



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



PROGRAMA 5 – ALGORITMO DE PLANIFICACIÓN RR (ROUND-ROBIN)

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



28 DE MAYO DE 2021

ACTIVIDAD DE APRENDIZAJE 10 – ALGORITMO DE PLANIFICACIÓN RR (ROUND ROBIN)

OBJETIVO

En esta actividad se ha de implementar el algoritmo de planificación *Round Robin*, con el cual ya estamos familiarizados debido a que en investigaciones previas se analizó su funcionamiento y conceptos básicos como el *Quantum*, el cuál, es la unidad de tiempo designada para que cada proceso dure la cantidad de tiempo especificada en el procesador antes de ser cambiador por otro proceso; de esta manera se logra un uso de la CPU más “equitativo” en términos de tiempo, puesto que tarde o temprano los procesos con un tiempo estimado menor, saldrán mucho antes que si fuesen los últimos en un FCFS.

En general, se deberá adecuar el programa pasado para que ahora funcione no con el algoritmo FCFS, sino que empleé el algoritmo RR. Esto conlleva un par de añadidos que se han de implementar, por ejemplo, ahora no basta con solicitar solo el número de procesos al usuario, sino que también es necesario solicitarle el valor del Quantum (en mi caso, expresado en segundos). Este *Quantum* deberá ser asignado a cada proceso de tal manera que cada proceso que se genere posea como *Quantum* la cantidad solicitada por el usuario. Existirán ocasiones en las que será necesario reiniciar el valor de este; el caso más claro es cuando este se agota, una vez un proceso agotó su *Quantum* este deberá salir de ejecución y pasar a la cola de listos para posterior a esto, seleccionar al siguiente proceso en la cola de listos. Otro caso en el que se debe reiniciar el valor del *Quantum* es cuando mandamos al proceso en ejecución a la cola de bloqueados, es decir, cuando ocurre una interrupción, esto debido a que si no se reiniciase el proceso duraría mucho menos en CPU de lo que establece el *Quantum*. Un aspecto importante para considerar es el caso del proceso nulo, pues este también deberá tener un Quantum e independientemente de como termine, este permanecerá en ejecución hasta que haya procesos listos de nuevo. Además, se deberá de poder apreciar el cómo se van intercalando estos procesos en base a su *Quantum* por lo que se debe notar como el tiempo transcurrido de cada proceso va acorde a lo que el *Quantum* avanza; por lo que será necesario mostrar el valor de este en todo momento cuando se está ejecutando el proceso. En caso de que solo quedara un proceso en ejecución, el Quantum se reiniciará continuamente hasta que el proceso termine su ejecución.

Respecto a las funcionalidades anteriormente implementadas, el programa deberá seguir contando con todas las funcionalidades realizadas, es decir, se debe tener la opción de interrumpir un proceso, finalizarlo por error, agregar nuevos procesos, pausarlo y mostrar la tabla de procesos con sus respectivos BCP; de manera que los verdaderos cambios sustanciales están en el como se ejecutarán los procesos. Además de que se deberán conservar otros aspectos básicos manejados en programas anteriores como lo son la visualización de todos los procesos listos, finalizados, en ejecución y bloqueados.

DESARROLLO

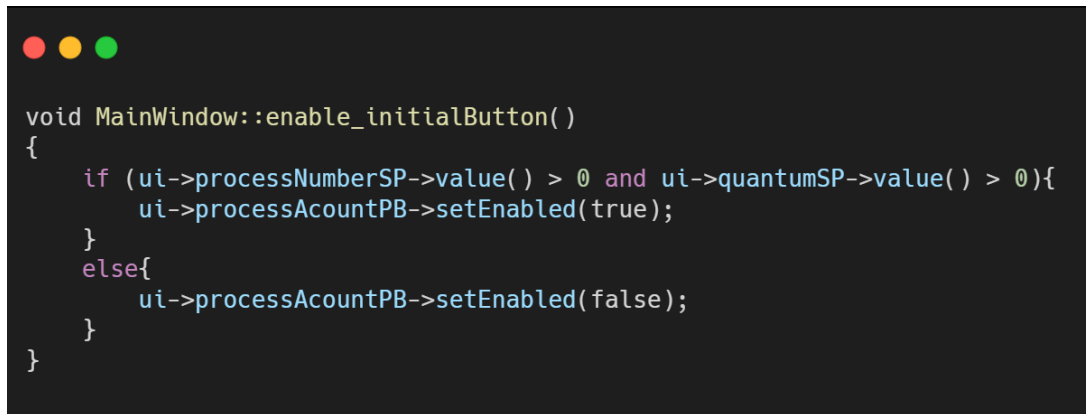
Lenguaje y herramientas utilizadas:

Para este programa, decidí seguir trabajando con el lenguaje de C++ y el *framework* de Qt, ya que, para esta actividad, necesitaba las bases planteadas en los programas anteriores para que el desarrollo de este no fuese tan complicado, o más bien, para no empezar de 0 un programa que ya tiene cierto grado de complejidad en cuanto a código y lógica. De igual manera, conforme han pasado los programas pasados, considero que ya le he encontrado el punto tanto al lenguaje como al *framework*, por lo que estoy bastante cómodo con ambas herramientas y considero que cambiarlas en este punto representaría un atraso en la entrega de las actividades puesto que aunque ya estoy familiarizado con C# o Java para el manejo de interfaces gráficas, lo cierto es que aún no me siento tan seguro en esos lenguajes y pese a que son bastantes similares entre ellos, poseen una curva de aprendizaje muy diferente a la que hay en C++, por lo que estoy seguro que si decido cambiar a escasas semanas de terminar el semestre, entorpecería mi entrega de trabajos y probablemente me atrasaría más con los programas posteriores. No obstante, soy partidario de pensar que no hay que descartar posibles opciones y quién sabe, quizá en los futuros programas me resulté mucho más fácil implementarlos en otros lenguajes que seguir haciéndolo con C++ y Qt; pero de momento sigo trabajando con estos para el desarrollo de esta actividad.

Funcionamiento del programa:

Lo primero que se tuvo que realizar para esta actividad fue modificar la clase del “Proceso” para que ahora se pudiera almacenar y recuperar el valor del *Quantum* en cada proceso. Por lo que se añadió un atributo de carácter privado para poder gestionar este elemento dentro del programa. El *Quantum* será de tipo entero y decidí dejarlo como un atributo privado debido a que, si se llega a modificar de manera errónea, podría afectar al desempeño del algoritmo. Agregar este atributo de manera privada, conlleva que también se añadan nuevos métodos para poder almacenar su valor y recuperarla de manera segura, así como la modificación de otros métodos ya existentes como la sobrecarga de operadores en donde ahora también se debe añadir este atributo; cambios mínimos pero que si no se llegan a tener en consideración para la realización de la actividad pueden generar problemas o dolores de cabeza. Debido a la sencillez de los cambios realizados en esta clase, no considero necesario mostrar alguna parte del código en concreto puesto que considero que con la explicación previamente realizada es suficiente para entender la manera en la que se llevó acabo dicho añadido en la clase del proceso y las repercusiones que esta trajo consigo.

Posterior a añadir este atributo a la clase correspondiente del proceso, procedí a modificar la interfaz gráfica para que esta pidiera los datos necesarios, es decir, la cantidad de procesos y el *Quantum*, además de añadir en la tabla de proceso en ejecución la fila correspondiente al valor del *Quantum*, también cambios mínimos pero que por lo menos en el caso de pedir el valor del *Quantum* requirió que modificara las validaciones para poder habilitar el botón de “Aceptar”, el cual inicializa todos los procesos del programa cuando es presionado.



```
void MainWindow::enable_initialButton()
{
    if (ui->processNumberSP->value() > 0 and ui->quantumSP->value() > 0){
        ui->processAccountPB->setEnabled(true);
    }
    else{
        ui->processAccountPB->setEnabled(false);
    }
}
```

Ilustración 1 - Validación para poder generar los procesos

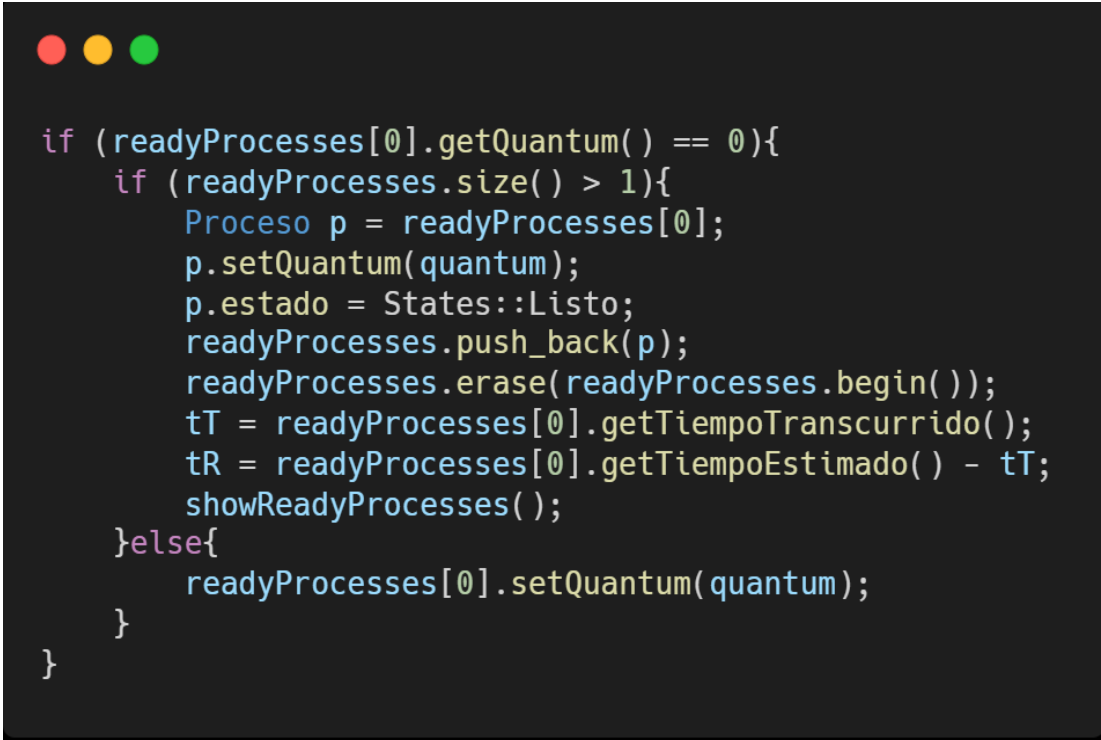
Como podemos apreciar en la imagen anterior, no consideré necesario validar la cantidad del *Quantum* más allá de que este fuera mayor a 0 puesto que además de no ser un requisito como tal, consideré que la esencia de este algoritmo es que su funcionamiento óptimo depende de la cantidad del *Quantum* con la que el algoritmo trabaje, puesto que como ya vimos si se le da un valor bajo, este puede tardar más de lo normal en ejecutar todos los procesos o si se le da un tiempo muy alto este pasará a comportarse como FCFS sin más; es por eso que decidí dejar la opción abierta a que se puedan probar los casos de un *Quantum* no eficiente así como los casos de un valor intermedio que funcione para poder trabajar con varios procesos.

Antes de inicializar los procesos es importante recuperar el valor del *Quantum* ingresado por el usuario, para esto se creó un atributo en la clase principal del programa que lo almacena; de esta manera, cada que generemos un proceso o reiniciemos su *Quantum* tendremos el valor original que el usuario puso. Cada que generamos un nuevo es necesario fijar este valor al atributo correspondiente del proceso mediante su método *setter* previamente implementado en la clase del proceso. Ya que tenemos el valor del *Quantum* fijado en todos los procesos, podemos describir como es el funcionamiento del programa ahora que ya se ejecutan mediante el algoritmo de *Round Robin*.

Lo primero que se hace al iniciar a trabajar con los procesos es cargar los procesos correspondientes en la cola de listos y de ahí tomar el proceso que esté al inicio de la cola (al igual que en programas pasados). Cada que seleccionemos el proceso que se va a estar ejecutando su estado cambiará a ejecución y por cada segundo que pase su tiempo transcurrido

aumentará mientras que el tiempo de su *Quantum* disminuirá. Podemos decrementar el valor del *Quantum* de cada proceso mediante la invocación de sus métodos para fijar y recuperar el valor de este, de manera que cada segundo fijamos el valor actual del *Quantum* menos 1; en otras palabras, usamos el *setter* para fijar el valor que retorne el *getter* menos 1. Esto último fue realizado de esa manera con el fin de no tener que crear variables auxiliares innecesarias para el funcionamiento de la ejecución de un proceso.

Cada que decrementemos el valor del *Quantum* será necesario validar si este ya se agotó, esto se realiza mediante un disparador el cual compara el valor del *Quantum* del proceso con 0, en caso de que sea cierto, es decir, ya se agotó su *Quantum* tenemos que checar si hay más procesos en la cola de listos ya que en caso de que ya no haya más esto querrá decir que solo queda un proceso para ejecutar y por ende la dinámica de como se trata al proceso cambia un poco. Cuando hay más de un proceso en la cola de listos recuperamos al proceso que se está ejecutando, le reseteamos el valor original del *Quantum* ingresado por el usuario y cambiamos su estado de *Ejecución* a *Listo* para posteriormente mandarlo al final de la cola de listos y obtener los nuevos contadores globales del tiempo transcurrido y tiempo restante según el nuevo proceso que esté en el inicio de la cola. En el caso de que solo quede un proceso en la cola de listos, es decir, un proceso para ejecutar entonces solo basta con resetear su *Quantum*, ya que al ser el único en la cola de listos y, por ende, el único que se va a estar ejecutando no tiene sentido que lo saquemos y agreguemos de nuevo a la cola. La parte del código descrita anteriormente es la siguiente:



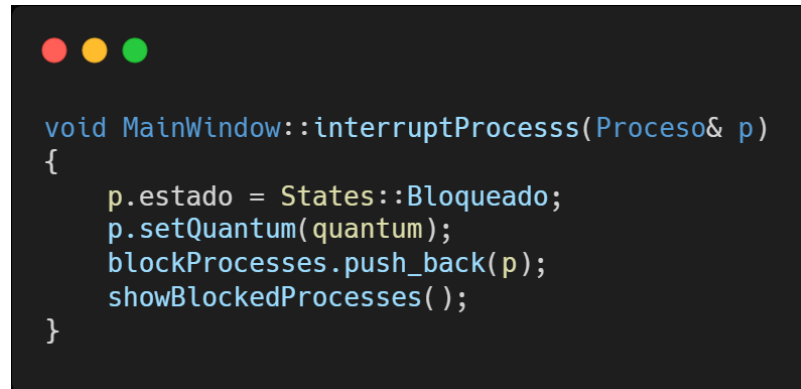
```

if (readyProcesses[0].getQuantum() == 0){
    if (readyProcesses.size() > 1){
        Proceso p = readyProcesses[0];
        p.setQuantum(quantum);
        p.estado = States::Listo;
        readyProcesses.push_back(p);
        readyProcesses.erase(readyProcesses.begin());
        tT = readyProcesses[0].getTiempoTranscurrido();
        tR = readyProcesses[0].getTiempoEstimado() - tT;
        showReadyProcesses();
    }else{
        readyProcesses[0].setQuantum(quantum);
    }
}

```

Ilustración 2 - Validación en caso de que el tiempo del Quantum se agote

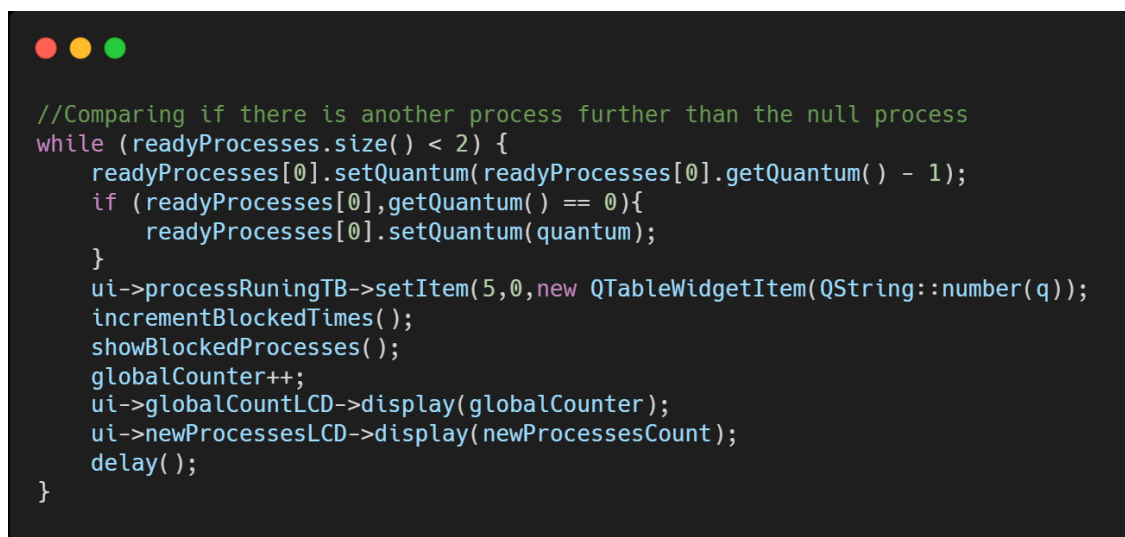
Cómo ya se había mencionado previamente, el valor del *Quantum* se puede reiniciar cada que este se agotó o en el caso de que el proceso sea interrumpido. Lo reestablecemos ya que cada vuelta dentro del algoritmo de *Round Robin* debe respetar el *Quantum* que se estableció, de otra manera no durarían lo mismo todos los procesos en ejecución y se perdería ese toque de “equidad” que busca este algoritmo. La función de interrupción del proceso fue modificada solo para recibir por referencia al proceso indicado y resetear su *Quantum*:



```
void MainWindow::interruptProcesss(Proceso& p)
{
    p.estado = States::Bloqueado;
    p.setQuantum(quantum);
    blockProcesses.push_back(p);
    showBlockedProcesses();
}
```

Ilustración 3 - Reinicio del Quantum cuando el proceso es interrumpido

En el caso en el que el proceso nulo sea el que está en ejecución debido a que los procesos que estaban en ejecución pasaron a estar en la cola de bloqueados, se deberá fijar a dicho proceso nulo su correspondiente *Quantum* y también, cada que esté se agota se reiniciará, aunque en cuanto se detecte un nuevo proceso en la cola de listos este se quitará de la ejecución pero es importante contemplar el caso en el que el proceso nulo pueda estar más de 5 segundos en ejecución, así que no está demás hacer esta validación y de paso no discriminamos a ningún proceso con su *Quantum* ni si quiera al nulo.



```
//Comparing if there is another process further than the null process
while (readyProcesses.size() < 2) {
    readyProcesses[0].setQuantum(readyProcesses[0].getQuantum() - 1);
    if (readyProcesses[0].getQuantum() == 0){
        readyProcesses[0].setQuantum(quantum);
    }
    ui->processRuningTB->setItem(5,0,new QTableWidgetItem(QString::number(q)));
    incrementBlockedTimes();
    showBlockedProcesses();
    globalCounter++;
    ui->globalCountLCD->display(globalCounter);
    ui->newProcessesLCD->display(newProcessesCount);
    delay();
}
```

Ilustración 4 - Contemplando el Quantum cuando el proceso nulo es ejecutado

PROGRAMA 5 – ALGORITMO DE PLANIFICACIÓN RR (ROUND-ROBIN)

Una vez cubiertos los aspectos relacionados a la implementación del algoritmo de *Round Robin* en el programa. Podemos mostrar algunas capturas del programa en ejecución con el fin de poder mostrar los resultados obtenidos con todo lo realizado. Puede que algunos aspectos sean más notables en el vídeo más que en las capturas, pero se hará énfasis en las funcionalidades descritas a lo largo del reporte.

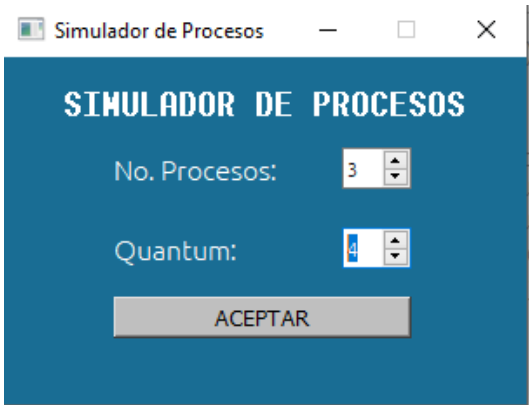


Ilustración 5 - Pantalla de la captura de la cantidad de procesos y el Quantum



Ilustración 6 - Primera vuelta de RR



Ilustración 7 - Segunda vuelta de RR

En las imágenes previamente mostradas podemos apreciar como en la primera vuelta los procesos en la cola de listos cuentan con el mismo tiempo transcurrido, el cual es equivalente al valor del *Quantum* especificado en la primera captura de pantalla, es decir, el valor de 4. En la segunda vuelta podemos apreciar como ahora los valores del tiempo transcurrido de los procesos en la cola de listas valen el doble del valor del *Quantum* original, o sea, el valor de 8.

CONCLUSIÓN

Esta actividad resultó ser un tanto más sencilla que los 2 últimos programas realizados, considero que en gran medida esto se debe a que iniciando este programa ya tenía muchas de las bases planteadas para este programa, por lo que solo fue necesario realizar las adecuaciones necesarias para poder implementar el algoritmo de *Round Robin*. Lo más complicado de esta actividad realmente fue el lograr que no hubiese ningún desfase entre los tiempos transcurridos y el tiempo actual del *Quantum* por que dependiendo de como este fuera decrementado podría darse el caso en el que el tiempo transcurrido fuese de 3 y el *quantum* de 4, no obstante, me di cuenta que esto era por un *bug* que surgió cuando, en esta actividad me quise ahorrar una línea de código haciendo la sumatoria del tiempo transcurrido después de mostrarlo cuando esto

debería ser al revés, en pocas palabras, estaba teniendo problemas para poder mostrar de manera correcta los tiempos transcurridos en los procesos y lo peor es que fue por intentar factorizar una parte que realmente no era necesaria de factorizar por lo que me tomó un rato dar con el error que yo mismo generé tiempo atrás.

En general, considero que este programa no me resultó tan complicado como los otros, o mejor dicho, laborioso, si es cierto que las validaciones tuvieron su chiste pero como ya dije, el mayor problema que tuve con esta actividad no fue debido a la implementación del algoritmo o de alguna especificación solicitada sino que lo generé sin querer tratando de ahorrar código, por lo que realmente creo que esta actividad fue más de dedicarle tiempo a que todo estuviera bien y a hacer las validaciones correctamente en lugar de tener que estar rompiéndome la cabeza pensando como podía corregir algún problema que surgiese durante el desarrollo de esta. Creo que la razón de que esta actividad me resultará más sencilla sin duda alguna es porque el programa pasado ya estaba bastante pulido y funcionaba correctamente, considero que, si no lo hubiera estado, este programa me hubiera dado muchos problemas respecto a que los tiempos funcionasen de manera correcta y a que todo cuadrara en la interfaz.

Como conclusión puedo decir que estoy bastante satisfecho con el resultado final y el como funciona el programa, considero que se cumplió el objetivo de la actividad y también creo que el tiempo que se nos fue brindado me permitió trabajar con más calma en esta actividad y de manera más eficiente. Además de que me permitió realizar algunas adecuaciones al código fuente y dejarlo un tanto más legible al igual que limpio. Llegados a este punto puedo decir que estoy bastante feliz con todo el trabajo que hasta el momento he realizado, cada programa de investigación me ha aportado de una manera u otra, nuevos conocimientos que me facilitan la implementación de este tipo de actividades. Estoy bastante conforme con el desarrollo de este tipo de actividades y expectante de los programas restantes.

Link del vídeo del programa:

- [Programa 5 \(Algoritmo Round Robin\) - SSP Sistemas Operativos - YouTube](#)