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

Sistemas Operativos - Primer cuatrimestre 2013
Trabajo Práctico Número 1
Scheduling

L.U.	Integrante	E-Mail
125/10	Giordano, Mauro	mauro.foxh@gmail.com
039/10	Tastzian, Juan Manuel	jm@tast.com.ar
500/10	Vallejo, Nicolás Agustín	nicopr08@gmail.com

Índice

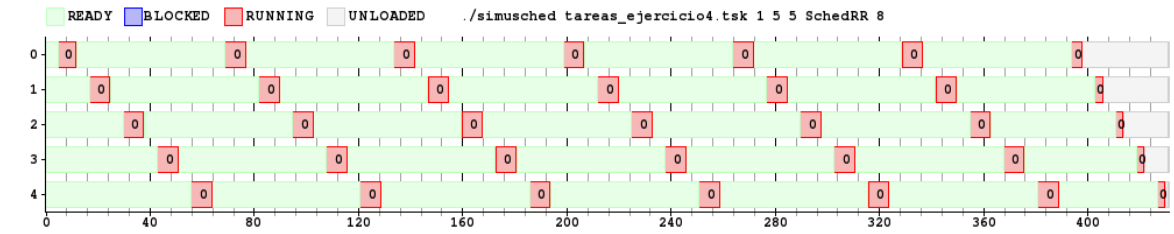
1. Ejercicio 2

A continuación mostramos 3 corridas del algoritmo *First-come, First-served*, con una simulación de 1, 2 y 3 cores respectivamente:

2. Ejercicio 4

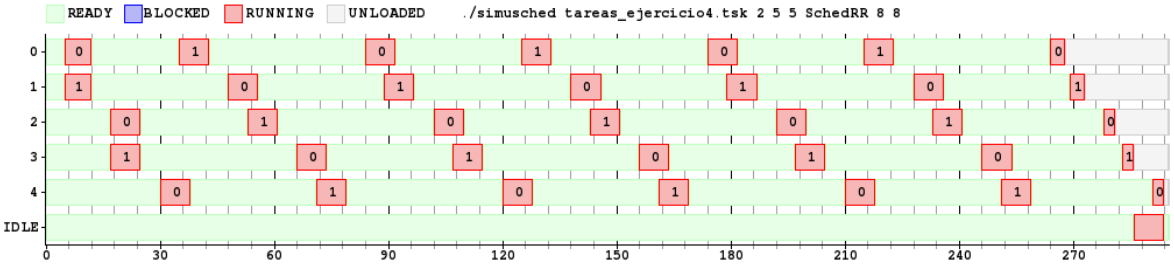
A continuación mostramos algunas corridas del algoritmo *Round Robin*. Todas fueron ejecutadas con un lote de 5 tareas TaskCPU de 50 ticks de duración definidas en *tareas_ejercicio4.tsk*. En todas las corridas, tanto el costo de cambio de contexto como el costo de cambio de núcleo de procesamiento son iguales, para facilitar la lectura de los gráficos.

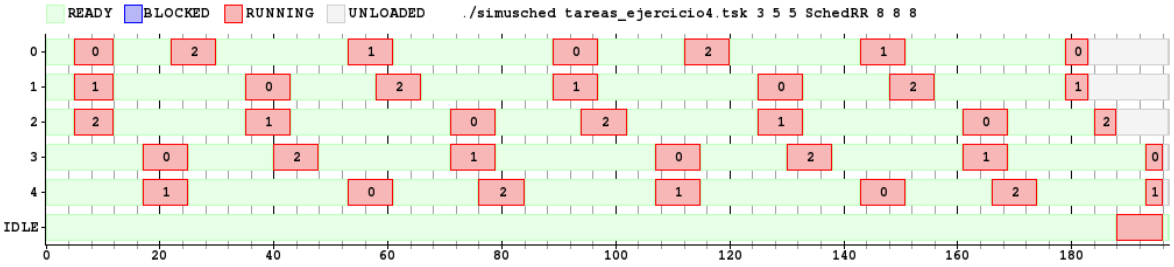
2.1. Single-core



En esta corrida del lote se puede ver claramente el comportamiento del algoritmo *Round Robin*, dado que con un quantum de 8 ticks para cada tarea, se ejecuta en el único core de prueba, alguna tarea durante ese tiempo hasta que terminan todas.

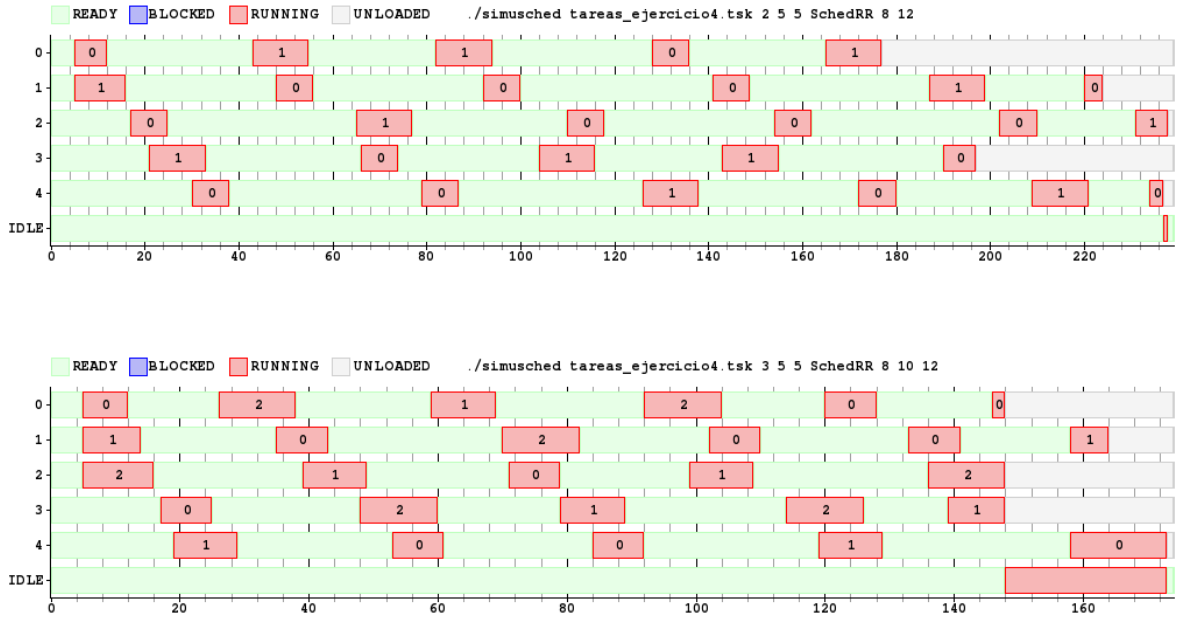
2.2. Multi-core, igual quantum





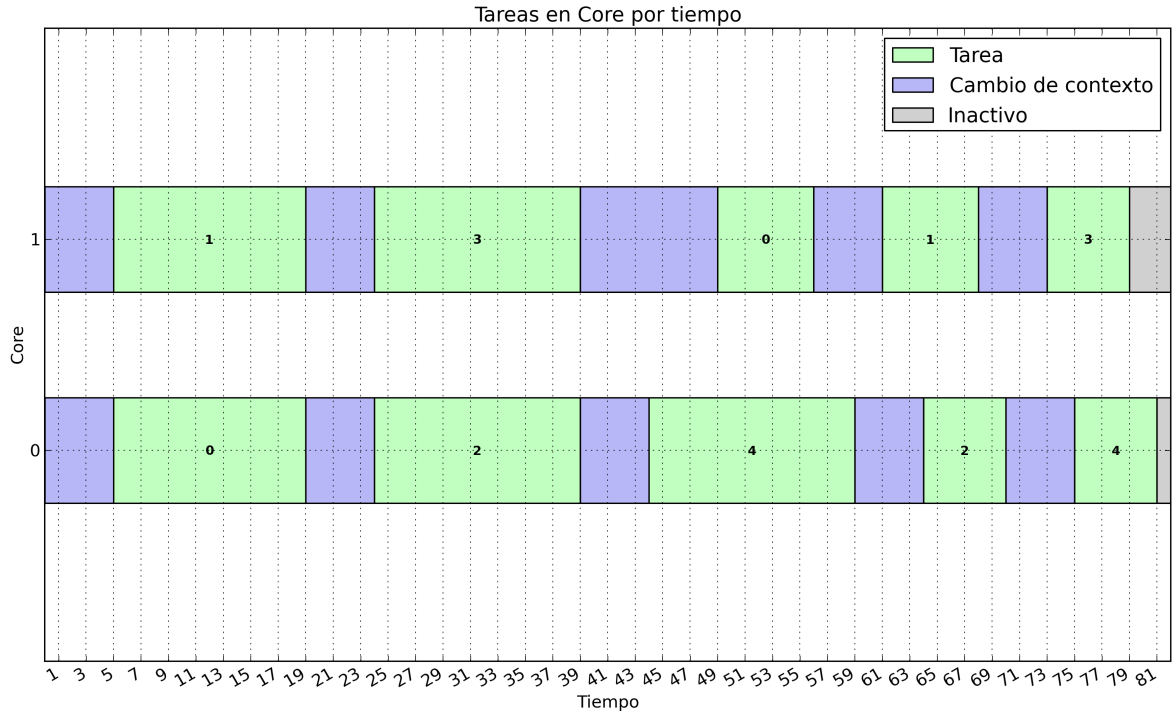
En estas corridas del lote se ve como se ejecutan a la vez 2 y 3 tareas durante un quantum fijo e igual para todos los núcleos. Es fácil ver como durante los primeros 2 quantums se mantiene visible el comportamiento del algoritmo. Más adelante se hace un poco más difícil de ver, pero nunca deja de apreciarse la manera secuencial característica de asignación de CPU de este algoritmo.

2.3. Multi-core, distinto quantum



Estas corridas son similares a las multi-core con igual quantum. La única diferencia es el tiempo que permanece cada tarea en ejecución en alguno de los CPUs. Una misma tarea puede utilizar distinta cantidad de quantum según el core en el que se esté ejecutando. Nuevamente, se ve el comportamiento cíclico del scheduling de los procesos del algoritmo de *Round Robin*.

2.4. Distintas perspectivas de un mismo lote



Estos 2 gráficos son dos perspectivas diferentes de una misma corrida de un lote con 2 cores. Lo que se puede ver en el primer gráfico, es la forma en la que *Round Robin* delega las tareas de manera iterativa como vimos anteriormente. En el segundo se ve la carga de cada core en un momento dado. Al ponerle el mismo quantum a ambos cores, es más fácil de ver el accionar del algoritmo.

Lo que se puede mencionar como detalle presente en estos 2 gráficos, es la forma de elegir los cores que tiene el algoritmo al finalizar su quantum por primera vez la tarea 4. Se ve claramente que, por más que al momento de ejecutarse la tarea 1 por segunda vez, el core 0 esté libre, al momento de terminar de ejecutar la tarea 0, el mismo todavía estaba siendo ocupado por la tarea 4. Por este motivo, el scheduler elige nuevamente el core 1 para trabajar, y continúa la ejecución normalmente.

3. Ejercicio 7

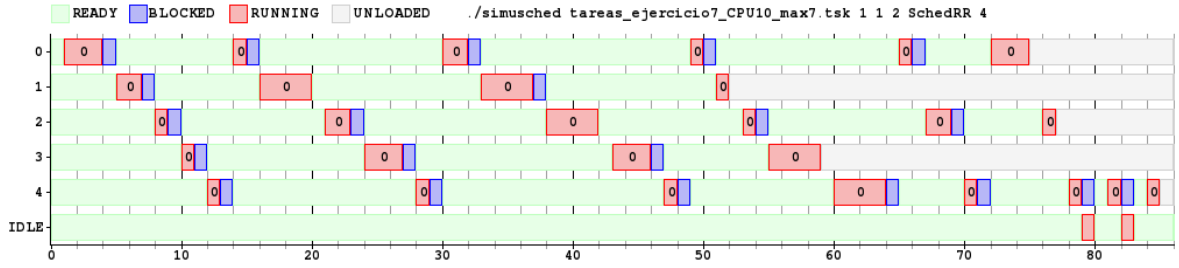
Para la resolución de este ejercicio debimos elegir al menos 2 métricas para evaluar la performance de nuestro algoritmo *Round Robin*. A continuación, nuestra experimentación y sus resultados.

3.1. CPU Utilization

Para evaluar la performance de nuestro algoritmo con la métrica de utilización del CPU hicimos tanto pruebas single-core como pruebas multi-core, pero lo que queremos mostrar se ve de manera más clara en las pruebas single-core. Los experimentos fueron los siguientes:

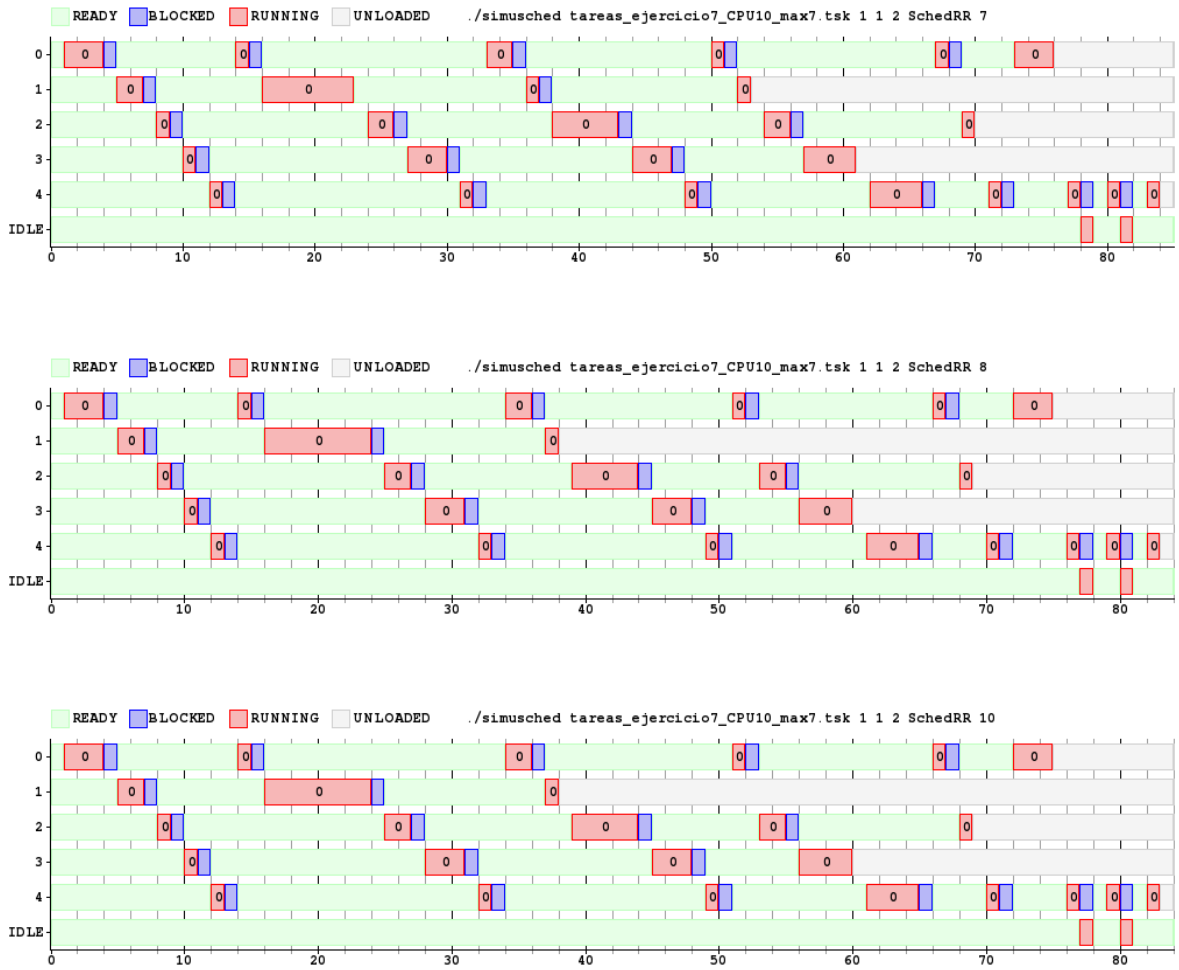
En la primera prueba, con un quantum de 2 ticks para cada core, podemos ver que la utilización del único CPU disponible es durante un tiempo considerable (poco más de 83 %, siendo lo normal entre 40 y 90 % en un escenario real[1]).

Aumentando levemente el quantum a 3 ticks por turno, se ve que el *turnaround time*, del cual hablaremos en breve, disminuye, y también el uso del CPU mejora, teniendo menos momentos de CPU en estado ocioso (86∼87 % de tiempo de CPU ocupado).



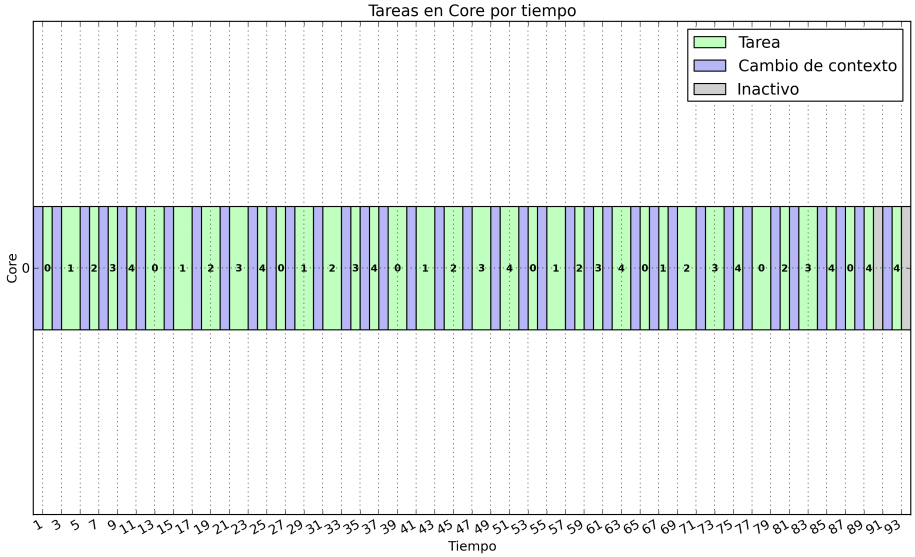
Con un quantum de 4 ticks, el rendimiento del CPU llega a un 90∼91 %, disminuyendo otra vez el *turnaround time*, pero aumentando el quantum todavía se puede mejorar, como veremos a continuación.

[1] Operating System Concepts - 7th Edition - página 157

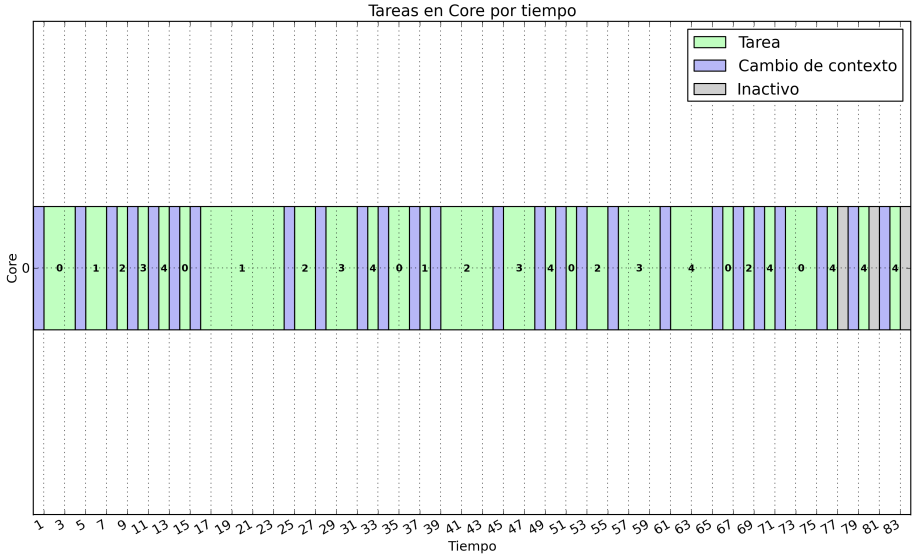


Hasta un quantum de 7 ticks inclusive, la tarea 1 debe consumir al menos 4 quantums para completarse. A partir de los 8 ticks, en 3 quantums la misma completa su ejecución y termina, disminuyendo drásticamente su *turnaround time*, dado que su *waiting time* se recorta en un turno completo. En el sistema en general, el aumentar el quantum más allá de los 8 ticks por turno ya no mejora la performance del mismo.

Perspectiva del core: quantum = 2:



Perspectiva del core: quantum = 10:



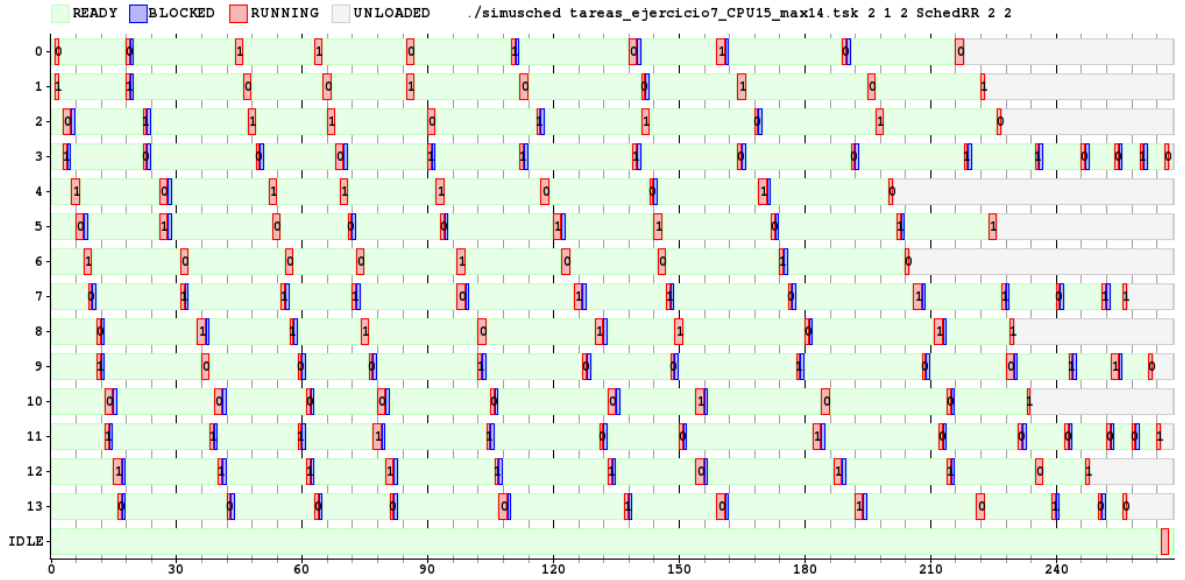
Para terminar el análisis de este criterio, podemos concluir que el quantum óptimo para este experimento es 8 ticks, y como comentario agregado, podemos decir que se nota la importancia de una buena elección del quantum, viendo que entre el uso de CPU con quantum 2 y el uso con quantum 8, hay una mejora de un 10.63 % aproximadamente.

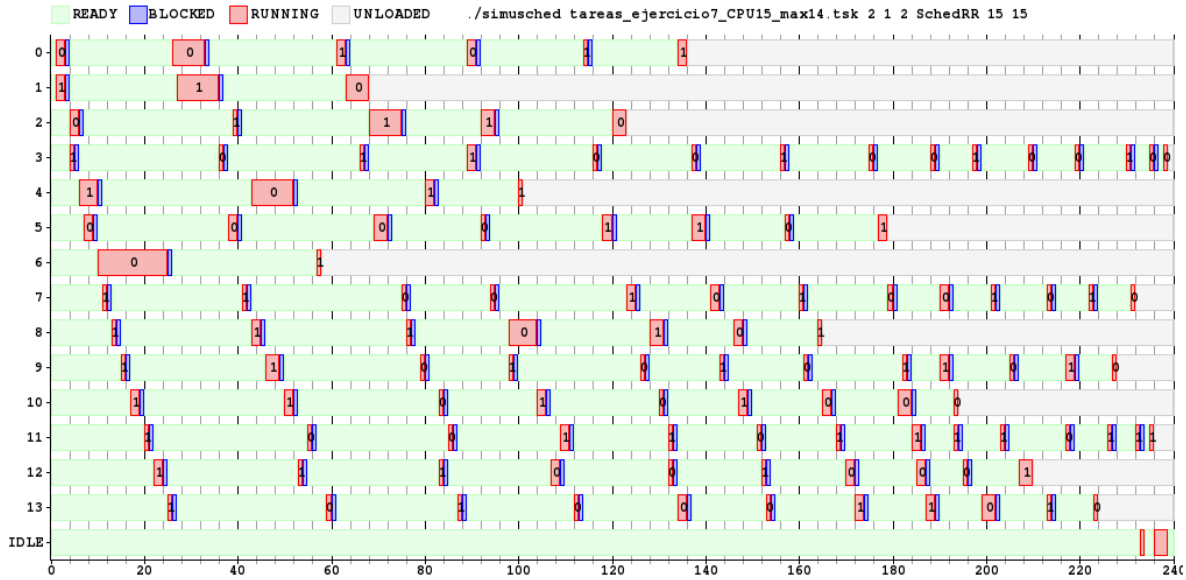
3.2. Turnaround time y Waiting time

Decidimos evaluar el desempeño del algoritmo con los criterios de *turnaround time* y *waiting time* de manera conjunta, ya que creemos que en este tipo de experimentos están íntimamente ligados, al menos en este escenario de prueba. Esto es porque las tareas no son interactivas, en cuyo caso habría que ver también (y por separado) el *response time*, que se encarga de evaluar el tiempo neto utilizado por el

CPU una vez que se terminó la interacción con el usuario[2], que en general, es órdenes de magnitud más lento que lo que vimos hasta ahora.

2 cores:



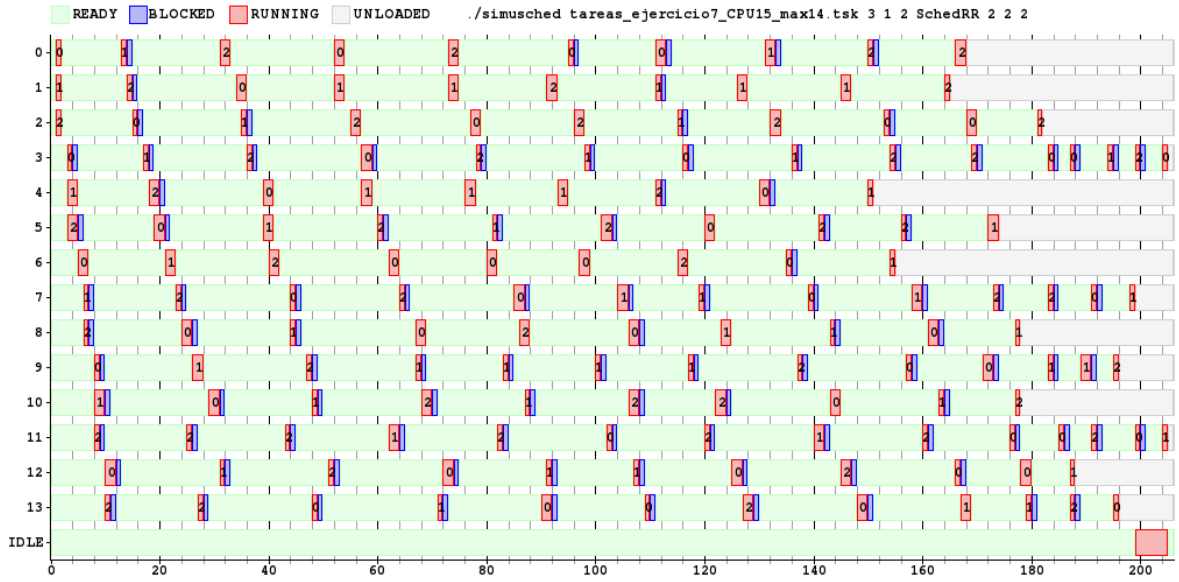


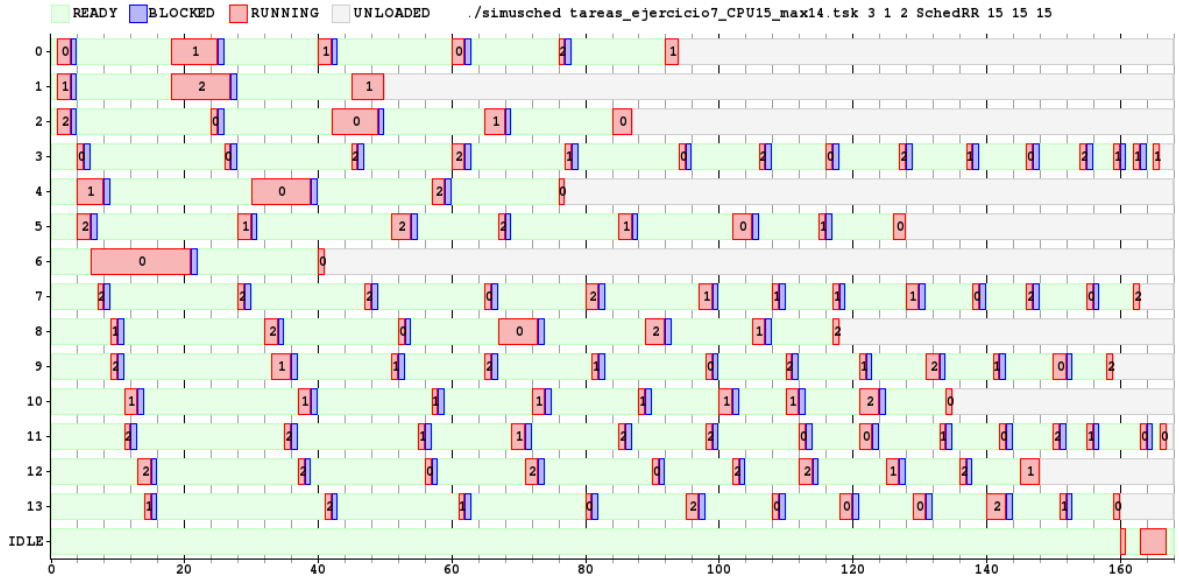
En el ejemplo con 2 núcleos se puede ver una mejora sustancial entre un quantum pequeño (2 ticks) y que corta muchas veces la ejecución de cada tarea, contra el quantum óptimo (15 ticks) que interrumpe menos veces la ejecución. Dicha mejora es de alrededor de 24 ticks, o aproximadamente un 10 % menos en el *turnaround time*.

Además se puede apreciar como en la mayoría de las tareas, el *waiting time* es drásticamente reducido gracias al aumento del quantum por turno, teniendo hasta 7 turnos menos que ejecutar en casos como la tarea 1 o la 6, reduciendo el *waiting time* de las tareas de manera altamente sustancial, hasta más de 120 ticks en algunos casos como por ejemplo en la tarea 6.

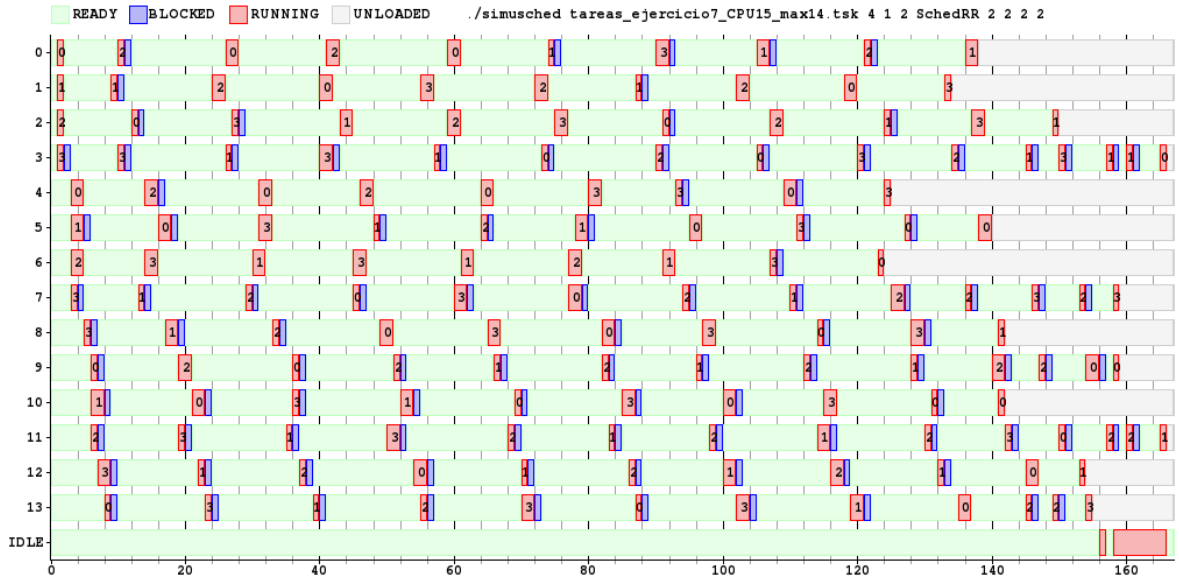
[2]Operating System Concepts - 7th Edition - página 157, 158

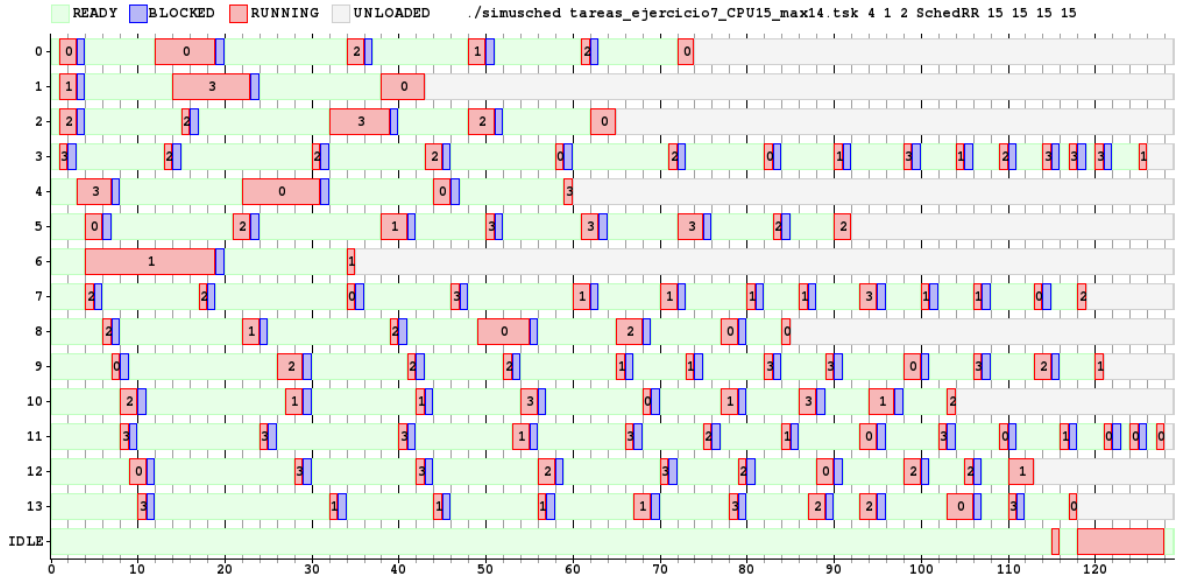
3 cores:


READY BLOCKED RUNNING UNLOADED ./simusched tareas_ejercicio7_CPU15_max14.tsk 3 1 2 SchedRR 2 2 2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
IDLE
0
20
40
60
80
100
120
140
160
180
200


READY BLOCKED RUNNING UNLOADED ./simusched tareas_ejercicio7_CPU15_max14.tsk 3 1 2 SchedRR 15 15 15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
IDLE
0
20
40
60
80
100
120
140
160

4 cores:





Estos comportamientos se ven reflejados de manera similar en los casos con más cores (en este caso mostramos los casos de 3 y 4 cores), siendo la disminución del *waiting time* proporcional a la disminución del *turnaround time* total, gracias al aumento de *throughput* (cantidad de tareas finalizadas en una unidad de tiempo[3]) debido al agregado de cores de trabajo.

[3]Operating System Concepts - 7th Edition - página 157

4. Ejercicio 8

Acá va el ejercicio 8

5. Ejercicio 9

Acá va el ejercicio 9