Tecnológico de Costa Rica

Ing. En Computación - Principios de Sistemas Operativos

Profesora:
Erika Marín Schumann

Estudiantes:
Carlos Girón Alas 2014113159
Julián J. Méndez Oconitrillo 2014121700
Jasson Moya Álvarez 2014082335

Sincronizador de Procesos

Fecha de entrega: Martes 25 de octubre

Tabla de Contenidos:

1. Manual de compilación _____ 3
- **1.1** Para correr los ejecutables _____ 3
- **1.2** ¿Cómo compilar los archivos ejecutables? _____ 3
- **1.3** Sobre la ejecución. _____ 3
  - **1.** Starter_____ 3
  - **2.** Writer _____ 3
  - **3.** Reader _____ 3
  - **4.** Selfish Reader_____ 3
  - **5.** Spy_____ 3
  - **6.** Finisher_____ 4

2. Casos de prueba _____ 5
- **2.1** Starter _____ 5
- **2.2** Writer _____ 5
- **2.3** Reader _____ 6
- **2.4** Selfish Reader _____ 6
- **2.5** Spy _____ 7
- **2.6** Finisher _____ 7

3. Análisis de resultados _____ 9

4. Información extra _____ 10
- **4.1** ¿Qué tipo de semáforos se utilizó y por qué? _____ 10
- **4.2** ¿Cómo se logró la sincronización? _____ 10
- **4.3** ¿Qué hubiera sido diferente si hubiera usado mmap? _____ 10

1. Manual de compilación

Es importante tener en cuenta que este proyecto fue desarrollado en Linux, Ubuntu 16.04 en lenguaje C. Los pasos para la compilación se muestran a continuación. Adjunto a la carpeta está el archivo makefile que tiene las instrucciones del compilador para que se generen los archivos .o y ejecutable. Se tienen seis programas en total que se comunican mediante la técnica de memoria compartida.

1.1 Para correr los ejecutables:

1. Abra el terminal montando la carpeta en la que se encuentran los 6 archivos. Repita este proceso 6 veces, para poner a correr los seis programas.
2. Para ejecutar el inicializador, corra *./starter* en una terminal.
3. Para ejecutar el escritor, corra *./writer* en otra terminal.
4. Para ejecutar el lector, utilice el comando *./reader.*
5. Para ejecutar el lector egoísta, utilice el comando *./selfish_reader.*
6. Para ejecutar el espía, utilice el comando *./spy.*
7. Para ejecutar el programa final, utilice el comando *./finisher.*
8. La forma más rápida de montar las carpetas en este proyecto es dar click derecho en la carpeta donde se encuentran los archivos y elegir Open in terminal o Abrir en terminal.

1.2 ¿Cómo compilar los archivos ejecutables?

El proyecto se desarrolló haciendo uso del archivo makefile adjunto.

1. Para el *inicializador* se corre *make starter*
2. Para el *escritor* se corre *make writer*
3. Para el *lector* se corre *make reader*
4. Para el *lector egoísta* se corre *make selfish_reader*
5. Para el *espía,* se corre *make spy*
6. Para el *programa final,* se corre *make finisher*

1.3 Sobre la ejecución.

Cada uno de los archivos o se ejecuta de manera automática, o presenta opciones que se describen a continuación:

1. **Starter:** Se crean todo los espacios de memoria compartida
2. **Writer:** Recibe de parámetro *Amount_Of_Writers, Write_Time, Sleep_time.* Crea los escritores y empieza a escribir el archivo físico y compartido.
3. **Reader:** Recibe de parámetro *Amount_Of_Readers, Read_Time, Sleep_Time.* Crea los lectores y empieza a leer el archivo físico y compartido.
4. **Selfish_reader:** Recibe de parámetro *Amount_Of_Readers, Read_Time, Sleep_Time*. Crea los lectores egoístas, y empieza su proceso de elección aleatoria.
5. **Spy:** Muestra las diferentes opciones disponibles para observar procesos y archivos. Las opciones son las siguiente:
    a. Observar el archivo actual
    b. Observar los escritores
    c. Observar los lectores

d. Observar los lectores egoístas
e. Salir

6. **Finisher:** Termina el programa y elimina los campos de memoria compartida que creó el starter.

2. Casos de prueba

2.1 Starter

Caso de prueba	Comportamiento esperado.	Se cumple
1. Creación de memoria compartida	El programa del starter crea con éxito 10 espacios de memoria.	Sí
2. Error en la creación de memoria compartida	Se muestra un error a la hora de crear los espacios de memoria compartida.	Sí
3. Creación de archivos iniciales	Los archivos “resultados.txt” y “log.txt” se crean correctamente.	Sí
4. Error al crear archivos iniciales	Se muestra un error al crear los archivos iniciales.	Sí

2.2 Writer

Caso de prueba	Comportamiento esperado.	Se cumple
1. Creación de enlace con memoria compartida	Se enlaza correctamente el programa con los espacios de memoria creados	Sí
2. Introducción de parámetros erróneos	Se muestra un error al usuario cuando no introduce bien los parámetros.	Sí
3. Creación de hilos para cada writer	Se crean los hilos para cada writer.	Sí
4. Escritura en archivo local	Se escribe correctamente en el archivo local	Sí
5. Escritura en memoria compartida	Se escribe correctamente en la memoria compartida	Sí
6. Writer espera su turno	El writer deja de escribir mientras duerme	Sí
7. Writer se reactiva	El writer en descanso se reactiva luego de su descanso.	Sí
8. Se bloquea el writer	El writer es bloqueado por otro	Sí
9. Se libera el writer	El writer es liberado luego de que un proceso lo estuviera bloqueando	Sí

2.3 Reader

Caso de prueba	Comportamiento esperado.	Se cumple
1. Creación de enlace con memoria compartida	Se enlaza correctamente el programa con los espacios de memoria creados	Sí
2. Introducción de parámetros erróneos	Se muestra un error al usuario cuando no introduce bien los parámetros.	Sí
3. Creación de hilos para cada reader	Se crean los hilos para cada reader.	Sí (80%)
4. Lectura de archivo local	Se lee correctamente el archivo local, con una y varios readers a la vez	Sí
5. Lectura de memoria compartida	Se lee correctamente en la memoria compartida, con uno y varios readers a la vez	Sí (80%)
6. Reader espera su turno	El reader deja de leer mientras duerme	Sí
7. Reader se reactiva	El reader en descanso se reactiva luego de su descanso.	Sí
8. Se bloquea el reader	El reader es bloqueado por otro	Sí
9. Se libera el reader	El reader se libera luego de que otro proceso lo tuviera bloqueado	Sí

2.4 Selfish Reader

Caso de prueba	Comportamiento esperado.	Se cumple
1. Creación de enlace con memoria compartida	Se enlaza correctamente el programa con los espacios de memoria creados	Sí
2. Introducción de parámetros erróneos	Se muestra un error al usuario cuando no introduce bien los parámetros.	Sí
3. Creación de hilos para cada selfish reader	Se crean los hilos para cada selfish reader.	Sí
4. Lectura de archivo local	Se lee correctamente el archivo local	Sí
5. Lectura de memoria	Se lee correctamente en la memoria	Sí

compartida	compartida	
6. Selfish reader espera su turno	El selfish reader deja de leer mientras duerme	Sí
7. Selfish reader se reactiva	El selfish reader en descanso se reactiva luego de su descanso.	Sí
8. Se bloquea el selfish reader	El selfish reader es bloqueado por otro	Sí
9. Se libera el selfish reader	El selfish reader se libera luego de que otro proceso lo tuviera bloqueado	Sí
10. Ningún otro proceso puede leer o escribir mientras esté el selfish reader	Los procesos de lectura y escritura no pueden hacer nada mientras el egoísta esté activo.	Sí

2.5 Spy

Caso de prueba	**Comportamiento esperado.**	**Se cumple**
1. Creación de enlace con memoria compartida	Se enlaza correctamente el programa con los espacios de memoria creados.	Sí
2. Introducción de opciones inválidas	Se muestra un error al usuario cuando no introduce alguna de las opciones válidas.	Sí
3. Se muestra el archivo actual en determinado momento	El archivo de texto actual se imprime en la consola correctamente.	Sí
4. Se muestran los writers actuales	Todos los writers se muestran y se muestra también su estado en el momento.	Sí
5. Se muestran los readers actuales	Todos los readers se muestran y se muestra también su estado en el momento.	Sí
6. Se muestran los selfish readers actuales	Todos los selfish readers se muestran y se muestra también su estado en el momento.	Sí

2.6 Finisher

Caso de prueba	**Comportamiento esperado.**	**Se cumple**
1. Finaliza la memoria	Todos los procesos y la memoria compartida	Sí

compartida	es borrada y el programa finaliza	

3. Análisis de resultados

Objetivo	Éxito
1. Los 6 programas inician correctamente y ejecutan sus funciones siempre que el inicializador haya corrido antes que los otros 5.	100%
2. Los 6 programas despliegan error cuando alguna de las operaciones de creación de archivos o separación de espacios de memoria.	100%
3. Los 6 programas despliegan errores cuando no se introducen los parámetros adecuados (cantidad de parámetros y tipo de parámetros).	100%
4. El inicializador crea todos los espacios de memoria que se esperan usar para la escritura de acuerdo al parámetro introducido.	100%
5. El programa reader genera la cantidad de hilos introducidos por el parámetro y comienzan sus procesos de lectura cuando se reconoce en las variables de validación que hay líneas para ser escritas.	85%
6. El programa writer crea la cantidad de hilos con proceso de lectura de acuerdo a los parámetros y ejecutan sus funciones cuando se cumplen las condiciones de que los archivos compartidos están disponibles y no hay otro proceso ocupando los archivos.	100%
7. El programa lector egoísta o selfish_reader crear la cantidad de lectores egoístas de acuerdo a los parámetros y se bloquean de acuerdo al límite 3 para los lectores egoístas que puede usar los archivos.	100%
8. El programa espía busca los datos que se le piden en el menú de acuerdo a la especificación: cantidad de lectores/escritores/lectores egoístas, etc.	100%
9. El programa finalizador elimina todos los espacios de memoria compartida así como el archivo de resultados que se usó de forma análoga a la memoria compartida.	100%
10. Todos los programas que leen o escriben en memoria compartida escriben en el archivo de bitácora lo que hicieron, en el momento que lo hicieron.	100%
11. La sincronización de procesos se hace correctamente con la implementación de semáforos y esperas cuando un recurso compartido está siendo utilizado.	100%
12. Los programas de escritura, escritura egoísta y lectura no se detienen automáticamente, sino que deben ser cancelados, ya sea por el usuario o al correr el programa finalizador, que elimina los procesos que están corriendo.	100%

13. Los programas no utilizan ningún protocolo externo de comunicación a parte de la memoria compartida y semáforos.	100%
14. Se implementa la función de librería shm_get para la alocación de memoria compartida y no se usa mmap.	100%

4. Información Extra

4.1 ¿Qué tipo de semáforos se utilizó y por qué?

La mayoría son binarios, entre ellos, los que se usan para el funcionamiento del writer. Por otro lado, también se usan tres semáforos contadores para la administración de los readers al archivo. Se usaron los dos tipos de semáforos porque el problema lo amerita, ya que por una parte, el manejo de los writers se realiza mediante uno a la vez, por lo que con un semáforo binario basta. Pero en el extremo del reader, un semáforo de este tipo queda inservible, ya que pueden existir varios readers leyendo al mismo tiempo una línea, y el control de estos es diferente.

4.2 ¿Cómo se logró la sincronización?

La memoria compartida en linux permite el manejo de segmentos, los cuales sirven para cualquier diferentes valores, como la cantidad de procesos escritores o lectores, o también el estado en el que están, de esta manera todos pueden acceder a la información de todos, y de esta manera crear semáforos mutex, y contadores, los cuales manejan con diferentes funciones (que incluyen la pausa y continuidad de los threads) la sincronización del proyecto.

En general, se implementan semáforos con las funciones sem_wait, y sem_post, de la librería semaphore.h.

4.3 ¿Qué hubiera sido diferente si hubiera usado mmap?

La función mmap() establece un mapeo entre un espacio de dirección de memoria, y un archivo, o un objeto de memoria compartido. Con shmget, deja a múltiples procesos atar un segmento de memoria física a sus espacios de dirección virtual. La principal diferencia, es el uso de archivos para cualquier situación, que incluye al manejo de la cantidad de procesos, de qué tipo son, por dónde van, etc. Con shmget se pudo realizar un manejo más eficiente de la memoria compartida, ya que en diferentes segmentos de la memoria, se fue guardando toda la información necesaria, incluyendo el archivo de texto en un array de chars.

En definitiva, es más sencillo usar shmget para compartir la memoria, ya que da la sensación de estar utilizando la memoria compartida directamente, y no mediante un mapeo de archivos u objetos.