**PROCESOS**

**Libro de Andrew S. Tanenbaum**

**(Sistemas Operativos Modernos)**

**2.1 PROCESOS** (83)

En cualquier sistema de multiprogramación, la CPU conmuta de un proceso a otro con rapidez, ejecutando cada uno durante décimas o centésimas de milisegundos: hablando en sentido estricto, en cualquier instante la CPU está ejecutando la apariencia de un paralelismo.

**2.1.1 El modelo de proceso** (84)

En este modelo, todo el software ejecutable en la computadora se organiza en varios procesos. Un proceso no es más que una instancia de un programa en ejecución, incluyendo los valores actuales del contador de programa, los registros y las variables. En concepto, cada proceso tiene su propia CPU virtual; en realidad, la CPU real conmuta de un proceso a otro, pero para entender el sistema es mucho más fácil pensar en una colección de procesos que se ejecutan en (pseudo) paralelo, en vez de tratar de llevar la cuenta de cómo la CPU conmuta de programa en programa. Esta conmutacion rapida de un proceso a otro se conoce como multiprogramación.

Cada proceso tiene su propio flujo de control (es decir su propio contador de programa lógico) y cada uno se ejecuta en forma independiente. Desde luego que solo hay un contador de programa físico, por lo que cuando se ejecuta cada proceso, se carga su contador de programa lógico en el contador de programa real. Cuando termina, el contador de programa físico se guarda en el contador de programa lógico almacenado, del proceso en memoria.

La *diferencia entre un proceso y un programa* es sutil pero crucial. Un proceso es un programa en ejecución. A diferencia del programa un proceso es dinámico y tiene un contador de programa.

**2.1.2 Creación de un proceso**

Hay cuatro eventos principales que provocan la creación de procesos:

1. El arranque del sistema.
2. La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.
3. Una petición de usuario para crear un proceso.
4. El inicio de un trabajo por lotes.

Generalmente, cuando se arranca un sistema operativo se crean varios procesos. Algunos de ellos son procesos en primer plano; es decir, procesos que interactúan con los usuarios y realizan trabajos para ellos. Otros son procesos en segundo plano, que no están asociados con usuarios específicos sino con una función específica. Los procesos que permanecen en segundo plano para manejar ciertas actividades como correo electronico, paginas Web, noticias, etc, se conocen como demonios (daemons).

En todos los los casos mencionados para crear un proceso hay que hacer que un proceso existente ejecute una llamada al sistema de creación de proceso. Ese proceso puede ser un proceso de usuario en ejecución, un proceso del sistema invocado mediante el teclado o ratón, o un proceso del administrador de procesamiento por lotes. Lo que hace en todo caso es ejecutar una llamada al sistema para crear el proceso. Esta llamada al sistema indica al sistema operativo que cree un proceso y le indica, directa o indirectamente, cual programa debe ejecutarlo.

En UNIX sólo hay una llamada al sistema para crear un proceso: **fork**. Esta llamada crea un clon exacto del proceso que hizo la llamada. Después de fork, los dos procesos (padre e hijo) tienen la misma imagen en memoria, las mismas cadenas de entorno y los mismos archivos abiertos. Eso es todo. Por lo general, el proceso hijo ejecuta después a execve o una llamada al sistema similar para cambiar su imagen de memoria y ejecutar un nuevo programa. Por ejemplo, cuando un usuario escribe un comando tal como sort en el shell, este crea un proceso hijo, que a su vez ejecuta a sort. La razón de este proceso de dos pasos es para permitir al hijo manipular sus descriptores de archivos después de fork, pero antes de execve, para poder lograr la redirección de la entrada estándar, la salida estándar y el error estándar.

**2.1.3 Terminación de procesos**

Una vez que se crea un proceso, empieza a ejecutarse y realiza el trabajo al que está destinado. Tarde o temprano el nuevo proceso terminara, por lo general debido a una de las siguientes condiciones:

1. Salida normal
2. Salida por error
3. Error fatal
4. Eliminado por otro proceso

La mayoría de los procesos terminan debido a que han concluido su trabajo. Cuando un compilador ha compilado el programa que recibe, ejecuta una llamada al sistema para indicar al sistema operativo que ha terminado. Esta llamada es exit en UNIX y exitProcess en Windows. Los programas orientados a pantalla también admiten la terminación voluntaria.

**2.1.4 Jerarquía de procesos** (89)

En algunos sistemas, cuando un proceso crea otro, el proceso padre y el proceso hijo continúan asociados en ciertas formas. El proceso hijo puede crear por sí mismo más procesos, formando una jerarquía de procesos. Un proceso solo tiene un padre y cero, uno, dos o más hijos.

**2.1.5 Estado de un proceso**

Aunque cada proceso es una entidad independiente, con su propio contador de programa y estado interno, a menudo los procesos necesitan interactuar con otros. Un proceso puede generar cierta salida que otro proceso utiliza como entrada. *Ej cat cap1 cap2 cap3 | grep árbol.* El primer proceso, que ejecuta cat concatena tres archivos y los envía como salida. El segundo proceso, que ejecuta grep, selecciona todas las líneas que contengan la palabra “árbol”. Dependiendo de la velocidad relativa de los dos procesos, puede ocurrir que grep esté listo para ejecutarse, pero que no haya una entrada esperándolo. Entonces debe bloquearse hasta que haya una entrada disponible

Cuando un proceso se bloquea, lo hace debido a que por lógica no puede continuar, comúnmente porque está esperando una entrada que todavía no está disponible. También es posible que un proceso, que esté listo en concepto y pueda ejecutarse, se detenga debido a que el sistema operativo ha decidido asignar la CPU a otro proceso por cierto tiempo. Estas dos condiciones son completamente distintas. En el primer caso, la suspensión está inerte en el problema (no se puede procesar la línea de comandos del usuario sino hasta que este la haya escrito mediante el teclado). En el segundo caso, es un tecnicismo del sistema (no hay suficientes CPUs com para otorgar a cada proceso su propio procesador privado).

Un proceso se puede encontrar en 1 de tres estados:

1. En ejecución (usando la CPU)
2. Listo (ejecutable, se detuvo temporalmente para dejar que se ejecute otro proceso)
3. Bloqueado (no puede ejecutarse sino hasta que ocurra cierto evento externo).

Pueden ocurrir 4 tipos de transiciones:

1. En ejecución → Bloqueado
2. En ejecución → Listo
3. Listo → En ejecución
4. Bloqueado → Listo

En UNIX, cuando un proceso lee datos de un archivo especial (como una terminal) y no hay entrada disponible, el proceso se bloquea en forma automática.

La transición 2 y 3 son producidas por el planificador de procesos, una parte del sistema operativo, sin que el proceso sepa siquiera acerca de ellas. La transición 2 ocurre cuando el planificador decide que el proceso en ejecución se ha ejecutado el tiempo suficiente y es momento de dejar que otro proceso tenga una parte del tiempo de la CPU. La transición 3 ocurre cuando todos los demás procesos han tenido su parte del tiempo de la CPU y es momento de que el primer proceso obtenga la CPU para ejecutarse de nuevo.

La transición 4 ocurre cuando se produce el evento externo por el que un proceso estaba esperando. Si no hay otro proceso en ejecución en ese instante, se activa la transición 3 y el proceso empieza a ejecutarse. En caso contrario, tal vez tenga que esperar en el estado listo por unos instantes, hasta que la CPU esté disponible y sea su turno de utilizarla.

**2.1.6 Implementación de los procesos** (91)

Para implementar el modelo de procesos, el sistema operativo mantiene una tabla (un arreglo de estructuras) llamada tabla de procesos, solo una entrada por cada proceso (algunos autores llaman a esta entradas bloques de control de procesos PCB). Esta entrada contiene información importante acerca del estado del proceso, incluyendo su contador de programa, apuntador de pila, asignación de memoria, estado de sus archivos abiertos, información de contabilidad y planificación y todo lo demás que debe guardarse acerca del proceso cuando este cambia del estado en ejecución a listo o bloqueado, de manera que se pueda reiniciar posteriormente como si nunca se hubiese detenido.

**2.4 PLANIFICACIÓN** (145)

Cuando una computadora se multiprograma, con frecuencia tiene varios procesos o hilos que compiten por la CPU al mismo tiempo. Esta situación ocurre cada vez que dos o más procesos se encuentran al mismo tiempo en el estado listo. Si solo hay una CPU disponible, hay que decidir cuál proceso se va a ejecutar a continuación. La parte del sistema operativo que realiza esa decisión se conoce como planificador de procesos y el algoritmo que utiliza se conoce como algoritmo de planificación.

**Comportamiento de un proceso**

Casi todos los procesos alternan rafagas de calculos con peticiones E/S (de disco). Por lo general la CPU opera durante cierto tiempo sin detenerse, después se realiza una llamada al sistema para leer datos de un archivo o escribirlos en el mismo.

Algunos procesos invierten la mayor parte de su tiempo realizando cálculos (**cpu-bound**), mientras que otros, invierten la mayor parte de su tiempo esperando la ES (**i/o-bound**). Por lo general, los procesos cpu-bound tienen rafagas de CPU largas y en consecuencia, esperas infrecuentes por la E/S, mientras que los procesos limitados a E/S tienen rafagas de CPU cortas y por ende, pas esperas frecuentes de E/S.

La idea básica aquí es que, si un proceso limitado a E/S desea ejecutarse, debe obtener rápidamente la oportunidad de hacerlo para que pueda emitir su petición de disco y mantener el disco ocupado.

**Cuándo planificar procesos**

Hay una variedad de situaciones en las que se necesita la planificación.

En primer lugar, cuando se crea un nuevo proceso se debe tomar una decisión en cuanto a si se debe ejecutar el proceso padre o el proceso hijo, ya que ambos se encuentran en estado de listo puede ejecutarse cualquiera de los dos proceso, el programador puede elegir ejecutar de forma legítima, ya sea el padre o el hijo.

En segundo lugar, se debe tomar una decisión de planificación cuando un proceso termina. Debe elegirse algún otro proceso del conjunto de procesos listos. Si no hay un proceso listo, por lo general se ejecuta un proceso inactivo suministrado por el sistema.

En tercer lugar, cuando un proceso se bloquea por esperar una operación de E/S o por otra razón, hay que elegir otro proceso para ejecutarlo. Algunas veces la razón del bloque puede jugar un papel en la elección. Sin embargo, el problema es que el planificador comúnmente no tiene la información necesaria para tomar en cuenta dependencias.

En cuarto lugar, cuando ocurre una interrupción de E/S tal vez haya que tomar una decisión de planificación. Si la interrupción proviene de un dispositivo de E/S tal vez haya que ha terminado su trabajo, tal vez ahora un proceso que haya estado bloqueado en espera de esa operación de E/S esté listo para ejecutarse.

Si un reloj de hardware proporciona interrupciones periódicas, se puede tomar una decisión de planificación en cada interrupción de reloj. Los algoritmos de planificación se pueden dividir en dos categorías con respecto a la forma en que manejan las interrupciones del reloj. Un algoritmo de programación **no apropiativo** selecciona un proceso para ejecutarlo y después solo deja que se ejecute hasta que el mismo se bloquea o hasta que libera la CPU de forma voluntaria. Incluso aunque se ejecute durante horas, no se suspenderá de forma forzosa. En efecto no se toman decisiones de planificación durante las interrupciones de reloj.

Por el contrario, un algoritmo de planificación **apropiativa** selecciona un proceso y deja que se ejecute por un máximo de tiempo fijo. Si sigue en ejecución al final del intervalo de tiempo, se suspende y el planificador selecciona otro proceso para ejecutarlo (si hay uno disponible). Para llevar a cabo la planificación apropiativa, es necesario que ocurra una interrupción de reloj al final del intervalo de tiempo para que la CPU regrese el control al planificador. Si no hay un reloj disponible, la planificación no apropiativa es la única opción.

**Categorías de los algoritmos de planificación** (149)

Distintos entornos requieren algoritmos de planificación diferentes. Esto quiere decir que lo que el planificador debe optimizar no es lo mismo en todos los sistemas. Tres de los entornos que vale la pena mencionar son:

1. Procesamiento por lotes
2. Interactivo
3. De tiempo real

En los sistemas de procesamiento por lotes no hay usuarios que esperan impacientemente en sus terminales para obtener una respuesta rápida a una petición corta. En consecuencia, son aceptables los algoritmos no apropiativos (o apropiativos con largos periodos para cada proceso). Este método reduce la conmutación de procesos y por ende, mejora el rendimiento.

En un entorno con usuarios interactivos, la apropiación es esencial para evitar que un proceso acapara la CPU y niegue el servicio a los demás.

En los sistemas con restricciones de tiempo real, aunque parezca extraño, la apropiación a veces no es necesaria debido a que los procesos saben que no se pueden ejecutar durante periodos extensos, que por lo general realizan su trabajo y se bloquean con rapidez.

**2.4.2 Planificación en sistemas de procesamiento por lotes** (152)

**Primero en entrar, primero en ser atendido**

FCFS (First-Come, First-Served) algoritmo no apropiativo. Con este algoritmo, la CPU se asigna a los procesos en el orden en el que la solicitan. En esencia hay una sola cola de procesos listos. A cada proceso se le permite ejecutarse todo el tiempo que desee.

**El trabajo más corto primero**

Este algoritmo supone que los tiempos de ejecución se conocen de antemano. Cuando hay varios trabajos de igual importancia esperando a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero (SJF, Shortest Job First).

**El menor tiempo restante a continuación**

Con este algoritmo, el planificador siempre selecciona el proceso cuyo tiempo de ejecución sea el más corto. Se debe conocer el tiempo de ejecución de antemano.

**Planificación por prioridad**

A cada proceso se le asigna una prioridad y el proceso ejecutable cona la prioridad más alta es el que se puede ejecutar

**2.4.3 Planificación en sistemas interactivos**

**Planificación por turno circular (round-robin)**

También conocido como round-robin**.** Uno de los algoritmos más antiguos, simples equitativos y de mayor uso. A cada proceso se le asigna un intervalo de tiempo conocido como quantum, durante el cual se le permite ejecutarse. Si el proceso se sigue ejecutando al final del quantum, la CPU es apropiada para dársela a otro proceso. Si el proceso se bloquea o termina antes de que haya transcurrido el quantum, la conmutación de la CPU se realiza cuando el proceso se bloquea.

Para implementar el algoritmo round-robin solo se necesita mantener una lista de procesos ejecutables. cuando el proceso utiliza su quantum, se coloca al final de la lista.

Si se establece el quantum demasiado corto se producen demasiadas conmutaciones de procesos y se reduce la eficiencia de la CPU, pero si se establece demasiado largo se puede producir una mala respuesta a las peticiones interactivas cortas.

**Colas multinivel**

**El proceso más corto a continuación**

**Planificación garantizada**

**Planificación por sorteo**

**Planificación por partes equitativas**

**Procesos y bloques de control de procesos (PCB)**

Los elementos esenciales de un proceso son: el **código del programa** (que puede compartirse con otros procesos que estén ejecutando el mismo programa), un **conjunto de datos** asociados a dicho código y **Stack(s**) (datos temporarios: parámetros, variables temporales y direcciones de retorno)

**Atributos de un proceso:**

* **Identificadores:**
  + Un identificador del proceso único asociado a este proceso, para distinguirlo del resto de procesos.
  + Identificador del proceso padre.
  + Identificador del usuario que lo disparó.
  + Si hay estructura de grupos, grupo que lo disparó.
  + En ambientes multiusuario, desde que terminal y quien lo ejecutó.
* **Estado.** Si el proceso está actualmente corriendo, está en el estado en ejecución.
* **Prioridad:** Nivel de prioridad relativo al resto del proceso
* **Contador de programa.** La dirección de la siguiente instrucción del programa que se ejecutará.
* **Punteros a memoria.** Incluye los punteros al código de programa y los datos asociados a dicho proceso, además de cualquier bloque de memoria compartido con otro procesos
* **Datos de contexto.** Estos son datos que están presentes en los registros del procesador cuando el proceso está corriendo.
* **Información de estado de E/S.** Incluye las peticiones de E/S pendientes, dispositivos de E/S (por ejemplo, una unidad de cinta) asignados a dicho proceso, una lista de los ficheros en uso por el mismo, etc.
* **Información de auditoría.** Puede incluir la cantidad de tiempo de procesador y de tiempo de reloj utilizados, así como los límites de tiempo, registros contables, etc.

La información de la lista anterior se almacena en una estructura de datos, que se suele llamar Process Control Block (PCB), que el sistema operativo crea y gestiona. El punto más significativo en relación al PCB es que contiene suficiente información de forma que es posible interrumpir el proceso cuándo está corriendo y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna. El PCB es la herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación. Cuándo un proceso se interrumpe, los valores actuales del contador de programa y los registro del procesador (datos de contexto) se guardan en los campos correspondientes del PCB y el estado del proceso de cambia a cualquier otro valor, como bloqueado o listo. El sistema operativo es libre ahora para poner otro proceso en estado de ejecución. El contador de programa y los datos de contexto se recuperan y cargan en los registro del procesador y este proceso comienza a correr.

Existe una por proceso. Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina. Contiene la información asociada con cada proceso.

**El contexto de un proceso**

Incluye toda la información que el SO necesita para administrar el proceso, y la CPU para ejecutarlo correctamente. Son parte del contexto, los registros de cpu, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, etc.

**Cambio de contexto**

Se produce cuando la CPU cambia de un proceso a otro. Se debe resguardar el contexto del proceso saliente, que pasa a espera y retornará después la CPU. Se debe cargar el contexto del nuevo proceso y comenzar desde la instrucción siguiente a la última ejecutada en dicho contexto. Es tiempo no productivo de CPU. El tiempo que consume depende del soporte de HW

**Espacio de direcciones de un proceso**

Es el conjunto de direcciones de memoria que ocupa el proceso (stack, texto y datos) No incluye su PCB o tablas asociadas. Un proceso en modo usuario puede acceder sólo a su espacio de direcciones; En modo kernel, se puede acceder a estructuras internas o a espacios de direcciones de otros procesos.

**Modelo de procesos**

La responsabilidad principal del sistema operativo es controlar la ejecución de los procesos. El primer paso en el diseño de un sistema operativo para el control de procesos es describir el comportamiento que se desea que tengan los procesos.

**Creación y terminación de procesos**

**Creación de un proceso.** Cuándo se va añadir un nuevo proceso a aquellos que se están gestionando en un determinado momento, el sistema operativo construye las estructuras de datos que se usan para manejar el proceso y reserva el espacio de direcciones en memoria principal para el proceso. Estas acciones constituyen la creación de un nuevo proceso.

En UNIX sólo hay una llamada al sistema para crear un proceso: fork. Está llamada crea un clon exacto del proceso que hizo la llamada. Después de fork, los dos procesos (padre e hijo) tienen la misma imagen de memoria, las mismas cadenas de entorno y los mismos archivos abiertos. Eso es todo. Por lo general, el proceso hijo ejecuta después a execve o una llamada al sistema similar para cambiar su imagen de memoria y ejecutar un nuevo programa.

Por el contrario, en Windows una sola llamada a una función de Win32 (CreateProcess) maneja la creación de procesos y carga el programa correcto en el nuevo proceso. Esta llamada tiene 10 parámetros, que incluyen el programa a ejecutar, los parámetros de la línea de comandos para introducir datos a ese programa, varios atributos de seguridad, bits que controlan si los archivos abiertos se heredan, información de prioridad, una especificación de la ventana que se va a crear para el proceso (si se va a crear una) y un apuntador a una estructura en donde se devuelve al proceso que hizo la llamada la información acerca del proceso recién creado. Además de CreateProcess, Win32 tiene cerca de 100 funciones más para administrar y sincronizar procesos y temas relacionados.

Tanto en UNIX como en Windows, una vez que se crea un proceso, el padre y el hijo tienen sus propios espacios de direcciones distintos. Si cualquiera de los procesos modifica una palabra en su espacio de direcciones, esta modificación no es visible para el otro proceso. En UNIX, el espacio de direcciones inicial del hijo es una copia del padre, pero en definitiva hay dos espacios de direcciones distintos involucrados; no se comparte memoria en la que se pueda escribir. Sin embargo, es posible para un proceso recién creado compartir algunos de los otros recursos de su creador, como los archivos abiertos. En Windows, los espacios de direcciones del hijo y del padre son distintos desde el principio.

**Terminación de procesos.** Todo sistema debe proporcionar los mecanismos mediante los cuáles un proceso indica su finalización, o que ha completado su tarea. Para esto se crea una solicitud de un servicio al sistema operativo para terminar con el proceso solicitante. Adicionalmente, un número de error o una condición de fallo puede llevar a la finalización de un proceso.

**Modelo de proceso de cinco estados**

**Ejecutando (elegido por el scheduler de corto plazo).** El proceso está actualmente en ejecución. Asumiendo que el computador tiene un único procesador, de forma que solo un proceso puede estar en este estado en un instante determinado. El scheduler de corto plazo lo eligió para asignarle CPU Tendrá la CPU hasta que se termine el período de tiempo asignado (quantum o time slice), termine o hasta que necesite realizar alguna operación de E/S.

**Listo (elegido por el scheduler de largo plazo).** Un proceso que se prepara para ejecutar cuando tenga oportunidad. Luego que el scheduler de largo plazo eligió al proceso para cargarlo memoria, el proceso queda en estado listo El proceso sólo necesita que se le asigne CPU Está en la cola de procesos listos (ready queue).

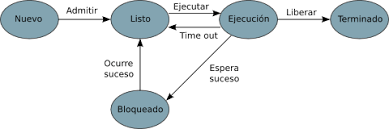
**Bloqueado.** Un proceso que no puede ejecutar hasta que se cumpla un evento determinado o se complete una operación E/S.

El proceso necesita que se cumpla el evento esperado para continuar. El evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso. Sigue en memoria, pero no tiene la CPU. Al cumplirse el evento, pasará al estado de listo.

**Nuevo.** Un proceso que se acaba de crear y que aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo. Típicamente, se trata de un nuevo proceso que no ha sido cargado en memoria principal, aunque su bloque de control de proceso (PCB) si ha sido creado.

Un usuario “dispara” el proceso. Un proceso es creado por otro proceso: su proceso padre. En este estado se crean las estructuras asociadas, y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria.

**Saliente.** Un proceso que ha sido liberado del grupo de procesos ejecutables por el sistema operativo, debido a que ha sido detenido o que ha sido abortado por alguna razón.



Los estados Nuevo y Saliente son útiles para construir la gestión de procesos. El estado Nuevo se corresponde con un proceso que acaba de ser definido. Primero, el sistema operativo realiza todas las tareas internas que correspondan. Se asocia un identificar a dicho proceso. Se reservan y construyen todas aquellas tablas que se necesiten para gestionar al proceso. En este punto, el procesos se encuentra en el estado Nuevo. Esto significa que el sistema operativo ha realizado todas las tareas necesarias para crear el proceso pero el proceso en si, aún no se ha puesto en ejecución. Mientras un procesos está en el estado Nuevo, la información relativa al proceso que se necesite por parte del sistema operativo se mantiene en tablas de control de memoria principal. Sin embargo, el proceso en sí mismo no se encuentra en memoria principal. Esto es, el código de programa a ejecutar no se encuentra en memoria principal, y no se ha reservado ningún espacio para los datos asociados al programa. Cando un proceso se encuentra en estado Nuevo, el programa permanece en almacenamiento secundario, normalmente en disco.

De forma similar, un proceso sale del sistema en dos fases. Primero, el proceso termina cuando alcanza su punto de finalización natural, cuando es abortado debido a un error no recuperable, o cuándo otro proceso con autoridad apropiada causa que el proceso se aborte. La terminación mueve el proceso al estado Saliente. En este punto, el proceso no es elegible de nuevo para su ejecución. Las tablas y otra información asociado con el trabajo se encuentran temporalmente preservadas por el sistema operativo, el cual proporciona tiempo para que programas auxiliares o de soporte extraigan la información necesaria. Una vez que estos programas han extraído la información necesaria, el sistema operativo no necesita mantener ningún dato relativo al proceso y el proceso se borra del sistema.

**Null → Nuevo.** Se crea un nuevo proceso para ejecutar un programa.

**Nuevo → Listo.** El sistema operativo mueve a un proceso del estado nuevo al estado listo cuándo éste se encuentre preparado para ejecutar un nuevo proceso. Por elección del scheduler de largo plazo, se carga en memoria.

**Listo → Ejecutando.** Cuándo llega el momento de seleccionar un nuevo proceso para ejecutar, el sistema operativo selecciona uno de los procesos que se encuentra en el estado Listo. Está es una tarea llevada a cabo por el planificador. Por elección del scheduler de corto plazo, asignación de cpu.

**Ejecutando → Saliente.** El proceso actual en ejecución se finaliza por parte del sistema operativo tanto si el proceso indica que ha completado su ejecución como si este se aborta.

**Ejecutando → Listo.** La razón más habitual para esta transición es que el proceso en ejecución haya alcanzado el máximo tiempo posible de ejecución de forma ininterrumpida.

**Ejecutando → Bloqueado.** Un proceso se pone en el estado bloqueado si solicita algo por lo cual debe esperar.

**Bloqueado → Listo.** Un proceso en estado Bloqueado se mueve al estado Listo cuándo sucede el evento por el cual estaba esperando.

**Listo → Saliente.** Por claridad, esta transición no se muestra en el diagrama de estados. En algunos sistemas, un padre puede terminar la ejecución de un proceso hijo en cualquier momento. También, si el padre termina, todos los procesos hijos asociados con dicho padre pueden finalizarse.

**Bloqueado → Saliente.** Se aplican los comentarios indicados en el caso anterior.

**CREACIÓN DE PROCESOS**

Cuándo el SO decide crear un proceso procederá de la siguiente manera:

1. **Asignar un identificador de proceso único al proceso.** En este instante se añade una nueva entrada a la tabla primaria de procesos, que contiene una entrada por proceso.
2. **Reservar espacio para proceso.** Esto incluye todos los elementos de la imagen del proceso para ello, el sistema operativo debe conocer cuánta memoria se requiere para el espacio de direcciones privado (programas y datos) y para la pila de usuario. Estos valores se pueden asignar por defecto basándonos en el tipo de proceso, o pueden fijarse en base a la solicitud de creación del trabajo remitido por el usuario. Si un proceso es creado por otro proceso, el proceso padre puede pasar los parámetros requeridos por el sistema operativo como parte de la solicitud de la creación de proceso. Si existe una parte del espacio direcciones compartidas por este nuevo proceso, se fijan los enlaces apropiados. Por último, se debe reservar el espacio para el bloque de control de proceso (PCB)
3. **Inicialización del bloque de control de proceso.** La parte de identificación de proceso del PCB contiene el identificador del proceso así como otros posibles identificadores, tal como el indicador del proceso padre. En la información de estado de proceso del PCB, habitualmente se inicializa con la mayoría de entradas a 0, excepto el contador de programa (fijado en el punto entrada del programa) y los puntos de pila del sistema (fijados para definir los límites de la pila del proceso) La parte de información de control de procesos se inicializa en base a los valores por omisión, considerando también los atributos que han sido solicitados para este proceso. Por ejemplo, el estado del proceso se puede inicializar normalmente a Listo o Listo/Suspendido. La prioridad se puede fijar por defecto a la prioridad más baja, a menos que una solicitud explícita la eleve a una prioridad mayor. Inicialmente el proceso no debe poseer ningún recurso (dispositivos de E/S, ficheros) a menos que exista una indicación explícita de ello o que haya sido heredados del padre.
4. **Establecer los enlaces apropiados.** Por ejemplo, si el sistema operativo mantiene cada cola del planificador como una lista enlazada, el nuevo proceso debe situarse en la cola de Listos o en la cola de Listos/Suspendidos.
5. **Creación o expansión de otras estructuras de datos.** Por ejemplo, el sistema operativo puede mantener un registro de auditoría por cada proceso que se puede utilizar posteriormente a efectos de facturación y/o de análisis de rendimiento del sistema.

**Creación de procesos en Unix**

La creación de procesos en UNIX se realiza por medio de la llamada al sistema

fork(). Cuando con un proceso solicita una llamada fork, el sistema operativo realiza las siguientes funciones:

1. Solicita la entrada en la tabla de procesos para el nuevo proceso.
2. Asigna un identificador de proceso único al proceso hijo.
3. Hace una copia de la imagen del proceso padre, con excepción de las regiones de memoria compartidas.
4. Incrementa el contador de cualquier fichero en posesión del padre, para reflejar el proceso adicional que ahora también posee dichos ficheros.
5. Asigna al proceso hijo el estado Listo para Ejecutar.
6. Devuelve el identificador del proceso hijo al proceso padre, y un valor 0 al proceso hijo.

Todo este trabajo se realiza en modo núcleo, dentro del proceso padre. Cuando el núcleo ha completado estas funciones puede realizar cualquiera de las siguientes acciones, como parte de la rutina del activador:

1. Continuar con el proceso padre. El control vuelve a modo usuario en el punto en el que se realizó la llamada fork por parte del padre.
2. Transferir el control al proceso hijo. El proceso hijo comienza ejecutar en el mismo punto del código del padre, es decir en el punto de retorno de la llamada fork
3. Transferir el control a otro proceso. Ambos procesos, padre e hijo, permanecen en el estado Listos para Ejecutar.

Puede resultar quizá un poco difícil visualizar este modo de creación de procesos debido a que ambos procesos, padre e hijo, están ejecutando el mismo segmento de código. La diferencia reside en que: cuando se retorna de la función fork, el parámetro de retorno se comprueba. Si el valor es 0, entonces este es el proceso hijo, y se puede realizar una bifurcación en la ejecución del programa para continuar con la ejecución de programa hijo. Si el valor no es 0, éste es el proceso padre, y puede continuar con la línea principal ejecución.

**Módulos de planificación**

Son módulos (SW) del S.O que realizan distintas tareas asociadas a la planificación. Se ejecutan ante determinados eventos que así lo requieren:

Creación/Terminación de procesos

Eventos de Sincronización o de E/S

Finalización de lapso de tiempo Etc

**Dispatcher:** hace cambio de contexto, cambio de modo de ejecución...”despacha” el proceso elegido por el Short Term (es decir, “salta” a la instrucción a ejecutar).

**Loader:** carga en memoria el proceso elegido por el long term.

**Long Term Scheduler:** Controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria. Puede no existir este scheduler y absorber esta tarea el de short term.

**Medio Term Scheduler:** Si es necesario, reduce el grado de multiprogramación Saca temporariamente de memoria los procesos que sea necesario para mantener el equilibrio del sistema. Términos asociados: swap out (sacar de memoria), swap in (volver a memoria).

**Short Term Scheduler:** Decide a cuál de los procesos en la cola de listos se elige para que use la CPU. Términos asociados: apropiativo, no apropiativo, algoritmo de scheduling.

**Planificación**

Necesidad de determinar cuál de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado.

Algoritmo utilizado para realizar la planificación del sistema.