TEC | Tecnológico de Costa Rica

# Principios de sistemas operativos

## IC-6600

Proyecto #2

Profesora:

Erika Marín Shumann

Estudiantes:

Gloriana Fernández Quesada

José Fabio Hidalgo Rodríguez

Juan Carlos Ruiz Sing

Entrega: 2 de diciembre, 2020

# Índice

## Semáforos

Para la elaboración de esta tarea programada utilizamos dos tipos semáforos distintos, varios mutex y un semáforo global que compartían los procesos. Los mutex se encargaban de permitir que solo un hilo entrase a la zona crítica (memoria compartida). Estos mutex se utilizaron en los writers y en los readers egoístas ya que, según el enunciado del proyecto, solo puede haber un hilo de writer/reader egoísta en la memoria a la vez. Por otro lado, un reader permite que otros readers entren a memoria cuando él está leyendo (no son mutuamente excluyentes entre ellos).

El semáforo global se encarga de restringir el acceso a la zona crítica a solo un tipo de proceso a la vez, es decir, entra un writer, entra un reader egoísta o entran los readers, pero no una combinación de todo. Esto es porque los procesos son mutuamente excluyentes entre ellos.

## Sincronización

La sincronización se logró mediante el uso de semáforos y se estableció que la zona crítica sería la memoria compartida ya que aquí se realizarán las escrituras, lecturas y borrados. Como se mencionó en el apartado anterior, se usó un semáforo global para que le de permisos de entrar a la zona crítica a un solo tipo de proceso a la vez. Con esto se logró la sincronización interproceso, cada vez que entra un proceso, decrementa el contador (lo cual hace que ningún otro tipo de proceso pueda entrar a la zona crítica), hace lo que le corresponde e incrementa el contador. Al incrementar el contador, se le dará permiso a uno de los otros procesos que están compitiendo por entrar en memoria.

Por otro lado, el mutex se encarga de que solo un hilo de writer y uno de reader egoísta estén compitiendo por el semáforo. Este semáforo más que todo está como medida de precaución, ya que en teoría el semáforo global se encargaría de solo dejar pasar a un solo writer o reader egoísta, pero se optó por utilizar el mutex para que solo hubiese un hilo de cada uno en espera del semáforo global.

Con el uso de estos semáforos, nos aseguramos que la zona crítica es mutuamente excluyente y aseguramos la sincronización entre procesos. Además, no se pueden generar deadlocks ya que, en la ejecución de la zona crítica, no se requieren de otros recursos que se tengan que esperar, con lo cual no hay esperas circulares ni ninguna otra situación que pueda generar un deadlock.

## Mmap

Shmget es un comando utilizado para alocar un segmento de memoria compartido con base al System V. (Kerrisk, 2020). Por su parte mmap permite realizar un mapeo de archivos o dispositivos a la memoria y emplea el sistema POSIX, por ende es más reciente. Por su parte shmget es más antiguo y a cambio posee mayor uso y soporte.

Shmget utiliza una llave única creada a base de un path y número y además el tamaño del segmento de memoria que se desea crear, por su parte mmap no requiere de una llave pero sí requiere del tamaño y de la dirección de memoria que se ocupa. (Kerrisk, 2020).

Mmap es más sencillo de usar pero posee mayores restricciones mientras que shmget, como requiere de una llave única, pueden haber conflictos con llaves que ya existen y en ambos casos, tanto mmap como shmget son thread-safe.(Kerrisk, 2020).

## Análisis de resultados

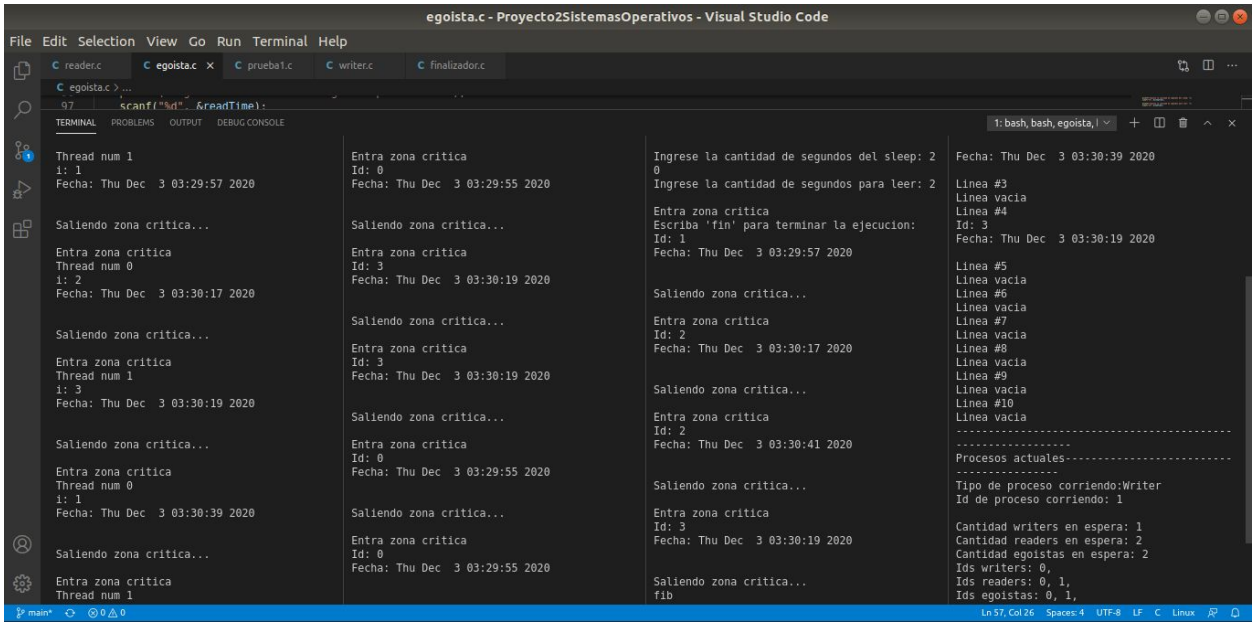
En cuanto al estado de los programas, el inicializador, el writer, el reader, el reader egoísta y el finalizador están completos y correctos. El programa espía es capaz de mostrar el programa con el control de la memoria en un momento dado (junto con su id), además de la cantidad y el id de los procesos en espera (diferenciando entre writers, readers y readers egoístas) y todas las líneas de la memoria. Sin embargo, el espía no logra diferenciar entre procesos bloqueados y dormidos, ya que esto requiere un uso de arreglos, lo cual se complica a la hora de manejar memoria compartida en tiempo de ejecución.

Entre las decisiones de diseño que se tomaron, se incluye que para simular correctamente el funcionamiento de estos programas, se requiere tener un número grande de segundos para el sleep, dado que si fuese un sleep más pequeño, es difícil de visualizar el flujo del programa. Como se detalló anteriormente, los semáforos son de 2 tipos (mutex para utilizarlo de manera local en los writers y semáforos no binarios para la sincronización global). La memoria compartida consiste en una serie de structs (message e information) de donde lee cada programa.

Casos de prueba:

Writers: 2, 20s de sleep, 2s de escritura
Readers: 2, 10s de sleep, 2s de lectura
Egoístas: 2, 20s de sleep, 2s de lectura

Resultados:



(Cada columna corresponde a una terminal corriendo writers, readers, readers egoístas y espía)

## Manual de usuario

Para compilar los archivos del proyecto, se deben ejecutar los siguientes comandos en la consola dentro del directorio del proyecto (no es necesario seguir un orden específico):

- gcc inicializador.c -o inicializador
- gcc writer.c -o writer -lpthread -lrt
- gcc reader.c -o reader -lpthread -lrt
- gcc egoista.c -o egoista -lpthread -lrt
- gcc espia.c -o espia
- gcc finalizador.c -o finalizador

Para la ejecución del proyecto, se deben seguir los siguientes pasos:

1. Ejecutar ./inicializador en la terminal
2. Indicar la cantidad de líneas que se quieren abrir para la memoria compartida
3. Ejecutar ./writer en la terminal
4. Indicar la cantidad de writers que se quieren crear, la cantidad de segundos del sleep para cada uno, así como la cantidad de segundos de escritura

5. Ejecutar ./reader en la terminal
6. Indicar la cantidad de reader que se quieren crear, la cantidad de segundos de sleep para cada uno, así como la cantidad de segundos de lectura
7. Ejecutar ./egoista en la terminal
8. Indicar la cantidad de readers que se quieren crear, la cantidad de segundos de sleep para cada uno, así como la cantidad de segundos de lectura
9. Ejecutar ./espia en la terminal en cualquier momento de la ejecución para ver las líneas de la memoria y la cantidad de programas y sus estados respectivos
10. Para terminar la ejecución del programa de writers, readers o readers egoístas, en su terminal correspondiente se puede ejecutar el comando “fin”
11. Para liberar la memoria y destruir los semáforos, ejecutar ./finalizador en la terminal

## Bitácora de trabajo

13/11/2020: Reunión para aclarar detalles del plan y diseño del proyecto
14/11/2020: Reunión para probar detalles de creación de segmentos de memoria.
18/11/2020: Reunión para definir las partes necesarias de la implementación.
21/11/2020: Desarrollo del inicializador y el writer.
22/11/2020: Desarrollo del reader y finalizador.
23/11/2020: Pulir detalles del reader y writer.
25/11/2020: Primera implementación de los semáforos.
27/11/2020: Creación del egoísta.
28/11/2020: Reader, writer y egoísta listos.
29/11/2020: Refinar detalles de los semáforos.
30/11/2020: Refinar detalles de los semáforos, dado a un error que se encontró.
1/12/2020: Semáforos y bitácora listos.
2/12/2020: Espía listo y detalles completados.

## Bibliografía

Kerrisk, M (2020-11-01) The Linux Programming Interface. Linux man-pages project. Recuperado de: https://man7.org/linux/man-pages/man2/shmget.2.html

Kerrisk, M (2020-11-01) The Linux Programming Interface. Linux man-pages project. Recuperado de: https://man7.org/linux/man-pages/man2/mmap.2.html