Sistemas informáticos

Procesos en Linux

Semana 25

Mayo 2023

Cruz García, Iago

Introducción

Procesos

- Estados
  - Tipos
  - Hilos
  - Demonios
- Gestor de procesos
  - Contexto
  - Cargador de procesos
  - Planificador de procesos
  - Pila simple vs pila múltiple

Introducción

Hemos visto que la **gestión de procesos** es uno de los pilares más importantes para los Sistemas Operativos debido a la necesidad de asignar los tiempos en CPU de las tareas.

Preparar un **planificador de procesos** es una tarea más compleja, pero gestionar los mismos en un Sistema Operativo es importante para conocer lo que ocurre en nuestro equipo en todo momento.

En este tema veremos los estados en los que se puede encontrar un proceso, cómo se gestionan en Windows y Linux o cómo se reparten las tareas el gestor y el planificador de procesos.

Procesos

Un **proceso** es un programa en ejecución. Más concretamente, es el conjunto de instrucciones y variables que conforman un programa en ejecución en memoria. Y puede que un programa ejecute uno, dos o más procesos a lo largo de su vida útil, por lo que la definición más común de "Proceso=programa" se empieza a quedar atrás.

Estos procesos siguen una serie de estados, desde su inicio hasta su terminación, manejados por el Sistema Operativo, permiten comprobar la ejecución de un proceso.

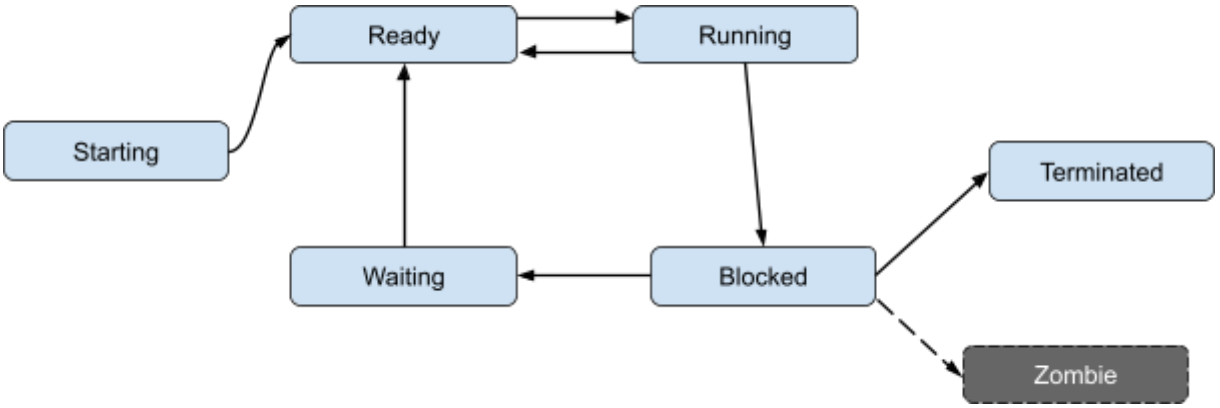
Estados

Entendemos por **estado** la actividad actual del proceso en el conjunto del Sistema Operativo.

Existen 4 estados primitivos, independiente del Sistema Operativo:

- **Ready State**: Esperando a que se le asigne un procesador.
- **Running State**: Ejecutando en CPU.
- **Waiting State**: En espera de un evento (generalmente, E/S)
- **Terminated State**: Finalizando su ejecución.

Estos se suceden e intercambian entre ellos pero un estado sólo puede estar en un estado en cada instante. Un ejemplo de sucesión en Linux se puede observar en el siguiente esquema:



Esquema de estados de procesos

Estas transiciones tienen un par de particularidades:

- Aparece el estado **Blocked**, donde se lanzan los procesos que pueden ser terminados o enviados a la lista de espera.
- Los estados **Ready** y **Running** son estados a los que puede transicionar el proceso de forma bidireccional. Sería la representación de un proceso que todavía no terminó sus operaciones en CPU.
- En algunos casos, **Waiting** y **Ready** pueden representarse como el mismo estado.
- El estado **Zombie** es un caso en el que un proceso se da por terminado pero permanece ocupando un hueco en la lista de procesos, sin consumir recursos más allá de lo visual para representarlo.

- Incluso a día de hoy con multiprocesadores, multihilo, multiprogramación, etc, los procesos tendrán que respetar la secuencia de transiciones.

Tipos

Algo a tener en cuenta sobre los procesos es que suelen compartir recursos del sistema y la comunicación entre ellos es común. En estos términos, podemos clasificarlos de la siguiente forma:

- **Independientes:** Procesos que no interactúan de ninguna forma con otros procesos. Esto quiere decir que no utilizan recursos de ningún otro proceso ni compiten por su tiempo en CPU. Son muy poco frecuentes y generalmente asociados a tareas nucleares del Sistema Operativo.
- **Cooperativo:** Procesos que requieren recursos compartidos con otros. Pueden ser recursos como datos o tiempo de procesador compartido (procesos del mismo programa).
- **Competitivo:** Procesos que requieren los mismos recursos que otros, sean datos o uso en CPU. Generalmente hablamos de procesos ajenos unos a otros, bien por programas diferentes o trabajos en hilos diferentes.

Hilos

Los **hilos** son "partes" de un procesador. Realmente son procesos especiales, muy ligeros, que se encargan de gestionar procesos. Podríamos decir que

son procesadores virtuales, pues tendrán información compartida con la CPU que ejecute dicho hilo.

Esto quiere decir que los hilos de una misma CPU compartirán datos, pero los externos no podrán. Además, tienen la información de la pila, el registro de CPU y del contador de programa, para gestionar los procesos de forma paralela.

Demonios

Los **demonios** son procesos muy particulares. Se ejecutarán siempre en segundo plano, sin ningún tipo de interacción con el usuario (ni interfaz ni consola de comandos), con su propio usuario o permisos. La única forma de comprobar su estado podría ser comprobar los servicios activos o, en algunos casos, la generación de registros o "logs" de cambios de estado.

Gestor de procesos

Hay dos protagonistas involucrados en esta tarea. Uno ya lo conocemos, el **planificador de procesos.** Pero el otro, el **cargador de procesos** es igual de importante.

Contexto

Entendemos por **contexto** el estado e información de un proceso en un instante de tiempo. Cuando se cambia de un estado a otro, se almacenará el contexto del proceso, se realizará la interrupción de su ejecución y el

cambio de estado necesario, almacenando la información para continuar con el mismo tras el cambio.

Cargador de procesos

Su función principal es la de cargar cada instrucción en memoria y realizar el cambio de contexto de cada proceso. Estos cambios se agruparán en **eventos**, ocasiones no predecibles (para los procesos) que interrumpirán su ejecución:

- **Interrupción:** La CPU deja de procesar la información. Puede darse por software (el usuario terminó el programa o detuvo el proceso) o por hardware (otros procesos requieren la CPU)
- **Excepción:** Un error en la ejecución del proceso obliga a él mismo a detenerse, evitando un desarrollo erróneo o imprevisible del programa.

El **cargador de procesos** entonces almacena información en este cambio de contexto: PID (número identificador de cada proceso), estado, prioridad… entre otros.

Planificador de procesos

Como ya estudiamos en temas anteriores, se encargará de decidir qué proceso debe ocupar la CPU en cada instante de tiempo. Para ello, utilizará **algoritmos** que decidirán qué proceso debe ejecutarse en cada momento.

Estos deben cumplir unos objetivos:

- **Equidad**: Deben ser justos con la asignación de tiempos.
- **Eficientes**: Deben ocupar el procesador el mayor tiempo posible, sin tiempos idílicos.
- **Latencia**: Tiempo de respuesta a los usuarios lo más bajo posible.
- **Rendimiento**: Debe conseguir ejecutar la mayor cantidad de procesos en el menor tiempo posible.

Cada uno de estos objetivos tiende a desequilibrar a otro. Por ejemplo, un algoritmo que trate de ser lo más equitativo posible, reducirá el rendimiento considerablemente.

Algunos de los algoritmos básicos más utilizados son:

- **Prioridades:** Algoritmo clásico, los procesos son asignados una prioridad y dependiendo de la prioridad en el Sistema Operativo será un proceso u otro el que use la CPU.
- **FIFO (First In First Out):** También conocido como **FCFS (First Come, First Served)** se trata de un algoritmo que establece que el proceso que primero entra en memoria será el primero en finalizar. Esto puede darse de dos formas:
    - **Exclusiva:** El proceso entra, bloquea la CPU y no la libera hasta terminar.
    - **Inclusiva:** El proceso entra en CPU pero permite su uso intercambiándose con otros procesos. Como fué el primero en entrar y el resto van en orden, será el primero en terminar igualmente pero el resto pueden actualizarse y ocupar mejor el tiempo en procesador.

- **SJF (Shortest Job to Finish):** El proceso que ocupa la CPU será aquel que terminará antes o que le llevó menos tiempo de uso de CPU global.
- **SRT (Shortest Remaining Time):** El proceso al que le quede menos tiempo de uso de CPU será el que la utilice en cada momento.
- **RR (Round Robin):** También conocido como "espera circular", el algoritmo indica la cantidad de tiempo que puede estar un proceso en CPU hasta que sea expulsado para dar paso al siguiente.
    - A tener en cuenta: si la espera circular está asignada a 2 tiempos, por ejemplo, y un proceso está solo en la lista, podrá estar de forma indefinida aunque el algoritmo indique 2 tiempos. Pero en el momento que otro proceso aparezca, será expulsado si lleva el tiempo suficiente.

Todos los algoritmos pueden utilizarse como criterios principales o subcriterios. En caso de empate debido a prioridades similares (en SJF por ejemplo, trabajos con cargas de tiempo idénticas), se podrá utilizar el subcriterio para decidir el uso de CPU.

Pila simple vs pila múltiple

El concepto de la pila aparece en los planificadores por naturaleza de la arquitectura de computadores de Von Neumann. En un principio, con una pila simple, los procesos deben ejecutarse tal cuál son ordenados la primera vez. Esto puede provocar deficiencias a la hora de asignar tiempos en CPU,

pues un proceso que entre de nuevo en la pila no podrá "colarse" o posicionarse en un lugar más favorable.

Sin embargo, con una pila múltiple, el algoritmo de planificación ordenará las pilas varias veces (dependiendo de la cantidad de pilas) para que la ejecución en CPU sea lo más justa y eficiente posible.

Evidentemente esto conlleva un consumo de proceso, pues el tiempo en el que se ejecuta el algoritmo de planificación, no se están ejecutando procesos en espera.