Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Sistemas Operativos

Simulador-Paginación-segmentación

Profesora:

Erika Marín Schumann

Estudiantes:

María José Cortés - 2018138674

Esteban Cruz López - 2018104794

Luis Venegas Leiva – 2019079322

I Semestre 2022

Contenido

Introducción

Este proyecto se realizó como parte del curso de Principios de Sistemas Operativos de la carrera de Ingeniería en Computación en el Instituto Tecnológico de Costa Rica en el primer semestre de 2022 con la profesora Erika Marín Shumann.

Se trata de un simulador de paginación o segmentación, según sea inicializado el simulador. Para esto, según fue mencionado, se requiere la implementación de un programa inicializador que solicite el espacio compartido al sistema operativo (Linux) para que el programa productor.c simule la creación y ejecución de procesos que van a requerir ser ejecutados por cierto tiempo, así como un determinado tamaño en memoria y la especificación del tamaño de sus segmentos si es que el simulador fue iniciado en modo segmentación.

También se requiere un programa 'espía' para la observación del estado del simulador y finalmente un programa dedicado a finiquitar el compartimiento de memoria.

Según el enunciado del proyecto será implementado en C en Linux, utilizando la biblioteca sys/shm.h para generar los compartimientos de memoria. También se hizo uso del manejo de versiones que provee github y de los semáforos e hilos que provee pthreads.

Estrategia de solución

Este proyecto consta de 4 programas que interactúan entre ellos por medio de una memoria compartida que simula una memoria principal en la que deben alojarse programas para ser

ejecutados, una vez terminado su tiempo de ejecución estos salen de la memoria terminando su ciclo de vida.

Para asegurar la comunicación y la sincronización de los procesos se utilizan named semaphores para impedir que haya más de un proceso en la región crítica, la cual es en la que se lee, interpreta y/o modifica la memoria compartida, logrando una correcta integridad de los datos que allí se ubican.

En la siguiente captura se muestra la sección del código que se considera crítica debido a que es allí donde los procesos buscan su espacio en memoria y si la encuentran se ubican en esos espacios, si no encuentra el espacio debe morir, como se puede observar al final de la región crítica.

```
75      // inicio región crítica
76      if (sem_trywait(sem) != 0){
77          registrar_espera(i);
78          sem_wait(sem);
79      }
80
81      registrar_buscar(i);
82
83      int* arr;
84      int shmid = shmget((key_t)2345, 0, 0666 | IPC_EXCL);
85      arr = shmat(shmid, NULL, 0);
86
87      size_t act = 0;
88      int pagLibres = 0;
89
90      while (arr[act] != -1 && cantPaginas > pagLibres) {
91          if (arr[act] == 0)
92              pagLibres += 1;
93          act++;
94      }
95      printf("Paginas libres:%i\n", pagLibres);
96      act = 0;
97      char paginasAsignadas[20];
98  >   if (pagLibres == cantPaginas) {···
112     else {
113         registrar_muerte(i);
114         sem_post(sem);
115         pthread_exit(NULL);
116     }
117     sem_post(sem);
118     registrar_inicio(i, paginasAsignadas);
```

Note que lo primero que hace el proceso es tratar de pedir la memoria compartida, si logra pedirla el resultado es 0 y entonces puede entrar a la región crítica, sino el resultado es distinto a 0 y entra al if en el cual se registra la espera y se le obliga a esperar. Esta variable semáforo se inicializa en el inicializador y se comparte entre los procesos por medio de los métodos sem_unlink y sem_open, como se muestra a continuación:

```
21  int main() {
22      sem_unlink ("/semaforoMemoria");
23      sem_unlink ("/semaforoBitacora");
24
25      sem=sem_open("/semaforoMemoria",  O_CREAT,0666,1);
26      semBit=sem_open("/semaforoBitacora",  O_CREAT,0666,1);
```

Logrando que cuando el espía busca por la información para mostrársela al usuario, debe esperar si algún proceso está en la región crítica, como se muestra a continuación:

```
130 void mostrar_memoria(){
131     sem_wait(sem);
132     int *arr;
133     int shmid = shmget((key_t)2345,0,0666|IPC_EXCL);
134     arr = shmat(shmid, NULL, 0);
135     int j=0;
136     while(arr[j]!=-1){
137         printf("%d  ", arr[j] );
138         j+=1;
139     }
140     printf("\n");
141     sem_post(sem);
142 }
```

También los programas se comunican por medio de la bitácora, ya que como los procesos imprimen ahí todos los cambios de estado que presenten, el programa espía lee estos cambios de estado para concluir el estado actual de los procesos.

Finalmente, el finalizador ejecuta el siguiente código para eliminar los semáforos y cerrar la memoria compartida, finiquitando la simulación poniendo fecha de fin a la bitácora.

```c
sem_t * sem;
sem_t * semBit;
int main() {
    sem=sem_open("/semaforoMemoria",  O_RDWR);
    semBit=sem_open("/semaforoBitacora",  O_RDWR);
    sem_unlink("/semaforoMemoria");
    sem_unlink("/semaforoBitacora");
    sem_close(sem);
    sem_close(semBit);
    void *shared_memory;
    int shmid;
    shmid = shmget((key_t)2345, 0, 0666);
    printf("Llave de la memoria compartida: %d\n", shmid);
    shared_memory = shmat(shmid, NULL, 0);
    shmdt(shared_memory);
    shmctl(shmid, IPC_RMID, NULL);
    if (shmid >= 0)
        printf("Memoria liberada.\n");
    char buff[100];
    time_t now = time(NULL);
    strftime(buff, 100, "bitacora del %d-%m-%Y a las %H hora
    int r = rename("bitacora.txt", buff);
    printf("Resultado del rename: %d\n", r);
    if (r == 0)
        printf("Se generó la %s\n", buff);
}
```

Análisis de resultados

Las funcionalidades se encuentran implementadas al 100%, cada uno de los programas cumple con su función a la perfección, generando bitácoras como la siguiente:

bitacora del 22-05-2022 a las 18 horas con 09 minutos y 59 segundos.txt

```
Se ha escogido el algoritmo de segmentación.
0000: El proceso con pid = 001 busca su campo en memoria.
0000: El proceso con pid = 001 se empezó a ejecutar habiendo encontrado estos campos 00, 01, 02, 03, 04,
0019: El proceso con pid = 001 terminó de ejecutarse y salió de la memoria.
0040: El proceso con pid = 002 busca su campo en memoria.
0040: El proceso con pid = 002 se empezó a ejecutar habiendo encontrado estos campos 00, 01, 02, 03, 04,
0064: El proceso con pid = 002 terminó de ejecutarse y salió de la memoria.
0077: El proceso con pid = 003 busca su campo en memoria.
0077: El proceso con pid = 003 se empezó a ejecutar habiendo encontrado estos campos 00, .
0099: El proceso con pid = 003 terminó de ejecutarse y salió de la memoria.
0137: El proceso con pid = 004 busca su campo en memoria.
0137: El proceso con pid = 004 se empezó a ejecutar habiendo encontrado estos campos 00, 01, 02, 03, .
0168: El proceso con pid = 004 terminó de ejecutarse y salió de la memoria.

```

En la sección de pruebas podrá encontrar más detalle de la ejecución de la simulación. Por programa, las funciones completadas son las siguientes:

Inicializador

Pide al usuario tipo de simulación.

Pide al usuario tamaño de memoria.

Pide al sistema operativo memoria compartida.

Productor

Generar datos base de los procesos

Ejecutar la simulación de los procesos

Tener cuidado con las regiones críticas

Espía

Mostrar al usuario el estado de los procesos

Mostrar al usuario el estado de la memoria.

Finalizador

Matar todos los procesos que estén en escena.

Devolver los recursos que se solicitaron en el inicializador.

Cerrar la bitácora.

Lecciones aprendidas

Durante el desarrollo de este proyecto pudimos aprender los distintos métodos para manejar la memoria compartida en el lenguaje de programación c, en este caso con shm y mmap, así como los distintos comandos que permiten visualizar todos los segmentos de memoria compartidos en el sistema operativo.

Además, también aprendimos a aplicar los conocimientos adquiridos en el curso sobre semáforos y comprendimos la forma en la que estos se crea, modifican y destruyen en el lenguaje de programación c.

Este proyecto también nos ayudó a reforzar, aplicar e implementar en un lenguaje de programación una simulación los algoritmos de paginación y segmentación.

Pruebas

Para ejecutar una simulación con nuestro proyecto es bastante fácil si se puede utilizar el comando make para compilar todos los programas de una vez, pero si no basta con compilarlos de la siguiente forma:

```
gcc inicializador.c -o inicializador -pthread
```

gcc productor.c -o productor -pthread

gcc espia.c -o espia -pthread

gcc finalizador.c -o finalizador -pthread

Una vez compilado sin problema nuestro proyecto se puede proceder a ejecutarlo y a seguir las instrucciones que son realmente bastante básicas, por ejemplo ejecutando el inicializador con el siguiente comando:

./inicializador

El programa deberá mostrar algo similar a los siguiente:

```
zesteban22@DESKTOP-03UBQRF:/mnt/c/Users/User/Desktop/Simulador-Paginacion-Segmentacion$ ./inicializador
Ingrese la cantidad de unidades espaciales de memoria.
> 20
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Escoja por favor el método de manejo de memoria: P - paginación o S - segmentación
> s
Se ha escogido el algoritmo de segmentación.
```

Posteriormente se podrá ejecutar sin problema el productor y el espía, además de manera simultánea en varias terminales.

./productor

./espia

El productor no requiere ninguna interación extra, pero el espía le va a solicitar si quiere ver la memoria o el estado de los procesos, como se muestra en la siguiente imágen:

```
/productor
¿Es paginación?
No.

Proceso numero 1 generado
Tiempo entre generación: 40

Empieza a crearse el proceso 1
Duracion del sleep: 56
Cantidad de segmentos: 3
2  3  2
```

```
/espia
¿Desea ver el estado de la (M)emoria, ver el estado de los (P)r
ocesos o (S)alir?
> m
1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
¿Desea ver el estado de la (M)emoria, ver el estado de los (P)r
ocesos o (S)alir?
> m
1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
¿Desea ver el estado de la (M)emoria, ver el estado de los (P)r
ocesos o (S)alir?
>
```

Una vez terminada la simulación se procede a interrumpir la ejecución del productor de procesos y a ejecutar el finalizador de la siguiente forma:

```
zesteban22@DESKTOP-03UBQRF:/mnt/c/Users/User/Desktop/Simulador-Pagi
nacion-Segmentacion$ ./finalizador
Llave de la memoria compartida: 0
Memoria liberada.
Resultado del rename: 0
Se generó la bitacora del 24-05-2022 a las 09 horas con 16 minutos
y 20 segundos.txt
```

Concluyendo así el ciclo de simulación de nuestro proyecto con el resultado de la bitácora generada.

mmap vs shmget

La principal diferencia entre mmap y shmget radica en que mmap es un poco más restrictiva, pero también es más simple o fácil de usar, shmget es un método más antiguo, por lo que tiene un soporte más completo.

Mmap mapea el contenido de un archivo a una sección de memoria y este archivo puede ser modificado y leído cuando se lee o modifica esta sección de memoria, y los procesos pueden acceder al archivo de la forma que acceden a memoria, sin llamar las instrucciones del sistema (read, write).

Mientras que shm mapea directamente el espacio de la memoria virtual del proceso a la memoria física.

Por lo tanto, mmap es más simple de usar y más conveniente que shm, por lo que la mayoria de personas prefieren utilizar mmap.

Otra ventaja de mmap es que cuando la computadora se reinicia, mmap no se perderá, ya que guarda el archivo en disco, y este archivo también guarda la imagen de sincronización del sistema operativo.\

Manual de usuario

1.La solución está planteada para ejecutarse en un ambiente Linux, para compilar en C mediante el compilador gcc. Primeramente, para la compilación de los 4 programas que componen este proyecto se cuenta con un archivo makefile, por lo que, para realizar la compilación se debe ejecutar el comando 'make'.

2. Una vez compilados todos los programas, se debe realizar la ejecución del archivo llamado 'inicializador', el cuál se encarga de realizar la asignación de la memoria compartida que se utilizará. Este programa pedirá al usuario elegir el tipo de algoritmo que desea implementar para la simulación, además de controlar la cantidad de espacios que se utilizarán como espacios de memoria durante la simulación.

3. Seguidamente se debe ejecutar el archivo 'productor', esta ventana se debe mantener activa siempre, ya que es la encargada de generar la simulación de procesos de forma constante, además muestra información de los procesos que son creados y el manejo de memoria compartida de estos.

4. Si se desea conocer los detalles de la simulación, se debe ejecutar el archivo 'espia', el cuál muestra al usuario todos los datos correspondientes a esta.

5. Cuando se desee detener la simulación se debe ejecutar el programa "finalizador", que se encarga de devolver los segmentos de memoria compartida y cerrar la bitácora.

Bitácora de trabajo

- 7 de mayo: Reunión inicial con todos los integrantes del grupo para acordar el método de trabajo general. Duración 1 hora.
- Del 8 al 14 de mayo se trabajó individualmente en una base para cada uno de los programas.
- 15 de mayo: Reunión en la que ya se cuenta con una base inicial de los 4 programas y se analizan la estructura final que tendrá el programa productor. Duración 1 hora.
  16 de mayo: Se realizó parte del algoritmo de paginación.
- 17 de mayo: Nos reunimos para finalizar el desarrollo de la simulación del algoritmo de paginación y avanzar con la bitácora de procesos. Duración 2.5 horas
- 18 de mayo: Se trabajó de forma asincrónica en la elaboración de la bitácora. Duración 4 horas
- 19 de mayo: Se trabajó de forma asincrónica en desarrollo del algoritmo de segmentación. Duración 3 horas
- 20-21 de mayo: Se diseña el programa espía con todas sus funcionalidades (ver estado de memoria e información de procesos). Duración 4 horas
- 24 de mayo: Se le dio los últimos toques a la documentación y se entregó el proyecto. Duración 3 horas.

Bibliografía

ExchangeStatus(2014). Linux shared memory: shmget() vs mmap()?. Extraído de: https://exchangetuts.com/linux-shared-memory-shmget-vs-mmap-1639552984343707

Programmer Group(2021). Shared memory performance comparison of mmap, shm and MappedByteBuffer. Extraído de: https://programmer.group/shared-memory-performance-comparison-of-mmap-shm-and-mappedbytebuffer.html