Instituto Tecnológico de Costa Rica
Sede Central de Cartago
Escuela de Ingeniería en Computación
Sistemas Operativos

"Proyecto2 - Memoria Compartida"

Estudiantes: Andrey Mendoza Fernández - 2015082908
Armando López Cordero - 2015125414
Prof: Ing. Erika Marín Schumann
Periodo: 2017 - II Semestre
Fecha de Entrega: 25 de octubre del 2017

Abstract

Nowadays, the shared memory structure is a fundamental part of computer systems. Through it, it's possible the communication between different processes that goes from the synchronization to the sharing of resources between them. In order to be used it's necessary to use semaphores, to restrict access and thus maintain mutual exclusion to avoid errors when performing synchronization. This document shows the results of an implementation of the shared memory between process.

Tabla de Contenidos

Introducción **3**

Objetivos del Proyecto **3**

Sincronización **4**

Diferencias entre mmap y shmget **5**

Análisis de Resultados y Casos de Prueba **5**

- Inicializador 5
- Productor 7
- Espía 9
- Finalizador 10
- Resumen de Completitud de los Requerimientos 11

Tipos de Semáforo Utilizados **12**

- Semáforos Binarios POSIX 12
- Razón de la Elección 13

Bitácora del Trabajo **13**

Compilación y Ejecución del Proyecto **14**

- Compilación 14
- Ejecución 15
  - Inicializador 15
  - Productor 15
  - Espía 16
  - Finalizador 16

Conclusiones **17**

Referencias **17**

Introducción

La estructura de memoria compartida es parte fundamental de los sistemas computacionales de la actualidad. Mediante ella, es posible la comunicación entre distintos procesos que va desde la sincronización hasta el compartimiento de recursos entre los mismos. Para poder utilizarse es necesario hacer uso de semáforos, esto para restringir el acceso y por ende mantener la exclusión mutua para evitar errores a la hora de realizar la sincronización.

Por otro lado, se encuentran dos posibles esquemas para compartir memoria que serían la paginación y la segmentación. Donde las misma presentan ciertas características como el registro base y límite que poseen de la memoria compartida, y en añadidura, el segmento utiliza un tamaño para controlar cuánta memoria se puede acceder desde el registro base que tiene asignado.

No obstante, se considera importante dar un vistazo a lo que se ha desarrollado a lo largo del proyecto asignado en el curso de Sistemas Operativos, el cual como se mencionó anteriormente consiste en implementar un sistema de memoria compartida por medio de la cual varios procesos puedan compartir recursos y comunicarse, se brindará una serie de secciones donde se analiza el trabajo realizado y los resultados obtenidos a lo largo de la ejecución del proyecto.

Objetivos del Proyecto

- Implementar un problema de sincronización.
- Profundizar el conocimiento de Semáforos.
- Conocer algunas funciones de Linux para la sincronización.
- Repasar los conocimientos teóricos de Paginación/Segmentación

Sincronización

A continuación, se hará un análisis del trabajo realizado para determinar cómo se logró llevar a cabo el proceso de sincronización a la hora de acceder a la memoria compartida por medio de la implementación de regiones críticas.

En primera instancia cabe resaltar la utilización de semáforos para restringir el acceso, manteniendo la exclusión mutua en la región crítica, eso sí, sin dejar de lado el progreso continuo si dicha región se encuentra desocupada. Estos semáforos cuentan con un número que representa el estado en el que se encuentra, en este caso si el semáforo posee un cero, indica que se encuentra desocupado o lo que es equivalente a que ningún hilo lo esté utilizando, pero si el número es negativo, indicará que el semáforo se encuentra ocupado, entonces, un proceso que lo solicite quedará en la cola del semáforo esperando a que se libere.

De esta manera es que se logra la sincronización para acceder a la región crítica que es la memoria compartida. Cabe resaltar que existen dos regiones donde los hilos concurrentes deben pedir el semáforo para acceder y es cuando intentan ingresar a la memoria compartida ya sea para escribir o para eliminar el espacio que tienen asignado.

Para finalizar, se implementó el uso de sincronización por parte de semáforos en el programa espía, pues para este se utilizó otra memoria compartida donde los hilos van registrando sus acciones o estados, y desde el espía se accede a esta información en un determinado instante, entonces, para que el programa espía no tuviera conflicto con los hilos que estaban registrando sus estados, se aplicó un segundo semáforo para esa memoria. Igualmente, el bloqueo de la memoria de registro de acciones por parte de procesos, permite evitar la lectura de información “basura”, haciendo que el programa sea más fiable.

Diferencias entre mmap y shmget

A continuación, se presenta un breve lista acerca de las principales diferencias que presenta el método mmap respecto a shmget. Note que estos son métodos utilizados para la creación o petición de memoria compartida.

- Se dice que el método mmap es un poco más restrictivo que el shmget, pero que es más fácil de usar.
- Shmget es el modelo viejo del sistema V de memoria compartida
- Shmget tiene un soporte mucho más amplio y esto está relacionado con el punto anterior.
- La principal diferencia entre los de memoria compartida de sistemas V (shmem) y el mapeo de memoria (mmap), es que el sistema shmem es persistente. a menos que sea eliminado explícitamente por un proceso, se guarda en la memoria y permanece disponible hasta que el sistema se apaga.
- La memoria mmap no es persistente entre las ejecuciones de la aplicación (a menos que esté respaldada por un archivo).

Análisis de Resultados y Casos de Prueba

A continuación, se hará un análisis de resultados sobre el cumplimiento de cada requisito. En cada uno de ellos, se describen algunas tomas de decisiones y/o puntos importantes que debieron tomarse en cuenta a la hora de realizar la implementación. Además, se demostrará su funcionamiento mediante una prueba descrita detalladamente.

Inicializador

Es el programa encargado de solicitar la memoria compartida al sistema y guardar los ID de estos espacios en un archivo para que pueda ser accedido por otros procesos. Para el cumplimiento de todos los requisitos solicitados, se decidió hacer uso de dos espacios de memoria compartida:

- **Información de procesos:** está dividida en bloques que van a contener el ID de los procesos que se encuentran haciendo uso de ciertos recursos. Estos bloques están organizados secuencialmente de la siguiente manera:
    a. Id del único programa que se encuentra haciendo uso de la región crítica en dado momento.
    b. Arreglo de procesos bloqueados, que contiene los procesos que se encuentran esperando que el semáforo sea liberado para poder acceder a la región crítica.
    c. Arreglo de procesos muertos, es decir, los que finalizaron sin poder reservar espacio en la memoria simulada.
    d. Arreglo de procesos finalizados.
    e. Arreglo de procesos en espera, que son los procesos que lograron reservar espacio en la memoria simulada y en este momento están en sleep.
- **Memoria simulada:** almacena la memoria simulada, la cual va a ser accedida por cualquier proceso con el fin de reservar espacio, y por ende, liberar al finalizar el sleep.

Los semáforos no son inicializados por este programa, ya que a la hora de intentar abrirlos desde cualquier proceso, si existe, simplemente se toma el puntero del mismo, de lo contrario, se crea.

Caso de prueba:

Descripción: consistirá en realizar la ejecución del programa inicializador, el cual va a solicitar dos espacios distintos de memoria al sistema operativo. Se ejecutará el programa productor con el fin de verificar que el espacio de memoria sigue disponible aún después de que el inicializador haya finalizado.

Resultado esperado: el ID de los dos espacios de memoria creados será guardado en un archivo de texto dentro de la carpeta “Data” y el

proceso productor podrá ejecutarse con normalidad porque los espacios de memoria necesarios existen.

Resultado obtenido: efectivamente los archivos de texto con el ID de cada espacio de memoria fueron creados. El programa productor pudo funcionar con normalidad.

Productor

Este programa se encarga de crear la cantidad de segmentos o páginas solicitadas en el inicializador por el usuario. Una vez inicializadas las celdas de memoria, se hace un llamado al proceso encargado de generar hilos que van a hacer uso de la memoria simulada situada en el espacio de la memoria compartida.

Existen dos maneras de ejecutar el programa productor para indicar el tipo de memoria que tiene que simular:

- **Modo Paginación:** la memoria simulada contendrá estructuras de tipo “Página”, la cual guarda el número de página lógica, el estado (ocupada, disponible) y el ID del proceso que se encuentra usándola (si fuera el caso).

  Para almacenar memoria, simplemente se hace una verificación de la cantidad de páginas disponibles, que se encuentra disponible en la memoria compartida. Si la cantidad disponible es mayor o igual a la cantidad que el proceso va a utilizar, se hace la reservación de memoria, haciendo un recorrido lineal y asignado páginas conforme se vayan encontrando desocupadas, sin la necesidad de que estas se encuentren juntas.

  Una vez finalizado el tiempo de sleep, se procede a liberar los espacios de memoria, haciendo nuevamente el recorrido lineal de la memoria.
- **Modo Segmentación:** en este caso la memoria simulada almacenará estructuras de tipo “Segmento”, que contendrán el número de

segmento del proceso, registro base, tamaño, estado (disponible, ocupado) y el ID del proceso que se encuentra haciendo uso de la misma.

Para almacenar memoria de tipo segmento, inicialmente se debe verificar que la cantidad de celdas disponibles es mayor o igual a la cantidad de segmentos multiplicado por el tamaño en celdas de cada segmento. Si la primera condición se cumple, se procede a verificar que exista una cantidad igual o mayor de celdas juntas para **cada segmento**, sin importar si los segmentos quedan juntos o no. Finalmente, si todo el flujo anterior es correcto, se procede a realizar el almacenamiento respectivo leyendo la memoria simulada de manera secuencial.

Para liberar la memoria, solamente se debe recorrer secuencialmente la memoria y desasignar los segmentos.

Es importante destacar que para ambos tipos de memoria simulada, se decidió almacenar la cantidad de celdas totales y las disponibles con el fin de hacer una primera verificación de costo barato para disminuir el overhead.

Para poder llevar un control de los procesos y poder espiarlos con el programa espía, en cada operación también se accede a la memoria compartida de procesos y se guarda en su respectivo estado. Los estados se encuentran descritos en el inicializador.

Caso de prueba:

Descripción: consistirá en realizar la ejecución del programa inicializador con una memoria de 10 espacios de largo. Se ejecutará el programa productor con el modo de paginación. Esta memoria al ser tan pequeña, provocará casos donde habrá procesos muertos, pero también algunos sí podrán almacenar la memoria, todo depende del resultado del número random generado.

Resultado esperado: Al utilizar el programa espía, se observarán procesos muertos, así como varios registros de fallos en la bitácora, causados por falta de espacios disponibles. También se podrá ver que en determinado momento algunos espacios son ocupados y luego liberados para luego ser ocupados por otro proceso.

Resultado obtenido: efectivamente se logró ver que varios procesos murieron debido a la falta de espacio. También, en la bitácora quedaron los registros de fallas. Al solicitar constantemente al programa espía el estado actual de la memoria, este muestra cambios de ID de procesos para una celda, lo que quiere decir que se está liberando correctamente los espacios.

Espía

El programa espía se encarga de informar al usuario los datos con que el sistema está trabajando en un determinado momento. Esto lo hace por medio del acceso al recurso de memoria compartida, que almacena información de los procesos y por supuesto la memoria principal que vendría a ser la simulada, misma que almacena la threads, por medio del algoritmo de segmentación o paginación con que trabaja el sistema.

Caso de prueba:

Descripción: La prueba que se realizó consistió en inicializar los recursos por medio del módulo inic y un tamaño random para la memoria, luego se ejecutó el productor para que creara múltiples threads y estos hicieran uso de los recursos (memoria y semáforos), donde se registran los diferentes flujos de ejecución y estados, ya sea en las bitácoras o en las mismas memorias.

Note que estos datos posteriormente serán accedidos por el espía e impresos en consola para que el usuario pueda visualizarlos, eso sí, el

espía deberá solicitar los semáforos respectivos a lo que esté espiando (memoria o procesos).

Los siguientes dos comandos fueron los ingresados para probar al espía:

./espia mem (para espiar el estado de la memoria)

./espia proc (para espiar los estados de los procesos)

Resultado esperado: Se espera que el sistema imprima los datos de memoria o procesos, donde los mismos deben reflejar con claridad el estado con que se encuentra el sistema en un determinado momento.

Resultado obtenido: Exitoso, esto se afirma pues se hizo uso del debugger para visualizar los datos que se tenían en ejecución, entonces al realizar la solicitud del espía, los datos deberían coincidir y en efecto fue así.

Finalizador

El finalizador está demás describirlo ya que su función es sumamente clara. Respecto al cumplimiento del propósito que este tiene se podría decir que fue completado a cabalidad pues la eliminación de recursos (memoria compartida, semáforos, y threads) se realiza correctamente, este se puede comprobar por medio del caso de prueba que se muestra a continuación.

Caso de prueba:

Descripción: La prueba que se realizó consistió en inicializar los recursos por medio del módulo inic y un tamaño random para la memoria, luego se ejecutó el productor para que creara múltiples threads y que estos estuvieran “vivos” a la hora de realizar la liberación de recursos y así poder visualizar la eliminación correcta. Note que los recursos generados por el inicializador son los semáforos y las

memorias compartidas, por lo tanto el finalizador libera tres tipos de “estructuras” o objetos en memoria (memoria compartida, semáforos y threads).

Resultado esperado: Se espera que el sistema libere los recursos de manera eficiente, y esto hace énfasis que la eliminación permanente y sin dejar recursos “zombies” en memoria.

Resultado obtenido: Exitoso, esto se afirma pues se hizo uso de los comandos que proporciona **ipcs** para visualizar recursos en memoria. El comando empleado en este caso fue: **ipcs -a**

Resumen de Completitud de los Requerimientos

Para resumir la completitud de los requerimientos solicitados, a continuación, se mostrará una tabla con el requerimiento y su respectivo porcentaje de completitud.

Requerimiento	Porcentaje
Inicializador	100%
Productor de Procesos	100%
Programa Espía	100%
Bitácora	100%
Finalizador	100%
Sincronización de Procesos	100%
Interfaz del Sistema	100%
Lógica de Procesos	100%
Pedir semáforo de memoria	100%
Buscar su ubicación	100%
Escribir en bitácora	100%

Devolver semáforo de memoria	100%
Sleep	100%
Pedir Semáforo de memoria	100%
Liberar memoria	100%
Escribir en Bitácora	100%
Devolver semáforo de memoria	100%
Documentación	100%

Tipos de Semáforo Utilizados

Semáforos Binarios POSIX

Para mantener un control sobre los procesos que van a acceder a la memoria compartida, se hizo uso de **semáforos binarios** de tipo POSIX, que son los más recientes, por lo que es importante tomar en cuenta que al ser más nuevos, puede que no funcionen en los sistemas viejos basados en UNIX. Las llamadas de los semáforos de tipo POSIX son más sencillas que las de los semáforos tradicionales, lo que fue un punto importante a la hora de elegir cual de los dos utilizar.

Existen dos formas de trabajar con semáforos POSIX:

- **Sin nombre:** son utilizados solamente por los procesos *locales*, es decir, solamente el creador del mismo tiene derecho a utilizarlo.
- **Nombrados:** se identifican por un nombre y son guardados en la carpeta de memoria compartida del sistema operativo con el fin de que puedan ser utilizados por cualquier otro proceso.

Razón de la Elección

Como el objetivo del proyecto es el acceso a memoria compartida desde diferentes procesos, los semáforos finalmente utilizados fueron los **POSIX nombrados**, que utilizan la teoría de **semáforos binarios**.

Bitácora del Trabajo

Esta sección describe una serie de anotaciones que se llevaron a cabo a lo largo de la ejecución del proyecto, donde se incorporan datos importantes que se encontraron para ayudar con el desarrollo. Evidenciando el progreso realizado por el grupo de trabajo. Cabe resaltar que el trabajo realizado fue llevado a cabo utilizando un repositorio para el manejo de actividades y versiones realizadas a lo largo del proyecto. Este se puede consultar por medio del siguiente enlace:

https://github.com/AndreyMendoza/C_Shared_Memory.git

Descripción del trabajo realizado	Fecha de realización
Buscar la biblioteca encargada de solicitar espacio de memoria compartida al sistema operativo	16/10/17
Investigación tipos de semáforos para utilizar desde distintos procesos	17/10/17
Implementación básica de memoria compartida y uso de semáforos	18/10/17
Desarrollo del programa inicializador de la memoria compartida	21/10/17
Inicio de desarrollo del productor de memoria	23/10/17
Finalización del productor de memoria de tipo paginación y segmentación	24/10/17
Depuración del productor	24/10/17
Desarrollo del programa espía y el finalizador	24/10/17
Redacción de la documentación del proyecto	25/10/17

Compilación y Ejecución del Proyecto

Compilación

Respecto a la compilación del proyecto se debe tener en cuenta que es necesario tener instalado Cmake & Make en el ordenador, esto con la finalidad de poder generar el makefile y otros archivos necesarios para la ejecución del programa. A continuación se presenta una sugerencia de instalación con los respectivos pasos que se deben llevar a cabo para instalar dichos complementos.

1. sudo apt-get install build-essential
2. wget http://www.cmake.org/files/v3.2/cmake-3.2.2.tar.gz
3. tar xf cmake-3.2.2.tar.gz
4. cd cmake-3.2.2
5. ./configure
6. make
7. sudo apt-get install checkinstall
8. sudo checkinstall

Una vez completados estos pasos ya se contará con los complementos necesarios para ejecutar el proyecto desde la terminal, no obstante, primero se debe contar con una carpeta la cual almacenará los ejecutables de la aplicación, para ello hay que dirigirse a la ubicación del proyecto e ingresar el siguiente comando para crear dicha carpeta <mkdir Bin>. Luego hay que ubicarse dentro de la carpeta recién creada (Bin) y posteriormente realizar <cmake ..> dentro de la misma.

Llegando a este punto bastará con ingresar el comando <make> con el cual el programa estará compilado y por ende permitirá la ejecución de la aplicación.

Ejecución

La ejecución del proyecto se debe llevar a cabo una vez que el mismo se encuentra compilado correctamente, sólo bastará con ubicarse dentro de la carpeta Bin creada con el <mkdir Bin>, note que si ya existe dicha carpeta sólo bastará con dirigirse dentro.

Una vez hecho esto se deberán ingresar los parámetros que se van a enviar desde consola y por supuesto el nombre de la aplicación que se está intentando ejecutar. En este caso se cuenta con el proyecto llamado SharedMemory dentro del cual se deberá ejecutar el archivo main.c que contiene el menú de opciones que se ajusta a los parámetros pasados desde consola.

A continuación se muestran algunos ejemplos de ejecución y su respectiva explicación.

Inicializador

Tal como la sección lo indica, el inicializador es el encargado de la creación de todos los recursos necesarios para la ejecución de los otros módulos del sistema. Específicamente este inicializa las memorias compartidas con las que trabaja el productor y espía. Además, crea los semáforos para restringir el acceso a la región crítica.

Para la ejecución de este subprograma se debe ingresar además del nombre de módulo, el tamaño con que se debe crear la memoria compartida, es decir, la cantidad de celdas/bloques que contendrá la memoria. Entonces, un ejemplo claro de la ejecución sería:

./inic 50

Productor

Tal como la sección lo indica, el productor es el encargado de la creación de todos los hilos que realizan el proceso de asignación en memoria y de registrar los diferentes estados que tienen a lo largo de su ejecución. Específicamente este inicializa la memoria compartida con las estructuras que se desean trabajar

(paginación / segmentación), además asigna valores a dichas estructuras para que posteriormente los hilos hagan uso de los datos para realizar recorridos en memoria, ya sea para asignarse un espacio o para liberarlo.

Otro parte importante de este módulo, es que después de inicializar la memoria con las estructuras base, el productor pasa a un ciclo infinito que se encarga de estar generando hilos cada 30-60 segundos y asigna al hilo creado la función de buscar espacio en memoria con sus propias configuraciones.

Para la ejecución de este subprograma se debe ingresar además del nombre de módulo, el modo con que se debe trabajar sobre la memoria compartida, es decir, la segmentación o paginación. Entonces, un ejemplo claro de la ejecución sería:

./prod -s (modo segmentación)

./prod -p (modo paginación)

Espía

Tal como la sección lo indica, el espía es el encargado de brindar información al usuario, que en este caso sería el estado de los procesos y el estado en que se encuentra la memoria compartida en un determinado momento.

Para la ejecución de este subprograma se debe ingresar además del nombre de módulo, el modo de espiar con que se desea correr, es decir, mem (para espiar la memoria) o proc (para espiar los procesos). Entonces, un ejemplo claro de la ejecución sería:

./espia mem (espia la memoria)

./espia proc (espia los procesos)

Finalizador

Este módulo se encarga de finalizar y liberar todos los recursos que el inicializador y productor crearon, en este caso serían las dos memorias compartidas, los hilos creados y los semáforos para el control de acceso a las regiones críticas. Entonces, un ejemplo claro de la ejecución sería:

./fin

Conclusiones

Respecto al aprendizaje obtenido con el desarrollo de este proyecto, se podría decir que el objetivo de desarrollar un esquema de memoria compartida se ha cumplido a cabalidad y por ende los resultados han sido de gran enriquecimiento para el grupo de trabajo.

Por otra parte, la sincronización por medio de semáforos que fue implementada en el proyecto permitió emplear una solución muy convincente en términos de rendimiento, cumplimiento de requerimientos entre otros. Permitiendo comprender de mejor manera la comunicación y sincronización que existe entre procesos a la hora de intentar acceder a una misma zona crítica.

Finalmente, la implementación de procesos y semáforos fue de gran ayuda para aprender mucho más del lenguaje C, ya que el manejo de memoria y datos resulta muy cercano a lo que realmente sucede “por debajo” del código, entonces todo el proceso de creación y sincronización recae en el programador y en el buen uso de las instrucciones que el mismo emplee.

Referencias

K, Gaughan. (2003). *Shared Memory and Semaphores.* Recuperado de https://stereochro.me/assets/uploads/notes/dcom3/shmem.pdf

Delis, A. (2012). *Shared Memory and POSIX Semaphores.* Recuperado de: http://cgi.di.uoa.gr/~ad/k22/k22-lab-notes4.pdf

Kent State University. (2010). *Using a Semaphore to Synchronize Access to a Shared Memory Segment*. Recuperado de: http://www.cs.kent.edu/~ruttan/sysprog/lectures/shmem/shared-mem-with-semaphore.c

Damian, M. (2016). *Synchronizing Threads with POSIX Semaphores.* Universidad Villanova. Recuperado de: http://www.csc.villanova.edu/~mdamian/threads/posixsem.html