

operativos hasta llegar a algo muy de moda como es el *cloud computing* (computación en la nube), cuyos servidores están cada vez más virtualizados.

Debido a este interés por la virtualización de servidores, se ha decidido utilizar máquinas virtuales para ver no sólo el rendimiento de los sistemas operativos ante un problema dado, sino también como estos responden siendo instalados en máquinas virtuales.

## 3. Gestión de procesos

### 3.1. Definición

Utilizando la definición dada en el material de la asignatura Sistemas Operativos se va a explicar qué es un proceso. La definición es la siguiente: “*Un proceso es básicamente un entorno formado por todos los recursos necesarios para ejecutar programas. Desde el punto de vista del SO, un proceso es un objeto más que hay que gestionar y al cual hay que dar servicio*”<sup>4</sup>.

Podemos decir que un programa es una entidad pasiva, en tanto en cuanto es un conjunto de instrucciones de código máquina y datos almacenados en un ejecutable. Mientras que un proceso sería la ejecución de ese programa, es decir, el programa en acción. Esto indica que los procesos son dinámicos, están en constante cambio debido a estos recursos necesarios, ya que al intentar realizar algún tipo de acción puede ser que tenga que permanecer a la espera de que dicho recurso esté disponible, por ejemplo una petición de lectura del disco duro, y que el brazo lector del disco duro lo esté utilizando otro proceso.

Al igual que las instrucciones de programa, los procesos incluyen los contadores de programa que indican la dirección de la siguiente instrucción que se ejecutará de ese proceso y los registros de CPU, así como las pilas que contienen datos temporales, como son los parámetros de subrutina, las direcciones de retorno y variables locales. Los procesos también contienen una sección de datos con variables globales y memoria dinámica. Todo ello permite gestionar de una manera más eficaz los procesos en los sistemas operativos multiprocesos, ya que cada proceso es independiente, por lo que el bloqueo de uno no debe de hacer que otro proceso en el sistema se bloquee.

En estos sistemas operativos multiproceso se intenta maximizar la utilización de la CPU, por lo que los procesos se ejecutan simultáneamente en la CPU y sólo quedan a la espera de ejecución si requieren de algún recurso del sistema que esté ocupado en ese momento, en cuanto obtiene dicho recurso podrá ejecutarse de nuevo. Todo este proceso de gestión lo realiza el sistema operativo, por lo que es el que decide si un proceso es más prioritario que

---

<sup>4</sup> Definición dada en el punto 1 del módulo 6 (La gestión de procesos) dentro de la asignatura *Sistemas Operativos*.

otros. Es el programador el que decide esta prioridad, por ejemplo en el caso de Linux se utiliza un número en la programación de estrategias para garantizar la equidad de los procesos.

Cada proceso se representa en el sistema operativo con un bloque de control de proceso (PCB, Process Control Block). En este PCB se guardan una serie de elementos de información de los mismos. Estos elementos son: el identificador del proceso, el estado del proceso, registros de CPU (acumuladores, punteros de la pila, registros índice y registros generales), información de planificación de CPU (prioridad de proceso, punteros a colas de planificación, etc.), información de gestión de memoria, información de contabilidad (tiempo de uso de CPU, números de procesos, etc.), información de estado de dispositivos E/S (lista de archivos abiertos, etc.).

### 3.2. Estados de un proceso

En un sistema multiprogramado o multitarea donde existen muchos procesos y un procesador, puede ocurrir que en un momento dado sólo se ejecute un proceso o varios y los demás estén esperando a ser procesado o esperen la finalización de una operación de E/S. Los pasos por los que puede pasar un proceso se pueden representar con un diagrama de estado como el de la figura<sup>5</sup> 4. Así se puede apreciar que a medida que un proceso se ejecuta va cambiando de estado dependiendo de las preferencias que cada uno tengan asignadas, por lo que será el procesador el que se encargue de ejecutar unos u otros.

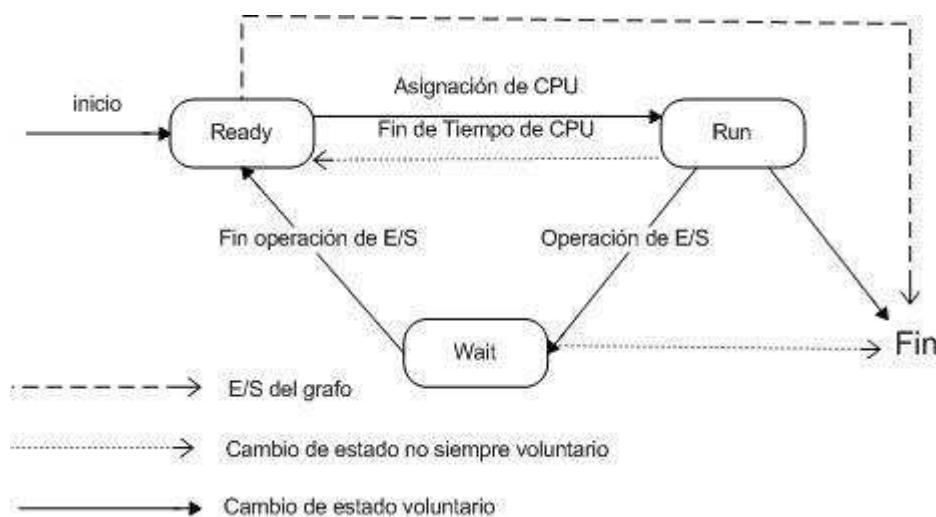


Figura 4 Diagrama de Estados de un proceso

Cómo se aprecia en la imagen 4 los estados por los que puede pasar un proceso son los siguientes:

<sup>5</sup> Diagrama obtenido del material de la asignatura Sistemas Operativos (módulo 6. Gestión de procesos)

- Nuevo. En los sistemas operativos hay varias razones por las que se crea un proceso. Entre éstas se pueden destacar; la inicialización del sistema, cuando se arranca el sistema se generan una serie de procesos ya bien sean para interactuar con el usuario o procesos en segundo plano con una función específica, como por ejemplo el aceptar la solicitud de una página web que está en dicha máquina; ejecución de una llamada al sistema por parte de otro proceso, un proceso puede requerir la descarga de ficheros, por lo que serán otros procesos los que se encarguen de ubicar el archivo o archivos en la ubicación específica; por medio de la acción de un usuario, por ejemplo al hacer doble clic en un ícono; mediante el inicio de un trabajo por lotes.

Una vez el proceso ha sido creado queda a la espera de ser admitido, por lo que si es así pasaría a estado preparado o en caso contrario terminaría dicho proceso.

- Preparado (ready o listo). Un proceso en este estado está esperando a que se le asigne un procesador. Como se puede apreciar en la figura 4, un proceso en este estado puede, o bien finalizar, lo que ocurriría por la acción de otro proceso o por algún acontecimiento externo, o bien el proceso pasa al estado ejecución, ya que el gestor de procesos le asigna una CPU para ser ejecutado.
- Ejecución (run). El proceso en este estado está en la CPU ejecutando instrucciones. Puede ocurrir tres situaciones; que el proceso execute todas las instrucciones hasta su última línea de código y finaliza; pasa a estado bloqueado (wait) por que espera una acción externa como la entrada de información por teclado; o bien el proceso pasa a estado preparado debido a que ha agotado su tiempo de ejecución, por lo que cede su tiempo de ejecución.
- Bloqueado (wait, en espera). El proceso está esperando a que se produzca un evento externo, como una señal de E/S, y pasaría a estado ejecución. Al igual que el estado preparado, el proceso puede finalizar debido a un acontecimiento externo.

El sistema operativo utiliza varias colas para gestionar los estados, cada cola puede tener una política diferente. Así, podemos encontrar una cola para los estados preparados y una cola para los estados en espera. El planificador del procesador al examinar estas colas asigna el procesador al proceso más conveniente.

### 3.3. Planificación de un proceso

El sistema operativo es el encargado de decidir qué procesos entran en la CPU cuando ésta queda libre, y en qué momento sale de la CPU el proceso que está en ejecución. Todo ello se lleva a cabo a través de una política de planificación de procesos.

Se pueden definir múltiples políticas de planificación de procesos: por orden de llegada, primero la tarea más breve, por orden de prioridad, etc. En definitiva, lo que una política de planificación debe conseguir es que los procesos obtengan adecuadamente sus turnos de ejecución por lo que son tratados de la misma forma, que no se produzca sobrecarga, es decir, el planificador debe responder rápidamente ante cargas de trabajo ligera y responder de la misma forma ante cargas de trabajo similares. Y obtener un buen rendimiento, por lo que se debe lograr finalizar el mayor número de procesos y maximizar el tiempo de respuesta.

No existe una política de planificación óptima para todas las computadoras, sino que depende de las características de los procesos. Así se puede ver cómo una política obtiene unos resultados excelentes en un sistema, sin embargo en otro sistema el rendimiento es mucho menor. Ello se debe a las características de los procesos, donde cada uno puede tener una cantidad de operaciones de E/S enorme como es el caso de las bases de datos, otros usan mayormente la CPU, otros realizan una mayor lectura de datos frente a otros, hay procesos que requieren una prioridad máxima en los turnos de ejecución, es el caso de los procesos de tiempo real, y hay procesos que requieren más tiempo de ejecución que otros, por lo que habrá que valorar si terminar primero los cortos o no.

Existen diferentes planificadores en el sistema. Primero nos encontramos el planificador a largo plazo, el cual es el encargado de controlar el grado de multiprogramación en el sistema, intentando conseguir una mezcla adecuada de trabajos en CPU y E/S. Es por tanto el encargado de suministrar los procesos a la cola de planificación a corto plazo.

Existe también un planificador a medio plazo. Es el encargado de suspender y posteriormente restaurar procesos de poco interés, realizando el intercambio de los mismos entre la memoria principal y el disco o memoria secundaria. Dicho proceso es conocido como swapping, y se ejecuta cuando hay escasez de recursos.

El planificador a corto a plazo es el encargado de asignar y desasignar la CPU. Su trabajo es coger un proceso de la cola de procesos preparados y asignarle una CPU. Hay dos tipos de planificadores a corto plazo; no expulsivas, el proceso abandona la CPU cuando termina o a la espera de un suceso externo; y expulsivas, el proceso que se está ejecutando puede pasar a estado listo enviado por parte del sistema operativo, es el caso de los sistemas de tiempo compartido y tiempo real, como UNIX, Windows NT/XP o superior, MAC OS X. Las políticas de planificación de procesos más comunes son las siguientes:

- Primero en llegar, primero en salir (FCFS). El primero proceso que llega a la cola de preparados será el primero en ser planificado y pasado a la CPU. Es no expulsiva, por lo que no es adecuada para los sistemas de tiempo compartido, por lo que provocan una especie de convoy con los procesos de E/S.
- Primero el proceso más corto (SJF). Cuando el proceso que está en la CPU cesa su ejecución se elige de la cola de procesos preparados aquel cuya ráfaga de CPU sea menor, es decir, que su tiempo de ejecución sea menor. Es del tipo no expulsiva, aunque existe una versión expulsiva (SRTF) que cuando llega un proceso más pequeño que el que se está ejecutando a la cola de preparados, éste es bloqueado y pasa a estado preparado mientras se ejecuta el nuevo proceso.
- Prioridades. Cada proceso tiene asignada una prioridad ya sea por medio del sistema operativo o por el usuario. Los procesos se dividen en distintas colas dependiendo de la prioridad, por lo que el planificador elegirá primero los procesos de la primera cola mediante FIFO, y cuando quede vacía elegirá los de la segunda cola. Esta política puede ser expulsiva o no. Esto puede provocar que haya procesos que se queden sin ejecutar debido a esta prioridad, la solución pasaría por aumentar la prioridad progresivamente a los procesos en espera.
- Turno rotatorio (Round-Robin). Adecuado para los sistemas de tiempo compartido. Consiste en generar periódicamente una interrupción de reloj, donde cada proceso dispone de un cuento de tiempo máximo (*quantum*), por lo que cuando termina este tiempo, el proceso en ejecución pasa a preparado y pasa a ejecutarse el siguiente proceso en la cola de preparados según FIFO.
- Retroalimentación. Otro tipo de planificación es trabajar con diferentes colas de preparados cada una con una política diferente. Así si un proceso que ha pasado del estado ejecución al estado preparado, primero estaba en la cola 1 y ahora

pasa a la cola 2, y así sucesivamente hasta llegar a la última cola hasta que termina.

Los procesos en Linux pueden ser divididos en tres categorías: interactivos, tiempo real o por lotes. Los procesos de tiempo real son manejados bien por un algoritmo FIFO o turno rotatorio (Round-Robin), ya que al ser procesos considerados como prioritarios deben de ser ejecutados antes que los demás. Los demás procesos son manejados utilizando una planificación Round-Robin con un sistema de envejecimiento utilizado en la planificación de prioridades mencionada

Mientras en Windows, la planificación de procesos se basa en la utilización de colas múltiples de prioridades. Posee 23 niveles de colas clasificadas de la 31-16 en clase de tiempo real y las demás en clase variable. Cada cola es manejada mediante Round-Robin, pero si llega un proceso con mayor prioridad, se le es asignado el procesador.

El caso de Mac OS X es parecido al de Windows, utilizando varias colas de procesos cada una con un nivel de prioridad. Un hilo puede pasar de una cola a otra dependiendo de los requerimientos. Estos niveles se pueden manejar mediante llamadas al sistema. Los niveles son: normal, alta, Kernel y tiempo real.

### 3.4. Creación y destrucción de procesos

Los sistemas operativos que son objeto de estudio tienen diferentes formas en la gestión de procesos, por lo que en este apartado faremos una distinción entre el sistema UNIX y el sistema Windows.

El por qué se hace el estudio del sistema UNIX es debido a que tanto el sistema Linux como el sistema Mac OS X están basados en dicho sistema, por lo que es más razonable hacer la mención a UNIX y no a estos dos sistemas. Las únicas diferencias fundamentales detectadas entre UNIX y Mac OS X se basan en el sistema de ficheros, donde Mac OS X utiliza HFS+ frente al UFS de UNIX y el estándar ext3 de Linux.

Respecto al Kernel, ambos sistemas están basados en el Kernel de UNIX, pero actualmente Mac OS X utiliza un híbrido del núcleo de UNIX Mach llamado XNU, mientras que Linux utiliza un núcleo monolítico llamado Linux.

- **UNIX.**

Los procesos en los sistemas UNIX están identificados por un número que es único (PID), además de que cada proceso el espacio de memoria utilizado, formado por tres segmentos: el código, los datos y la pila. También contiene la información de control del proceso, que indica la planificación y estado del proceso, la estructuración de datos y la comunicación entre procesos. Otra información importante es el número de identificación de usuario (UID) y el número de identificación del grupo de usuarios al que pertenece el proceso (GID). Todo ello pertenece al bloque de control de proceso (PCB).

La creación y destrucción de procesos en UNIX se ajusta a la filosofía de la manera más sencilla posible, así las llamadas al sistema tienen el mínimo número de parámetros. Las llamadas correspondientes a la creación, destrucción y bloque o espera de un proceso son respectivamente: fork, exit y wait.

La llamada fork crea un nuevo proceso hijo idéntico al proceso padre. Tienen la misma imagen de memoria, el mismo bloque de control de proceso y los mismos archivos abiertos, aunque situados en distintos espacios de memoria. Para poder distinguir a ambos procesos, la llamada fork devuelve distintos valores, el hijo recibe un valor 0 y el padre el PID del hijo.

La llamada exit finaliza un proceso. Cuando se produce esta finalización del proceso hijo, se manda al sistema la llamada wait, para que el padre se bloquee a la espera de la finalización del hijo. Si el proceso hijo finalizara antes de que el padre recibiera esta llamada, el proceso hijo se convertiría en un proceso en estado zombie, y hasta que no se ejecute esta llamada wait el proceso no se eliminará. Para evitar la acumulación de procesos UNIX prevé un límite de números de procesos zombie y aquellos procesos hijos que se destruyen más tarde que sus procesos padres, al quedar huérfanos sería el primer proceso del sistema (init) el que se encarga de recoger su estado de finalización.

La llamada wait bloquea el proceso que lo ha llamado hasta que uno de sus procesos hijos es destruido, por lo que si el proceso no tiene hijos wait regresa y el valor devuelto es igual al Pid de dicho proceso hijo.

En UNIX se puede utilizar un comando para eliminar un proceso que se está ejecutando por la razón que estimemos oportuno; gran consumo de recursos, ya no es necesario, etc. Esta orden es “*kill Pid proceso a eliminar*”, por lo que para eliminar un proceso primero se debe conocer su Pid.

- **Windows.**

Un programa en Windows es controlado por eventos. Así el programa principal espera la llegada de un evento como puede ser al presionar una tecla, y posteriormente invoca un procedimiento para procesar dicho evento, actualización de pantalla, del programa, etc.

Windows también tiene llamadas al sistema al igual que UNIX, de hecho el número de llamadas es extremadamente grande. En Windows encontramos que por cada llamada al sistema existe un procedimiento de biblioteca que los invoca. Por ello Windows ha creado un conjunto de procedimientos, llamado API Win32 [5], que se ha de utilizar para solicitar servicios al sistema operativo.

La creación de procesos en Windows se genera mediante la llamada CreateProcess, que tanto crea el proceso como carga el programa en el nuevo proceso. Esta llamada tiene 10 parámetros: el programa a ejecutar, atributos de seguridad, bits de control de archivos abiertos, prioridad, especificación de la ventana a crear y un apuntador a la estructura a la que al invocador se le devuelve información del proceso recién creado. CreateProcess tiene 100 funciones más para administrar y sincronizar procesos.

Al igual que en UNIX, en Windows se crea un nuevo proceso hijo a partir del padre, el cual tiene su propio espacio de direcciones como en UNIX, pero mientras en UNIX el espacio de direcciones del hijo era una copia del padre, en Windows el espacio de direcciones del hijo es completamente diferente al del padre desde el principio.

La llamada en Windows relativa a la destrucción de un proceso es ExitProcess, y al igual que ocurría con UNIX cuando se especifica la llamada WaitForSingleObject junto con un parámetro que especifica un proceso, quien lo invoca espera hasta que se produce la finalización del proceso.

En Windows encontramos multitud de llamadas al sistema igual que en UNIX (Figura 25), aunque hay algunas en UNIX que no se pueden utilizar en Windows, como es el manejo de enlaces, manejo de montaje de unidades, etc. Si en UNIX utilizamos la orden kill Pid para eliminar un proceso, en Windows existe la función TerminateProcess.

<b>UNIX</b>	<b>WIN32</b>	<b>Descripción</b>
Fork	CreateProcess	Crea un proceso nuevo
Waitpid	WaitForSingleObject	Puede esperar a que un proceso termine
Execve	(ninguna)	CreateProcess= fork + execve
Exit	ExitProcess	Termina la ejecución
Open	CreateFile	Crea un archivo o abre uno existente
Close	CloseHandle	Cierra un archivo
Read	ReadFile	Lee datos de un archivo
Write	WriteFile	Escribe datos en un archivo
Lseek	SetFilePointer	Mueve el apuntador de archivo
Stat	GetFileAttributesEx	Obtiene diversos atributos de archivo
Mkdir	CreateDirectory	Crea un directorio nuevo
Rmdir	RemoveDirectory	Elimina un directorio vacío
Link	(ninguna)	Win32 no maneja enlaces
Mount	(ninguna)	Win32 no maneja montajes
umount	(ninguna)	Win32 no maneja montajes
Chdir	SetCurrentDirectory	Cambio el directorio de trabajo actual
Kill	TerminateProcess	Elimina un proceso
Time	GetLocalTime	Obtiene la hora actual

Figura 5. Llamadas de la API Win32 que corresponden con llamadas de Unix

## 4. Gestión de memoria

La gestión de memoria se encarga de asignar la memoria física del sistema a los programas, éstos se expanden hasta llenar la memoria con que se cuenta.

Todas las computadoras tienen una jerarquía de memoria, con una pequeña cantidad de memoria caché, una cantidad mucho mayor de memoria principal (RAM) y decenas o centenas de gigabyte de almacenamiento en disco.

El administrador de memoria es el encargado de administrar la jerarquía de memoria. Es el encargado de saber qué partes de la memoria están en uso o no, asignar y liberar la memoria principal a los procesos que la requieren, y administrar los intercambios entre la memoria principal y el disco.

Se puede decir que los objetivos principales de un sistema de gestión de memoria pasan por ofrecer a cada proceso un espacio lógico propio proporcionando una protección entre los procesos, permitir que los procesos compartan la memoria. Además se debe maximizar el rendimiento del sistema y proporcionar a los procesos mapas de memoria grandes.

En un sistema de multiprogramación cada programa debe contener dentro del código referencias al espacio de memoria a utilizar, ya que el mismo no siempre será el mismo, por tanto el sistema tendrá que realizar una reubicación de las direcciones