**Algoritmos de planificación de procesos, según la arquitectura:**

*<Vale aclarar que para estos dos tipos de arquitecturas trabajaremos con el esquema de 5 estados (explicado en el apunte de “Programas a Procesos”>*

**ARQUITECTURAS**

*Hay dos tipos:*

**-Apropiativos o Preemtive:**

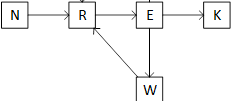
***Este tipo de arquitectura se caracteriza por su planificación a través de la READY QUEUE.***

Una vez que el proceso entra en ejecución continúa ejecutándose hasta terminar o cuando solicita entradas o salidas o algún servicio del sistema operativo.

*-FIFO.*

*-SJF.*

*-SPN (también puede ser expulsivo).*



**New** (N): Es la creación de un nuevo proceso.

**Ready** (R): El proceso está en la cola de espera del procesador.

**Execute** (E): El proceso entra en ejecución en el CPU.

**Wait** (W): El proceso entra en la cola de espera de un dispositivo.

**Kill** (K): Finaliza el proceso.

Secuencias posibles dentro de Apropiativo:

* New a Ready.
* Ready a Execute.
* Execute a Kill.
* Execute a Wait.
* Wait a Ready.

**-Expulsivos o Non-Preemtive:**

Un proceso en ejecución puede ser expulsado y devuelto a la Ready Queue. La decisión puede ser tomada cuando llegue un proceso nuevo, cuando un proceso vuelve de la Wait Queue o cuando se cumple un tiempo de ejecución determinado (quantum).

*-Round Robin.*

*-STR.*

*-Prioridades.*



**New** (N): Es la creación de un nuevo proceso.

**Ready** (R): El proceso está en la cola de espera del procesador.

**Execute** (E): El proceso entra en ejecución en el CPU.

**Wait** (W): El proceso entra en la cola de espera de un dispositivo.

**Kill** (K): Finaliza el proceso.

Secuencias posibles dentro de Apropiativo:

* New a Ready.
* Ready a Execute.
* Execute a Ready.
* Execute a Kill.
* Execute a Wait.
* Wait a Ready.

Los procesos siempre tienen que entrar en CPU antes de poder ir a entradas o salidas y mueren en CPU.

**Arquitecturas Apropiativas Ejemplos de Algoritmos**

**FIFO (First in, First Out):**

Los procesos se ejecutan en orden de llegada y en caso de haber un empate en el arribo se resuelve por orden alfabético, mientras que las colisiones entre dos procesos que vuelven a la Ready Queue (o uno que vuelve y otro que está arribando desde New) se resuelven por la antigüedad del proceso (el que tiene el arribo más chico primero). Las colisiones en I/O siempre se resuelven por FIFO, no importa el algoritmo implementado.

Imaginense por ejemplo una caja de supermercado, donde los clientes son atendidos en el orden de llegada a la cola.

Ejemplo:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Proc | A | CPU | I/O1 | I/O2 | CPU |
| A | 2 | 10 |  | 20 | 5 |
| B | 5 | 20 |  | 10 | 15 |
| C | 3 | 15 | 5 |  | 5 |
| D | 0 | 5 | 15 |  | 10 |

Notese como los procesos van ejecutándose por orden de llegada. En T35 puede observarse una colisión entre A y C volviendo a la Ready Queue desde entradas y salidas, ganando la posición el proceso A por ser el más viejo en el sistema.

**SJF (Shortest Job First):**

Se suman los tiempos de trabajo de CPU e I/O, dando por resultado el “peso” del proceso. Se ejecutan los procesos según su peso, empezando por el menor. En caso de arribar un proceso a la Ready Queue, se vuelve a reordenar para ubicarlo en la posición que le corresponda. Corre el riesgo de producirse “Starvation” (inanición), cuando hay procesos demasiado pesados, es decir, que los procesos más pesados serán constantemente pateados al final de la Ready Queue.

En el ejemplo de la caja del supermercado, podemos ver la analogía con una caja en la que se atiende primero a los que tengan menor cantidad de productos, que en este caso sería el peso del proceso.

Tanto FIFO como SJF son algoritmos apropiativos, es decir que una vez que los procesos toman la CPU no la sueltan hasta terminar la totalidad de su carga de CPU.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Proc | A | CPU | I/O1 | I/O2 | CPU |  |
| A | 3 | 10 |  | 20 | 5 | 35 |
| B | 2 | 20 | 15 |  | 15 | 50 |
| C | 7 | 15 | 5 |  | 5 | 25 |
| D | 0 | 5 | 15 |  | 10 | 30 |

**SPN (Shortest Process Next) (Expulsivo):**

Se suma solo los tiempos de trabajo del CPU, dando por resultado el “peso” del proceso. La Ready Queue se va reordenando de menor a mayor peso a medida que van llegando los procesos. Al ser expulsivo, un proceso de menor peso puede expulsar a uno de mayor peso que se encuentre en ejecución, devolviendo a este último a la Ready Queue y entrando inmediatamente en ejecución.

**Arquitecturas Expulsivas Ejemplos de Algoritmos**

**Round Robin:**

Los procesos se ejecutan por orden de llegada (como en el algoritmo FIFO) pero el sistema operativo les da un “quantum” (un tiempo máximo que pueden permanecer en ejecución antes de ser expulsados de vuelta a la Ready Queue). Si un proceso tiene un tiempo de procesador de 7 y el quantum del Round Robin es de 5, solo ejecutará 5 unidades de tiempo y volverá al final de la Ready Queue para dejar lugar al siguiente proceso, quedándole pendientes dos tiempos de ejecución.

**Múltiples núcleos:**

Ejemplo:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Proc | A | CPU | I/O1 | I/O2 | CPU |
| A | 0 | 4 | 2 |  | 2 |
| B | 6 | 6 | 8 |  | 6 |
| C | 1 | 8 |  | 2 | 6 |
| D | 3 | 10 | 4 |  | 4 |

CPU1: R.R. = 2

CPU2: R.R. = 4

**Hijos e hilos:**

Los procesos hijos son creados por el padre y son una copia casi exacta, es decir, al momento de su creación, se le asigna un PID (identificador de proceso) y se copia toda la información del padre (código, pila, PCB…) y es visto por el sistema operativo como un proceso completamente nuevo.

Los hilos, en cambio, no son un proceso nuevo. Para el sistema operativo, los hilos son vistos como el proceso padre (por lo que no poseen un PID propio). Cada hilo va a tener su propia pila y su propio TCB, pero comparten el BCP y el espacio de direcciones del padre. La planificación de los hilos está a cargo de una biblioteca y esta puede diferir de la planificación del sistema operativo. Por ejemplo, el sistema operativo puede trabajar con Round Robin mientras que la biblioteca de los hilos trabaja con FIFO. Cabe destacar en el ejemplo dado que si bien el algoritmo FIFO no es expulsivo, los hilos serán expulsados igualmente al superar el quantum de la planificación del sistema operativo.

Los estados de un hilo son:

-**Ready**: cuando está listo para ser ejecutado.

-**Execute**: cuando esta en ejecución.

-**Wait**: cuando espera a que ocurra un evento antes de poder continuar ejecutándose.

Ventajas de los hilos:

-Se tarda mucho menos tiempo en crear un nuevo hilo en un proceso existente que en crear un nuevo proceso.

-Se tarda mucho menos tiempo en terminar un hilo que un proceso.

-Se tarda mucho menos tiempo en conmutar entre hilos de un mismo proceso que entre procesos.

-Los hilos hacen más rápida la comunicación entre procesos, ya que al compartir memoria y recursos, se pueden comunicar entre sí sin invocar el núcleo del SO.

Ejemplo:

Planificación S.O.: R.R. = 5

Planificación biblioteca hilos: R.R. = 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Proc | A | CPU | I/O1 | I/O2 | CPU |
| KLT | 4 | 7 | 10 |  | 5 |
| P1 | 2 | 10 |  |  | 5 |
| ULT1.1 |  | 6 |  | 5 | 5 |
| ULT1.2 |  | 10 | 10 |  | 4 |
| P2 | 0 | 7 |  |  | 9 |
| ULT2.1 |  | 1 | 5 |  | 10 |
| ULT2.2 |  | 6 |  | 5 | 10 |

