udp Escuela de Informática y Telecomunicaciones

# "Tarea 2: Sistemas Operativos"
## Sistema COVID-F

Integrantes:
Bastián Castro
Joaquín Fernández

Profesor: Martín Gutiérrez
Ayudante: Yerko Ortíz

28 de mayo del 2021

# Índice

# 1. Introducción

En esta experiencia se debe desarrollar un entorno pasarela a partir del enunciado entregado. El que consta con las restricciones asociadas a 3 entidades fundamentales; Tales son Carabineros (N), Tribunal (M) y Capacidad máxima del sistema (K). De esta forma se busca generar una simulación de los pasos de productor y consumidor; Similar a la fábula del Guaton Parrillero. Por lo tanto los Carabineros realizan la detención de transeúntes que no cumplan con la cuarentena asociada al COVID, estos últimos son procesados en tribunales y la idea es que los tribunales no repitan al mismo individuo en más de uno; Puesto que un tribunal debe de supervisar a una única persona y así mismo si es que resulta culpable, debe de ser trasladado a un centro de formalización y debe de pasar un tiempo de espera para que pueda ser reingresado a la sociedad. Concluyendo así con un sistema de concurrencia de procesos incluyendo herramientas para la generación del espacio tales son threads, semáforos, mutex y monitor si es que llega a ser necesario dependiendo de la solución implementada.

# 2. Desarrollo

## 2.1. Enunciado

COVID del ass nos tiene a todos en jaque:Muchas de as personas confinadas no pueden salir a trabajar,pero salen igual por necesidad, mientras otras salen de juerga. Son estas últimas las que nos interesa identificar para mandarlas a ser procesadas, multadas y puedan tener juerga (en la cárcel) hasta el final de los tiempos.

Son Uds., dignos alumnos de Martín Gutiérrez (ahem...ahem!!! :P) a quienes ha buscado la Subsecretaría de Persecución de Giles (SPG, suena asigla seria) para implementar un sistema de pasarela que conecte a las distintas entidades de Carabineros que retienen a los giles juergueros hacia su proceso judicial en los tribunales.

Dicho sistema de pasarela funciona de la siguiente forma:

1. Cada entidad de Carabineros (hay N) indica los y datos de las personas siendo retenidas y amonestadas.
2. Un tribunal (de entre M) procesa y formaliza (nunca he entendido este verbo, pero bueno...) a los acusados.
3. El sistema aguanta un máximo de K procesados, pero no debe perder ninguno de ellos durante su funcionamiento.
4. $K < M < N$

## 2.2. Herramientas utilizadas

**Thread:**

Es una secuencia de tareas encadenadas muy pequeñas que puede ser ejecutada por un sistema operativo. En este caso uniendo procesos y vinculándolos dependiendo del espacio aplicado.

**Mutex:**

Es una bandera o flag mutuamente exclusiva. Actúa como un guardián o como filtro en una sección de código que permite el acceso de un thread y bloquea el paso de los demás. Esto asegura que el código que se esta controlando o supervisando será solo afectado por un único hilo a la vez.

**Semáforos:**

Es una estructura diseñada para sincronizar dos o más threads o procesos, de modo que su ejecución se realice de forma ordenada y sin conflictos entre ellos.

## 2.3. Análisis

Respecto al enunciado del problema se puede identificar la fábula asociada a Productor Consumidor en donde se ha de contextualizar 2 relaciones.

Productor-Consumidor referente a Carabineros procesando a N retenidos y M tribunales consumiendo a las personas retenidas.

Productor-Consumidor entre Tribunal, produciendo M procesados y el centro de formación consumiendo K procesados.

Se aprovechó la instancia de usar arreglos para el papel de Buffers juntos a contadores para simplificar la dificultad asociada al algoritmo. A la vez se empleó un semáforo que controla o hace gestión de la cantidad de personas que son procesadas por el tribunal en caso del script usado como solución, condenadas. Por otro lado el uso de mutex fue necesario en la solución para crear variaciones en la cantidad de personas procesadas por Carabineros y así mismo en Tribunales. Para Carabineros fue de suma importancia usar mutex puesto que al no usarlos se podría generar posibles deadlocks y así mismo ocasionar concurrencia con otros procesos que involucren threads tal es el caso de tribunal que debe de asignar una condena o no a un único individuo y no más de uno al mismo tiempo; También se puede ver al momento de ver la capacidad máxima del centro de formación, ya que si es que llega a tener un límite de 2 personas y hay mas esperando, las que se encuentran dentro deben de cumplir con su respectiva condena y así los nuevos reclusos podrán usar los cupos disponibles.

### 2.3.1. Dificultades encontradas

**Deadlocks:**

Primero que todo ¿Qué es un "Deadlock"?

Cuando se trabaja en programación paralela o multisubproceso. Por lo que es una situación difícil de reproducir, y se dará solo en algunas circunstancias muy concretas. Tal es el caso en la actividad donde 2 o más threads esperan un recurso u otro thread que nunca les va a ser otorgado.

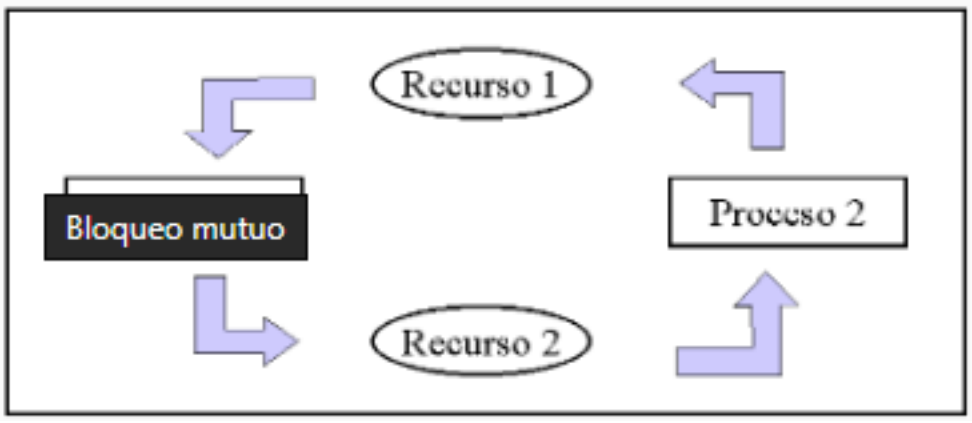


Figura 1: Representación grafica Deadlock

**Deadlocks Econtrados:**

```
void* juhgao()
{
    boolean seguir = T;
    while(seguir)
    {
        pthread_mutex_lock(&mutex);
        if(personas > 0)
        {
            personas--;
            pthread_mutex_unlock(&mutex); //
            if(GenerateRandom() > 0.5)
            {
                printf("Soy el paopa de jorge nos vemos en tribunales (===)\n");
                condenar();
            }
            else
            {
                printf("no era kripi era oregano rey (===)\n");
                pthread_mutex_lock(&mutex2);
                libre++;
                pthread_mutex_unlock(&mutex2);
            }
        }
        /*else
        {
            pthread_mutex_unlock(&mutex);
        }*/
        if((condenao+libre) >= n_personas)
        {
            seguir = F;
        }
    }
    pthread_exit(NULL);
}
```

```
joaquin@joaquin-Aspire-E5-575G:~/Operativos/tarea2$ gcc tarea2b.c -o tarea2b -lpthread
joaquin@joaquin-Aspire-E5-575G:~/Operativos/tarea2$ ./tarea2b
Cantidad de personas que seran atrapadas: 15
Ingrese cantidad de pacos que llevan personas al tribunal: 8
Ingrese cantidad de tribunales: 6
Maximo de procesados: 3
```

Figura 2: Resultado de Deadlock con mutex unlock comentado en juhgao

Un posible caso visible de un Deadlock es al momento de comentar en la función “juhgao” el “else”; siendo este último, el cual se encarga de desbloquear el mutex que ayuda a coordinar a los carabineros y los threads de los tribunales, teniendo un bloqueo permanente al momento de que la condición del “if” no se cumpla, así todos los threads quedarán esperando a que se libere el mutex por el que entró anteriormente, a la zona crítica.

Otro deadlock ocurrido fue al momento antes de implementar el mutex en la función “threadfunction”; puesto que no se tenía el contador de personas atrapadas actualmente; Por lo que al estar fuera del mutex, no coincide con las personas que se habían entrado a la cola, teniendo valores más altos que el

número de individuos que ingresaron a la "cola", haciendo que los carabineros no atrapen a la cantidad de personas de forma exacta y el tribunal no tendrá los procesados que necesita para terminar su ciclo while.

## 2.4. Script Actividades

### 2.4.1. Variables Importantes:

Lo primero dentro de la solución es designar variables de entorno para el desarrollo de la actividad usando threads. Para este caso, se tiene la siguiente instancia de parámetros:

```
int n_personas;
int n,m,k;
int libre, condenao;
int atrapados=0;
int personas=0;
sem_t semaforo1;
pthread_mutex_t mutex, mutex2;
```

Se tienes el número de personas que es la cantidad de individuos procesados por Carabineros, n es la cantidad de carabineros que trasladan a los detenidos a tribunales, en donde la cantidad de tribunales que designan la condena es m; También la capacidad máxima perteneciente al número de centro de formación es k. La variable personas es un contador que actúa como buffer y personas es una cifra cambiante que asemeja a la cantidad de personas actuales detenidas por carabineros.

"libre" es la suma de personas que en tribunal no se le designó una condena por lo que no fueron movilizadas a comisaria; Caso contrario de a "condenao" que corresponde a la multitud con la característica de haber sido llevado a comisaría.

"semaforo1" haciendo mención de su nombre es un objeto de tipo semáforo que se encarga regular o filtrar la cantidad k en carcel de sujetos. A través de variables threads con comportamiento mutex tal es el caso de "mutex2" y "mutex" realiza un trabajo similar en tribunal.

### 2.4.2. Creación de Threads:

```
    //============================tribunales==============================
    pthread_t tribunales[m];
    for (int i = 0; i < m; i++)
    {
        pthread_create(&tribunales[i],NULL,juhgao,NULL);
    }
    //============================pacos
    pthread_t Pacos[n]; //
    for (int i = 0; i < n; i++)
    {
        pthread_create(&Pacos[i],NULL,threadFunction,NULL);
    }
```

Este tramo se encarga de crear los threads del main asociados a Carabineros y Tribunales, estos mismos ubicados en estructuras "array" con el papel de Buffers. Cada uno dependiendo de un límite asociado por el input entregado al momento que el programa este en ejecución. A cada uno se les vincula con una dirección de memoria para luego que esta sea el detonante para desarrollar los cambios a futuro según las variaciones efectuadas por el manejo de threads.

### 2.4.3. Función para Carabineros

```
void* threadFunction()
{
    while((atrapados) < n_personas)
    {
        pthread_mutex_lock(&mutex);
        if(atrapados < n_personas)
        {
            atrapados++;
            personas++;
            printf("Martin lo pillaron jugando Mundial Ronaldinho Soccer 64 en la plaza\n");
        }
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}
```

En este caso se ve la función, en la cual trabajan los N threads que corresponden a los carabineros, teniendo lo que es un while para mantener a los carabineros trabajando hasta atrapar a la cantidad indicada de n_personas.

Cuando los carabineros empiezan a encontrar gente los llevan a una "cola" hacia el tribunal, que en este caso se representa como un contador.

Lo importante es que los carabineros estén sincronizados para anotar a la gente e ingresarla en la cola y así no tener problemas de inconsistencia; para solucionar esto se hizo uso de un mutex, el cuál actúa al momento de preguntar cuantos atrapados hay, y aumentar la cantidad en la cola para el tribunal, siendo ese el instante en que el thread entra nuestra zona crítica, cabe destacar que el tribunal igual intentará tomar personas y este mutex funcionará para los dos, como se verá a continuación.

### 2.4.4. Función de Tribunales

```
void* juhgao()
{
    boolean seguir = T;
    while(seguir)
    {
        pthread_mutex_lock(&mutex);
        if(personas > 0)
        {
            personas--;
            pthread_mutex_unlock(&mutex); //
            if(GenerateRandom() > 0.5)
            {
                printf("Soy el paopa de jorge nos vemos en tribunales (===)\n");
                condenar();
            }
            else
            {
                printf("no era kripi era oregano rey (===)\n");
                pthread_mutex_lock(&mutex2);
                libre++;
                pthread_mutex_unlock(&mutex2);
```

```
            }
        }
        else
        {
            pthread_mutex_unlock(&mutex);
        }
        if((condenao+libre) >= n_personas)
        {
            seguir = F;
        }
    }
    pthread_exit(NULL);
}
```

Esta función hace que los tribunales actúen de una forma parecida a los carabineros teniendo que trabajar hasta que se acaben las personas, en este caso, estos funcionan con un contador para las personas que son "condenadas" y otro para las personas que quedan "libres".

El uso de mutex en este caso tiene el mismo propósito pudiendo coordinar los threads en los tribunales y carabineros, se debe usar antes de verificar que hayan personas en la cola y en el momento que exista por lo menos una entidad se restará y el thread volverá a habilitar el paso por ese mutex, puesto que, este dejó de hacer uso de la cola y ya sacó a la persona que va a procesar, por otro lado cuando no hayan personas dentro del contador se abrirá el mutex de nuevo para no generar un deadlock.

Lo siguiente que queda es determinar si la persona es libre o se le condena, para esto se utilizó una función "Random" que genera un numero entre 0 y 1 para así tomar esta difícil decisión.
Cuando se decide dejar libre a la persona, esta se registra en el contador al igual que en el otro caso en que queda condenada y se deba reformar usando la función "condenar", teniendo en cuenta esto cabe mencionar que nuestros contadores deben ser exactos, ya que, al momento de que las personas dejen de llegar, los tribunales quedarán esperando hasta que haya alguien nuevo en la cola, pero en realidad puede que esta entidad se haya perdido en los registros y esto termina en un ciclo infinito para los tribunales.

### 2.4.5. Función Condenar:

```
void* condenar()
{
    sem_wait(&semaforo1);
    pthread_mutex_lock(&mutex2);
    condenao++;
    pthread_mutex_unlock(&mutex2);

    printf("Entro a proceso \n");

    printf("Eso no era cilantro, era peregil 10 anios de carcel para ti\n");
    sleep(1); //10 a os en 1 segundo
    printf("fin del proceso\n");

    sem_post(&semaforo1);
}
```

El semáforo aquí se encarga de colar o dejar pasar únicamente a un único thread o así mismo la cantidad de threads que usen el espacio librado por los reclusos, ya con su condena cumplida; De esta forma se tiene un contador que verifica la llegada de nuevos detenidos y así mismo nivelar el límite asociado al presidio y así el semáforo vuelve a permitir la realización de este proceso; A través de un mutex que gestiona el control de flujo por el semáforo valga la redundancia.

# 3. Conclusiones

Para abordar esta actividad en detalle, se desarrollo una solución en base a la fábula de Productor-Consumidor, esta misma utiliza herramientas de contadores, mutex, y semáforos para así poder evitar condiciones de carrera y deadlock. Esta problemática llega a ser muy recurrente al trabajar con multithreading y también considera el uso de recursos del ordenador. De igual forma se presentaron estas dificultades, ya que al no ubicar un thread o así mismo ubicar un mutex o mal empleamiento de la estructura semáforo producía el conflictivo deadlock.

También se debe considerar el aspecto que promueve esta programa que es el conocimiento en campos teóricos y prácticos de impedimentos al momento de trabajar con multithreading. Si bien se generan posibles deadlocks esta tarea dio la posibilidad de identificar posiblemente en donde y que momento, debiesen de crearse puntos muertos o deadlocks. Evidentemente es posible optimizar o crear una mejor versión del código propuesto como solución así mismo el uso de otros posibles lenguajes que incluyen ya estructuras para abordar de forma más sencilla y eficiente el problema. Y de esta forma facilitar futuros trabajos elaborados a partir de aspectos vistos en cátedra y ayudantía.

# 4. Bibliografía:


- CodeVault, Thread Course https://www.youtube.com/playlist?list=PLfqABt5AS4FmuQf70psXrsMLEDQXNkLq2
- CodingBison http://codingbison.com/c/c-pthreads-deadlocks.html