Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

TP1 - Scheduling

Sistemas Operativos

Primer Cuatrimestre de 2015

Apellido y Nombre	LU	E-mail
Cisneros Rodrigo	920/10	rodricis@hotmail.com
Rodríguez, Agustín	120/10	agustinrodriguez90@hotmail.com
Tripodi, Guido	843/10	guido.tripodi@hotmail.com

Contents

1 Introducción

En este Trabajo Práctico estudiaremos diversas implementaciones de algoritmos de scheduling. Haciendo uso de un simulador provisto por la cátedra podremos reprensentar el comportamiento de estos algoritmos. Implementaremos dos Round-Robin, uno que permite migración de tareas entre núcleos y otro que no y a través de experimentación intentaremos comparar ambos algoritmos. Asimismo, basándonos en un paper implementaremos una versión del algoritmo para schedulers con prioridades dinamicas y estaticas y mediante experimentos intentaremos comprobar ciertas propiedades que cumple el algoritmo.

2 Desarrollo y Resultados

3 Parte I – Entendiendo el simulador simusched

3.1 Ejercicios

- **Ejercicio 1** Programar un tipo de tarea TaskConsola, que simulara una tarea interactiva. La tarea debe realizar n llamadas bloqueantes, cada una de una duracion al azar 1 entre bmin y bmax (inclusive). La tarea debe recibir tres parametros: n, bmin y bmax (en ese orden) que seran interpretados como los tres elementos del vector de enteros que recibe la funcion.

- **Ejercicio 2** Escribir un lote de 3 tareas distintas: una intensiva en CPU y las otras dos de tipo interactivo (TaskConsola). Ejecutar y graficar la simulacion usando el algoritmo FCFS para 1, 2 y 3 nucleos.

3.2 Resultados y Conclusiones

3.2.1 Ejercicio 1

Dada la simpleza del código, optamos por mostrar nuestra implementación, en vez de comentarlo detalladamente.

Realizamos un ciclo de i ¡ params[0], donde utilizamos la función dada por la catedra, uso_IO a la cual le pasamos el pid correspondiente y un entero ciclos que es el valor random obtenido entre *bmin* y *bmax*

```
ciclos = rand() % (params[2] - params[1] + 1) + params[1];
```

A continuacion, el codigo mencionado:

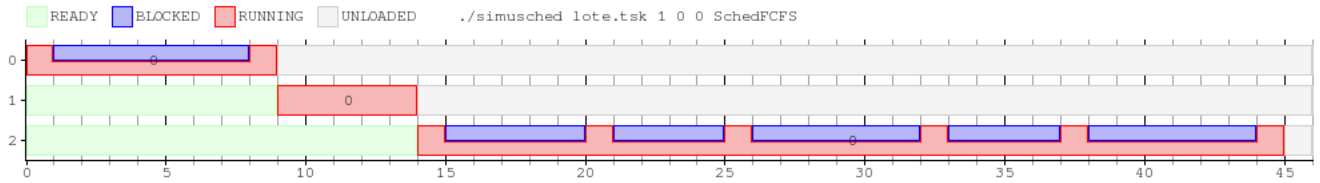
```
void TaskConsola(int pid, vector<int> params) {
     int i, ciclos;
     for (i = 0; i < params[0]; i++) {
            ciclos = rand() % (params[2] - params[1] + 1) + params[1];
            uso_IO(pid, ciclos);
     }
}
```

3.2.2 Ejercicio 2

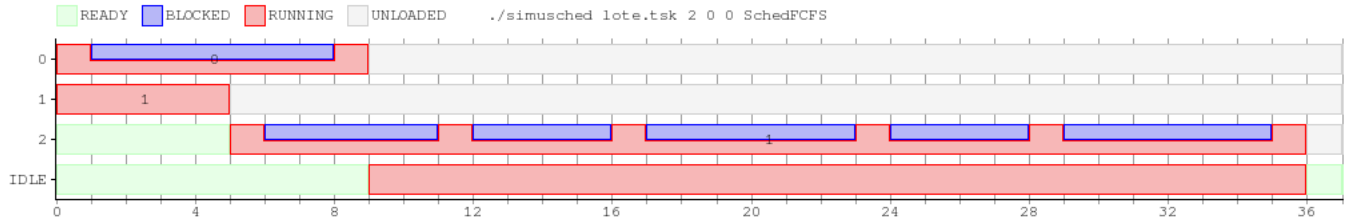
Para este punto, utilizamos el siguiente lote de tareas:

```
TaskConsola 1 4 8
TaskCPU 4
TaskConsola 5 3 6
```

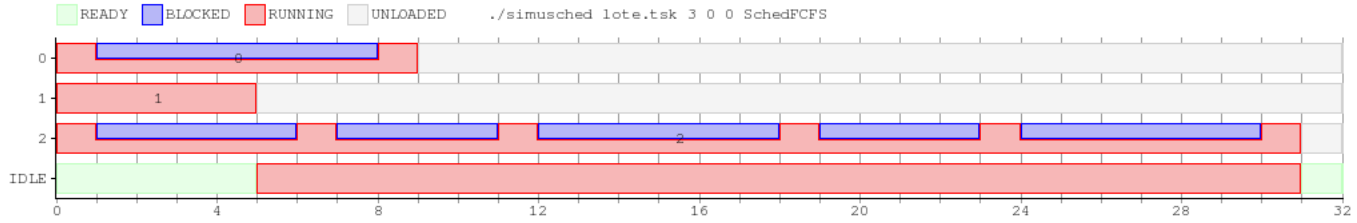
El mismo, presenta una tarea de uso intensivo $TaskCPU$ que dura unos 4 ticks, y otras dos interactivas, las cuales se bloquean 1 y 5 ticks respectivamente con una duración de entre 4 y 8 para la primera y 3 y 6 para la segunda.
A continuacion, los respectivos graficos de mediciones



*Lote*1 Scheduler FCFS - 1 core



*Lote*1 Scheduler FCFS - 2 core



*Lote*1 Scheduler FCFS - 3 core

Debido a las tareas de tipo TaskConsola, se observa que en los tres casos la duración de cada bloqueo es distinta.

A modo de análisis, se puede observar por medio de los gráficos como aumenta el paralelismo a mayor cantidad de núcleos. En este scheduler en particular esto ayuda de gran manera al rendimiento del sistema, puesto que un nucleo podrá ejecutar otra tarea recién cuando haya terminado la anterior. Las consecuencias de este comportamiento son visibles en los 3 graficos. Agregando un core más, el tiempo que se tarda en ejecutar por completo todas las tareas de reduce casi a la mitad.

4 Parte II: Extendiendo el simulador con nuevos schedulers

4.1 Ejercicios

- **Ejercicio 3** Completar la implementacion del scheduler Round-Robin implementando los metodos de la clase SchedRR en los archivos sched rr.cpp y sched rr.h. La implementacion recibe como primer parametro la cantidad de nucleos y a continuacion los valores de sus respectivos quantums. Debe utilizar una unica cola global, permitiendo ası la migracion de procesos entre nucleos.
- **Ejercicio 4** Diseñar uno o mas lotes de tareas para ejecutar con el algoritmo del ejercicio anterior. Graficar las simulaciones y comentarlas, justificando brevemente por que el comportamiento observado es efectivamente el esperable de un algoritmo Round-Robin.
- **Ejercicio 5** A partir del articulo:

  – Liu, Chung Laung, and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) 20.1 (1973): 46-61.

  1. Responda:
     1. ¿Que problema estan intentando resolver los autores?
     2. ¿Por que introducen el algoritmo de la seccion 7? ¿Que problema buscan resolver con esto?
     3. Explicar coloquialmente el significado del teorema 7.

  2. Disenar e implementar un scheduler basado en prioridades fijas y otro en prioridades dinamicas. Para eso complete las clases $SchedFixed$ y $SchedDynamic$ que se encuentran en los archivos $sched\_fixed.[h|cpp]$ y $sched\_dynamic.[h|cpp]$ respectivamente.

4.2 Resultados y Conclusiones

4.2.1 Ejercicio 3

Para desarrollar la implementacion del scheduler $Round - Robin$ y que este funcione de una forma correcta utilizamos una serie de estructuras puntuales.
Las mismas son las siguientes:

1. Una cola global, la cual nombramos q, esta contiene los PID de los procesos activos que no estan bloqueados y en el tope de la misma se encuentra el proximo proceso a correr. Esta cola, fue desarrollada para que cuando se desaloje un proceso por finalizar su *quantum* la misma pase al final de la cola y generando el ciclo acorde al comportamiento de este scheduler.
2. Un vector denominado *cores*, este tiene en su elemento i el pid correspondiente a al proceso que está corriendo en el core $i + 1$. Inicializamos todos los elementos en -1, esto corresponde a la Idle Task, de esta forma reconocemos que no se cargaron procesos en los núcleos.
3. Un vector *quantum* guarda en la posicion i el quantum que se dispuso a cada núcleo.
4. Un vector *quantumActual* aqui guardaremos la cantidad de ticks que le quedan al proceso desde que fue cargado en el core.
5. Una lista de *bloqueados* esta tendra procesos que se bloquearon cuando estaban corriendo.

De esta manera, con estas estructuras nos permiten determinar para cada tarea, cuando, y cuanto de su quantum consumio de forma que podamos desalojarla correctamente.

A su vez, tomamos ciertas decisiones en esta implementación:

- Si una tarea se encuentra bloqueada cuando se produce el tick del reloj, esta misma es desalojada de la cola global, y agregada en un lista de bloqueados. A su vez, sera reseteado el quantum, se le dara inicio a la proxima tarea que se encuentre ready y cuando el sistema operativo, nos envie una señal de unblock, la tarea desalojada regresara al final de la cola global.

4.2.2 Ejercicio 4

4.2.3 Ejercicio 5

5 Parte 3: Evaluando los algoritmos de scheduling

5.1 Ejercicios

- **Ejercicio 6** Programar un tipo de tarea TaskBatch que reciba dos parametros: total cpu y cant bloqueos. Una tarea de este tipo debera realizar cant bloqueos llamadas bloqueantes, en momentos elegidos pseudoaleatoriamente. En cada tal ocasion, la tarea debera permanecer bloqueada durante exactamente un (1) ciclo de reloj. El tiempo de CPU total que utilice una tarea TaskBatch debera ser de total cpu ciclos de reloj (incluyendo el tiempo utilizado para lanzar las llamadas bloqueantes; no ası el tiempo en que la tarea permanezca bloqueada).

- **Ejercicio 7** Elegir al menos dos metricas diferentes, definirlas y explicar la semantica de su definicion. Diseñar un lote de tareas TaskBatch, todas ellas con igual uso de CPU, pero con diversas cantidades de bloqueos. Simular este lote utilizando el algoritmo SchedRR y una variedad apropiada de valores de quantum. Mantener fijo en un (1) ciclo de reloj el costo de cambio de contexto y dos (2) ciclos el de migracion. Deben variar la cantidad de nucleos de procesamiento. Para cada una de las metricas elegidas, concluir cual es el valor optimo de quantum a los efectos de dicha metrica.

- **Ejercicio 8** Implemente un scheduler Round-Robin que no permita la migracion de procesos entre nucleos (SchedRR2). La asignacion de CPU se debe realizar en el momento en que se produce la carga de un proceso (load). El nucleo correspondiente a un nuevo proceso sera aquel con menor cantidad de procesos activos totales (RUNNING + BLOCKED + READY). Diseñe y realice un conjunto de experimentos que permita evaluar comparativamente las dos implementaciones de Round-Robin.

- **Ejercicio 9** Disenar un lote de tareas cuyo scheduling no sea factible para el algoritmo de prioridades fijas pero sı para el algoritmo de prioridades dinamicas.

- **Ejercicio 10** Disenar un lote de tareas, cuyo scheduling sı sea factible con el algoritmo de prioridades fijas, donde se observe un mejor uso del CPU por parte del algoritmo de prioridades dinamicas.

5.2 Resultados y Conclusiones

5.2.1 Ejercicio 6

Al igual que con la tarea TaskConsola, mencionaremos nuestro implementación y acontinuación de la misma explicaremos ciertos puntos de la misma.

```
void TaskBatch(int pid, vector<int> params) {
     int total_cpu = params[0];
     int cant_bloqueos = params[1];
     srand(time(NULL));
     vector<bool> uso = vector<bool>(total_cpu);
     for(int i=0;i<(int)uso.size();i++)
        uso[i] = false;
for(int i=0;i<cant_bloqueos;i++) {
            int j = rand()%(uso.size());
            if(!uso[j])
               uso[j] = true;
            else
               i--;
            }
     for(int i=0;i<(int)uso.size();i++) {
         if( uso[i] )
             uso_IO(pid,1);
         else
             uso_CPU(pid, 1);
        }
```

```
}
```

Para este tipo de tarea, creamos un vector de tamaño igual a $total_cpu$ el cual tendra bool, ya sea true o false dependiendo del uso que se le de dentro de la tarea, ya sea uso_IO o uso_CPU. En caso de ser uso_IO sera true, y sino false.
Luego, utilizaremos un ciclo que ira desde 0 hasta el tamaño del vector y dependiendo el valor booleano, usará la funciones dadas por la catedra uso_IO o uso_CPU.

5.2.2 Ejercicio 7

Las métricas elegidas fueron:

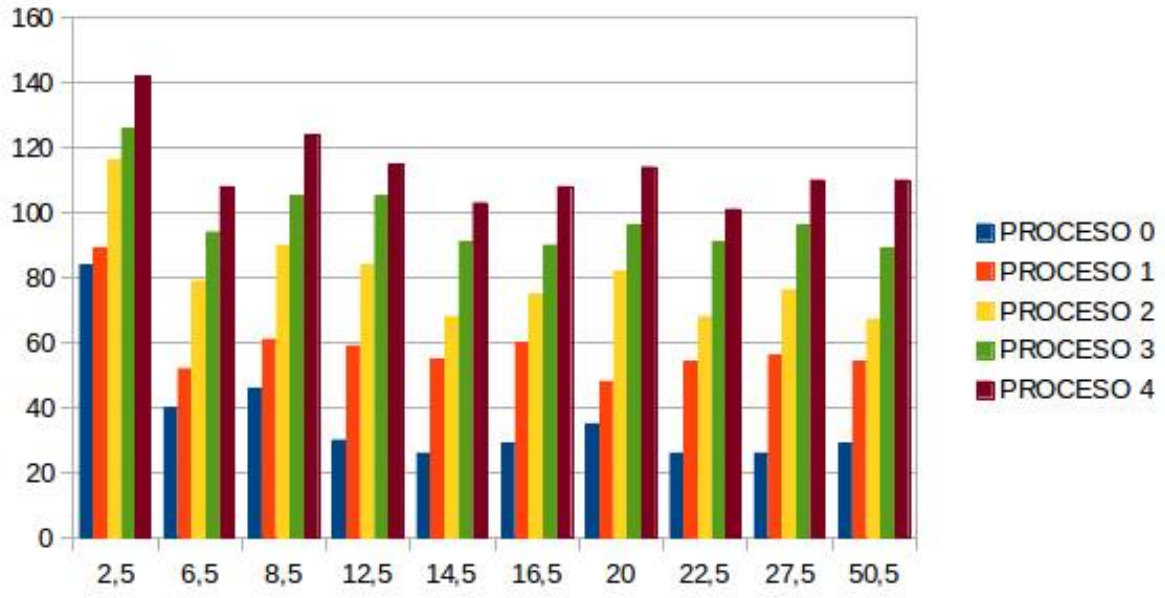
- **Turnaround**: Es el intervalo de tiempo desde que un proceso es cargado hasta que este finaliza su ejecución.
- **Waiting Time**: Es la suma de los intervalos de tiempo que un proceso estuvo en la cola de procesos *ready*.

Como las tareas TaskBatch se bloquean pseudoaleatoriamente, para obtener datos relevantes tomamos un promedio de las mediciones.

A la hora de encarar la experimentación, lo que realizamos fue simular corridas con varios quantum para poder obtener una aproximación del efecto del *quantum* en la ejecución del lote de tareas.

De esta aproximación, se confeccionaron gráficos de turnaround time en función del quantum, referente al estudio con 2 y 3 núcleos, los cuales se proveerán a continuación:

Luego de realizar mediciones con distintos *quantum*, tomamos la decisión de trabajar con los mismos *quantum* para cada nucleo al trabajar con más de 1 core. (Se muestra acontinuación un gráfico para demostrar que no era una decisión acertada trabajar con disversos *quantum* para cada core).



Turnaround - 2 core - Prueba Quantum distintos por Core
$EjeX = Quantum$;
$EjeY = Tiempo$

Al realizar, este gráfico y varias mediciones con distintos *quantum* por core, nos dimos cuenta que no terminaban siendo mediciones rigurosas, ya que una tarea podia estar corriendo con distintos tiempos por la migracion de procesos por core.

Por ende, optamos por realizar mediciones con igualdad de *quantum* por core para obtener la mejor

performance posible.

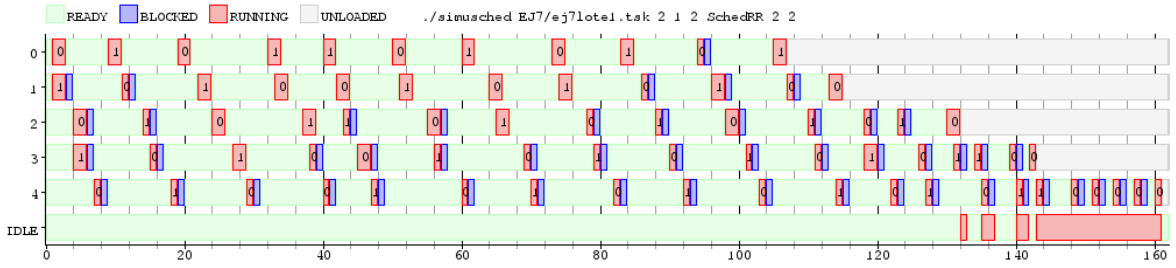
Por consiguiente, al trabajar con las nuevas mediciones, conjeturamos las siguientes hipótesis:

- Con 2 nucleos las mediciones de tiempo tienden a estabilizarse a partir de un *quantum* igual a 9.
- Con 3 nucleos las mediciones de tiempo tienden a estabilizarse a partir de un *quantum* igual a 11.

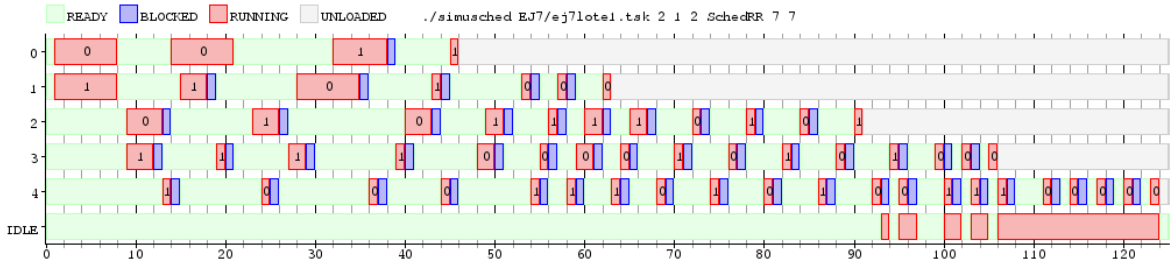
Turnaround Time

2 Core

A continuación se muestran los Diagramas de Gantt más relevantes:

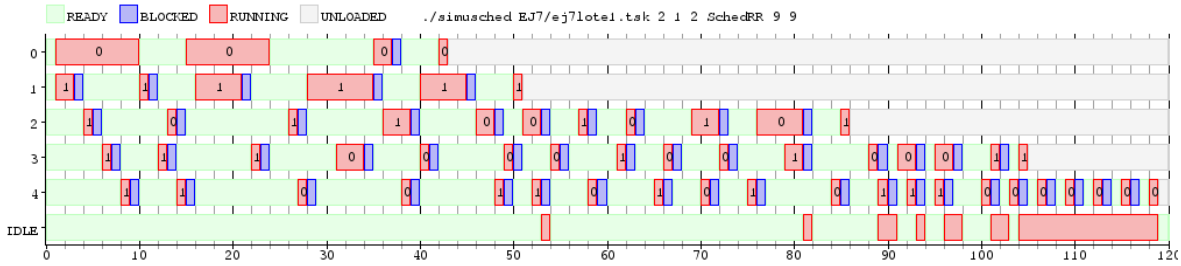


*Lote*1 - Turnaround - 2 core - Quantum igual a 2



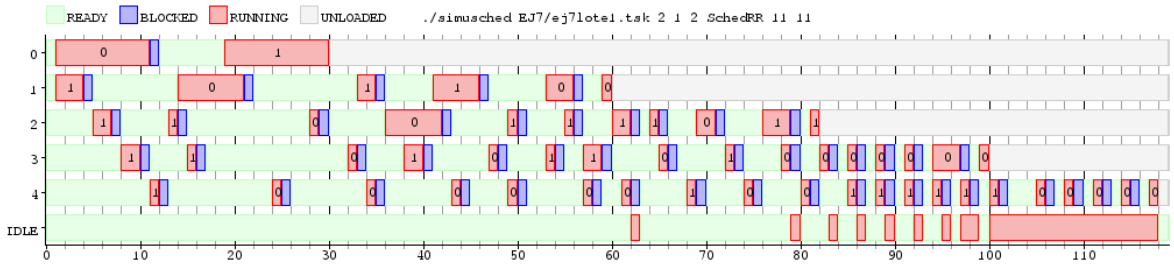
*Lote*1 - Turnaround - 2 core - Quantum igual a 7

La performance empieza a mejorar a medida que el *quantum* aumenta.

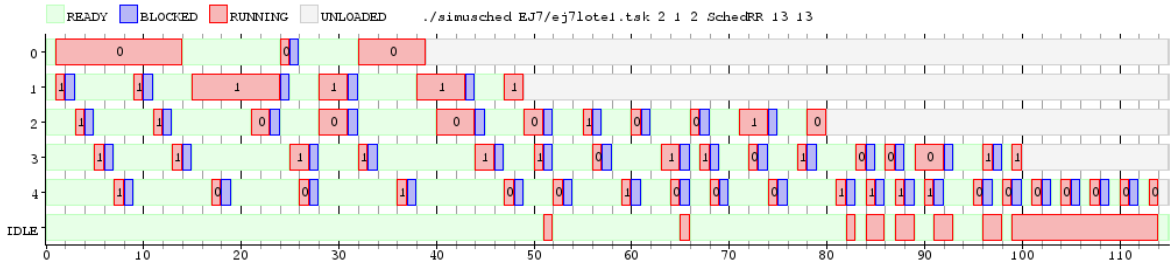


*Lote*1 - Turnaround - 2 core - Quantum igual a 9

La performance sigue mejorando,a partir de este valor, el desempeño comienza a estabilizarse, como muestran los siguiente dos gráficos. Las pequeñas diferencias en los valores responden a la pseudoaleatoridad de las tareas TaskBatch.

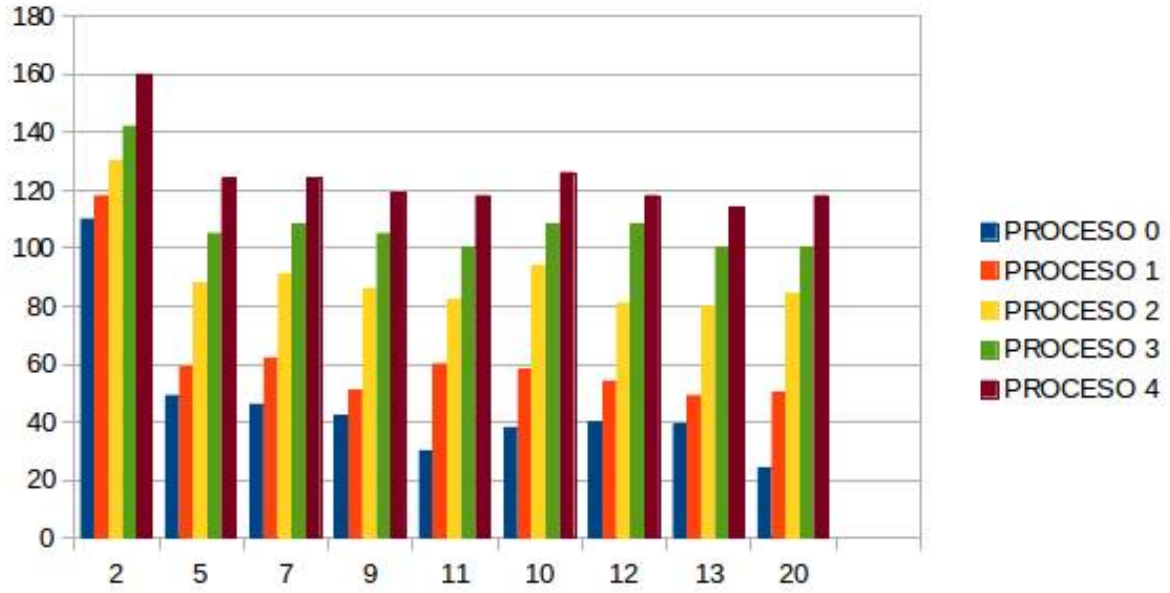


*Lote*1 - Turnaround - 2 core - Quantum igual a 11



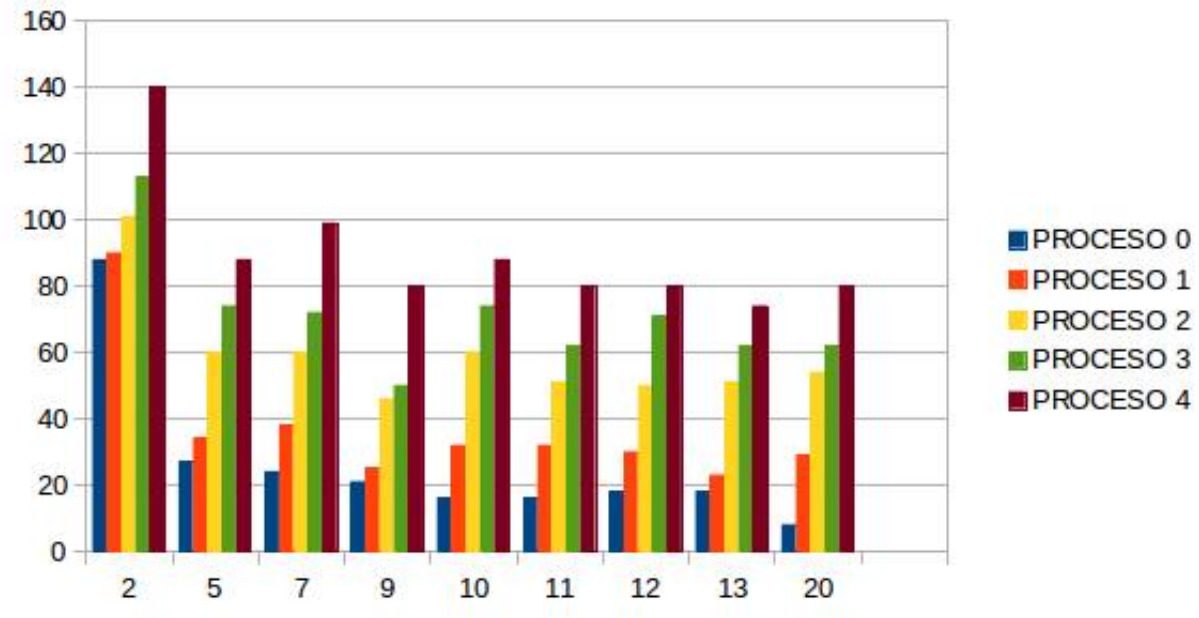
*Lote*1 - Turnaround - 2 core - Quantum igual a 13

Como mencionamos, las mediciones tienden a estabilizarse con un *quantum* igual a 9, pero la mejor performance obtenida es con un *quantum* igual a 13, teniendo en cuenta la pseudoaleatoridad del tipo de tarea utilizada. Se puede observar en la primer figura que a pesar de trabajar con 2 cores, al tener un quantum bajo (igual a 2) el costo es alto.



Turnaround - 2 core
$EjeX = Quantum$
$EjeY = Tiempo$

Waiting Time

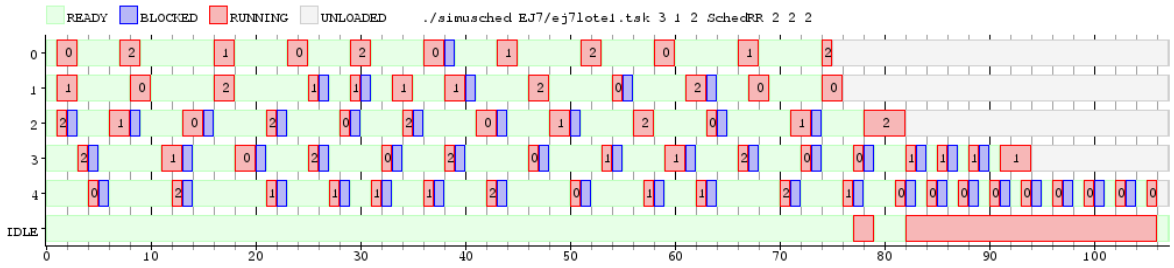


Waiting Time - 2 core
$EjeX = Quantum$
$EjeY = Tiempo$

Con este tipo de metrica, se comienza a estabilizar a partir del *quantum* igual a 5, obteniendo su mejor performance con el *quantum* igual a 13.

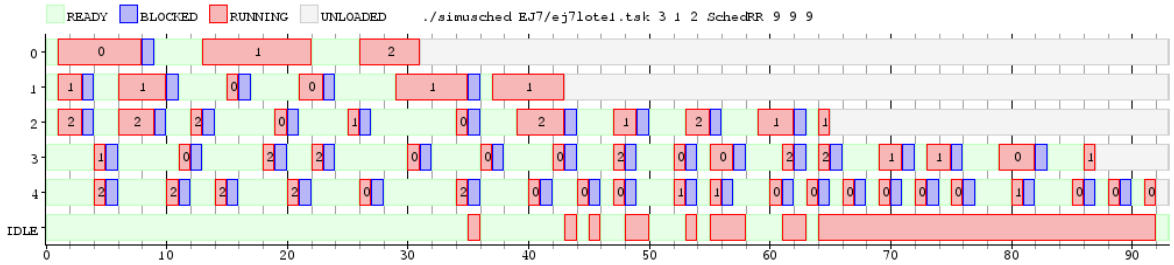
3 Core

A continuación, al igual que con 2 cores, mostraremos los Diagramas de Gantt mas relevantes:



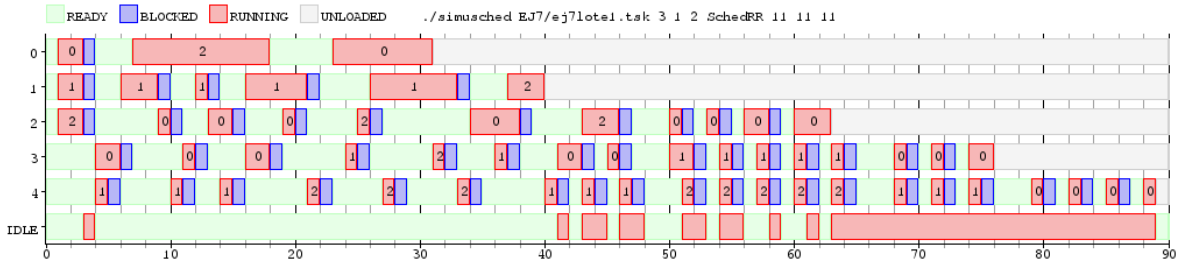
*Lote*1 - Turnaround - 3 core - Quantum igual a 2

Se observa que al tener otro core mas,a diferencia de con 2, a pesar de estar con un *quantum* bajo, la performance mejora bastante.

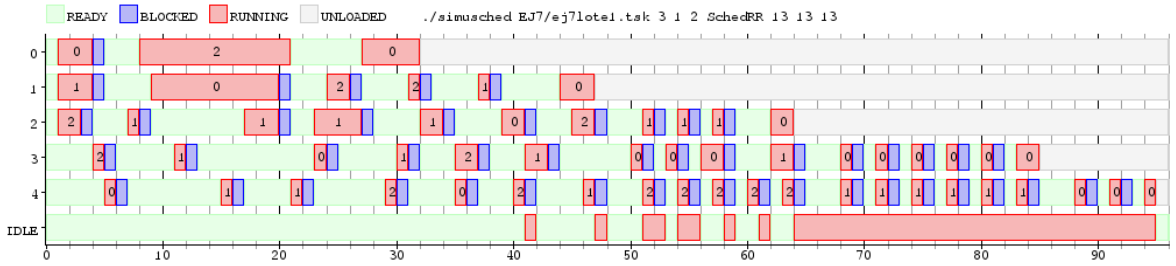


*Lote*1 - Turnaround - 3 core - Quantum igual a 9

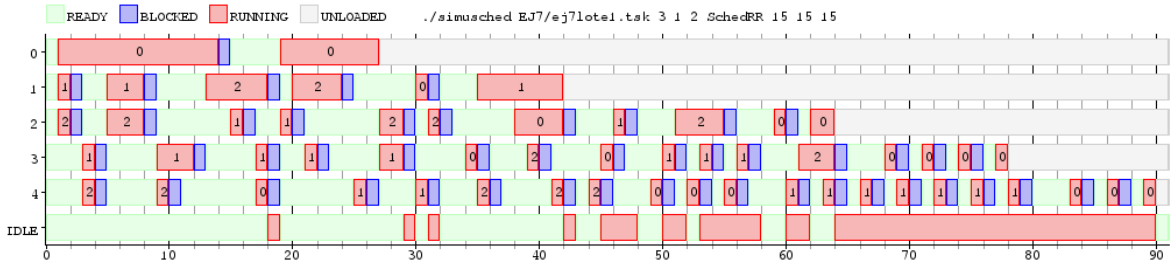
La performance empieza a mejorar a medida que el *quantum* aumenta.



*Lote*1 - Turnaround - 3 core - Quantum igual a 11

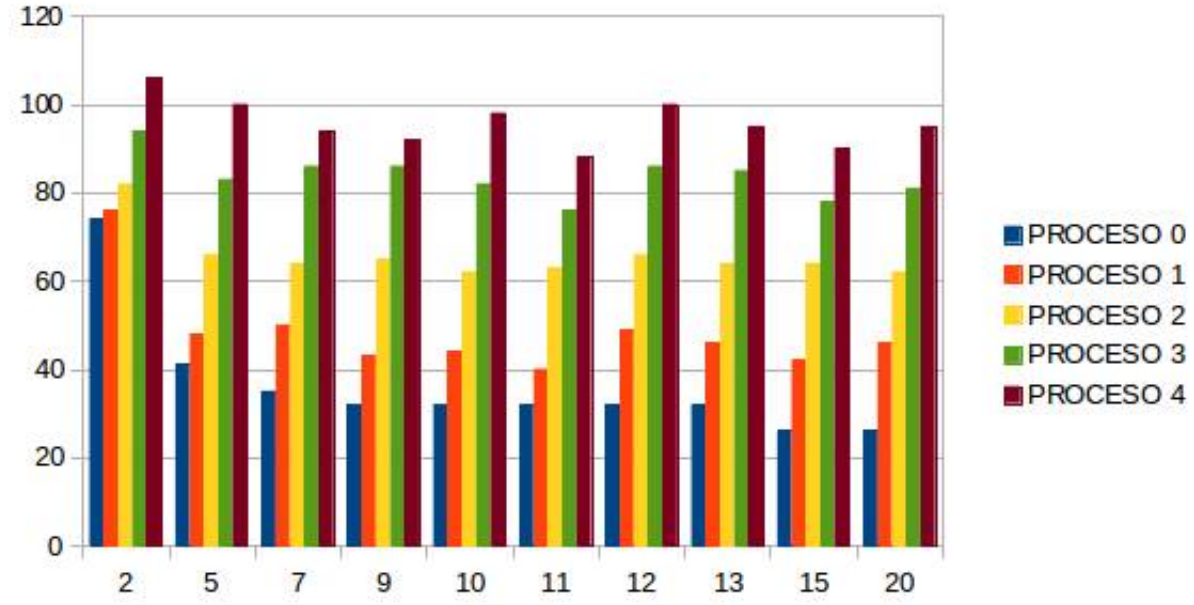


*Lote*1 - Turnaround - 3 core - Quantum igual a 13



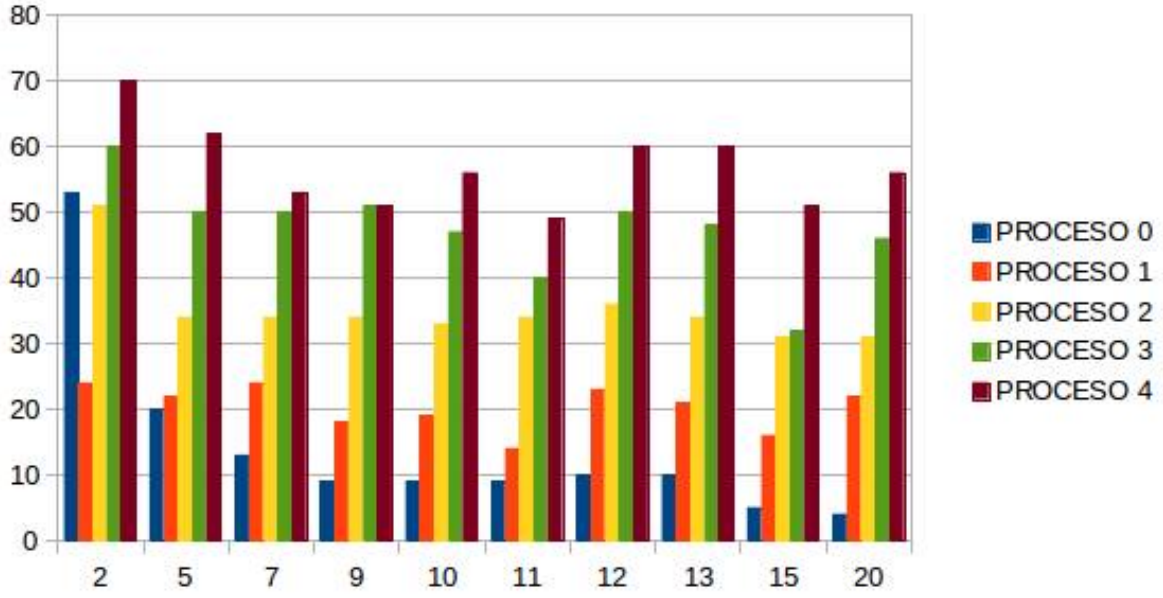
*Lote*1 - Turnaround - 3 core - Quantum igual a 15

A diferencia que en nuestra hipótesis conjeturada para con dos cores, con un *quantum* igual a 11 se obtiene la mejor performance, pero teniendo en cuenta la pseudoaleatoridad del tipo de tarea con la que se trabajo, a partir del *quantum* igual a 9 ya se estabiliza notoriamente.



Turnaround - 3 core
$EjeX = Quantum$
$EjeY = Tiempo$

Waiting Time



Waiting Time - 3 core
$EjeX = Quantum$
$EjeY = Tiempo$

Con este tipo de metrica,se obteniene a diferencia de con Turnaround su mejor performance con el *quantum* igual a 15.

Conclusiones

La diferencia entre los valores de quantum entre los casos se puede atribuir a que cada vez que agregamos un núcleo aumentamos la posibilidad de una migración de la tareas.

En todos lo casos se observa la influencia negativa que proviene de elegir un quantum con valores pequeños.

Agregar núcleos de procesamiento mejora significativamente la performance de acuerdo a la métricas con las que trabajamos, al permitir más procesamiento en pararelo y disminuyendo los waiting time de las tareas.

Fijada una cantidad de núcleos, aumentar el valor del quantum también mejora la performance, especialmente los tiempos referidos a las tareas que menos cantidad de bloqueos tienen. Igualmente, a partir de cierto valor de quantum, las mejoras en la performance dejan de ser muy significativas. Esto se produce a que las tareas con mas cantidad de bloqueos en algun momento dejan de consumir todo su quantum si seguimos aumentando el valor.

5.2.3 Ejercicio 8

La idea principal de esta nueva version de $Round - Robin$ se centraliza en que no permita migracion entre cores, esto se basa principalmente en utilizar una cola para cada nucleo por separado, y en cada cola respectiva se encolaran las tareas que fueron asignadas inicialmente a cada nucleo.
Para desarrollar este tipo de algoritmo, el cual denominaremos $RR2$, utilizamos estructuras puntuales, enunciadas a continuacion:

- Un vector *quantum* y otro *quantumActual*, los cuales siguen cumpliendo la misma funcion que en Round-Robin 1.
- Un vector de colas denominado *colas*, en el cual la posicion i encontraremos la cola correspondiente a ese nucleo de procesamiento.
- Un diccionario de *Bloqueados*, donde la clave contendra el numero de core, y en definicion la tareas bloqueadas de ese core. Esto nos beneficiara cuando haya que reubicarla en la cola de procesos ready.
- Un vector de enteros *cantidad*, que como la palabra lo define, tendra en cada posicion i la totalidad de las tareas, ya sea bloqueadas, activas o en estado ready que tiene asignado ese core, beneficiandonos la determinacion del nucleo al que se asignara la tarea al momento de cargarla.

Cuando se carga una tarea, previamente, se chequeara que core tiene menor cantidad de procesos totales asignados (aqui es donde el vector *cantidad* entra en juego). Una vez que se obtiene este nucleo, se agrega la tarea a la cola correspondiente y se actualiza la cantidad sumando una unidad.

Al bloquearse un proceso, se define una nueva entrada en el diccionario *bloqueados* con el pid y el nucleo correspondiente. De esta forma, al desbloquearse, colocamos la tarea en la cola del core correspondiente y eliminamos la entrada del diccionario. Así logramos resolver el inconveniente de la nula migracion entre nucleos.

Finalmente, cuando una tarea finaliza, la quitamos y descontamos una unidad a la posicion i del vector *cantidad*. Esta es la unica vez, en la cual se descuenta. Aunque una tarea se bloquee, la misma seguira contando en el vector. De esta forma se cumplira, que las tareas son asignadas a los cores con menor cantidad de tareas.
Luego de realizar dicha implementacion, en comparacion al Round-Robin original, hemos conjeturado las siguientes hipotesis:

1. Dados un mismo lote de tareas y una misma configuracion del scheduler (mismos costo en cambio de contexto y quantum) un unico nucleo de procesamiento, ambos algoritmos deben comportarse de la misma manera.
2. Comportamiento menos eficiente en el RR2 con respecto al paralelismo, ya que al no permitir migracion de nucleos este lo pierde.
3. Comportamiento mas eficiente en el RR2 con lotes de tareas que se bloquean un gran numero de veces. Esto surge ya que el Round-Robin original, es mas proclive a realizar cambios de contexto con la posibilidad de darse un cambio de core.

5.2.4 Ejercicio 9

5.2.5 Ejercicio 10

6 Bibliografía

- Cátedra de Sistemas Operativos - Clases teóricas y prácticas (2º Cuatrimestre 2014)
- Facultad de Ingenieria Uruguay
  (*https* : *//eva.fing.edu.uy/pluginfile.php/*75120*/mod_resource/content/*1*/*6*−SO−Teo−Planificacion.pdf*)
- Operating Systems Concepts, Abraham Silberschatz & Peter B. Galvin