Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2333 - Sistemas Operativos y Redes - 2/2018

Tarea 1: Scheduling

Nombre: Manuel Becker
Número Alumno: 13622986

1. ¿Cómo se podría extender su implementación para funcionar en un sistema multicore (> 1 núcleos)? ¿Podría disminuir alguno de los tiempos de los procesos (turnaround, response, waiting). Describa qué decisiones de diseño debería tomar. No es necesario que lo implemente.

Para la implementación de Round Robin en esta tarea fue necesario tener 4 colas: Cola de nuevos procesos, cola de procesos *READY,* cola de procesos *WAITING* y cola de procesos *FINISHED*. Esto fue todo lo necesario para poder simular el comportamiento con una sola *CPU*. En el caso de tener múltiples núcleos, se podrían realizar algunas variaciones.

Se puede tener una cola de procesos *READY* por cada núcleo. De esta manera, desde la cola de nuevos procesos se pueden ir añadiendo a cada una de esas colas *READY* los procesos según la cantidad de procesos en la cola, tiempo que requiera la *CPU* para procesarlos, o con alguna otra características que para la cola de prioridades respectiva, priorizando la asignación de nuevas tareas a la cola en la que serían procesados en un tiempo menor. El funcionamiento de la cola de procesos *READY* se podría mantener de la misma forma que en la implementación de la tarea, asignando según la disponibilidad de su *CPU* los procesos a procesar. De igual forma, la cola de procesos *WAITING* y *FINISHED* podría ser común a todos los procesos, independiente del núcleo que los está procesando, y su funcionamiento sería igual que en la implementación.

Esta modificación de la asignación de los procesos desde la cola de procesos nuevos a la cola de procesos *READY*, disminuiría el *response time* de los procesos, dado que disminuiría el tiempo de la primera atención al haber más núcleos capaces de recibirlo para procesar. Dado que la asignación desde la cola de procesos nuevo aseguraría la asignación de los procesos a la cola que tenga menos procesos y que las colas de procesos *READY* por núcleo tengan el tiempo de ejecución más o menos distribuidos para evitar que una cola se lleve más procesos de alto consumo de *CPU.* También el *turnaround time* y *waiting time* se verían reducidos de la misma forma que el *response time.* Más núcleos y colas de procesos *READY* implica que el número de procesos a procesar antes que un nuevo proceso se divide entre todos los núcleos, por lo que el tiempo desde que llega el proceso hasta que entra a un núcleo se disminuye (*response time)*. Y de la misma forma, ya que el tiempo que deben esperar por *I/O Burst* es constante, al estar divididos en un mayor número de núcleos y colas *READY*, el *waiting_time* es menor, ya que pasan menos tiempo en la cola *READY.*

2. Investigue el funcionamiento del scheduler asignado. Describa brevemente su funcionamiento, para qué sistema operativo ha sido usado, que características de diseño hace que el sistema sea interactivo y cómo se comportaría ante una carga intensiva en uso de CPU.

Lottery scheduling

Lottery scheduling es un *scheduler* probabilístico que puede ser o no *preemptive.* La idea detrás del *scheduler* es que cada proceso recibe al menos un *ticket*, donde tener un ticket implica la posibilidad de poder ser seleccionado como ganador e ingresar a la *CPU.* Esto implica directamente que *Lottery scheduling* resuelve el problema de *starvation*, ya que necesariamente cada proceso recibe al menos un *ticket*, lo que le da una probabilidad mayor a cero de ser elegido. Un proceso que tenga más número de *tickets* tendrá mayores posibilidades de ser el ganador. Este último punto le da el enfoque probabilístico al *scheduler* y además, se tiene un nivel de aleatoriedad para la asignación de los *tickets* por proceso. Si existen 2 procesos (P1 y P2), y P1 tiene 75 tickets mientras que P2 tiene 25 tickets, implica que el uso de la *CPU* tendrá un 75% de recursos asignados a P1 y 25% a P2, aunque esto no se cumplirá siempre debido a su naturalez probabilística. (http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-lottery.pdf). **Fue usado en el sistema operativo *FreeBSD*.**

La característica de diseño que hace del *scheduler* uno interactivo es que los tickets entre procesos son repartidos entre todos, con una distribución pareja. De esta manera, todos los procesos tendrían igual probabilidad (o parecida) de ser ganadores y los procesos, disminuyendo el tiempo de ejecución de un conjunto de procesos que están en cola.

El uso de *Lottery scheduling* en una carga intensiva en uso de *CPU* dependerá de varias cosas. El número de *tickets* que se repartan entre los diferentes procesos. Si estos distribuyen relativamente bien (normal), el manejo de los procesos dependerá del resultado del sorteo. Dado que es *random* la selección, y si estos distribuyen normal, implicará que todos podrán ser elegidos con igual probabilidad y será posible entonces ejecutar los programas de manera similar a otros *schedulers*. De igual forma, debido a la naturaleza *random* de la selección, se podría dar que algunos procesos salgan más veces y otros no sean seleccionados tanto, lo que implicaría un aumento en los *waiting time*, *turnaround time* y *response time* según los diferentes procesos.

En el caso de que los *tickets* no sean repartidos con una distribución normal entre todos los procesos, algunos de estos tendrán más porcentaje de uso de la *CPU*, y probablemente usen la *CPU* más que otros procesos. En esta situación, las estadísticas mencionadas anteriormente aumentarían considerablemente para los procesos con menor cantidad de *tickets.*