Tecnológico de Costa Rica

Ing. En Computación - Principios de Sistemas Operativos

Profesora:
Erika Marín Schumann

Estudiantes:
Carlos Girón Alas 2014113159
Julián J. Méndez Oconitrillo 2014121700
Jasson Moya Álvarez 2014082335

Planificador de CPU

Fecha de entrega: Domingo 25 de setiembre

Tabla de Contenidos:

1. Manual de compilación

Es importante tener en cuenta que este proyecto fue desarrollado en Linux, Ubuntu 16.04 en lenguaje C. Los pasos para la compilación se muestran a continuación. Adjunto a la carpeta está el archivo makefile que tiene las instrucciones del compilador para que se generen los archivos .o y ejecutable. Como el proyecto es una conexión entre servidor y cliente, se proporcionan 3 archivos ejecutables. Un servidor principal que se genera del archivo server.c, un servidor adjunto que muestra los resultados: server_results.c y un archivo client.c que tiene la responsabilidad de generar procesos y enviarlos mediante threads y a través de un socket hacia los servidores.

En la carpeta del Proyecto I -> Ejecutable, está el archivo ejecutable de la aplicación. Sin embargo, es recomendable que se corra como se explica a continuación.

1.1 Para correr los ejecutables:

1. Abra el terminal montando la carpeta en la que se encuentran los 3 archivos. Repita este proceso 3 veces, u opcionalmente pueden correrse 3 terminales en 3 equipos diferentes, o preferiblemente 2 terminales en un equipo (ambos servidores) y la otra en un segundo equipo que correrá al cliente. Cada terminal ejecutará una de las partes del proyecto.
2. Para ejecutar el servidor principal, corra *./server* en una terminal.
3. Para ejecutar el servidor de resultados, corra *./server_results* en otra terminal.
4. Para ejecutar el cliente (preferiblemente en un equipo diferente al que corre los otros dos servidores), utilice el comando *./client.*
5. La forma más rápida de montar las carpetas en este proyecto es dar click derecho en la carpeta donde se encuentran los archivos y elegir Open in terminal o Abrir en terminal.

1.2 ¿Cómo compilar los archivos ejecutables?

El proyecto se desarrolló haciendo uso del archivo makefile adjunto.

1. Para el servidor general se corre *make server*
2. Para el servidor de resultados se corre *make server_results*
3. Para el cliente se corre *make client*

1.3 Sobre la ejecución.

Cada uno de los archivos o se ejecuta de manera automática, o presenta opciones que se describen a continuación:

1. **Server:** Aquí se encuentra la lógica principal del programa, y se ejecuta con 4 algoritmos de planeación diferentes.
    a. La opción 1. First In First Out
    b. La opción 2. Shortest Job First
    c. La opción 3. Highest Priority First
    d. La opción 4. Round Robin, que a su vez pide una especificación del quantum deseado para la ejecución del algoritmo.

2. **Server Results:** Esta ejecución no requiere de ninguna elección.
3. **Cliente:** Aquí se debe escoger únicamente entre los dos modos 1. Manual y 2. Automático que se diferencian en que el manual lee los procesos del archivo processes.txt para enviarlos al servidor, mientras que automático genera de forma aleatoria procesos para ser enviados y procesados.

2. Casos de prueba

2.1 Cliente.

Caso de prueba	**Comportamiento esperado.**	**Se cumple**
1. Conexión de socket entre el cliente y el servidor.	El programa del cliente se conecta exitosamente con el programa del servidor mediante el puerto 9090.	Sí
2. Creación de procesos manual.	Se lee el archivo .txt con la información de procesos.	Sí
3. Lectura de archivo inválido.	Se muestra un mensaje de error si el archivo no es válido.	Sí
4. Creación de procesos automáticos.	Se crean procesos aleatoriamente y de manera indefinida.	Sí
5. Problemas de conexión con el servidor.	Muestra el mensaje de error cuando la conexión del socket no se logra por diferentes causas.	Sí
6. Creación de Threads para cada proceso.	Los procesos tienen un thread propio que empieza a la hora de nacer el proceso, y termina cuando se envía al servidor.	Sí
7. Envío de procesos por socket.	Cada uno de los procesos son enviados por el socket hacia el servidor.	Sí

2.2 Servidor.

Caso de prueba	**Comportamiento esperado.**	**Se cumple**
1. Conexión con el cliente.	El servidor se conecta exitosamente con el cliente mediante el puerto 9090.	Sí
2. Error en conexión con el cliente.	El servidor envía un mensaje con el error a la hora de conectar con el cliente.	Sí
3. Recibir procesos del cliente.	Se reciben los procesos del cliente de manera exitosa.	Sí
4. Procesos en cola de Job Scheduler.	Al recibir un proceso del cliente, se ingresa en la cola del Job Scheduler.	Sí
5. Cola del Job Scheduler llena.	Cuando la cola del Job Scheduler está llena, se presenta un mensaje correspondiente, y los procesos sin campo son desechados.	Sí
6. Proceso fuera de la cola del Job Scheduler.	El CPU Scheduler recoge un proceso de la cola del Job Scheduler exitosamente y se libera un campo en la cola del Job Scheduler.	Sí
7. Ejecución general de un proceso en el CPU Scheduler.	Se ejecuta un proceso recogido por el Job scheduler de manera exitosa, en el CPU Scheduler-	Sí
8. Elección de algoritmo para ejecutar en el CPU Scheduler.	Se presenta un menú y se puede seleccionar un algoritmo para la ejecución del CPU Scheduler.	Sí
9. Ejecución de FIFO (First In First Out).	Se ejecuta exitosamente el algoritmo seleccionado como FIFO.	Sí
10. Ejecución de SJF (Shortest Job First).	Se ejecuta exitosamente el algoritmo seleccionado como SJF.	Sí
11. Ejecución de HPF (Highest Priority First).	Se ejecuta exitosamente el algoritmo seleccionado como HPF.	Sí
12. Ejecución de RR (Round Robin) con quantum especificado.	Se ejecuta exitosamente el algoritmo seleccionado como RR, y se puede seleccionar un quantum específico.	Sí
13. Se muestra una tabla con los procesos actuales en la cola del Job Scheduler.	Al presionar la tecla “d”, se presenta un tabla con todos los procesos que están en el momento en la cola del Job Scheduler.	Sí
14. Se detiene el trabajo del CPU Scheduler.	Al presionar la tecla “Esc”, el procesamiento del CPU Scheduler termina, y se presenta una tabla con los resultados del proceso.	Sí

15. Se guardan los procesos ejecutados.	Al final del proceso, se guarda en un archivo .txt todos los procesos que han sido ejecutados.	Sí
16. Se calcula el TAT (Turn Around Time) para todos los procesos ejecutados.	Al final, se muestra el TAT para los procesos ejecutados por el CPU Scheduler.	Sí
17. Se calcula el WT (Waiting Time) para todos los procesos ejecutados.	Al final, se muestra el WT para los procesos ejecutados por el CPU Scheduler.	Sí
18. Se mandan los procesos al servidor de resultados.	Cada vez que se termina de ejecutar un proceso, se envía mediante un socket en el puerto 9080, el resultado de ese proceso.	Sí

2.3 Servidor de Respuesta

Caso de prueba	**Comportamiento esperado.**	**Se cumple**
1. El servidor se conecta con el servidor principal.	La conexión entre el servidor de respuesta y el servidor principal es exitosa.	Sí
2. Error en la conexión entre el servidor y el servidor principal.	Se muestra un mensaje de error cuando la conexión entre el servidor de respuesta y el servidor principal no es exitosa.	Sí
3. Entrada de procesos desde el servidor principal.	Se reciben los procesos terminados enviados por el servidor principal.	Sí
4. Demostración de procesos en la consola.	Se muestra la información recibida por el servidor principal en la consola.	Sí
5. Se termina el proceso.	Cuando el servidor deja de enviar procesos, el servidor de respuesta termina su labor.	Sí

3. Análisis de resultados

Objetivo	Éxito
1. El programa realiza correctamente las conexiones de sockets y envíos de datos en los puertos definidos.	100%
2. El programa despliega el mensaje de error apropiado cuando las conexiones no son posibles o terminan de forma abrupta.	100%
3. El programa valida correctamente la existencia de los archivos de procesos y de resultados y devuelve el mensaje de error apropiado cuando no están disponibles.	100%
4. El programa tiene implementado correctamente el modo manual para el cliente y lee correctamente del archivo processes.txt siempre que las entradas de dicho archivo tengan el formato correcto.	100%
5. Ante la presencia de errores en el archivo processes.txt, se desvalida la línea con el error.	100%
6. El programa envía correctamente los datos de los procesos leídos en modo manual y generados en modo automático.	100%
7. El programa genera correctamente los números aleatorios en los rangos establecidos para la ejecución (PID, bursts, prioridades).	100%
8. El servidor escucha los mensajes entrantes del socket en el puerto establecido y procesa correctamente los datos entrantes.	100%
9. Los menús de los 3 ejecutables no permiten opciones diferentes a las mostradas en pantalla y devuelven los mensajes de error adecuados cuando los input no son correctos.	100%
10. El programa implementa los 4 algoritmos solicitados: FIFO, SJF, HPF y Round Robin con quantum parametrizable.	100%
11. El programa en FIFO trabaja los procesos de acuerdo a su PID.	100%
12. El programa en SJF compara los procesos pendientes en la cola de Ready de acuerdo a la duración de su burst.	100%
13. El programa en HPF compara los procesos pendientes en la cola de Ready de acuerdo a la prioridad (busca el número menor).	100%
14. El programa en Round Robin ejecuta los procesos de acuerdo a su PID (FIFO) sólo durante el tiempo asignado en el quantum.	100%
15. El programa descarta los procesos que no pudieron entrar a la cola de Ready pues la misma estaba llena.	100%
16. El programa informa al servidor de resultados cada vez que termina un proceso en ejecución.	100%

17. El servidor muestra el estado actual de la cola al presionar la tecla d o D para “display”	100%
18. El servidor termina al presionar la tecla ESC	100%
19. Al terminar la ejecución, se muestran los resultados promedio de TAT, WT y el resultado final de cantidad de procesos terminados así como la duración total del CPU ocioso.	100%
20. Los procesos terminados se almacenan en el archivo results.txt con sus datos respectivos.	100%