Sistemas Operativos

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

Trabajo Práctico 1 - Cachalotes

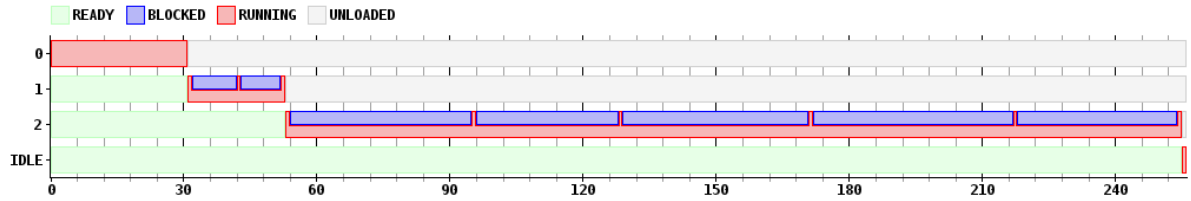
Integrante	LU	Correo electrónico
Ortiz de Zarate, Juan Manuel	403/10	jmanuoz@gmail.com
Kujawski, Kevin	459/10	kevinkuja@gmail.com
Carreiro, Martin	45/10	carreiromartin@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Ejercicio 2



En el diagrama se puede observar como el scheduler ejecuta las tareas en el orden en que entraron. Esto es debido al tipo de scheduler que utilizamos para correr el lote de tareas, este scheduler (FIFO / FCFS) además de ejecutar las tareas en el orden de entrada, no cambia de tarea hasta que la que está ejecutando termine.

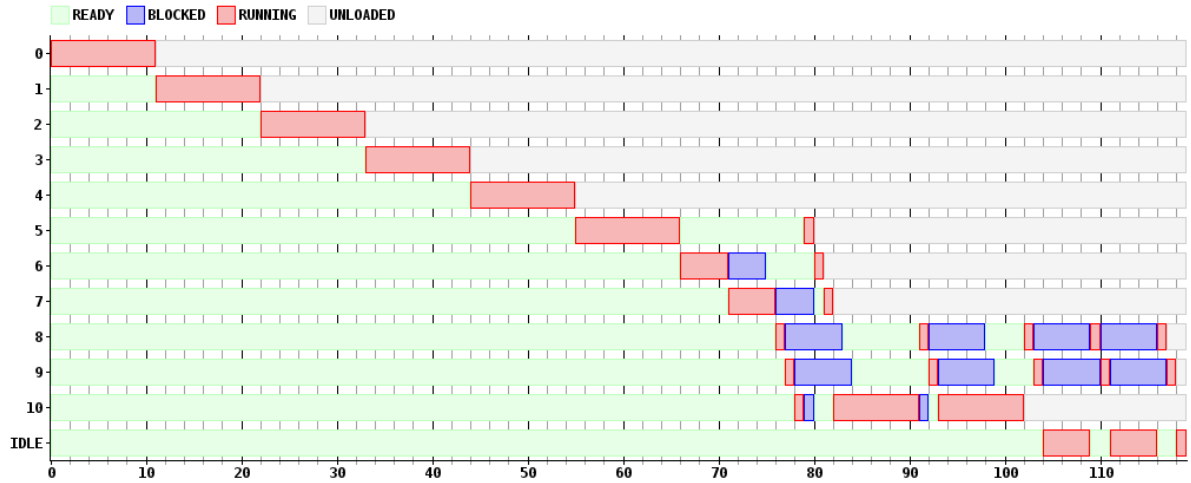
Por esto se puede observar que la primera tarea realiza 30 ciclos de reloj y recién después pasa a la próxima tarea. Esto es porque la primera tarea que se ejecuta es "TaskCPU 30"la cual realiza exactamente 30 ciclos de reloj. Lo mismo sucede con las otras 2 tareas, se ejecutan por completo y no hace el switch hasta finalizar la tarea acutal.

También en el gráfico se puede observar que el tiempo de bloqueo no es siempre el mismo pero que siempre se encuentra dentro de los rangos enviados (establecidos en el lote de tareas).

Finalmente vale destacar que las tareas interactivas cumplen lo pedido ya que la primera tarea ejecuta 2 bloqueos de aproximadenete 10 ciclos y la segunda 5 bloqueos de aproximadamente 30 ciclos cada uno. Lo que se ajusta los parametros de entrada ya que la primera tarea tiene un rango de duracion por ciclo de entre 1 y 10 y la segunda entre 20 y 50.

2. Ejercicio 4

En el siguiente gráfico se puede observar el diagrama de Gantt para el lote LoteEj4.tsk ejecutadas mediante el scheduler Round-Robin con un quantum igual 11.



Descripción del diagrama: En la ejecución de las primeras 5 tareas se parece al scheduler FIFO, debido a que puede ejecutarlas enteras en 1 solo Quantum. Pero a partir de la quinta tarea podemos observar el funcionamiento del Round-Robin en todo su esplendor. La tarea número 5 se ejecuta durante 11 ciclos y antes de ser finalizada el scheduler la saca y pone a la siguiente debido a que se termino su 'tiempo' (llego a los 11 ciclos, es decir alcanzo el quantum). La tarea número 6 se ejecuta durante 4 ciclos y luego se bloquea durante otros 4 utlizando un ciclo extra para la ejecución de la llamada. Cuando esta tarea procede a bloquearse el scheduler (a diferencia del FIFO) cambia a la tarea siguiente y esta procede de la misma manera. Las 2 tareas siguientes se comportan muy parecido nada mas que lo primero que hacen es bloquearse y por lo tanto el scheduler las deja solo 1 ciclo en ejecucion. Por último la tarea 10 también realiza un bloqueo apenas se ejecuta por una cuestion aleatoria (la tarea 10 es la taskBatch y el momento en que ejecuta el bloqueo es aleatorio). Luego de haber ejecutado lo correspondiente a todas las tareas vuelve a la número 5 (que no había sido terminada de ejecutar), una vez que ésta termina, cambia a la siguiente (aunque no se haya cumplido el quantum, si no desperdiciaria ciclos de las tareas que ya se encuentran en estado 'ready'). Lo mismo sucede con las dos tareas siguientes. Como la tarea número 8 sigue bloqueada, no la ejecuta al igual que la tarea 9. Carga la tarea 10 hasta que esta se vuelve a bloquear y la cambia por la tarea 8 (que ya no se encuentra bloqueada) ésta se bloquea y repite el procedimiento para la tarea 9. Termina de ejecutar la tarea 10 y acá podemos observar que hay un momento en que si bien las tareas 8 y 9 son las únicas tareas cargadas no finalizadas aún no las carga debido a que se encuentran bloqueadas y por eso se corre la tarea IDLE hasta que se desbloquea alguna de las 2. Finalmente repite esa ultima secuencia 1 vez mas y finaliza la ejecución de todas las tareas.

Resumen: A diferencia del FIFO este scheduler no (necesariamente) ejecuta una tarea hasta que esta finalize, sino que hasta que se cumpla el Quantum de ciclos, el proceso se bloquee o efectivamente concluya. Por otro lado también cabe enfatizar que el orden en que las alterna esta dado por una cola en la que las tareas se encuentran ordenandas por orden llegada, donde una vez ejecutada la última vuelve a la primera no acabada y sólo ejecuta la tarea idle en caso que el resto de las tareas se encuentren bloqueadas. Finalmente este scheduler es mejor (en comparación al FIFO) cuando se desea dar la sensación de que se está ejecutando todo en simultáneo (multiprogramación), obviamente con un rango de quantum tal que sea imperceptible para el usuario y hace un mejor uso del cpu ya que cuando una tarea se encuentra bloqueada la quita y pone a otra tarea que esté en estado READY para no desperdiciar ciclos de ejecución.

3. Ejercicio 5

Se indican las simplificaciones tomadas:
Primero queremos aclarar que se decicidió que a cada tarea se le asigna 1 ticket y este número irá incrementándose a medida que gane compensaciones por bloqueo. A partir de este momento, la tarea tiene más probabilidad de salir ganador en los próximos sorteos. Cuando resulta ganador de la lotería, esta compensación se le es retirada volviendo a 1, cantidad inicial.
Decidimos omitir el hecho de que una tarea pueda subdividirse en threads, junto con el manejo de tickets del usuario. Con esto queremos decir que al no existir usuarios (solo uno) no existe la posibilidad del cambio de cantidad de tickets bajo un mismo usuario.

4. Ejercicio 7

Lote simulado con Round Robin:

TaskBatch 17 1
TaskBatch 17 2
TaskBatch 17 3
TaskBatch 17 4
TaskBatch 17 5
TaskBatch 17 6
TaskBatch 17 7

Diagrama con 2 ciclos de quantum:

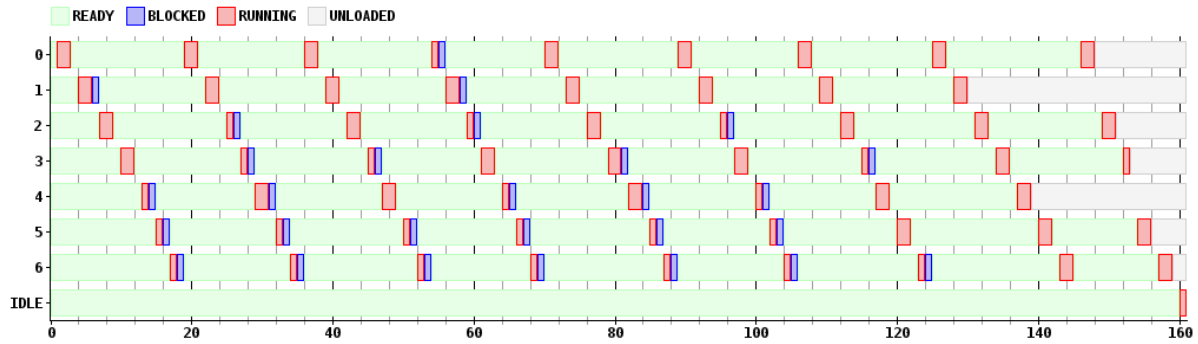


Diagrama con 5 ciclos de quantum:

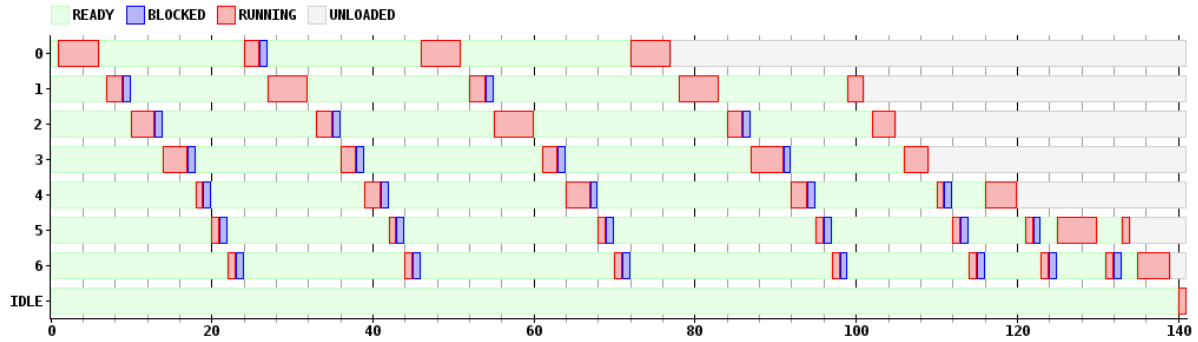


Diagrama con 7 ciclos de quantum:

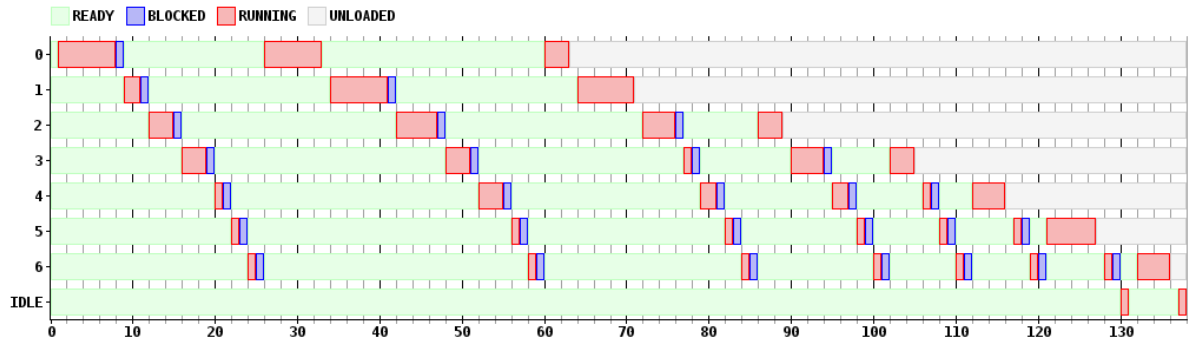


Diagrama con 9 ciclos de quantum:

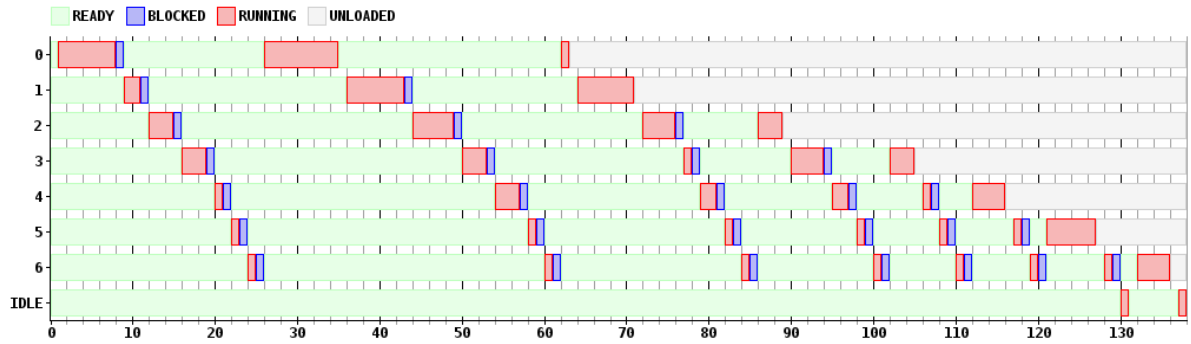


Diagrama con 12 ciclos de quantum:

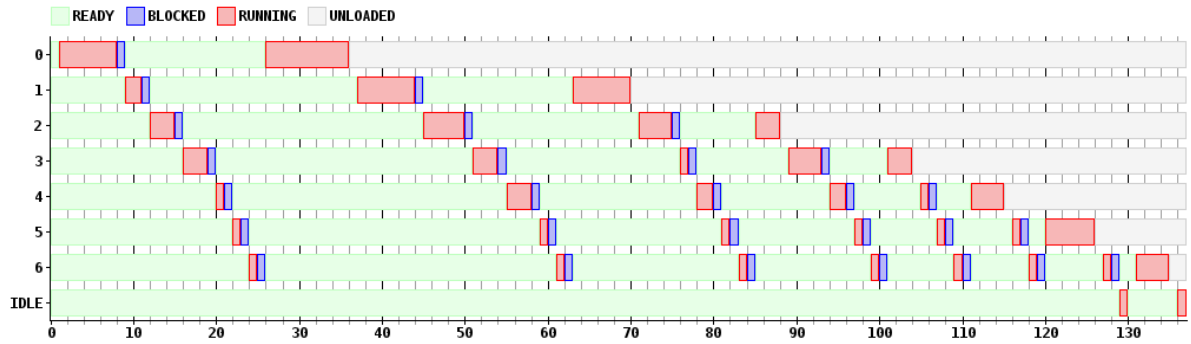


Diagrama con 15 ciclos de quantum:

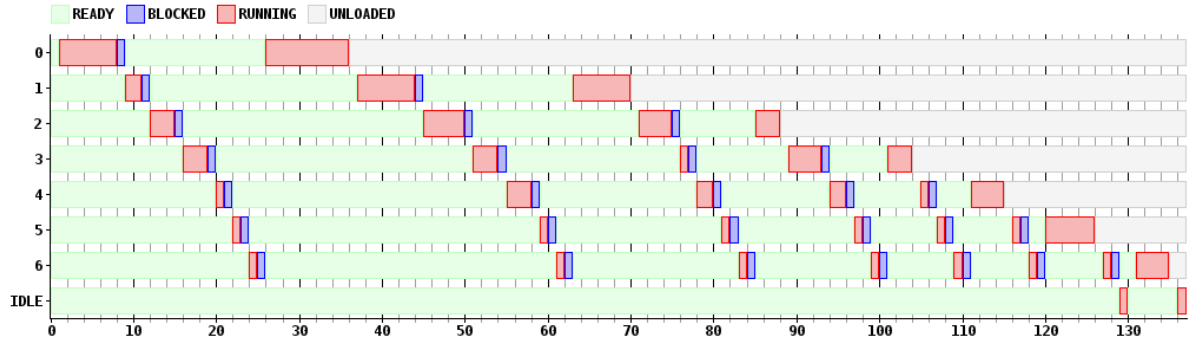
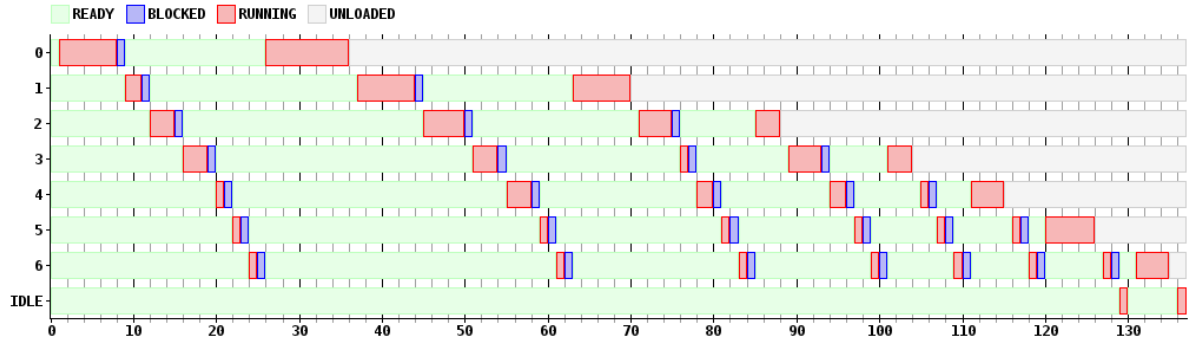


Diagrama con 21 ciclos de quantum:



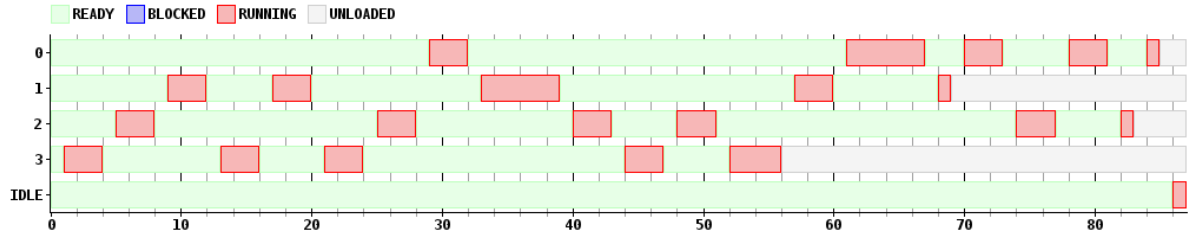
Se puede observar que a medida que los ciclos del quantum aumentan, las tareas finalizan más rapido, debido a que se les da más tiempo de ejecución y hay una menor cantidad de cambios de contexto entre los procesos (ahorra el ciclo que tarda en realizarlo). Además, el aumento de quantum incide en el tiempo total de ejecución de un proceso, ya que al ocurrir menos bloqueos, provoca que la tarea finalice más rapido.
Para este lote de tareas en particular, suele ser más eficiente asignarle una cantidad grande de ciclos por quantum, ya que se realizarán menos bloqueos y menos cambios de contexto, pero no excesiva, porque, como se aprecia en las imágenes, a partir de 12 ciclos por quantum todos tardan lo mismo en finalizar y no mejora la eficiencia, esto es debido a que las tareas se bloquean en menor cantidad y en ningun caso utilizan más del quantum provisto. Por lo tanto, la mejor elección para el quantum es de 12 ciclos.
Con un quantum de 12 ciclos en adelante, se puede notar que lo que más influye en cada proceso son los bloqueos que se producen y no tanto el corte porque no le quedan más ciclos del quantum, produciendo cambios de contexto obligatorios y empeorando bastante el tiempo total de finalización de las tareas, causando un desaprovechamiento de las ventajas del Round Robin frente a otros algoritmos de scheduling, ya que si algo caracteriza a este algoritmo, es de repartir de manera equitativa y racional los recursos, y en este caso se asemeja más a un algoritmo gobernado por bloqueos.

5. Ejercicio 8

Presentamos el siguiente lote corrido con distintos semillas de aleatoridad y tamaño de quantum. El objetivo es mostrar que al ser pseudo-random, la semilla no produce favoritismo entre los procesos y que los diferentes quantum no van a alterar el fairness del algoritmo.
Sabemos que la probabilidad de que gane al no tener entrada salida es 1/n ya que todos los procesos cuentan en cada lotería con la misma cantidad de tickets. Y notemos que la cantidad de veces que gana un proceso la lotería va a ser la cantidad de veces que se realiza la lotería sobre la cantidad de procesos corriendo.

*4 TaskCPU 15

Semilla de Aleatoridad = 1 y Quantum = 3

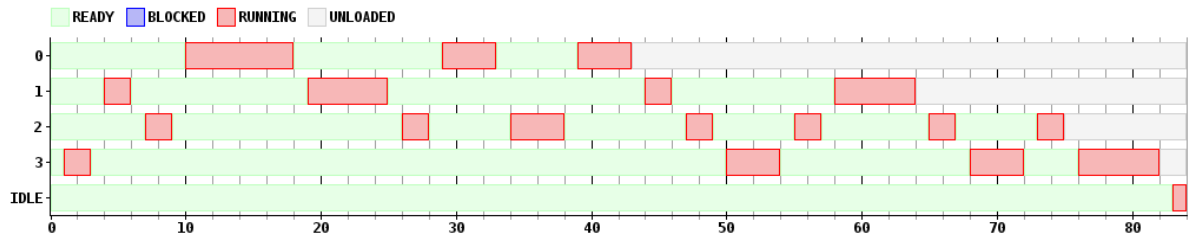


En esta oportunidad notamos como el proceso 3 se ve favorecido, y queremos ver que esto no es cierto cuando n tiende a infinito. Para ello corremos el mismo proceso n veces, con n grande, con distintas semillas de aleatoridad manteniendo el quantum y observamos qué proceso termina primero en cada corrida. A continuación se muestran los resultados de correr el mismo lote 500 veces con semillas de aleatoridad distintas.
for ((i=0; i ¡500 ; i++)) ; do ./simusched lotes/loteEj8.tsk 1 SchedLottery $RANDOM 3 — grep EXIT — awk 'print$3' — head -n 1 ; done — sort — uniq -c

123 0
125 1
119 2
133 3

n=500 y 500/4 = 125
Claramente podemos notar como los procesos tienden a terminar 1/n veces primero, es decir, obtuvieron de forma equitativa el recurso.

Semilla de Aleatoridad = 5 y Quantum = 2

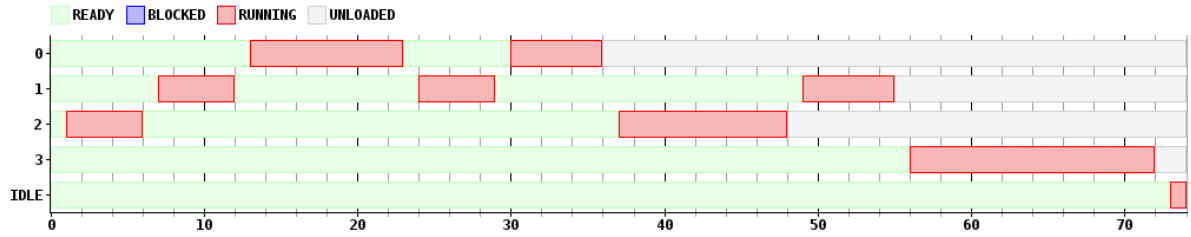


Presentamos otra corrida con otro quantum.
En esta oportunidad notamos como el proceso 1 se ve favorecido. Utilizamos el mismo método para probar que se mantiene la uniformidad.
A continuación se muestran los resultados de correr el mismo lote 500 veces con semillas de aleatoridad distintas.
for ((i=0; i ¡500 ; i++)) ; do ./simusched lotes/loteEj8.tsk 1 SchedLottery $RANDOM 2 — grep EXIT — awk 'print$3' — head -n 1 ; done — sort — uniq -c

119 0
126 1
133 2
122 3

n=500 y 500/4 = 125
Claramente podemos notar como los procesos tienden a terminar 1/n = 1/4 = 125 veces primero, es decir, obtuvieron de forma equitativa el recurso.

Semilla de Aleatoridad = 3 y Quantum = 5



Presentamos otra corrida con otro quantum.
En esta oportunidad notamos como el proceso 1 se ve favorecido. Utilizamos el mismo método para probar que se mantiene la uniformidad.
A continuación se muestran los resultados de correr el mismo lote 500 veces con semillas de aleatoridad distintas.
for ((i=0; i ¡500 ; i++)) ; do ./simusched lotes/loteEj8.tsk 1 SchedLottery $RANDOM 5 — grep EXIT — awk 'print$3' — head -n 1 ; done — sort — uniq -c

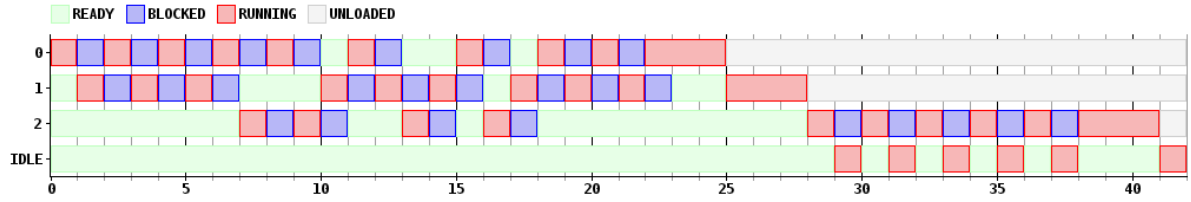
126 0
130 1
122 2
122 3

n=500 y 500/4 = 125
Claramente podemos notar como los procesos tienden a terminar 1/n = 1/4 = 125 veces primero, es decir, obtuvieron de forma equitativa el recurso.

Para la parte b) decidimos tomar el siguiente lote de tareas para mostar la compensación de tickets:

*3 TaskBatch 20 9

Semilla de Aleatoridad = 3 y Quantum = 4



Acá el proceso 1 ganó mucho la lotería porque la cantidad de tickets después de cada bloqueo es mucho mayor ya que es proporcional a la cantidad de tiempo que no usás y al ser un quantum grande, aquel que se bloquea gana muchos tickets para el próximo sorteo. Aquí se puede notar como la compensación de tickets varía según si los procesos se bloquean más que otros, y si el quantum es grande dándole más posiblidad de obtener más tickets.

En conclusión, el algoritmo tiende a ser fairness como RoundRobin o tantos otros. Esto va a depender de la configuración y mientras más sepamos de los procesos de entrada podemos acomodarlos a nuestra ventaja. En caso de que elijamos quantum grandes, la compensación va a ser más grande en caso de un bloqueo ya que lo más probable es que haya gastado poco tiempo, por lo cual para el próximo sorteo en el que participe, va a tener muchos más tickets que el resto teniendo así más probabilidad de ganar la lotería. Es decir que correr el mismo proceso n veces la probabilidad de que gane, va a ser igual en cada uno de estos n experimentos.
Debemos aclarar que si bien estámos hablando de un algoritmo random, computacionalmente es pseudo-random, es por eso que se probaron distintas semillas de aleatoridad y se ve claramente que no afecta al algortimo.