



UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN



PROGRAMA 3 – ALGORITMO DE PLANIFICACIÓN FCFS (FIRST COME, FIRST SERVED)

SEMINARIO DE SOLUCIÓN DE PROBLEMAS DE SISTEMAS OPERATIVOS

SECCIÓN: D01

CÓDIGO: 215468696

PROFESOR: VIOLETA DEL ROCIO BECERRA VELAZQUEZ

ALUMNO: DOMÍNGUEZ DURAN OSCAR ALEJANDRO

CARRERA: INGENIERÍA EN COMPUTACIÓN



23 DE ABRIL DE 2021

ACTIVIDAD DE APRENDIZAJE 6 – ALGORITMO DE PLANIFICACIÓN FCFS (FIRST COME, FIRST SERVE)

OBJETIVO

La mayor diferencia de este programa respecto al último realizado es que se deja de lado todo lo relacionado con los lotes, es decir, ya no vamos a trabajar con todos los procesos separados de 5 en 5 sino que ahora todos los procesos serán independientes de pertenecer a un lote. Este cambio representa una significativa diferencia en la manera en la que se debe abordar el programa, ya que ahora se deben contemplar más factores, por ejemplo, todos los tiempos que se generan en el ciclo de vida de cada proceso.

Se debe implementar el algoritmo de planificación FCFS, lo que conlleva la aplicación en código de los 5 estados de un proceso según el diagrama de 5 estados (*nuevo*, *listo*, *ejecución*, *bloqueado*, *terminado*). Al iniciar el programa se han de cargar los primeros 5 procesos en memoria, es decir, en la cola de procesos listos. Posterior a esta carga, los procesos nuevos se irán añadiendo a la cola de procesos listos cada que un proceso finalice (ya sea por error o de manera normal). Mientras un proceso es ejecutado este puede ser interrumpido lo que, a diferencia del programa pasado, cada interrupción mandará al proceso en ejecución a una cola de procesos bloqueados; en dicha cola, se habrá de esperar 5 segundos para después volver a la cola de procesos listos

Una vez finalizada la ejecución de todos los procesos, se deberá desplegar una tabla que muestre: el ID, la operación, el resultado y todos los tiempos solicitados en el documento correspondiente a esta actividad. Existen tiempos que pueden ser calculados conforme el proceso avanza en su ejecución y algunos tiempos que dependen de otros, por lo que será necesario tener un buen control y manejo de los tiempos de cada proceso.

DESARROLLO

Lenguaje y herramientas utilizadas:

Al igual que en el programa pasado, sigo empleando C++ y el *framework* de Qt ya que, pese a las complicaciones presentadas en las actividades previas, considero que he aprendido a utilizarlos de mejor manera, además, llegados a este punto, pese a que cada programa puede ser abordado desde el principio con algún otro lenguaje y/o *framework* sin problemas, la verdad es que no hay ningún otro en el que me sienta así de cómodo para llevar a cabo este tipo de programas. En esta actividad consideraré trasladar todo a Java o C#, ya que ambos lenguajes son buenos y cuentan con sus propias herramientas para desarrollar interfaces gráficas, pero siento que aún no los domino tanto como para implementar un programa de este tipo. Es por esto que,

por ahora, considero que los programas posteriores a este también serán realizados con C++ y Qt.

Funcionamiento del programa:

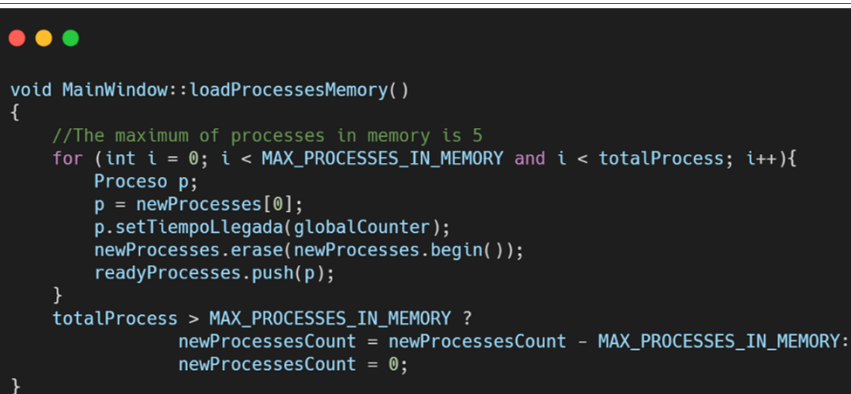
Para este punto si tengo bastantes cosas que mencionar respecto a los cambios que mi programa sufrió a lo largo de esta actividad, así que vamos detallando en que consistió cada uno de los cambios y como se abordó la solución. Lo primero que quiero mencionar es la nueva manera en la que organizo la información, para esto, será necesario recordar que en las actividades pasadas todos los procesos eran almacenados en un vector de vectores, en donde cada vector de este vector representaba un lote. Teniendo lo anterior en mente, para esta actividad consideré innecesario tener esa estructura de datos y si concebimos los estados de los procesos como arreglos, es más entendible manejar una estructura de datos independiente para cada estado del diagrama, es decir, un vector para los procesos nuevos, otro para los procesos finalizados, una cola para los procesos listos y otra para los procesos bloqueados. Adjunto imagen con la declaración de las estructuras correspondientes:



```
std::vector<Proceso> newProcesses;  
std::queue<Proceso> readyProcesses;  
std::queue<Proceso> blockProcesses;  
std::vector<Proceso> finishedProcesses;
```

Ilustración 1 - Estructuras para los estados de los procesos

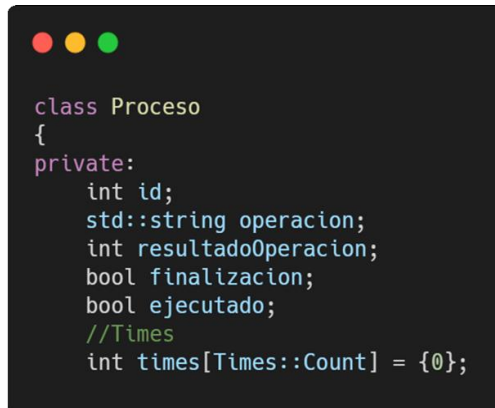
De esta manera, la captura de procesos es más sencilla. Todos los procesos que se generan al inicio estarán el vector de nuevos procesos e incrementamos el contador de procesos nuevos. Como al inicio solo podemos cargar 5 procesos, basta con tomar los primeros 5 de este arreglo y pasarlos a la cola de procesos listos; lo cual significa que serían eliminados del vector correspondiente al ser añadidos a la cola. Realicé una función que se encargue de lo anterior, así como de asignar el valor correspondiente al contador de procesos nuevos:



```
void MainWindow::loadProcessesMemory()  
{  
    //The maximum of processes in memory is 5  
    for (int i = 0; i < MAX_PROCESSES_IN_MEMORY and i < totalProcess; i++){  
        Proceso p;  
        p = newProcesses[0];  
        p.setTiempoLlegada(globalCounter);  
        newProcesses.erase(newProcesses.begin());  
        readyProcesses.push(p);  
    }  
    totalProcess > MAX_PROCESSES_IN_MEMORY ?  
        newProcessesCount = newProcessesCount - MAX_PROCESSES_IN_MEMORY:  
        newProcessesCount = 0;  
}
```

Ilustración 2 - Función para cargar los primeros 5 procesos en memoria

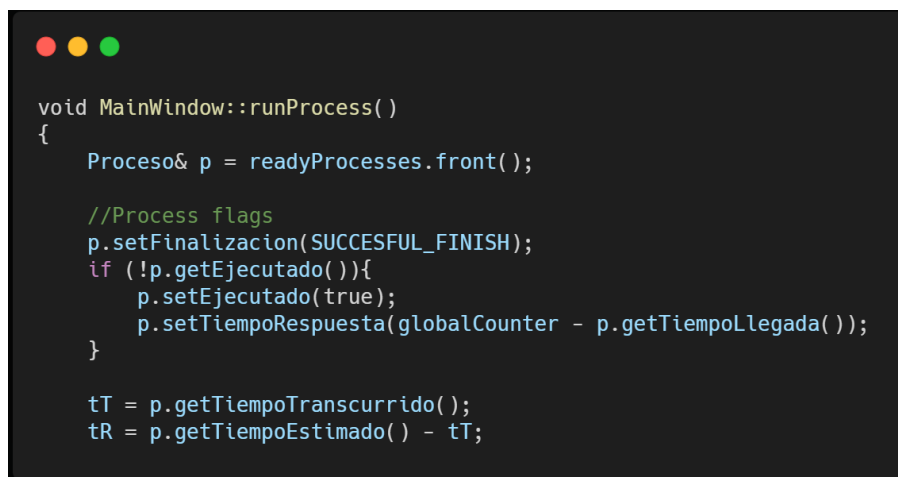
En la imagen anterior podemos apreciar que se les asigna a todos los procesos sus tiempos de llegada con el contador global, el cual, como aún no se han ejecutado los procesos, será igual a 0. Hablando de los tiempos, la clase correspondiente a moldear un proceso sufrió una serie de cambios, siendo el más relevante la incorporación de un arreglo con todos los tiempos necesarios. La clase “Proceso” termina teniendo los atributos utilizados en programas anteriores con el añadido del arreglo y una bandera, la cual será descrita más adelante. Adjunto imagen correspondiente a los atributos de la clase:



```
class Proceso
{
private:
    int id;
    std::string operacion;
    int resultadoOperacion;
    bool finalizacion;
    bool ejecutado;
    //Times
    int times[Times::Count] = {0};
```

Ilustración 3 - Atributos de la clase "Proceso"

Volviendo al funcionamiento del programa, cuando los procesos comienzan a ejecutarse lo primero que se hace es tomar el elemento tope de la cola de procesos listos (cabe mencionar que elegí una cola para los procesos listos ya que las colas siguen el funcionamiento FIFO), por cada proceso que vamos a correr, tenemos que marcar que esté se ha ejecutado mediante su bandera, ya que es necesario asignar su tiempo de respuesta, el cual se calcula tomando el contador global en ese momento y restando el tiempo de llegada, este tiempo solo debe ser calculado una sola vez cuando este se ejecuta por primera ocasión. Lo anterior está descrito en las siguientes líneas de código:



```
void MainWindow::runProcess()
{
    Proceso& p = readyProcesses.front();

    //Process flags
    p.setFinalizacion(SUCCESSFUL_FINISH);
    if (!p.getEjecutado()){
        p.setEjecutado(true);
        p.setTiempoRespuesta(globalCounter - p.getTiempoLlegada());
    }

    tT = p.getTiempoTranscurrido();
    tR = p.getTiempoEstimado() - tT;
```

Ilustración 4 - Asignación del tiempo de respuesta

Cuando ya tenemos los datos previos, asignamos el valor del tiempo transcurrido del proceso al contador correspondiente, mostramos la información del proceso en la tabla de procesos en ejecución y, al igual que el programa pasado, la información en pantalla se estará actualizando cada segundo gracias a la función que añade el *delay* de un segundo que se mostró en el programa pasado. A la par que ejecutamos el proceso actual, es necesario actualizar el contador de tiempo bloqueado para los procesos que se encuentren en dicho estado, pero ¿Cómo se maneja ese cambio de ejecución a bloqueado?, pues bien, la respuesta es un tanto larga ya que necesitamos recordar algunos aspectos previos. En el programa pasado se utilizaba la clase/función para la detección de las teclas que el propio Qt posee, aquí la seguimos utilizando obviamente, con el añadido de que cada que se pulse una tecla ya no mandarán a llamar a una función sino que solo activarán unas bandera, las cuales me permiten realizar las funciones indicadas fuera de ese flujo del programa; esto porque uno de los principales problemas que me topé fue que cada que se pulsa una tecla e inmediatamente mando a llamar a una función, hay un desfase entre el flujo principal del programa y lo que se está ejecutando, por lo que me salían datos erróneos, principalmente relacionados con los tiempos del proceso, sobre todo el tiempo transcurrido.

Para evitar el problema descrito anteriormente, fue necesario agregar dos banderas a la clase de la ventana principal: una bandera para saber cuándo el proceso finaliza por error y otra para cuando ese genera una interrupción y tenemos que mandar un proceso a la cola de bloqueados, de manera que cada que presiono la tecla correspondiente activo una u otra bandera. De esta forma, evité que sucedieran cosas extrañas con el control del programa y los tiempos de cada proceso, ya que le toma mucho menos tiempo al programa el solo cambiar el valor de una bandera a tener que mandar a llamar funciones. Bien, retomando el cuestionamiento de como pasamos un proceso en ejecución a la cola de procesos bloqueados... lo que se tiene que hacer es detectar la interrupción, una vez detectada, procedemos a tomar el proceso actual que se estaba ejecutando y lo pasamos a la cola de procesos bloqueados, una vez añadido a dicha cola, lo eliminamos de la cola de procesos listos. Para mostrarlo en su tabla hacemos algo similar que, con las otras tablas, es decir, recorremos la estructura de datos, recuperamos los valores que nos interesan y los mostramos en el *QWidget* correspondiente.

Cuando tenemos procesos bloqueados es necesario incrementar su tiempo de bloqueo, algo que a priori suena sencillo, pero me tomó mi tiempo hacer que se actualizará la información del proceso bloqueado en su tabla correspondiente. El ciclo en el cual incrementamos los tiempos transcurridos es el ciclo en donde la información necesita actualizarse cada segundo (las otras tablas dependen de si un proceso llega o si termina), como al bloquear un proceso el programa toma al siguiente para ejecutarlo, se debe de observar como al mismo tiempo uno aumenta su tiempo transcurrido y el otro su tiempo en bloqueado. Para lograr mostrar la información mencionada de esa manera, fue necesario incrementar dentro del ciclo mencionado el tiempo bloqueado de cada proceso en la cola de procesos bloqueados, claro, siempre y cuando esta no esté vacía. Por si no quedó del todo claro (lo cual es perfectamente entendible porque es un tanto engorroso de describir en texto) mostraré dos partes del código para esto: la primera

será una parte del ciclo “principal” mencionado y la otra será del método para incrementar los tiempos de bloqueo de cada proceso en la cola:

```
//Increment de tT and decrement the tR
while (tT < p.getTiempoEstimado() and tT != ACTION_CODE){
    tR--;
    tT++;
    p.setTiempoTranscurrido(tT);
    ui->processRuningTB->setItem(3,0,new QTableWidgetItem(QString::number(p.getTiempoTranscurrido())));
    ui->processRuningTB->setItem(4,0,new QTableWidgetItem(QString::number(tR)));

    //Incrment blocked process count
    if (!blockProcesses.empty()){
        incrementBlockedTimes();
    }
    showBlockedProcesses();
    globalCounter++;
    ui->globalCountLCD->display(globalCounter);
    ui->newProcessesLCD->display(newProcessesCount);
    delay();
}
```

Ilustración 5 - Ciclo en donde se incrementan los contadores mostrados en pantalla

Si la cola de procesos bloqueados no está vacía quiere decir que aumentaremos en 1 el tiempo de bloqueo de cada proceso que se encuentre aquí. Debido a que es una cola sencilla, para recorrerla es necesario sacar al primer elemento y luego ponerlo al final. En esta misma función se necesita validar si el tiempo ya llegó al valor de 5 (el tiempo máximo de bloqueo), una vez el tiempo de bloqueo del proceso llega a esa cifra, se elimina de la cola de procesos bloqueados y se agrega a la cola de procesos listos; lo cual suena bien, ¿no?, pues hay un pequeño problema, ya que para iterar en la cola se compara con el tamaño de esta, y si se elimina una en pleno ciclo, puede que nos saltamos algunos procesos bloqueados, es por eso que solo incrementamos el valor de “i” cuando el no hay eliminación de un elemento, de esta manera el valor se quedaría en la posición disponible siguiente y podríamos incrementar los tiempos de bloqueo de todos los procesos restantes. Y, de hecho, pensándolo bien, al ser una cola siempre el primero que llegó será el primero en salir, por lo que no habría conflictos con que un elemento más arriba del primero termine, ya que no podrían llegar procesos al mismo tiempo a esa cola,

siempre hay por lo menos un segundo de diferencia. Bueno, aclarado lo anterior, adjunto la parte del código descrita del lado izquierdo.

```
void MainWindow::incrementBlockedTimes()
{
    int bt,i;

    //Increment the blocked time of each process
    for (i = 0; i < (int)blockProcesses.size();){
        Proceso bp = blockProcesses.front();
        bt = bp.getTiempoBloqueado() + 1;
        blockProcesses.pop();
        bp.setTiempoBloqueado(bt);
        if (bp.getTiempoBloqueado() > BLOCKED_TIME){
            bp.setTiempoBloqueado(0);
            readyProcesses.push(bp);
            showReadyProcesses();
        }
        else{
            blockProcesses.push(bp);
            i++;
        }
    }
}
```

Ilustración 6 - Función para incrementar los tiempos de bloqueo

Bien, ya vista la parte correspondiente a lo de la transición de un proceso listo a bloqueado y de bloqueado a listo podemos pasar al siguiente punto: cada que un proceso termina de ejecutarse se tiene que tomar otro proceso del vector de procesos nuevos. Aquí radica la principal diferencia respecto al programa anterior, mientras que antes cargábamos lote por lote, aquí tenemos que cargar un proceso cada que haya un espacio disponible en memoria, o sea, en la cola de procesos listos. Hacer esto es sencillo realmente, basta con extraer el primer elemento del arreglo de procesos nuevos y agregarlo a la cola de procesos listos, posterior a esto, eliminamos dicho elemento del arreglo de procesos nuevos y decrementamos el contador de los procesos nuevos. Cada que un nuevo proceso se añade a la cola de procesos listos, se fija su tiempo de llegada con el valor del contador global. Adjunto la parte del código correspondiente.



```
//Adding a new process
if (newProcesses.size()){
    Proceso p;
    p = newProcesses[0];
    newProcesses.erase(newProcesses.begin());
    //Setting arrive time for each new process in ready processes
    p.setTiempoLlegada(globalCounter);
    readyProcesses.push(p);
    newProcessesCount--;
}
```

Ilustración 7 - Añadir nuevo proceso a la cola de listos

Hasta el momento, si se observa con detalle, ya hemos obtenido por cada proceso sus tiempos de llegada, respuesta y el transcurrido que siempre está presente, el resto de los tiempos los calculamos cuando el proceso finaliza su ejecución. Para evitar calcular uno por uno mientras el programa ejecuta los procesos, preferí ir añadiendo al vector de procesos finalizados a cada proceso que terminase por error o normalmente, este arreglo me serviría para posteriormente mostrar todos los tiempos. Una vez todos los procesos terminan de ejecutarse, ejecuto un método que recorre el arreglo de procesos terminados y calcula sus tiempos faltantes (retorno, espera y servicio, aunque este último está implícito en el tiempo transcurrido). Adjunto la parte del código correspondiente al método descrito.



```
void MainWindow::calculateProcessesTimes()
{
    for (int i = 0; i < (int)finishedProcesses.size(); i++){
        Proceso& p = finishedProcesses[i];
        p.setTiempoRetorno(p.getTiempoFinalizacion()-p.getTiempoLlegada());
        p.setTiempoServicio(p.getTiempoTranscurrido());
        p.setTiempoEspera(p.getTiempoRetorno()-p.getTiempoServicio());
    }
}
```

Ilustración 8 - Método para calcular los tiempos necesarios del proceso

Un punto importante que no había mencionado hasta el momento es la existencia de un proceso nulo, el cual se usa única y exclusivamente cuando la cola de procesos listos se queda vacía, algo que es más propenso a ocurrir cuando quedan menos de 4 procesos, pero que, en efecto puede pasar. Pues bien, lo que ocurre es que cuando detectamos que la cola de procesos listos se ha quedado vacía, generamos uno nuevo con datos en 0, de esta manera al agregarlo a la cola de procesos listos, el programa no se detiene, sin embargo, cabe aclarar que este proceso realmente no es ejecutado como tal, ya que al poseer información nula, no puede incrementar su tiempo transcurrido, por lo que solo lo utilizo para que el programa no se detenga en lo que un proceso bloqueado sale de la cola de bloqueados, una vez la cola de listos tiene otro proceso, eliminamos el nulo y dejamos de mostrar su información en pantalla. Adjunto las partes descritas anteriormente:

```
//Generating a null process
if (readyProcesses.size() == 0 and blockProcesses.size() > 0){
    Proceso nullP(0,"NULL",0,false);
    readyProcesses.push(nullP);
}

//Comparing if there is another process further than the null process
while (readyProcesses.size() < 2) {
    incrementBlockedTimes();
    globalCounter++;
    ui->globalCountLCD->display(globalCounter);
    ui->newProcessesLCD->display(newProcessesCount);
    delay();
}
```

Ilustración 9 - Creación y eliminación de un proceso nulo

Finalmente, una vez ejecutados todos los procesos y calculados todos los tiempos, se muestra en pantalla una tabla con toda la información recabada. Esta tabla se despliega en una nueva página del *stackedwidget* del programa, por lo que no se puede considerar como una nueva ventana ni nada por el estilo. Debido a que es complicado describir en capturas la corrida del programa, solo mostraré una captura del programa a media ejecución y otra de la tabla una vez todo haya finalizado.

Nuevos procesos:

SIMULADOR DE PROCESOS

Listos

ID	TME	TT
2	6	2
4	12	2

Proceso en ejecución

ID	7
OPE	-69%-60
TME	14
TT	3
TR	11

Bloqueados

ID	TTB
6	4
1	3

Terminados

ID	OPE	RL
3	12/29	ERROR
5	98/2	ERROR

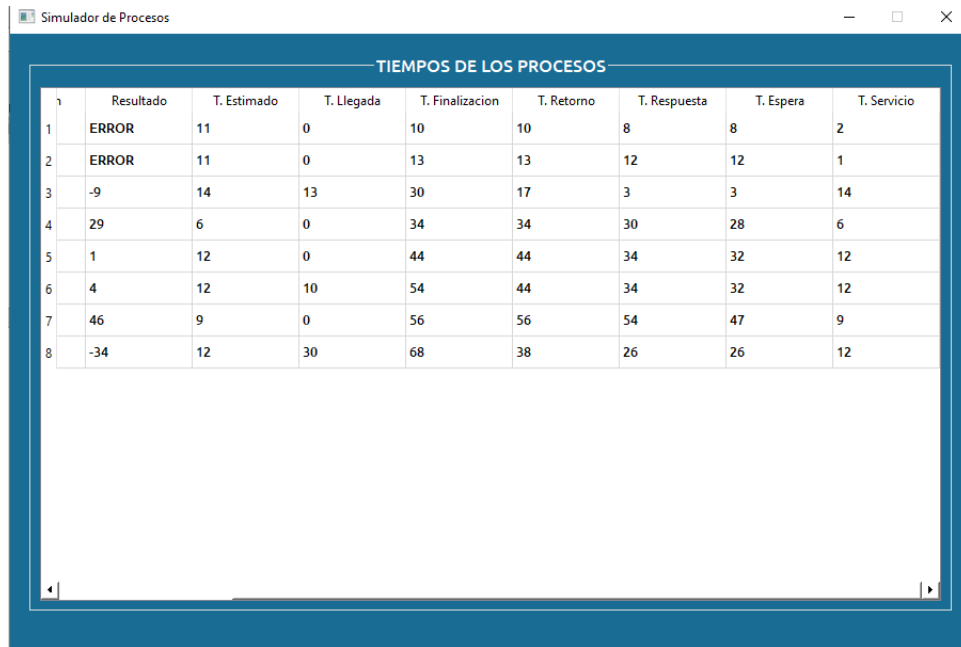
Contador global:

49

Comenzar

Ilustración 10 - Programa corriendo con 8 procesos

En la siguiente captura, es necesario mencionar que se ve la cabecera horizontal con los números de cada fila, no es el ID, esto ya que la tabla es bastante larga y no se puede mostrar todo en pantalla:



	Resultado	T. Estimado	T. Llegada	T. Finalizacion	T. Retorno	T. Respuesta	T. Espera	T. Servicio
1	ERROR	11	0	10	10	8	8	2
2	ERROR	11	0	13	13	12	12	1
3	-9	14	13	30	17	3	3	14
4	29	6	0	34	34	30	28	6
5	1	12	0	44	44	34	32	12
6	4	12	10	54	44	34	32	12
7	46	9	0	56	56	54	47	9
8	-34	12	30	68	38	26	26	12

Ilustración 11 - Tabla con todos los tiempos de los procesos

CONCLUSIÓN

En esta actividad no me sentí tan angustiado por los problemas que surgieron a lo largo de esta, a decir verdad, creo que no se comparan con los problemas que tuve en los primeros dos programas, puede que sea la práctica, ya que mis problemas se han debido al como mostrar la información en la interfaz gráfica de Qt, algo que en un principio yo consideré que no sería tan complicado, pero conforme avanzan los programas termino aprendiendo cosas nuevas del *framework* o del lenguaje incluso. También creo que otro factor que afectó al desarrollo del programa es que ya no le pude dedicar tanto tiempo por día, debido a que conseguí un trabajo y mis tiempos cambiaron un poco. Definitivamente son programas que pese a no ser muy complejos son un tanto engorrosos y requieren de tiempo y claro que este programa no sería la excepción.

Respecto al programa funcionando, considero que estoy satisfecho al verlo ejecutarse y con los resultados obtenidos, observar las transiciones vistas en el diagrama de 5 estados reflejadas en el programa es algo que me hace pensar que realmente estamos aplicando la teoría en la práctica. Las únicas partes donde me atoré realmente fueron la parte del proceso nulo, ya que no sabía como representarlo sin alterar el correcto funcionamiento del programa y sobre todo, lo que más me tomó tiempo fue el arreglar el problema que tenían los tiempos cada que generaba una interrupción, ya que manejaba de manera errónea al proceso actual, por lo que cada que mandaba a llamar a una función dentro del evento del tecleo, generaba un desfase que

me hacía perder la referencia del proceso con el que estaba trabajando. También cabe mencionar, que me tomó mi tiempo el comprender como mostrar de manera “síncrona” como se actualizaba la tabla de proceso en ejecución y la tabla de procesos bloqueados; es por eso por lo que digo que mis problemas siempre van ligados más al como mostrar la información que a la implementación del algoritmo en cuestión. Por último, me gustaría mencionar que, pese a todas las trabas y los reajustes a mi tiempo, el poder haber concluido en tiempo y forma representa una gran satisfacción y de alguna manera u otra me motiva a seguir con la clase y administrarme mejor, claro está.

Link del vídeo del programa:

- <https://youtu.be/F7mJjZarOBi>