Control y Descripción de Procesos II

Modelo de procesos con dos estados

- Un proceso puede estar en uno de estos dos estados
  - Ejecución (Running)
  - No-Ejecución (Not-running)

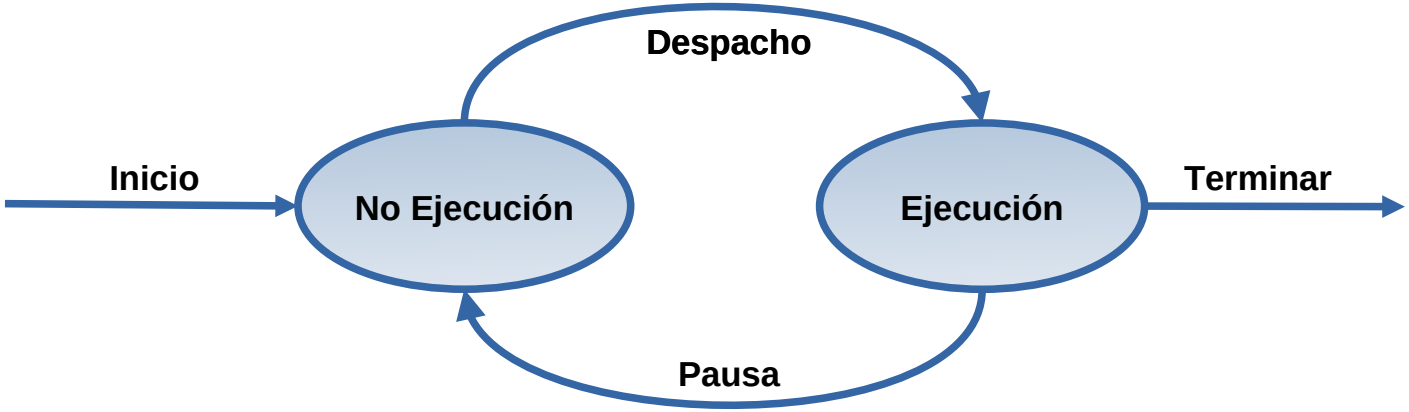


Diagrama de Transición de Estados

Cola de procesos en No ejecución

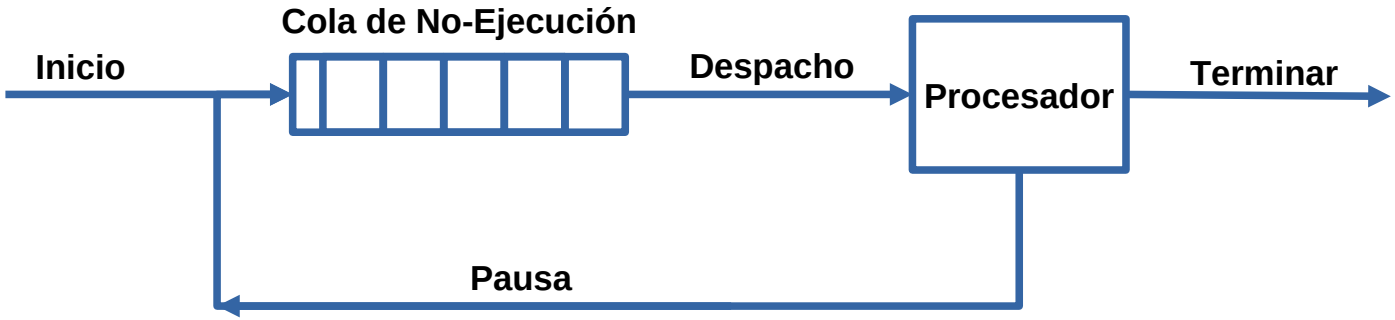


Diagrama de Colas

- La cola es una lista FIFO, (First-in, First-Out)
- Cada bloque de la cola representa a un proceso
- El comportamiento del despachador se puede describir en términos de un diagrama de colas
- El procesador opera según un turno rotatorio (round-robin)

Creación de un proceso

- En un entorno de trabajo por lotes, un proceso se crea como respuesta a la remisión de un trabajo.
- En un entorno interactivo, se crea un proceso cuando un nuevo usuario intenta conectarse.
- El sistema operativo crea un proceso para dar servicio a una aplicación sin que el usuario tenga que esperar, Ej. imprimir.
- Un proceso puede originar la creación de otro proceso *(process spawning)*, Ejemplo: cuando se escribe en la linea de comandos el nombre de un programa, la shell crea un proceso para ejecutarlo.

Terminación de un proceso

Terminación normal	El proceso ejecuta una llamada a un servicio del SO que indica que se ha terminado de ejecutar
Tiempo limite excedido	El proceso se ha ejecutado mas que el limite especificado. Hay varias posibilidades para la clase de tiempo que se mide. Entre éstas se incluyen el tiempo total transcurrido (“tiempo de reloj”), el tiempo que se ha estado ejecutando (“tiempo de CPU”) y, en el caso de un proceso interactivo, el tiempo transcurrido desde que el usuario real realizó su última entrada de datos
No hay memoria disponible	El proceso necesita más memoria de la que el sistema le puede proporcionar
Violación de limites	El proceso trata de acceder a una posición de memoria a la que no le está permitido acceder
Error de protección	El proceso intenta utilizar un recurso o un archivo que no le está permitido utilizar, o trata de utilizarlo de forma incorrecta, como escribir en un archivo que es solo de lectura.
Error Aritmético	El proceso intenta hacer un cálculo prohibido, como una división por cero, o trata de almacenar un número mayor del que el hardware acepta.

Terminación de un proceso

Tiempo máx. de espera rebasado	El proceso ha esperado más allá del tiempo máximo especificado para que se produzca cierto suceso.
Fallo de E/S	Se produce un error en la entrada o la salida, tal como la incapacidad de encontrar un archivo, un fallo de r/w después de un número máximo de intentos (cuando, por ejemplo, hay un región defectuosa en una cinta), o una operación ilegal (como intentar leer de una impresora)
Instrucción inválida	El proceso intenta ejecutar una instrucción inexistente (a menudo como resultado de un salto a una zona de datos para intentar ejecutar los datos).
Instrucción privilegiada	El proceso intenta usar una instrucción reservada para el sistema operativo
Mal uso de los datos	Un elemento de dato es de un tipo equivocado o no está inicializado.
Intervención del operador o del SO	Por alguna razón el operador o el sistema operativo termina con el proceso (por ejemplo, si existe un interbloqueo).

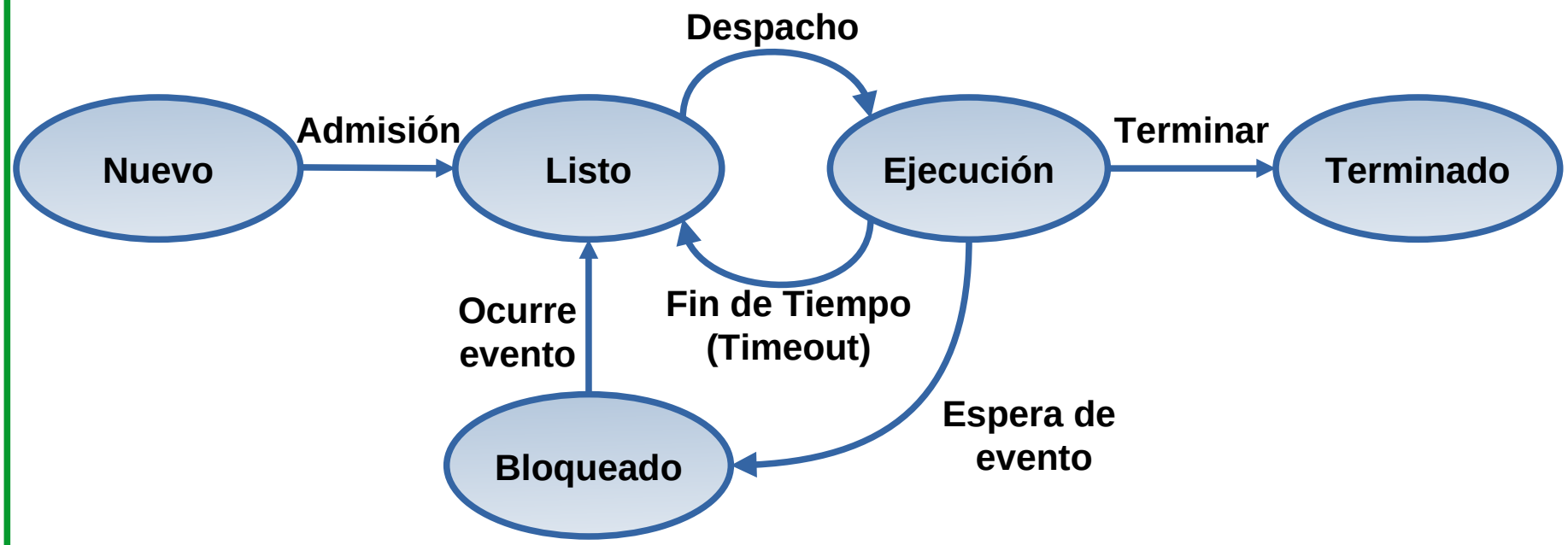
Terminación de un proceso

Terminación del padre	Cuando un proceso padre finaliza, el sistema operativo puede diseñarse para terminar automáticamente con todos sus descendientes.
• **Solicitud del padre**	Un proceso padre tiene normalmente la autoridad de terminar con cualquiera de sus descendientes

Limitaciones del modelo

- Dentro de los procesos en estado de No-Ejecución tenemos
  - Listos para ejecutar
  - Bloqueados (Esperando que termine una operación de E/S).
- El despachador debe seleccionar un proceso que no esté bloqueado.
- El despachador tendría que recorrer la lista buscándolo y eso lleva tiempo adicional.
- Eso no conviene, el despachador debe realizar el intercambio de tareas lo más rapido posible.

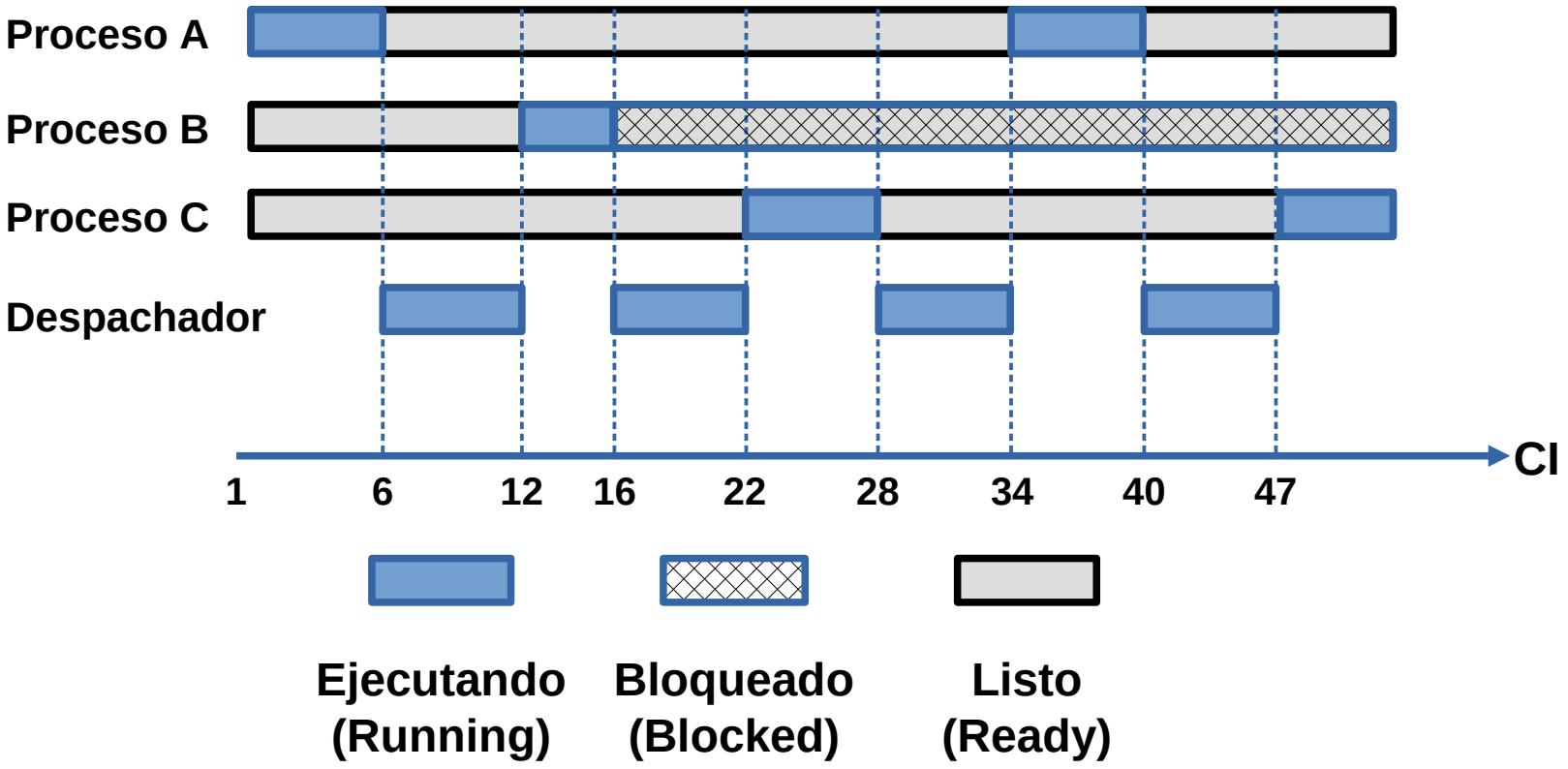
Modelo de cinco estados



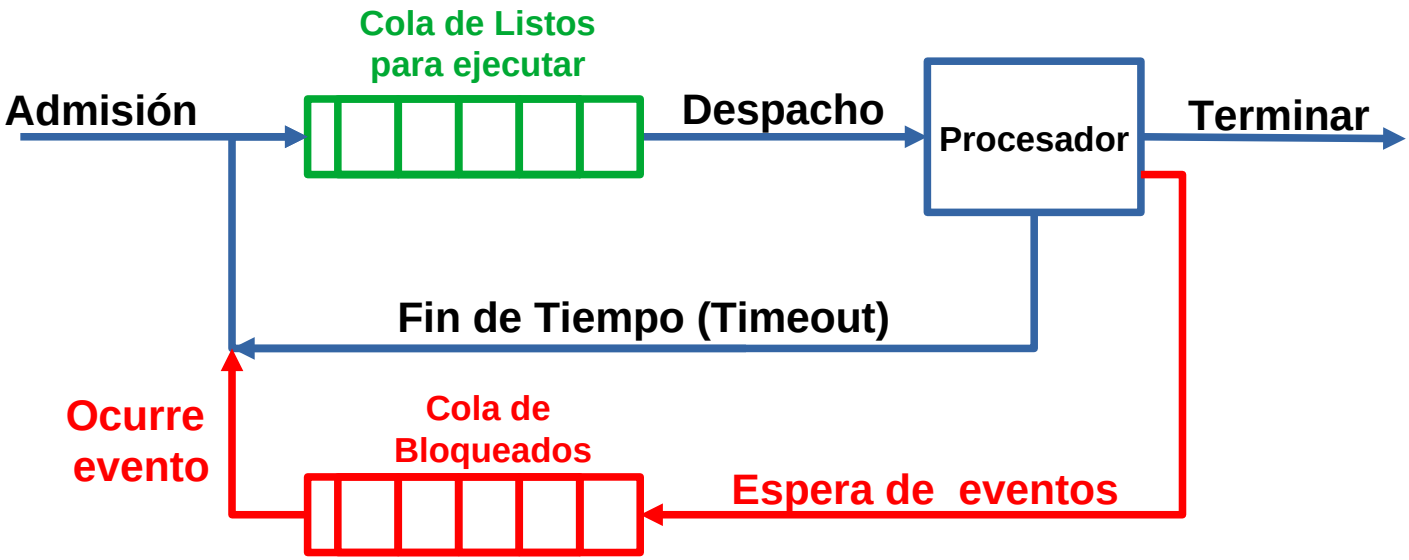
Modelo de cinco estados

- Ejecución (Running)
- Listo para ejecutar (Ready)
- Bloqueado (Blocked)
- Nuevo (New)
- Terminado (Exit)

Estados de un Proceso

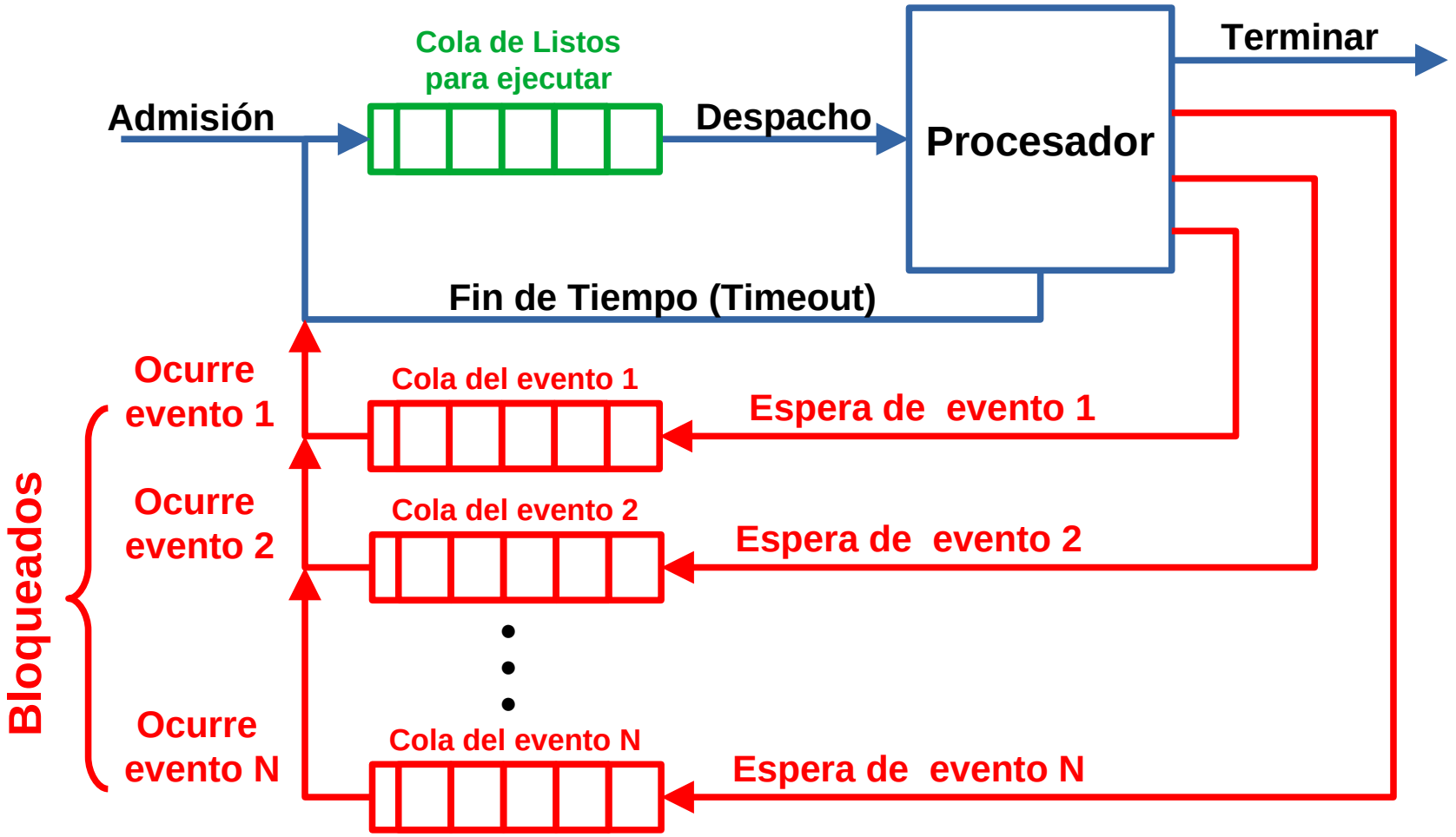


Modelo de cinco estados usando colas

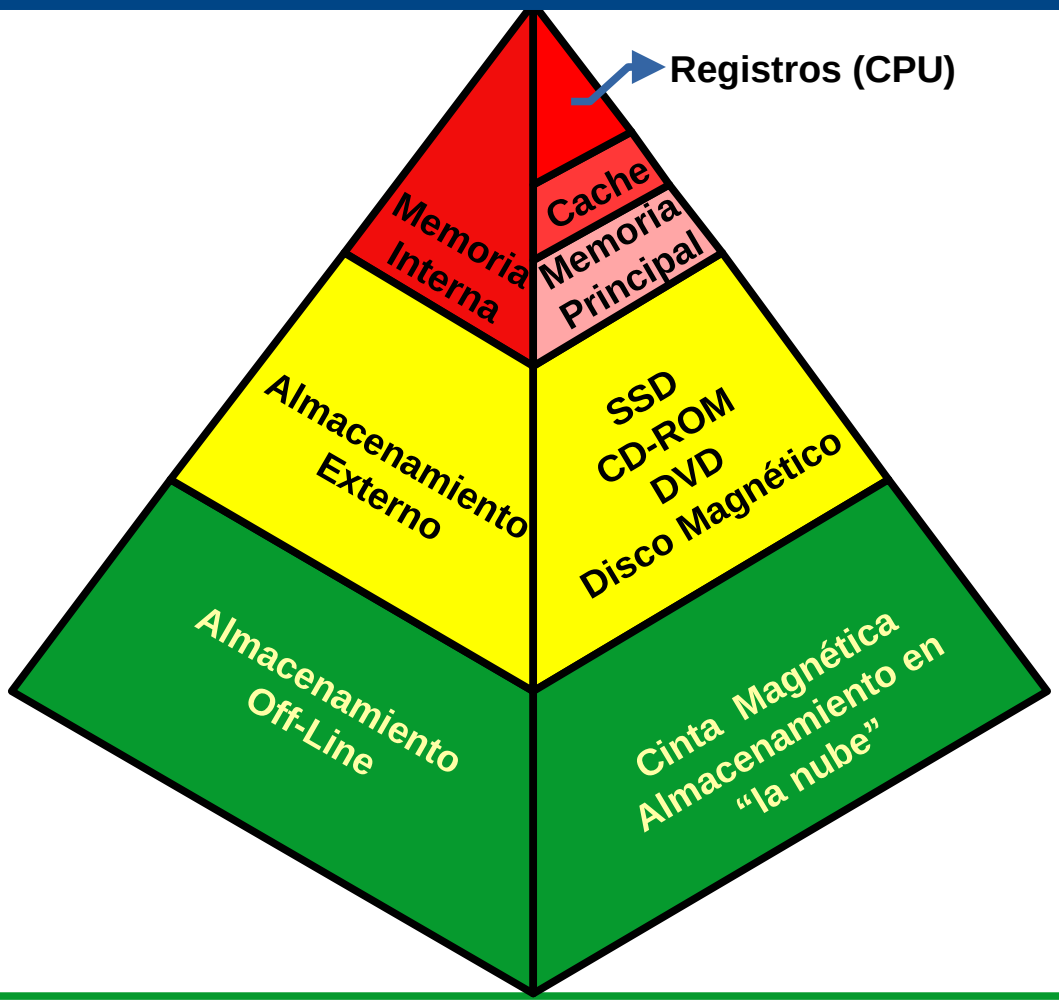


- Ahora el despachador, siempre tiene un proceso listo para ejecutar.
- Pero cuando se produce un evento, el planificador del SO debe recorrer toda la Cola de Bloqueados buscando aquellos procesos que esperan al evento
- Podemos hacer una Cola de Bloqueados para cada evento, cuando ocurre el evento, el planificador del SO directamente toma el proceso mas antiguo .

Múltiples colas de Bloqueados



Jerarquías de Memoria



Memoria Interna:
Muy rápida – costosa – escasa

Almacenamiento Externo
Más lento – Costo medio – abundante

Almacenamiento OFF-Line
Muy lento – Costo ínfimo –

Capacidad → ∞

Jerarquias de Memoria

- A medida que se desciende en la jerarquía se tiene:
  - Costo por bit decreciente
  - Capacidad creciente
  - Tiempo de acceso creciente
  - Frecuencia de acceso por el procesador decreciente

Tiempo de acceso -Throughput

Tipo de almacenamiento	Tiempo de Acceso	Throughput	Tamaño
Registros	< 1ns	≈ 1 TiB/s	< 1KiB
Cache	1 – 10 ns	≈ 400 GiB/s	< 16 MiB
Memoria	< 10 ns	≈ 3 Gib/s	< 128 GiB
Disco SSD	< 100 us	≈ 1000 MiB/s	< 10TiB
Disco Magnético	< 10 ms	≈ 1000 MiB/s	< 10 TiB
Cinta – Red	> 10 ms	• ≈ 400 MiB/s	> 10 TiB

Memoria Virtual

- El procesador es tan rápido comparado con la I/O que todos los procesos en memoria podrían estar bloqueados
- El intercambio (Swap) de estos procesos a Memoria secundaria (Disco) podría liberar memoria para otro proceso o para crear uno nuevo.
- El estado bloqueado se transforma a suspendido cuando el proceso está en Memoria Secundaria.

Memoria Virtual

- En la gran mayoria de los SO actuales esto se realiza mediante el uso de ***Memoria Virtual***
- Desde el punto de vista de la aplicación, esta ve un espacio de memoria al que puede acceder a través de direcciones lógicas (*logical addresses*).
- Desde el punto de vista del SO, se divide la memoria física en porciones que dependiendo del Hardware pueden ser segmentos *(segments)* o páginas *(pages).* Estas porciones de memoria tendrán asignadas direcciones fisicas, que son las que realmente accede el procesador.

Memoria Virtual

- Las *direcciones lógicas* están descriptas por un *segmento* o por una *página* y dentro de éstos por un *offset* o *desplazamiento.*
  - *Este sistema independiza a los procesos de las ubicaciones físicas reales, y permite la reubicación de los procesos.*
- En la gran mayoría de los casos donde se usa memoria virtual existe un dispositivo de hardware llamado unidad de administración de Memoria *(Memory Management Unit - MMU)* que se ocupa de el manejo de las conversiones.
- Las páginas pueden almacenarse en memoria principal o en memoria secundaria (disco).
  - Memoria principal, los procesos pueden acceder a ellas directamente.
  - Memoria Secundaria: para acceder, primero deben cargarse en Memoria Principal

Memoria Virtual

- Acceso del proceso a Memoria Virtual:
  - Si un proceso trata de acceder a una dirección lógica, la MMU verifica cual es la dirección física correspondiente, pueden darse tres casos.
  - La dirección corresponde a una página presente en la memoria principal. En ese caso se accede directamente.
  - La dirección lógica es válida y no está presente en la memoria principal. En ese caso se genera un *Page Fault,* y la MMU traerá la página faltante a la memoria principal.
    - Si hay espacio suficiente, directamente la carga en memoria
    - Si no hay espacio suficiente, previamente mueve otra pagina desde memoria principal hacia memoria secundaria.
  - La dirección lógica no es válida o intenta acceder a un espacio no permitido. La MMU generará un aviso al SO, que probablemente termine el programa.

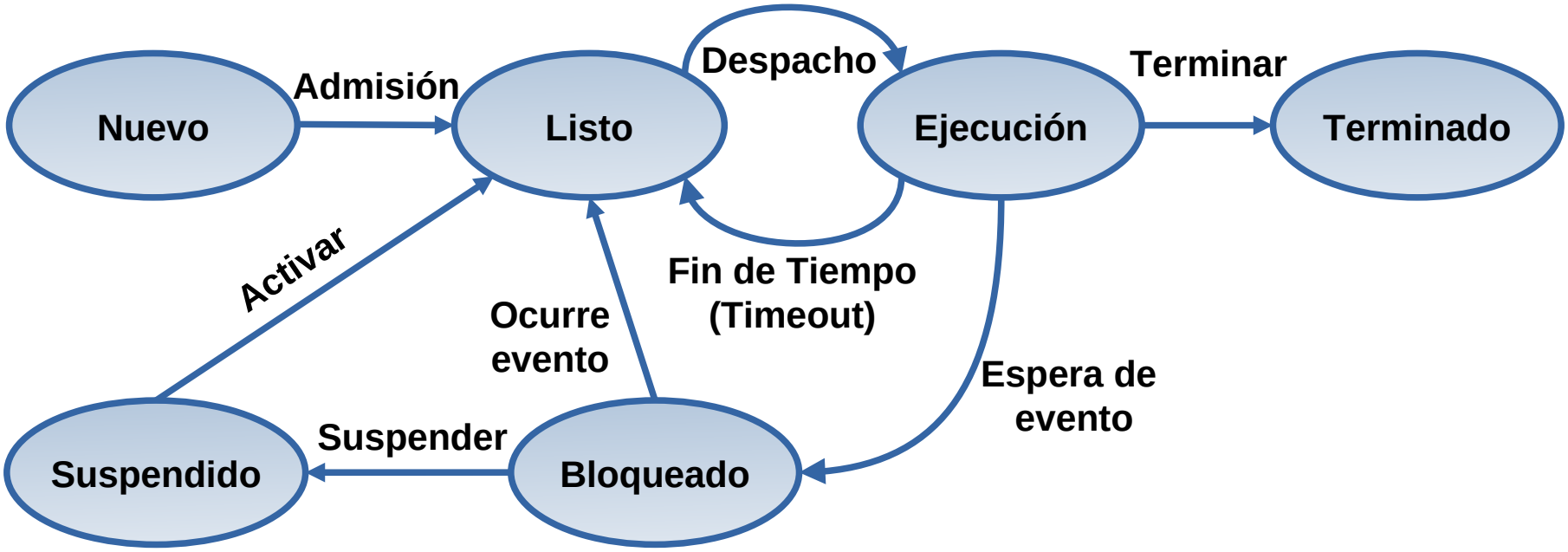
Memoria Virtual

- Cuando la MMU quita una página de memoria principal para almacenarla en memoria secundaria, el proceso que la usaba ya no puede utilizarla directamente.
- De la misma manera, un proceso en el que se genera un *page fault* debe esperar que se cargue en memoria principal la página que necesita.
- El resultado es que ambos procesos van a bloquearse.
- Ahora vamos a tener procesos que se bloquean porque están esperando entrada/salida y otros porque el SO está realizando un movimiento de memoria Principal a Secundaria *(swapping)*

Estado Suspendido

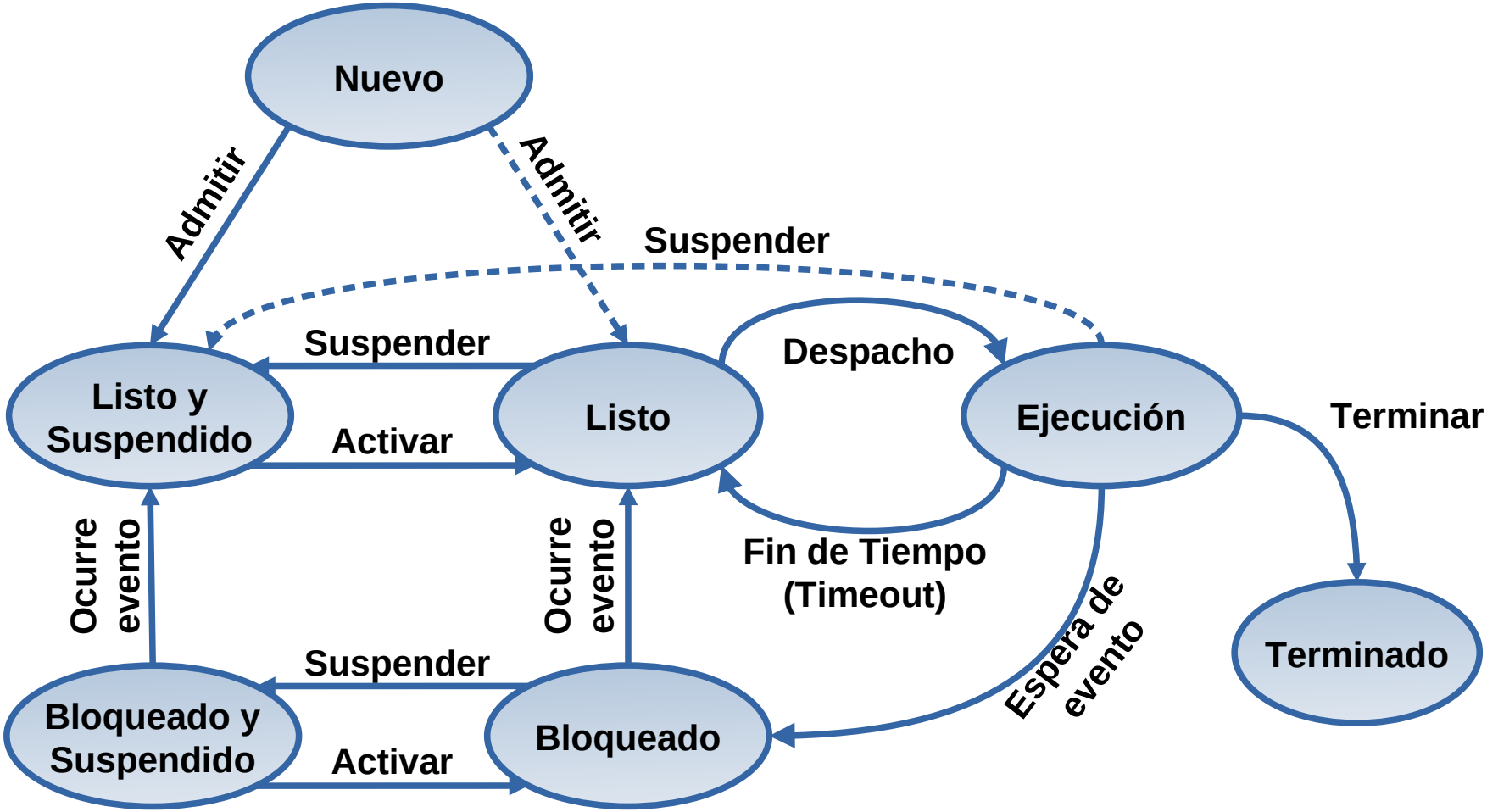
- Si vemos el modelo de 5 estados que habíamos elaborado, podemos mejorarlo si podemos separar ambos tipos de bloqueo
- Así podemos proponer un estado adicional “suspendido” como se muestra en la siguiente transparencia

Un estado suspendido



Todos los procesos que son suspendidos están en el estado Bloqueado
Pero puede haber otras posibilidades...

Dos estados suspendidos



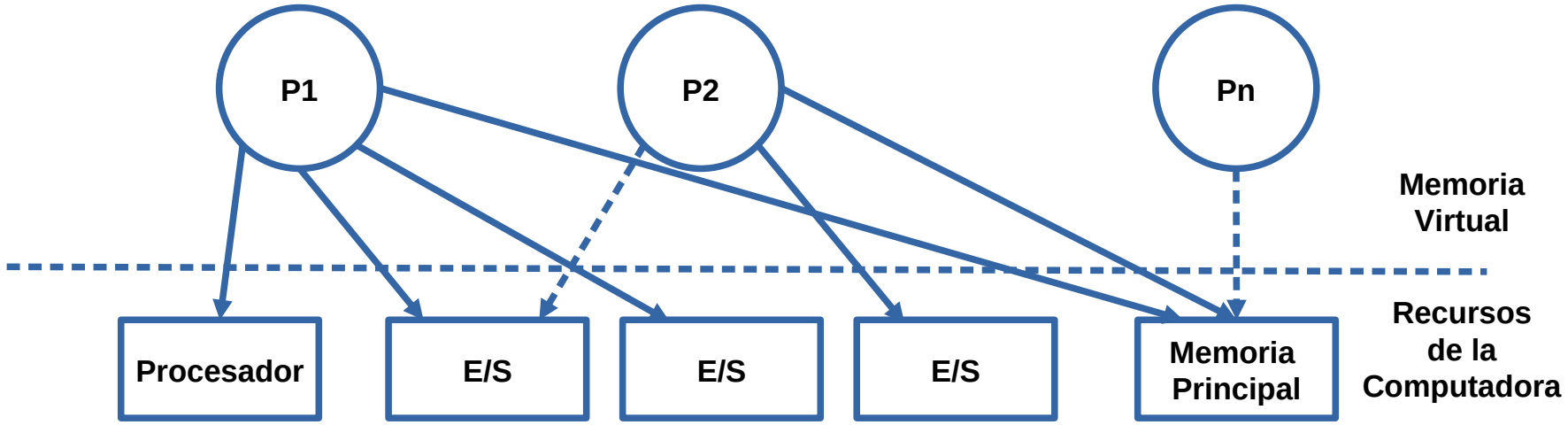
Modelo con dos estados suspendidos

- Al modelo de 5 estados se les agregan los dos siguientes:
- Bloqueado y Suspendido: El proceso está en disco esperando un suceso.
- Listo y Suspendido: El proceso está en disco pero está disponible para su ejecución tan pronto como se cargue en la memoria principal.

Razones para suspender un proceso

Intercambio	El sistema operativo necesita liberar suficiente memoria principal para cargar un proceso que está listo para ejecutarse
Solicitud del proceso padre	Un proceso padre puede querer suspender a ejecución de un proceso descendiente para examinar o modificar el proceso suspendido o para coordinar la actividad de varios descendientes
Solicitud de un Usuario	Un usuario puede querer suspender la ejecución de un programa con fines de depuración o en conexión con el uso de un recurso
Por Tiempo	Un proceso puede ejecutarse periódicamente (por ejemplo, un proceso o de supervisión del sistema) y puede ser suspendido mientras espera el siguiente intervalo de tiempo
Otra Razón del Sistema Operativo	El sistema operativo puede suspender un proceso de fondo, de utilidad o cualquier proceso que se sospecha sea el causante de un problema

Procesos y recursos



- **P1** está ejecutándose; una parte del proceso está en **memoria principal**; tiene el control de **dos** dispositivos de E/S
- **P2** también está en **memoria principal**, pero está **bloqueado** esperando al dispositivo de E/S que está asignado a P1
- El proceso **Pn** Está en Memoria Virtual y por tanto, está suspendido

Procesos y Recursos

- Para poder manejar la planificación y asignación de recursos a los procesos, el Sistema Operativo mantiene tablas de información sobre los recursos que está administrando.
- Información sobre el estado actual de cada proceso
- Información sobre el estado actual de cada recurso
- Esto lo realiza mediante tablas de:
  - Memoria
  - Entrada / Salida
  - Archivos
  - Procesos

Tabla de Memoria

- La asignación de memoria principal a los procesos
- La asignación de memoria secundaria a los procesos
- Atributos de protección de segmentos de memoria principal o virtual tales como qué procesos pueden acceder a ciertas regiones compartidas de memoria
- Información necesaria para gestionar la memoria virtual

Tabla de Entrada / Salida (E/S)

- La E/S puede estar disponible o estar asignada a un proceso
- Estado de una operación de E/S en curso
- La posición de memoria principal que se está utilizando como origen o destino de una transferencia de E/S

Tabla de Archivos

- Existencia de los archivos
- Su posición en la memoria secundaria
- Su estado actual
- Otros atributos
- Utilizada por un sistema de gestión de archivos

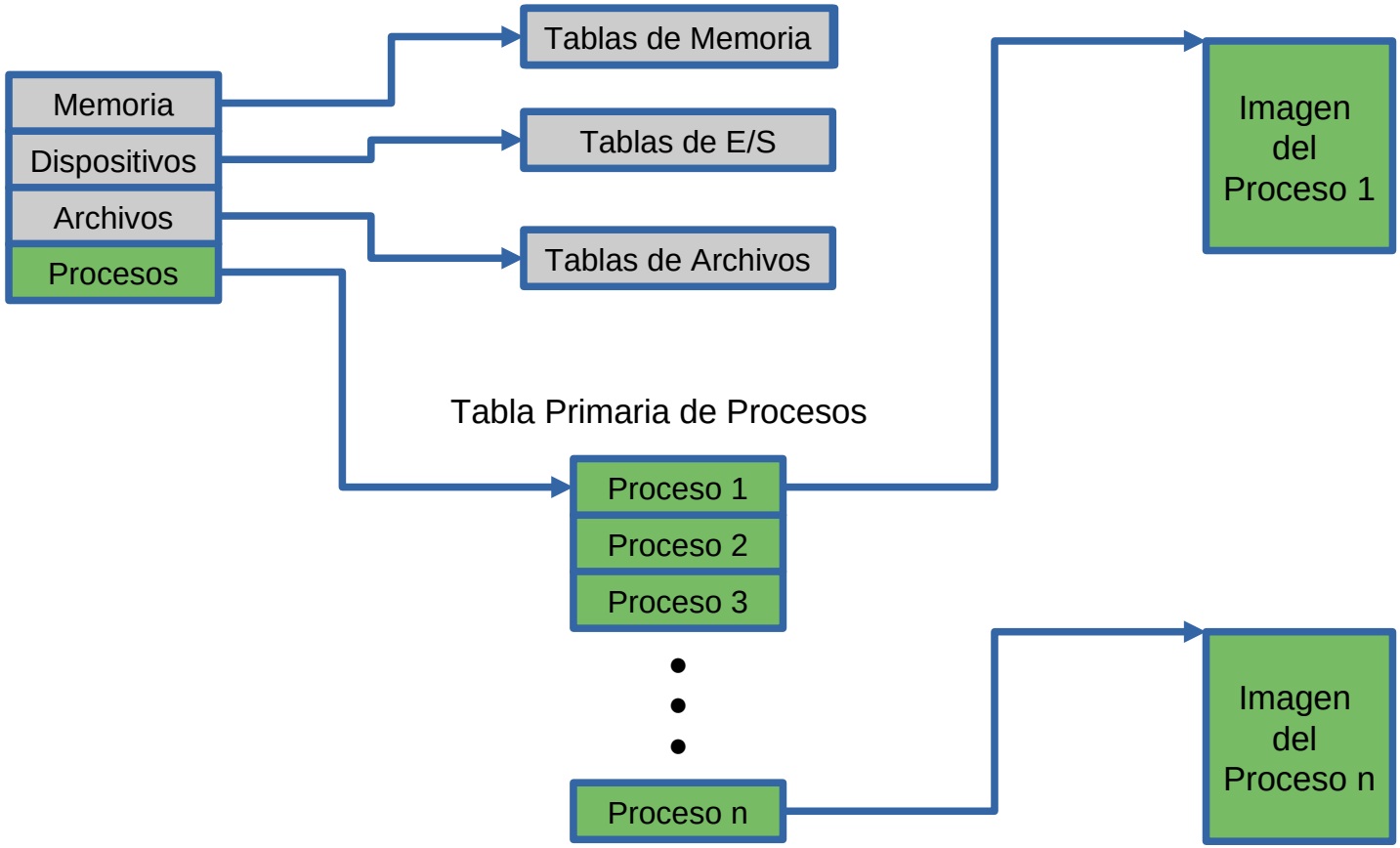
Tabla de Procesos

- Dónde está ubicado el proceso
- Atributos del proceso
  - Programa
  - Datos
  - Stack

Imagen del Proceso

- Elementos típicos de una Imagen de Proceso
- Datos de Usuario
  - La parte modificable del espacio de usuario. Puede guardar datos del programa, una zona para una pila del usuario y programas que pueden modificarse.
- Programa de Usuario
  - El programa a ejecutar (binario ejecutable)
- Pila del Sistema
  - Cada proceso tiene una o más pilas asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno
- Bloque de Control de Proceso
  - Información necesaria para que el sistema operativo controle al proceso.

Estructuras de Control del SO



Bloque de Control de Procesos

- El SO maneja distintos tipos de información dentro del Bloque de Control de Procesos:
  - Identificación del Proceso
  - Información de Estado del Procesador
  - Información de Control del Proceso

BCP: Identificadores

- Para cada proceso el SO maneja distintos identificadores numéricos:
  - Identificador de este proceso
  - Identificador del proceso que creó a este proceso (el proceso padre)
  - Identificador del usuario

BCP: Información de Estado del Procesador

- Registros Visibles para el Usuario
  - Un registro visible para el usuario es aquél al que puede hacerse referencia por medio del lenguaje máquina que ejecuta el procesador. Normalmente, existen de 8 a 32 de estos registros, aunque algunas implementaciones RISC tienen más de 100.
- Registros de Control y de Estado
  - Hay varios registros del procesador que se emplean para controlar su funcionamiento. Entre estos se incluyen:
  - Contador de Programa : Contiene la dirección de la próxima instrucción a ser tratada
  - Códigos de condición: Muestran el resultado de la operación aritmética o lógica más reciente (signo, cero, acarreo, igualdad, desbordamiento).
  - Información de estado: incluye los indicadores de habilitación o inhabilitación de interrupciones y el modo de ejecución

BCP: Información de Estado del Procesador

- Stack Pointers
  - Cada proceso tiene una o más pilas LIFO del sistema asociadas. Las pilas se utilizan para almacenar los parámetros y las direcciones de retorno de los procedimientos y de las llamadas al sistema. El puntero de pila siempre apunta a la cima de la pila.

BCP: Información de Control del Proceso

- Información de Planificación y de Estado
  - Esta es la información que se necesita por el sistema operativo para llevar a cabo sus funciones de planificación. Los elementos típicos de esta información son los siguientes:
  - **Estado del proceso:** Define la disposición del proceso para ser planificado para ejecutar (en ejecución, listo, esperando, detenido).
  - **Prioridad:** Se puede usar uno o más campos para describir la prioridad de planificación de los procesos. En algunos sistemas se necesitan varios valores (por omisión, actual, la más alta permitida).
  - **Información de planificación:** Esta dependerá del algoritmo de planificación utilizado. Como ejemplos se tienen la cantidad de tiempo que el proceso ha estado esperando y la cantidad de tiempo que el proceso ejecutó la última vez.
  - **Eventos:** La identidad del evento que el proceso está esperando antes de poder reanudarse.

BCP: Información de Control del Proceso

- Estructuración de Datos
  - Un proceso puede estar enlazado con otros procesos en una cola o alguna otra estructura. Por ejemplo todos los procesos que están en estado de espera de un nivel determinado de prioridad pueden estar enlazados en una cola. Un proceso puede mostrar una relación padre-hijo con otro proceso. El BCP puede contener punteros a otros procesos para dar soporte a estas estructuras.

BCP: Información de Control del Proceso

- Comunicación entre Procesos
  - Puede haber varios indicadores, señales y mensajes asociados con la comunicación entre dos procesos independientes. Una parte de esta información o toda ella se puede guardar en el bloque de control de proceso.
- Privilegios de los procesos
  - A los procesos se es otorgan privilegios en términos de la memoria a la que pueden acceder y el tipo de instrucciones que pueden ejecutar. Además, también se pueden aplicar privilegios al uso de los servicios y utilidades del sistema.

BCP: Información de Control del Proceso

- Gestión de Memoria
  - Esta sección puede incluir punteros a las tablas de páginas y/o segmentos que describen la memoria virtual asignada al proceso.
- Propiedad de los Recursos y Utilización
  - Se pueden indicar los recursos controlados por el proceso, tales como los archivos abiertos. También se puede incluir un histórico de la utilización del procesador o de otros recursos; esta información puede ser necesaria para el planificador.

Control de Procesos: Modos de ejecución

- Modo de usuario
  - Modo menos privilegiado
  - Los programas de usuario ejecutan normalmente en ese modo
- Modo de sistema, control, o kernel (núcleo)
  - Modo más privilegiado
  - Kernel del sistema operativo

Control de Procesos: Creación de Procesos

- Asignar un único identificador al nuevo proceso
- Asignar espacio para el proceso (imagen)
- Inicializar el bloque de control del proceso
- Establecer los enlaces apropiados
  - Por ejemplo, si el sistema operativo mantiene cada cola de planificación como una lista enlazada, entonces el proceso nuevo se debe poner en la cola de Listos o de Listos y suspendidos
- Crear o ampliar otras estructuras de datos
  - Ej.: El sistema operativo puede mantener un archivo de contabilidad
  - Ej. Actualizar la información acerca de procesos hijos en el proceso que lo creó

Control de Procesos: Cambio de Procesos I

- Interrupción de reloj:
  - Si el SO determina que el proceso que está en ejecución, se ha estado ejecutando durante la fracción máxima de tiempo permitida, el proceso debe pasar al estado Listo y se debe expedir otro proceso
- Interrupción de E/S:
  - Si la acción constituye un suceso que están esperando uno o más procesos, entonces el SO traslada todos los procesos bloqueados correspondientes al estado Listo o Listo suspendido
- Fallo de memoria:
  - Una referencia a una dirección de memoria virtual no está en memoria principal. El SO puede llevar a cabo un cambio de contexto para reanudar la ejecución de otro proceso; el proceso que cometió el fallo de memoria se pasa a estado Bloqueado.

Control de Procesos: Cambio de Procesos II

- Trap (Trampa)
  - Error o excepción
  - Si es fatal puede causar que el proceso pase al estado terminado
- Llamada al sistema
  - Ej: abrir un archivo. El proceso de usuario pasa al estado suspendido

Control de Procesos: Cambio de Estado de Procesos

- Salvar el contexto del procesador, incluyendo el contador de programa y otros registros
- Actualizar el bloque de control del proceso que estaba en estado de Ejecución
- Mover el bloque de control del proceso a la cola apropiada: Listos, Bloqueados, Listos y suspendidos
- Seleccionar otro proceso para ejecución

Control de Procesos: Cambio de Estado de Procesos

- Actualizar el bloque de control del proceso seleccionado
- Actualizar las estructuras de datos de gestión de memoria
- Restaurar el contexto del procesador a aquel que existía en el momento en el que el proceso seleccionado dejó por última vez el estado de Ejecución

Kernel o Núcleo

- El Núcleo o Kernel del Sistema Operativo, es el conjunto de funciones que realiza las tareas más usuales.
- Permanece en memoria principal, para poder ejecutarse cuando sea necesario

Funciones del Kernel

- Gestión de Procesos
  - Creación y terminación de los procesos
  - Planificación y despacho de los procesos
  - Cambio de procesos
  - Sincronización de procesos y soporte para la comunicación entre procesos
  - Gestión de los bloques de control de procesos
- Gestión de memoria
  - Asignación de espacios de direcciones a los procesos
  - Intercambio
  - Gestión de páginas y segmentos
- Gestión de E/S
  - Gestión de buffers
  - Asignación de canales de E/S y dispositivos a los procesos
- Funciones de Soporte
  - Tratamiento de interrupciones
  - Contabilidad
  - Supervisión

Procesos en Linux

Estados

- TASK_RUNNING (0): Indica que el proceso en cuestión se está ejecutando o listo para ejecutarse. En este segundo caso, el proceso dispone de todos los recursos necesarios excepto el procesador.
- TASK_ INTERRUPTIBLE (1) : el proceso está suspendido a la espera de alguna señal para pasar a listo para ejecutarse. Generalmente se debe a que el proceso está esperando a que otro proceso del sistema le preste algún servicio solicitado.
- TASK_UNINTERRUPTIBLE (2): el proceso está bloqueado esperando a que se le conceda algún recurso hardware que ha solicitado (cuando una señal no es capaz de “despertarlo”).
- TASK_ZOMBIE (4): el proceso ha finalizado pero aún no se ha eliminado todo rastro del mismo del sistema. Esto es habitualmente causado porque el proceso padre todavía lo espera con una wai t ( ) .
- TASK_STOPPED (8): el proceso ha sido detenido por una señal o bien mediante el uso de ptrace( ) para ser trazado.

Colas de Procesamiento

- Cola de Ejecución o runqueue: procesos en estado TASK_RUNNING .
- Colas de Espera o wait queues: procesos en estado TASK_ INTERRUPTIBLE ó TASK_ UNINTERRUPTIBLE.
- Los procesos en estado TASK_ZOMBIE ó TASK_STOPPED no necesitan colas para ser gestionados.

Identificadores

- Cada proceso posee un identificador único (**`PID`**)
- Si un proceso es “hijo” de otro, podemos conocer el identificador de su “padre” **`PPID`.**
- Además de los identificadores, los procesos poseen credenciales, que les permiten acceder a distintos recursos:
  - uid_t uid,euid,suid,fsuid;
  - gid_t gid,egid,sgid,fsgid;

Información sobre procesos?

- La información sobre procesos que está contenida en el BCP puede accederse a través del sistema de archivos virtual montado en `/proc`
- Los programas `ps` y `pstree` permiten acceder a esa información.
- En su forma mas básica `ps` me da:
  - PID,
  - TTY asociada al proceso
  - tiempo de ejecución
  - comando que generó la ejecución.

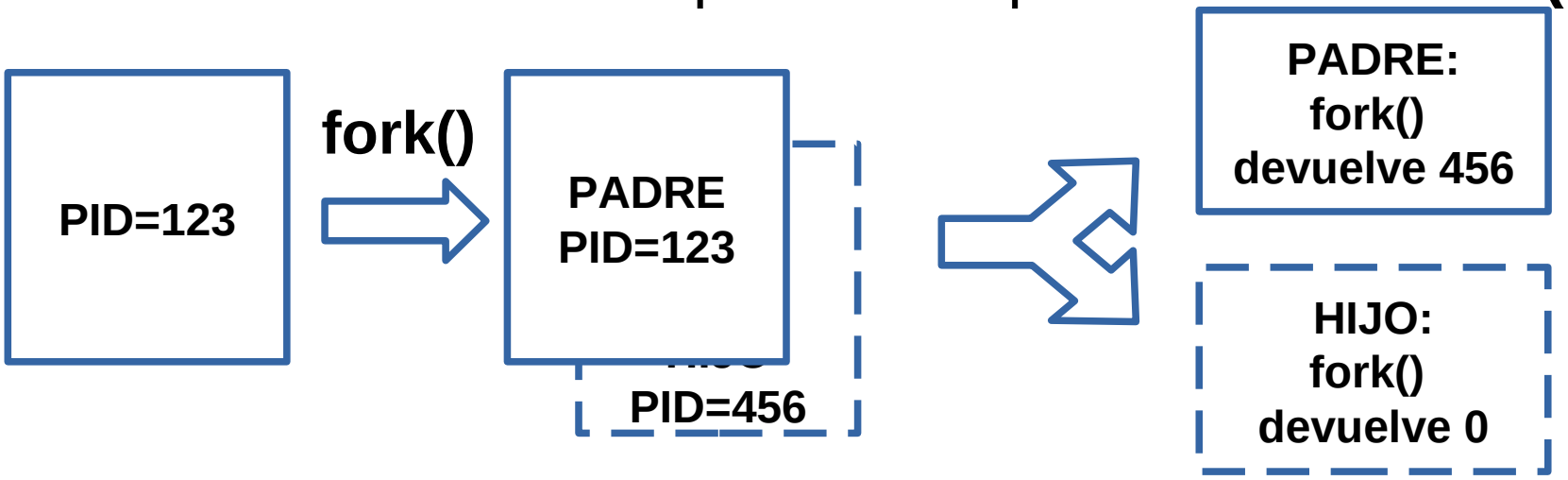
Información sobre procesos?

Salida del comando ps a

```
  PID  TTY      STAT   TIME COMMAND
 1096 tty7     Ssl+   2:55 /usr/lib/xorg/Xorg ...
 1099 tty1     Ss+    0:00 /sbin/agetty -o -p ...
 7079 pts/0    Ss     0:00 bash
 9600 pts/1    Ss     0:00 bash
 9771 pts/1    S+     0:00 man ps
 9781 pts/1    S+     0:00 pager
10214 pts/0    R+     0:00 ps a
```

Creación de Procesos

- La llamada al sistema para crear procesos es **`fork()`**.



- **fork () genera una copia de la imagen del proceso padre.**
- **También asigna una nueva entrada en el BCP y nuevos identificadores para el proceso Hijo**
- **Para distinguir uno de otro fork() devuelve 0 al hijo y el PID del Hijo al Padre**

Ejemplo:

```
pid_t p;
p = fork();
if (p!=0)
   {
   /* Acá sigue el padre, p == PID del hijo. Esta parte del
      programa la ejecuta solo el padre */
   }
else
   {
   /* Acá sigue el hijo, p == 0. Esta parte del programa la
      ejecuta solo el hijo */
   }
/* Esta parte del programa la ejecutan tanto el padre como
   el hijo*/
```

Llamadas al sistema

`getpid() – getppid() – wait() – waitpid()`

- `getpid()` devuelve el PID del proceso que hacela llamada al sistema.
- `getppid()` devuelve el PID del proceso **padre** del que hace la llamada al sistema.
- `wait()` es una llamada que hace el padre para que cuando cualquier hijo termine, se libere los recursos que ocupa
- `waitpid()` es una llamada que hace el padre para que se liberen los recursos del hijo con un PID específico cuando este termine.

Llamada al Sistema execve()

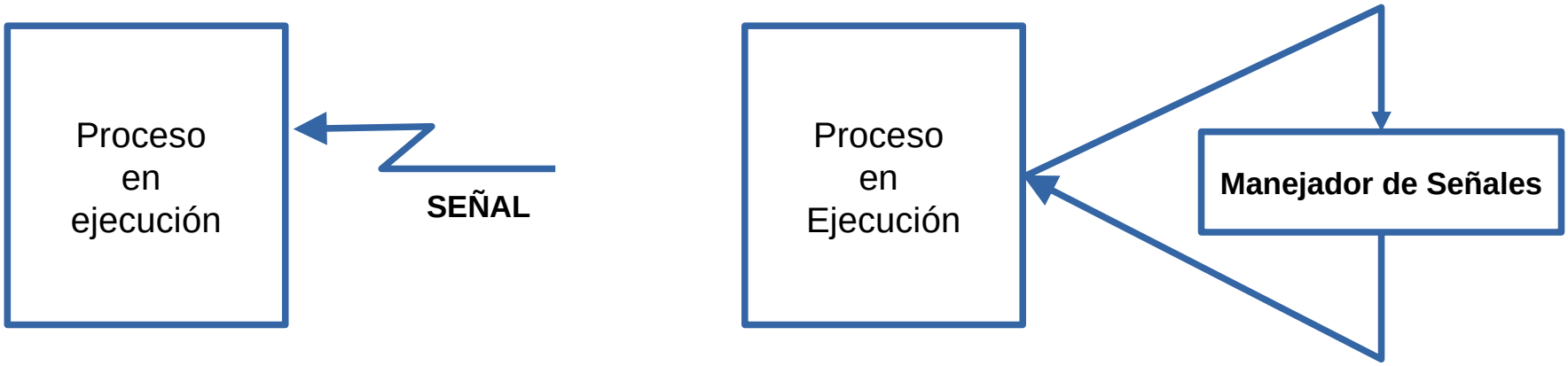
```
int execve( const char *filename, char *const argv[],
            char *const envp[]);
```

- `execve()` ejecuta el programa cuyo nombre se pasa en `filename`, debe ser un ejecutable binario o un script, cuya primera linea comience con `!#interprete. (p.Ej.  !#/bin/bash)`
- `execve()` reemplaza la imagen del proceso que lo llama con la imagen del nuevo programa.
- `argv[]` es un arreglo de punteros donde se pasan los argumentos para `filename.`
- `envp[]` es un arreglo de punteros donde se pasan las variables de ambiente en la forma VARIABLE=VALOR.
- Tanto argv, como envp deben terminar en un puntero nulo.

Variantes de execve()

```
int execl(const char *path, const char *arg,
    /* (char  *) NULL */);
int execlp(const char *file, const char *arg, ...
    /* (char  *) NULL */);
int execle(const char *path, const char *arg, ...
    /*, (char *) NULL, char * const envp[] */);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
    char *const envp[]);
```

Señales



El SO permite enviar señales a los procesos
Existen hasta 64 señales posibles.
El manejador de señales por defecto termina el programa
Dentro del programa puedo instalar un manejador de señales “a medida”

Señales

1. SIGHUP
2. SIGINT
3. SIGQUIT
4. SIGILL
5. SIGTRAP
6. SIGABRT
7. SIGBUS
8. SIGFPE
9. SIGKILL
10. SIGUSR1
11. SIGSEGV
12. SIGUSR2
13. SIGPIPE
14. SIGALRM
15. SIGTERM
16. SIGSTKFLT
17. SIGCHLD
18. SIGCONT
19. SIGSTOP
20. SIGTSTP
21. SIGTTIN
22. SIGTTOU
23. SIGURG
24. SIGXCPU
25. SIGXFSZ
26. SIGVTALRM
27. SIGPROF
28. SIGWINCH
29. SIGIO
30. SIGPWR
31. SIGSYS

Manejo de Señales (recepción)

- Definición de tipo

```
typedef void (*sighandler_t)(int);
```

- Función signal

```
sighandler_t signal(int signum,
                    sighandler_t handler);
```

- Función de manejo de señal

```
void handler(int);
```

pause()

- Cuando queremos que un proceso espere a que le llegue una señal, usamos la función `pause()`.

```
#include <unistd.h>
int pause (void);
```

- Esta función provoca que el proceso en cuestión “duerma” hasta que le llegue una señal. (El SO pone al proceso en estado suspendido)
- Para capturar esa señal,el proceso deberá haber establecido un tratamiento de la misma con la función `signal()`.

Ejemplo

```
void trapper(int);
int main(int argc, char *argv[]) {
        int i;
        do {
            printf( “ingrese numero de señal a manejar\n”);
            scanf(“%d”,&i);
            signal(i, trapper);
            printf("Identificativo de proceso: %d\n espero una señal para
            continuar...\n", getpid() );
            pause();
            printf("Continuando...\n");
            } while (1);
        return 0;
        }
void trapper(int sig) {
        printf("Recibida la señal: %d\n", sig);
        }
```

alarm() y sleep()

- Se puede usar SIGALARM para crear timers en nuestros programas.
- La función `alarm()` hace que nuestro proceso se envíe a sí mismo una señal SIGALARM en el número de segundos que especifiquemos.

```
unsigned int alarm(unsigned int seconds);
```

- La función **`sleep(n)`** permite esperar **`n`** segundos antes de continuar.

```
unsigned int sleep(unsigned int seconds);
```

- **PRECAUCIÓN**: `alarm()` y `sleep()` pueden interferir entre sí . No usarlas juntas.

Señales, padres e hijos.

- Los procesos envían una señal SIGCHLD a sus padres cuando terminan.
- Esta señal es la que activa las funciones `wait()` y `waitpid()`