TEC | Tecnológico de Costa Rica

Proyecto 2

Simulación Paginación / Segmentación

IC-6600: Principios de sistemas operativos

TEC Costa Rica, Cartago

Escuela de Computación, Ingeniería en Computación

I Semestre 2022 – 22 de mayo

Prof. Erika Marin Schumann, grupo 2

2019205716 - Jacob Guzmán Sánchez

2018102445 - Ary-El Durán Ballestero

2018189196 - Isaac Ortega Arguedas

TEC | Tecnológico de Costa Rica

Índice

Introducción

A la hora de ejecutar programas en nuestra computadora el sistema operativo se encarga de ordenar estos procesos en memoria para que los programas no utilicen direcciones de memoria no válidas las cuales podrían ocasionar un daño en nuestra computadora o que algún programa no se pueda ejecutar.

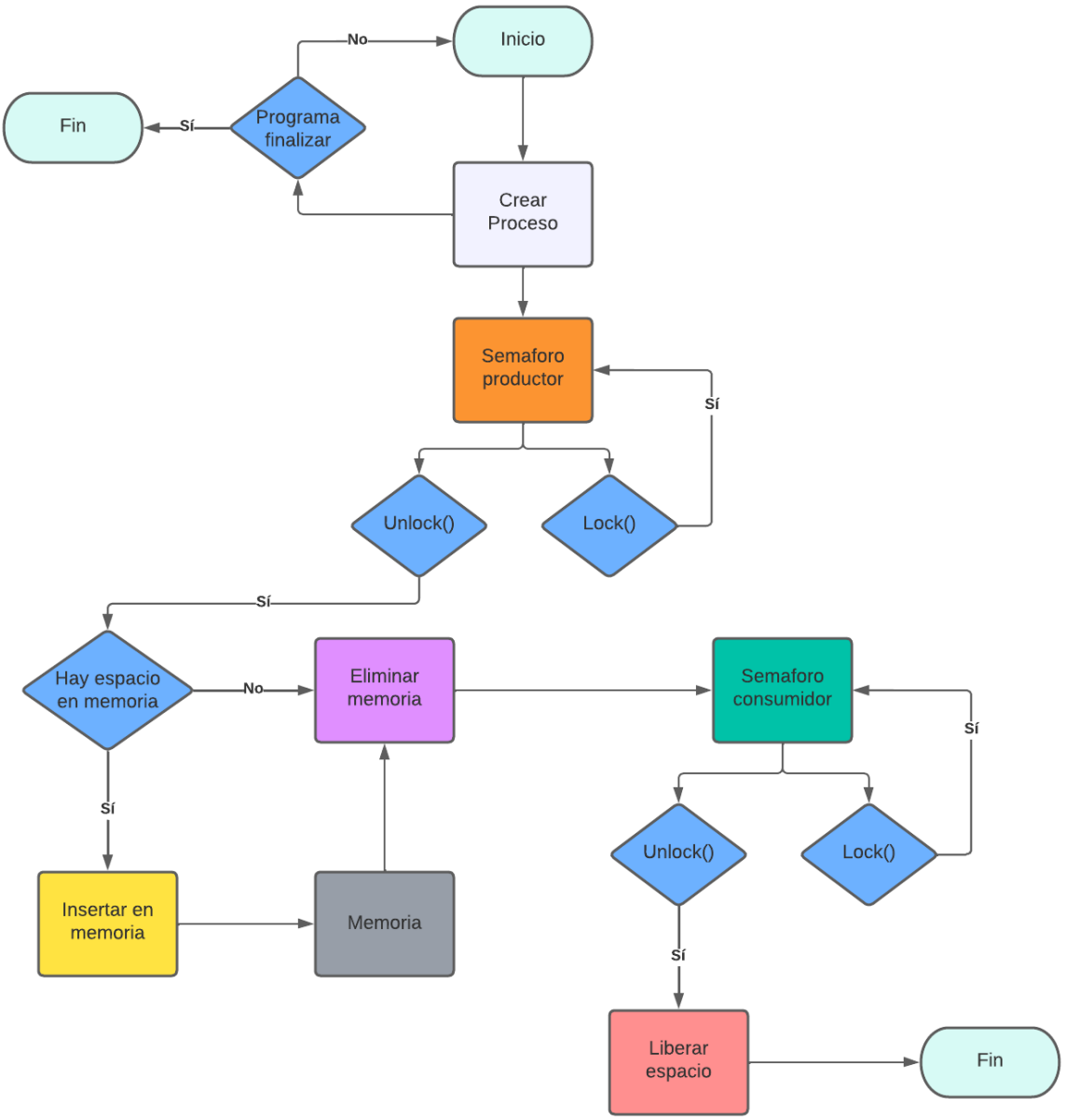
Existen dos estrategias para administrar la memoria de la mejor manera las cuales serían paginación y segmentación. Estas priorizan el buen manejo de la memoria por parte de los diferentes procesos. En nuestro proyecto debemos simular el ingreso de procesos a memoria y el buen manejo de estos; esto quiere decir, que no existan dos procesos luchando por el mismo recurso o procesos que accedan a direcciones de memoria no asignadas, también que dos procesos no traten de acceder a un mismo campo de memoria al mismo tiempo. Para lograr todo esto se desarrollarán las diferentes técnicas que vimos en clase como el buen direccionamiento de los procesos, el uso de los semáforos y simular la técnica de la paginación y segmentación de la mejor manera. De esta manera desarrollaremos un

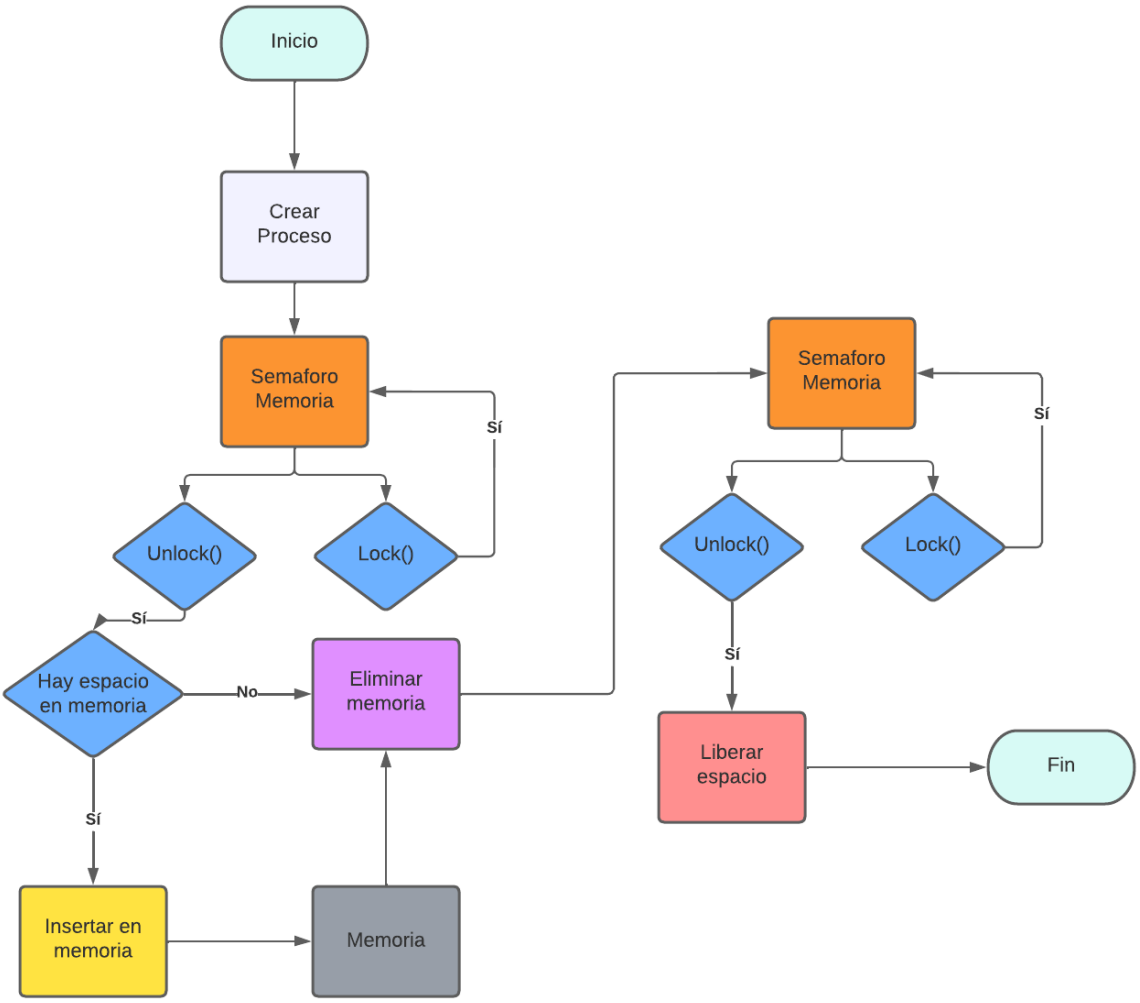
proyecto que simula los procesos en memoria pero con una buena sincronización de las partes con un uso óptimo de la memoria.

Por último, se llevará un registro de los estados del proceso para que el usuario sepa en todo momento qué acción se está realizando y pueda visualizar así el proceso dentro de la memoria.

Estrategia de solución

1. Semáforo que se utilizó y ¿por qué? El único semáforo que se utilizó en este proyecto son los de mutex presentes en la librería de “pthread.h”, este fue por dos razones específicas, uno porque como ya nos encontrábamos usando la librería para tomar ventaja de los threads y porque durante nuestra investigación de qué tipo de semáforo usar, hubieron recomendaciones de usar mutex sobre los semáforos de “semaphores.h” ya que, en nuestro caso, el mismo proceso que pide y obtiene el recurso debe ser el que lo devuelva.
2. Cómo se logra la sincronización: En el proyecto de la simulación de paginación o segmentación creamos procesos con diferentes atributos como Pid, tiempo, estado, entre otros. Cada proceso es un hilo, esto quiere decir que todos los procesos son independientes y una vez son creados se ponen en ejecución. Los procesos durante su ejecución primeramente van y buscar en memoria, sin embargo, si hay otro proceso buscando en memoria o saliendo de memoria se deben de bloquear hasta que la región crítica esté libre, esto se realiza usando el mutex. Una vez el proceso esté desbloqueado, busca en memoria y, cuando encuentre espacio, manda la señal para que otro proceso entre a la región crítica si no encuentra espacio se elimina. Una vez que este proceso haya terminado su ejecución el espacio que se encuentre usando durante este será sacado de la memoria, por ende debe bloquear de nuevo los procesos para que no entren a la región crítica y este pueda retirarse, una vez que se retire, los demás procesos pueden solicitar salir o entrar a memoria.


Inicio
No
Programa finalizar
Sí
Fin
Crear Proceso
Semaforo productor
Sí
Unlock()
Lock()
Sí
Hay espacio en memoria
No
Eliminar memoria
Semaforo consumidor
Sí
Sí
Insertar en memoria
Memoria
Unlock()
Lock()
Sí
Liberar espacio
Fin



Análisis de resultados

Programa Inicializador: Realización 100%

El programa inicializador le pregunta al usuario que tan grande quiere que sea la memoria compartida por bytes y luego la crea. Esto lo realiza de manera correcta.

Programa Productor de Procesos: Realización 100%

Este programa se encarga de la generación de procesos a intervalos de 30 a 60 segundos. Algunos datos de los procesos son generados de manera aleatoria, según las especificaciones dadas.

Programa Espia: Realización 100%

Se encarga de revisar los datos de los procesos generados hasta el momento de que es llamado. Imprime los datos recopilados en consola con el formato: PID, espacios de cada segmento o página, sus registros base y su estado actual.

Bitácora: Realización 100%

A medida que el programa se ejecuta se hacen entradas a un archivo de texto. Cada entrada es sobre el estado de un proceso.

Programa Finalizador: Realización 100%

Se encarga de finalizar el programa, si hubiese un hilo corriendo se asegura de terminar su ejecución.

Lecciones aprendidas

Primero que todo, se aprendió a utilizar la memoria compartida, este es un recurso que los tres no habíamos utilizado antes y nos da una nueva manera de manipular los recursos que usan diferentes procesos, en lugar de comunicarse con un txt o otro tipo de archivo que se use para la comunicación entre estos, tenemos la memoria compartida que nos permite crear este espacio que nos va a facilitar el intercambio de este tipo de información, en especial si no se necesita salvar la información después de que estos proceso ya hayan terminado de usar el espacio.

Otro recurso, que es el que hemos estado viendo mucho en clase es lo de los semáforos, aunque ya hemos tenido tiempo donde corremos procesos de manera concurrente, nunca habíamos necesitado el uso de algún tipo de bloqueo, cuál la verdad nos sorprende no haberlo visto antes porque esta información es increíblemente importante para el desarrollo de procesos concurrentes.

Por último están los método de guardar en la memoria, la paginación y la segmentación nos ayuda a ver como una memoria realmente va acomodando los procesos y cómo utiliza espacios que se encuentren separados por finalización de otros procesos nos da perspectiva de lo importante que es la desfragmentación.

Casos de pruebas

En esta parte seguirá el formato: número de prueba, cantidad de espacios de memoria que se asignaron y el tipo de manejo que se escogió. Se incluirá una imagen del programa espía y se agregan anexos con la bitácora de la respectiva prueba.

Un detalle a anotar es que durante las pruebas se realizó usando un tiempo específico que era más corto para la generación de procesos, esto fue solo para acelerar el proceso de pruebas, este esta de manera correcta en el código enviado.

Prueba #1: 10 bytes de memoria y paginación

```
------------------------------ PROGRAMA ESPIA ------------------------------
Memoria: 1000000000
Procesos acomodados en memoria: 13,0,0,0,0,0,0,0,0,0,
*** PID EN MEMORIA ***
[Proceso: 13,Espacios:  1,Registro:  0,Estado: EnMemoria]

*** PID BUSCANDO ENTRAR A MEMORIA ***
*** PID BLOQUEADOS ESPERANDO ***

*** PID MUERTO POR NO HABER ESPACIO ***
[Proceso: 2,Espacios:  1, 1, 1, 1, 1, 1, 1,Registro:  -1, 5, 6, 7, 8, 9, 0,Estado: Muerto sin espacio]

[Proceso: 4,Espacios:  1, 1, 1, 1, 1,Registro:  -1, 0, 0, 0, 0,Estado: Muerto sin espacio]

[Proceso: 5,Espacios:  1, 1, 1, 1, 1, 1, 1, 1,Registro:  -1, 1, 2, 3, 0, 0, 0, 0,Estado: Muerto sin espacio]

[Proceso: 6,Espacios:  1, 1, 1, 1, 1, 1, 1, 1, 1,Registro:  -1, 1, 2, 3, 0, 0, 0, 0, 0,Estado: Muerto sin espacio]

[Proceso: 9,Espacios:  1, 1, 1, 1, 1, 1,Registro:  -1, 0, 0, 0, 0, 0,Estado: Muerto sin espacio]

[Proceso: 12,Espacios:  1, 1, 1, 1, 1, 1, 1, 1,Registro:  -1, 1, 7, 8, 0, 0, 0, 0,Estado: Muerto sin espacio]

*** PID PROCESOS QUE TERMINARON ***
[Proceso: 1,Espacios:  1, 1, 1, 1,Registro:  0, 1, 2, 3,Estado: Ejecutado]

[Proceso: 3,Espacios:  1, 1, 1, 1, 1, 1,Registro:  4, 5, 6, 7, 8, 9,Estado: Ejecutado]

[Proceso: 7,Espacios:  1, 1,Registro:  0, 1,Estado: Ejecutado]

[Proceso: 8,Espacios:  1, 1, 1, 1, 1, 1, 1,Registro:  2, 3, 4, 5, 6, 7, 8,Estado: Ejecutado]

[Proceso: 10,Espacios:  1,Registro:  9,Estado: Ejecutado]

[Proceso: 11,Espacios:  1, 1, 1, 1, 1,Registro:  2, 3, 4, 5, 6,Estado: Ejecutado]
```

Prueba #2: 10 bytes de memoria y segmentación

```
-------------------------------- PROGRAMA ESPIA --------------------------------
Memoria: 1111111110
Procesos acomodados en memoria: 5,5,6,6,6,6,7,7,7,0,
*** PID EN MEMORIA ***
[Proceso: 5,Espacios:  2,Registro:  0,Estado: EnMemoria]

[Proceso: 6,Espacios:  2, 2,Registro:  2, 4,Estado: EnMemoria]

[Proceso: 7,Espacios:  1, 2,Registro:  6, 7,Estado: EnMemoria]

*** PID BUSCANDO ENTRAR A MEMORIA ***
*** PID BLOQUEADOS ESPERANDO ***

*** PID MUERTO POR NO HABER ESPACIO ***
[Proceso: 3,Espacios:  1, 1, 2,Registro:  -1, 8, 0,Estado: Muerto sin espacio]

[Proceso: 4,Espacios:  2, 2,Registro:  -1, 0,Estado: Muerto sin espacio]

[Proceso: 8,Espacios:  2, 1,Registro:  -1, 0,Estado: Muerto sin espacio]

*** PID PROCESOS QUE TERMINARON ***
[Proceso: 1,Espacios:  1, 2, 2,Registro:  0, 1, 3,Estado: Ejecutado]

[Proceso: 2,Espacios:  1, 1,Registro:  5, 6,Estado: Ejecutado]
```

Prueba #3: 100 bytes de memoria y paginación

```
-------------------------------- PROGRAMA ESPIA --------------------------------
Memoria: 11111111000000110001111111111111100000000000000000000000000000000000000000000000000000000000000000
Procesos acomodados en memoria: 8,8,8,8,9,9,9,9,0,0,0,0,0,7,7,0,0,0,6,6,6,6,6,7,7,7,7,7,7,7,9,9,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*** PID EN MEMORIA ***
[Proceso: 6,Espacios:  1, 1, 1, 1, 1,Registro:  19, 20, 21, 22, 23,Estado: EnMemoria]

[Proceso: 7,Espacios:  1, 1, 1, 1, 1, 1, 1, 1, 1,Registro:  14, 15, 24, 25, 26, 27, 28, 29, 30,Estado: EnMemoria]

[Proceso: 8,Espacios:  1, 1, 1, 1,Registro:  0, 1, 2, 3,Estado: EnMemoria]

[Proceso: 9,Espacios:  1, 1, 1, 1, 1, 1,Registro:  4, 5, 6, 7, 31, 32,Estado: EnMemoria]

*** PID BUSCANDO ENTRAR A MEMORIA ***
*** PID BLOQUEADOS ESPERANDO ***

*** PID MUERTO POR NO HABER ESPACIO ***
*** PID PROCESOS QUE TERMINARON ***
[Proceso: 1,Espacios:  1, 1, 1, 1, 1, 1, 1, 1,Registro:  0, 1, 2, 3, 4, 5, 6, 7,Estado: Ejecutado]

[Proceso: 2,Espacios:  1, 1, 1, 1, 1, 1,Registro:  8, 9, 10, 11, 12, 13,Estado: Ejecutado]

[Proceso: 3,Espacios:  1, 1,Registro:  14, 15,Estado: Ejecutado]

[Proceso: 4,Espacios:  1, 1, 1, 1, 1, 1, 1, 1,Registro:  0, 1, 2, 3, 4, 5, 6, 7,Estado: Ejecutado]

[Proceso: 5,Espacios:  1, 1, 1, 1, 1, 1, 1, 1, 1,Registro:  8, 9, 10, 11, 12, 13, 16, 17, 18,Estado: Ejecutado]
```

Prueba #4: 100 bytes de memoria y segmentación

```
-------------------------------- PROGRAMA ESPIA --------------------------------
Memoria: 11111110110000000110000000000000000000000000000000000000000000000000000000000000000000000000000000
Procesos acomodados en memoria: 14,15,14,14,15,15,16,0,12,12,0,0,0,0,0,0,14,14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,
*** PID EN MEMORIA ***
[Proceso: 12,Espacios:  1, 1,Registro:  8, 9,Estado: EnMemoria]

[Proceso: 14,Espacios:  2, 2, 1,Registro:  2, 17, 0,Estado: EnMemoria]

[Proceso: 15,Espacios:  1, 1, 1,Registro:  1, 4, 5,Estado: EnMemoria]

[Proceso: 16,Espacios:  1,Registro:  6,Estado: EnMemoria]

*** PID BUSCANDO ENTRAR A MEMORIA ***
*** PID BLOQUEADOS ESPERANDO ***

*** PID MUERTO POR NO HABER ESPACIO ***
*** PID PROCESOS QUE TERMINARON ***
[Proceso: 1,Espacios:  2,Registro:  0,Estado: Ejecutado]

[Proceso: 2,Espacios:  2, 2, 1,Registro:  2, 4, 6,Estado: Ejecutado]

[Proceso: 3,Espacios:  2,Registro:  7,Estado: Ejecutado]

[Proceso: 4,Espacios:  2, 2, 2,Registro:  0, 9, 11,Estado: Ejecutado]

[Proceso: 5,Espacios:  2,Registro:  13,Estado: Ejecutado]

[Proceso: 6,Espacios:  2, 2, 2,Registro:  2, 4, 6,Estado: Ejecutado]

[Proceso: 7,Espacios:  1, 2, 2, 2,Registro:  0, 8, 10, 15,Estado: Ejecutado]

[Proceso: 8,Espacios:  2, 2,Registro:  12, 17,Estado: Ejecutado]

[Proceso: 9,Espacios:  1, 1, 1,Registro:  1, 14, 19,Estado: Ejecutado]

[Proceso: 10,Espacios:  1, 2, 1,Registro:  0, 2, 4,Estado: Ejecutado]

[Proceso: 11,Espacios:  1, 1, 1,Registro:  5, 6, 7,Estado: Ejecutado]

[Proceso: 13,Espacios:  1, 1, 2, 2,Registro:  10, 11, 12, 15,Estado: Ejecutado]
```

TEC | Tecnológico de Costa Rica

Diferencia entre mmap y shmget

Los métodos mmap y shmget ambos crean segmentos de memoria compartida de cierta cantidad de bytes, sin embargo, el método mmap utiliza el disco real para guardar los procesos a diferencia del shmget que utiliza la memoria ram esto quiere decir que el mmap será un poco más lento en la lectura y escritura de procesos, pero tendrá un mayor espacio. En cambio con el shmget la velocidad de lectura y escritura será más rápida, pero tendrá menor espacio para crear una memoria física. La evaluación de los procesos utilizando Mmap() se realiza de forma perezosa siendo así solo cuando es necesario mediante que con shmget se realizan de forma ansiosa.
https://blog.fearcat.in/a?ID=00200-3220c97d-f600-4db2-a932-b047b1085fc9

Manual de usuario

Cuando se corre el archivo llamado “Proyecto.c”, se va a comenzar el mismo proyecto, este va a iniciar pidiéndole que tan grande va a querer la memoria compartida en cantidad de bytes, puede insertar lo que deseé, pero tiene que ser un número, ningún tipo de carácter o signo será aceptado. De ahí se iniciará la parte del programa que se encargará de reservar el espacio.

Luego de la creación de la memoria compartida se le va a pedir al usuario si quiere trabajar con paginación o segmentación, se escribe “1” si quiere trabajar con paginación y “2” si es segmentación lo que se desea utilizar. Una vez seleccionado no se puede cambiar este.

Los procesos se van a generar automáticamente y ya no se puede realizar nada para afectar su ejecución excepto terminar todo el programa.

El programa va a comenzar a ejecutarse, le va a presentar un menú que le dice que apriete “1” para terminar el programa y apreté “2” para utilizar el programa espía, este le va a presentar los estados de todos los procesos que han sido generados hasta el momento, al volver apretar “2” será actualizado para ver los cambios ocurridos durante la primera y segunda solicitud.

Al terminar el programa se va a generar un archivo llamado “Bitacora.txt” que va a contener todas las acciones que realizaron todos los procesos que fueron generados y se puede abrir para ver todos estos detalles.

Bitácora de trabajo

Consultas hechas entre el telegram de la profesora y Ary-El


May 4
AD
Disculpe profe, los procesos que se meten a memoria, donde se define el tamaño que tienen 5:13 PM
A la hora de crearlos 8:42 PM
00:17, 74.3 KB 8:42 PM
00:06, 27.5 KB 8:43 PM
May 18
AD
Disculpe profe, para el proyecto 2, se pueden usar mutex en lugar de semaforos verdad? 6:03 PM
Si 8:36 PM
May 20
AD
ultima pregunta, el programa espia debe trabajar con los locks, osea esperar a que los recursos sean soltados o puede entrar sin problema 7:04 PM
May 21
Al ser consulta podria entrar sin embargo pueda que cause algun problema. Los locks son buena idea 7:48 AM
May 23
AD
Profe, una pregunta, en la parte de estrategia de solución esta lo de que semáforo se utilizó, se refiere como tipo "semaphores.h" o mutex? En el trabajo escrito 12:35 PM
Ambas cosas 12:50 PM
AD
digamos las estrategias pueden ser tipo problema productor consumidor o se refiere a como logramos la sincronizacion 12:52 PM
A como lograron la sincronizacion 7:18 PM

TEC | Tecnológico de Costa Rica

Bibliografía

Sorber Jacob. [Jacob Sorber](30 de junio del 2020). *How to Set up Shared Memory in Your Linux and MacOS Programs. (shmget, shmat, shmdt, shmctl, ftok)* [Video]. Youtube. https://www.youtube.com/watch?v=WgVSq-sgHOc&ab_channel=JacobSorber

Sorber Jacob. [Jacob Sorber](25 de agosto del 2020). *What is a semaphore? How do they work?* [Video]. Youtube. https://www.youtube.com/watch?v=ukM_zzrIeXs&ab_channel=JacobSorber

Sorber Jacob. [Jacob Sorber](8 de enero del 2019). *Safety and Speed Issues with Threads. (pthreads, mutex, locks)* [Video]. Youtube. https://www.youtube.com/watch?v=9axu8CUvOKY&ab_channel=JacobSorber