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.

Características

- Un proceso consta de código, datos y bloque de control de procesos.
- Los procesos existen en una jerarquía de árbol (varios hijos, un sólo padre).
- El sistema asigna un identificador de proceso (***PID***) único al iniciar el proceso.
- El planificador de tareas asigna un tiempo compartido para el proceso según su prioridad.

Tipos de procesos

- **Ejecución en 1*er* plano:** proceso interactivo o iniciado por el usuario.
- **Ejecución en 2*do* plano:** proceso no interactivo que no necesita ser iniciado por el usuario.
- **Demonio:** proceso en 2*o* plano siempre disponible, que da servicio a varias tareas (debe ser propiedad del usuario *root*).
- **Proceso zombie:** proceso parado que queda en la tabla de procesos hasta que termine su padre. Este hecho se produce cuando el proceso padre no recoge el código de salida del proceso hijo.
- **Proceso huérfano:** proceso en ejecución cuyo padre ha finalizado.

La conmutación rápida de un proceso a otro se conoce como multiprogramación. En la figura 1 podemos ver que durante un intervalo suficientemente largo todos los procesos han progresado, pero en cualquier momento dado sólo hay un proceso en ejecución.

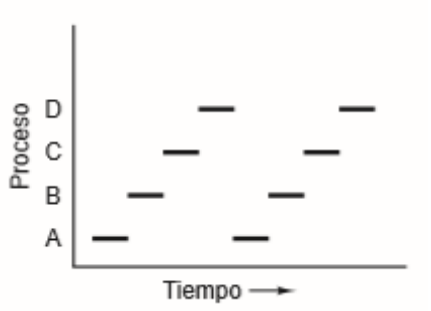

Figura 1. Procesos en Multiprogramación

Dado que la CPU conmuta rápidamente entre un proceso y otro, la velocidad a la que un proceso ejecuta sus cálculos no es uniforme y tal vez ni siquiera sea reproducible si se ejecutan los mismos procesos de nuevo.

La diferencia entre un proceso y un programa es sutil pero crucial. Aquí podría ayudarnos una analogía: un científico computacional con mente culinaria hornea un pastel de cumpleaños para su hija; tiene la receta para un pastel de cumpleaños y una cocina bien

equipada con todos los ingredientes: harina, huevos, azúcar, extracto de vainilla, etcétera. En esta analogía, la receta es el programa (es decir, un algoritmo expresado en cierta notación adecuada), el científico computacional es el procesador (CPU) y los ingredientes del pastel son los datos de entrada. El proceso es la actividad que consiste en que nuestro cocinero vaya leyendo la receta, obteniendo los ingredientes y horneando el pastel.

Ahora imagine que el hijo del científico entra corriendo y gritando que una abeja acaba de picarlo. El científico computacional registra el punto de la receta en el que estaba (el estado del proceso en curso se guarda), saca un libro de primeros auxilios y empieza a seguir las instrucciones que contiene. Aquí el procesador conmuta de un proceso (hornear el pastel) a uno de mayor prioridad (administrar cuidados médicos), cada uno de los cuales tiene un programa distinto (la receta y el libro de primeros auxilios). Cuando ya se ha ocupado de la picadura de abeja, el científico computacional regresa a su pastel y continúa en el punto en el que se había quedado. La idea clave es que un proceso es una actividad de cierto tipo: tiene un programa, una entrada, una salida y un estado.

Varios procesos pueden compartir un solo procesador mediante el uso de un algoritmo de planificación para determinar cuándo se debe detener el trabajo en un proceso para dar servicio a otro. Vale la pena recalcar que si un programa se está ejecutando por duplicado cuenta como dos procesos. Por ejemplo, a menudo es posible iniciar un procesador de palabras dos veces o imprimir dos archivos al mismo tiempo si hay dos impresoras disponibles. El hecho de que dos procesos en ejecución tengan el mismo programa no importa; **son procesos distintos**. El sistema operativo puede compartir el código entre ellos de manera que sólo haya una copia en la memoria, pero ése es un detalle técnico que no cambia la situación conceptual de dos procesos en ejecución.

Ciclo de vida de un proceso

Creación de un proceso

Para crear un proceso hay que hacer que un proceso existente ejecute una llamada al sistema de creación de proceso. 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, cuál programa debe ejecutarlo.

Hay cuatro eventos principales que provocan la creación de procesos:

1. **El arranque del sistema.** 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 (humanos) y realizan trabajo para ellos. Otros son procesos en segundo plano, que no están asociados con usuarios específicos sino con una función específica.
2. **La ejecución, desde un proceso, de una llamada al sistema para creación de procesos.** Además de los procesos que se crean al arranque, posteriormente se pueden crear otros. A menudo, un proceso en ejecución emitirá llamadas al sistema para crear uno o más procesos nuevos, para que le ayuden a realizar su trabajo. En especial, es útil crear procesos cuando el trabajo a realizar se puede formular fácilmente en términos de varios procesos interactivos relacionados entre sí, pero

independientes en los demás aspectos. Por ejemplo, si se va a obtener una gran cantidad de datos a través de una red para su posterior procesamiento, puede ser conveniente crear un proceso para obtener los datos y colocarlos en un búfer compartido, mientras un segundo proceso remueve los elementos de datos y los procesa

3. **Una petición de usuario para crear un proceso.** los usuarios pueden iniciar un programa escribiendo un comando o haciendo (doble) clic en un icono. En los sistemas modernos, los usuarios pueden tener varias ventanas abiertas a la vez, cada una ejecutando algún proceso. Mediante el ratón, el usuario puede seleccionar una ventana e interactuar con el proceso, por ejemplo para proveer datos cuando sea necesario.
4. **El inicio de un trabajo por lotes.** se aplica sólo a los sistemas de procesamiento por lotes que se encuentran en las mainframes grandes. Aquí los usuarios pueden enviar trabajos de procesamiento por lotes al sistema (posiblemente en forma remota). Cuando el sistema operativo decide que tiene los recursos para ejecutar otro trabajo, crea un proceso y ejecuta el siguiente trabajo de la cola de entrada.

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 Espacio de direcciones (rango de memoria válido) El hijo es un duplicado del padre (UNIX), pero cada uno tiene su propia memoria física. Además, padre e hijo, en el momento de la creación, tienen el mismo contexto de ejecución (los registros de la CPU valen lo mismo). El hijo ejecuta un código diferente.

Un proceso puede crear un proceso hijo, y este a su vez otros procesos, obteniéndose un árbol de procesos. En UNIX todos los procesos de usuario son parte de un árbol cuya cabeza es el proceso **init**. De esta forma el Shell es hijo de init, y los procesos de usuario los nietos, y todos los procesos en el sistema son parte de un mismo árbol.

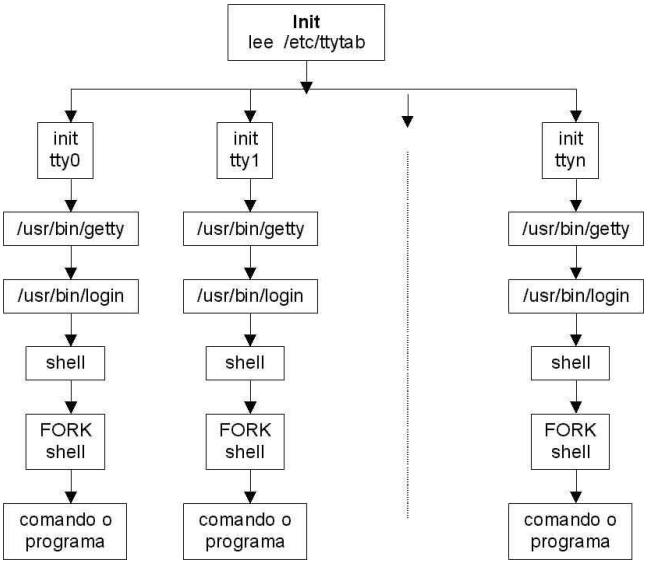

Figura 2. Árbol de procesos en UNIX

Cuando se ejecuta un programa, se le asigna un trozo de memoria, que la retiene a la largo de su vida y cuyo tamaño está especificado en el encabezamiento del programa. Toda la información de un proceso creado se guarda en la **tabla de procesos**, dividida en tres partes el núcleo, manejo de memoria y sistema de ficheros.

Terminación de procesos

Una vez que se crea un proceso, empieza a ejecutarse y realiza el trabajo al que está destinado. Sin embargo, nada dura para siempre, ni siquiera los procesos. Tarde o temprano el nuevo proceso terminará, por lo general debido a una de las siguientes condiciones:

1. **Salida normal (voluntaria).** 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. Los procesadores de palabras, navegadores de Internet y programas similares siempre tienen un icono o elemento de menú en el que el usuario puede hacer clic para indicar al proceso que elimine todos los archivos temporales que tenga abiertos y después termine.
2. **Salida por error (voluntaria).** La segunda razón de terminación es que el proceso descubra un error. Por ejemplo, si un usuario escribe el comando foo.c para compilar el programa foo.c y no existe dicho archivo, el compilador simplemente termina. Los procesos interactivos orientados a pantalla por lo general no terminan cuando reciben parámetros incorrectos. En vez de ello, aparece un cuadro de diálogo y se le pide al usuario que intente de nuevo
3. **Error fatal (involuntaria).** La tercera razón de terminación es un error fatal producido por el proceso, a menudo debido a un error en el programa. Algunos ejemplos incluyen el ejecutar una instrucción ilegal, hacer referencia a una parte de memoria no existente o la división entre cero. En algunos sistemas (como UNIX), un proceso puede indicar al sistema operativo que desea manejar ciertos errores por sí mismo, en cuyo caso el proceso recibe una señal (se interrumpe) en vez de terminar.
4. **Eliminado por otro proceso (involuntaria).** La cuarta razón por la que un proceso podría terminar es que ejecute una llamada al sistema que indique al sistema operativo que elimine otros procesos. En UNIX esta llamada es kill. La función correspondiente en Win32 es TerminateProcess. En ambos casos, el proceso eliminador debe tener la autorización necesaria para realizar la eliminación.

Jerarquías de procesos

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.

Veamos la forma en que UNIX se inicializa a sí mismo cuando se enciende la computadora. Hay un proceso especial (llamado init) en la imagen de inicio. Cuando empieza a ejecutarse, lee un archivo que le indica cuántas terminales hay. Después utiliza fork para crear un proceso por cada terminal. Estos procesos esperan a que alguien inicie la sesión. Si un inicio de sesión tiene éxito, el proceso de inicio de sesión ejecuta un shell para aceptar comandos. Éstos pueden iniciar más procesos y así sucesivamente. Por ende, todos los procesos en el sistema completo pertenecen a un solo árbol, con init en la raíz.

En contraste, Windows no tiene un concepto de una jerarquía de procesos. Todos los procesos son iguales. La única sugerencia de una jerarquía de procesos es que, cuando se crea un proceso, el padre recibe un indicador especial un token (llamado manejador) que puede utilizar para controlar al hijo. Sin embargo, tiene la libertad de pasar este indicador a otros procesos, con lo cual invalida la jerarquía. Los procesos en UNIX no pueden desheredar a sus hijos.

Implementación de los procesos

Para implementar el modelo de procesos, el sistema operativo mantiene una tabla (un arreglo de estructuras) llamada **tabla de procesos**, con sólo una entrada por cada proceso (algunos autores llaman a estas entradas **bloques de control de procesos**). 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 éste cambia del estado en ejecución a listo o bloqueado, de manera que se pueda reiniciar posteriormente como si nunca se hubiera detenido.

La figura 3 muestra algunos de los campos clave en un sistema típico. Los campos en la primera columna se relacionan con la administración de procesos; los otros dos se relacionan con la administración de memoria y archivos, respectivamente. Hay que recalcar que los campos contenidos en la tabla de procesos varían de un sistema a otro, pero esta figura nos da una idea general de los tipos de información necesaria.

Administración de procesos	**Administración de memoria**	**Administración de archivos**
Registros	Apuntador a la información del segmento de texto	Directorio raíz
Contador del programa	Apuntador a la información del segmento de datos	Directorio de trabajo
Palabra de estado del programa	Apuntador a la información del segmento de pila	Descripciones de archivos
Apuntador de la pila		ID de usuario
Estado del proceso		ID de grupo
Prioridad		
Parámetros de planificación		
ID del proceso		
Proceso padre		
Grupo de procesos		
Señales		
Tiempo de inicio del proceso		
Tiempo utilizado de la CPU		
Tiempo de la CPU utilizado por el hijo		
Hora de la siguiente alarma		

Figura 3. Bloque de control de procesos.

Cuando el proceso termina, su BCP es borrado y el registro puede ser utilizado para otros procesos. Un proceso resulta conocido para el sistema operativo y por tanto elegible para competir por los recursos del sistema sólo cuando existe un BCP activo asociado a él. El bloque de control de proceso es una estructura de datos con campos para registrar los diferentes aspectos de la ejecución del proceso y de la utilización de recursos.

Estados 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. En el comando de Shell 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 "arbol". Dependiendo de la velocidad relativa de los dos procesos (que dependen tanto de la complejidad relativa de los programas, como de cuánto tiempo ha tenido cada uno la CPU), puede ocurrir que grep esté listo para ejecutarse, pero que no haya una entrada esperándolo. Entonces debe bloquear 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 inherente en el problema (no se puede procesar la línea de comandos del usuario sino hasta que éste la haya escrito mediante el teclado). En el segundo caso, es un tecnicismo del sistema (no hay suficientes CPUs como para otorgar a cada proceso su propio procesador privado). En la figura 4 podemos ver un diagrama de estados que muestra los tres estados en los que se puede encontrar un proceso:

1. En ejecución (en realidad está usando la CPU en ese instante).
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).

En sentido lógico, los primeros dos estados son similares. En ambos casos el proceso está deseoso de ejecutarse; sólo en el segundo no hay temporalmente una CPU para él. El tercer estado es distinto de los primeros dos en cuanto a que el proceso no se puede ejecutar, incluso aunque la CPU no tenga nada que hacer.

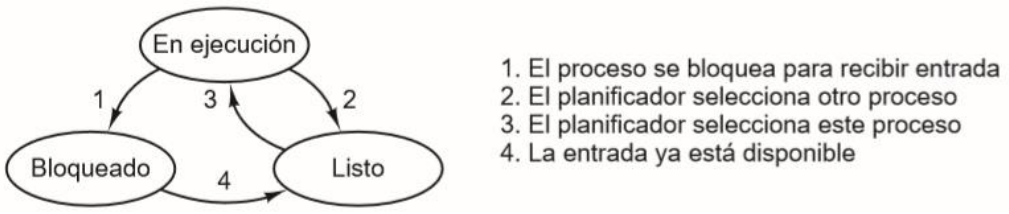

Figura 4. Estado de los procesos y las transiciones posibles

Hay cuatro transiciones posibles entre estos tres estados, como se indica. La transición 1 ocurre cuando el sistema operativo descubre que un proceso no puede continuar justo en ese momento. En algunos sistemas el proceso puede ejecutar una llamada al sistema, como pause, para entrar al estado bloqueado. En otros sistemas, incluyendo a UNIX, cuando un proceso lee datos de una canalización o de un archivo

especial (como una terminal) y no hay entrada disponible, el proceso se bloquea en forma automática.

Las transiciones 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. El tema de la planificación de procesos (decidir qué proceso debe ejecutarse en qué momento y por cuánto tiempo) es importante; más adelante lo analizaremos.

La transición 4 ocurre cuando se produce el evento externo por el que un proceso estaba esperando (como la llegada de ciertos datos de entrada). 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. Si utilizamos el modelo de los procesos, es mucho más fácil pensar en lo que está ocurriendo dentro del sistema.

Algunos de los procesos ejecutan programas que llevan a cabo los comandos que escribe un usuario; otros son parte del sistema y se encargan de tareas como cumplir con las peticiones de los servicios de archivos o administrar los detalles de ejecutar una unidad de disco. Cuando ocurre una interrupción de disco, el sistema toma una decisión para dejar de ejecutar el proceso actual y ejecutar el proceso de disco que está bloqueado esperando esta interrupción. Así, en vez de pensar en las interrupciones, podemos pensar en los procesos de usuario, procesos de disco, procesos de terminal, etc., que se bloquean cuando están esperando a que algo ocurra. Cuando se ha leído el disco o se ha escrito el carácter, el proceso que espera se desbloquea y es elegible para continuar ejecutándose.

Modelo de proceso de cinco estados

Para gestionar correctamente el control de los procesos, se han añadido dos estados adicionales Nuevo y saliente que resultarán muy útiles. Estos cinco estados en el nuevo diagrama son los siguientes:

• **Ejecutando.** El proceso está actualmente en ejecución. Asumimos que el computador tiene un único procesador, de forma que sólo un proceso puede estar en este estado en un instante determinado.

• **Listo.** Un proceso que se prepara para ejecutar cuando tenga oportunidad.

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

• **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 (BCP) si ha sido creado.

• **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. Por ejemplo, si un nuevo usuario intenta entrar dentro de un sistema de tiempo compartido o cuando se solicita un nuevo trabajo a un sistema de proceso por lotes, el sistema operativo puede definir un nuevo proceso en dos etapas. Primero, el sistema operativo realiza todas las tareas internas que correspondan. Se asocia un identificador a dicho proceso. Se reservan y construyen todas aquellas tablas que se necesiten para gestionar al proceso. En este punto, el proceso se encuentra el estado Nuevo. Esto significa que el sistema operativo ha realizado todas las tareas necesarias para crear el proceso pero el proceso en sí, aún no se ha puesto en ejecución.

Por ejemplo, un sistema operativo puede limitar el número de procesos que puede haber en el sistema por razones de rendimiento o limitaciones de memoria principal. Mientras un proceso 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. Cuando un proceso se encuentra en el 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 cuando 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 asociada 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.

Por ejemplo, un programa de auditoría puede requerir registrar el tiempo de proceso y otros recursos utilizados por este proceso saliente con objeto de realizar una contabilidad de los recursos del sistema. Un programa de utilidad puede requerir extraer información sobre el histórico de los procesos por temas relativos con el rendimiento o análisis de la utilización. 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. La Figura 5 indica que tipos de eventos llevan a cada transición de estado para cada proceso; las posibles transiciones son las siguientes:

Figura 5. Modelo de Procesos de 5 estados

• Null -> Nuevo. Se crea un nuevo proceso para ejecutar un programa. Este evento ocurre por cualquiera de las relaciones ya indicadas

• Nuevo -> Listo. El sistema operativo mueve a un proceso del estado nuevo al estado listo cuando éste se encuentre preparado para ejecutar un nuevo proceso. La mayoría de sistemas fijan un límite basado en el número de procesos existentes o la cantidad de memoria virtual que se podrá utilizar por parte de los procesos existentes. Este límite asegura que no haya demasiados procesos activos y que se degrade el rendimiento sistema.

• Listo -> Ejecutando. Cuando llega el momento de seleccionar un nuevo proceso para ejecutar, el sistema operativo selecciona uno los procesos que se encuentre en el estado Listo. Esta es una tarea la lleva acabo el planificador.

• 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 éste 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; prácticamente todos los sistemas operativos multiprogramados imponen este tipo de restricción de tiempo. Existen otras posibles causas alternativas para esta transición, que no están incluidas en todos los sistemas operativos. Es de particular importancia el caso en el cual el sistema operativo asigna diferentes niveles de prioridad a diferentes procesos.

Supóngase, por ejemplo, que el proceso A está ejecutando a un determinado nivel de prioridad, y el proceso B, a un nivel de prioridad mayor, y que se encuentra bloqueado. Si el sistema operativo se da cuenta de que se produce un evento al cual el proceso B está esperando, moverá el proceso B al estado de Listo. Esto puede interrumpir al proceso A y poner en ejecución al proceso B. Decimos, en este caso, que el sistema operativo ha expulsado al proceso A. Adicionalmente, un proceso puede voluntariamente dejar de utilizar el procesador. Un ejemplo son los procesos que realiza alguna función de auditoría o de mantenimiento de forma periódica.

• Ejecutando -> Bloqueado. Un proceso se pone en el estado Bloqueado si solicita algo por lo cual debe esperar. Una solicitud al sistema operativo se realiza habitualmente por medio de una llamada al sistema; esto es, una llamada del proceso en ejecución a un procedimiento que es parte del código del sistema operativo. Por ejemplo, un proceso ha solicitado un servicio que el sistema operativo no puede realizar en ese momento. Puede solicitar un recurso, como por ejemplo un fichero o una sección compartida de memoria virtual, que no está inmediatamente disponible. Cuando un proceso quiere iniciar una acción, tal como una operación de E/S, que debe completarse antes de que el proceso continúe. Cuando un proceso se comunica con otro, un proceso puede bloquearse mientras está esperando a que otro proceso le proporcione datos o esperando un mensaje de ese otro proceso.

• Bloqueado -> Listo. Un proceso en estado Bloqueado se mueve al estado Listo cuando 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.

La figura 6 sugiere la forma de aplicar un esquema de dos colas: la colas de Listos y la cola de Bloqueados. Cada proceso admitido por el sistema, se coloca en la cola de Listos. Cuando llega el momento de que el sistema operativo seleccione otro proceso a ejecutar, selecciona uno de la cola de Listos. En ausencia de un esquema de prioridad, esta cola

puede ser una lista de tipo FIFO (first-in-first-out). Cuando el proceso en ejecución termina de utilizar el procesador, o bien finaliza o bien se coloca en la cola de Listos o de Bloqueados, dependiendo de las circunstancias. Por último, cuando sucede un evento, cualquier proceso en la cola de Bloqueados que únicamente esté esperando a dicho evento, se mueve a la cola de Listos. Esta última transición significa que, cuando sucede un evento, el sistema operativo debe recorrer la cola entera de Bloqueados, buscando aquellos procesos que estén esperando por dicho evento.

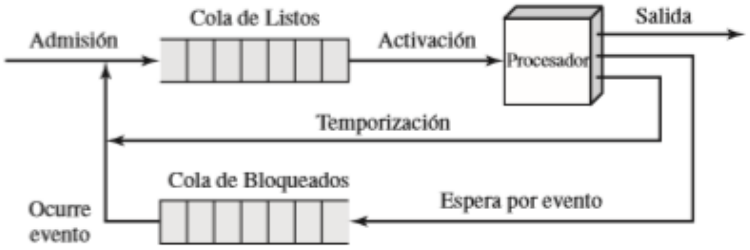

Figura 6. Cola simple de bloqueados

En los sistemas operativos con muchos procesos, esto puede significar cientos o incluso miles de procesos en esta lista, por lo que sería mucho más eficiente tener una cola por cada evento. De esta forma, cuando sucede un evento, la lista entera de procesos de la cola correspondiente se movería al estado de Listo como se ve en ña figura 7.

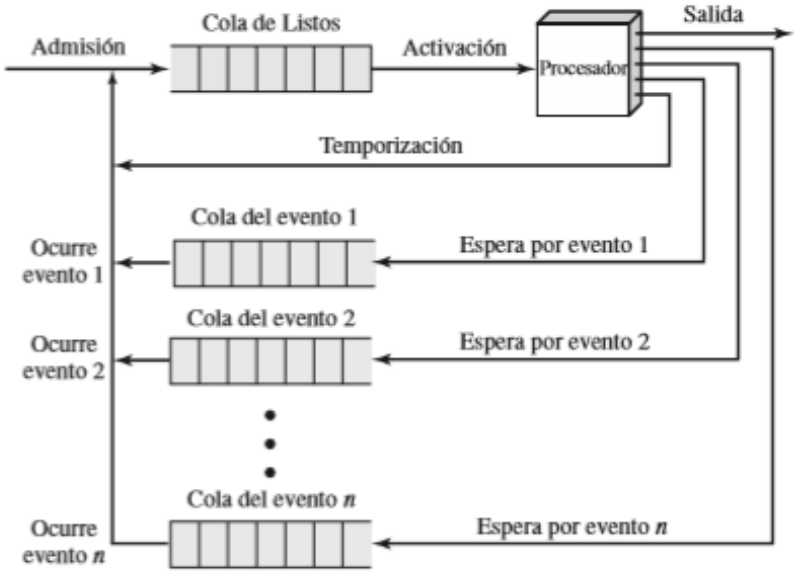

Figura 7. Múltiples colas de bloqueados

Un refinamiento final sería: si la activación de procesos está dictada por un esquema de prioridades, sería conveniente tener varias colas de procesos listos, una por cada nivel de

prioridad. El sistema operativo podría determinar cuál es el proceso listo de mayor prioridad simplemente seleccionando éstas en orden.

Investigar: *Llamadas al sistema para gestión de procesos concurrentes en UNIX. Fork, exit, wait y exec. Sintaxis y funcionamiento.*