TEC | Tecnológico de Costa Rica

Principios de Sistemas Operativos

Grupo 1

Proyecto 2

Simulación Particiones Dinámicas

Integrantes:

Agustín Blanco Acuña

Estiven Fernández Berrocal

Gustavo Blanco Alfaro

Luis Gustavo Alvarado

Introducción

El proyecto pretende crear una simulación de asignación de memoria a procesos mediante los diferentes algoritmos de particiones dinámicas, para ello se debe implementar un problema de sincronización de procesos. Además de la creación y utilización de memorias compartidas entre procesos, asimismo se aplicaron varios semáforos para llevar un orden de la entrada y salida de los recursos. Todo esto en el lenguaje de programación de C.

Se dividió el proyecto en cuatro programas diferentes, uno para inicializar las memorias compartidas requeridas. Otro para simular la producción de procesos, un tercero el cual tiene el trabajo de “espiar” al productor y mostrar resultados. Finalmente, un programa finalizador que detiene al productor y libera las memorias compartidas.

Este documento tiene por objetivo especificar la estrategia de solución utilizada en el desarrollo de los programas, así como los semáforos utilizados. Además de secciones tales como el análisis de resultados, lecciones aprendidas, los casos de pruebas empleados, un manual de usuario, una bitácora y, finalmente, la bibliografía utilizada.

Estrategia de Solución

Para el proyecto realizamos semáforos binarios, ya que nos brindan la funcionalidad perfecta para restringir a otros procesos el acceso a funciones como escribir en memoria, salir de la región crítica o entrar en ella.

Para desarrollar una sincronización lo más eficiente posible, definimos una serie de semáforos que considerábamos de suma importancia. El primero está definido entre nuestros Hilos con el objetivo de asegurar el orden entre los procesos a la hora de entrar o salir de la región crítica; el segundo Hilo, se encuentra entre el programa espía y el programa Productor, para asegurar que cuando el espía solicite la lectura de la memoria compartida no se esté realizando un cambio que genere una lectura inconsistente o deficiente; y el último Hilo se encuentra entre el programa Finalizador y el programa Productor con el objetivo de avisarle a nuestro programa Productor que debe detenerse de generar más Hilos.

En conjunto con lo anterior desarrollamos una lista extra de requerimientos que nosotros consideramos necesaria para alcanzar la completitud óptima del proyecto, comenzando con una investigación de uso en distintos sitios webs, sobre cómo se hacían cada una de las tareas básicas que necesitamos para el cumplimiento de las tareas en el siguiente orden:

1. Aprender a compartir información y memoria entre programas: para esto buscamos en nuestra primera referencia, el sitio web de Chuidiang sobre cómo funciona la memoria compartida, una vez comprendido, éramos capaces de crear nuestro programa inicializador.
2. Para evitar conflicto de sincronización usamos nuestras capacidades para implementar semáforos que impidan a procesos retirar recursos los unos a los otros, para ello usamos nuestra tercera referencia.
3. Crear estructuras de datos es una disciplina en la que somos más que expertos, para implementar una estructura que nos funcione y sea eficiente, pensamos en añadir una estructura para cada la memoria, que cumpla los requerimientos del proyecto y a cada proceso le guardamos su ID, duración, tamaño, base de

ubicación y el estado en el que se encuentra, para que otros proceso puedan saber dónde ubicarse o no hacerlo.

4. La creación de Hilos es algo que ya manejamos desde proyectos pasados, no requirió mayor investigación o estrategia para conseguirlo.
5. Manipular la región crítica es algo complicado, una vez asignada la memoria de la que va a disponer el proyecto necesitamos de un buen manejo de banderas para asegurar una lectura y escritura sincrónica de archivos y funciones.
6. Los algoritmos de acomodamiento de procesos siguen una serie de patrones, empezando por el first fit:
    a. Este lo que hace es buscar un espacio libre y comienza a contar las líneas libres a partir de la primera libre que encuentre y las que le sigan, en caso de haber suficientes libres entonces inserta ahí al proceso.
    b. El Best fit tiene una idea similar, pero para ello antes recorremos todas los grupos de celdas que cumplan con el tamaño mínimo que ocupamos, y luego descartamos los tamaños de mayor a menor para encontrar el menor posible.
    c. El Worst fit hace lo mismo pero descarta los grupos de línea en los que se puede encajar de menor a mayor dejándonos el más grande disponible.
7. Para el programa espía accede a la memoria compartida que teníamos definida previamente, pero antes de poder leer pregunta a una bandera, para evitar que lea mientras un proceso está entrando o saliendo y genere una inconsistencia, el resto del programa requiere de conocimiento aprendido usando las partes previas.
8. Para escribir en bitácora usamos los distintos movimientos que hacemos a lo largo de los procesos que nos arrojan un mensaje en pantalla, en simultáneo, usamos el archivo Log o bitácora y escribimos ahí en pantalla.
9. Con la primera referencia aprendimos la creación de espacios en memoria virtual y también la eliminación de los mismos, al ejecutar este programa busca la ID de todos los procesos, les ordena terminar y una vez que se hayan salido todos, entonces elimina de la memoria el segmento que se usaba para compartir información entre los distintos procesos.

Análisis de Resultados

Tabla de actividades

Actividades	Porcentaje
Programa inicializador	100%
Programa productor	100%
Programa espía	100%
Programa finalizador	100%
Bitácora	100%
Sincronización de procesos	100%
Interfaz	100%

Análisis de fragmentación

Con las pruebas realizadas al finalizar las distintas etapas del desarrollo del proyecto, nos dimos cuenta de que el algoritmo de Best Fit es el menos eficiente, al menos al momento de manejar la fragmentación, ya que en la mayoría de los casos el mismo encaja dentro de la memoria dejando adyacente al proceso asignado espacios en memoria tan pequeños que ningún otro proceso podrá aprovechar, dejando en la mayoría de los casos un conjunto de muchos pequeños espacios de memoria en los que de manera separada ningún proceso puede usar, pero si no hubiesen quedado tan separados se les hubiese podido asignar a algún proceso.

Lecciones aprendidas

- Con el proyecto hemos aprendido una serie de funcionamientos que se ejecutan en nuestro sistema operativo sin que nosotros nos demos cuenta, para asegurar una buena experiencia de usuario y mejorar la calidad y estabilidad del sistema como:
    a. La sincronización de procesos, vital para que muchos cálculos no se vuelvan inconsistentes y lleguen a causar errores a la larga.
    b. El manejo de memoria y manera de compartirla entre distintos programas, lo más complicado aprendido durante este proyecto y sin duda lo más interesante; aprendimos como en C se desarrollan la memoria compartida y cómo ésta es capaz de pasar datos e información entre distintos programas para realizar tareas más complejas sin tener que guardar esta información en disco haciendo así los procesos más lentos y complejos.
    c. Organización de memoria; aprendimos como la memoria puede distribuir los recursos de los que dispone para distintos procesos y cómo la misma se puede fragmentar impidiendo en algunos casos que un proceso comience a ejecutarse a pesar de que haya suficiente memoria libre, pero estando la misma fragmentada y repartida en distintos sectores.
    d. Banderas entre programas; aprendimos no solo como sincronizar un proceso y un sector de memoria, si no que con ayuda de este proyecto ahora tenemos una idea mucho más clara de cómo se desarrolla un sistema de banderas y semáforos entre distintos procesos de un mismo programa, inclusive distintos procesos de distintos programas.

Casos de pruebas

First Fit:

	PID	Tamaño	Tipo	Accion	Hora	Lineas asignadas
2	7399	7	Asignacion	Alocado en memoria	Thu Oct 28 13:03:26 2021	(0,7)
3	7399	7	Desasignacion	Liberando memoria	Thu Oct 28 13:03:46 2021	(0,7)
4	7405	4	Asignacion	Alocado en memoria	Thu Oct 28 13:04:06 2021	(0,4)
5	7413	3	Asignacion	Alocado en memoria	Thu Oct 28 13:04:43 2021	(4,7)
6	7405	4	Desasignacion	Liberando memoria	Thu Oct 28 13:04:45 2021	(0,4)
7	7417	3	Asignacion	Alocado en memoria	Thu Oct 28 13:05:20 2021	(0,3)
8	7413	3	Desasignacion	Liberando memoria	Thu Oct 28 13:05:26 2021	(4,7)
9	7417	3	Desasignacion	Liberando memoria	Thu Oct 28 13:05:46 2021	(0,3)
10	7423	10	Asignacion	Alocado en memoria	Thu Oct 28 13:06:12 2021	(0,10)
11	7423	10	Desasignacion	Liberando memoria	Thu Oct 28 13:06:40 2021	(0,10)
12	7428	1	Asignacion	Alocado en memoria	Thu Oct 28 13:07:05 2021	(0,1)
13	7431	7	Asignacion	Alocado en memoria	Thu Oct 28 13:07:40 2021	(1,8)
14	7428	1	Desasignacion	Liberando memoria	Thu Oct 28 13:07:44 2021	(0,1)
15	7437	8	Asignacion	Alocado en memoria	Thu Oct 28 13:08:14 2021	(8,16)
16	7431	7	Desasignacion	Liberando memoria	Thu Oct 28 13:08:26 2021	(1,8)
17	7441	1	Asignacion	Alocado en memoria	Thu Oct 28 13:08:45 2021	(0,1)
18	7437	8	Desasignacion	Liberando memoria	Thu Oct 28 13:08:55 2021	(8,16)
19	7441	1	Desasignacion	Liberando memoria	Thu Oct 28 13:09:29 2021	(0,1)
20	7449	8	Asignacion	Alocado en memoria	Thu Oct 28 13:09:43 2021	(0,8)
21	7451	3	Asignacion	Alocado en memoria	Thu Oct 28 13:10:32 2021	(8,11)
22	7449	8	Desasignacion	Liberando memoria	Thu Oct 28 13:10:35 2021	(0,8)
23	7469	10	Cancelado	Sin espacio suficiente	Thu Oct 28 13:11:14 2021	na
24	7451	3	Desasignacion	Liberando memoria	Thu Oct 28 13:11:20 2021	(8,11)
25	7487	7	Asignacion	Alocado en memoria	Thu Oct 28 13:11:57 2021	(0,7)

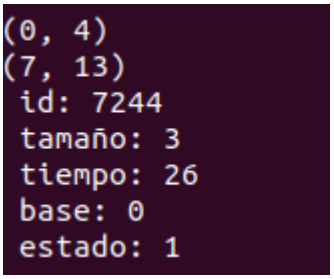
Este es el caso más sencillo de evaluar, básicamente queremos que los procesos se vayan insertando lo más atrás posible, en esta corrida de prueba con una memoria de 20 espacios, se puede apreciar este caso por ejemplo en la línea 7, donde antes de esto los espacios del 0 al 7 están ocupados y se libera el espacio de 0 a 4, para luego ingresar un proceso de espacio 3, por lo que puede ingresar en los espacios entre 0 a 4 y de 7 a 19, pero como es First Fit elige el primer espacio.

También se puede apreciar en la línea 23 que hay un proceso que no logró ocupar la memoria, es un proceso que necesita 10 espacios y en las líneas anteriores se puede ver que hay espacios ocupados 8 a 11, por lo que no tiene espacio, pues de 0 a 8 y de 11 a 19 no hay 10 espacios. Este mismo caso sucedió con el Best Fit y el Worst Fit que no cupo un proceso, ambos se ven en la línea 23 de igual manera.

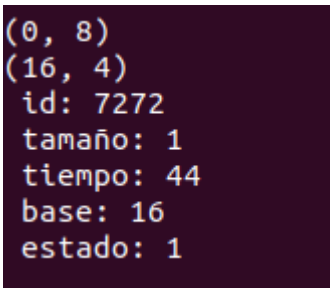
Best Fit:

	PID	Tamaño	Tipo	Accion	Hora	Lineas asignadas
2	7123	7	Asignacion	Alocado en memoria	Thu Oct 28 12:37:36 2021	(0,7)
3	7123	7	Desasignacion	Liberando memoria	Thu Oct 28 12:37:56 2021	(0,7)
4	7236	4	Asignacion	Alocado en memoria	Thu Oct 28 12:38:16 2021	(0,4)
5	7239	3	Asignacion	Alocado en memoria	Thu Oct 28 12:38:53 2021	(4,7)
6	7236	4	Desasignacion	Liberando memoria	Thu Oct 28 12:38:55 2021	(0,4)
7	7244	3	Asignacion	Alocado en memoria	Thu Oct 28 12:39:30 2021	(0,3)
8	7239	3	Desasignacion	Liberando memoria	Thu Oct 28 12:39:36 2021	(4,7)
9	7244	3	Desasignacion	Liberando memoria	Thu Oct 28 12:39:56 2021	(0,3)
10	7250	10	Asignacion	Alocado en memoria	Thu Oct 28 12:40:22 2021	(0,10)
11	7250	10	Desasignacion	Liberando memoria	Thu Oct 28 12:40:50 2021	(0,10)
12	7255	1	Asignacion	Alocado en memoria	Thu Oct 28 12:41:15 2021	(0,1)
13	7259	7	Asignacion	Alocado en memoria	Thu Oct 28 12:41:50 2021	(1,8)
14	7255	1	Desasignacion	Liberando memoria	Thu Oct 28 12:41:54 2021	(0,1)
15	7265	8	Asignacion	Alocado en memoria	Thu Oct 28 12:42:24 2021	(8,16)
16	7259	7	Desasignacion	Liberando memoria	Thu Oct 28 12:42:36 2021	(1,8)
17	7272	1	Asignacion	Alocado en memoria	Thu Oct 28 12:42:55 2021	(16,17)
18	7265	8	Desasignacion	Liberando memoria	Thu Oct 28 12:43:05 2021	(8,16)
19	7272	1	Desasignacion	Liberando memoria	Thu Oct 28 12:43:39 2021	(16,17)
20	7278	8	Asignacion	Alocado en memoria	Thu Oct 28 12:43:53 2021	(0,8)
21	7280	3	Asignacion	Alocado en memoria	Thu Oct 28 12:44:42 2021	(8,11)
22	7278	8	Desasignacion	Liberando memoria	Thu Oct 28 12:44:45 2021	(0,8)
23	7293	10	Cancelado	Sin espacio suficiente	Thu Oct 28 12:45:24 2021	na
24	7280	3	Desasignacion	Liberando memoria	Thu Oct 28 12:45:30 2021	(8,11)
25	7298	7	Asignacion	Alocado en memoria	Thu Oct 28 12:46:07 2021	(0,7)

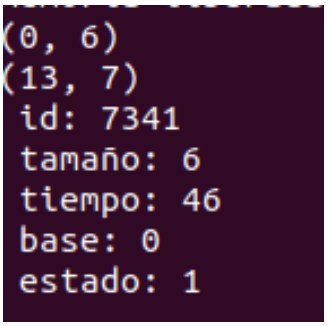
En el caso del Best Fit se espera que los procesos elijan ingresar en los lugares donde tengan menor espacio, y esto se ve nuevamente en la línea 4, queda un espacio entre 0 y 4 y entre 7 y 19, así que elige el primer espacio porque es el más pequeño.
En las siguiente capturas del programa en ejecución se ve de forma más fácil los casos en donde se encuentran varios huecos disponibles y elige el más pequeño:



Este proceso de tamaño 3 encuentra dos espacios en ese momento, uno con base en 0 de tamaño 4 y otro de base 7 de tamaño 13, como el algoritmo de Best Fit tiene que elegir el espacio más pequeño entonces elige el de base 0, y como se ve la base del proceso es 0.



En este otro caso el espacio más pequeño es en la base 16 con 4 espacios, así que el proceso elige la memoria en la base 16.



En este último ejemplo llega un proceso de tamaño 6 y hay dos espacios disponibles, uno en la base 0 de tamaño 6 y otro en la base 13 de tamaño 7, así que se elige la base 0 porque es el tamaño más pequeño.

Worst Fit:

	PID	Tamaño	Tipo	Accion	Hora	Lineas asignadas
2	7531	7	Asignacion	Alocado en memoria	Thu Oct 28 13:14:38 2021	(0,7)
3	7531	7	Desasignacion	Liberando memoria	Thu Oct 28 13:14:58 2021	(0,7)
4	7535	4	Asignacion	Alocado en memoria	Thu Oct 28 13:15:18 2021	(0,4)
5	7544	3	Asignacion	Alocado en memoria	Thu Oct 28 13:15:55 2021	(4,7)
6	7535	4	Desasignacion	Liberando memoria	Thu Oct 28 13:15:57 2021	(0,4)
7	7548	3	Asignacion	Alocado en memoria	Thu Oct 28 13:16:32 2021	(7,10)
8	7544	3	Desasignacion	Liberando memoria	Thu Oct 28 13:16:38 2021	(4,7)
9	7548	3	Desasignacion	Liberando memoria	Thu Oct 28 13:16:58 2021	(7,10)
10	7557	10	Asignacion	Alocado en memoria	Thu Oct 28 13:17:24 2021	(0,10)
11	7557	10	Desasignacion	Liberando memoria	Thu Oct 28 13:17:52 2021	(0,10)
12	7564	1	Asignacion	Alocado en memoria	Thu Oct 28 13:18:17 2021	(0,1)
13	7567	7	Asignacion	Alocado en memoria	Thu Oct 28 13:18:52 2021	(1,8)
14	7564	1	Desasignacion	Liberando memoria	Thu Oct 28 13:18:56 2021	(0,1)
15	7576	8	Asignacion	Alocado en memoria	Thu Oct 28 13:19:26 2021	(8,16)
16	7567	7	Desasignacion	Liberando memoria	Thu Oct 28 13:19:38 2021	(1,8)
17	7580	1	Asignacion	Alocado en memoria	Thu Oct 28 13:19:57 2021	(0,1)
18	7576	8	Desasignacion	Liberando memoria	Thu Oct 28 13:20:07 2021	(8,16)
19	7580	1	Desasignacion	Liberando memoria	Thu Oct 28 13:20:41 2021	(0,1)
20	7586	8	Asignacion	Alocado en memoria	Thu Oct 28 13:20:55 2021	(0,8)
21	7588	3	Asignacion	Alocado en memoria	Thu Oct 28 13:21:44 2021	(8,11)
22	7586	8	Desasignacion	Liberando memoria	Thu Oct 28 13:21:47 2021	(0,8)
23	7593	10	Cancelado	Sin espacio suficiente	Thu Oct 28 13:22:26 2021	na
24	7588	3	Desasignacion	Liberando memoria	Thu Oct 28 13:22:32 2021	(8,11)
25	7598	7	Asignacion	Alocado en memoria	Thu Oct 28 13:23:09 2021	(0,7)

Para el algoritmo de Worst Fit los procesos deben elegir la memoria con mayor espacio, así que de forma contraria al Best Fit, este algoritmo elige los espacios contrarios a los que se eligió en el ejemplo anterior:

```
(0, 4)
(7, 13)
 id: 7548
 tamaño: 3
 tiempo: 26
 base: 7
 estado: 1
```

En el mismo caso de la memoria con espacios de 4 y 13 espacios, el Worst Fit elige la base 7 con tamaño de 13.

```
(0, 8)
(16, 4)
 id: 7580
 tamaño: 1
 tiempo: 44
 base: 0
 estado: 1
```

En el segundo caso elige el primer espacio de tamaño 8 en lugar del espacio de tamaño 4 como pasó en el Best Fit.

```
(0, 6)
(13, 7)
 id: 7659
 tamaño: 6
 tiempo: 46
 base: 13
 estado: 1
```

Y finalmente el último caso en donde entre los espacios de 6 y 7 elige el espacio de 7 con la base de 13.

Comparación entre funciones de memoria compartida

La primera diferencia se presenta en el origen de cada función. En el caso de mmap(), la cual forma parte de las llamadas al sistema parte de los sistemas POSIX. Para dar un poco de contexto, POSIX son las siglas al conjunto de estándares de la IEEE llamado “Portable Operating System Interface”. Por otro lado, shmget, tuvo su origen un tiempo antes con System V que era una versión de Unix.

En cuanto a implementación, entre los dos sí varían bastante. Por ejemplo, en el caso de shmget debemos primero obtener una llave, para después poder generar el ID que nos permitirá, ya sea, crear por primera vez y generar la memoria para darle uso. Lo que resulta en la necesidad de un total de tres funciones para poder hacer uso de una memoria compartida (ftok(), shmget() y shmat()) . En contraparte, el mmap, solo es necesario utilizar una sola función (mmap()) para poder generar la memoria compartida.

Otra comparación son las situaciones en donde implementar cada función. Por ejemplo, mmap es considerado el favorito al tener que introducir memoria compartida creando procesos con forks(). Debido a que ya trae consigo múltiples flags que ayudan bastante y son fáciles de implementar. Pero, también se puede argumentar que, sabiendo que shmget tiene más tiempo de existir, lo que significa que ha sido desarrollado e introducido a los sistemas de Unix por más tiempo, como si fuera un veterano.

En conclusión, es cuestión de experimentar con ambas y analizar cuál de las dos es más adaptable al proyecto que está siendo desarrollado.

Manual de usuario

1. Para compilar el proyecto debe ejecutarse el comando:

gcc inicializador.c funciones.c -o p1 && gcc productor.c funciones.c -o p2 -pthread && gcc espia.c funciones.c -o p3 && gcc finalizador.c funciones.c -o p4

2. Para **inicializar el proyecto** primero se ejecuta el programa **p1**, este inicializa la memoria y prepara el archivo de la bitácora, en este programa se le solicita al usuario la cantidad de memoria que dispondrán los procesos.

3. Seguidamente se ejecuta el programa **p2**, este es el **productor de procesos**, le pide al usuario si acomodar los procesos siguiendo el algoritmo de First Fit (1), Best Fit (2) o Worst Fit (3), una vez ingresada la opción se empezarán a generar procesos y asignándoles en la memoria.

4. Opcionalmente se puede ejecutar el programa **p3** luego de dejar ejecutando el programa **p2**, este **es el espía**, al ejecutarlo se despliega un menú con el que se podrán realizar 3 acciones: Visualizar el estado de la memoria (1), Visualizar el estado de los procesos (2) o salir del espía (3).

5. Finalmente para **finalizar** la ejecución del productor se ejecuta el programa **p4**.

Bitácora de trabajo

17/10/2021: Iniciación del proyecto. Se investigó sobre cómo usar la memoria compartida entre archivos y se implementó con datos de prueba, logrando implementar la funcionalidad básica de crear la memoria, inicializar algunos datos, leerlos y limpiar la memoria, esto desde archivos diferentes.

20/10/2021: Modularización y acomodo del código.

23/10/2021: Se desarrolló la creación de los procesos y su asignación a memoria con los algoritmos de First Fit, Best Fit y Worst Fit. Se implementó la funcionalidad del programa espía. Se implementó la funcionalidad de la bitácora. Se implementó el finalizador.

27/10/2021: Corrección de un error producido por los hilos, finalización de la documentación y con esto finalización del proyecto.

Bibliografía

[1] Abellán, J. (2007, 4 de febrero). Memoria compartida en C para Linux. Ejemplos de Java y C/Linux. http://www.chuidiang.org/clinux/ipcs/mem_comp.php

[2] EZOIC. (2021, 11 de marzo). Utilice shmget para asignar memoria compartida en C. Delft Stack. https://www.delftstack.com/es/howto/c/shmget-example-in-c/

[3] Abellán, J. (2007, 4 de febrero). Semáforos en C para Unix/Linux. Ejemplos de Java y C/Linux. http://www.chuidiang.org/clinux/ipcs/semaforo.php

[4] Linux shared memory: shmget() vs mmap()? (2014, 23 de enero). Stack Overflow. https://stackoverflow.com/questions/21311080/linux-shared-memory-shmget-vs-mmap

[5] Shmget (2021, 22 de marzo). Linux Manual Page. shmget(2) - Linux manual page (man7.org)

[6] Shmget (2021, 22 de marzo). Linux Manual Page. mmap(2) - Linux manual page (man7.org)