Proyecto 1: Banda de paquetes

Gutiérrez Sánchez Roberto, Salas Alfaro Jimena, Vargas Gamboa Josafat
roberm98a@gmail.com jimena.salas.alfaro@gmail.com josav09@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—En este proyecto se simula una banda transportadora de paquetes a nivel de software y hardware utilizando diferentes calendarizadores que se basan en los de sistemas operativos, tales como Round Robin, por prioridad, el más corto primero, *FIFO*, y tiempo real. Cada una de las bandas tiene una lista de paquetes a cada lado, con diferentes parámetros. Para la simulación de las bandas se realiza una implementación propia de la biblioteca pthread de Linux en C.**

***Palabras clave*—Sistemas operativos, calendarizadores, Round Robin, RTOS, FIFO, pthread, Linux**

I. Introducción

Una de las funciones más importantes que realiza el sistema operativo es la calendarización de diferentes procesos. Distintos métodos de calendarización se han desarrollado y propuesto a través de los años [2]. Para este proyecto se simulan métodos de calendarización como una banda transportadora que simula el procesador de una computadora, de esta manera este es el único objeto disponible para procesar y se tiene que compartir entre distintos paquetes para que todos logren llegar a su destino.

Los métodos de calendarización a simular son los de Round Robin, por prioridad, el más corto primero, *FIFO*, y tiempo real [3], además de esto tanto las bandas como los paquetes tienen diferentes configuraciones como el lado de la banda a escoger, la fuerza de la banda, la distribución de probabilidad para la generación de paquetes, los cuales pueden tener un nivel de prioridad dependiendo de su tipo ya sea radioactivo, urgente y normal. Las diferentes configuraciones afectan los calendarizadores dependiendo de su tipo y cada una de estas configuraciones se ve reflejada directamente en las bandas.

Para la implementación del sistema, se hace uso de los hilos de Linux con clone [4] [5], en donde similar a como proceso es la ejecución de un programa, los hilos permiten a un proceso realizar tareas del programa de forma paralela para el usuario. Internamente este proceso conlleva cambiar entre los hilos a una velocidad que impida al usuario distinguir entre las tareas que se están ejecutando. A este proceso de hacer el cambio entre cada uno de los hilos se le conoce como cambio de contexto [6], que ocurre en la solución de las bandas al cambiar un paquete por otro.

En este documento se muestran resultados de la implementación de la simulación de la banda de paquetes como calendarizadores de sistemas operativos, en la sección II se muestran los detalles del ambiente de desarrollo del sistema. En la sección III se muestran los detalles del diseño del sistema, en la sección IV se muestran las instrucciones de uso para realizar la simulación, en la sección V se muestra la bitácora de trabajo, en la sección VI se muestran las conclusiones referentes al desarrollo del sistema, y en la VII las sugerencias y recomendaciones.

II. Ambiente de desarrollo

El programa se desarrolla en C ya que este lenguaje le da al usuario un acceso de bajo nivel que permite acceder a los comandos utilizados por el sistema operativo para el manejo de los hilos.

El programa se comunica con la interfaz gráfica por medio de archivos. La interfaz, desarrollada en Python, lee estos archivos y despliega esta información en la interfaz. En la interfaz se aprovecha el parseo de la información que se está realizando para generar la información que se le envía al hardware por medio de comunicación serial.

Python ofrece muchas herramientas para desarrollar una interfaz gráfica, pero lo que lo separa de otros programas es la facilidad que ofrece para manejar archivos y comunicarse por medio de protocoles seriales con el arduino. La gran cantidad de documentación y la familiaridad del equipo con el lenguaje lo hace una elección sencilla. En este caso se hace uso de Python junto con las bibliotecas de PySerial y Tkinter.

II-A. Hardware

Se utiliza la tarjeta de desarrollo Arduino UNO para realizar la implementación del hardware, esta elección se basa en que es la que el equipo más ha utilizado y es la que se tiene disponible, además de que los pines necesarios para realizar la implementación son exactamente los que tiene la tarjeta.

III. Detalles de diseño

III-A. Biblioteca de hilos

En la implementación de la biblioteca de hilos, se comparan las funciones que provee la biblioteca pthread en C, de tal forma que se construya una equivalencia entre ambas.

Para ello se implementan las siguientes funciones:

- **Lthread_create**, la cual crea un nuevo thread y lo inicia automáticamente.
- **Lthread_exit**, la cual finaliza un thread sin importar su estado en ese momento.
- **Lthread_yield**, que pone el hilo ejecutándose de último en la cola de hilos.
- **Lthread_join**, la cual deja el contexto esperando a que se termine de ejecutar un thread.

- **Lthread_detach**, la cual no permite que un thread se le pueda hacer join.
- **Lmutex_init**, que inicializa una variable del tipo lpthread_mutex.
- **Lmutex_destroy**, que destruye una variable del tipo lpthread_mutex.
- **Lmutex_unlock**, que desbloquea la variable del mutex.
- **Lmutex_trylock**, que intenta bloquear la variable del mutex, si no está bloqueada, solamente sigue.
- **Lmutex_lock**, bloquea la variable del mutex, si no está libre se espera a obtenerlo.

Para realizar la implementación de la biblioteca, se hace uso los headers de linux, como los siguientes:

- *sched.h*: Para llamar al calendarizador de Linux, y que se logre colocar el hilo ejecutándose de último en la lista de hilos a ejecutar y que seleccione otro de manera forzada. Esto para la función yield thread.
- *signal.h*: Para hacer uso de señales entre procesos.
- *sys/types.h*: Para tener el tipo del id del proceso pid_t.
- *sys/wait.h*: Para poder esperar por algún proceso.
- *unistd.h*: Para las funciones de obtener ids de procesos.
- *sys/syscall.h*: Realizar llamadas al sistema.

Para la implementación de la biblioteca de hilos se construye un arreglo con la máxima cantidad de hilos posibles a ejecutar en un momento dado, cada uno de los elementos de este arreglo contiene la estructura del hilo, de esta manera cuando se crea un nuevo hilo se le asigna uno de ellos. Esta estructura de hilos, tiene los parámetros de puntero al stack, id del proceso correspondiente al hilo.

A continuación se detallan las funciones más importantes de la biblioteca liblpthread

- Lthread_create: Para poder implementar esta función se simula el uso como en pthread, en donde se pasan los punteros de la función y un puntero único de los parámetros para la misma como void. Se hace uso de la función clone de Linux, la cual genera un proceso más ligero que puede ser tratado como un hilo. Esta llamada a la función se le comparten diferentes flags para compartir los archivos de quienes lo llaman y además de diferentes señales para que reciba el hilo desde el padre o que envíe hacia él. Para poder hacer uso de la función clone se tiene que alocar el espacio de memoria para el stack de la función y pasar el puntero del mismo.
- Lthread_exit: Para cumplir con la implementación de la función exit, solamente se le envía la señal de SIGKILL al hilo utilizando su id de proceso correspondiente.
- Lthread_join: Para el join, se utiliza la función waitpid de Linux, la cual lo que hace es esperar a que un proceso termine, en este caso se pone a esperar a quien sea haya llamado la función, debido a que se debe de pasar la estructura de threads y se hace uso del id del proceso al que se llama.
- Lmutex_init: Esta función inicializa una instancia de la estructura mutex, para que contenga información acerca de si está bloqueado o no inicialemnte.
- Lmutex_lock: Para hacer el lock, se hace un spin lock en la variable del mutex que indica si este está o no bloqueado, es decir se hace un ciclo infinito hasta que se desbloquea el mismo, y aquí se trata de obtener.
- Lmutex_unlock: Simplemente se desactiva ella bandera de bloqueado en el atributo del mutex, de esta manera ya los threads haciendo spin lock podrán obtener el candado.

Uno de los principales problemas a la hora de realizar la implementación fue obtener una simulación de la teminación de los hilos cuando el proceso padre finalizara. Para poder realizar esto, se hace uso de las señales de linux y de la función atexit. De esta forma cuando se finalice el proceso desde consola, se obtendrá la señal de SIGINT y se terminarán todos los threads, y cuando finalice el proceso en sí, se llama la función atexit.

III-B. Manejo de datos

Para el proyecto fue necesario desarrollar estructuras de datos para listas enlazadas, con el fin de hacer que la alocación de memoria sea dinámica, además de que el lenguaje no provee esta estructura con las características deseadas. Otra complicación de C es que la estructura de paquetes y bandas era fácil de abstraer por medio de orientación a objetos, lo cual no contiene este lenguaje, pero gracias al tipo de dato compuesto que ofrece en el `struct` no resultó un inconveniente mayor. La estructura de datos implementada se compone de un valor y un puntero al siguiente elemento de la lista tal como se muestra en la Figura 1. Además, posee métodos para añadir, remover, obtener y configurar elementos específicos.

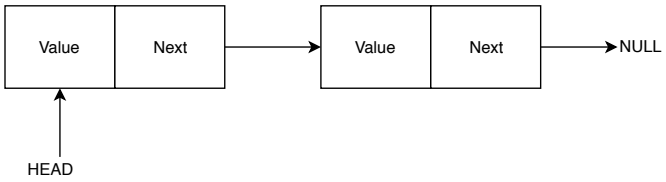

Figura 1: Estructura de la lista enlazada.

III-C. Calendarizadores

Para esta sección se realizaron cinco tipos de calendarizadores los cuales se detallan a continuación:

- **Round Robin**: El algoritmo consiste en revisar si el tiempo utilizado por el paquete actual ya alcanzó el límite del *quantum*, y si esto se cumple entonces guarda el progreso que llevaba este y pone a contar el tiempo del siguiente paquete. Para asegurar esto, se monitorea el tiempo utilizado por cada paquete mediante la función *clock()*, la cual si se resta con un tiempo establecido inicialmente indica el tiempo utilizado en segundos. Esto llevó a una dificultad pues para evitar tantas llamadas al calendarizador se definió un tiempo de $0{,}1s$ entre cada llamada, pero para contar ese tiempo no se puede dormir el thread porque sino los ticks de *clock()* también dejan de pasar. Para solucionar esto, se implementó una función contador que simplemente retorna hasta que se cumplen cierta cantidad de ticks, y así no hay necesidad de dormir el thread.

- **Por prioridad**: Este algoritmo consiste en ordenar la lista de paquetes de forma ascendente según su prioridad (0 = radiactivo, 1 = urgente, 2 = normal), lo cual se realiza mediante un ordenamiento de tipo *quicksort*. Como este algoritmo es de carácter no apropiativo, se llama al calendarizador únicamente cuando un paquete termina de pasar por la banda.
- **El más corto primero**: Este algoritmo de forma similar al anterior, únicamente consiste en ordenar de forma ascendente los paquetes según el tiempo total que duran en cruzar la banda, el cual viene definido por condiciones físicas. También se realiza mediante un ordenamiento *quicksort* y se llama cada vez que termina de cruzar un paquete por su carácter no apropiativo.
- **FIFO**: El algoritmo indica que el primero que entra es el primero que debe salir y que además también es de carácter no apropiativo. Debido a que en la generación de paquetes se añaden estos al final de la lista, su comportamiento es como si fuera una cola, por lo que para este tipo de calendarizador no es necesario agregar ninguna lógica en especial.
- **Tiempo real**: Para este algoritmo se debe asegurar que los paquetes radiactivos duren cruzando la banda un tiempo menor a un límite, el 100 % de las veces. En el caso de los paquetes urgentes puede pasarse de este tiempo unas cuantas veces, y en el caso de los normales no hay un límite establecido. Como el algoritmo también es de carácter apropiativo se debe estar recalendarizando cada vez que entren nuevos paquetes, para que en caso de que entre uno con mayor prioridad y que se ejecute en menos tiempo, se haga el cambio inmediatamente. Para la implementación de este algoritmo es necesario primero ordenar de acuerdo a la prioridad de los paquetes, y posteriormente ordenar los paquetes dentro de la misma prioridad de acuerdo al tiempo restante para finalizar, por lo que al final siempre se estará ejecutando aquel paquete con mayor prioridad y menor tiempo restante para cumplir con el algoritmo de *earliest deadline first*. Además, al igual que en Round Robin, es necesario guardar el progreso de aquellos paquetes que son cambiados en media ejecución para que cuando vuelvan a ejecutarse empiecen desde dónde quedaron la última vez.

III-D. Algoritmos de control de la banda

Para elegir cual es el lado de la banda en el cual se debe de mover un paquete, con el fin de simular los paquetes que se envían o los que se devuelven, se implementan los siguientes algoritmos de control de flujo de la banda.

- **Equidad de paquetes**: Se envían paquetes solamente de un lado de la banda hasta que se haya llegado a un límite de los mismos, para esto se define el parámetro W. En este caso solamente se le pone un contador para que de esta manera se sepa cuando se llega a un limite.
- **Letrero por tiempo**, en el cual solamente se tiene un hilo corriendo en el cual cada cierto tiempo se cambia la orientación del letrero.
- **Aleatorio**, en el cual cada vez que se tiene que escoger por un lado de la banda, se evalúa un aleatorio entre 0 y 100.

En la implementación de software, se crea una estructura de controlador para la banda, que tiene los punteros a cada una de las listas, la cantidad que se han movido, el tipo de controlador, el signo y el lado hacia donde se ha estado moviendo.

III-E. Generación de paquetes

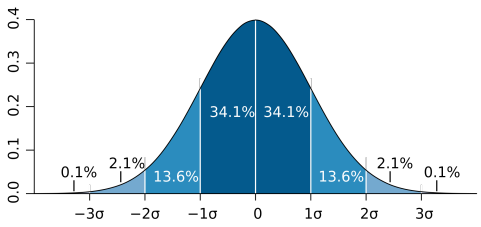

Figura 2: Rangos de una distribución normal

Para la generación de paquetes se utilizan los rangos de la distribución estándar, una aproximación de los percentiles de la distribución gamma y una distribución constante basada en el aleatorio generado por C. Estos valores se obtienen a partir de casillas y un valor al azar según la desviación estándar definida en el archivo de configuración. Se asegura que la distribución se cumpla generando un número entre 0 y 100 y comparándolo contra rangos que siguen la probabilidad de que el valor esté en el rango.

El peso se calcula también con un aleatorio en un rango predefinido y el tiempo que va a tardar el paquete se calcula simplificando la solución de la segunda derivada para obtener de $F = ma$ y $V = \frac{d}{t} = \frac{dx}{dt}$:

$$\begin{aligned} \tfrac{F}{m} &= \tfrac{d^2x}{dt^2} \\ dt^2 &= \tfrac{m}{F} d^2x \\ tdt &= \tfrac{m}{F} dx \\ t &= \tfrac{m}{F} * \tfrac{dx}{dt} \\ t^2 &= \tfrac{m*x}{F} \\ t &= \sqrt{\tfrac{m*x}{F}} \end{aligned}$$

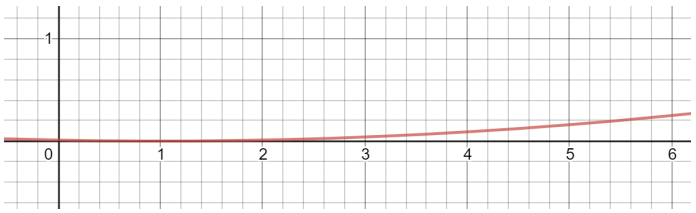

Figura 3: Cálculo del peso basado en el quantum

Al despejar está ecuación se manipula el peso que van a tener los paquetes radioactivos para el caso de tiempo real, para asegurar de que se cumplan los deadlines. Se agrega un tiempo de laxitud y se recomienda un quantum (deadlines) no menor a 3, este valor nos permite tener para una fuerza

mínima de 10 una distancia de banda de hasta 1000 unidades, siguiendo la abstracción para el programa. Para hacer los pesos aleatorios se aumenta la longitud de la banda aleatoriamente a la hora de hacer el cálculo. El quanto, naturalmente, debe ser mayor que la laxitud. Estos cálculos se basan en la figura 3. El despeje de la ecuación se realiza de la siguiente manera:

$$
\begin{aligned}
t &=< \sqrt{\frac{m*x}{F}} \\
t < (q - lax) &= \sqrt{\frac{m*x}{F}} \\
(q - lax)^2 &= \frac{m*x}{F} \\
m &= \frac{F*(q-lax)^2}{x}
\end{aligned}
$$

III-F. Diagramas de diseño

III-F1. Software: En el siguiente link se encuentra el diagrama UML del sistema con el fin de que pueda ser visualizado correctamente. En este se resumen las principales funciones implementadas para la solución del problema.

III-F2. Hardware: Inicialmente se definen las entradas y salidas del circuito como el estado de la interfaz gráfica y el estado de los leds. El circuito se puede comunicar de vuelta con el computador, a través de botones. El principal problema de la solución implementada es la cantidad de estados que hay que mostrar en el circuito en relación a la cantidad de pines disponibles en el microcontrolador. En general, la solución consta de 6 bloques principales. La PC es la encargada de correr el programa en sí y la interfaz, de donde se toman los valores que queremos ver en el hardware, mientras que el microcontrolador maneja la comunicación entre estos. Se utiliza una etapa con registros de corrimiento para reducir la cantidad de pines necesaria para controlar los leds.

El circuito consiste de 3 bandas, según la especificación, pero si se utiliza un microcontrolador con más puertos se podría aumentar la cantidad de bandas. En la figura 4 se muestra a un tercer nivel el diagrama de la solución por hardware, mientras que en la figura 5 se muestra las conexiones eléctricas para una banda.

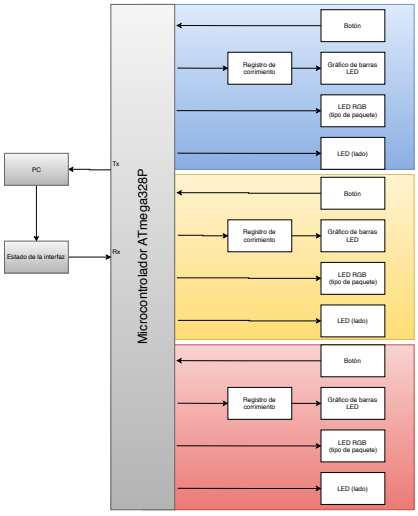

Figura 4: Diagrama de tercer nivel

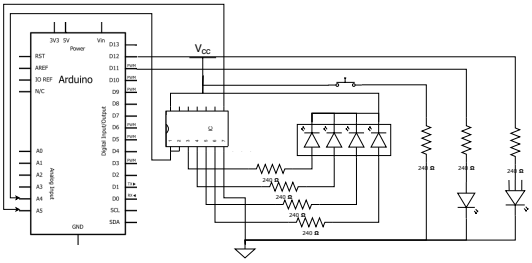

Figura 5: Diagrama de quinto nivel: conexiones eléctricas

IV. Instrucciones de Uso

Conecte el circuito por medio de USB y obtenga el puerto al que está conectado. Ejecute `make clean` y luego `make` en Schedulers, posteriormente intercambie en el archivo gui/gui.py el puerto donde esté conectado el arduino.

Para modificar los valores de las bandas entre en schedulers/config y modifique los valores numéricos.

- **bandID** El identificador de la banda, 0 es la superior, 1 la del medio y 2 la inferior. No modifique este valor.
- **bandStrength** La fuerza de la banda, se recomienda que esté entre 10 y 150.
- **bandLength** La longitud de la banda. Se recomienda un valor entre 1 y 100.
- **bandDistro** La distribución de probabilidad con la que se generan paquetes.
    0. **Constant** genera entre bandMean y bandStdDev paquetes al azar.
    1. **Gaussian** utiliza bandMean y bandStdDev con normalidad.
    2. **Gamma** utiliza bandMean de forma normal y bandStdDev para generar paquetes aleatorios en relación con la distribución.
- **bandMean** Parámetro utilizado para la generación de paquetes.
- **bandStdDev** Parámetro utilizado para la generación de paquetes, máximo 1/4 de bandMean.
- **packageRadsP** Paquetes que serán radioactivos (%).
- **packageUrgeP** Paquetes que serán urgentes (%).
- **packageLeftP** Paquetes que serán colocados al lado izquierdo de la banda (%).
- **bandQuantum** Tiempo para Round Robin o algoritmo de tiempo real.
- **bandParameter** Parámetro específico al algoritmo de la banda (signo, w, random).
- **bandAlgorithm** El algoritmo de control de la banda: 0 = W, 1 = Letrero, 2 = Random.
- **bandScheduler** El tipo de calendarizador: 0 = Round Robin, 1 = Prioridad, 2 = Más corto primero, 3 = FIFO, 4 = Tiempo Real.

Más información de los calendarizadores en la sección III-C y para los algoritmos de control de banda en la sección III-D

V. Coevaluación

Cuadro I:
Rúbrica de coevaluación (máx:5, min:1)

Rubro	Jimena	Roberto	Josafat
Iniciativa	5	5	5
Disponibilidad	5	5	5
Comunicación	5	5	5
Ejecución de tareas asignadas	5	5	5
Apertura a escuchar opiniones	5	5	5

VI. Bitácora

Cuadro II:
Tabla de actividades por día, de Josafat Vargas (JV) y Jimena Salas (JS)

Actividad	Fecha	Tiempo	Estudiante
Generación de paquetes	3/10	3h	JV
Actualización de probabilidade de generación	4/10	3h	JV
Cálculo de tiempo del paquete en la banda	5/10	2h	JV
Manejo de paquetes para la GUI	7/10	2h	JV
Lectura de archivos de configuración	8/10	3h	JV
Integración de generación de paquetes con listas enlazadas	9/10	3h	JV
Integración de lectura de archivos	10/10	3h	JV
Generación de paquetes para cada banda	11/10	2h	JV
Integración de generación de paquetes con resto del programa	12/10	8h	JV
Inclusión de restricciones para algoritmo de tiempo real	15/10	3h	JV
Integración de hardware	21/10	10h	JV
Ajustes menores en generación de paquetes	21/10	1h	JV
Creación de estructura de datos para listas enlazadas	08/10	4h	JS
Implementación de calendarizadores FIFO, shortest first y Round Robin	09/10	4h	JS
Actualización de progreso en Round Robin y visualización en GUI	10/10	4h	JS
Implementación en interfaz y lógica para soporte de movimiento en ambos lados	12/10	4h	JS
Integración de generación de paquetes, archivos de configuración y muestra de paquetes pendientes en la interfaz	12/10	5h	JS
Implementación de algoritmo de tiempo real	14/10	4h	JS
Sistema de pausa por software y hardware	21/10	5h	JS

Cuadro III:
Tabla de actividades por día de Roberto Gutiérrez

Actividad	Fecha	Tiempo	Estudiante
Búsqueda de ejemplos para la biblioteca de threads	03/10	3h	RG
Comienza implementación de la bilbioteca de lpthread utilizando contextos con ucontext	04/10	2h	RG
Intento de implementación del join utilizando contextos	05/10	4h	RG
Creación de la biblitoeca de threads con el makefile	07/10	1.5h	RG
Cambios a los threads para que funcione el join	08/10	4h	RG
Implementación del yield y exit del thread	08/10	3h	RG
Implementación de mutex para los threads	09/10	2h	RG
Cambios en biblioteca de threads lpthread para consistencia con pthread y para finalizar llamadas con señales	10/10	6h	RG
Creación de makefile general para el proyecto con todas las dependencias	12/10	2h	RG
Implementación de los algoritmos de control de la banda para la lista de paquetes en espera	12/10	3h	RG
Implementación inicial del hardware	17/10	4h	RG
Integración del hardware con el sistema de simulación	21/10	7h	RG

VII. Conclusiones y Recomendaciones

La estadística permite generar modelos que facilitan el diseño de sistemas al dar información sobre las variables que van a afectar al sistema. Con esta información se puede determinar, por ejemplo, la factibilidad de un sistema en tiempo real o los valores que debe tener el sistema para trabajar de manera correcta. Por otro lado, se logra comprender la importancia de los calendarizadores en un sistema operativo, ya que si estos no funcionaran de manera correcta, el usuario notaría el funcionamiento incorrecto del sistema.

Como recomendación se podría hacer una comunicación más robusta y eficiente entre la interfaz y la lógica, ya que leer y escribir archivos agrega mucho overhead al programa. Además, también podría optarse por realizar la comunicación con el Arduino mediante el lenguaje C y no por Python.

Referencias

[1] D. C. Montgomery and G. C. Runger, *Applied statistics and probability for engineers* New York: Wiley, 2003.
[2] Tanenbaum. *Sistemas Operativos: diseño e implementación*. Prentice Hall. 1988
[3] Goel, Aakash & Sharma, Gaurav. (2015). Challenges in Scheduling in Operating System.
[4] Daniel P. Bovet and Marco Cesati, Understanding the Linux Kernel, Third Edition. O'Reilly Media, 2005.
[5] Drepper, U., & Molnar, I. (2002). The Native POSIX Thread Library for Linux.
[6] Chowdhury, Subrata. (2018). Survey on various Scheduling Algorithms. Imperial Journal of Interdisciplinary Research (IJIR). 3. 4.