Escuela de Computación

Principios de Sistemas Operativos

Documentación

Segundo Proyecto Programado

Grupo 1

Profesora:
Erika Schumann Marín

Estudiantes:
Ariana Bermúdez Venegas
Ximena Bolaños Fonseca

Carnet:
2015019596
2015073844

II Semestre
2017

Índice

Tipo de Semáforos Utilizados

Los semáforos utilizados en este caso fueron los Binarios (mutex), debido a que este caso es muy similar al que es Readers y Writers. Ya que en este caso los Writers serían los hilos que agregan procesos a las líneas de código y las eliminan después de cierto tiempo y en el caso de los Readers sería algo muy similar a lo que es el espía que cada vez que se llama, muestra una bitácora de que se ha hecho.

Sincronización

La sincronización se logró gracias a lo que fueron los semáforos, de la librería semaphore.h, de los cuales tenían semáforos mutex de los cuales, a cómo planeamos el uso del sistema calza para evitar que hayan lecturas sucias. Se sincronizó de manera que pedía lo que era el recurso de la memoria y luego se devolvía una vez que se fuera a desocupar, entonces si alguno pedía la memoria mientras otro proceso la tenía se queda esperando, una vez que se suelta el que es más viejo con la petición, por lo que se utilizó FIFO para esto.

Las principales funciones de la librería que permite la sincronización entre procesos son:

- sem_open(): la cual permite crear un semáforo o abrir uno ya hecho.
- sem_wait(): bloquea la región crítica.
- sem_post(): desbloquea la región crítica.

Diferencia entre mmap y shmget

Para definir la diferencia entre lo que es mmap y shmget, es importante definir qué son y para que sirven para entender la diferencia bien. Primero se hablará un poco de como funciona el shmget.

El shmget sirve para poder acceder a una zona actualmente existente compartida o bien sirve para crear ese espacio del segmento. A este comando se le ingresa un key que es un entero, un size que es el tamaño del segmento y un shmflg que es una bandera inicial para el manejo del segmento. Este sirve únicamente para definir el segmento pero con este no se puede escribir en el. Este una vez que se le ingresan los parámetros tira un entero que en este caso sería el schmid que es el entero que define cuál es el segmento. El tamaño asignado tiene que ser múltiplo de 4 KB si es que se quiere crear en caso de que ya esté en existencia debe de llamarse con un lenguaje igual o menor del que está definido.

El shmflg que se puede componer según leclinux(n.f.) con:

- *IPC_CREAT: Para crear un nuevo segmento. Si este flag no se precisa, shmget() únicamente buscará el segmento asociado con la clave y comprobará si el usuario tiene permisos para acceder a él.*
- *IPC_EXCL: Se utiliza junto a IPC_CREAT para asegurar el fallo si el segmento existe.*

- *mode_flags: Los 9 bits menos significativos del número están reservados para el establecimiento de permisos de acceso. Actualmente los permisos de ejecución no son utilizados por el sistema.(p.5)*

Owner	Group	World

Por otro lado, se tiene la función mmap sirve para la ubicación de ficheros en memoria, en donde se tiene como entradas void *start, size_t length, int prot , int flags, int fd y off_t offset. En donde length es el tamaño del segmento que se quiere reservar, el offset es el desplazamiento que es preferible que sea justo del start. Esta dirección es preferible que se ponga como 0. La dirección que tira el mmap nunca será 0. La entrada prot significa la protección que se tendrá. Entre las opciones que define manpages-es_1.55-10_all(n.f.) están son :

- *PROT_EXEC Las páginas deben ser ejecutadas.*
- *PROT_READ Las páginas deben ser leídas.*
- *PROT_WRITE Las páginas deben ser escritas.*
- *PROT_NONE Las páginas no pueden ser accedidas.(p.1)*

El parámetro flags sirve para la identificación de objetos insertados y la modificación de los mismos, ya sea de manera privada o compartidas. Entre los tipos que se tienen según manpages-es_1.55-10_all(n.f.) están:

- MAP_FIXED No seleccionar una dirección diferente a la especificada. Si la dirección especificada no puede ser utilizada, mmap fallará. Si MAP_FIXED es especificado, start debe ser un múltiplo del tamaño de página. Utilizar esta opción es desaconsejable.
- MAP_SHARED Comparte este área con todos los otros objetos que señalan a este objeto. Almacenar en la región es equivalente a escribir en el fichero. El fichero puede no actualizarse hasta que se llame a msync(2) o munmap(2).
- MAP_PRIVATE Crear un área privada "copy-on-write". Almacenar en la región no afecta al fichero original. Es indefinido si los cambios hechos al fichero después de la llamada a mmap son visibles en la región mapeada.(p.1)

Es importante resaltar que en Linux también se permiten otro tipo de parámetros no tradicionales. El parámetro fd indica el fichero en el que se va a encontrar el trozo de memoria, a menos que sea anónimo(MAP_ANONYMOUS). Por último, el offset sería el tamaño que debe de ser múltiplo de la página. La función mmap es conservada después de un fork().

Una vez definidas se puede decir que la diferencia explícita entre lo que es mmap y shmget es que shmget se utiliza especialmente con lo que es la memoria compartida entre dos procesos y el mmap se utiliza especialmente en lo que es la exposición de un pedazo de memoria determinado a un espacio de usuario.

Otra de las diferencias que hay entre ambas es la facilidad de uso, ya que con mmap es mucho más fácil que con shmget obtener el pedazo de segmento, ya que es más flexible. A diferencia de shmget, mmap mapea un archivo en donde se encuentra toda la información

que se va a almacenar en el segmento de memoria, mientras que en shmget utiliza una estructura definida.
Una vez que es establecida la conexión con el segmento las llamadas al sistema son más especializadas en mmap que en shmget.
En el caso en el que mmap realice un archivo no volátil, una vez que se apaga el sistema, se vuelve disponible.
Finalmente, se puede concluir que la diferencia principal entre ambos es que mmap a través del parámetro fd mapea un archivo que será subido a memoria mientras que shmget accede a un segmento de memoria compartida especificando el tamaño.

Resultados

En la siguiente tabla se van a mostrar los objetivos que en este caso representan los ítems que se lograron durante el proceso y del otro lado que tanto del 1 al 100 de porcentaje se logró.

Resumen

Objetivos	**Porcentaje de Solucionado**
Programa Inicializador	100%
Programa Productor en Paginación	100%
Programa Productor en Segmentación	100%
Programa Espía	100%
Programa Finalizador	100%
Bitácora	100%
Implementación de Semáforos	100%
Memoria Compartida	100%

Detalle y Análisis de Resultados

Objetivos	**Porcentaje de Solucionado**
Se crean los 4 programas de manera correcta	100%
Cada uno, según sus especificaciones y parámetros inician de manera correcta, y se ejecutan.	100%
Geración y notificación de errores para el	100%

usuario en caso de no utilizar de manera indicada el programa	
El programa inicializador crea la memoria compartida, lista para que otros procesos la utilicen.	100%
El programa productor, genera lo procesos según el tipo solicitado.	100%
El programa espía logra solicitar la memoria para la lectura y/o ver el estado de los procesos.	100%
La sincronización entre programas se logró gracias a los semáforos.	100%
El programa finalizador llega a terminar con el productor, y elimina la memoria compartida.	100%
La bitácora si cuenta con lo solicitado para los registros de que sucede con los procesos, gracias a los otros programas.	100%

Casos de Prueba

Inicializador

Caso de prueba	Detalles	Porcentaje de solucionado
1. Solicita la cantidad de espacios que desea	El programa está pensado para solicitar los espacios de memoria deseados	100%
2. Crea la memoria compartida	Crea memoria compartida de la cantidad de espacios solicitados	100%
3. Mostrar error ante no parámetros	Si no hay parámetros que muestre un mensaje y no se caiga el programa.	100%
4. Mostrar error ante más parámetros de	Si hay más parámetros de lo esperado que muestre un	100%

la cuenta	mensaje y no se caiga el programa.	
5. Mostrar error ante no crear memoria	Si el programa no puede crear la memoria por algún motivo, notifica al usuario	100%

Finalizador

Caso de prueba	Detalles	Porcentaje de solucionado
1. Termina con el productor	El programa logra detener el otro proceso, el productor.	100%
2. Borra la memoria compartida	Elimina la memoria compartida creada por el inicializador.	100%
3. Cierra el archivo de bitácora	Deja completamente cerrado el archivo de bitácora	100%

Espía

Caso de prueba	Detalles	Porcentaje de solucionado
1. Muestra la memoria	El programa espía muestra el estado de la memoria	100%
2. Muestra los procesos	Debe mostrar el estado de los procesos, quienes estén en memoria, quienes estén buscando memoria, quienes están bloqueados esperando región crítica, y de quienes ya terminaron ejecución.	100%

3. Tener un menú	Tener un menú que maneje las dos posibles solicitudes del usuario	100%

Productor

Caso de prueba	Detalles	Porcentaje de solucionado
1. Crea procesos que llegan al “Ready”	Crea procesos cada cierto tiempo con valores dependiendo del modo	100%
2. Pregunta el modo productor	Pregunta si va a ser paginación o sincronización.	100%
3. Generación correcta de la paginación (páginas)	Genera procesos de paginación entre la cantidad de páginas establecidas	100%
4. Generación correcta de la paginación (tiempo)	Genera procesos de paginación entre la cantidad de tiempo establecido	100%
5. Generación correcta de la segmentación (segmentos)	Genera procesos de segmentación entre la cantidad de segmentos establecidos	100%
6. Generación correcta de la segmentación (espacios)	Genera procesos de segmentación entre la cantidad de espacios de memoria por segmento establecidos	100%
7. Generación correcta de la segmentación (tiempo)	Genera procesos de segmentación entre la cantidad de tiempo por segmento establecido	100%
8. La distribución genera procesos en un tiempo aleatorio	La generación de procesos se da en un tiempo establecidos dentro de un rango establecido.	100%

Bitácora

Caso de prueba	Detalles	Porcentaje de solucionado
1. Mostrar PID	En la bitácora se muestra el ID del proceso	100%
2. Mostrar Tipo	En la bitácora se muestra el tipo de acción (asignación, desasignación, si muere)	100%
3. Mostrar Hora	En la bitácora se muestra la hora de los eventos.	100%
4. Mostrar espacio asignado	En la bitácora dependiendo del tipo de productor, si es paginación, se muestra el espacio asignado a las páginas, si es segmentación, se muestra el espacio asignado al segmento.	100%
5. Registro de procesos que no entraron a memoria	El productor registra en bitácora que no entró un proceso a memoria por falta de espacio en la bitácora	100%

Bitácora de Trabajo

En la siguiente tabla se presenta una bitácora de trabajo en donde se demuestra la contribución de ambos miembros del equipo durante el desarrollo del proyecto. En donde las acciones son lo que se logró en ese día en específico o bien reuniones de trabajo que hubieron ese día.

Día	**Logro**	**Persona**
14/10/2017	Memoria Compartida (Programa Inicializador)	Ximena
19/10/2017	Montar Estructura en donde van a estar las páginas y los	Ariana

	Segmentos.(Programa Inicializador)	
20/10/2017	Reunión de trabajo para la organización de entradas y salidas de los programas.	Ariana y Ximena
21/10/2017	Creación del Programa Productor.	Ximena
22/10/2017	Creación del Programa Finalizador,Creación del Programa Espía y Documentación.	Ariana y Ximena
23/10/2017	Continuación de la Documentación.	Ariana
24/10/2017	Generación de cambios por mejoras a los programas, y validaciones.	Ariana y Ximena
25/10/2017	Finalización de la documentación.	Ariana y Ximena

Compilación y Ejecución

¿Cómo compilar?

Luego de descargar el proyecto, del Tec Digital, descomprimirlo, va a encontrar una carpeta que se llama Proyecto-II-SO, ingresa a esa carpeta, y desde ahí abre la terminal. Ubicado en esa carpeta en la terminal, escribe el comando “make”. Y se llevan a cabo la creación de los ejecutables para el proyecto. El make contiene las instrucciones de todos los programas, para que sea más fácil para el usuario.

¿Cómo ejecutar?

Una vez que tiene todos los ejecutables, se procede a ejecutar los programas, preferiblemente en este orden y con los siguiente parámetros:

Inicializador:

- Correrlo con el comando: ./inicializador
- Parámetros: cantidad de espacios
- Ejemplo: ./inicializador 50

Productor:

- Correrlo con el comando: ./productor
- Parámetros: tipo (0 para paginación y 1 para segmentación)
- Ejemplo: ./productor 0

Espía

- Correrlo con el comando: ./espia

- Parámetros: solicitud (1 para ver la memoria y 0 para ver los procesos)
- Ejemplo: ./espia 1

Finalizador

- Correrlo con el comando: ./finalizador
- Parámetros: ninguno
- Ejemplo: ./finalizador

¿Qué deben tener en cuenta?

El proyecto se hizo para Linux (se utilizó la versión 16.04), escrito en C, ya que el uso de programas en C con Linux es simple de ejecutar.

Referencias

Leclinux, n.f. *LECCIÓN 24 MEMORIA COMPARTIDA.* Recuperado de http://sopa.dis.ulpgc.es/ii-dso/leclinux/ipc/mem_compartida/lec24_memcomp.pdf

manpages-es_1.55-10_all, n.f. UBUNTU manuals. Recuperado de http://manpages.ubuntu.com/manpages/precise/es/man2/mmap.2.html

William, n.f. Shared Memory. Recuperado de http://home.deib.polimi.it/fornacia/lib/exe/fetch.php?media=teaching:piatt_sw_rete_polimi:2013_sharedmemory_william.pdf