Sistemas Operativos
Trabajo Práctico I

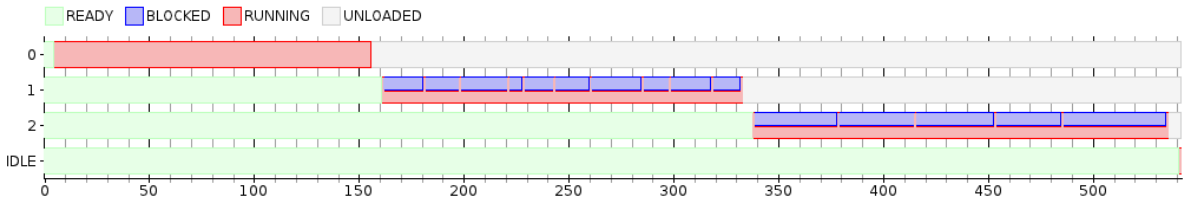
Rodrigo Campos (561/06)

December 12, 2011

Ejercicio 2

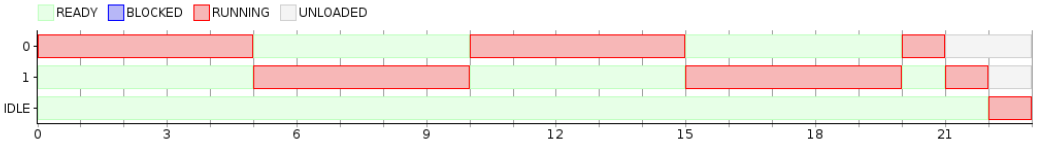
Como lote de tareas se uso el siguiente:

$TaskCPU$ 150
$TaskCon$ 10 5 25
$TaskCon$ 5 30 55
El diagrama de Gnatt generado es:



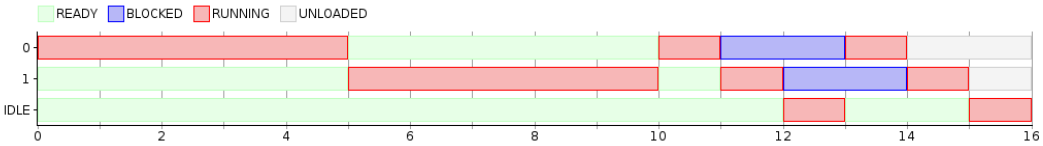
Ejercicio 4

Para mostrar que el comportamiento es el esperado se hicieron varias pruebas. La primera consiste de dos tareas que usan 10 ciclos cada una de CPU. El quantum usado es 5, por lo que se ve que cambian a los 5 ciclos. En total se usa un ciclo mas (11 ciclos) porque, según pudimos debugguear el simulador, un ciclo se usa al cargar la tarea por primera vez (y aunque esta tarea no se esta ejecutando, se le atribuye el tiempo de CPU)

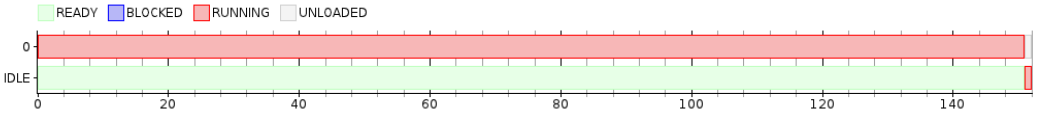


La segunda consiste también en dos tareas, pero en este caso cada un tiene el siguiente patrón de consumo de recursos: 5 ciclos de CPU y luego 2 ciclos IO. Y como quantum se usó 5 ciclos también.

El primer ciclo de ambas tareas no es ocupado por la tarea en sí, sino que es necesario al cargar la tarea por primera vez. Luego hace los 5 ciclos de CPU, interrumpidos porque se excede el quantum y cambia a la otra tarea, y luego se bloquea 2 ciclos. Desspués de desbloquearse se utiliza un ciclo de CPU más, porque al desbloquearse la tarea se consume un ciclo. Es importante notar que cuando ambas tareas estan bloqueadas se ejecuta la tarea idle y ni bien se desbloquea una tarea se la ejecuta.

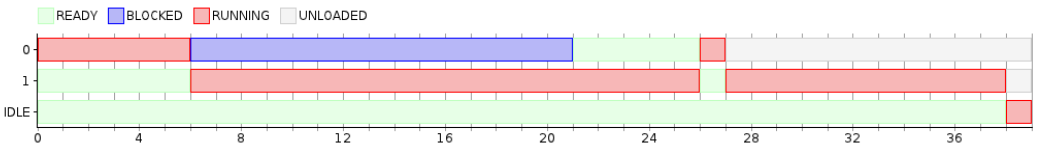


La tercer prueba consistió en solo una tarea ejecutando, para verificar que no haya ningún problema con este caso "borde" y que se le renueve el quantum:



Se utilizó un quantum de 10 y como se puede ver en el gráfico, efectivamente se renueva como es esperado.

La cuarta prueba es similar a la anterior, pero se probó con 2 tareas y que se renueve el quantum si la otra tarea está bloqueada. Y efectivamente esto es lo que ocurrió como muestra el gráfico a continuación:



Entonces, el comportamiento en todas las pruebas fue el esperado.

Ejercicio 5

Como sugirió la cátedra, se realizaron simplificaciones sobre el algoritmo propuesto en el paper. Se decidió implementar la idea general del algoritmo con la extensión de "Compensation Tickets" (previa autorización). Se dejó de lado: "Ticket Transfers" porque es principalmente util para solucionar un problema que excede al simulador que tenemos, "Ticket Inflation" porque "it violates desirable modularity and load insulation properties" y "Ticket Currencies" porque nos pareció demasiado complejo.

Detalles de implementación

Se utiliza una lista donde se tiene cada tarea y la cantidad de tickets que tiene. Inicialmente todas las tareas comienzan con un ticket. Cada ticket representa un número, pero el número que representa depende del lugar que ocupen en la lista. Sea k la cantidad de tickets total de las tareas anteriores a la tarea n, k' la cantidad de tickets de la tarea n, entonces el rango de valores representados por los tickets de la tarea n va desde hasta k hasta $k + k'$. Por ejemplo, si se tienen 2 tareas donde la primera tiene 5 tickets y la segunda 2, el rango de valores representados por los tickets de la segunda tarea es de 6-7 (ambos inclusive).

Cuando se debe elegir el siguiente proceso a ejecutar lo que se hace es generar un número random entre 1 y la cantidad total de tickets usados por las tareas activas. La tarea que tenga el ticket que representa al número random elegido será la siguiente en ejecutar. Es importante notar que si bien al cambiar el orden de la lista se cambia los valores representados por los tickets de cada tarea, no se cambia la probabilidad de que una tarea sea elegida.

También se extendió usando la idea de "Compensation Tickets". Para esto lo que se hizo es que cuando una tarea utiliza menos de su quantum, se le asignan más tickets. Luego, la siguiente vez que se elija un proceso a ejecutar, ésta tendrá mayor probabilidad de ser elegida. Una vez que fue elegida para ejecutar, se le vuelve a la cantidad original de tickets (salvo que de nuevo no utilice todo su quantum).

Ejercicio 7

A continuación se justifica para cada una de las medidas, la mejor elección de quantum.

Ecuanimidad (fairness): que cada proceso reciba una dosis "justa" de CPU (para alguna definicion de justicia)

Si por justa se entiende que todos reciben el mismo quantum, todos los procesos reciben una dosis justa. Esto no tiene que ver con algun valor de quantum usado, sino que es consecuencia de ser un scheduler Round Robin.

Eficiencia: tratar de que la CPU este ocupada todo el tiempo.

Bajo esta definición, no importa el valor de quantum elegido, el scheduler siempre trata de que esté ocupado todo el tiempo la CPU. Si un proceso se bloquea, entonces comienza a ejecturar otro. Solo se ejecutará el proceso "idle" cuando ningún otro proceso pueda ejecutarse. Y se dejará de ejecutar la tarea "idle" ni bien otro proceso esté en estado listo. Luego, trata de que el CPU esté ocupado todo el tiempo.

Sin embargo es importante notar que usando otra definición de eficiencia, esta conclusión podría no ser cierta. En particular usando una definición que tenga en cuenta que el tiempo de CPU sea usado en algo "útil" (para alguna definición de utilidad), a priori el razonamiento previo no aplica.

Tiempo de respuesta: minimizar el tiempo de respuesta percibido por los usuarios interactivos.

Valores pequeños del quantum parecen mejorar esta medida ya que están menos tiempo desde que están en estado listo hasta que vuelven a ejecutar. Entonces, el tiempo de respuesta percibido por el usuario es menor.

Latencia: minimizar el tiempo requerido para que un proceso empiece a dar resultados.

No pudimos ver luego de cuantos ciclos un proceso comienza a dar resultados. El simulador no nos ayuda para este escenario. Sin embargo, se puede ver que siendo q el quantum y n la cantidad de procesos, en peor caso el tiempo que tardará una tarea en ejecutarse es $q \times (n-1)$

Supongamos entonces que la tarea tarda m ciclos en empezar a dar resultados y que $m = q \times k$ (k, q y m enteros). Luego, esta tarea para ejecutar m ciclos, y comenzar a dar resultados, requiere ejecutarse una cantidad k de veces. Y la demora al esperar ejecutarse k veces es:

$k \times q \times (n-1) = m \times (n-1)$

Es decir, en el peor caso y con m múltiplo de q, es independiente del quantum la demora que tenga un proceso en empezar a dar resultados (si el quantum es menor o igual a la cantidad de ciclos que requiere el proceso en empezar a dar resultados)

Tiempo de ejecucion: minimizar el tiempo total que le toma a un proceso ejecutar completamente.

Los ciclos de reloj utilizados para el cambio de contexto son ciclos en los que no se aprovechan para ejecutar ningún proceso en sí, sino mas bien para "administración". El resto de los ciclos sí son usados para ejecutar los procesos. Intuitivamente parece que al maximizar el quantum se minimizan los cambios

de contexto, haciendo así que la cantidad de ciclos de reloj total sea menor. Si se tiene n procesos que cada uno tarda en ejecutar n_i ciclos, el proceso p necesitará esperar el CPU al menos $\left\lceil \frac{n_p}{q} \right\rceil$ veces, y con valores mas grandes de q necesitará esperarlo menos veces ya que $\left\lceil \frac{n_p}{q} \right\rceil$ es menor.

Asumiendo que ningun proceso termina mientras el proceso p ejecuta, todos los procesos ejecutan durante todo su quantum y cs es la cantidad de ciclos de reloj que lleva un cambio de contexto, el proceso p tardará al menos:

$$n_p + \left\lceil \frac{n_p}{q} \right\rceil \times \sum_{i=1}^{n-1} (q + cs) = n_p + \left\lceil \frac{n_p}{q} \right\rceil \times ((n-1)q + (n-1)cs) = n_p + \left\lceil \frac{n_p}{q} \right\rceil (n-1)q + \left\lceil \frac{n_p}{q} \right\rceil (n-1)cs$$

ciclos en terminar desde que comienza a ejecutar.

Sea $q > q'$ y q, q' divisores de n_p se tiene:

$$n_p + \left\lceil \frac{n_p}{q} \right\rceil (n-1)q + \left\lceil \frac{n_p}{q} \right\rceil (n-1)cs < n_p + \left\lceil \frac{n_p}{q'} \right\rceil (n-1)q' + \left\lceil \frac{n_p}{q'} \right\rceil (n-1)cs$$

$$\frac{n_p}{q}(n-1)q + \frac{n_p}{q}(n-1)cs < \frac{n_p}{q'}(n-1)q' + \frac{n_p}{q'}(n-1)cs$$

$$n_p(n-1) + \frac{n_p}{q}(n-1)cs < n_p(n-1) + \frac{n_p}{q'}(n-1)cs$$

$$\frac{n_p}{q}(n-1)cs < \frac{n_p}{q'}(n-1)cs$$

$$\frac{n_p}{q} < \frac{n_p}{q'}$$

$$n_p q' < n_p q$$

$$q' < q$$

Luego, valores de cuantum más grandes, pero múltiplos de la cantidad de ciclos a ejecutar por un proceso, minimizan la cantidad de ciclos que le lleva a un proceso terminar.

Rendimiento (throughput): maximizar el numero de procesos terminados por unidad de tiempo.

Intuitivamente pareciera que si se minimiza la cantidad de ciclos que le lleva a un proceso terminar, la cantidad de procesos terminados por unidad de tiempo va a ser mayor. Trataremos de ver experimentalmente si esto es así o no.

Para hacer esto se utilizaron tareas batch (interactivas) que hagan una cantidad de bloqueos baja en relación al tiempo de CPU que usan. Esto es porque

si la cantidad de bloqueos es la máxima, fuerza a que se ejecute necesariamente de la única forma posible y para muchos valores de quantum se bloqueará antes de que éste termine. En cambio, si la cantidad de bloqueos es baja, podrá aprovechar el quantum. Como lote de tareas se usó:

$$TaskBatch\,50\,1$$

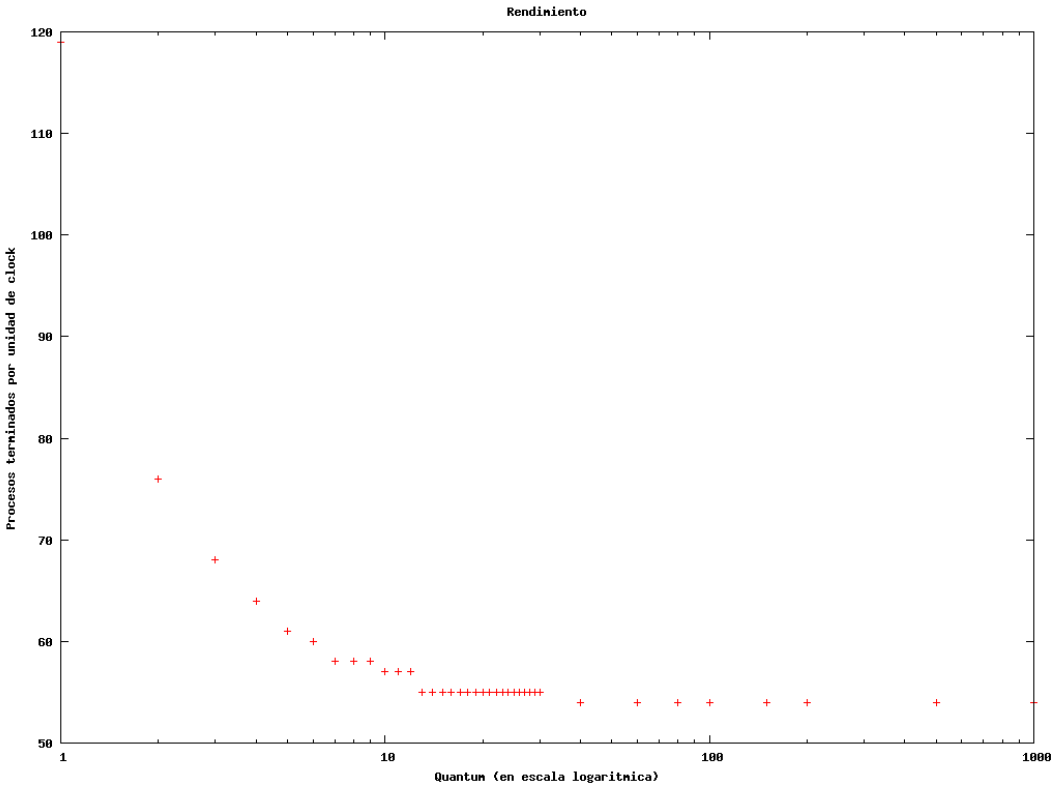
$$TaskBatch\,50\,2$$

$$TaskBatch\,50\,3$$

$$TaskBatch\,50\,4$$

$$TaskBatch\,50\,5$$

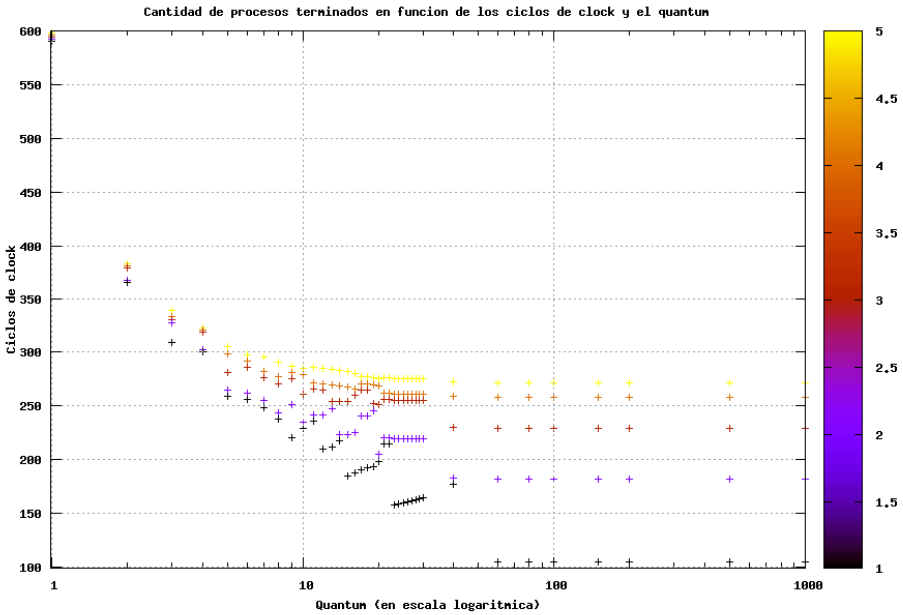
A continuación se puede ver graficada la cantidad total de ciclos de CPU usados para correr todas las tareas del lote sobre la cantidad de tareas, para distintos quantums. Es decir para cada quantum se grafica $\frac{<\#ciclos\,totales>}{5}$



Se decidió graficar $\frac{<\#ciclos\,totales>}{5}$ y no la fracción inversa, que se corresponde con el rendimiento, porque de esta forma el resultado es un número más grande que permite graficarlo más fácilmente. Tener en cuenta esto al analizar el gráfico.

Acá se puede ver que la cantidad de procesos terminados por ciclo de clock es mayor con quantums más grande, hasta cierto punto donde esto ya no mejora más. Esto tiene sentido pues de seguro nunca tardaran menos en ejecutar que la suma de los tiempos que requiere cada tarea, y probablemente haya otra cota inferior más grande que no viene al caso analizar.

También nos pareció interesante graficar la cantidad de procesos terminados en funcion de los ciclos de clock, para distintos quantums. Es decir, para todos los quantum, cuantos procesos habian terminado hasta un ciclo de clock cualquiera:



Este gráfico muestra que con quantums mas grandes el primer proceso termina en menos ciclos, lo mismo para el segundo proceso, etc. Luego, la cantidad de procesos terminados por ciclo de clock es siempre mayor o igual con un quantum más grande, para cualquier intervalo que comience en el ciclo de clock 0.

Es decir, un quantum mas grande parece maximizar el throughput.

Liberacion de recursos: hacer que terminen cuanto antes los procesos que tiene reservados mas recursos.

No importa el quantum, esta medida se ignora totalmente. El scheduler no tiene ninguna noción de los recursos usados por una tarea.

Ejercicio 8

En el paper "Lottery scheduling: Flexible proportional-share resource management" citado en el enunciado del trabajo práctico, menciona que si X es la variable aleatoria que cuenta la cantidad de veces que un proceso salió sorteado como siguiente proceso a ejecutar, entonces X tiene distribución binomial. La cantidad de sorteos es las veces que se repite el experimento y p la probabilidad de ganar un sorteo, que depende de los tickets que tenga el proceso.

Trataremos de estimar experimentalmente la esperanza al cabo de varios sorteos y la compararemos con la esperanza teórica de la distribución binomial para los parámetros dados. Es importante notar que si un proceso se crea, o muere, entonces p varía. Lo mismo pasa si una tarea se bloquea, ya que se le dan "Compensation tickets", alterando así las probabilidades. Para centrarnos solo en la aleatoriedad del algoritmo se deja de lado estas situaciones y se analizará un lote de tareas que sean solo de uso de CPU (pues si una tarea se bloquea se le otorgan "Compensation tickets" modificando p), todas las tareas del lote comenzarán a la vez y solo se analizará mientras todas las tareas del lote estén corriendo (de caso contrario se crearían/morirían procesos alterando el valor de p). Entonces, como lote de tareas se decidió usar el siguiente:

$$TaskCPU\ 100$$

$$TaskCPU\ 100$$

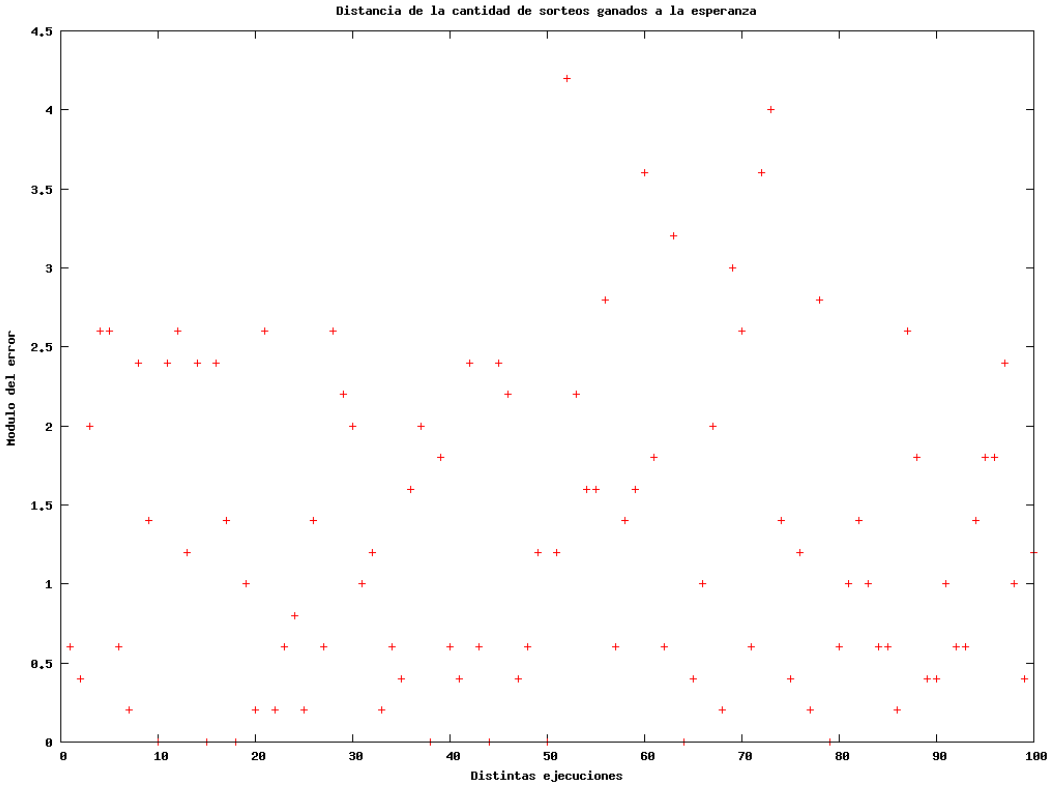
$$TaskCPU\ 100$$

$$TaskCPU\ 100$$

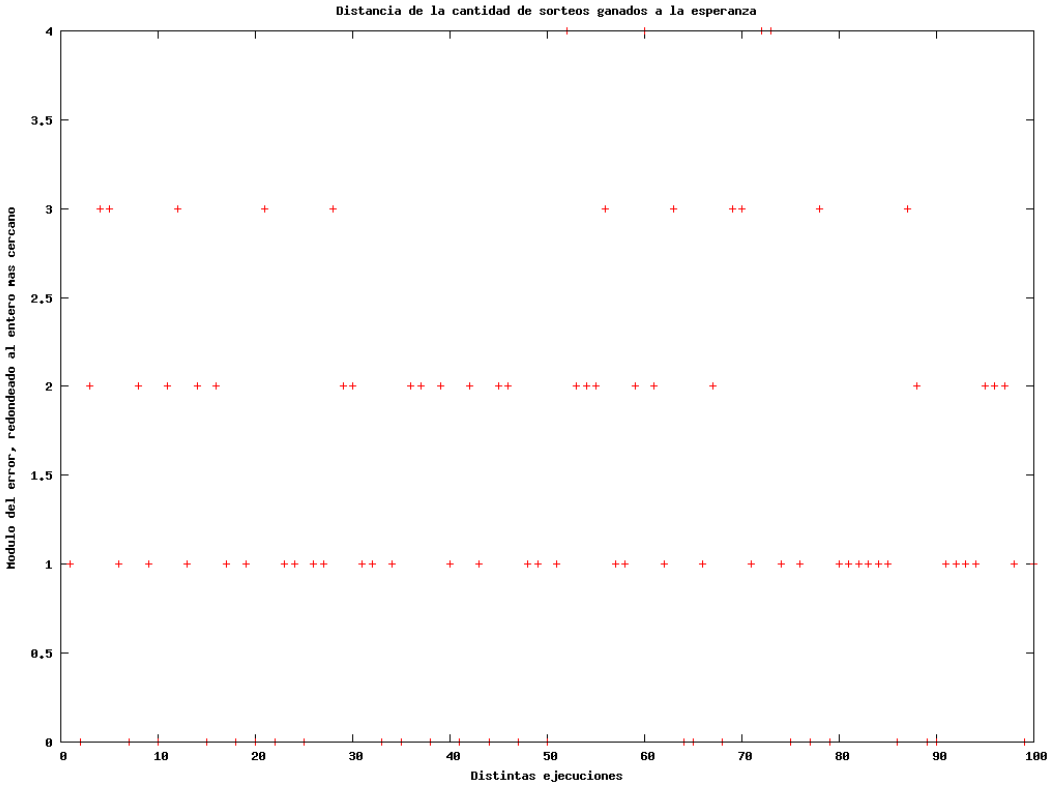
$$TaskCPU\ 100$$

La elección de cuantos ciclos de CPU usarán se hizo en función del quantum para que el lote implique una cantidad no despreciables de sorteos cada vez que se corre. Entonces se eligió 10 como quantum y 100 como tiempo de CPU que requiere una tarea del lote. Además, como solo es importante para este análisis qué proceso ganó cada sorteo, se usó 0 como tiempo de cambio de contexto, ya que no aportaba realmente nada.

A continuación se grafica para 100 corridas con semilla elegida al azar la distancia entre la esperanza y la cantidad de sorteos que ganó un proceso:

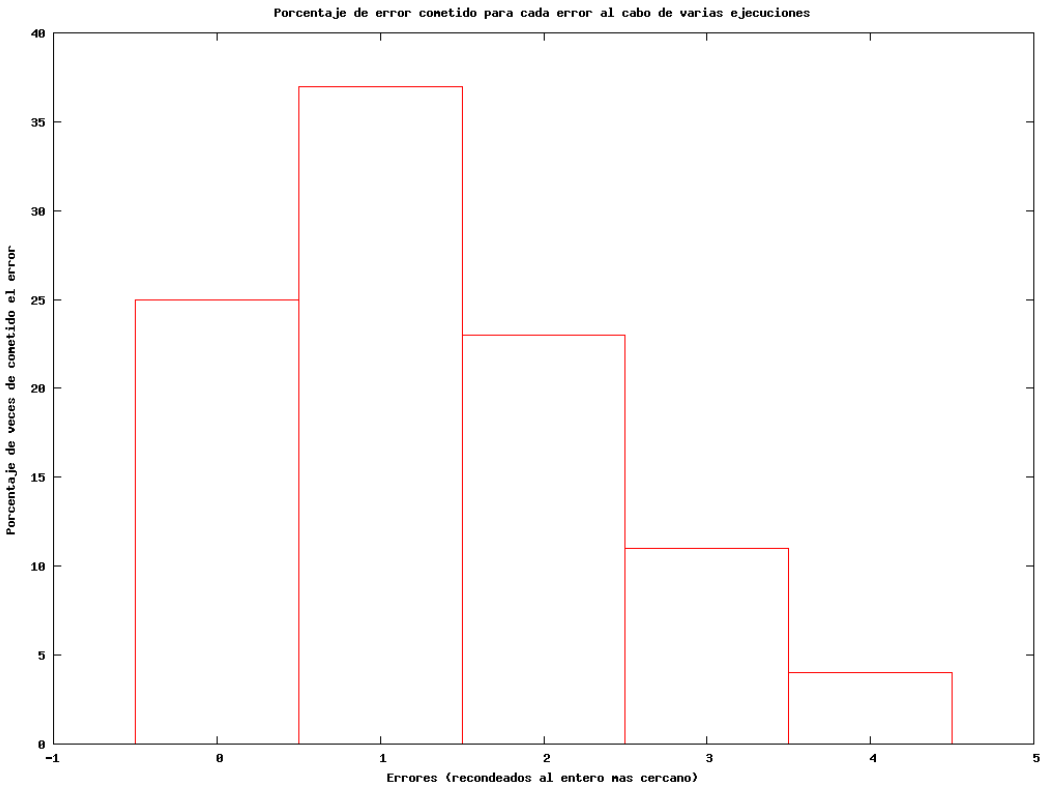


Como los puntos están muy dispersos los puntos en el gráfico, se decidió hacer otro gráfico usando redondeo al entero más cercano del error. De esta forma se puede ver mejor la distribución de los puntos (aunque perdiendo "precisión"). Viendo el siguiente gráfico nos ayudó a interpretar mejor el gráfico anterior, ya que podemos ver cómo están dispersos y en el siguiente se puede ver mejor cómo se distribuyen los puntos cuando se redondea al entero mas cercano.



También, es importante notar que la variable aleatoria X: "cantidad de sorteos ganados por el proceso 0" es discreta y siempre tomará un valor entero. Por lo que si la esperanza es 8.2 y X en nuestra muestra es 8, es el entero más cercano. Entonces tiene sentido interpretar al error como 0, ya que no hay otro entero más cercano. Esto fue motivo, también, para realizar el gráfico anterior.

Sin embargo, a modo de resúmen, se graficó el porcentaje de veces a lo largo de las corridas que el error, redondeando al entero mas cercano, fue 0, 1, 2, 3 y 4 para los distintos procesos. Se obtuvo el siguiente gráfico de barras:



Acá se pude ver que en más del $25\% + 35\% + 20\% = 80\%$ de las veces la distancia de la cantidad de sorteos ganados a la esperanza fue menor a 2.