



Instituto Tecnológico de Costa Rica

Escuela de Computación

Principios de Sistemas Operativos GR2

Proyecto #1

Planificador del CPU

Profesora:

Erika Marin Schumann

Integrantes

Moisés Higuerey Hernández (2018092411)

Josue Alvarado Mares (2018143069)

Valeria Quesada Benavides (2018085308)

Fecha de Entrega:

12/4/2021

I Semestre 2022

Estrategia de Solución	3
Tipo de solución implementada	3
Comunicación cliente-servidor (Sockets)	3
Cliente Manual	3
Cliente Automatico	3
Procesos y PCB	3
Colas	4
Job Scheduler	5
CPU Scheduler	5
Algoritmos	5
Comandos	6
Menús	6
Análisis de Resultados	7
Lecciones Aprendidas	8
Moises Higuerey:	8
Josue Alvarado:	8
Valeria Quesada:	8
Casos de Pruebas	10
Conexión socket cliente-servidor	10
Cliente Manual	11
Cliente Automático	12
Algoritmos	13
FIFO	13
SJF	15
HPF	17
Round Robin ($q = 5$)	19
Cálculos	23
TAT	23
WT	23
Cantidad de procesos ejecutados	24
Comandos	25
Quit	25
Print cola de ready	26
Comparación	27
Pthread C	27
Threads Java	27
Manual de Usuario	28
Bitácora	38

Estrategia de Solución

Tipo de solución implementada

Se utilizó el “divide y vencerás”, por lo que se dividieron tareas por participantes y días. Estas se consideraban cortas debido a que se consideraba que no eran de más de dos horas diarias, por lo que las reuniones se intentaban realizar a cada dos o tres días.

Comunicación cliente-servidor (Sockets)

Para el cliente, se realizó una implementación simple, que busca una conexión a un servidor en el puerto 8080, por medio del cual se comunicaría para la transferencia de datos.

Para el servidor, la comunicación por medio de sockets se realiza dentro del Job Scheduler, donde hay un ciclo por el cual en cada iteración se comunica con un cliente y procesa la información recibida. Al terminar la iteración, vuelve a esperar un cliente nuevo.

Cliente Manual

El cliente manual lee datos de un archivo línea por línea. Cada línea del archivo son dos números separados por un espacio, el primero es el burst y el segundo es la prioridad. Utilizando los datos del archivo, se envía la información al servidor línea por línea, con un sleep de duración aleatoria (3-8 segundos). Para el envío de procesos, se crea un hilo por cada proceso a enviar.

Cliente Automatico

La implementación es similar al manual. Como únicas diferencias, los procesos son creados de manera aleatoria (rangos de 1-5 para burst y prioridad) y el sleep entre procesos es un valor definido por el usuario mediante la consola.

Procesos y PCB

Los procesos son representados en nuestro programa mediante nodos de una lista enlazada. La estructura del nodo se puede ver en la siguiente imagen:

```

struct PCB
{
    int pId;
    int burst;
    int priority;
    int timeExecute;
    int startTime;
    int endTime;
};

```

El PCB guarda el pId, burst, y prioridad como datos básicos. El PCB también tiene el tiempo ejecutado, que es utilizado para guardar el progreso de ejecución cuando se utiliza el algoritmo Round Robin. También se guardan el tiempo de entrada y salida para calcular el TAT.

Los nodos que guardan el PCB tienen la siguiente estructura:

```

struct Node
{
    struct Node *next;
    struct PCB process;
};

```

Para formar la lista enlazada, primero se tiene un puntero al siguiente nodo. Además se guarda el PCB correspondiente al nodo.

Colas

El manejo de los procesos se hace mediante dos colas; una cola de Ready y otra de Done. La cola Ready guarda los procesos que aún faltan por ejecutar, mientras la cola Done guarda los procesos que terminaron. La estructura utilizada para las colas se puede observar en la siguiente imagen:

```

struct Queue
{
    struct Node *first;

    struct Node *last;
};

```

Las colas simplemente guardan el primer nodo de la lista, junto al último nodo para poder realizar inserciones de manera rápida. Se tienen implementadas varias funciones para el manejo de las colas y sus procesos. Se tiene inserción, eliminación (que en realidad es una transferencia de la cola de ready a la de done) y búsqueda (por id, por burst, por prioridad).

Job Scheduler

El Job Scheduler se encarga de la comunicación con el cliente. El Job Scheduler espera a los datos enviados por el cliente, los cuales utiliza para crear los PCBs y nodos a insertar en la cola de Ready. El Job Scheduler tiene un ciclo de conexión con el cliente, dentro del cual hay otro ciclo de comunicación y recepción de datos. Hasta que el cliente cierre la conexión el Job Scheduler sigue obteniendo los datos mandados por el cliente. Cuando finaliza la conexión, sigue una nueva iteración en donde vuelve a esperar un cliente. Cada vez que llegan los datos de un proceso, se crea un nuevo hilo para crear el proceso e insertarlo en la cola.

CPU Scheduler

El CPU Scheduler funciona mediante el uso de ciclos que verifican el estado de la cola de Ready constantemente, para saber si hay nodos que ejecutar o no. El CPU Scheduler es un hilo propio, en donde primero revisa mediante un switch el algoritmo que va a ejecutar. Después de entrar al caso de uso correspondiente al algoritmo escogido por el usuario, el CPU Scheduler se encicla, revisando la cola de Ready y simulando la ejecución de procesos mediante llamadas a la función del algoritmo escogido. En el caso de que se tengan procesos en la cola del Ready, se realiza la ejecución normalmente. En el caso de que no hayan procesos listos, se incrementa un contador global que cuenta el tiempo del cpu ocioso, además de hacer un sleep por un segundo. El CPU Scheduler sigue corriendo hasta que la simulación sea terminada por el usuario.

Algoritmos

FIFO: El algoritmo fifo fue implementado de una manera simple. Cada vez que la función de FIFO es llamada, el algoritmo busca el primer proceso de la cola de Ready y lo ejecuta, pasando el nodo a la cola de Done.

SJF: Para este algoritmo, se utiliza una función de manejo de colas que busca el nodo con el menor burst dentro de la cola de Ready. Cada vez que la función de SJF es llamada, se busca el nodo con el menor burst existente, lo ejecuta y lo pasa a la cola de Done.

HPF: Similarmente al algoritmo SJF, se utiliza una función de manejo de colas, está buscando el nodo con la mayor prioridad (menor número = más prioridad). Cada vez que la función de HPF es llamada, se busca el nodo más prioritario de la cola de Ready y se ejecuta, transfiriendo el nodo a la cola de Done.

Round Robin: La implementación de este algoritmo tiene más complicaciones. Para este algoritmo, el CPU Scheduler tiene que guardar nodo que va a ejecutar y después preparar el siguiente nodo para la próxima iteración de RR. En caso de que no se tenga un siguiente nodo, se guarda el pld del primer nodo para la próxima iteración. En el caso de que en la

siguiente iteración no se encuentre un nodo con el pld guardado, pero si haya nodos en la cola, se vuelve a asignar el pld al primer nodo. Con este proceso, se realizan iteraciones a través de todos los nodos de la lista, en orden, ejecutando cada proceso con el quantum asignado por el usuario. A diferencia de los otros algoritmos, los procesos que terminan son transferidos en la función del CPU Scheduler, para poder guardar el pld del siguiente proceso.

Comandos

Los comandos fueron implementados mediante el uso de input de usuario. En cualquier momento, el usuario puede ingresar una “q” o una “p” en la consola, para terminar la simulación o imprimir la cola de Ready respectivamente. La manera en que se implementó esto fue utilizando ciclos que verifican caracteres que son ingresados por el usuario.

Menús

Para el menú de cliente, se tiene la opción de escoger cliente automático o cliente manual. Al escoger el cliente manual, se pregunta por el nombre del archivo a leer. Al escoger el cliente automático, se pregunta por el tiempo de sleep entre procesos.

Para el menú de server, se tiene como opciones los cuatro algoritmos implementados. Como opción adicional al escoger Round Robin, se le pregunta al usuario el quantum a utilizar.

Análisis de Resultados

#	Tarea	Prioridad	% Completado	Listo
1	Cliente			●
1.1	Cliente Manual	HIGH	100%	✓
1.1.1	Lectura de archivo	MEDIUM	100%	✓
1.1.2	Thread por proceso y sleep	LOW	100%	✓
1.1.3	Envio por socket	HIGH	100%	✓
1.2	Cliente Automatico	HIGH	100%	✓
1.2.1	Generación de randoms	MEDIUM	100%	✓
1.2.2	Thread por proceso y sleep	LOW	100%	✓
1.2.3	Envio por socket	HIGH	100%	✓
1.3	Notificacion de proceso recibido	LOW	100%	✓
2	Server			●
2.1	Selección de algoritmo	LOW	100%	✓
2.2	Job Scheduler	HIGH	100%	✓
2.2.1	Recibir información por socket	HIGH	100%	✓
2.2.2	Generacion de PID	HIGH	100%	✓
2.2.3	Envio de mensaje de confirmación al cliente	LOW	100%	✓
2.2.4	Creación de PCB	HIGH	100%	✓
2.2.5	Creación de proceso	HIGH	100%	✓
2.3	CPU Scheduler	HIGH	100%	✓
2.3.1	Verificacion de cola	MEDIUM	100%	✓
2.3.2	Ejecucion de procesos por algoritmo (FIFO, SJF, HPF, RR)	HIGH	100%	✓
2.3.3	Print del context switch	LOW	100%	✓
2.3.4	Print de proceso terminado	LOW	100%	✓
2.3.5	Sleep y CPU ocioso	LOW	100%	✓
2.4	Consulta de cola de Ready	MEDIUM	100%	✓
2.5	Detener simulación	MEDIUM	100%	✓
2.6	Despliege de info final	MEDIUM	100%	✓

Lecciones Aprendidas

Moises Higuerey:

- Se aprendió a utilizar las librerías de *pthread*s para el manejo de hilos en el ambiente de programación C en sistemas operativos basados en Linux.
- Se aprendió el manejo de conexiones mediante sockets en el ambiente de Linux, utilizado en librerías de C, con comunicación cliente-servidor bidireccional.
- Se aprendió a utilizar hilos para manejar procesos simultáneos que comparten recursos, en el ambiente de Linux en C.
- Se obtuvo un mayor conocimiento sobre el comportamiento de los algoritmos FIFO, SJF, HPF y Round Robin en un ambiente aproximado a la realidad mediante simulaciones.
- Se desarrollaron las habilidades de trabajo en equipo con el apoyo de herramientas de colaboración como repositorios remotos (Github) y reuniones en línea.

Josue Alvarado:

- Se aprendió sobre el desarrollo de manera remota haciendo trabajo sincrónico (reuniones) y asincrónico con el apoyo de herramientas como Liveshare.
- Se recordó el cuidado necesario en el manejo de punteros propio del lenguaje C.
- Se realizaron reuniones compartiendo distintos puntos de vista sobre posibles soluciones a problemas encontrados.
- Se obtuvo de manera extensa y concisa sobre el cuidado de la memoria en el intercambio de información entre sockets.
- Se aprendieron habilidades con el uso de varias librerías como “pthread”, “socket”, “time” entre otras.

Valeria Quesada:

- Se aprendió sobre el manejo de los sockets, la forma en la que deben ser asignados y el cuidado que se debe tener con los IP privados y los puertos, ya que si uno de estos es de carácter privado, otra computadora no puede acceder a esto.
- Se aprendió sobre el manejo de los hilos, el cuidado que se debe tener debido a que si se encuentran en un loop, se tienden a enciclar.
- Se obtuvo conocimiento sobre el manejo del teclado (IO) y el cuidado que se debe tener.
- Se aprendió sobre el trabajo grupal, el ayudar a otros a solucionar sus errores puede ahorrarle tiempo a todo el grupo.
- Se obtuvo conocimiento sobre el manejo de Linux, ya que nunca antes había sido utilizado, lo que requirió organización y tiempo.

- Se aprendió sobre la importancia de realizar consultas, quien lee puede tener una perspectiva diferente a quien escribió el documento, por lo que la comunicación es importante.

Casos de Pruebas

Conexión socket cliente-servidor

Objetivo de la prueba

Revisión de la conexión entre sockets.

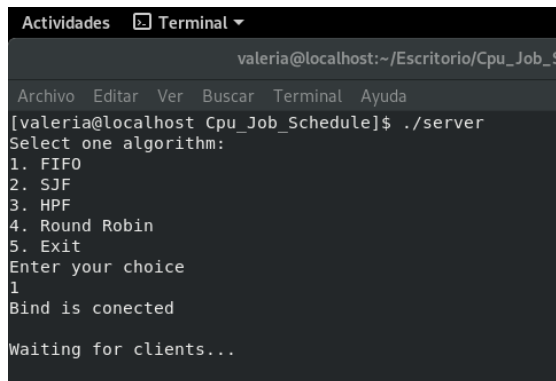
Lo esperado

Al correr el servidor muestra un menú en donde se seleccionará el número, posteriormente el bind está conectado y esperando por clientes.

Al correr el cliente se muestra un mensaje en el servidor indicando que la conexión se inició. En el cliente se encontrará un menú.

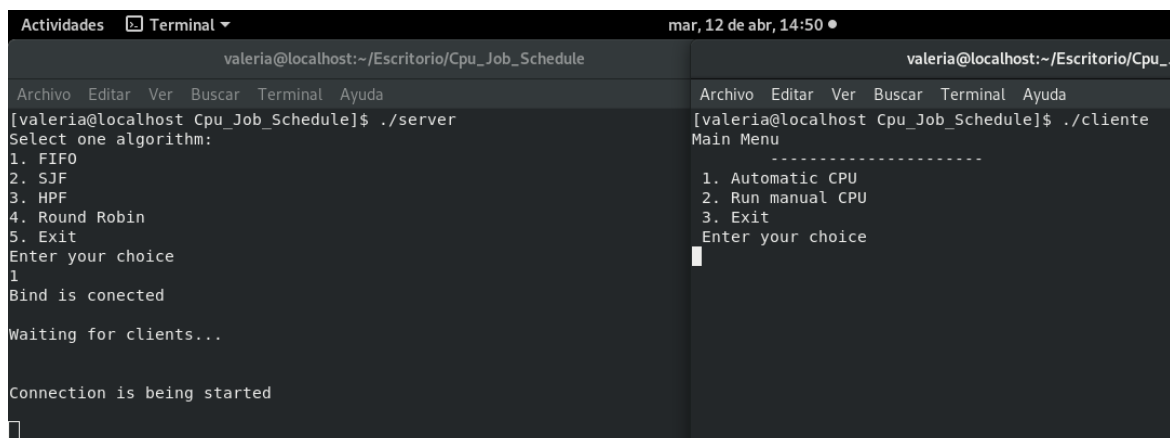
Resultado

Al correr el servidor muestra un menú en donde se seleccionará el número, posteriormente el bind está conectado y esperando por clientes.



```
Actividades Terminal
valeria@localhost:~/Escritorio/Cpu_Job_Schedule
Archivo Editar Ver Buscar Terminal Ayuda
[valeria@localhost Cpu_Job_Schedule]$ ./server
Select one algorithm:
1. FIFO
2. SJF
3. HPF
4. Round Robin
5. Exit
Enter your choice
1
Bind is conected
Waiting for clients...
```

Al correr el cliente se muestra un mensaje en el servidor indicando que la conexión se inició. En el cliente se encontrará un menú.



```
Actividades Terminal mar, 12 de abr, 14:50
valeria@localhost:~/Escritorio/Cpu_Job_Schedule
Archivo Editar Ver Buscar Terminal Ayuda
[valeria@localhost Cpu_Job_Schedule]$ ./server
Select one algorithm:
1. FIFO
2. SJF
3. HPF
4. Round Robin
5. Exit
Enter your choice
1
Bind is conected
Waiting for clients...
Connection is being started

valeria@localhost:~/Escritorio/Cpu_Job_Schedule
Archivo Editar Ver Buscar Terminal Ayuda
[valeria@localhost Cpu_Job_Schedule]$ ./cliente
Main Menu
-----
1. Automatic CPU
2. Run manual CPU
3. Exit
Enter your choice
```

Cliente Manual

Objetivo de la prueba

Leer de manera correcta el archivo solicitado. Es decir, sin cambiar números de burst y prioridad.

Lo esperado

Se lee el archivo que el cliente ingresó, el burst y la prioridad obtenidas por el sistema son los mismos encontrados en los archivos.

La lectura del archivo tiene un random con el sleep de 3 a 8 segundos entre procesos.

Los hilos de los procesos se crean y envían la información 2 segundos después.

Resultado

Lee el archivo que el cliente ingresó, el burst y la prioridad obtenidas por el sistema son los mismos encontrados en los archivos.

La lectura del archivo cumple el rango del aleatorio (3-8), obteniendo 4, 6 y 8.

Los hilos de los procesos se crean y envían la información 2 segundos después. Se demuestra por la resta del “Sent at” - “Created at” en cliente.

Actividades	Terminal
valeria@localhost:~/Escritorio/Cpu_Job_Schedule	valeria@localhost:~/Escritorio/Cpu_Job_Schedule
Archivo Editar Ver Buscar Terminal Ayuda	Archivo Editar Ver Buscar Terminal Ayuda
[valeria@localhost Cpu_Job_Schedule]\$./server	2
Select one algorithm:	Write file name:
1. FIFO	test.txt
2. SJF	
3. HPF	
4. Round Robin	Created at: 15:03:31
5. Exit	Burst: 1
Enter your choice	Priority: 5
1	Process id: 1
Bind is connected	Timeout to create next process: 8
	Sent at: 15:03:33
Waiting for clients...	Created process with PID #1
Connection is being started	Created at: 15:03:39
	Burst: 2
New process at: 15:03:33	Priority: 3
Burst: 1	Process id: 2
Priority: 5	Timeout to create next process: 6
Process ID: 1	Sent at: 15:03:41
	Created process with PID #2
New process at: 15:03:41	Created at: 15:03:45
Burst: 2	Burst: 9
Priority: 3	Priority: 3
Process ID: 2	Process id: 3
	Timeout to create next process: 4
	Sent at: 15:03:47
New process at: 15:03:47	Created process with PID #3
Burst: 9	
Priority: 3	Created at: 15:03:49
Process ID: 3	Burst: 10
	Priority: 2
	Process id: 4
New process at: 15:03:51	Timeout to create next process: 8
Burst: 10	Sent at: 15:03:51
Priority: 2	Created process with PID #4
Process ID: 4	

Cliente Automático

Objetivo de la prueba

Crear datos según los datos solicitados por el cliente.

Lo esperado

Se leen los datos sobre el rango de tiempo de espera entre los procesos y crea un random entre los dos parámetros ingresados. Para este caso será entre 4 y 6.

Se leen los datos sobre el rango burst de espera en procesos y crea un random entre los dos parámetros ingresados. En este caso es entre 1 y 4.

Se genera una prioridad en un random de 1-5 entre los procesos.

Los hilos envían la información en 2 segundos al servidor.

Resultado

Se leen los datos sobre el rango de tiempo de espera entre los procesos y crea un random entre los dos parámetros ingresados. En este caso se obtuvieron los datos 4 y 5.

Se leen los datos sobre el rango burst de espera en procesos y crea un random entre los dos parámetros ingresados. En este caso se obtuvieron los datos 4 y 3.

Se genera una prioridad en un random de 1-5 entre los procesos. En este caso se obtuvieron los datos 2, 4 y 3.

La información se envía 2 segundos después al servidor. Se demuestra por la resta del "Sent at" - "Created at" en cliente.

```
valeria@localhost:~/Escritorio/Cpu_Job_Schedule$ ./server
Select one algorithm:
1. FIFO
2. SJF
3. HPF
4. Round Robin
5. Exit
Enter your choice
1
Bind is connected
Waiting for clients...

Connection is being started

New process at: 14:50:34
Burst: 4
Priority: 2
Process ID: 1

New process at: 14:50:38
Burst: 4
Priority: 4
Process ID: 2

New process at: 14:50:43
Burst: 3
Priority: 3
Process ID: 3

valeria@localhost:~/Escritorio/Cpu_Job_Schedule$ ./client
1
Write waiting time and burst range for random at creating process:
4 6 1 4

Process id: 1
Created at: 14:50:32
Burst: 4
Priority: 2
Timeout to create next process: 4
Sent at: 14:50:34
Created process with PID #1

Process id: 2
Created at: 14:50:36
Burst: 4
Priority: 4
Timeout to create next process: 5
Sent at: 14:50:38
Created process with PID #2

Process id: 3
Created at: 14:50:41
Burst: 3
Priority: 3
Timeout to create next process: 4
Sent at: 14:50:43
Created process with PID #3

q
Process id: 4
Created at: 14:50:45
Burst: 2
```

Algoritmos

En el siguiente apartado se hacen pruebas de los algoritmos.

FIFO

Objetivo de la prueba

Para esta prueba, se quiere probar el algoritmo de “First In-First Out” con el cliente manual, utilizando el archivo de pruebas incluido con el código (“prueba.txt”).

Lo esperado

El comportamiento esperado para este algoritmo es la ejecución en orden de cada uno de los procesos. No importaría el tiempo entre envío de procesos para este caso. Deberá ejecutarse primero el 1, después del 2, seguido el 3 y por último el 4.

Resultados

El primero en llegar y ejecutarse es el proceso 1, seguido del 2, 3 y 4. Es posible ver los procesos que llegaron por medio de Ready Starts que presenta la cola ready después de cada proceso.

```

Archivo Editor Ver Buscar Terminal Ayuda
Processing Node: 1
Process ID:1
Burst:16
Priority:1
<-----,-----,-----,-----,-----,-----,>
Process #1 has ended with burst: 16

-----READY STARTS-----
-----
Process ID:2
Burst:5
Priority:3

Process ID:3
Burst:3
Priority:1
-----READY ENDS-----
-----

Processing Node: 2
Process ID:2
Burst:5
Priority:3
<-----,-----,-----,>
Process #2 has ended with burst: 5

-----READY STARTS-----
-----
Process ID:3
Burst:3
Priority:1

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----

```

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Processing Node: 3
Process ID:3
Burst:3
Priority:1
<-----,-----,>
Process #3 has ended with burst: 3

-----READY STARTS-----
-----

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----

Processing Node: 4
Process ID:4
Burst:11
Priority:2
<-----,-----,>
Process #4 has ended with burst: 11
q

Number of executed processes 4
Amount in seconds of CPU Idle: 49

-----Results table-----
Process ID:      ID      TAT      WT
Process ID:      1      16      0
Process ID:      2      13      8
Process ID:      3      10      7
Process ID:      4      17      6

-----End of table-----

TAT average of process: 14.000000
WT average of process: 5.250000

```

SJF

Objetivo de la prueba

Para esta prueba, se quiere probar el algoritmo de “Shortest Job First” con el cliente manual, utilizando el archivo de pruebas incluido con el código (“prueba.txt”).

Lo esperado

El comportamiento esperado para este algoritmo es la ejecución de cada proceso que llega, con prioridad a aquellos que tengan el menor burst dentro de la cola del ready. El tiempo entre envío de procesos afectará la ejecución de este proceso, en caso de que el burst de los procesos sea suficientemente corto para ser ejecutado antes de que llegue más de un proceso, en cuyo caso el algoritmo se comportará similar al FIFO.

Resultados

Tal y como se esperaba, los procesos con menor burst en la cola del ready son los seleccionados como el proceso a ejecutar.

```
Archivo Editor Ver Buscar Terminal Ayuda
Processing Node: 1
Process ID:1
Burst:16
Priority:1
<-----,-----,-----,-----,-----,-----,-----,-----,-----,>
Process #1 has ended with burst: 16

-----READY STARTS-----
-----

Process ID:2
Burst:5
Priority:3

Process ID:3
Burst:3
Priority:1
-----READY ENDS-----
-----

Processing Node: 3
Process ID:3
Burst:3
Priority:1
<-----,-----,>
Process #3 has ended with burst: 3

-----READY STARTS-----
-----

Process ID:2
Burst:5
Priority:3
-----READY ENDS-----
-----
```

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda

Processing Node: 2
Process ID:2
Burst:5
Priority:3
<-----,-----,-----,-----,>
Process #2 has ended with burst: 5

-----READY STARTS-----
-----

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----

Processing Node: 4
Process ID:4
Burst:11
Priority:2
<-----,-----,-----,-----,-----,-----,-----,-----,>
Process #4 has ended with burst: 11
q

Number of executed processes 4
Amount in seconds of CPU Idle: 357

-----Results table-----
Process ID:      ID      TAT      WT
Process ID:      1      16      0
Process ID:      3       5      2
Process ID:      2      16     11
Process ID:      4      13      2

-----End of table-----

TAT average of process: 12.500000
WT average of process: 3.750000

```


HPF

Objetivo de la prueba

Para esta prueba, se quiere probar el algoritmo de “High Priority First” con el cliente manual, utilizando el archivo de pruebas incluido con el código (“prueba.txt”).

Lo esperado

El comportamiento esperado para este algoritmo es la ejecución de cada proceso que llega, dando prioridad a aquellos procesos que tengan el número de prioridad más bajo. El tiempo entre envío de procesos afectará la ejecución de este proceso, de una manera similar al SJF.

Resultados

Tal y como se esperaba, los procesos con menor prioridad (el número más bajo) en la cola del ready son los seleccionados como el proceso a ejecutar.

```
Archivo Editar Ver Buscar Terminal Ayuda
Processing Node: 1
Process ID:1
Burst:16
Priority:1
<-----,-----,-----,-----,-----,-----,>
Process #1 has ended with burst: 16

-----READY STARTS-----
-----
Process ID:2
Burst:5
Priority:3

Process ID:3
Burst:3
Priority:1
-----READY ENDS-----
-----

Processing Node: 3
Process ID:3
Burst:3
Priority:1
<-----,-----,>
Process #3 has ended with burst: 3

-----READY STARTS-----
-----

Process ID:2
Burst:5
Priority:3

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----
```

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Processing Node: 4
Process ID:4
Burst:11
Priority:2
<-----,-----,-----,-----,-----,-----,-----,-----,>
Process #4 has ended with burst: 11

-----READY STARTS-----
-----

Process ID:2
Burst:5
Priority:3
-----READY ENDS-----
-----

Processing Node: 2
Process ID:2
Burst:5
Priority:3
<-----,-----,-----,-----,>
Process #2 has ended with burst: 5
q

Number of executed processes 4
Amount in seconds of CPU Idle: 24

-----Results table-----
Process ID:    ID      TAT      WT
Process ID:    1        16        0
Process ID:    3         6        3
Process ID:    4        12        1
Process ID:    2        29       24

-----End of table-----

TAT average of process: 15.750000
WT average of process: 7.000000

```

Round Robin (q = 5)

Objetivo de la prueba

Para esta prueba, se quiere probar el algoritmo de "Round Robin" utilizando un quantum de 3 con el cliente manual, utilizando el archivo de pruebas incluido con el código ("prueba.txt").

Lo esperado

El comportamiento esperado para este algoritmo es la ejecución de cada proceso que llega, restando al burst el quantum a cada proceso en orden de llegada, repitiendo estos pasos hasta que terminen todos los procesos. El tiempo entre envío de procesos tiene la posibilidad de afectar la ejecución del algoritmo, en caso de que el burst sea menor o igual al quantum y no haya siguientes. En casos donde se repita este comportamiento, se podría ver similar a FIFO.

Resultados

Dependiendo de lo indicado por el usuario, se posee un quantum y el proceso termina en el indicado. Se continúa con los encontrados en cola en forma de FIFO y cada uno ejecuta un máximo de 5 bursts.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Processing Node: 1
Process ID:1
Burst:16
Priority:1
<-----,-----,-----,-----,>
Process #1 has been executed with quantum of 5

-----READY STARTS-----
-----

Process ID:1
Burst:16
Priority:1
-----READY ENDS-----
-----

Processing Node: 1
Process ID:1
Burst:16
Priority:1
<-----,-----,-----,-----,>
Process #1 has been executed with quantum of 5

-----READY STARTS-----
-----

Process ID:1
Burst:16
Priority:1

Process ID:2
Burst:5
Priority:3
-----READY ENDS-----
-----
```

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Processing Node: 2
Process ID:2
Burst:5
Priority:3
<-----,-----,-----,>
Process #2 has ended with burst: 5

-----READY STARTS-----
-----

Process ID:1
Burst:16
Priority:1

Process ID:3
Burst:3
Priority:1
-----READY ENDS-----
-----

Processing Node: 3
Process ID:3
Burst:3
Priority:1
<-----,-----,>
Process #3 has ended with burst: 3

-----READY STARTS-----
-----

Process ID:1
Burst:16
Priority:1

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----
```

```
Archivo  Editor  Ver  Buscar  Terminal  Ayuda
Processing Node: 4
Process ID:4
Burst:11
Priority:2
<-----,-----,-----,-----,>
Process #4 has been executed with quantum of 5

-----READY STARTS-----
-----

Process ID:1
Burst:16
Priority:1

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----

Processing Node: 1
Process ID:1
Burst:16
Priority:1
<-----,-----,-----,-----,>
Process #1 has been executed with quantum of 5

-----READY STARTS-----
-----

Process ID:1
Burst:16
Priority:1

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----
```

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Processing Node: 4
Process ID:4
Burst:11
Priority:2
<-----,-----,-----,-----,>
Process #4 has been executed with quantum of 5

-----READY STARTS-----
-----

Process ID:1
Burst:16
Priority:1

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----

Processing Node: 1
Process ID:1
Burst:16
Priority:1
<-----,>
Process #1 has ended with burst: 16

-----READY STARTS-----
-----

Process ID:4
Burst:11
Priority:2
-----READY ENDS-----
-----

```

```

Processing Node: 4
Process ID:4
Burst:11
Priority:2
<-----,>
Process #4 has ended with burst: 11
q

Number of executed processes 4
Amount in seconds of CPU Idle: 23

-----Results table-----
Process ID:   ID      TAT      WT
Process ID:   2       10       5
Process ID:   3        5       2
Process ID:   1      35      19
Process ID:   4       19       8
-----End of table-----

TAT average of process: 17.250000
WT average of process: 8.500000

Server has done
[valeria@localhost Cpu_Job_Schedule]$ █

```

Cálculos

En esta sección se explican los cálculos del WT y TAT.

TAT

Objetivo de la prueba

Comprobar el cálculo del TAT de cada proceso así como el promedio en general.

Lo esperado

Se espera que la resta entre el tiempo de salida - el tiempo de llegada de cada proceso sea correcta. Además, verificar que sea un número positivo.

El promedio es la suma de cada TAT dividido entre el total de procesos.

Resultados

En este caso los resultados son correctos, es el tiempo de salida - el tiempo de llegada de cada proceso:

- $17-0=17$
- $22-6=16$
- $25-11=14$
- $36-18=18$

El promedio es correcto:

- $(17+16+14+18)/4=16.25$

```
Number of executed processes 4
Amount in seconds of CPU Idle: 16

-----Results table-----
      ID      TAT
Start time: 0
End time: 17
Process ID: 1      17
Start time: 6
End time: 22
Process ID: 2      16
Start time: 11
End time: 25
Process ID: 3      14
Start time: 18
End time: 36
Process ID: 4      18
-----End of table-----

TAT average of process: 16.250000
WT average of process: 7.500000
```

WT

Objetivo de la prueba

Comprobar el cálculo del TAT de cada proceso así como el promedio en general.

Lo esperado

Se espera que la resta entre el TAT - burst de cada proceso sea correcta, es decir, el WT. Además, verificar que sea un número positivo.

El promedio es la suma de cada WT dividido entre el total de procesos.

Resultados

En este caso los WT son correctos:

- $17-16=1$
- $16-5=11$
- $16-3=13$
- $23-11=12$

El promedio es correcto:

- $(1+11+13+12)/4=9.25$

```
Number of executed processes 4
Amount in seconds of CPU Idle: 16

-----Results table-----
      ID          TAT      WT
Start time: 0
End time: 17
Burst: 16
Process ID:   1          17          1
Start time: 6
End time: 22
Burst: 5
Process ID:   2          16          11
Start time: 9
End time: 25
Burst: 3
Process ID:   3          16          13
Start time: 13
End time: 36
Burst: 11
Process ID:   4          23          12

-----End of table-----

TAT average of process: 18.000000
WT average of process: 9.250000
```

Cantidad de procesos ejecutados

Objetivo de la prueba

Comprobar el cálculo de la cantidad de procesos.

Lo esperado

Que la cantidad de procesos sea el mismo número que el ID del último proceso en la tabla final.

Resultados

La suma de los procesos es el mismo número que el ID del último proceso en la tabla final.


```

Number of executed processes 4
Amount in seconds of CPU Idle: 16

-----Results table-----
      ID          TAT          WT
Start time: 0
End time: 17
Burst: 16
Process ID: 1          17          1
Start time: 6
End time: 22
Burst: 5
Process ID: 2          16          11
Start time: 9
End time: 25
Burst: 3
Process ID: 3          16          13
Start time: 13
End time: 36
Burst: 11
Process ID: 4          23          12

-----End of table-----

TAT average of process: 18.000000
WT average of process: 9.250000

```

Comandos

Quit

Objetivo de la prueba

Para esta prueba, se utilizará el comando de quit para terminar la simulación, para el cliente y el servidor.

Lo esperado

El comando se puede utilizar escribiendo el carácter “q” en la consola del programa y presionando enter. Se espera que al ejecutar el comando, se termine la ejecución, ya sea del servidor o del cliente.

Cuando el servidor termina la ejecución se muestra una tabla con el WT y TAT de cada proceso, tiempo del CPU ocioso, cantidad de procesos ejecutados, promedios del WT y TAT. Además debe mostrar un mensaje indicando que terminó.

Cuando el cliente termina la ejecución, se muestra un mensaje indicando que la ejecución terminó.

Resultados

El servidor muestra los datos anteriormente mencionados.

```

Number of executed processes 4
Amount in seconds of CPU Idle: 16

-----Results table-----
      ID          TAT          WT
Start time: 0
End time: 17
Burst: 16
Process ID: 1          17          1
Start time: 6
End time: 22
Burst: 5
Process ID: 2          16          11
Start time: 9
End time: 25
Burst: 3
Process ID: 3          16          13
Start time: 13
End time: 36
Burst: 11
Process ID: 4          23          12

-----End of table-----

TAT average of process: 18.000000
WT average of process: 9.250000

```

El cliente muestra un mensaje indicando que se terminó la ejecución.

```

Created process with PID #1
Created process with PID #2
Created process with PID #3
Created process with PID #4
Created process with PID #5
q
The client has stopped

```

Print cola de ready

Objetivo de la prueba

Para esta prueba, se utilizará el comando de print para visualizar la cola de ready en consola, durante la ejecución del servidor.

Lo esperado

El comando se puede utilizar escribiendo el carácter “p” en la consola del programa y presionando enter en el servidor. Se espera poder visualizar los procesos que faltan por ejecutar en el momento de ingresar el comando. Mostrando el proceso, burst y prioridad.

Resultados

Al presionar la tecla “p” en el servidor se muestra la cola del ready.

```
Priority:4  
-----,-----,-----,-----,-----  
  
-----READY STARTS-----  
-----  
  
Process ID:2  
Burst:9  
Priority:4  
  
Process ID:3  
Burst:8  
Priority:3  
  
-----READY ENDS-----  
-----
```

Comparación

Este apartado tiene como fin poder presentar puntos sobre la funcionalidad y el rendimiento de: Pthreads C y Threads Java y con ello poder generar un criterio de comparación a partir de su uso y la investigación sobre estos.

Pthread C

Este apartado trata de identificar de manera rápida y concisa la opinión sobre usabilidad, detalle técnico y rendimiento por parte de la librería Pthread en base a su uso y lo investigado de los apartados [\[2\]](#) [\[3\]](#) de la bibliografía.

Usabilidad:

- Curva de aprendizaje mediana-alta.
- Requiere un struct para enviar varios parámetros.
- Permite una mejor división de tareas.

Detalle técnico y Rendimiento:

- Permite una mejor administración de recursos entre los procesos (Memoria)
- Recomendable si se utiliza un sistema de colas
- Permite una control más extenso y especializado

Threads Java

Este apartado trata de identificar de manera rápida y concisa la opinión sobre funcionalidad y rendimiento por parte de la librería Pthread en base a su uso y lo investigado de los apartados [1] [4] de la bibliografía.

Usabilidad:

- Curva de aprendizaje baja-mediana.
- Su implementación es rápida y autogestionable.

Detalle técnico y Rendimiento:

- Se recomienda hacer una correcta utilización de los mismos, debido a que si no se realiza esto pueden llegar a ser más perjudiciales que no utilizarlo.
- Presenta funcionalidades necesarias para realizar control sobre los ejecutado.

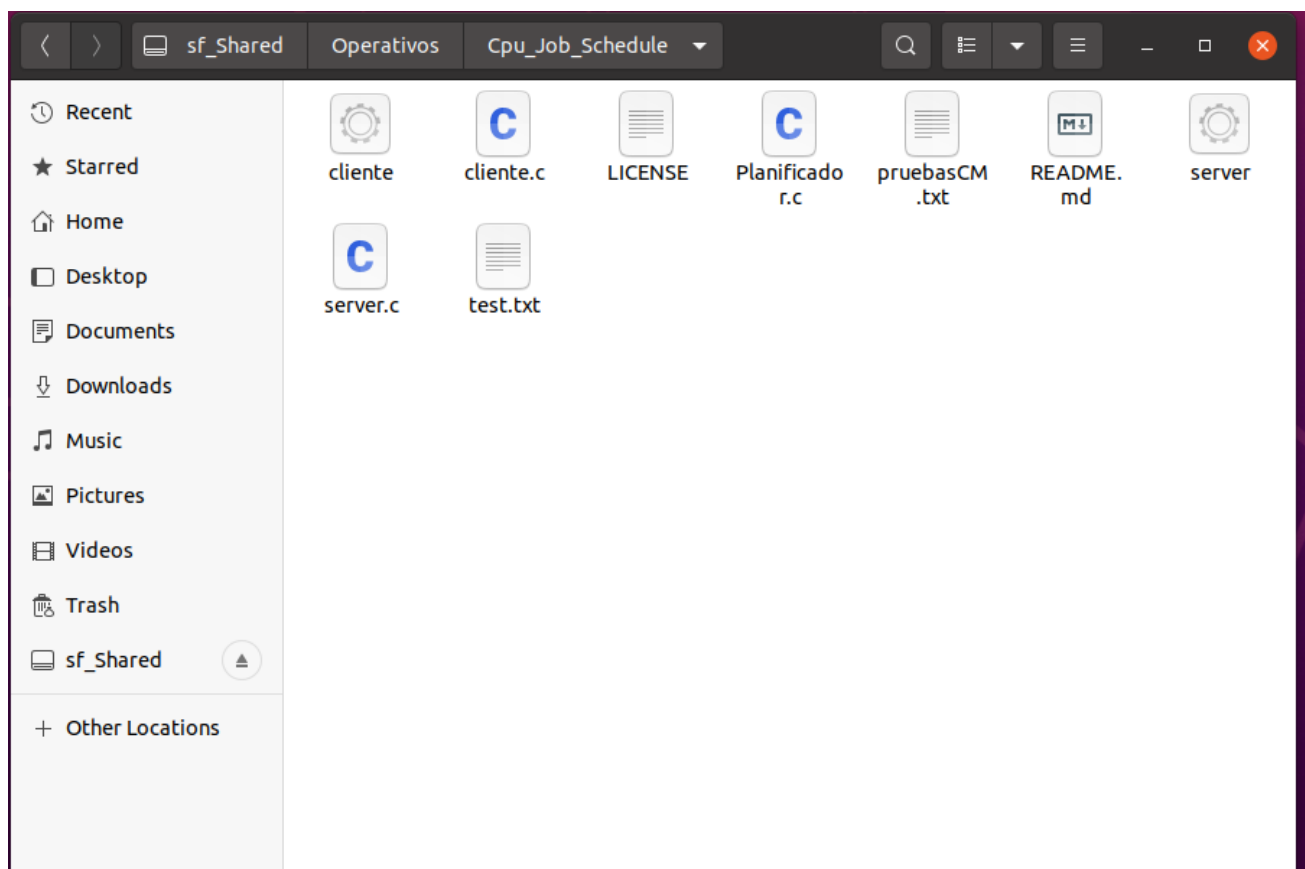
Basado en lo mostrado anteriormente podemos concluir lo siguiente:

La librería PThread en C requiere un aprendizaje más extenso, pero esta brinda un enfoque más especializado sobre la gestión.

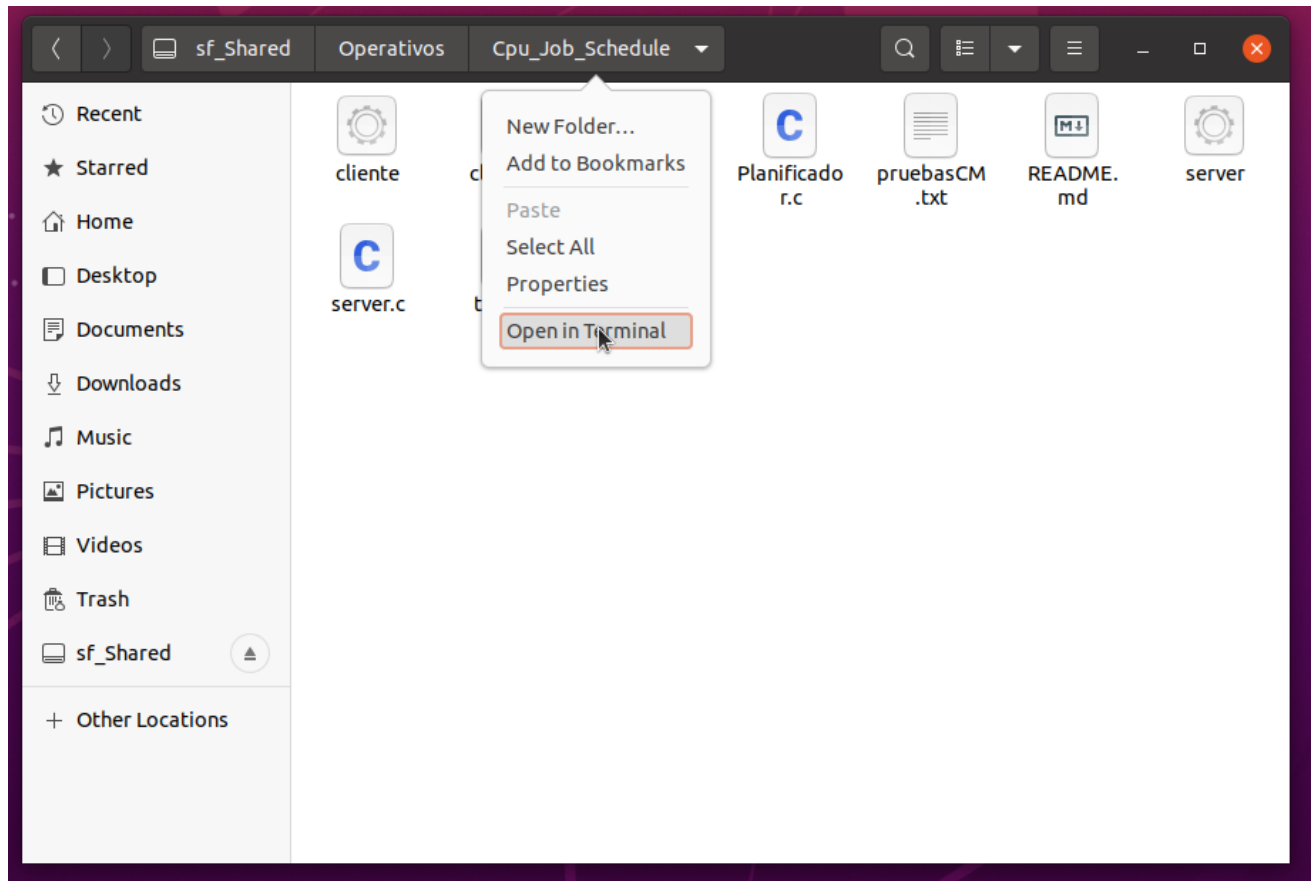
La librería Threads en Java requiere un aprendizaje más rápido, pero esta brinda un enfoque más general y autogestionable.

Manual de Usuario

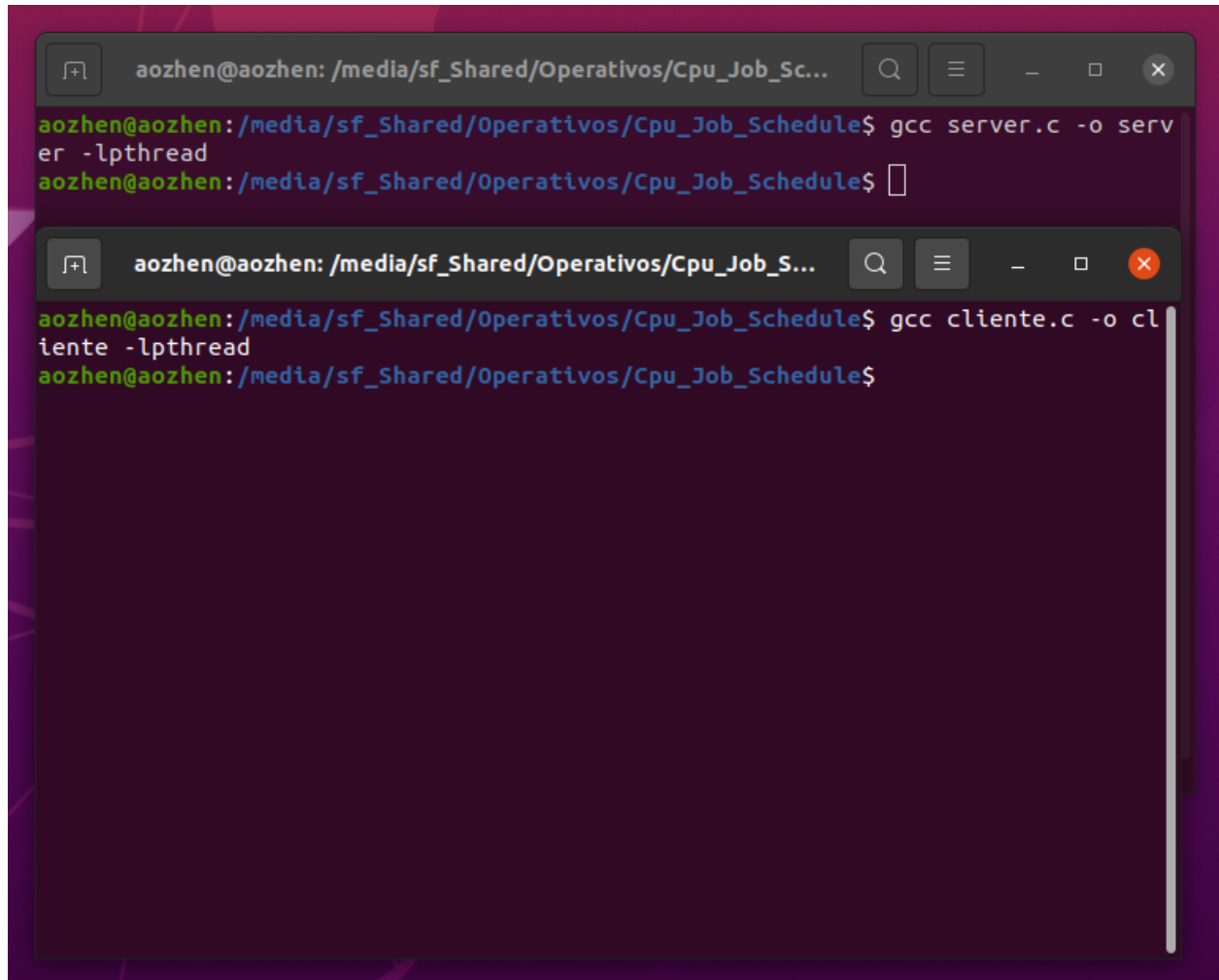
Para compilar y ejecutar el proyecto, se debe primero descargar y extraer los archivos del repositorio en una máquina con sistema operativo basado en Linux. Una vez extraídos, al ingresar a la carpeta con los archivos debería ver los siguientes:



Una vez se tenga la carpeta, abra dos terminales dentro de la carpeta. Una terminal se utilizará para el servidor y la otra para el cliente:



Cuando se tenga las terminales abiertas, se pueden ingresar los siguientes comandos para compilar el servidor y el cliente:



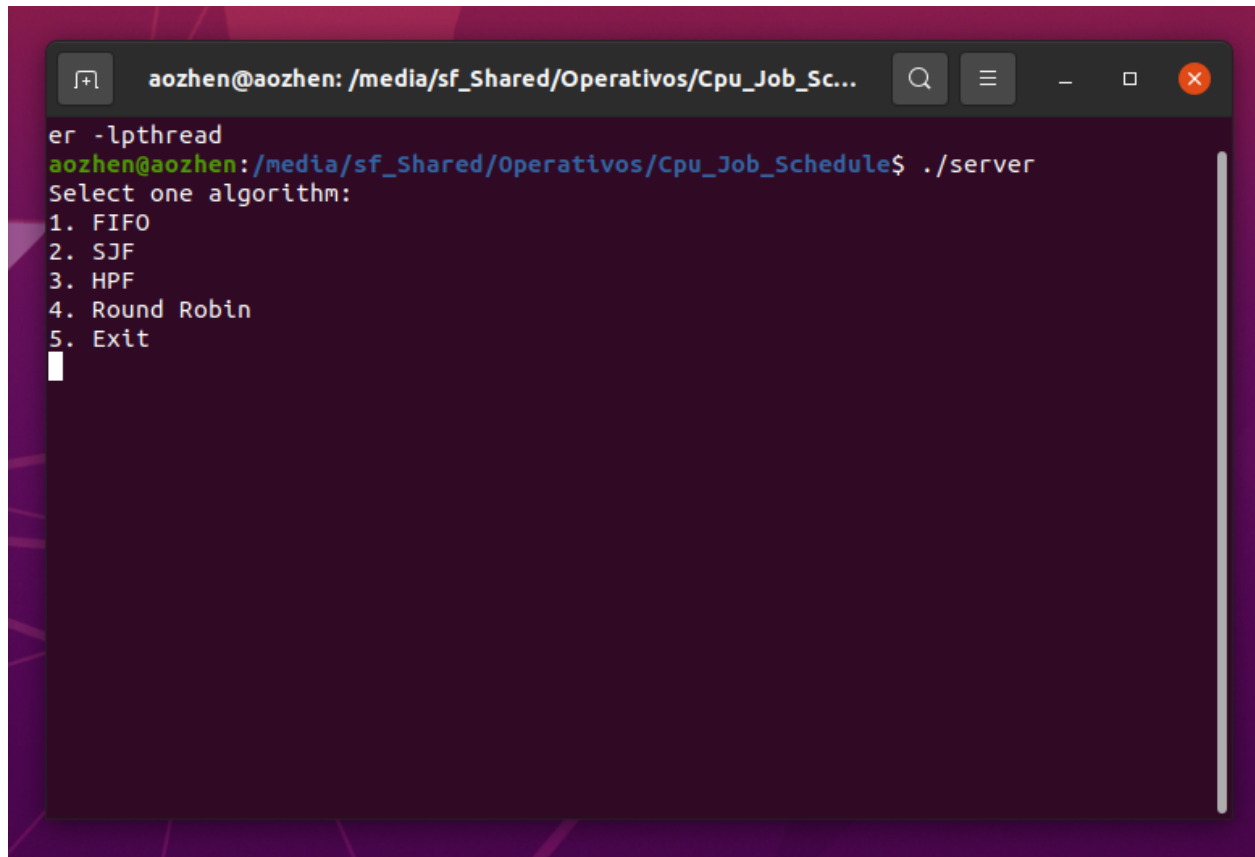
```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
aozhen@aozhen:/media/sf_Shared/Operativos/Cpu_Job_Schedule$ gcc server.c -o server -lpthread
aozhen@aozhen:/media/sf_Shared/Operativos/Cpu_Job_Schedule$

aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_S...
aozhen@aozhen:/media/sf_Shared/Operativos/Cpu_Job_Schedule$ gcc cliente.c -o cliente -lpthread
aozhen@aozhen:/media/sf_Shared/Operativos/Cpu_Job_Schedule$
```

Cliente: "gcc cliente.c -o cliente -lpthread"

Servidor: "gcc server.c -o server -lpthread"

Cuando se tengan compilados, se pueden ejecutar cada uno con `./server` para ejecutar el servidor y `./cliente` para ejecutar el cliente. Recuerde que se debe ejecutar el servidor primero:

A terminal window with a dark background and light-colored text. The window title is 'aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...'. The prompt is 'aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule\$'. The user has entered './server'. The output shows a menu for selecting an algorithm: 'Select one algorithm:', '1. FIFO', '2. SJF', '3. HPF', '4. Round Robin', '5. Exit'. A cursor is visible at the end of the '5. Exit' line.

```
er -lpthread
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./server
Select one algorithm:
1. FIFO
2. SJF
3. HPF
4. Round Robin
5. Exit
█
```

Al ejecutar el servidor, se mostrará un menú con 4 opciones para el algoritmo y una opción para salir de la simulación. Las opciones se pueden seleccionar escribiendo en la consola el número respectivo. Para efectos de este manual, se utiliza la opción cuatro para utilizar el algoritmo Round Robin, que tiene un aspecto extra:

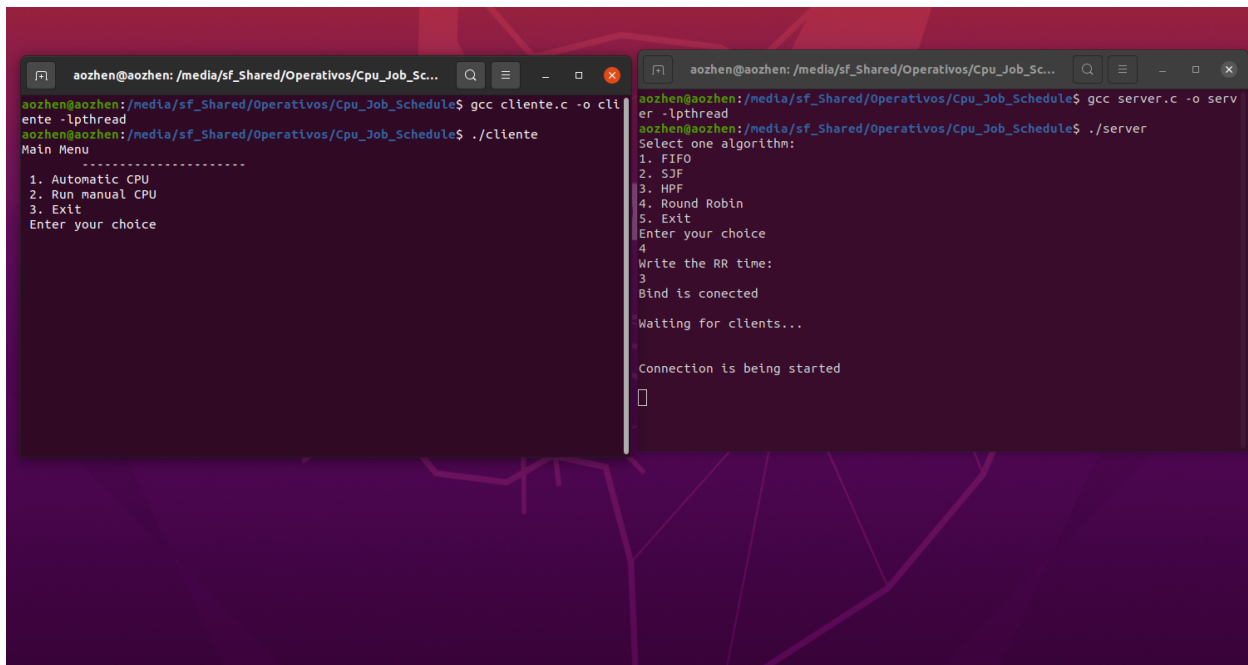
```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
aozhen@aozhen:/media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./server
Select one algorithm:
1. FIFO
2. SJF
3. HPF
4. Round Robin
5. Exit
4
Write the RR time:
█
```

Aquí pide el *quantum* para el algoritmo de Round Robin. Para efectos de este manual, se utilizará un *quantum* de 3:

```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
aozhen@aozhen:/media/sf_Shared/Operativos/Cpu_Job_Schedule$ gcc server.c -o server -lpthread
aozhen@aozhen:/media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./server
Select one algorithm:
1. FIFO
2. SJF
3. HPF
4. Round Robin
5. Exit
Enter your choice
4
Write the RR time:
3
Bind is connected

Waiting for clients...
```


Una vez realizado el *bind*, se puede ejecutar el cliente:

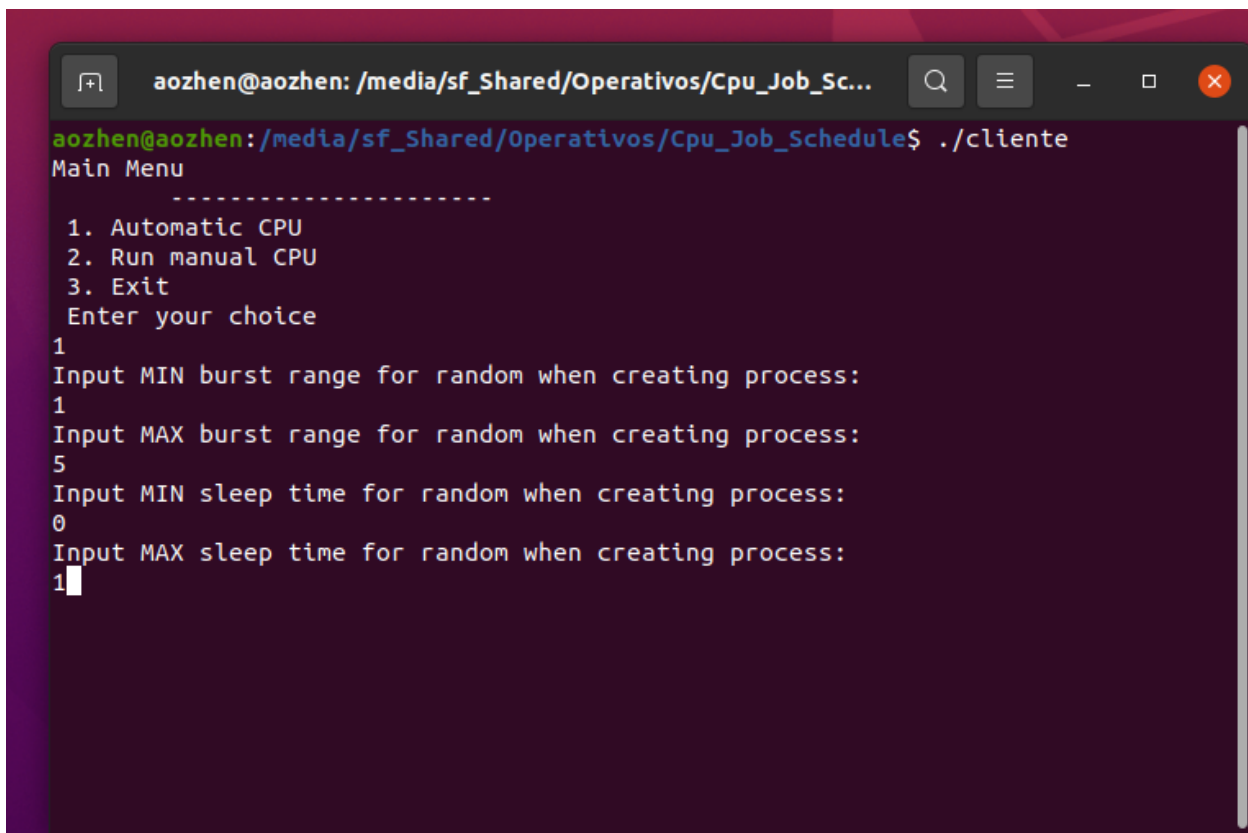


The image shows two terminal windows side-by-side. The left window shows the compilation of 'cliente.c' and its execution, displaying a menu with three options: 1. Automatic CPU, 2. Run manual CPU, and 3. Exit. The right window shows the compilation of 'server.c' and its execution, displaying a list of scheduling algorithms: 1. FIFO, 2. SJF, 3. HRF, 4. Round Robin, and 5. Exit. It also shows the server waiting for clients and a connection being established.

```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ gcc cliente.c -o cliente -lpthread
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./cliente
Main Menu
-----
1. Automatic CPU
2. Run manual CPU
3. Exit
Enter your choice

aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ gcc server.c -o server -lpthread
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./server
Select one algorithm:
1. FIFO
2. SJF
3. HRF
4. Round Robin
5. Exit
Enter your choice
4
Write the RR time:
3
Bind is connected
Waiting for clients...
Connection is being started
█
```

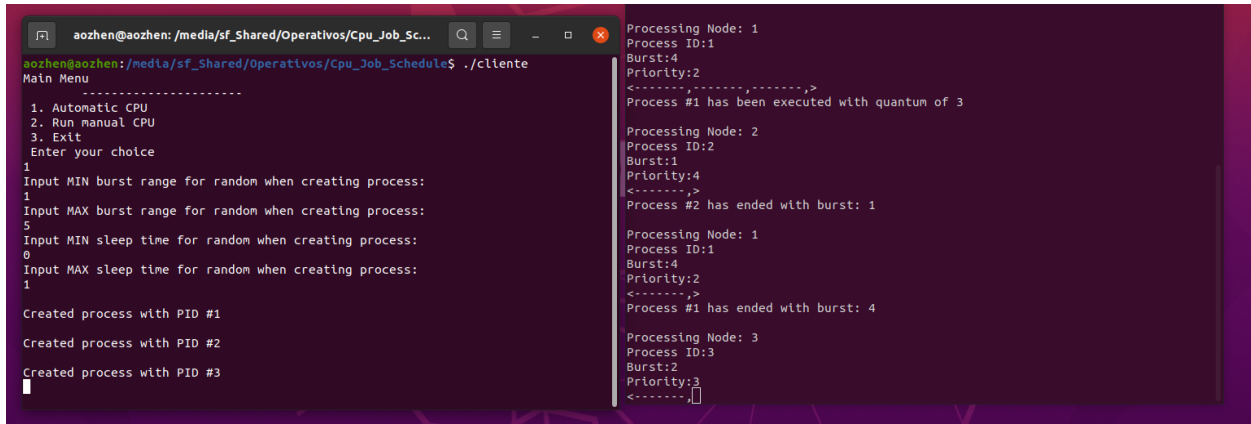
El cliente tiene un menú con dos opciones de ejecución y una opción para salir. La selección de opción es ingresando el número respectivo en la consola. Primero se utilizara el cliente automatico:



The image shows a terminal window with the client program running. It displays the same menu as before, but with option 1 selected. It then prompts for input for MIN burst range, MAX burst range, MIN sleep time, and MAX sleep time, with values 1, 5, 0, and 1 respectively.

```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./cliente
Main Menu
-----
1. Automatic CPU
2. Run manual CPU
3. Exit
Enter your choice
1
Input MIN burst range for random when creating process:
1
Input MAX burst range for random when creating process:
5
Input MIN sleep time for random when creating process:
0
Input MAX sleep time for random when creating process:
1
█
```

Al ingresar al cliente automático, se pide un dos rangos de números que serán para la generación de random del *burst* y del *sleep* respectivamente, que se hará entre cada generación de proceso. Para efectos del manual se utilizará 1-5 para el *burst* y 0-1 para el *sleep*:



```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./cliente
Main Menu
-----
1. Automatic CPU
2. Run manual CPU
3. Exit
Enter your choice
1
Input MIN burst range for random when creating process:
1
Input MAX burst range for random when creating process:
5
Input MIN sleep time for random when creating process:
0
Input MAX sleep time for random when creating process:
1
Created process with PID #1
Created process with PID #2
Created process with PID #3

```

```
Processing Node: 1
Process ID:1
Burst:4
Priority:2
<-----,>
Process #1 has been executed with quantum of 3

Processing Node: 2
Process ID:2
Burst:1
Priority:4
<-----,>
Process #2 has ended with burst: 1

Processing Node: 1
Process ID:1
Burst:4
Priority:2
<-----,>
Process #1 has ended with burst: 4

Processing Node: 3
Process ID:3
Burst:2
Priority:3
<-----,>

```

Una vez que se ingrese el tiempo, se empezará a generar procesos de manera aleatoria y se enviarán al servidor para ser procesados según el algoritmo escogido. El servidor puede desplegar la cola de Ready actual ingresado el carácter “p” en la consola en cualquier momento de la ejecución:

```
<-----,-----,-----,p
-----READY-----
-----

Process ID:40
Burst:4
Priority:1

Process ID:41
Burst:3
Priority:4

Process ID:42
Burst:2
Priority:4

Process ID:43
Burst:4
Priority:5

Process ID:44
Burst:4
Priority:2

Process ID:45
Burst:5
Priority:5

Process ID:46
Burst:3
Priority:1

Process ID:47
Burst:2
Priority:4

Process ID:48
Burst:5
Priority:3
-----READY END-----
-----
```

En el cliente y en el servidor, se puede ingresar el carácter “q” para terminar la ejecución de cada uno. Al finalizar la ejecución del servidor, el cliente automáticamente finaliza también:

```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
Created process with PID #6
Created process with PID #7
Created process with PID #8
Created process with PID #9
Created process with PID #10
Created process with PID #11
Created process with PID #12
Created process with PID #13
Created process with PID #14
Created process with PID #15
q
The client has stopped

Client has done
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$
```

El servidor sigue esperando conexiones nuevas de clientes mientras que no sea finalizado con el comando. Ahora ejecutaremos un cliente nuevo en manual para observar cómo funciona:

```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
Se crea el proceso con el PID #38
Se crea el proceso con el PID #39
Se crea el proceso con el PID #40
Se crea el proceso con el PID #41
Se crea el proceso con el PID #42
Se crea el proceso con el PID #43
Se crea el proceso con el PID #44
Se crea el proceso con el PID #45
Se crea el proceso con el PID #46
Se crea el proceso con el PID #47
Se crea el proceso con el PID #48
q
The client has stopped
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./cliente
Main Menu
-----
1. Automatic CPU
2. Run manual CPU
3. Exit
Enter your choice
2
Write file name:
```

```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
Process ID:46
Burst:3
Priority:1
<-----,>
Process #46 has ended with burst: 3

Processing Node: 47
Process ID:47
Burst:2
Priority:4
<-----,>
Process #47 has ended with burst: 2

Processing Node: 48
Process ID:48
Burst:5
Priority:3
<-----,>
Process #48 has ended with burst: 5

Connection is being started
```

Cuando se ingresa al cliente manual, se le pide el nombre del archivo a procesar. Este archivo se recomienda tenerlo en la carpeta del programa para facilidad, y tiene el siguiente formato:

1	1	5
2	2	3
3	9	3
4	10	2
5	5	11

El primer número simboliza el *burst* del proceso, mientras que el segundo es el *priority*. Al ingresar el nombre, se realizará la simulación con cada línea del archivo, finalizando cuando se acabe el archivo:

```

aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
Created process with PID #13
Created process with PID #14
Created process with PID #15
q
The client has stopped

Client has done
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Schedule$ ./cliente
Main Menu
-----
1. Automatic CPU
2. Run manual CPU
3. Exit
Enter your choice
2
Write file name:
test.txt

Created process with PID #16
Created process with PID #17

aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
Processing Node: 13
Process ID:13
Burst:4
Priority:2
<-----,>
Process #13 has ended with burst: 4

Connection is being started

Processing Node: 16
Process ID:16
Burst:1
Priority:5
<-----,>
Process #16 has ended with burst: 1

Processing Node: 17
Process ID:17
Burst:2
Priority:3
<-----,>
Process #17 has ended with burst: 2

```

Una vez quiera finalizar la simulación y obtener los datos finales, puede realizar el comando para terminar la simulación en el servidor. Para facilidad de lectura, se utilizará solo el procesamiento del archivo “test.txt” mostrado anteriormente con el algoritmo Round Robin. En la siguiente imagen se puede observar los cálculos finales:

```
aozhen@aozhen: /media/sf_Shared/Operativos/Cpu_Job_Sc...
Number of executed processes 18
Amount in seconds of CPU Idle: 575

-----Results table-----
Process ID:      ID      TAT      WT
Process ID:      2       3       2
Process ID:      3       3       1
Process ID:      4       3       1
Process ID:      5       2       1
Process ID:      1       11      7
Process ID:      6       3       1
Process ID:      7       4       1
Process ID:      9       6       3
Process ID:     14      12       9
Process ID:     15      12      10
Process ID:      8      27      23
Process ID:     10      24      20
Process ID:     11      24      19
Process ID:     12      23      19
Process ID:     13      22      18
Process ID:     16       1       0
Process ID:     17       2       0
Process ID:     18      13       4

-----End of table-----

TAT average of process: 10.833333
WT average of process: 7.722222
```

Esto finaliza el manual de usuario para el proyecto de Planificador de CPU.

Bitácora

Debido a su extensión, se decidió realizar un archivo por aparte. Esta se encuentra en este mismo repositorio con el nombre de "Bitácora.pdf".

Bibliografía

- [1] Rendimiento de Java Thread Worker Group frente al rendimiento de bucle. It-Helper. (2022). Retrieved 11 April 2022, from <https://codenostra.com/es/rendimiento-de-java-thread-worker-group-frente-al-rendimiento-de-bucle-integrado.html>.
- [2] Dell, M., Paniego, J., Pi Puig, M., Naiouf, M., & De Guisti, A. (2015). *Análisis del impacto de distintas técnicas de optimización de rendimiento en multicore* [Ebook] (pp. 3-9). CONICET. Retrieved 11 April 2022, from <http://sedici.unlp.edu.ar/handle/10915/50188>.

[3] Zhangyy, M. (2016). *Uso de contraste de pthread y std :: thread - programador clic*. Programmerclick.com. Retrieved 11 April 2022, from <https://programmerclick.com/article/777858572/>.

[4] SQC, A. (2020). *Uso de hilos en Java*. Platzi. Retrieved 11 April 2022, from <https://platzi.com/tutoriales/1236-java-avanzado/4842-uso-de-hilos-en-javaparte-3/>.