", id);
32                sleep(1);
33                continue;
34            }
35            pthread_mutex_lock(&mutexB);
36            trying_betflix = 0;
37            printf("Profesor Netflix %d en región crítica\n", id);
38            if (TseriesB >= Tvisualizacion) {
39                TseriesB -= Tvisualizacion;
40                TdeCadaProfesorB[id - 7] = Tvisualizacion;
41            } else {
42                Tvisualizacion = TseriesB;
43                TdeCadaProfesorB[id - 7] = Tvisualizacion;
44                TseriesB = 0;
45            }
46            pthread_mutex_unlock(&mutexB);
47            break;
48        }
49        pthread_mutex_lock(&mutexdebool);
50        Bool++;
51        pthread_mutex_unlock(&mutexdebool);
52        if(Bool == (Pd + Pb)){
53            printf("-----\n");
54            printf("Print de datos, que no se verá porque esta en livelock\n");
55            printf("-----\n");
56        }
57    }
58    free(arg);
59    return NULL;
60 }
61 }
```

Figura 9: Código Livelock.

Descripción del Funcionamiento

- **Variables Volátiles:** Las variables globales `trying_dasney` y `trying_betflix`, declaradas como `volatile int`, indican si los profesores de cada plataforma están intentando acceder a la sección crítica. Idealmente, estas variables deberían estar inicializadas en 0, ya que inicialmente ningún grupo está intentando acceder a la sección crítica. Sin embargo, para evitar condiciones de espera adicionales y simplificar el flujo de ejecución, se optó por inicializarlas en 1, permitiendo que el proceso de verificación y cesión de paso inicie sin una condición de espera previa.
- **Ciclo de Livelock:**
 - Cada profesor establece que está intentando acceder a la sección crítica asignando `trying_dasney = 1` o `trying_betflix = 1`.

- Verifica si el otro grupo está intentando acceder:
 - * Si el otro grupo está intentando, cede el paso, imprime un mensaje y vuelve al inicio del ciclo tras esperar 1 segundo, gracias a la función sleep.
 - * Si no, procede a establecer su intención a 0 y entra en la sección crítica.
- **Situación de Livelock:** Debido a que ambos grupos siempre detectan que el otro está intentando acceder, ambos ceden el paso repetidamente sin avanzar, generando un ciclo infinito de inacción.
- **Eliminación de lógica de series:** Al igual que en la sección anterior, se eliminará la lógica de visualización de series para simplificar el análisis del código y hacer más comprensibles las condiciones que generan el livelock. En la figura 10, se muestra el código de la función de livelock sin esta lógica adicional.

```

1 volatile int trying_dasney = 1;
2 volatile int trying_betflix = 1;
3 void *livelock_function(void *arg) {
4     int id = *(int *)arg;
5     float Tvisualizacion = t_serie();
6
7     while (1) {
8         if (id < (Pd + 1)) { // Profesor Dasney
9             trying_dasney = 1;
10            if (trying_betflix) {
11                printf("Profesor Dasney %d cediendo a profesores de Betflix...\n", id);
12                sleep(1);
13                continue;
14            }
15            pthread_mutex_lock(&mutexD);
16            trying_dasney = 0;
17            printf("Profesor Dasney %d en región crítica\n", id);
18            // Logica de series
19            pthread_mutex_unlock(&mutexD);
20            break;
21        } else { // Profesor Betflix
22            trying_betflix = 1;
23            if (trying_dasney) {
24                printf("Profesor Betflix %d cediendo a profesores de Dasney...\n", id);
25                sleep(1);
26                continue;
27            }
28            // Procede si el otro no está intentando
29            pthread_mutex_lock(&mutexB);
30            trying_betflix = 0;
31            printf("Profesor Betflix %d en región crítica\n", id);
32            // Logica de series
33            pthread_mutex_unlock(&mutexB);
34            break;
35        }
36    }
37    free(arg);
38    return NULL;
39 }

```

Figura 10: Código de Livelock sin lógica de visualización de series.

3 Resultados

3.1 Solución Principal

Para ejecutar el código, es necesario compilarlo antes de ello utilizando el comando `gcc -o series series.c` dentro de la terminal. En la Figura 11 se muestra el resultado de la compilación.

```

4: gcc -o series series.c
series.c:1405:5: warning: 'sem_init' is deprecated [-Wdeprecated-declarations]
 55 |     sem_init(&semFor0, 0, P0);
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/sema.h:55:42: note: 'sem_init' has been explicitly marked deprecated here
|     55 | int sem_init(_In_opt_ int, unsigned int) __deprecated;
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h:214:40: note: expanded from macro '__deprecated'
| 214 | #define __deprecated __attribute__((__deprecated__))
series.c:1405:5: warning: 'sem_init' is deprecated [-Wdeprecated-declarations]
 56 |     sem_init(&semFor0, 0, P0);
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/sema.h:55:42: note: 'sem_init' has been explicitly marked deprecated here
|     56 | int sem_init(_In_opt_ int, unsigned int) __deprecated;
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h:214:40: note: expanded from macro '__deprecated'
| 214 | #define __deprecated __attribute__((__deprecated__))
series.c:211:5: warning: 'sem_destroy' is deprecated [-Wdeprecated-declarations]
 211 |     sem_destroy(&semFor0);
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/sema.h:53:26: note: 'sem_destroy' has been explicitly marked deprecated here
|     53 | int sem_destroy(_In_opt_ int) __deprecated;
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h:214:40: note: expanded from macro '__deprecated'
| 214 | #define __deprecated __attribute__((__deprecated__))
series.c:211:5: warning: 'sem_destroy' is deprecated [-Wdeprecated-declarations]
 212 |     sem_destroy(&semFor0);
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/sema.h:53:26: note: 'sem_destroy' has been explicitly marked deprecated here
|     53 | int sem_destroy(_In_opt_ int) __deprecated;
| Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h:214:40: note: expanded from macro '__deprecated'
| 214 | #define __deprecated __attribute__((__deprecated__))
4 warnings generated.

```

Figura 11: Compilación de la solución principal.

Al compilar, se presentan varias advertencias generadas por el sistema operativo macOS, debido a la obsolescencia de semáforos sin nombre en este sistema. En particular, las funciones `sem_init` y `sem_destroy` están marcadas como obsoletas, lo que podría significar que no se soporten en el futuro. Como el código utiliza dos semáforos sin nombre, se generan cuatro de esas advertencias en total. Durante la investigación, se encontró una alternativa mediante el uso de semáforos con nombre, sin embargo, esta opción generó problemas en el funcionamiento del código. Dado que los semáforos en su estado actual cumplen con la funcionalidad requerida y tienen un uso simbólico, se decidió continuar pese a los warnings.

En las Figuras 12, 13 y 14 se muestra la ejecución del código. La Figura 12 presenta el mensaje inicial donde se elige el tiempo de ejecución deseado, desde un mes, seis meses y un año.

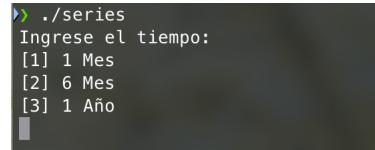


Figura 12: Ejecución de la solución principal: Selección de tiempo.

La Figura 13 muestra la salida correspondiente a la primera semana al seleccionar la opción de un mes.

```

Ingrese el tiempo:
[1] 1 Mes
[2] 6 Mes
[3] 1 Año
1
-----
Semana 1
-----
Esta semana salieron 15.0 series de Disney
Esta semana salieron 14.0 series de Betflix
-----
El profesor 1 vio 2.0 series de Disney
El profesor 2 vio 1.0 series de Disney
El profesor 3 vio 1.5 series de Disney
El profesor 4 vio 0.5 series de Disney
El profesor 5 vio 1.5 series de Disney
El profesor 6 vio 0.5 series de Disney
-----
El profesor 1 vio 2.0 series de Betflix
El profesor 2 vio 1.0 series de Betflix
El profesor 3 vio 1.5 series de Betflix
El profesor 4 vio 1.5 series de Betflix
El profesor 5 vio 2.0 series de Betflix
El profesor 6 vio 1.0 series de Betflix
-----
Los profesores vieron 7.0 series de Disney
Los profesores vieron 9.0 series de Betflix
-----
Quedan 8.0 series de totales de Disney por ver
Quedan 5.0 series de totales de Betflix por ver
-----
Semana 2
-----
Esta semana salieron 10.0 series de Disney
Esta semana salieron 13.0 series de Betflix
-----
Series acumuladas faltantes por ver de la semana pasada de Disney: 8.0
Series acumuladas faltantes por ver de la semana pasada de Betflix: 5.0
-----
Series acumuladas por ver Disney: 18.0
Series acumuladas por ver Netflix: 18.0
-----
El profesor 1 vio 1.0 series de Disney

```

Figura 13: Ejecución de la solución principal: Semana 1.

Finalmente, la Figura 14 presenta la salida de la cuarta semana y la finalización exitosa de la ejecución.

```

Quedan 11.5 series de totales de Betflix por ver
-----
Semana 4
-----
Esta semana salieron 10.0 series de Dasney
Esta semana salieron 13.0 series de Betflix
-----
Series acumuladas faltantes por ver de las semanas pasadas de Dasney: 19.5
Series acumuladas faltantes por ver de las semanas pasadas de Betflix: 11.5
-----
Series acumuladas por ver Dasney: 29.5
Series acumuladas por ver Netflex: 24.5
-----
El profesor 1 vio 1.5 series de Dasney
El profesor 2 vio 1.0 series de Dasney
El profesor 3 vio 1.0 series de Dasney
El profesor 4 vio 2.0 series de Dasney
El profesor 5 vio 0.5 series de Dasney
El profesor 6 vio 2.0 series de Dasney
-----
El profesor 1 vio 1.0 series de Betflix
El profesor 2 vio 2.0 series de Betflix
El profesor 3 vio 1.0 series de Betflix
El profesor 4 vio 1.5 series de Betflix
El profesor 5 vio 2.0 series de Betflix
El profesor 6 vio 0.5 series de Betflix
-----
Los profesores vieron 8.0 series de Dasney
Los profesores vieron 8.0 series de Betflix
-----
Quedan 21.5 series de totales de Dasney por ver
Quedan 16.5 series de totales de Betflix por ver

```

Figura 14: Ejecución de la solución principal: Semana 4.

Por otra parte, en la Figura 15 se muestra la ejecución de la última semana (semana 48), correspondiente a la opción de un año de ejecución.

```

Los profesores vieron 7.0 series de Dasney
Los profesores vieron 8.5 series de Betflix
-----
Quedan 209.5 series de totales de Disney por ver
Quedan 233.0 series de totales de Betflix por ver
-----
Semana 48
-----
Esta semana salieron 15.0 series de Disney
Esta semana salieron 13.0 series de Betflix
-----
Series acumuladas faltantes por ver de las semanas pasadas de Disney: 209.5
Series acumuladas faltantes por ver de las semanas pasadas de Betflix: 233.0
-----
Series acumuladas por ver Disney: 224.5
Series acumuladas por ver Netflix: 246.0
-----
El profesor 1 vio 0.5 series de Disney
El profesor 2 vio 0.5 series de Disney
El profesor 3 vio 1.0 series de Disney
El profesor 4 vio 1.5 series de Disney
El profesor 5 vio 1.5 series de Disney
El profesor 6 vio 1.0 series de Disney
-----
El profesor 1 vio 1.0 series de Betflix
El profesor 2 vio 2.0 series de Betflix
El profesor 3 vio 2.0 series de Betflix
El profesor 4 vio 1.5 series de Betflix
El profesor 5 vio 2.0 series de Betflix
El profesor 6 vio 0.5 series de Betflix
-----
Los profesores vieron 6.0 series de Disney
Los profesores vieron 9.0 series de Betflix
-----
Quedan 218.5 series de totales de Disney por ver
Quedan 237.0 series de totales de Betflix por ver

```

Figura 15: Ejecución de la solución principal: Semana 48.

3.2 Deadlock y Livelock

A diferencia de la solución principal, al haber eliminado los semáforos, ya no aparecerán las advertencias previas. En la Figura 16 se muestra la compilación de los códigos modificados.

```

> gcc -o DyL DyL.c

```

Figura 16: Compilación del código de Deadlock y Livelock.

Como se explicó anteriormente, tanto el código para generar el deadlock como el livelock se encuentran en el mismo archivo, por lo que primero será necesario seleccionar cuál de las funciones se utilizará. En este caso, se probará primero el deadlock y luego el livelock. En la Figura 17 se muestra el menú de selección de las demostraciones.

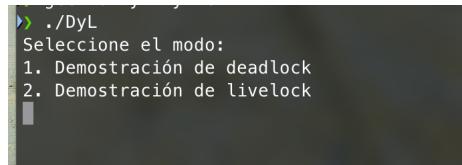


Figura 17: Ejecución del código y selección de función.

3.2.1 Deadlock

Al seleccionar la función de deadlock, se solicitará el tiempo de ejecución al usuario, como en la solución principal. Sin embargo, en este caso, la condición de deadlock se generará durante la primera iteración del ciclo ‘while’ que representa las semanas, por lo que la selección de tiempo no afecta su ejecución. En la Figura 18 se muestra la ejecución de la función de deadlock.

```
./DyL
Seleccione el modo:
1. Demostración de deadlock
2. Demostración de livelock
1
Ingrrese el tiempo:
[1] 1 Mes
[2] 6 Mes
[3] 1 Año
1
Tiempo: 4
-----
Semana 1
-----
Esta semana salieron 10.0 series de Disney
Esta semana salieron 10.0 series de Netflix
Profesor 1 intentando adquirir recursos para deadlock...
Profesor Disney 1 adquirió mutexD
Profesor 2 intentando adquirir recursos para deadlock...
Profesor 6 intentando adquirir recursos para deadlock...
Profesor 3 intentando adquirir recursos para deadlock...
Profesor 4 intentando adquirir recursos para deadlock...
Profesor 11 intentando adquirir recursos para deadlock...
Profesor Netflix 11 adquirió mutexB
Profesor 7 intentando adquirir recursos para deadlock...
Profesor 9 intentando adquirir recursos para deadlock...
Profesor 12 intentando adquirir recursos para deadlock...
Profesor 10 intentando adquirir recursos para deadlock...
Profesor 8 intentando adquirir recursos para deadlock...
Profesor 5 intentando adquirir recursos para deadlock...
```

Figura 18: Ejecución de la función de Deadlock.

3.2.2 Livelock

Al seleccionar la función de livelock, se solicitará también el tiempo de ejecución. Al igual que en el caso anterior, esta selección no influye en el funcionamiento del programa. En la Figura 19 se muestra la ejecución del livelock y el respectivo bucle que caracteriza esta condición.

```

> ./DyL
Seleccione el modo:
1. Demostración de deadlock
2. Demostración de livelock
2
Ingrese el tiempo:
[1] 1 Mes
[2] 6 Mes
[3] 1 Año
1
Tiempo: 4
-----
Semana 1
-----
Esta semana salieron 14.0 series de Dasney
Esta semana salieron 10.0 series de Betflix
Profesor Dasney 1 cediendo a profesores de Betflix...
Profesor Dasney 2 cediendo a profesores de Betflix...
Profesor Dasney 3 cediendo a profesores de Betflix...
Profesor Dasney 4 cediendo a profesores de Betflix...
Profesor Dasney 5 cediendo a profesores de Betflix...
Profesor Dasney 6 cediendo a profesores de Betflix...
Profesor Betflix 7 cediendo a profesores de Dasney...
Profesor Betflix 8 cediendo a profesores de Dasney...
Profesor Betflix 9 cediendo a profesores de Dasney...
Profesor Betflix 10 cediendo a profesores de Dasney...
Profesor Betflix 11 cediendo a profesores de Dasney...
Profesor Betflix 12 cediendo a profesores de Dasney...
Profesor Dasney 1 cediendo a profesores de Betflix...
Profesor Dasney 2 cediendo a profesores de Betflix...
Profesor Dasney 3 cediendo a profesores de Betflix...
Profesor Dasney 4 cediendo a profesores de Betflix...
Profesor Dasney 5 cediendo a profesores de Betflix...
Profesor Betflix 7 cediendo a profesores de Dasney...
Profesor Betflix 9 cediendo a profesores de Dasney...
Profesor Dasney 6 cediendo a profesores de Betflix...
Profesor Betflix 8 cediendo a profesores de Dasney...
Profesor Betflix 10 cediendo a profesores de Dasney...
Profesor Betflix 12 cediendo a profesores de Dasney...
Profesor Betflix 11 cediendo a profesores de Dasney...
Profesor Dasney 6 cediendo a profesores de Betflix...
Profesor Dasney 5 cediendo a profesores de Betflix...
Profesor Betflix 10 cediendo a profesores de Dasney...
Profesor Betflix 11 cediendo a profesores de Dasney...

```

Figura 19: Ejecución de la función de Livelock.

4 Análisis

4.1 Análisis de la solución principal

Se refleja en los resultados de la ejecución cómo los profesores consumen series de dos plataformas ficticias, Dasney y Betflix, con un enfoque en el sistema de acumulación de series pendientes y la distribución del tiempo de visualización. En cada semana, se generan exitosamente nuevas series para cada plataforma, y los profesores intentan consumirlas en función del tiempo disponible, registrado en valores aleatorios asignados a cada profesor. Se observó que, en la mayoría de las semanas, no todos los profesores logran ver la totalidad de las series lanzadas, generando un "saldo pendiente" que se acumula semana tras semana. Este fenómeno se refleja en el crecimiento de las series acumuladas por ver en ambas plataformas, lo cual indica que la capacidad de los profesores para consumir contenido es insuficiente para el ritmo de lanzamientos, resultando en una acumulación de tareas. Este saldo se calcula correctamente y se reporta al inicio de cada semana, destacando el incremento en la carga de visualización. En términos del código, esta simulación es efectiva para ilustrar la gestión de recursos compartidos mediante semáforos y mutex locks, asegurando que los datos de tiempo de visualización y series pendientes se calculen de manera segura y sincronizada entre los hilos. Aunque los números exactos pueden variar en cada ejecución debido a la aleatoriedad, el programa sigue una lógica consistente de asignación de recursos, acumulación y reporte de pendientes, lo que simula de forma coherente un sistema de consumo de contenido con recursos limitados.

4.2 Análisis de las Condiciones del Deadlock

Para que ocurra un deadlock, deben cumplirse las siguientes condiciones:

1. **Exclusión Mutua:** Los mutexes `mutexD` y `mutexB` solo pueden ser poseídos por un hilo a la vez.
2. **Retención y Espera:** Los hilos retienen un mutex mientras esperan por el otro.
3. **No Expropiación:** Los recursos no pueden ser forzadamente retirados de los hilos; deben ser liberados voluntariamente.
4. **Espera Circular:** Existe una cadena circular de hilos donde cada uno espera por un recurso retenido por otro hilo.

El código cumple con todas estas condiciones, provocando deliberadamente un deadlock en el sistema.

4.3 Análisis del Livelock

En esta implementación:

- **Actitud de Cortesía Mutua:** Los hilos intentan evitar conflictos cediendo el paso si detectan que el otro grupo también quiere acceder.
- **Falta de Progreso:** Aunque los hilos están activos y ejecutando código (imprimiendo mensajes), no realizan ningún avance en su tarea principal.
- **Ausencia de Condición de Salida:** Sin una condición que rompa el ciclo (como un contador de intentos o una señal externa), los hilos permanecerán en este estado indefinidamente.

4.3.1 Importancia de las Variables volatile

Las variables `trying_dasney` y `trying_betflix` son declaradas como `volatile` para garantizar que cada hilo lea siempre el valor más reciente de estas variables desde la memoria compartida, evitando optimizaciones del compilador que podrían almacenar valores en caché y romper la sincronización entre hilos.

5 Conclusión

En conclusión, el código desarrollado simula de manera efectiva un sistema de consumo de contenido en un entorno multihilo, donde varios "profesores" acceden a dos plataformas de series con un límite de tiempo semanal y acumulan pendientes si no logran ver todo el contenido lanzado. El uso de semáforos y mutexes en el código permite una gestión segura y sincronizada de los recursos compartidos, evitando condiciones de carrera y asegurando que las operaciones de lectura y escritura sobre las variables de estado sean precisas.

Los resultados de la simulación reflejan cómo, bajo ciertas condiciones, la carga de contenido no consumido puede acumularse de manera constante si la capacidad de los profesores para consumir series es insuficiente frente a la tasa de lanzamiento. Esto ofrece una visión útil de cómo el diseño de sistemas de concurrencia y sincronización puede afectar la eficiencia en entornos donde los recursos o el tiempo de procesamiento son limitados. Además, este ejercicio permite comprender la importancia de manejar de forma cuidadosa los recursos compartidos en sistemas concurrentes, destacando que incluso en situaciones simuladas, la acumulación de "tareas pendientes" puede ser un reflejo de problemas de eficiencia de los recursos, lo cual es fundamental en aplicaciones prácticas de concurrencia.

Tanto en el caso del deadlock como en el del livelock, el código está diseñado para ilustrar problemas clásicos en la programación concurrente:

- **Deadlock:** Muestra cómo la adquisición de recursos en orden inverso por diferentes hilos puede provocar un bloqueo total del sistema, donde ninguno puede avanzar.
- **Livelock:** Demuestra cómo una excesiva deferencia entre hilos puede resultar en una falta de progreso, incluso cuando los hilos están activos.

Estas implementaciones son útiles para comprender la importancia de diseñar protocolos de sincronización cuidadosos que eviten estos problemas y permitan que los hilos realicen su trabajo de manera eficiente.

6 Anexos