Nombre: Sebastián Cárdenas
N° Alumno: 13638211

Informe

1. ¿Como se podría extender su implementación para funcionar en un sistema multicore (> 1 núcleos)?

Para la implementación del método Round Robin en multi-cores o multi-cpu, se podría hacer de dos maneras. La primera es dejando un Queue para todas las CPU, esto provocaría que una disminución en los tiempos de espera de los procesos dentro del Queue. ya que la tasa de atención del conjunto de CPU aumentaría con respecto a que sólo una CPU esté integrada. También se utilizará un método FIFO en donde al desocuparse alguna de las CPU, se saca el primer elemento del Queue y se procesa. La segunda forma es que puede existir un Queue por CPU, esto provocaría un mejor manejo de prioridades entre los procesos ya que cada Queue corresponde a un rango de prioridad en específico con su CPU correspondiente, ahora si una Queue queda vacía y la CPU está en IDLE, se puede flexibilizar y hacer que procese algún proceso de la siguiente Queue con más prioridad.

¿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.

Primero haremos algunas definiciones básicas de estos parámetros, el “turnaround time” es el tiempo en que el proceso está dentro del sistema, es decir, desde que se crea hasta que finaliza. El “response time” es el tiempo desde que se crea hasta que se ejecuta por primera vez en la CPU y por último el “Waiting time” es el tiempo total en que el proceso a estado sin ejecutarse o en espera (tiempo total en Queue Ready + tiempo total en Queue Waiting).

Nuestro caso base para ser comparado es el de un proceso Round Robin one-core, por lo tanto bajo lo planteado en la pregunta 1). si usamos el primer escenario de solo un Queue con varias CPU, los tres parámetros disminuyen debido a que aumenta la tasa de atención como sistema esto provoca que haya un menor tiempo de espera en el Queue, ya que aumenta el flujo de salida de los procesos. En el segundo caso es como tener 4 Round Robin en paralelo esto disminuye el turnaround y el response debido a que existe un mejor manejo de prioridades y hay menos procesos por Queue comparado con el caso base y al existir esta separación también disminuye el tiempo de waiting.

2. Completely Fair Scheduling (CFS)

El Completely Fair Scheduling o “CFS”, fue desarrollado para ser utilizado en el Kernel de Linux. este método se centra en maximizar la utilización general de la CPU, mejorando el rendimiento de interacción con los procesos. El CFS intenta mantener una equidad entre las tareas, para que todos puedan ser ejecutados en CPU con un tiempo proporcional a su prioridad.

Según lo planteado por Jons M. Para determinar la prioridad de un proceso, el CFS mantiene la cantidad de tiempo otorgada a una tarea determinado el “Virtual Runtime”. Cuanto menor es el Virtual Runtime de una tarea, es decir, la menor cantidad de tiempo que una tarea necesita estar en la CPU, mayor es su necesidad de estar en el procesador. por lo tanto se usa el Virtual Runtime para determinar el orden de ejecución. También el CFS incluye el concepto de “sleeper fairness” para garantizar que las tareas que no están actualmente ejecutados puedan recibir una parte comparable de CPU para cuando finalmente lo necesiten.

El CFS no utiliza Queue sino que usa un Red-Black Tree para almacenar los procesos, esta estructura de datos es un árbol de búsqueda binaria que tiene 5 propiedades y la particular característica de auto-balancearse. Las características son:

- Todo nodo es o bien rojo o bien negro.
- La raíz es un nodo negro.
- Todas las hojas (NULL) son negras.
- Todo nodo rojo debe tener dos nodos hijos negros.
- Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

Este tipo de estructura siempre asegura estar balanceado con una altura de $2 \cdot log_2(n+1)$, por lo tanto su complejidad está acotada por $O(log\ n)$. haciendo eficiente el uso de scheduler y soportando cargas excesivas de CPU. tal como se mencionó antes, los procesos se clasifican con su Virtual Runtime, entonces estos se van agregando a la estructura de forma de que el Virtual Runtime más corto es el más prioritario y se ubica hacia la izquierda del árbol. por consecuencia siempre se ejecutan los procesos de izquierda a derecha del último nivel, respetando las propiedades y restricciones que tiene un Red-Black Tree.

Referencia

Jons, M. (2018). Inside the Linux 2.6 Completely Fair Scheduler. Retrieved from https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/index.html