UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Seminario de Solución de Problemas de Sistemas Operativos

Actividad 8

Profesor: Violeta del Rocío Becerra Velázquez
Abraham Baltazar Murillo Sandoval, INCO, 218744368
Jesús Uriel Guzmán Mendoza, INCO, 212451601
Sección D02, 2022-A

Actividad 8

Índice

Índice **2**
- Objetivo de la actividad 3
- Notas acerca del lenguaje 3
- TDA utilizados: 3
- ¿Qué hicimos para resolverlo? 4
- Conclusiones 5
- Video 5

Objetivo de la actividad

El objetivo de esta actividad es simular el algoritmo de planificación FCFS.

En el programa, se pedirán N procesos al usuario. La finalidad es que podamos realizar un programa sencillo en el cual se implementa el algoritmo de planificación FCFS que contenga 5 procesos como máximo en memoria contemplando al proceso que está en ejecución, los que están bloqueados y en la cola de listos.

Nosotros de manera aleatoria generamos los procesos.

Para simular las 4 instrucciones de interrupción (pasa el proceso a bloqueados), pausa, error (el proceso en ejecución termina), continuar; ingresaremos estos desde teclado haciendo uso de khit() y getch() que nos permitirán obtener el valor de estos en caso de haber presionado una tecla.

Notas acerca del lenguaje

Nosotros decidimos elegir el lenguaje de programación de C++, debido a que es el lenguaje de programación que ambos dominamos más y nos permite abstraer con claridad las operaciones que queremos conseguir

TDA utilizados:

- **FakeClock.** Es una estructura de datos que implementa un reloj falso, se utiliza para contabilizar el tiempo.
- **Operation.** La operación que se va a realizar. Contiene dos números: a y b, y un símbolo el cual representa la operación a realizar entre ambos números utilizados. Se valida que b sea diferente a 0 en el caso que el usuario ingrese a / b o a % b.
- **Process.** Tal como dice, contiene los datos que el usuario va a ingresar por consola como campos del proceso a ejecutar.
- **Processes.** Es un alias para la deque que se está utilizando para manejar el conjunto de procesos existentes.
- **Handler.** La estructura principal del programa. Contiene funciones para agregar procesos e imprimirlos. Este contiene la información del FCFS como 4 *Processes all, ready, blocked, finished* que son los que mantienen el algoritmo FCFS en conjunto con un *Process inExecution.* Las funciones principales son:
    - add(Process): Agrega un proceso a all
    - numOfProcessesInMemory(): Regresa cuantos procesos hay en memoria (ready + blocked + inExecution)
    - loadProcessesInMemory(): carga 5 procesos en memoria, desbloquea los que hayan terminando con su cuarentena y los agrega a la cola de

listos, obtiene más procesos de la lista de todos los procesos en caso de hacer falta y tener espacio para más.
    - updateTime(): Como nuestro tiempo es falso (sólo cuentas), aquí actualizamos el tiempo de todos los objetos que necesitan avanzar en su tiempo.
    - print(): Imprime todo lo que se encuentre en nuestro programa.
    - solve(): Realiza el algoritmo para manipular los procesos usando FCFS.
- **C++ queue.** Implementa una cola, en la que el primer elemento es el primero en salir. Esto se utilizó para implementar la cola de lotes, ya que se ejecutarán en el mismo orden que fueron ingresados por el usuario.
- **Random:** Esta clase genera números aleatorios en un rango [l, r] o dada una string, selecciona un carácter de estos.
- **VariadicTable:** Esta estructura fue tomada de un repositorio de github (https://github.com/friedmud/variadic_table) y nos permite imprimir tablas en C++ ya que esto no está disponible nativamente en el lenguaje, lo anterior para darle una mejor presentación al programa.

¿Qué hicimos para resolverlo?

Primero que nada, se utilizó una librería llamada <conio.h> para poder utilizar los métodos de khbit y getch, esto para poder detectar las interrupciones y así poder hacer control sobre los estados del programa tales como: interrumpir un proceso, mandar a programas finalizados un proceso con un error, pausar el programa, y continuar el programa.

Para calcular los tiempos, se utilizaron los siguientes lineamientos:

Tiempo de llegada: Hora en la que el proceso entra al sistema.

Tiempo de Finalización: Hora en la que el proceso terminó.

Tiempo de Retorno: Tiempo total desde que el proceso llega hasta que termina.

Tiempo de Respuesta: Tiempo transcurrido desde que llega hasta que es atendido por primera vez.

Tiempo de Espera: Tiempo que el proceso ha estado esperando para usar el procesador.

Tiempo de Servicio: Tiempo que el proceso ha estado dentro del procesador.

Estos tiempos se imprimieron al final haciendo uso de la librería externa de *Variadic Table* para imprimir en forma de tabla. Todos los tiempos se hicieron acorde al reloj global que utilizamos en el programa.

Además modularizamos el código para tener un generador de Procesos e insertarlos en nuestra cola donde están todos los procesos pendientes con presionar la tecla ‘n’ y con la tecla ‘t’ mostramos una tabla con los datos de todos los procesos existentes, en donde además incorporamos el estado actual del proceso y su resultado en caso de estar terminado.

```
Presionado e
-------------------------------------------------------------------------------------------------------------------------------------
| Id | Estado del proceso  | Operación y datos | Tiempo de llegada | Tiempo de finalización | Tiempo de retorno | Tiempo de espera | Tiempo de servicio | Tiempo restante en CPU | Tiempo de respuesta |
-------------------------------------------------------------------------------------------------------------------------------------
|  7 | Nuevo               | 33 % 65 = ?       | -                 | -                      | -                 | -                | -                  | -                      | -                   |
|  8 | Nuevo               | 43 % 21 = ?       | -                 | -                      | -                 | -                | -                  | -                      | -                   |
|  9 | Nuevo               | 66 % 77 = ?       | -                 | -                      | -                 | -                | -                  | -                      | -                   |
|  5 | Nuevo               | 61 - 50 = ?       | 0                 | -                      | -                 | -                | -                  | -                      | -                   |
|  6 | Nuevo               | 33 * 2 = ?        | 0                 | -                      | -                 | -                | -                  | -                      | -                   |
|  0 | Bloqueado           | 49 - 31 = ?       | 0                 | -                      | -                 | -                | -                  | 13                     | 0                   |
|  1 | Bloqueado           | 91 / 65 = ?       | 0                 | -                      | -                 | -                | -                  | 7                      | 0                   |
|  2 | Terminado con error | 63 / 16 = Error   | 0                 | 0                      | 0                 | 0                | 0                  | -                      | 0                   |
|  3 | Terminado con error | 15 / 34 = Error   | 0                 | 0                      | 0                 | 0                | 0                  | -                      | 0                   |
|  4 | En ejecución        | 77 % 44 = ?       | 0                 | -                      | -                 | -                | -                  | -                      | 0                   |
-------------------------------------------------------------------------------------------------------------------------------------
```

Conclusiones

En esta práctica, aprendimos sobre el algoritmo de planificación FCFS (First Come First Served), cómo crear nuevos procesos aleatorios, y cómo visualizar una tabla en la que el programa se pausara y visualizara el BCP de todos los procesos, así como poder regresar al estado anterior presionando la tecla “C”. Nos pareció muy interesante la práctica ya que pudimos aprender cómo mantener dos pantallas al mismo tiempo sin perder la información, algo que nos habíamos preguntado cómo se hacía desde hace tiempo.

Video

8. Algoritmo de planificación FCFS continuación.mp4