Instituto tecnológico de Costa Rica
Escuela de ingeniería en computación sede central
Curso IC-6600
Principios de sistemas operativos

Documentación del primer proyecto de sistemas operativos.
El proyecto fue elaborado en el I semestres del 2018 por el grupo 1 de principios de sistemas operativos

Documentación realizada por:

Olman Castillo Picado 2015148651
Jimmy Fallas Delgado 2015099456
Pablo Navarro Altamirano 2015114121

10 de abril del 2018

I-Introducción

El correcto desempeño de un sistema operativo depende de sus funcionalidades y gestores, unas de las funcionalidades importantes son el CPU scheduler y el JOB scheduler. Los cuales está encargado de distribuir y gestionar la entrada de los procesos, con el fin de comprender cómo es que funciona el CPU scheduler y El JOB scheduler se plantea como proyecto la creación de una simulación que ejemplifica el funcionamiento.

La simulación del CPU scheduler con el objetivo de comprender más acerca de los sistema operativos con principal objetivo de comprender el funcionamiento del CPU scheduler, se plantea como un proyecto que implementa un CPU scheduler utilizando un sistema cliente-servidor, en el cual el cliente está encargado de mandar procesos y el servidor sea el encargado de simular el CPU scheduler.

Para la elaboración del proyecto se realizará utilizando el lenguaje de programación C en el sistema operativo Ubuntu 16.04 LTS, esto en razón de la necesidad de explorar el funcionamiento y el cómo implementa C los hilos y los sockets, los cuales están sumamente relacionados con los sistemas operativos.

II-Estrategia de solución

La principal consideración que se toma a la hora de desarrollar el sistema es el plantear el cómo se realiza la comunicación entre el cliente y el servidor, considerando que en el planteamiento del problema se define que cada procesos será administrado por un hilo para poder mantenerse en espera hasta que el proceso sea recibido por el cliente y le mande el PID del proceso, por lo cual se plantea que cada proceso que tenga que ser enviado al servidor inicie una conexión independiente al servidor para poder realizar esto cada procesos se gestionará como un hilo.

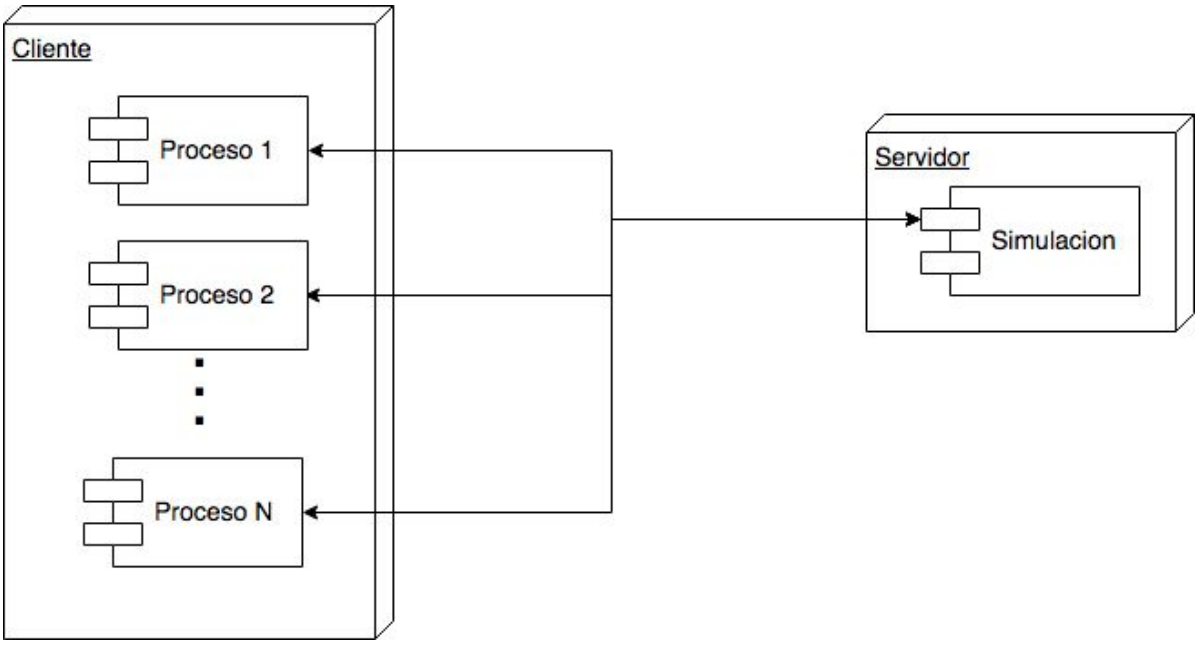


Imagen 2.1 Diagrama con fines ilustrativos

Tal como se muestra en la Imagen 2.1 el cliente podrá crear de 1 a N procesos para facilitar la gestión el conjunto de procesos se administra mediante una cola, para la creación de procesos el sistema cuenta con dos metodos de creacion de procesos

- El primer método es la creación de valores aleatorios en el cual se crearán dos datos el primero es un burst con un valor aleatorio entre 3 y 8, el segundo el la prioridad que será un valor entre 1 y 5.
- El segundo método es mediante un archivo de texto el cual debe tener la estructura

```
4 2
4 3
4 4
F
```

En el cual la primera columna indica el valor del burst y la segunda columna representa la prioridad del procesos, además se debe poner al final el carácter de escape F que indica la finalización del documento.

Los datos generados por cualquiera de estos métodos serán almacenados en una cola para ser gestionados por el cliente.

En la imagen 2.1 se muestra que el servidor contiene la simulación del CPU scheduler, la cual consta de 3 principales componentes que son el receptor de procesos, la colo de procesos y el componente encargado de la simulación del CPU scheduler.

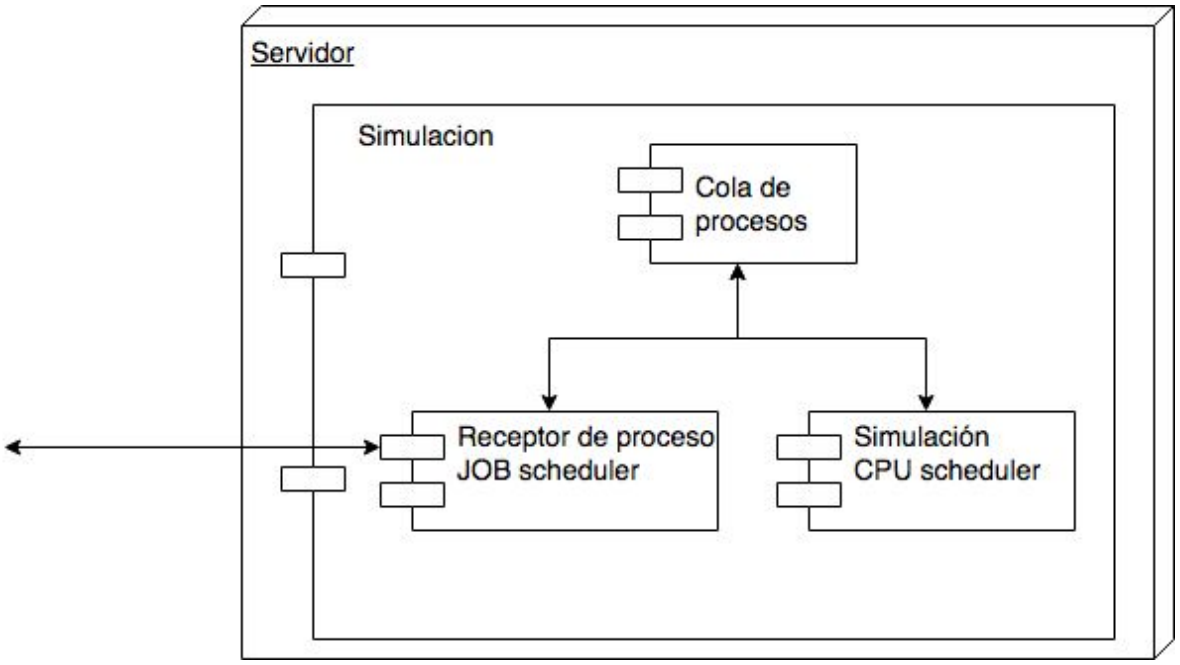


Imagen 2.2 Diagrama ilustrativo del servidor

En la imagen 2.2 se muestra la interacción que tiene los principales componentes del servidor.

- El primer componente es el receptor de procesos en el cual se reciben los procesos de parte del cliente luego de que son recibidos son almacenados en la cola y se les asigna un PID, el cual se le es comunicado al proceso emisor, para cada procesos se crea un nodo que cumple la función del PCB ya que contiene toda la información del proceso.
- La cola es la encargada de mantener el los procesos hasta el momento en que sean requeridos.
- La simulación se encarga de escoger un proceso de la cola utilizando el algoritmo (fifo,sjf,hpf o rr) que se seleccionó al iniciar el servidor.

Para ejecutar la simulación y la recepción al mismo tiempo se crea un hilo para cada uno con la cola como información compartida.

III-Análisis de resultados

Se mostrará una lista de objetos y se definirá tres categorías para determinar el estado del producto las categorías son:

- Completada
- Funcionamiento parcial
- Incompleto

En caso de determinar que la sección está incompleta se especificará porque está incompleta

- General
  - La comunicación por sockets: completada.
- Cliente
  - Cliente manual: completada.

    - Cliente automático: completada.
    - Recepción del PID: completada.
- Servidor
    - JOB scheduler
        - Asignar PID: completada.
        - Enviar mensaje al cliente: completada.
        - Le crea un PCB: completada.
    - CPU scheduler
        - Algoritmo FIFO : completada.
        - Algoritmo SJF: completada.
        - Algoritmo HPF: completada.
        - Algoritmo Round Robin: completada.
        - Mostrar proceso en ejecución en pantalla : completada.
        - Mostrar proceso finalizado en pantalla : completada.
        - Mostrar proceso en cola en pantalla : completada.
        - Información resumen : completada.

IV-Lecciones aprendidas

A continuación se enlistan un conjunto de problemas y soluciones, que se presentaron en el desarrollo del proyecto.

- Creación de la comunicación del cliente se encontró con la dificultad de crear varios mensajes sobre una conexión al servidor, por lo cual se cambio de perspectiva al crear distintas comunicaciones.
- Creación de hilos mediante pthread, surgió un problema a la hora de compilar ya que a pesar de que se incluye la librería no se detectaba, ante este problema se investigó [7] y se encontró que se debía enlazar la librería a la hora de compilar.
- Paso de argumentos a la hora de llamar el programa, para tratar de facilitar el uso de la aplicación se investigó [2] cómo pasar argumentos al main en C.

V-Casos de prueba

En lo casos de prueba se estructurará mediante la siguiente manera el número de prueba, el objetivo de la prueba, el dato de entrada información de salida y análisis de la salida.

- Prueba #1
    - Objetivo: verificar el tiempo de ocio se usará el algoritmo fifo en la parte del servidor.En la parte del cliente se usará la forma manual, además se establecerá un tiempo estático de 5 segundos a la hora de enviar los procesos.
    - Dato entrada

  Documento
  4 3
  4 3
  4 3

F

```
****************************************
*                CLIENTE                *
****************************************
PID: 1   Burst: 4        Prioridad: 3
PID: 2   Burst: 4        Prioridad: 3
PID: 3   Burst: 4        Prioridad: 3
```

- Dato de salida

```
****************************************************************************************************
*                                    SERVIDOR                                                      *
****************************************************************************************************
-----------------------------------  LLEGADA  ----------------------------------------------------
-----------------------------------  CPU      ----------------------------------------------------
-----------------------------------  SALIDA   ----------------------------------------------------
 PID: 1          B: 4    P: 3    E: 0    T_L: 0          T_S: 4          TAT: 4          WT: 0
 PID: 2          B: 4    P: 3    E: 0    T_L: 5          T_S: 9          TAT: 4          WT: 0
 PID: 3          B: 4    P: 3    E: 0    T_L: 10         T_S: 14         TAT: 4          WT: 0
-----------------------------------  INFOMACION  -------------------------------------------------
Promedio WT:0.00  Promedio TAT:3.00  Ocioso:2
```

- Análisis.

Se observa en la información de salida como en la parte inferior se contabilizan los dos segundos de ocio.

- Prueba #2
  - Objetivo: verificar el algoritmo FIFO en la parte del servidor.En la parte del cliente se usará la forma manual, además se establecerá un tiempo estático de 2 segundos a la hora de enviar los procesos.
  - Dato entrada

8 1
1 2
2 3
1 3
1 3
F

```
****************************************
*                CLIENTE                *
****************************************
PID: 1   Burst: 8        Prioridad: 1
PID: 2   Burst: 1        Prioridad: 2
PID: 3   Burst: 2        Prioridad: 3
PID: 4   Burst: 1        Prioridad: 3
PID: 5   Burst: 1        Prioridad: 3
```

- Dato de salida

```
****************************************************************************************************
*                                    SERVIDOR                                                      *
****************************************************************************************************
-----------------------------------  LLEGADA  ----------------------------------------------------
-----------------------------------  CPU      ----------------------------------------------------
-----------------------------------  SALIDA   ----------------------------------------------------
 PID: 1          B: 8    P: 1    E: 0    T_L: 0          T_S: 8          TAT: 8          WT: 0
 PID: 2          B: 1    P: 2    E: 0    T_L: 2          T_S: 9          TAT: 7          WT: 6
 PID: 3          B: 2    P: 3    E: 0    T_L: 4          T_S: 11         TAT: 7          WT: 5
 PID: 4          B: 1    P: 3    E: 0    T_L: 6          T_S: 12         TAT: 6          WT: 5
 PID: 5          B: 1    P: 3    E: 0    T_L: 8          T_S: 13         TAT: 5          WT: 4
-----------------------------------  INFOMACION  -------------------------------------------------
Promedio WT:4.00  Promedio TAT:5.50  Ocioso:2
```

- Análisis.

Se muestra en los datos de salida como los PID están organizados por lo cual

cumplen un correcto funcionamiento.

- Prueba #3
  - Objetivo: verificar el algoritmo SJF en la parte del servidor.En la parte del cliente se usará la forma manual, además se establecerá un tiempo estático de 2 segundos a la hora de enviar los procesos.
  - Dato entrada

8 1
3 2
2 3
1 3
1 3
5 3
F

```
*****************************************
*                 CLIENTE                *
*****************************************
PID: 1   Burst: 8        Prioridad: 1
PID: 2   Burst: 3        Prioridad: 2
PID: 3   Burst: 2        Prioridad: 3
PID: 4   Burst: 1        Prioridad: 3
PID: 5   Burst: 1        Prioridad: 3
PID: 6   Burst: 5        Prioridad: 3
```

  - Dato de salida

```
*************************************************************************************************
*                                        SERVIDOR                                               *
*************************************************************************************************
----------------------------------  LLEGADA  ------------------------------------------
----------------------------------  CPU      ------------------------------------------
----------------------------------  SALIDA   ------------------------------------------
 PID: 1          B: 8    P: 1    E: 0    T_L: 0          T_S: 8          TAT: 8          WT: 0
 PID: 4          B: 1    P: 3    E: 0    T_L: 6          T_S: 9          TAT: 3          WT: 2
 PID: 5          B: 1    P: 3    E: 0    T_L: 8          T_S: 10         TAT: 2          WT: 1
 PID: 3          B: 2    P: 3    E: 0    T_L: 4          T_S: 12         TAT: 8          WT: 6
 PID: 2          B: 3    P: 2    E: 0    T_L: 2          T_S: 15         TAT: 13         WT: 10
 PID: 6          B: 5    P: 3    E: 0    T_L: 10         T_S: 20         TAT: 10         WT: 5
----------------------------------  INFOMACION  ---------------------------------------
Promedio WT:4.00  Promedio TAT:6.29  Ocioso:3
```

  - Análisis.

Al considerar el orden de llegada de los proceso y el tiempo de burst que tienen se logra observar que los procesos están ordenadas de mayor a menor burst

- Prueba #4
  - Objetivo: verificar el algoritmo HPF en la parte del servidor.En la parte del cliente se usará la forma manual, además se establecerá un tiempo estático de 2 segundos a la hora de enviar los procesos.
  - Dato entrada

8 1
3 5
2 1
1 1
1 4
5 2
F

```
****************************************
*                CLIENTE               *
****************************************
PID: 1   Burst: 8        Prioridad: 1
PID: 2   Burst: 3        Prioridad: 5
PID: 3   Burst: 2        Prioridad: 1
PID: 4   Burst: 1        Prioridad: 1
PID: 5   Burst: 1        Prioridad: 4
PID: 6   Burst: 5        Prioridad: 2
```

- Dato de salida

```
****************************************************************************************************
*                                        SERVIDOR                                                  *
****************************************************************************************************
----------------------------------  LLEGADA  ----------------------------------------------
----------------------------------  CPU      ----------------------------------------------
----------------------------------  SALIDA   ----------------------------------------------
 PID: 1          B: 8    P: 1    E: 0    T_L: 0          T_S: 8          TAT: 8          WT: 0
 PID: 3          B: 2    P: 1    E: 0    T_L: 4          T_S: 10         TAT: 6          WT: 4
 PID: 4          B: 1    P: 1    E: 0    T_L: 6          T_S: 11         TAT: 5          WT: 4
 PID: 6          B: 5    P: 2    E: 0    T_L: 10         T_S: 16         TAT: 6          WT: 1
 PID: 5          B: 1    P: 4    E: 0    T_L: 8          T_S: 17         TAT: 9          WT: 8
 PID: 2          B: 3    P: 5    E: 0    T_L: 2          T_S: 20         TAT: 18         WT: 15
----------------------------------  INFOMACION  -------------------------------------------
Promedio WT:5.33  Promedio TAT:7.43  Ocioso:2
```

- Análisis.

Al analizar los datos de salida se puede determinar que los procesos están ordenados por la prioridad de ejecución que tienen.

- Prueba #5
  - Objetivo: verificar el algoritmo RR quantum 2 en la parte del servidor.En la parte del cliente se usará la forma manual, además se establecerá un tiempo estático de 2 segundos a la hora de enviar los procesos.
  - Dato entrada

8 3
4 3
1 2
7 4
4 1
F

```
****************************************
*                CLIENTE               *
****************************************
PID: 1   Burst: 8        Prioridad: 3
PID: 2   Burst: 4        Prioridad: 3
PID: 3   Burst: 1        Prioridad: 2
PID: 4   Burst: 7        Prioridad: 4
PID: 5   Burst: 4        Prioridad: 1
```

- Dato de salida

```
****************************************************************************************************
*                                        SERVIDOR                                                  *
****************************************************************************************************
----------------------------------  LLEGADA  ----------------------------------------------
----------------------------------  CPU      ----------------------------------------------
----------------------------------  SALIDA   ----------------------------------------------
 PID: 3          B: 1    P: 2    E: 0    T_L: 2          T_S: 5          TAT: 3          WT: 2
 PID: 2          B: 4    P: 3    E: 0    T_L: 1          T_S: 13         TAT: 12         WT: 8
 PID: 5          B: 4    P: 1    E: 0    T_L: 4          T_S: 17         TAT: 13         WT: 9
 PID: 1          B: 8    P: 3    E: 0    T_L: 0          T_S: 23         TAT: 23         WT: 15
 PID: 4          B: 7    P: 4    E: 0    T_L: 3          T_S: 24         TAT: 21         WT: 14
----------------------------------  INFOMACION  -------------------------------------------
Promedio WT:9.60  Promedio TAT:12.00  Ocioso:1346
```

- Análisis.

Se realizó la corrida del ejemplo dado en una práctica con el fin de verificar que funciona correctamente.

- Prueba #6
  - Objetivo: verificar el funcionamiento aleatorio del cliente, en el lado del servidor se usará FIFO para procesar la solicitudes.
  - Dato entrada

```
*****************************************
*                CLIENTE                *
*****************************************
PID: 1   Burst: 5        Prioridad: 2
PID: 2   Burst: 5        Prioridad: 2
PID: 3   Burst: 1        Prioridad: 5
PID: 4   Burst: 5        Prioridad: 1
PID: 5   Burst: 4        Prioridad: 3
PID: 6   Burst: 4        Prioridad: 5
PID: 7   Burst: 5        Prioridad: 2
PID: 8   Burst: 4        Prioridad: 1
PID: 0   Burst: 5        Prioridad: 5
```

  - Dato de salida

```
*************************************************************************************************
*                                        SERVIDOR                                               *
*************************************************************************************************
-----------------------------------  LLEGADA  ----------------------------------------------
-----------------------------------  CPU      ----------------------------------------------
-----------------------------------  SALIDA   ----------------------------------------------
 PID: 1          B: 5    P: 2    E: 0    T_L: 0          T_S: 5          TAT: 5          WT: 0
 PID: 2          B: 5    P: 2    E: 0    T_L: 3          T_S: 10         TAT: 7          WT: 2
 PID: 3          B: 1    P: 5    E: 0    T_L: 9          T_S: 11         TAT: 2          WT: 1
 PID: 4          B: 5    P: 1    E: 0    T_L: 16         T_S: 21         TAT: 5          WT: 0
 PID: 5          B: 4    P: 3    E: 0    T_L: 22         T_S: 26         TAT: 4          WT: 0
 PID: 6          B: 4    P: 5    E: 0    T_L: 26         T_S: 30         TAT: 4          WT: 0
 PID: 7          B: 5    P: 2    E: 0    T_L: 33         T_S: 38         TAT: 5          WT: 0
 PID: 8          B: 4    P: 1    E: 0    T_L: 39         T_S: 43         TAT: 4          WT: 0
-----------------------------------  INFOMACION  -------------------------------------------
Promedio WT:0.38  Promedio TAT:4.00  Ocioso:13
```

  - Análisis.

En la salida se muestra el conjunto aleatorio de datos creados.

- Prueba #7
  - Objetivo: verificar como el servidor muestra los procesos entrantes los que están en el CPU y lo que terminan, además del promedio TAT, promedio WT y el tiempo de ocio. En la parte del cliente se usará la forma manual.
  - Dato entrada

```
4 3
5 6
10 4
3 5
2 1
1 1
1 4
5 2
F
```

```
****************************************
*                CLIENTE                *
****************************************
PID: 1   Burst: 4        Prioridad: 3
PID: 2   Burst: 5        Prioridad: 6
PID: 3   Burst: 10       Prioridad: 4
PID: 4   Burst: 3        Prioridad: 5
PID: 5   Burst: 2        Prioridad: 1
PID: 6   Burst: 1        Prioridad: 1
PID: 7   Burst: 1        Prioridad: 4
PID: 8   Burst: 5        Prioridad: 2
```

- Dato de salida

```
*********************************************************************************************
*                                    SERVIDOR                                               *
*********************************************************************************************
----------------------------------  LLEGADA  ----------------------------------------------
 PID: 4          B: 3    P: 5    E: 3    T_L: 15         T_S: 0          TAT: 0          WT: 0
----------------------------------  CPU      ----------------------------------------------
 PID: 3          B: 10   P: 4    E: 4    T_L: 10         T_S: 0          TAT: 0          WT: 0
----------------------------------  SALIDA   ----------------------------------------------
 PID: 1          B: 4    P: 3    E: 0    T_L: 0          T_S: 4          TAT: 4          WT: 0
 PID: 2          B: 5    P: 6    E: 0    T_L: 6          T_S: 11         TAT: 5          WT: 0
----------------------------------  INFOMACION   ------------------------------------------
Promedio WT:0.00  Promedio TAT:3.00  Ocioso:2
```

- Análisis.

Se muestran las tres bandejas del servidor, además de la información requerida en la parte inferior.
La primera es la bandeja de LLEGADA en ella se muestran los procesos entrantes.
La segunda es la bandeja de CPU en ella se muestra el procesos que se ejecuta actual
La tercera es la bandeja de SALIDA en ella se muestra los procesos finalizados además de mostrar WT y TAT.
La última parte muestra la información del tiempo de ocio y los promedios.

VI-Comparación

Al comparar la implementación de los hilos en java y los hilos en C, se logra comprender que la dinámica de uso es similar en los dos ya que ambos cuentan con una función que es la que contiene el código que se ejecutará, en el caso de java esta es la función run en el caso de C en la función de tipo void * que se pasa por parámetro, esto hace que su implementación se similar.
Sin embargo a la hora del uso y de la implementación, el método propuesto por java es más fácil de comprender y aplicar, a diferencia del método de C el cual requiere más pasos para ser implementado.
En la parte del funcionamiento los hilos de C son más óptimos que los hilos de Java [5], ya que los hilos de c proven más manejabilidad, al funcionar directamente con la máquina a diferencia de java que usa JVM.

VII-Manual de usuario

Compilación:
El proyecto fue elaborado usando el entorno de desarrollo:

Eclipse IDE for C/C++ Developers
Version: Oxygen.3 Release (4.7.3RC3)
Build id: 20180301-1623

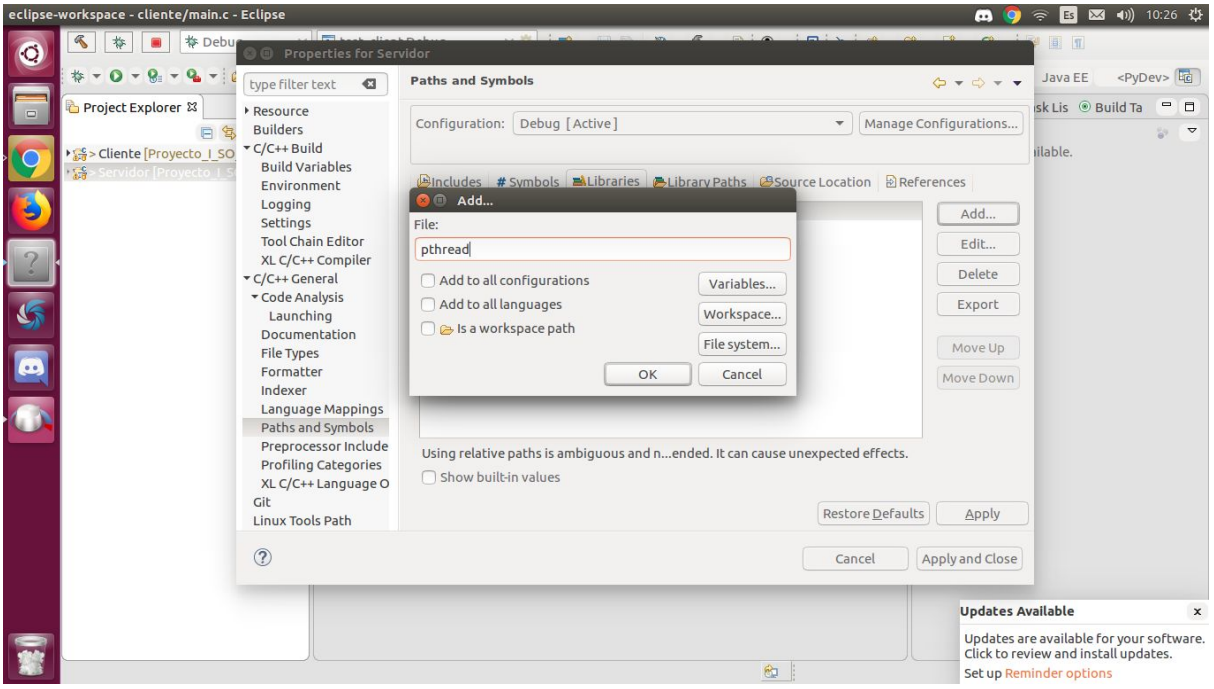
El IDE Eclipse [3], probe la opcion de compilación automática (Build) la cual fue utilizada para la compilación del proyecto.
La estructura del proyecto generada por el IDE eclipse es la siguiente
Cliente

```
Cliente
        ->Debug
        ->src
```

Servidor

```
Servidor
        ->Debug
        ->src
```

En la cual se crearía en la parte de Debug el archivo ejecutable luego del procesos de compilación, la carpeta src será la que contenga el código a compilar.
Para la compilación es necesario vincular la librería pthread al proyecto.
Se debe dar click derecho sobre el proyecto ir a la opción
properties -> C/C++ General -> Paths and Symbols -> Libraries -> Add



Se debe ingresar pthread y pulsar el botón ok.
Luego se da ctrl+b y se compila el programa.
Ejecución:

Para la ejecución se usará la terminal de ubuntu, se buscará carpeta Debug donde se encontrará el objeto ejecutable.

En el caso del cliente se puede ingresar de dos maneras.

- Manual mediante el comando:

`./Cliente /archivo.txt`

- Automático mediante el comando:

`./Cliente`

En el caso del servidor se puede iniciar de las siguientes maneras.

- FIFO

`./Servidor 1`

- SJF

`./Servidor 2`

- HPF

`./Servidor 3`

- RR

`./Servidor 4 2`

VIII-Bitácora

Tomando en cuenta el planteamiento del proyecto se determinó que el orden adecuado en la investigación y elaboración, será el siguiente.

- (25-3-2018) Investigación y puesta en práctica de sockets en C: se elaboraron ejemplos[1][4] de creación de hilos con la finalidad de comprender su funcionamiento.
- (26 y 27 3-2018) Creación del sistema cliente-servidor: se consultó la documentación [8] de sockets en C, con lo cual se pudo elaborar la primera interacción del proyecto.
- (28-3-2018) Investigación y puesta en práctica de hilos en C con la librería pthread: luego de obtener la comunicación cliente-servidor, se creo un hilo debido al conocimiento previo de el uso de los hilos no fue complicada la elaboración solo se necesito un elborar un ejemplo[6] de uso.
- (31-3-2018) Creación de la cola: la elaboración de la cola en C no presenta mayor complicación, ya que se tenía conocimientos previos.
- (31-3-2018) Algoritmos FIFO, SJF, HPF: con la cola se realizaron algoritmos de búsqueda sobre la cola para representar los métodos de selección de procesos.
- (2-4-2018) Creación de procesos aleatorios: se usó el método Rand() propio de C.
- (2-4-2018) Leer un archivo: se investigó [11] como se lee un archivo en C y se implementó.
- (7-4-2018) Método de despliegue en pantalla: se planteó un método en el cual se pueda borrar el contenido de la terminal, en razón de esto se investigó [10] y se encontró el código.

  ```
  printf("\033[H\033[J");
  ```

  El cual permite borrar el contenido de la consola de ubuntu.
- (8-4-2018) Optimización y solución de errores.

Bibliografía

[1]Code, C. (2018). *C Socket Programming for Linux with a Server and Client Example Code*. [online] Thegeekstuff.com. Disponible en: https://www.thegeekstuff.com/2011/12/c-socket-programming/?utm_source=feedburner [Acceso 11 Abr. 2018].

[2]Decsai.ugr.es. (2018). *Argumentos de main()*. [online] Disponible en: http://decsai.ugr.es/~jfv/ed1/c/cdrom/cap6/cap64.htm [Acceso 11 Abr. 2018].

[3]Eclipse.org. (2018). *Eclipse IDE for C/C++ Developers* | *Packages*. [online] Disponible en: https://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/oxygen3 [Acceso 11 Abr. 2018].

[4] Gist. (2018). *TCP echo client-server in C*. [online] Disponible en: https://gist.github.com/suyash/2488ff6996c98a8ee3a84fe3198a6f85 [Acceso 11 Abr. 2018].

[5]https://www.quora.com/. (2018). *What-is-the-difference-between-Java-thread-and-C-C++-thread*. [online] Disponible en: https://www.quora.com/What-is-the-difference-between-Java-thread-and-C-C++-thread [Acceso 11 Abr. 2018].

[6]Ippolito, G. (2018). *Linux Tutorial: POSIX Threads*. [online] Cs.cmu.edu. Disponible en: https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html [Acceso 11 Abr. 2018].

[7]pthread.h?, h. (2018). *how to compile a c program that uses pthread.h?*. [online] Askubuntu.com. Disponible en: https://askubuntu.com/questions/420722/how-to-compile-a-c-program-that-uses-pthread-h [Acceso 11 Abr. 2018].

[8]Pubs.opengroup.org. (2018). *<sys/socket.h>*. [online] Disponible en: http://pubs.opengroup.org/onlinepubs/7908799/xns/syssocket.h.html [Acceso 11 Abr. 2018].

[9]Silberschatz, A., Galvin, P. and Gagne, G. (n.d.). *Operating system concepts*.

[10]system?, C. (2018). *Clear screen in C and C++ on UNIX-based system?*. [online] Stackoverflow.com. Disponible en: https://stackoverflow.com/questions/17271576/clear-screen-in-c-and-c-on-unix-based-system/17271636 [Acceso 11 Abr. 2018].

[11]www.tutorialspoint.com. (2018). *C File I/O*. [online] Disponible en: https://www.tutorialspoint.com/cprogramming/c_file_io.htm [Acceso 11 Abr. 2018].