Objetivos del proyecto

- Aprender a desarrollar un esquema de cliente-servidor.
- Conocer la comunicación por medio de sockets.
- Desarrollar un programa donde se utilicen hilos.
- Poner en práctica la teoría del Planificador de CPU

El caso: Simulación de atención de procesos de un SO.

Este primer proyecto tiene como finalidad desarrollar un escenario de generación de procesos que puedan ser tramitados y administrados por el CPU por medio de los siguientes algoritmos:

- First Comes First Served - FCFS
- Shortest Job First - SJF
- Highest Priority First – HPF
- Round Robin – RR (con un quantum especificado por el usuario)

Se requiere que aplicación sea bajo la modalidad cliente-servidor, en la que un cliente enviará la información de los procesos que desea ejecutar en un servidor determinado.
Para esto, el grupo de trabajo deberá explorar sobre el uso de hilos y sockets de modo que pueda implementar la aplicación para que un servidor reciba los datos de los procesos que deberá atender.

La aplicación cliente tendrá dos modalidades de funcionamiento:

Manual y Automática.

En ambas modalidades el usuario debe establecer previamente el rango en el que se permitirán los valores de "*burst*" o tiempo de utilización del CPU. Ejemplo entre 1 y 20 unidades de tiempo.

- **Modo Manual**: En esta modalidad el cliente indica la existencia de un archivo con la información de los procesos que debe crear y enviar al servidor para su atención por medio de un "thread".

  Como parte de la información que debe suministrar dicho archivo para cada proceso es el "burst" o carga de trabajo que requiere del CPU, su prioridad (considere un valor entre 1: Min_priority y 10: Max_Priority donde el punto medio por default es 5).

  Debe haber un tiempo de sleep entre la lectura en el archivo de cada uno de los procesos. Esa duración será un tiempo random (3 a 8 segundos).

La información que tendrá el archivo de entrada es la siguiente: (ejemplo)

BURST	Prioridad
8	3
7	2

- **Modo Automático:** El cliente creará un conjunto de procesos con sus características ("burst", prioridad) al azar. La prioridad es aleatoria entre 1 y 10, el "burst" debe estar dentro del rango establecido previamente por el usuario). El rango de los valores para el random del Burst y de la tasa de creación será un parámetro de entrada al correr esta versión del cliente. El cliente se mantiene creando procesos hasta que el usuario detenga el proceso.

En ambas modalidades, una vez enviada la información de los procesos al servidor, los *threads* generados en el cliente, quedan esperan la notificación de recibido por parte del servidor, quien contesta con el número de ID asignado al proceso recibido, éste se debe desplegar en el cliente y luego el *thread* muere (se desconecta).

La aplicación servidora

En la aplicación servidora se contendrá el simulador del sistema operativo. Cuando el servidor levanta, debe indicarse el tipo de algoritmo de planificación que utilizará en el CPU Scheduler.

Se espera que el Planificador de CPU tenga al menos 2 *threads*: uno que reciba los mensajes del socket y ponga los procesos en la **"cola"** (*JOB Scheduler*) y otro que seleccione cual proceso tendrá el CPU (*CPU Scheduler*) y simule al CPU ejecutando.

El JOB Scheduler:

Será un thread que reciba los mensajes de los sockets y los vaya guardando en algún tipo de lista. Esta lista simulará la cola del estado **Ready**. Cuando este hilo recibe el proceso:

- Asigna un PID
- Envía un mensaje de confirmación al cliente con el valor del PID.
- "Crea" su PCB y lo guarda en la lista de espera.

El CPU Scheduler:

Constantemente deberá verificar si hay procesos en la cola. Si los hay debe ponerlos a ejecutar. Esto significa que según el algoritmo que se haya seleccionado, deberá escoger a alguien de la cola, "ejecutarlo" (aplicando su "*burst"* por medio de un sleep) y seleccionar el próximo proceso, según el criterio del algoritmo.

- El CPU Scheduler empezará la "ejecución" de los procesos tan pronto tenga un proceso en la cola de espera. La ejecución de un proceso debe ser de la duración real del "*burst*". Debe quedar claro que mientras se ejecutan procesos, otros irán llegando a la cola.

- Cada vez que hay un *context switch* y un proceso se ejecutará, se debe desplegar en pantalla (ejemplo: Proceso con PID 1: Burst x - Prioridad Y entra en ejecución)
- Cada vez que un proceso termina completamente su ejecución y deja de estar en espera debe desplegarlo en pantalla.
- En cualquier momento de la ejecución el usuario del servidor puede consultar la cola. Sólo debe desplegar los procesos que están en espera.
- **Se recomienda que tenga buen control del estado de los procesos.**
- Para el Round Robin, debe tomar en cuenta siempre lo que lleve ejecutado.
- Para el FIFO el criterio de selección será el PID.
- No se implementará ningún algoritmo apropiativo (salvo el RR)
- El Planificador de CPU puede dejar de funcionar en el momento que el usuario decida dar de baja al servidor. En este momento, el servidor deberá volcar el log que ha venido registrando a modo de resumen:
    - Cantidad de procesos ejecutados
    - Cantidad de segundos con CPU ocioso.
    - Tablas de TAT (*Turn Around Time*) y WT (W*aiting Time*) para los procesos ejecutados.
    - Promedio de *Waiting Time*

Generalidades:

- Se debe tener cuidado, de no dejar procesos sin matar o sockets abiertos.
- Puede ser que necesiten otros threads, para llevar tiempos, o para obtener comandos (despliegue cola / detener) la simulación o para lo que ocupen. Lo obligatorio es la existencia de los threads de los Schedulers.
- Diseñe bien la información que debe contener de cada proceso. Piense que tendrá un tipo de PCB para que el CPU Scheduler sepa a quien seleccionar.

- Dependiendo de cómo guarde la información de los procesos, pudiera necesitar un semáforo.

Entregables

1. Proyecto fuente y sus archivos de pruebas.
2. Documento formal donde especifique lo siguiente:
    a. Portada, índice, introducción
    b. Estrategia de Solución
    c. Análisis de Resultados: Deberá elaborar un listado de todas y cada una de las actividades y tareas que deben cubrirse a nivel funcional, para cada una de ellas debe aportar el porcentaje de realización y en caso de no ser el 100% debe justificarse.
    d. Lecciones aprendidas: Debe prepararse un listado de las lecciones aprendidas producto del desarrollo de la tarea programada. Las lecciones aprendidas pueden ser de carácter personal y/o técnico que involucre aspectos que han logrado un aprendizaje en temas de investigación, desarrollo de habilidades técnicas y habilidades blandas como trabajo en equipo, comunicación, forma de expresar ideas, entre otros.
    e. Casos de pruebas: se espera que definan claramente cada prueba, cuáles son los resultados esperados y cuáles fueron los resultados obtenidos. No es necesario que sean grandes, pero deben evaluar la funcionalidad completa del programa.
    f. Comparación: Investigación comparativa entre los hilos de Java y PThreads. La comparación debe incluir usabilidad, detalles técnicos y rendimiento.
    g. Manual de usuario: especificar como compilar y correr su tarea.
    h. Bitácora de trabajo durante las tres semanas de trabajo, incluyendo verificaciones realizadas (si existieran) de consultas realizadas con el profesor o asistente.

i. Bibliografía y fuentes digitales utilizadas

Aspectos Administrativos

- El desarrollo de este programa debe de realizarse en grupos de exactamente tres personas.
- El trabajo se debe de entregar antes del 10 de abril de 2018 a media noche, en la parte de Evaluaciones Primer Proyecto del tecDigital.
- Deben entregar el código fuente junto con el ejecutable.
- El proyecto debe ser desarrollado en C para Linux, utilizando la librería PThreads.