Instituto Tecnológico de Costa Rica

Escuela de Computación

Ingeniería en computación

Sistemas Operativos
Grupo 1

Profesor: Ericka Marín

Planificador del CPU

Adrián López Quesada, 2014081634
Josué Arrieta Salas, 2014008153

Cartago

Miércoles 21 de setiembre

Índice

Contents

Análisis de resultados

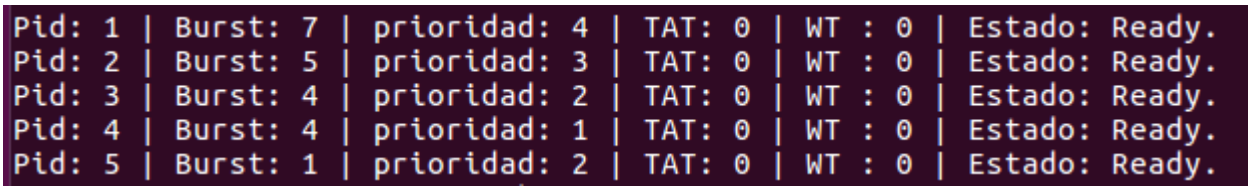
Objetivo	Porcentaje de éxito (100%)
Cliente Manual: se recibe un parámetro de archivo con una lista de procesos y se crean procesos a partir de ella.	100%, apenas termina de leer todo el archivo, el cliente se detiene.
Cliente Automático: la creación de procesos se hace mediante un proceso random (y se puede detener la ejecución del mismo)	100%, la detención se hace por medio de CTRL + C o la tecla E + ENTER. Se recomienda CTRL+C.
Por cada proceso creado se crea un thread que se comunica por medio de un socket con el servidor. Estos se crean en un intervalo de un tiempo random. Una vez enviada la información deberán morir.	100%, el random va desde 0 a 10 segundos, es posible modificar el valor pero para no saturar la cola del Ready se utilizó de 10 máximo.
Simulador: se debe seleccionar el algoritmo a correr (FIFO, SFJ, HPF y RR). En caso de ser RR también se puede especificar el quantum.	100%, el socket no se inicializa hasta que el usuario no seleccione un algoritmo.
Simulador: se crea un hilo para el job scheduler. Este recibe los mensajes del socket que vienen del cliente. Los mete a la lista de ready.	100%, el job scheduler es solo un socket abierto en conexión local.
Simulador: cpu scheduler, es otro hilo, se encarga de verificar si constantemente hay procesos en la cola. Si hay los ejecuta de acuerdo al algoritmo escogido.	100%, en caso de que no hayan procesos, se duerme el CPU scheduler por un segundo, aumento el tiempo ocioso
Simulador: hay un thread que está alerta siempre a las teclas que el usuario oprima. Esto es por si necesita ver la cola o para detener la simulación.	100%, se optó por poner esta funcionalidad en el thread principal.
Cada vez que hay un context switch se despliega en pantalla el proceso que se ejecutará.	100%, se muestra el PID, el burst y la prioridad.
Cada vez que un proceso termina por completo su ejecución, también se debe desplegar en pantalla.	100%, se muestra el PID.
En cualquier momento de la ejecución se puede consultar la cola de procesos en espera.	100%, además es posible ver todos los procesos terminados al presionar 3.
Se debe poder detener la simulación en cualquier	100%., al presionar 2 se detiene la

momento y se puede mostrar: - Cantidad de procesos ejecutados. - Cantidad de segundos de CPU ocioso. - Tabla de TAT y WT de procesos ejecutados. - Promedio de WT. - Promedio de TAT.	simulación y el cliente para también. .

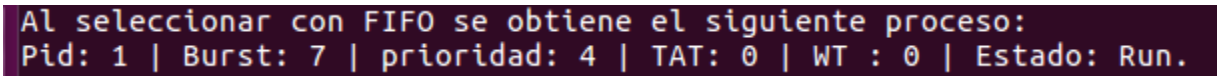
Casos de pruebas

Prueba 1 : FIFO

Objetivo: se prueba de manera aislada el algoritmo de selección FIFO.
Resultados esperados: se espera que de un grupo de procesos en Ready, seleccione el proceso con el menor pid.



Entrada: Se tienen los siguientes procesos en el Ready.
Resultados Obtenidos (correctos): se selecciona el proceso con el pid más bajo (1):



Prueba 2: SJF

Objetivo: se prueba de manera aislada el algoritmo de selección SJF.
Resultados esperados: se espera que de un grupo de procesos en Ready, seleccione el proceso con el menor burst.

Entrada: Se tienen los siguientes procesos en el Ready.

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
```

Resultados Obtenidos (incorrectos): se selecciona el proceso 3:

```
Al seleccionar con SJF se obtiene el siguiente proceso:
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Run.
```

Se encuentra un bug: el último proceso en la cola del Ready nunca era analizado. Se arregla el bug y se obtiene resultados (correctos) y esta vez se selecciona el proceso con el pid 5:

```
Al seleccionar con SJF se obtiene el siguiente proceso:
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Run.
```

Prueba 3 : HPF

Objetivo: se prueba de manera aislada el algoritmo de selección HPF.
Resultados esperados: se espera que de un grupo de procesos en Ready, seleccione el proceso con la menor prioridad.

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
```

Entrada: Se tienen los siguientes procesos en el Ready:

Resultados Obtenidos (correctos): se selecciona el proceso con la prioridad más alta (proceso con pid de 4):

```
Al seleccionar con HPF se obtiene el siguiente proceso:
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Run.
```

Prueba 4: Desplegar la cola de procesos

Objetivo: se prueba de manera aislada el despliegue de la cola de procesos.
Resultados esperados: se espera que de un grupo de procesos en Ready, se imprima la información de los mismos.

Entrada: Se tienen los siguientes procesos en el Ready.

```
//pPId, pBurst, pPrioridad, pEstado, pTiempoRestante
push(1, 7, 4, 1, 7);
push(2, 5, 3, 1, 5);
push(3, 4, 2, 1, 4);
push(4, 4, 1, 1, 4);
push(5, 1, 2, 1, 1);
```

Resultados Obtenidos (correctos): se muestra correctamente los procesos en Ready:

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Tiempo Restante: 7 | Estado: Ready.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Tiempo Restante: 5 | Estado: Ready.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Tiempo Restante: 4 | Estado: Ready.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Tiempo Restante: 4 | Estado: Ready.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Tiempo Restante: 1 | Estado: Ready.
```

Prueba 5: Round Robin

Objetivo: se prueba de manera aislada el algoritmo de selección RR. Se presentan varios casos, dónde cada caso se diferencia del proceso actual en ejecución.
Resultados esperados: Se espera que se seleccione el siguiente proceso en Ready de la lista de procesos, según las características del Round Robin.

Prueba 5.1: Caso “Normal”

Resultados Obtenidos y Entradas (correctos): se selecciona el proceso que siga después del proceso con pid de 2:

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Al seleccionar con RR se obtiene el siguiente proceso (id de proceso actual = 2):
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Run.
```

Prueba 5.2 : Caso “al final de la lista”

Resultados Obtenidos y Entradas (correctos): se selecciona el proceso que siga después del proceso con pid de 5. Se puede ver como el proceso es el último en la lista de procesos, de tal forma que se debe comenzar otra vez por el inicio de la lista:

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Al seleccionar con RR se obtiene el siguiente proceso (id de proceso actual = 5):
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Run.
```

Prueba 5.3: Caso en que es el único proceso en Ready.

Resultados Obtenidos y Entradas (correctos): se selecciona el proceso que siga después del proceso con pid de 5. Sin embargo en este caso no hay ningún proceso en Ready, y el algoritmo se debe de dar cuenta del caso:

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Al seleccionar con RR se obtiene el siguiente proceso (id de proceso actual = 5):
No se encontro procesos
```

Aunque sale que “No se encontró procesos”, se refiere a que no hubo procesos nuevos, de tal forma que el mismo proceso que estaba en ejecución, vuelve otra vez a ejecutar el quantun.

Prueba 5.4 : Caso en el que el siguiente proceso, en realidad es el anterior

Resultados Obtenidos y Entradas (incorrecto): se selecciona el proceso que siga después del proceso con pid de 5. En este caso debería retornar el proceso con pid de 4, sin embargo no se encuentra procesos:

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Al seleccionar con RR se obtiene el siguiente proceso (id de proceso actual = 5):
No se encontro procesos
```

Se arregla (correcto). Falto una condición de if que revisaba para el “último proceso”, es decir, para el proceso anterior al proceso que actualmente está en el round robin:

```
Pid: 1 | Burst: 7 | prioridad: 4 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 2 | Burst: 5 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 3 | Burst: 4 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Finalizado.
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 5 | Burst: 1 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
Al seleccionar con RR se obtiene el siguiente proceso (id de proceso actual = 5):
Pid: 4 | Burst: 4 | prioridad: 1 | TAT: 0 | WT : 0 | Estado: Run.
```

Prueba 4: Corrida General SFJ

Objetivo: se prueba de manera general todo el funcionamiento de una corrida SFJ. Esta prueba aunque es para SFJ también aplica para FIFO y HPF por igual ya que el algoritmo de ejecución es básicamente el mismo. Se ha de mencionar que estas pruebas incluyen: sockets y threads.
Resultados esperados: se espera que de un grupo de procesos en Ready, seleccione el proceso con el menor burst.

Resultados Obtenidos y entradas (incorrectos): se puede notar como en el resultado la transición de una operación de proceso a la siguiente. Es cierto que se escoge el proceso con el burst más bajo (proceso con pid 15), sin embargo en la siguiente iteración, muestra que este están en Run; pero también todos los procesos anteriores lo están (procesos con pid 12-14). Esto es imposible e incorrecto:

```
1
Pid: 1 | Burst: 17 | prioridad: 2 | TAT: 17 | WT : 0 | Estado: Finalizado.
Pid: 2 | Burst: 2 | prioridad: 4 | TAT: 19 | WT : 17 | Estado: Finalizado.
Pid: 3 | Burst: 2 | prioridad: 5 | TAT: 21 | WT : 19 | Estado: Finalizado.
Pid: 4 | Burst: 10 | prioridad: 2 | TAT: 10 | WT : 0 | Estado: Finalizado.
Pid: 5 | Burst: 1 | prioridad: 4 | TAT: 11 | WT : 10 | Estado: Finalizado.
Pid: 6 | Burst: 6 | prioridad: 1 | TAT: 7 | WT : 1 | Estado: Finalizado.
Pid: 7 | Burst: 7 | prioridad: 3 | TAT: 13 | WT : 6 | Estado: Finalizado.
Pid: 8 | Burst: 12 | prioridad: 5 | TAT: 19 | WT : 7 | Estado: Finalizado.
Pid: 9 | Burst: 19 | prioridad: 3 | TAT: 31 | WT : 12 | Estado: Finalizado.
Pid: 10 | Burst: 9 | prioridad: 2 | TAT: 28 | WT : 19 | Estado: Finalizado.
Pid: 11 | Burst: 10 | prioridad: 5 | TAT: 28 | WT : 28 | Estado: Run.
Pid: 12 | Burst: 17 | prioridad: 2 | TAT: 28 | WT : 28 | Estado: Ready.
Pid: 13 | Burst: 15 | prioridad: 5 | TAT: 9 | WT : 9 | Estado: Ready.
Pid: 14 | Burst: 12 | prioridad: 5 | TAT: 9 | WT : 9 | Estado: Ready.
Pid: 15 | Burst: 8 | prioridad: 2 | TAT: 0 | WT : 0 | Estado: Ready.
El proceso: 11 acaba de salir de ejecucion
Proceso: 15 con burst 8 y prioridad 2 entra en ejecucion
1
Pid: 1 | Burst: 17 | prioridad: 2 | TAT: 17 | WT : 0 | Estado: Finalizado.
Pid: 2 | Burst: 2 | prioridad: 4 | TAT: 19 | WT : 17 | Estado: Finalizado.
Pid: 3 | Burst: 2 | prioridad: 5 | TAT: 21 | WT : 19 | Estado: Finalizado.
Pid: 4 | Burst: 10 | prioridad: 2 | TAT: 10 | WT : 0 | Estado: Finalizado.
Pid: 5 | Burst: 1 | prioridad: 4 | TAT: 11 | WT : 10 | Estado: Finalizado.
Pid: 6 | Burst: 6 | prioridad: 1 | TAT: 7 | WT : 1 | Estado: Finalizado.
Pid: 7 | Burst: 7 | prioridad: 3 | TAT: 13 | WT : 6 | Estado: Finalizado.
Pid: 8 | Burst: 12 | prioridad: 5 | TAT: 19 | WT : 7 | Estado: Finalizado.
Pid: 9 | Burst: 19 | prioridad: 3 | TAT: 31 | WT : 12 | Estado: Finalizado.
Pid: 10 | Burst: 9 | prioridad: 2 | TAT: 28 | WT : 19 | Estado: Finalizado.
Pid: 11 | Burst: 10 | prioridad: 5 | TAT: 38 | WT : 28 | Estado: Finalizado.
Pid: 12 | Burst: 17 | prioridad: 2 | TAT: 38 | WT : 38 | Estado: Run.
Pid: 13 | Burst: 15 | prioridad: 5 | TAT: 19 | WT : 19 | Estado: Run.
Pid: 14 | Burst: 12 | prioridad: 5 | TAT: 19 | WT : 19 | Estado: Run.
Pid: 15 | Burst: 8 | prioridad: 2 | TAT: 10 | WT : 10 | Estado: Run.
Pid: 16 | Burst: 13 | prioridad: 5 | TAT: 10 | WT : 10 | Estado: Ready.
Pid: 17 | Burst: 16 | prioridad: 1 | TAT: 10 | WT : 10 | Estado: Ready.
Pid: 18 | Burst: 7 | prioridad: 5 | TAT: 0 | WT : 0 | Estado: Ready.
Pid: 19 | Burst: 14 | prioridad: 3 | TAT: 0 | WT : 0 | Estado: Ready.
El proceso: 15 acaba de salir de ejecucion
Proceso: 18 con burst 7 y prioridad 5 entra en ejecucion
1
```

Debido a que la prueba anterior fue corrida en cliente automático, es imposible volver a recrear el mismo caso de prueba, sin embargo se arregló el bug: lo que pasaba es que al recorrer la lista buscando el proceso con el burst más bajo, cada vez que se encontraba un burst más bajo que el anterior, este proceso se ponía en Run, lo cual es incorrecto. Se arregla el bug, y ahora solamente se pone el último proceso con el menor burst encontrado luego de recorrer toda la lista.

Prueba 5: Corrida General algoritmo RR

Objetivo: se prueba una corrida general el algoritmo de RR.
Resultados esperados: se espera que de un grupo de procesos en Ready, se selecciona el siguiente proceso en Ready, luego del proceso actual en Run.

Resultados Obtenidos y entradas (incorrectos): al iniciar el proceso, entra el proceso con burst 1. Sin embargo, ¡este nunca entró a ejecución! De hecho hubo un momento de CPU ocioso en dónde el proceso 1 estaba en lista y no se ejecutó. Fue hasta que entro el proceso 2 que el CPU dejó de ser ocioso.

```
Especifique el Quantum:
5
Seleccione 1 para ver la cola y 2 para detener la ejecucion en cualquier momento1
1
Pid: 1 | Burst: 2 | prioridad: 1 | TAT: 0 | WT : 0 | Tiempo Restante: 2 | Estado: Ready.
Proceso: 2 con burst 7, tiempo restante: 7 y prioridad 3 entra en ejecucion
Proceso: 3 con burst 12, tiempo restante: 12 y prioridad 3 entra en ejecucion
Proceso: 4 con burst 4, tiempo restante: 4 y prioridad 5 entra en ejecucion
El proceso: 4 acaba de salir de ejecucion
Proceso: 1 con burst 2, tiempo restante: 2 y prioridad 1 entra en ejecucion
El proceso: 1 acaba de salir de ejecucion
Proceso: 2 con burst 7, tiempo restante: 2 y prioridad 3 entra en ejecucion
El proceso: 2 acaba de salir de ejecucion
Proceso: 3 con burst 12, tiempo restante: 7 y prioridad 3 entra en ejecucion
1
Pid: 5 | Burst: 9 | prioridad: 5 | TAT: 4 | WT : 4 | Tiempo Restante: 9 | Estado: Ready.
Proceso: 5 con burst 9, tiempo restante: 9 y prioridad 5 entra en ejecucion
Proceso: 6 con burst 11, tiempo restante: 11 y prioridad 4 entra en ejecucion
Proceso: 7 con burst 19, tiempo restante: 19 y prioridad 3 entra en ejecucion
Proceso: 8 con burst 5, tiempo restante: 5 y prioridad 4 entra en ejecucion
El proceso: 8 acaba de salir de ejecucion
```

Debido a que la prueba anterior fue corrida en cliente automático, es imposible volver a recrear el mismo caso de prueba, sin embargo se arregló el bug: lo que pasaba es que si el tamaño de la lista es 1 (primeros seguros de la ejecución), el algoritmo se enciclaba, ya que trataba de buscar el siguiente, pero no existía, entonces volvía al inicio, intentaba buscar el siguiente y así infinitamente. Entonces es hasta que el proceso 2 entre que se des-enciclaba y el algoritmo se ejecutaba con normalidad. Se arregla el bug y se toma el caso en que solo hay un elemento en la lista.

Prueba 6: Resultados finales

Objetivo: se prueban los resultados finales, en este caso FIFO.
Resultados esperados: se espera obtener los mismos resultados al hacer los cálculos manualmente.

Resultados Obtenidos y entradas

Estos fueron los procesos que se ejecutaron y con los cuales se calcularon los resultados finales.

```
Pid: 1 | Burst: 15 | prioridad: 4 | TAT: 15 | WT : 0 | Estado: Finalizado.
Pid: 2 | Burst: 7 | prioridad: 4 | TAT: 22 | WT : 15 | Estado: Finalizado.
Pid: 3 | Burst: 1 | prioridad: 5 | TAT: 23 | WT : 22 | Estado: Finalizado.
Pid: 4 | Burst: 19 | prioridad: 4 | TAT: 27 | WT : 8 | Estado: Finalizado.
Pid: 5 | Burst: 14 | prioridad: 3 | TAT: 41 | WT : 27 | Estado: Finalizado.
Pid: 6 | Burst: 5 | prioridad: 2 | TAT: 38 | WT : 33 | Estado: Finalizado.
Pid: 7 | Burst: 10 | prioridad: 2 | TAT: 48 | WT : 38 | Estado: Finalizado.
Pid: 8 | Burst: 7 | prioridad: 2 | TAT: 55 | WT : 48 | Estado: Finalizado.
Pid: 9 | Burst: 6 | prioridad: 2 | TAT: 42 | WT : 36 | Estado: Finalizado.
Pid: 10 | Burst: 9 | prioridad: 3 | TAT: 37 | WT : 28 | Estado: Finalizado.
Pid: 11 | Burst: 2 | prioridad: 2 | TAT: 34 | WT : 32 | Estado: Finalizado.
Pid: 12 | Burst: 20 | prioridad: 3 | TAT: 54 | WT : 34 | Estado: Finalizado.
Pid: 13 | Burst: 3 | prioridad: 3 | TAT: 47 | WT : 44 | Estado: Finalizado.
Pid: 14 | Burst: 1 | prioridad: 4 | TAT: 48 | WT : 47 | Estado: Finalizado.
Pid: 15 | Burst: 14 | prioridad: 4 | TAT: 49 | WT : 35 | Estado: Finalizado.
Pid: 16 | Burst: 5 | prioridad: 3 | TAT: 54 | WT : 49 | Estado: Finalizado.
```

Resultados obtenidos:

```
Cantidad de procesos: 16
Cantidad de segundos ocioso: 18
Promedio TAT: 39.625000
Promedio WT: 31.000000
```

La cantidad de procesos se puede observar por medio de los PID. Los segundos ociosos es algo complicado de probar ya que el sistema no informa cuando está ocioso. Al hacer el cálculo del TAT manualmente da como resultado: 634/16 = 39.625 y el de WT: 496/16 = 31. Por lo tanto si dio los resultados esperados.

¿Cómo compilar y correr el Proyecto?

Ya que el proyecto está constituido de dos partes, se debe ir en terminal cada directorio respectivo:

Progra1/Cliente/ O Progra1/Server/ y ejecutar “make” para compilar.

Para correrlo, igual en cada directorio se utiliza ya sea: ./cliente o ./server para correr alguna de las dos partes. El cliente no presenta ninguna funcionalidad si el simulador no está corriendo, por lo que se requiere que el simulador se corra primero para que el cliente pueda ofrecer sus funcionalidades. Lo anterior es debido a que el cliente espera que su conexión del socket sea aceptada, al no serla, no puede seguir ejecutando y se cierra.