

Instituto Tecnológico de Costa Rica  
Escuela de Ingeniería en Computación

Principios de Sistemas Operativos  
Prof. Erika Marín Shumann

Planificador de CPU

Estudiantes:

Jake Herrera Hernández

2015106169

Jason Latouche Jiménez

2015146294

11 de abril

I Semestre 2018

# Índice

<b>Introducción</b>	<b>2</b>
<b>Estrategia de la solución</b>	<b>3</b>
Programa de simulación	3
Programa generador:	6
<b>Análisis de resultados</b>	<b>8</b>
<b>Lecciones aprendidas</b>	<b>9</b>
Trabajo en grupo:	9
Aspectos técnicos:	9
<b>Casos de prueba</b>	<b>10</b>
Prueba 1	10
Prueba 2	10
Prueba 3	11
Prueba 4	12
Prueba 5	13
<b>Comparación Pthreads y Java threads</b>	<b>16</b>
Usabilidad	16
Detalles técnicos	16
Rendimiento	16
<b>Manual de usuario</b>	<b>18</b>
Simulador	18
Generador	22
<b>Bitácora</b>	<b>27</b>
<b>Bibliografía</b>	<b>28</b>

# Introducción

Este trabajo tiene como objetivo desarrollar un simulador de planificador de CPU. El planificador es un componente funcional muy importante de los sistemas operativos multitarea y multiproceso, y es esencial en los sistemas operativos de tiempo real. Su función consiste en repartir el tiempo disponible de un microprocesador entre todos los procesos que están disponibles para su ejecución.

Un planificador puede usar diferentes métodos para elegir el siguiente proceso que se ejecutará en el CPU. El planificador de este simulador implementa los algoritmos FIFO, SJF, HPF y Round Robin con un quantum especificado por el usuario.

Además, este proyecto está desarrollado bajo un esquema cliente servidor, donde el Generador (cliente) enviará información de los procesos a ejecutar al Simulador (servidor).

El generador funciona de manera automática o manual. El modo manual lee desde un archivo la información de procesos (burst y prioridad). El modo automático crea un proceso con estos valores al azar. Cada vez que la información de un nuevo proceso se envía por socket, un nuevo hilo es creado para realizar la comunicación independientemente de la ejecución principal.

En el simulador el planificador de CPU recibe la información de los procesos, los pone en una cola de espera y empieza la simulación de los procesos.

El simulador tiene dos hilos principales: el Job Scheduler, quien se encarga de recibir la información de los procesos y ponerlos en la cola de ejecución, y el CPU Scheduler, quien se encarga administrar y asignar el recurso del CPU a cada proceso de la cola de ejecución.

Al finalizar cada simulación se muestran las estadísticas de los procesos ejecutados. Estas estadísticas muestran los datos de Waiting Time - que es el tiempo de inactividad - para cada proceso, así como el Turn Around Time - que se traduce como el tiempo que tarda un programa desde que entra hasta que finaliza. Además se muestran promedios de Waiting Time y Turn Around Time.

# Estrategia de la solución

El programa cuenta con dos partes, por un lado el programa que realiza la simulación del planificador que, en este caso, está contenido en un pequeño servidor creado para el proyecto, y por otro lado un programa cliente, que genera los procesos manipulados en el servidor, el cual también posee dos partes, el modo automático y el modo manual.

A continuación se detallan cada una de las partes mencionadas.

## Programa de simulación

- **Servidor**

Para la parte relativa al servidor, se establecen los métodos necesarios para permitir tanto esperar, escuchar y escribir en los clientes.

Por lo tanto se opta por crear el archivo Socket, el cual contiene las funciones para:

1. Configurar los datos del socket servidor y ubicarlo en un puerto específico.
2. Iniciar a escuchar a los clientes que van llegando.
3. Iniciar la comunicación con un cliente específico, según su orden de llegada, esta comunicación se hace a través de un thread por cliente, para lograr atenderlos a todos.
4. Una función auxiliar que realiza la conversión del mensaje recibido por el cliente para añadirlo a la cola de procesos.

- **Estructura PCB**

Esta estructura almacena los datos básicos del proceso, tales como su PID, burst, prioridad, tiempo de salida y tiempo de llegada, además de variables para ayudar en los cálculos necesarios y conocer su estado dentro de la cola de procesos.

Sobre dicha estructura se emplean funciones para:

1. Conocer el estado del proceso.
2. Cambiar las variables de estado y tiempos de salida.
3. Imprimir la información general del PCB
4. Imprimir el estado del proceso, una vez terminado su ciclo completo de ejecución para el turn around time(TAT) y waiting time(WT).
5. Calcular el TAT y WT.

- **Estructura PQueue**

Dado que el programa debe simular la ejecución de un del Job Scheduler y CPU scheduler, se establece crear una estructura tipo cola, cuyos nodos serán la representación de los procesos, estos nodos por su parte poseerán la estructura PCB ya mencionada.

Por otro lado, en este caso los procesos tendrán diferentes estados, ACTIVE, READY y ENDED, los cuales permitirán iterar sobre la cola para aplicar los algoritmos de búsqueda necesarios. Dada la existencia de estados en los nodos, se decide que estos no serán sacados de la cola, solo se cambiará su estado y de este dependen los resultados requeridos en la simulación.

Sobre esta estructura solo se tendrán funciones para:

1. Insertar nuevos nodos(Procesos).
2. Imprimir el estado de los procesos que contiene.
3. Movilizar el nodo actual en la cola si la simulación lo requiere.

- **Estructura Simulation**

Dado que la simulación está particionada en dos partes, JobScheduler y CPU Scheduler, se decide que exista una estructura que contenga los datos generales de la simulación según el proceso de planificación, tales como la cola de procesos, el tipo de algoritmo, los tiempos de reloj y ocioso, además del quantum si este se requiere, y una variable de control para saber si la simulación ha terminado o no.

Sobre esta estructura se decide emplear funciones para:

1. Pedir el tipo de algoritmo a utilizar.
2. cerrar la simulación.
3. incrementar los tiempos de reloj y ociosos.
4. Imprimir el resumen de la ejecución.

- **Job Scheduler**

Para simular esta parte de la planificación del SO, se decide que bastaría con emplear las funciones del socket, para gestionar los procesos enviados por el cliente, por lo tanto en este módulo, solo se activa un hilo que permita esperar clientes hasta que se cierre la simulación, si el usuario lo decide.

- **CPU Scheduler**

Para manipular los procesos en cola, se activa un hilo que empezará a leer la cola, todos los datos que se usen en este hilo serán los que estén dentro de la estructura simulación ya mencionada.

Por otro lado mientras no se lean procesos de la cola, se aumentarán los tiempos de reloj y ociosos en 1, también se aumentará en 1 los tiempos de reloj mientras los procesos estén en estado ACTIVE.

Dado que las ejecuciones no son apropiativas, se itera sobre el proceso poniéndolo en estado ACTIVE, hasta que este tenga un burst de 0 y pase a estado ENDED, excepto cuando se aplique el algoritmo RR, el cual estará en estado ACTIVE, hasta completar el quantum que el usuario ha definido, y luego regresa a un estado de READY, si su burst no ha terminado.

A continuación se explica el modo de selección de los algoritmos:

- ❖ FIFO: dado que la naturaleza del mismo es ejecutar de manera no apropiativa el proceso según su orden de llegada, la cola se mantendrá ordenada, teniendo a siempre a los procesos en estado ENDED al principio y los procesos en READY al final de la misma,

basta con buscar el nodo en la cola que posea un estado de READY, para ser retornado y ejecutado.

- ❖ SJF: Para este caso no se tendrá la cola de forma ordenada de todos los casos, por lo tanto, se toma el primer nodo que posea un estado de READY como el de menor burst; luego se compara con el resto a partir de ese nodo para saber cuál posee burst más bajo, luego se retorna y se ejecuta.
- ❖ HPF: la lógica aplicada es similar al SJF, salvo que en este caso el atributo a comparar es la prioridad de los PCB que poseen los nodos.
- ❖ RR: En este caso en particular al mantener el PID como parametro de búsqueda, se incurre en el uso del nodo actual de la cola, por tanto siempre se retornará este, el cual tomará la posición del primer nodo desde nodo actual con estado READY, luego para la siguiente búsqueda se posiciona a nodo actual en el que le sigue, para garantizar que se mueve entre los nodos por orden de PID.

En los cuatro casos si el nodo retornado es uno con estado ENDED este se ponen el NULL y no es procesado, y tiempo ocioso aumenta al igual que tiempos de reloj, esto para simular la cola del SO vacía.

## Programa generador:

- **Estructura socket**

Para este caso, se emplea una estructura de socket que contendrá la referencia del servidor al cual se le enviará el mensaje y también un puntero de char para almacenar dicho mensaje, además la comunicación siempre se ejecutará bajo un hilo distinto por mensaje.

Sobre esta estructura se establecen funciones para:

1. Conectar con el servidor.
2. Iniciar la comunicación con el servidor

- **Estructura Process**

El programa cliente debe poder generar valores random de burst y prioridad si se encuentran bajo el modo automático, para lo cual la estructura permite poder generar los valores del proceso y poder convertir a una cadena de texto para este sea enviado por medio del socket.

- **Estructura Message**

Esta estructura ayuda a almacenar los rangos de creación y del burst del proceso, el puerto y host del servidor.

Dado que el envío de procesos depende de dos modos manual y automático, se decide optar por usar un único método de envío de mensajes, implementar el envío automático a través de un hilo que genera valores random, el cual se interrumpe si el usuario lo decide; y para el modo manual se lee un archivo con los datos del proceso, por cada línea se realiza un envío de mensaje, y el programa cliente concluye una vez se ha leído todo el archivo.



# Análisis de resultados

Descripción	A	B	C	D	E
Cliente: procesos manuales (lectura del archivo)	X				
Cliente: procesos automáticos	X				
Cliente: threads por cada proceso	X				
Sockets entre cliente - servidor	X				
Simulador: thread por cada proceso recibido	X				
Simulador: Job Scheduler	X				
Simulador: CPU Scheduler	X				
Simulador: consultar cola	X				
Simulador: FIFO	X				
Simulador: SJF	X				
Simulador: HPF	X				
Simulador: RR	X				
Simulador: estadísticas de la ejecución	X				

donde:

**A:** Implementado

**B:** Implementado con errores leves

**C:** Implementado con errores graves

**D:** Implementación incompleta

**E:** No implementado

# Lecciones aprendidas

## 1. Trabajo en grupo:

- Mejorar la comunicación, en cuanto a que se espera del proyecto, establecer las prioridades, establecer formatos y estructura del mismo para evitar confusiones entre las partes.
- Comentar las ideas y luego ver cual de ellas se ve más viable para el desarrollo del proyecto.
- Establecer qué problemas o cambios en el proyecto deben ser notificados o cuáles de ellos se denotan como importantes.

## 2. Aspectos técnicos:

- La comunicación entre cliente y servidor no sigue un modo de envío de mensajes fijo, pues el cliente puede mantener la conexión abierta y con ella enviar un mensaje cada cierto tiempo o bien, abrir una nueva conexión enviar el mensaje y cerrar la conexión, todo dependerá de la necesidades del problema y el modo en que se quiera resolver.
- El servidor debe tener un buen control de cuántos clientes se recibirán en promedio, esto para evitar que los programas queden a la espera de nuevos clientes y esto de alguna forma evite el mismo quede colgado esperando nuevos clientes.
- En el manejo de hilos, es importante mantener una variable de control, que se pueda pasar por referencia, de manera que mantenga activos los hilos hasta que el usuario defina lo contrario.
- Si los datos que debe procesar un hilo son múltiples, es aconsejable crear estructuras que puedan enviarse a los hilos para que ellos las procesen.
- Aun cuando el lenguaje C no es orientado a objetos, este cuenta con structs, los cuales permiten hasta cierto punto, hacer que el programa mantenga un orden similar a uno orientado a objetos, con la diferencia que las funciones siempre deberán recibir el struct asociado.

# Casos de prueba

## Prueba 1

Se ejecutan 11 procesos con el algoritmo FIFO. La ejecución se realiza de la siguiente manera:

```

* --> PID: 1 | Initial Burst: 1 | Actual burst: 1 | Priority: 2 | Arrival time:
0
* --> PID: 2 | Initial Burst: 4 | Actual burst: 4 | Priority: 5 | Arrival time:
5
* --> PID: 3 | Initial Burst: 5 | Actual burst: 5 | Priority: 9 | Arrival time:
9
* --> PID: 4 | Initial Burst: 7 | Actual burst: 7 | Priority: 8 | Arrival time:
17
* --> PID: 5 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
25
* --> PID: 6 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
31
* --> PID: 7 | Initial Burst: 1 | Actual burst: 1 | Priority: 5 | Arrival time:
39
* --> PID: 8 | Initial Burst: 6 | Actual burst: 6 | Priority: 1 | Arrival time:
42
* --> PID: 9 | Initial Burst: 6 | Actual burst: 6 | Priority: 2 | Arrival time:
49
* --> PID: 10 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
54
* --> PID: 11 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
59

```

Las estadísticas resultan en:

```

*-----> Generating summary <-----
* PID: 1 | TAT: 1 | WT: 0 | Exit_time 1
* PID: 2 | TAT: 4 | WT: 0 | Exit_time 9
* PID: 3 | TAT: 5 | WT: 0 | Exit_time 14
* PID: 4 | TAT: 7 | WT: 0 | Exit_time 24
* PID: 5 | TAT: 8 | WT: 0 | Exit_time 33
* PID: 6 | TAT: 10 | WT: 2 | Exit_time 41
* PID: 7 | TAT: 3 | WT: 2 | Exit_time 42
* PID: 8 | TAT: 6 | WT: 0 | Exit_time 48
* PID: 9 | TAT: 6 | WT: 0 | Exit_time 55
* PID: 10 | TAT: 5 | WT: 1 | Exit_time 59
* PID: 11 | TAT: 4 | WT: 0 | Exit_time 63

* 1) Number of processes executed: 11
* 2) Idle CPU Time: 14
* 3) Average Turn Around Time: 5.363636
* 4) Average Waiting Time: 0.454545

```

## Prueba 2

Se ejecutan 11 procesos con el algoritmo SJF. La ejecución se realiza de la siguiente manera:

```

* --> PID: 1 | Initial Burst: 1 | Actual burst: 1 | Priority: 2 | Arrival time:
0
* --> PID: 2 | Initial Burst: 4 | Actual burst: 4 | Priority: 5 | Arrival time:
7

```

```

* --> PID: 3 | Initial Burst: 5 | Actual burst: 5 | Priority: 9 | Arrival time:
13
* --> PID: 4 | Initial Burst: 7 | Actual burst: 7 | Priority: 8 | Arrival time:
20
* --> PID: 5 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
26
* --> PID: 6 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
29
* --> PID: 7 | Initial Burst: 1 | Actual burst: 1 | Priority: 5 | Arrival time:
37
* --> PID: 8 | Initial Burst: 6 | Actual burst: 6 | Priority: 1 | Arrival time:
45
* --> PID: 9 | Initial Burst: 6 | Actual burst: 6 | Priority: 2 | Arrival time:
48
* --> PID: 10 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
53
* --> PID: 11 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
59

```

Las estadísticas resultan en:

```

*-----> Generating summary <-----
* PID: 1 | TAT: 1 | WT: 0 | Exit_time 1
* PID: 2 | TAT: 4 | WT: 0 | Exit_time 11
* PID: 3 | TAT: 5 | WT: 0 | Exit_time 18
* PID: 4 | TAT: 7 | WT: 0 | Exit_time 27
* PID: 5 | TAT: 9 | WT: 1 | Exit_time 35
* PID: 6 | TAT: 14 | WT: 6 | Exit_time 43
* PID: 7 | TAT: 7 | WT: 6 | Exit_time 44
* PID: 8 | TAT: 6 | WT: 0 | Exit_time 51
* PID: 9 | TAT: 9 | WT: 3 | Exit_time 57
* PID: 10 | TAT: 8 | WT: 4 | Exit_time 61
* PID: 11 | TAT: 6 | WT: 2 | Exit_time 65

* 1) Number of processes executed: 11
* 2) Idle CPU Time: 25
* 3) Average Turn Around Time: 6.909091
* 4) Average Waiting Time: 2.000000

```

### Prueba 3

Se ejecutan 11 procesos con el algoritmo HPF. La ejecución se realiza de la siguiente manera:

```

* --> PID: 1 | Initial Burst: 1 | Actual burst: 1 | Priority: 2 | Arrival time:
0
* --> PID: 2 | Initial Burst: 4 | Actual burst: 4 | Priority: 5 | Arrival time:
7
* --> PID: 3 | Initial Burst: 5 | Actual burst: 5 | Priority: 9 | Arrival time:
12
* --> PID: 4 | Initial Burst: 7 | Actual burst: 7 | Priority: 8 | Arrival time:
19

```

```

* --> PID: 5 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
24
* --> PID: 7 | Initial Burst: 1 | Actual burst: 1 | Priority: 5 | Arrival time:
31
* --> PID: 6 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
27
* --> PID: 8 | Initial Burst: 6 | Actual burst: 6 | Priority: 1 | Arrival time:
38
* --> PID: 9 | Initial Burst: 6 | Actual burst: 6 | Priority: 2 | Arrival time:
46
* --> PID: 10 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
54
* --> PID: 11 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
60

```

Las estadísticas resultan en:

```

*-----> Generating summary <-----
* PID: 1 | TAT: 1 | WT: 0 | Exit_time 1
* PID: 2 | TAT: 4 | WT: 0 | Exit_time 11
* PID: 3 | TAT: 5 | WT: 0 | Exit_time 17
* PID: 4 | TAT: 7 | WT: 0 | Exit_time 26
* PID: 5 | TAT: 10 | WT: 2 | Exit_time 34
* PID: 6 | TAT: 16 | WT: 8 | Exit_time 43
* PID: 7 | TAT: 4 | WT: 3 | Exit_time 35
* PID: 8 | TAT: 11 | WT: 5 | Exit_time 49
* PID: 9 | TAT: 9 | WT: 3 | Exit_time 55
* PID: 10 | TAT: 5 | WT: 1 | Exit_time 59
* PID: 11 | TAT: 4 | WT: 0 | Exit_time 64

* 1) Number of processes executed: 11
* 2) Idle CPU Time: 26
* 3) Average Turn Around Time: 6.909091
* 4) Average Waiting Time: 2.000000

```

## Prueba 4

Se ejecutan 11 procesos con el algoritmo RR y quantum de 2. La ejecución se realiza de la siguiente manera:

```

* --> PID: 1 | Initial Burst: 1 | Actual burst: 1 | Priority: 2 | Arrival time:
0
* --> PID: 2 | Initial Burst: 4 | Actual burst: 4 | Priority: 5 | Arrival time:
8
* --> PID: 2 | Initial Burst: 4 | Actual burst: 2 | Priority: 5 | Arrival time:
8
* --> PID: 3 | Initial Burst: 5 | Actual burst: 5 | Priority: 9 | Arrival time:
16
* --> PID: 3 | Initial Burst: 5 | Actual burst: 3 | Priority: 9 | Arrival time:
16
* --> PID: 3 | Initial Burst: 5 | Actual burst: 1 | Priority: 9 | Arrival time:
16

```

```

* --> PID: 4 | Initial Burst: 7 | Actual burst: 7 | Priority: 8 | Arrival time:
21
* --> PID: 4 | Initial Burst: 7 | Actual burst: 5 | Priority: 8 | Arrival time:
21
* --> PID: 4 | Initial Burst: 7 | Actual burst: 3 | Priority: 8 | Arrival time:
21
* --> PID: 5 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
27
* --> PID: 4 | Initial Burst: 7 | Actual burst: 1 | Priority: 8 | Arrival time:
21
* --> PID: 5 | Initial Burst: 8 | Actual burst: 6 | Priority: 7 | Arrival time:
27
* --> PID: 6 | Initial Burst: 8 | Actual burst: 8 | Priority: 7 | Arrival time:
30
* --> PID: 7 | Initial Burst: 1 | Actual burst: 1 | Priority: 5 | Arrival time:
33
* --> PID: 5 | Initial Burst: 8 | Actual burst: 4 | Priority: 7 | Arrival time:
27
* --> PID: 6 | Initial Burst: 8 | Actual burst: 6 | Priority: 7 | Arrival time:
30
* --> PID: 5 | Initial Burst: 8 | Actual burst: 2 | Priority: 7 | Arrival time:
27
* --> PID: 6 | Initial Burst: 8 | Actual burst: 4 | Priority: 7 | Arrival time:
30
* --> PID: 8 | Initial Burst: 6 | Actual burst: 6 | Priority: 1 | Arrival time:
41
* --> PID: 6 | Initial Burst: 8 | Actual burst: 2 | Priority: 7 | Arrival time:
30
* --> PID: 8 | Initial Burst: 6 | Actual burst: 4 | Priority: 1 | Arrival time:
4
* --> PID: 9 | Initial Burst: 6 | Actual burst: 6 | Priority: 2 | Arrival time:
47
* --> PID: 10 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
51
* --> PID: 8 | Initial Burst: 6 | Actual burst: 2 | Priority: 1 | Arrival time:
41
* --> PID: 9 | Initial Burst: 6 | Actual burst: 4 | Priority: 2 | Arrival time:
47
* --> PID: 10 | Initial Burst: 4 | Actual burst: 2 | Priority: 6 | Arrival time:
51
* --> PID: 11 | Initial Burst: 4 | Actual burst: 4 | Priority: 6 | Arrival time:
55
* --> PID: 9 | Initial Burst: 6 | Actual burst: 2 | Priority: 2 | Arrival time:
47
* --> PID: 11 | Initial Burst: 4 | Actual burst: 2 | Priority: 6 | Arrival time:
55

```

Las estadísticas resultan en:

```

*-----> Generating summary <-----
* PID: 1 | TAT: 1 | WT: 0 | Exit_time 1
* PID: 2 | TAT: 4 | WT: 0 | Exit_time 12
* PID: 3 | TAT: 5 | WT: 0 | Exit_time 21
* PID: 4 | TAT: 9 | WT: 2 | Exit_time 30

```

```

* PID: 5 | TAT: 15 | WT: 7 | Exit_time 42
* PID: 6 | TAT: 18 | WT: 10 | Exit_time 48
* PID: 7 | TAT: 2 | WT: 1 | Exit_time 35
* PID: 8 | TAT: 15 | WT: 9 | Exit_time 56
* PID: 9 | TAT: 17 | WT: 11 | Exit_time 64
* PID: 10 | TAT: 9 | WT: 5 | Exit_time 60
* PID: 11 | TAT: 11 | WT: 7 | Exit_time 66

* 1) Number of processes executed: 11
* 2) Idle CPU Time: 24
* 3) Average Turn Around Time: 9.636364
* 4) Average Waiting Time: 4.727273

```

## Prueba 5

Se presente poner a prueba el Simulador con 3 Generadores al mismo tiempo. El algoritmo a analizar será SJF. Los clientes generan procesos nuevos con prioridades, burst y taza de creación entre 1 y 10.

```

--> PID: 1 | Initial Burst: 5 | Actual burst: 5 | Priority: 4 | Arrival time: 0
--> PID: 3 | Initial Burst: 1 | Actual burst: 1 | Priority: 2 | Arrival time: 3
--> PID: 2 | Initial Burst: 4 | Actual burst: 4 | Priority: 7 | Arrival time: 2
--> PID: 4 | Initial Burst: 7 | Actual burst: 7 | Priority: 5 | Arrival time: 6
--> PID: 8 | Initial Burst: 3 | Actual burst: 3 | Priority: 7 | Arrival time:
14
--> PID: 9 | Initial Burst: 3 | Actual burst: 3 | Priority: 5 | Arrival time:
17
--> PID: 10 | Initial Burst: 3 | Actual burst: 3 | Priority: 5 | Arrival time:
20
--> PID: 11 | Initial Burst: 6 | Actual burst: 6 | Priority: 8 | Arrival time:
25
--> PID: 13 | Initial Burst: 3 | Actual burst: 3 | Priority: 9 | Arrival time:
28
--> PID: 17 | Initial Burst: 5 | Actual burst: 5 | Priority: 3 | Arrival time:
35
--> PID: 18 | Initial Burst: 2 | Actual burst: 2 | Priority: 6 | Arrival time:
37
--> PID: 20 | Initial Burst: 2 | Actual burst: 2 | Priority: 5 | Arrival time:
42
--> PID: 19 | Initial Burst: 5 | Actual burst: 5 | Priority: 1 | Arrival time:
42
--> PID: 23 | Initial Burst: 3 | Actual burst: 3 | Priority: 7 | Arrival time:
48
--> PID: 24 | Initial Burst: 1 | Actual burst: 1 | Priority: 8 | Arrival time:
50
--> PID: 27 | Initial Burst: 2 | Actual burst: 2 | Priority: 7 | Arrival time:
53
--> PID: 29 | Initial Burst: 3 | Actual burst: 3 | Priority: 8 | Arrival time:
55
--> PID: 30 | Initial Burst: 3 | Actual burst: 3 | Priority: 2 | Arrival time:
56
--> PID: 34 | Initial Burst: 2 | Actual burst: 2 | Priority: 6 | Arrival time:
61
--> PID: 22 | Initial Burst: 4 | Actual burst: 4 | Priority: 5 | Arrival time:

```

```

48
--> PID: 26 | Initial Burst: 4 | Actual burst: 4 | Priority: 4 | Arrival time:
53
--> PID: 37 | Initial Burst: 2 | Actual burst: 2 | Priority: 1 | Arrival time:
69
--> PID: 32 | Initial Burst: 4 | Actual burst: 4 | Priority: 4 | Arrival time:
60
--> PID: 33 | Initial Burst: 6 | Actual burst: 6 | Priority: 3 | Arrival time:
61
--> PID: 35 | Initial Burst: 6 | Actual burst: 6 | Priority: 4 | Arrival time:
65
--> PID: 38 | Initial Burst: 6 | Actual burst: 6 | Priority: 8 | Arrival time:
71
--> PID: 6 | Initial Burst: 7 | Actual burst: 7 | Priority: 9 | Arrival time: 7
--> PID: 7 | Initial Burst: 7 | Actual burst: 7 | Priority: 3 | Arrival time:
10
--> PID: 12 | Initial Burst: 7 | Actual burst: 7 | Priority: 1 | Arrival time:
26
--> PID: 21 | Initial Burst: 7 | Actual burst: 7 | Priority: 2 | Arrival time:
44
--> PID: 39 | Initial Burst: 7 | Actual burst: 7 | Priority: 3 | Arrival time:
73
--> PID: 5 | Initial Burst: 8 | Actual burst: 8 | Priority: 4 | Arrival time: 7
--> PID: 14 | Initial Burst: 8 | Actual burst: 8 | Priority: 4 | Arrival time:
30
--> PID: 16 | Initial Burst: 8 | Actual burst: 8 | Priority: 2 | Arrival time:
34
--> PID: 25 | Initial Burst: 8 | Actual burst: 8 | Priority: 1 | Arrival time:
52
--> PID: 28 | Initial Burst: 8 | Actual burst: 8 | Priority: 1 | Arrival time:
53
--> PID: 36 | Initial Burst: 8 | Actual burst: 8 | Priority: 4 | Arrival time:
67
--> PID: 15 | Initial Burst: 9 | Actual burst: 9 | Priority: 2 | Arrival time:
34
--> PID: 31 | Initial Burst: 9 | Actual burst: 9 | Priority: 2 | Arrival time:
57

```

Como podemos analizar, el planificador va dejando de último los procesos con mayor burst. Ahora veamos las estadísticas:

```

*-----> Generating summary <-----
* PID: 1 | TAT: 5 | WT: 0 | Exit_time 5
* PID: 2 | TAT: 8 | WT: 4 | Exit_time 10
* PID: 3 | TAT: 3 | WT: 2 | Exit_time 6
* PID: 4 | TAT: 11 | WT: 4 | Exit_time 17
* PID: 5 | TAT: 131 | WT: 123 | Exit_time 138
* PID: 6 | TAT: 95 | WT: 88 | Exit_time 102
* PID: 7 | TAT: 99 | WT: 92 | Exit_time 109
* PID: 8 | TAT: 6 | WT: 3 | Exit_time 20
* PID: 9 | TAT: 6 | WT: 3 | Exit_time 23
* PID: 10 | TAT: 6 | WT: 3 | Exit_time 26
* PID: 11 | TAT: 7 | WT: 1 | Exit_time 32
* PID: 12 | TAT: 90 | WT: 83 | Exit_time 116

```



```

* PID: 13 | TAT: 7 | WT: 4 | Exit_time 35
* PID: 14 | TAT: 116 | WT: 108 | Exit_time 146
* PID: 15 | TAT: 153 | WT: 144 | Exit_time 187
* PID: 16 | TAT: 120 | WT: 112 | Exit_time 154
* PID: 17 | TAT: 5 | WT: 0 | Exit_time 40
* PID: 18 | TAT: 5 | WT: 3 | Exit_time 42
* PID: 19 | TAT: 7 | WT: 2 | Exit_time 49
* PID: 20 | TAT: 2 | WT: 0 | Exit_time 44
* PID: 21 | TAT: 79 | WT: 72 | Exit_time 123
* PID: 22 | TAT: 19 | WT: 15 | Exit_time 67
* PID: 23 | TAT: 4 | WT: 1 | Exit_time 52
* PID: 24 | TAT: 3 | WT: 2 | Exit_time 53
* PID: 25 | TAT: 110 | WT: 102 | Exit_time 162
* PID: 26 | TAT: 18 | WT: 14 | Exit_time 71
* PID: 27 | TAT: 2 | WT: 0 | Exit_time 55
* PID: 28 | TAT: 117 | WT: 109 | Exit_time 170
* PID: 29 | TAT: 3 | WT: 0 | Exit_time 58
* PID: 30 | TAT: 5 | WT: 2 | Exit_time 61
* PID: 31 | TAT: 139 | WT: 130 | Exit_time 196
* PID: 32 | TAT: 17 | WT: 13 | Exit_time 77
* PID: 33 | TAT: 22 | WT: 16 | Exit_time 83
* PID: 34 | TAT: 2 | WT: 0 | Exit_time 63
* PID: 35 | TAT: 24 | WT: 18 | Exit_time 89
* PID: 36 | TAT: 111 | WT: 103 | Exit_time 178
* PID: 37 | TAT: 4 | WT: 2 | Exit_time 73
* PID: 38 | TAT: 24 | WT: 18 | Exit_time 95
* PID: 39 | TAT: 57 | WT: 50 | Exit_time 130

* 1) Number of processes executed: 39
* 2) Idle CPU Time: 117
* 3) Average Turn Around Time: 42.102566
* 4) Average Waiting Time: 37.076923

```

# Comparación Pthreads y Java threads

Según el estudio comparativo realizado con Castellanos, podemos realizar dicha comparación en tres aspectos:

## Usabilidad

- En primer lugar los java threads, son implementados a través de un objeto, lo cual obliga al programador crear una clase hija para realizar la tarea deseada, y las funciones asociadas al hilo están ligadas al objeto creado.
- Si bien es cierto los hilos creados bajo pthread requiere mantener datos y funciones por separado, no como un objeto como en java threads, pero de cierto modo da más control al poder manipular los datos del hilo de modo externo, aparte de las funciones que brinda la librería pthread para manipularlos.

## Detalles técnicos

- La creación del hilo se da bajo un objeto en java y mediante la llamada de la función `pthread_create` en c.
- java threads al ser objetos es más fácil de comparar dos hilos, para ello con la librería pthread se debe hacer uso de la función `pthread_equal(id1, id2)`.
- Los java threads pueden recibir varios atributos para su ejecución, pero aquellos hilos de tipo pthread solo pueden recibir el puntero de un único parámetro, si se desean pasar múltiples parámetros se deben hacer uso de *struct*.
- Los java threads tienen un alcance solo 10 prioridades, por otro lado pthreads tiene la posibilidad de 32 diferentes prioridades.
- Java thread solo permite detener el hilo por medio de la función `stop()`, ahora bien pthread posee `pthread_exit()` para detener el hilo desde sí mismo. Por otra parte se pueden cerrar hilos desde otros hilos usando las funciones de `pthread_kill()` y `pthread_cancel()`.

## Rendimiento

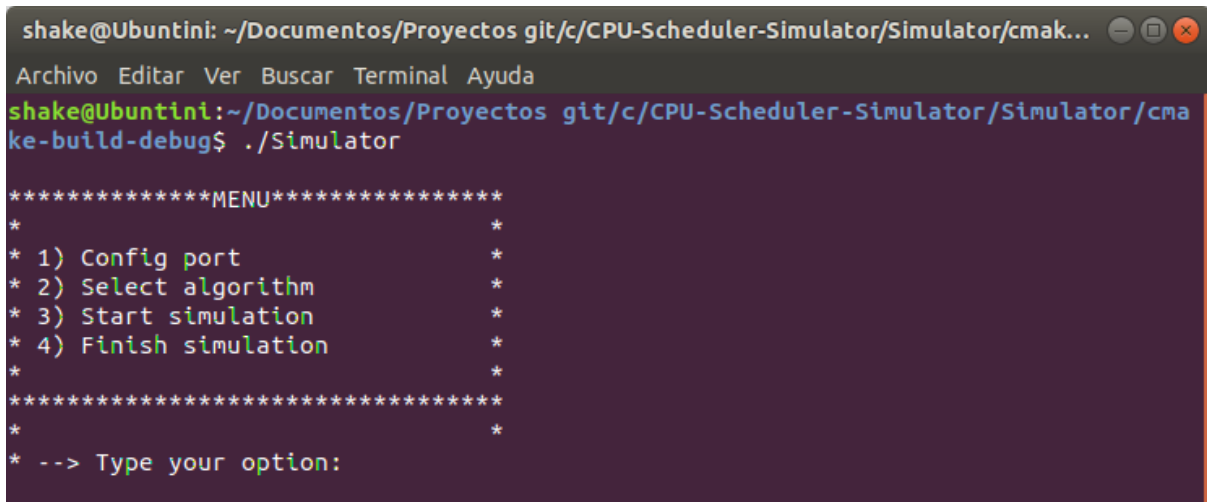
- La sincronización en ambos casos está presente, por un lado pthread presenta los mutex y por otro lado java da los llamados monitores, para pthread requiere que uno de los hilos maneje las funciones de `pthread_mutex_lock` y `pthread_mutex_unlock` para poder garantizar la

exclusividad de las operaciones sobre determinado espacio de memoria, o el uso de la funciones de *pthread\_join* en caso de sincronización secuencial, para el caso de java los monitores pueden reducir ciertos errores que c pueda presentar, pues java permite reservar espacios específicos para exclusión mutua para multitareas, con la diferencias que, por las características propias del lenguaje Java el código es más complejo, comparado la solución de un mismo problema resuelto en c, pero aun así los mecanismos de java permiten capturar error de manera más sencilla, pues el manejo de los mismo en el lenguaje C son más difíciles de capturar.

# Manual de usuario

## Simulador

- Ejecutar simulador
  1. Abrir la ubicación de proyecto, específicamente en CPU-Scheduler-Simulator/Simulator/cmake-build-debug
  2. Abrir una consola.
  3. Digitar la la instrucción *./Sumulator*



```
shake@Ubtuntini: ~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Simulator/cmak...
Archivo Editar Ver Buscar Terminal Ayuda
shake@Ubtuntini:~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Simulator/cma
ke-build-debug$ ./Simulator

*****MENU*****
*
* 1) Config port
* 2) Select algorithm
* 3) Start simulation
* 4) Finish simulation
*
*****
*
* --> Type your option:
```

- Configuración
  1. El simulador ya tiene definido el puerto 8080 por defecto, pero este puede ser cambiado digitando la opción 1 del menú.
  2. Después de digitado el puerto solo se debe dar un enter.
  3. Y listo el el puerto queda configurado.

```
shake@Ubuntini: ~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Simulator/cmak...
Archivo Editar Ver Buscar Terminal Ayuda
shake@Ubuntini:~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Simulator/cma
ke-build-debug$ ./Simulator

*****MENU*****
*
* 1) Config port
* 2) Select algorithm
* 3) Start simulation
* 4) Finish simulation
*
*****
*
* --> Type your option: 1
* --> Type new port: 9898
```

- Seleccionar Algoritmo de la ejecución
  1. Digitar la opción 2 del menú.
  2. Los algoritmos FIFO, SJF, HPF, no requieren ninguna configuración adicional, solo deben ser seleccionados, ya se posee el FIFO por defecto.

```
*****MENU*****
*
* 1) Config port
* 2) Select algorithm
* 3) Start simulation
* 4) Finish simulation
*
*****
*
* --> Type your option: 2

*****Algorithms*****
*
* 1) FIFO
* 2) SJF
* 3) HPF
* 4) RR
*
*****
*
* --> Type your option:
```

3. Para este ejemplo se demostrará la opción 4, RR
4. Una vez seleccionado RR, se debe digitar la cantidad de quantum cual desea trabajar, dar un enter y regresará al menú principal.

```

*****Algorithms*****
*
* 1) FIFO
* 2) SJF
* 3) HPF
* 4) RR
*
*****
*
* --> Type your option: 4
* --> Type quantum value: 3
*
*****MENU*****
*
* 1) Config port
* 2) Select algorithm
* 3) Start simulation
* 4) Finish simulation
*
*****
*
* --> Type your option:

```

- Iniciar la simulación
  1. seleccionar la opción 3, una vez dado enter iniciará a esperar a los clientes.

```

*****MENU*****
*
* 1) Config port
* 2) Select algorithm
* 3) Start simulation
* 4) Finish simulation
*
*****
*
* --> Type your option: 3
*
***** SUB MENU *****
*
* 1) See log
* 2) Print ready process
* 3) Finish simulation
*
*****
*
*--> Type your option:

```

- Activar mensajes
  1. si se desea activar los mensajes de los procesos ejecutados se debe seleccionar la opción 1 del submenú.
  2. si se desea desactivar basta con digitar cualquier tecla y enter.

```

***** SUB MENU *****
*
* 1) See log
* 2) Print ready process
* 3) Finish simulation
*
*****
*
*--> Type your option: 1
* Press any key to stop the log...

```

```

* --> PID: 1 | Iinitial Burst: 1 | Actual burst: 1 | Priority: 2 | Arrival time: 0
* Process To Execute:
* --> PID: 2 | Iinitial Burst: 2 | Actual burst: 2 | Priority: 1 | Arrival time: 4
* Process To Execute:
* --> PID: 3 | Iinitial Burst: 5 | Actual burst: 5 | Priority: 5 | Arrival time: 8
* Process To Execute:
* --> PID: 3 | Iinitial Burst: 5 | Actual burst: 2 | Priority: 5 | Arrival time: 8
1
* Process To Execute:
* --> PID: 4 | Iinitial Burst: 7 | Actual burst: 7 | Priority: 1 | Arrival time: 14

```

- Ver procesos en cola
  1. Desactivar los mensajes de los proceso en ejecución.
  2. Digitar la opción 2 del submenú.
  3. Dar enter y aparecerá en pantalla los procesos que hay en cola, puede no haber ninguno.

```

***** SUB MENU *****
*
* 1) See log
* 2) Print ready process
* 3) Finish simulation
*
*****
*
*--> Type your option: 2
* -----> Log <-----
* --> PID: 4 | Iinitial Burst: 7 | Actual burst: 1 | Priority: 1 | Arrival time: 14

```

- Finalizar simulación.
  1. Digitar la opción 3 del submenú.
  2. Aparecerán todas el resumen de la ejecución de los procesos.

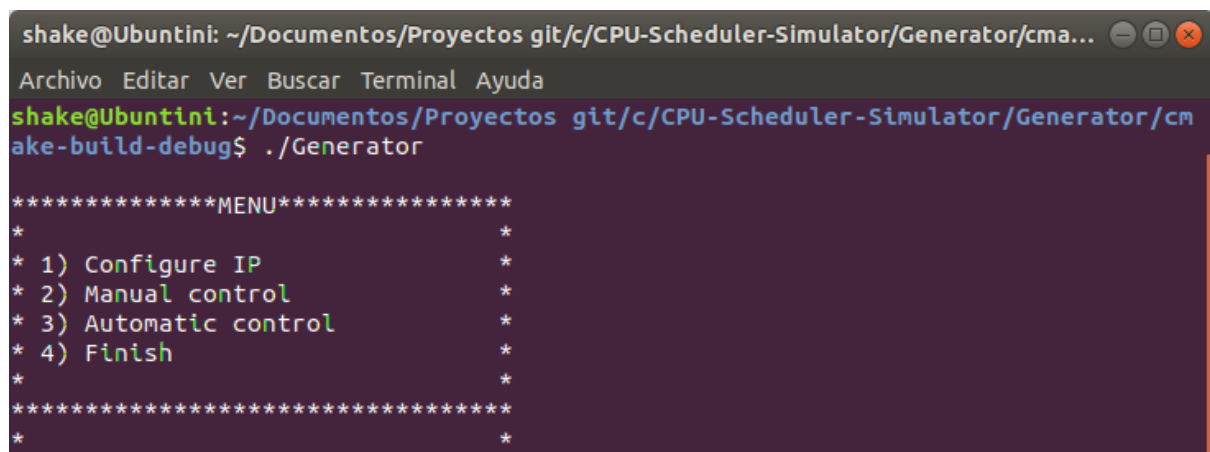
```

***** SUB MENU *****
*
* 1) See log
* 2) Print ready process
* 3) Finish simulation
*
*****
*
*--> Type your option: 3
*-----> Generating summary <-----
* PID: 1 | TAT: 1 | WT: 0 | Exit_time 1
* PID: 2 | TAT: 2 | WT: 0 | Exit_time 6
* PID: 3 | TAT: 5 | WT: 0 | Exit_time 13
* PID: 4 | TAT: 9 | WT: 2 | Exit_time 23
* PID: 5 | TAT: 4 | WT: 2 | Exit_time 22
* PID: 6 | TAT: 4 | WT: 0 | Exit_time 27
* PID: 7 | TAT: 2 | WT: 0 | Exit_time 30
* PID: 8 | TAT: 4 | WT: 0 | Exit_time 37
*
* 1) Number of processes executed: 8
* 2) Idle CPU Time: 434
* 3) Average Turn Around Time: 3
* 4) Average Waiting Time: 0

```

## Generador

- Ejecutar proyecto
  1. Abrir la ubicación de proyecto, específicamente en CPU-Scheduler-Simulator/Generator/cmake-build-debug
  2. Abrir una consola.
  3. Digitar la la instrucción *./Generator*



```

shake@Ubutini: ~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Generator/cma...
Archivo Editar Ver Buscar Terminal Ayuda
shake@Ubutini:~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Generator/cm
ake-build-debug$ ./Generator

*****MENU*****
*
* 1) Configure IP
* 2) Manual control
* 3) Automatic control
* 4) Finish
*
*****
*

```

- Configurar socket cliente
  1. Esta opción es opcional
  2. Seleccionar la opción uno del menú, el cual es opcional.
  3. Digitar la IP, del servidor, siempre estaía por defecto el IP, *127.0.0.1*
  4. Digitar el puerto, siempre estará por defecto el 8080.
  5. Luego regresará al menú principal



```
shake@Ubutini: ~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Generator/cma...
Archivo Editar Ver Buscar Terminal Ayuda
shake@Ubutini:~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Generator/cm
ake-build-debug$ ./Generator

*****MENU*****
*
* 1) Configure IP
* 2) Manual control
* 3) Automatic control
* 4) Finish
*
*****
* --> Type your option: 1
* --> IP: 127.0.0.1
* --> Port: 9898

*****MENU*****
*
* 1) Configure IP
* 2) Manual control
* 3) Automatic control
* 4) Finish
*
*****
```

- Opción Manual
  1. Asegurar de tener el archivo data.txt en la ubicación CPU-Scheduler-Simulator/Generator/cmake-build-debug del proyecto
  2. Tener el archivo de la forma burst -tab-priority en cada lines que se posea.
  3. Asegurar de hacer un salto de línea siempre en la última línea.
  4. Una vez teniendo esto en cuenta, digitar la opción 2, el programa empezará a enviar todo los procesos al servidor, según el archivo.

```

*                                     *
* --> Type your option: 2
*
*****MENU*****
*                                     *
* 1) Configure IP                    *
* 2) Manual control                  *
* 3) Automatic control               *
* 4) Finish                          *
*                                     *
*****
*                                     *
* --> Type your option:
Waiting interval: 5
Sending process...
Assigned PID: 1

Waiting interval: 6
Sending process...
Assigned PID: 2

```

- Opción automático.
  1. Digitar la opción 3 del menú
  2. Digitar los rangos de creación de los procesos, el más bajo primero y luego el más alto.
  3. Digitar los rangos para los burs de los procesos, el más bajo primero y luego el más alto.

```
shake@Ubutini: ~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Generator/cma...
Archivo Editar Ver Buscar Terminal Ayuda
*****MENU*****
*
* 1) Configure IP
* 2) Manual control
* 3) Automatic control
* 4) Finish
*
*****
*
* --> Type your option: 3
* --> Type the creation range:
* --> Lower: 1
* --> High: 3
* --> Type the burst range:
* --> Lower: 2
* --> High: 7
*
*****MENU*****
*
* 1) Configure IP
* 2) Manual control
* 3) Automatic control
* 4) Finish
*
*****
*
* --> Type your option: Waiting interval: 2
```

4. se empezará a ver en pantalla el intervalo de creación para cada proceso y el PID asignado por el servidor.

```
shake@Ubtuntini: ~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Generator/cma...
Archivo Editar Ver Buscar Terminal Ayuda
* --> High: 7

*****MENU*****
*
* 1) Configure IP
* 2) Manual control
* 3) Automatic control
* 4) Finish
*
*****
*
* --> Type your option: Waiting interval: 2
Sending process...
Assigned PID: 9
Waiting interval: 2
Sending process...
Assigned PID: 10
Waiting interval: 1
Sending process...
Assigned PID: 11
Waiting interval: 3
Sending process...
Assigned PID: 12
Waiting interval: 3
Sending process...
Assigned PID: 13
Waiting interval: 2
```

- Finalizar ejecución

1. Digitar la opción 4 del menú si se desea cancelar el envío de procesos.

```
Assigned PID: 68
4

*-----> Closing simulation <-----*
shake@Ubtuntini:~/Documentos/Proyectos git/c/CPU-Scheduler-Simulator/Generator/cm
ake-build-debug$
```

# Bitácora

Fecha	Actividad
29/3/18	Se investiga el funcionamiento de pthreads.
30/3/18	Se investiga el funcionamiento de sockets.
31/3/18	Se implementó un ejemplo de sockets usando pthreads.
1/4/18	Se inicializa un nuevo repositorio en GitHub y se agregan los nuevos proyectos. Se decide que el cliente se llamará Generador y el servidor se llamará Simulador.
2/4/18	Se crean las estructuras básicas para el Generador. Se le implementan los sockets y los threads para cada nueva conexión. Se prueban exitosamente.
3/4/18	Se crean las estructuras básicas para el Simulador. Se le implementan los sockets para el servicio de escucha y un thread para cada nueva información de proceso que se quiere recibir. Se prueban junto con el Generador exitosamente
4/4/18	Se implementa la estructura de cola en el Simulador. Esta estructura funcionará como lista de espera entre el job scheduler y el cpu scheduler.
5/4/18	Se implementan los algoritmos FIFO, HPF, SJF y RR. Se analizan y se corrigen bugs encontrados en el proceso.
7/4/18	Se comenta el código, se genera un menú bien corrongo y se refactoriza el código para optimizar el funcionamiento.
8/4/18	Se implementa el modo automático y manual en el Generador de procesos.
9/4/18	Se ha parametrizado la configuración de ambos programas y se ha mejorado la eficiencia. Se inicia la documentación.
10/4/18	Se realizan pruebas para agregar a la documentación. Se toman capturas de pantallas y se agrega una ayuda al programa.

# Bibliografía

ESTOS ENLACES SE DEBEN PONER MÁS LINDOS XD

1. <https://es.wikipedia.org/wiki/Planificador>
2. <http://servicio.bc.uc.edu.ve/facyt/v1n1/1-1-7.pdf>