# Gestión de procesos

Arquitectura y Sistemas Operativos

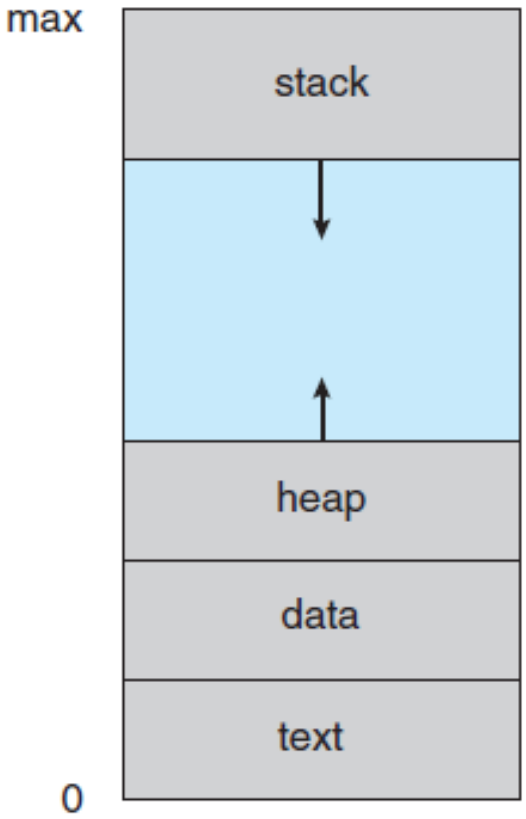
Unidad 2

Ing. Gustavo Alessandrini

# ¿Qué es un proceso?

- Un proceso es un programa en ejecución
- Un proceso necesita **recursos** para que se pueda ejecutar:
  - Tiempo de CPU,
  - Memoria,
  - Dispositivos de E/S, etc.
- Estos recursos se asignan al proceso cuando se crea o mientras se está ejecutando
- Áreas típicas de la memoria:
  - Código (también llamado «texto»)
  - Datos (variables globales + memoria dinámica)
  - Pila (datos locales de funciones y subrutinas)

# Organización de la memoria de un proceso




La pila *(stack)* contiene datos temporales (como parámetros de función, direcciones de retorno y variables locales)

El montón *(heap)* es memoria asignada dinámicamente durante el tiempo de ejecución del proceso

La sección de datos contiene variables globales

El código de programa, se conoce como la "sección de texto"

# Proceso – Información de control

En cualquier instante, mientras un proceso está en **ejecución**, este proceso se puede caracterizar por una serie de elementos, incluyendo los siguientes:

- **Identificador:** Identificador único, para distinguirlo del resto de procesos
- **Estado:** Indica el proceso actual del proceso
- **Prioridad:** Nivel de prioridad relativo al resto de procesos
- **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, y de bloques de memoria compartidos con otros procesos
- **Datos de contexto:** Los datos que están presenten 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 asignados a dicho proceso, lista de archivos en uso, 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.

# Proceso – Información de control

La información de la lista anterior se almacena en una estructura de datos, que se suele llamar bloque de control de proceso –BCP- (*process control block, PCB)* que el sistema operativo **crea *y* gestiona**

- El BCP contiene suficiente información de forma que es posible interrumpir el proceso cuando está corriendo y posteriormente restaurar su estado de ejecución como si no hubiera habido interrupción alguna
- El BCP es la herramienta clave que permite al sistema operativo dar soporte a múltiples procesos y proporcionar multiprogramación
  - Cuando un proceso se interrumpe, los valores actuales del contador de programa y los registros del procesador (datos de contexto) se guardan en los campos correspondientes del BCP y el estado del proceso se 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 registros del procesador y este proceso comienza a correr
- Se puede decir que un proceso está compuesto del código de programa y los datos asociados, más el bloque de control de proceso o BCP

# Bloque de Control de Proceso

- Cada proceso debe tener una estructura de datos que almacena su estado e información para su control
- Contiene los elementos del proceso
- El BCP es creado y gestionado por el sistema operativo
- Permite soporte para múltiples procesos

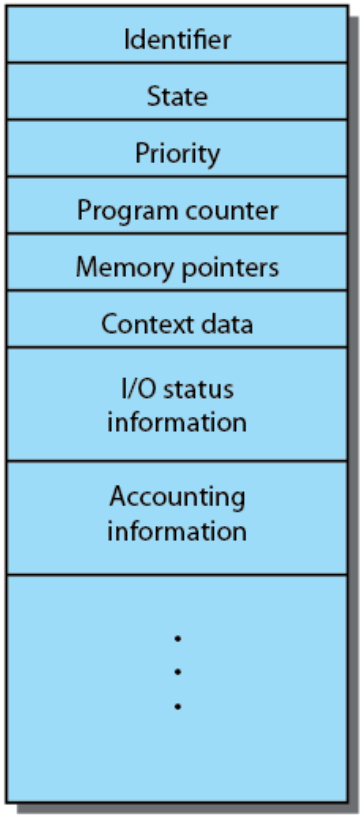



Figure 3.1 Simplified Process Control Block

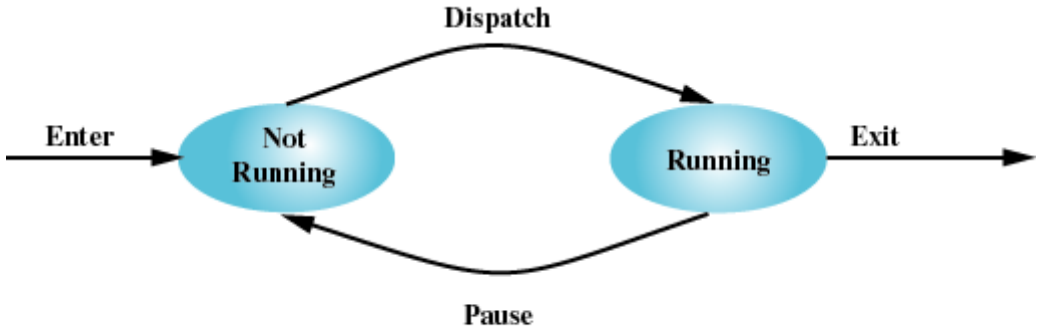
Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Modelo de proceso de 2 estados

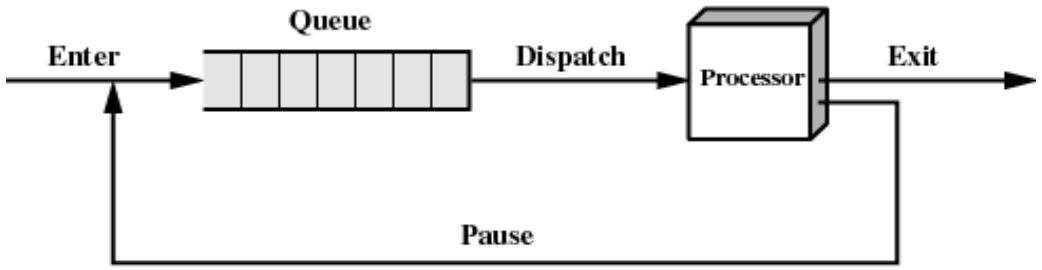
- La responsabilidad principal del sistema operativo es controlar la ejecución de los procesos; esto incluye determinar como entrelazarlos para la ejecución y asignar recursos a los procesos
- El primer paso en el diseño de un SO para el control de procesos es describir el comportamiento que se desea que tengan los procesos
- Se puede construir el modelo más simple posible observando que, en un instante dado, un proceso está siendo ejecutando por el procesador o no. En este modelo, un proceso puede estar en dos estados:
  - Ejecutando
  - No Ejecutando
- Cuando el SO crea un nuevo proceso, crea el BCP para el nuevo proceso e inserta dicho proceso en el sistema en estado **No Ejecutando**
  - El proceso existe, es conocido por el sistema operativo, y está esperando su oportunidad de ejecutar
  - De cuando en cuando, el proceso actualmente en ejecución se interrumpirá y una parte del sistema operativo, el activador *(dispatcher)*, seleccionará otro proceso para ejecutar

# Modelo de proceso de 2 estados

- Existe una cola cuyas entradas son punteros al BCP de un proceso en particular
- Un proceso que se interrumpe se transfiere a la cola de procesos en espera
- Si el proceso ha finalizado o ha sido abortado, se descarta (sale del sistema)
- El activador selecciona un proceso de la cola para ejecutar




(a) State transition diagram




(b) Queuing diagram

Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Modelo de proceso de 2 estados Limitaciones

- Si todos los procesos estuviesen siempre preparados para ejecutar, la gestión de colas proporcionada sería efectiva
  - La cola es una lista de tipo FIFO y el procesador opera siguiendo una estrategia cíclica (*round-robin o turno rotatorio) sobre todos los procesos disponibles* (cada proceso de la cola tiene cierta cantidad de tiempo, por turnos, para ejecutar y regresar de nuevo a la cola, a menos que se bloquee)
- Sin embargo, esta implementación es inadecuada
  - Algunos procesos que están en el estado No Ejecutando están **listos para ejecutar**, mientras que otros están **bloqueados**, esperando a que se complete una operación de E/S
- Por lo tanto, utilizando una única cola, el activador debería recorrer la lista buscando los procesos que no estén bloqueados y que lleven en la cola más tiempo
- Una forma más natural para manejar esta situación es dividir el estado de No Ejecutando en dos estados: **Listo** y **Bloqueado**
- Para gestionarlo correctamente, se añaden dos estados adicionales (**Nuevo** y **Terminado**) que resultarán muy útiles

# Modelo de 5 estados

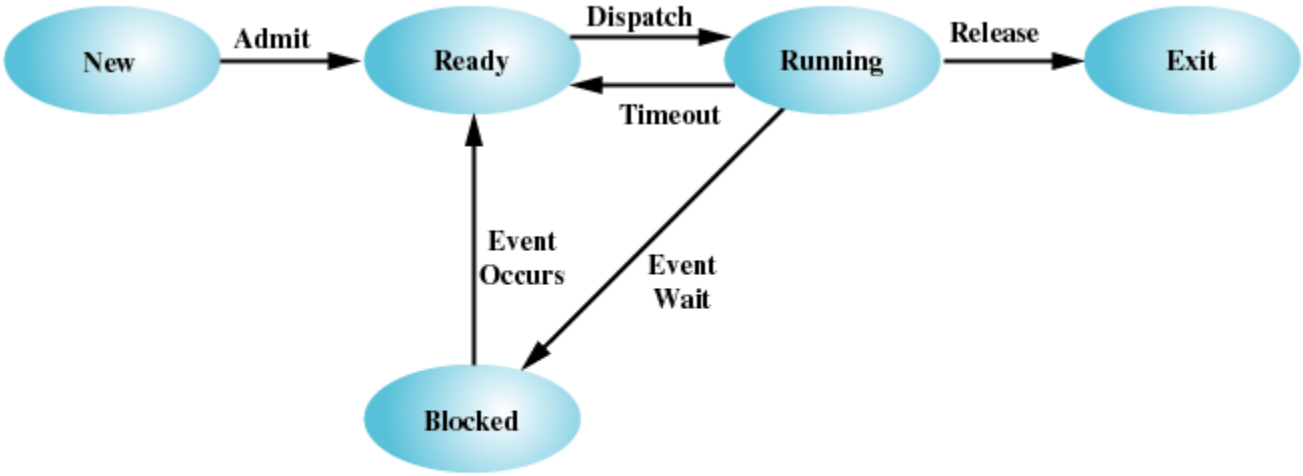



**Figure 3.6 Five-State Process Model**

Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Modelo de 5 estados

- **Ejecutando:** El proceso está actualmente en ejecución. (se asume que la computadora 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 se 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

# Creación de procesos

| | |
|---|---|
| Nuevo proceso de lotes | El sistema operativo dispone de un flujo de control de lotes de trabajos. Cuando el sistema operativo está listo para procesar un nuevo trabajo, leerá la siguiente secuencia de órdenes de control de trabajos |
| Sesión interactiva | Un usuario desde un terminal entra en el sistema |
| Creado por el sistema operativo para proporcionar un servicio | El sistema operativo puede crear un proceso para realizar una función en representación de un programa de usuario, sin que el un servicio usuario tenga que esperar (por ejemplo, un proceso para controlar la impresión) |
| Creado por un proceso existente | Por motivos de modularidad o para explotar el paralelismo, un programa de usuario puede ordenar la creación de un número de procesos |

# Terminación de procesos

| | |
|---|---|
| Finalización normal | El proceso ejecuta una llamada al SO para indicar que ha completado su ejecución |
| Límite de tiempo excedido | El proceso ha ejecutado más tiempo del especificado en un límite máximo. |
| Memoria no disponible | El proceso requiere más memoria de la que el sistema puede proporcionar |
| Violaciones de frontera | El proceso trata de acceder a una posición de memoria a la cual no tiene permiso |
| Error de protección | El proceso trata de usar un recurso, por ejemplo un archivo, al que no tiene permitido acceder, o trata de utilizarlo de una forma no apropiada |
| Error aritmético | El proceso trata de realizar una operación de cálculo no permitida (ej: división por 0) |
| Falla de E/S | Se ha producido un error durante una operación de E/S |
| Instrucción no válida | El proceso intenta ejecutar una instrucción inexistente |
| Instrucción privilegiada | El proceso intenta utilizar una instrucción reservada al sistema operativo |
| Mal uso de datos | Una porción de datos es de tipo erróneo o no se encuentra inicializada |
| Finalización del proceso padre | Cuando un proceso padre termina, el sistema operativo puede automáticamente finalizar todos los procesos hijos descendientes de dicho padre |
| Solicitud del proceso padre | Un proceso padre tiene autoridad para finalizar sus propios procesos descendientes |

# Ejemplo de ejecución

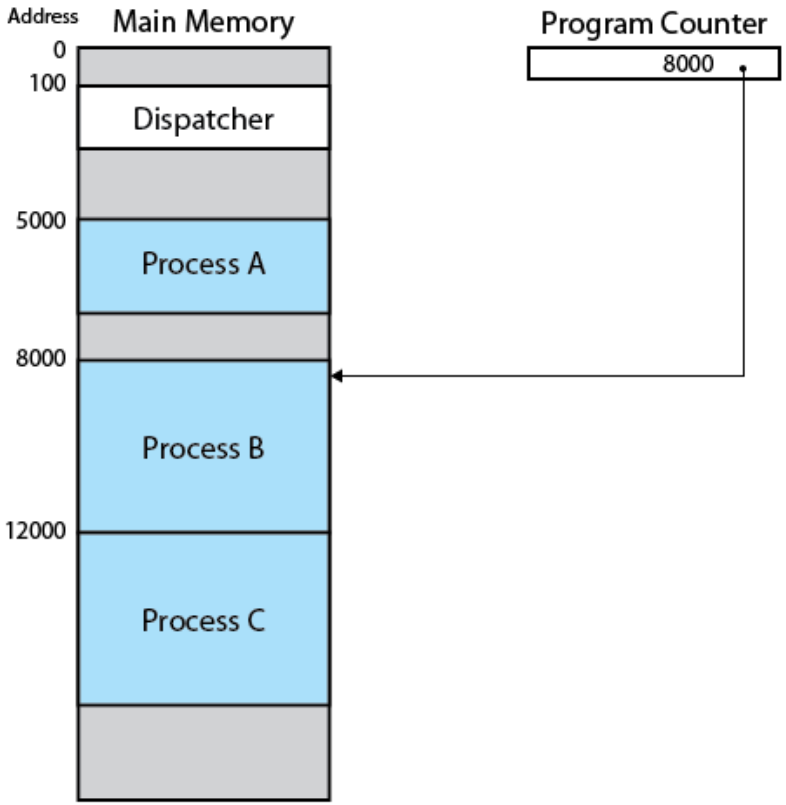



Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Ejemplo de ejecución

| (a) Trace of Process A | (b) Trace of Process B | (c) Trace of Process C |
|---|---|---|
| 5000 | 8000 | 12000 |
| 5001 | 8001 | 12001 |
| 5002 | 8002 | 12002 |
| 5003 | 8003 | 12003 |
| 5004 | | 12004 |
| 5005 | | 12005 |
| 5006 | | 12006 |
| 5007 | | 12007 |
| 5008 | | 12008 |
| 5009 | | 12009 |
| 5010 | | 12010 |
| 5011 | | 12011 |

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Ejemplo de ejecución

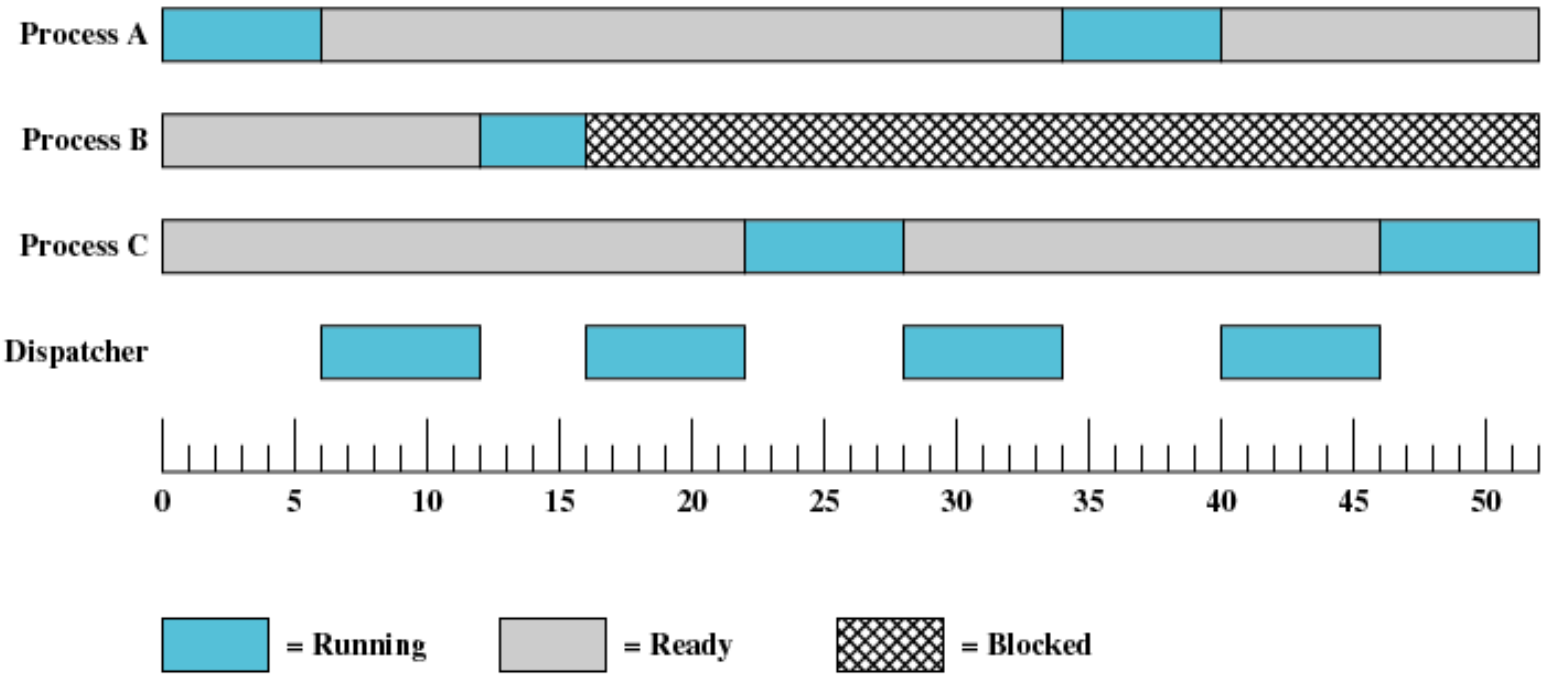
| | |
|---|---|
| 1 | 5000 |
| 2 | 5001 |
| 3 | 5002 |
| 4 | 5003 |
| 5 | 5004 |
| 6 | 5005 |
| ---------- | Time out |
| 7 | 100 |
| 8 | 101 |
| 9 | 102 |
| 10 | 103 |
| 11 | 104 |
| 12 | 105 |
| 13 | 8000 |
| 14 | 8001 |
| 15 | 8002 |
| 16 | 8003 |
| ---------- | I/O request |
| 17 | 100 |
| 18 | 101 |
| 19 | 102 |
| 20 | 103 |
| 21 | 104 |
| 22 | 105 |
| 23 | 12000 |
| 24 | 12001 |
| 25 | 12002 |
| 26 | 12003 |
| 27 | 12004 |
| 28 | 12005 |
| ---------- | Time out |
| 29 | 100 |
| 30 | 101 |
| 31 | 102 |
| 32 | 103 |
| 33 | 104 |
| 34 | 105 |
| 35 | 5006 |
| 36 | 5007 |
| 37 | 5008 |
| 38 | 5009 |
| 39 | 5010 |
| 40 | 5011 |
| ---------- | Time out |
| 41 | 100 |
| 42 | 101 |
| 43 | 102 |
| 44 | 103 |
| 45 | 104 |
| 46 | 105 |
| 47 | 12006 |
| 48 | 12007 |
| 49 | 12008 |
| 50 | 12009 |
| 51 | 12010 |
| 52 | 12011 |
| ---------- | Time out |

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Fuente: http://williamstallings.com/OS/OS5e-inst.html

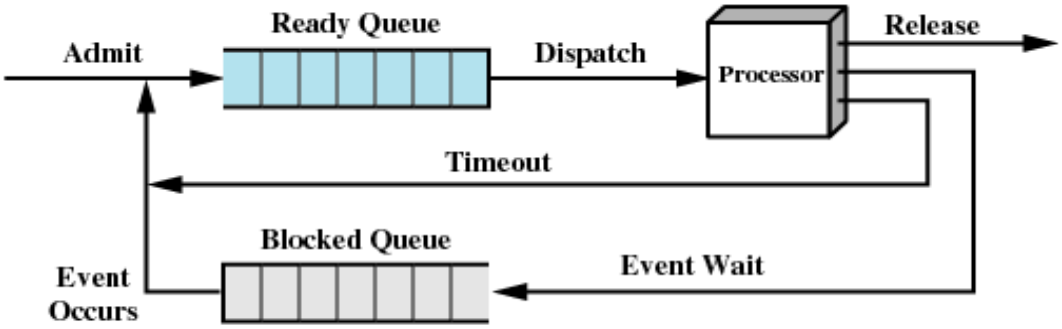
# Ejemplo de ejecución




Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Colas de procesos

- Modelo de 5 estados usando 2 colas

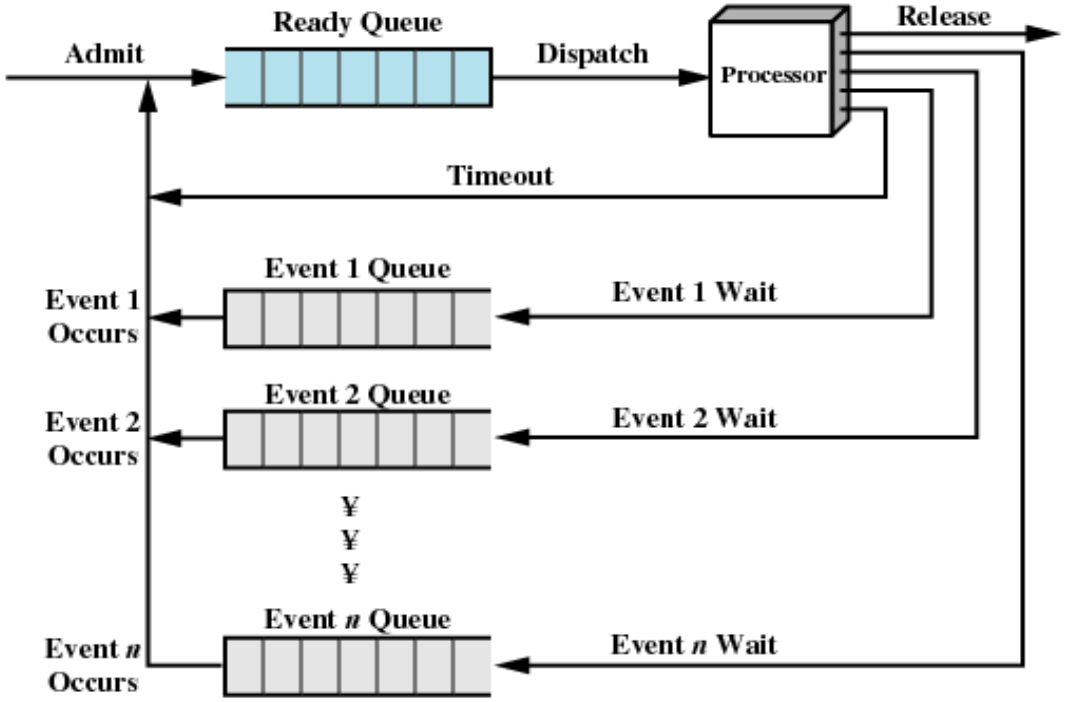



(a) Single blocked queue

Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Colas de procesos

- Modelo de 5 estados usando múltiples colas de procesos bloqueados




(b) Multiple blocked queues

Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Planificación de procesos

- El objetivo de la multiprogramación es tener en ejecución varios procesos al mismo tiempo con el fin de maximizar la utilización de la CPU
- El objetivo de los sistemas de tiempo compartido es conmutar la CPU entre los distintos procesos con la frecuencia suficiente para que los usuarios puedan interactuar con cada programa mientras el proceso se ejecuta
- El **planificador** de procesos selecciona un proceso disponible para ejecutarlo en la CPU
- En los sistemas de un solo procesador, nunca habrá mas de un proceso en ejecución
  - Si hay mas procesos, tendrán que esperar hasta que la CPU se libere y se pueda asignar a otro proceso

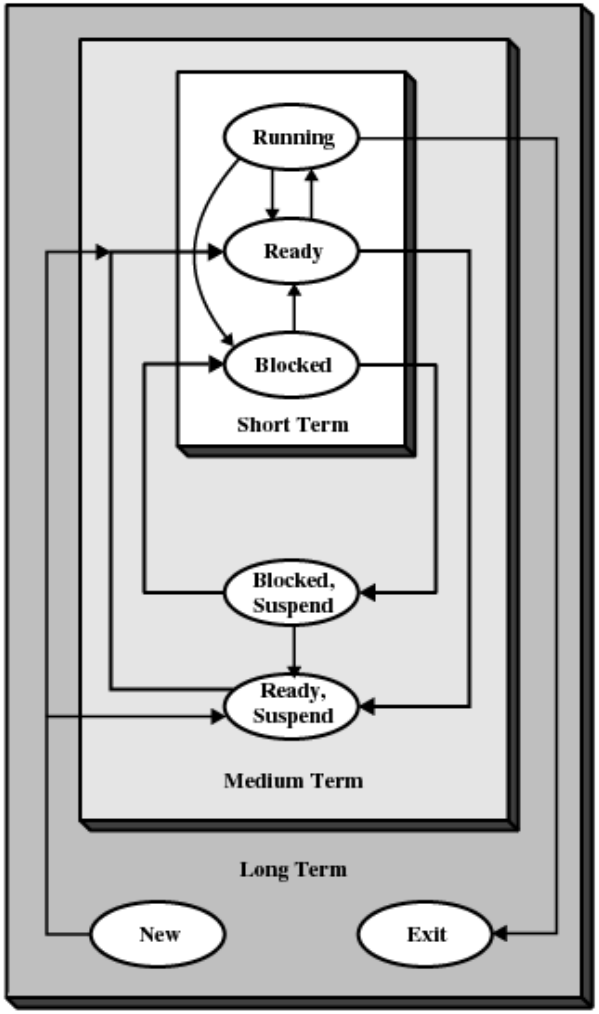
# Planificación de la CPU (CPU scheduling)

- ¿Qué proceso utiliza la CPU en cada momento?
- Objetivos diferentes según el tiempo de sistema:
  - Multiprogramación básica → sacar rendimiento al procesador
  - Tiempo compartido → garantizar tiempos de respuesta cortos
  - Tiempo real → garantizar plazos de ejecución
  - …

# Niveles de planificación

- El **planificador de largo plazo (PLP)** o de alto nivel, que suministra procesos a la cola de preparados
  - El PLP trata de conseguir una mezcla adecuada de trabajos intensivos en CPU y en E/S. Se ejecuta con poca frecuencia
- El **planificador de corto plazo (PCP)** ***(dispatcher)*** o de bajo nivel es el que asigna y desasigna la CPU
- El **planificador de medio plazo (*swapper)***
  - Envía al disco procesos bloqueados, para liberar memoria principal a los otros procesos a **intercambio (*swapping)***

# Niveles de planificación




Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Estructuras de control del SO

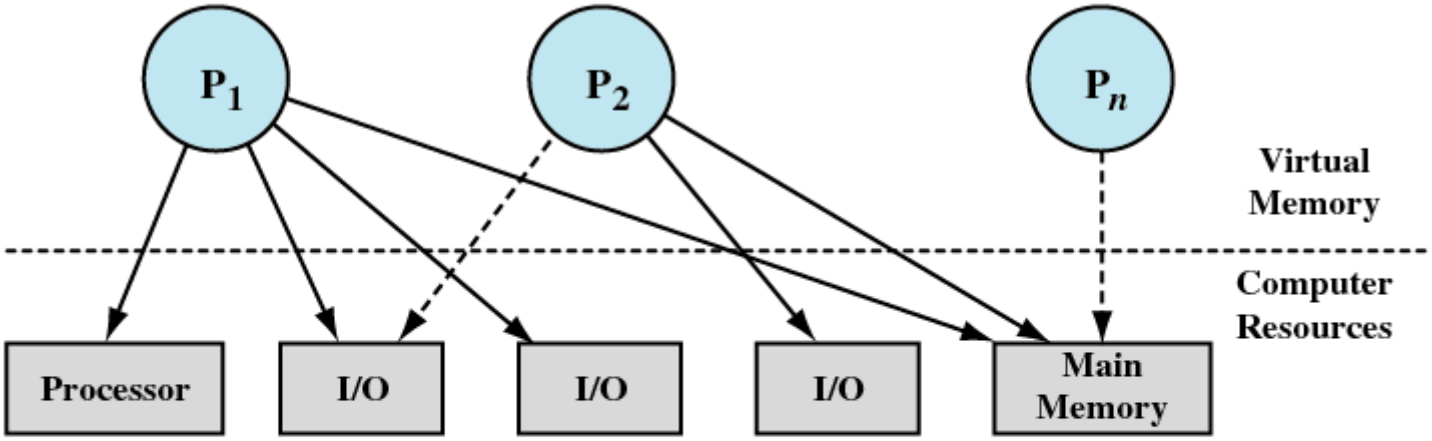



**Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)**

Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Estructuras de control del SO

- Información sobre el estado actual de cada proceso y recurso
- Se construyen tablas para cada entidad que el sistema operativo gestiona
  - Tablas de memoria
  - Tablas de dispositivos de E/S
  - Tablas de archivos
  - Tablas de procesos

# Tablas de memoria

- Asignación de memoria principal a procesos
- Asignación de memoria secundaria a procesos
- Atributos de protección para el acceso a regiones de memoria compartida
- Información necesaria para administrar la memoria virtual

# Tablas de E/S

- Dispositivo de E/S disponible o asignado
- Estado de la operación de E/S
- Ubicación en la memoria principal que se utiliza como origen o destino de la transferencia de E/S

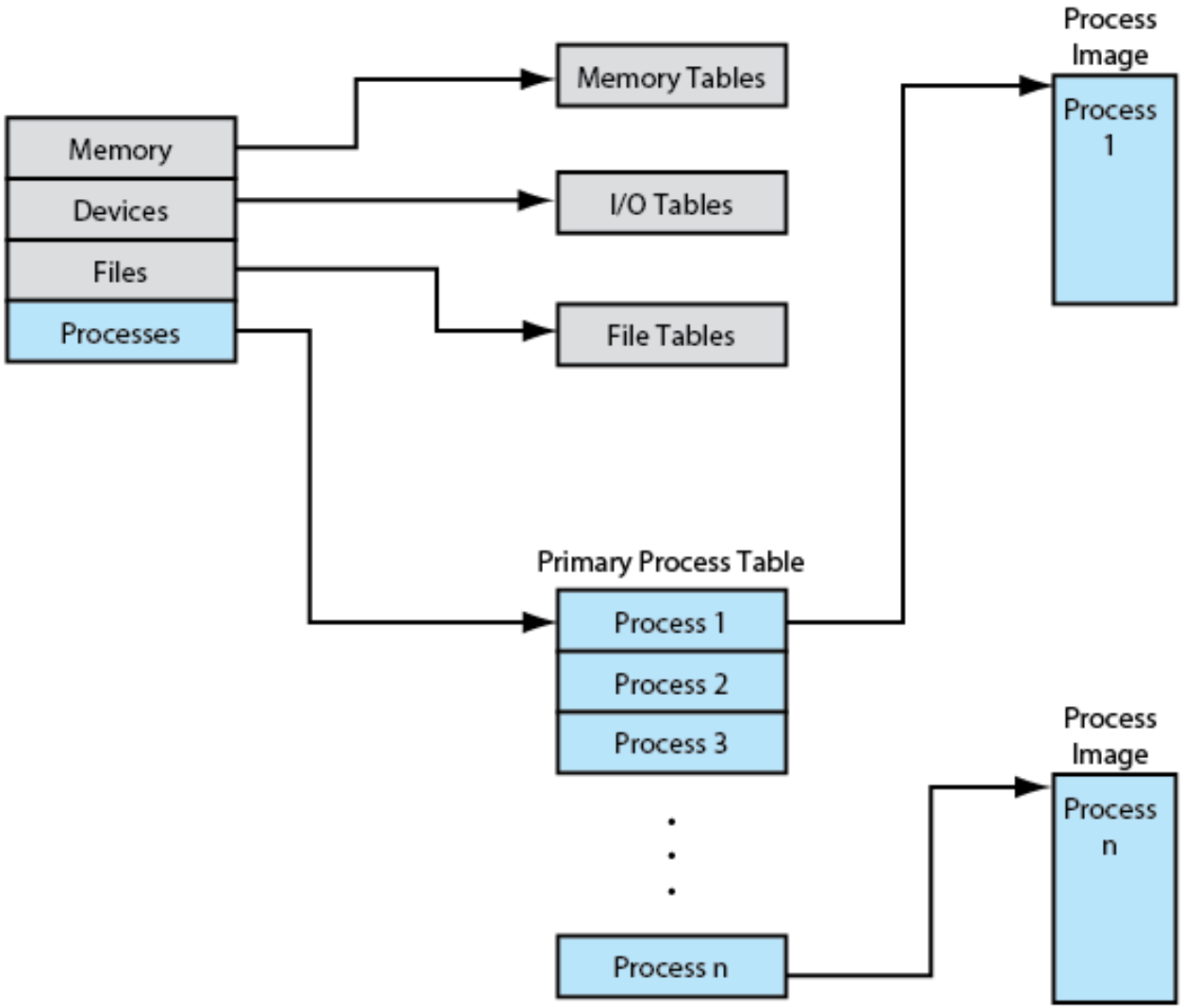
# Tablas de archivos

- Existencia de archivos
- Ubicación en la memoria secundaria
- Estado actual
- Atributos

# Tablas de procesos

- Donde se encuentra el proceso
- Atributos en el bloque de control de proceso
  - Programa
  - Datos
  - Pila

# Estructura general de las tablas de control del sistema operativo




Fuente: http://williamstallings.com/OS/OS5e-inst.html

# Imagen del proceso

## Elementos típicos de una imagen de proceso

- Datos del usuario
  - La parte modificable del espacio de usuario. Puede incluir datos de programa, área de pila de usuario, y programas que puedan ser modificados
- Programa de usuario
  - El programa a ejecutar
- Pila de sistema
  - Cada proceso tiene una o más pilas de sistema (LIFO) asociadas a él. Una pila se utiliza para almacenar los parámetros y las direcciones de retorno de los procedimientos y llamadas a sistema
- Bloque de control de proceso
  - Datos necesarios para que el sistema operativo pueda controlar los procesos

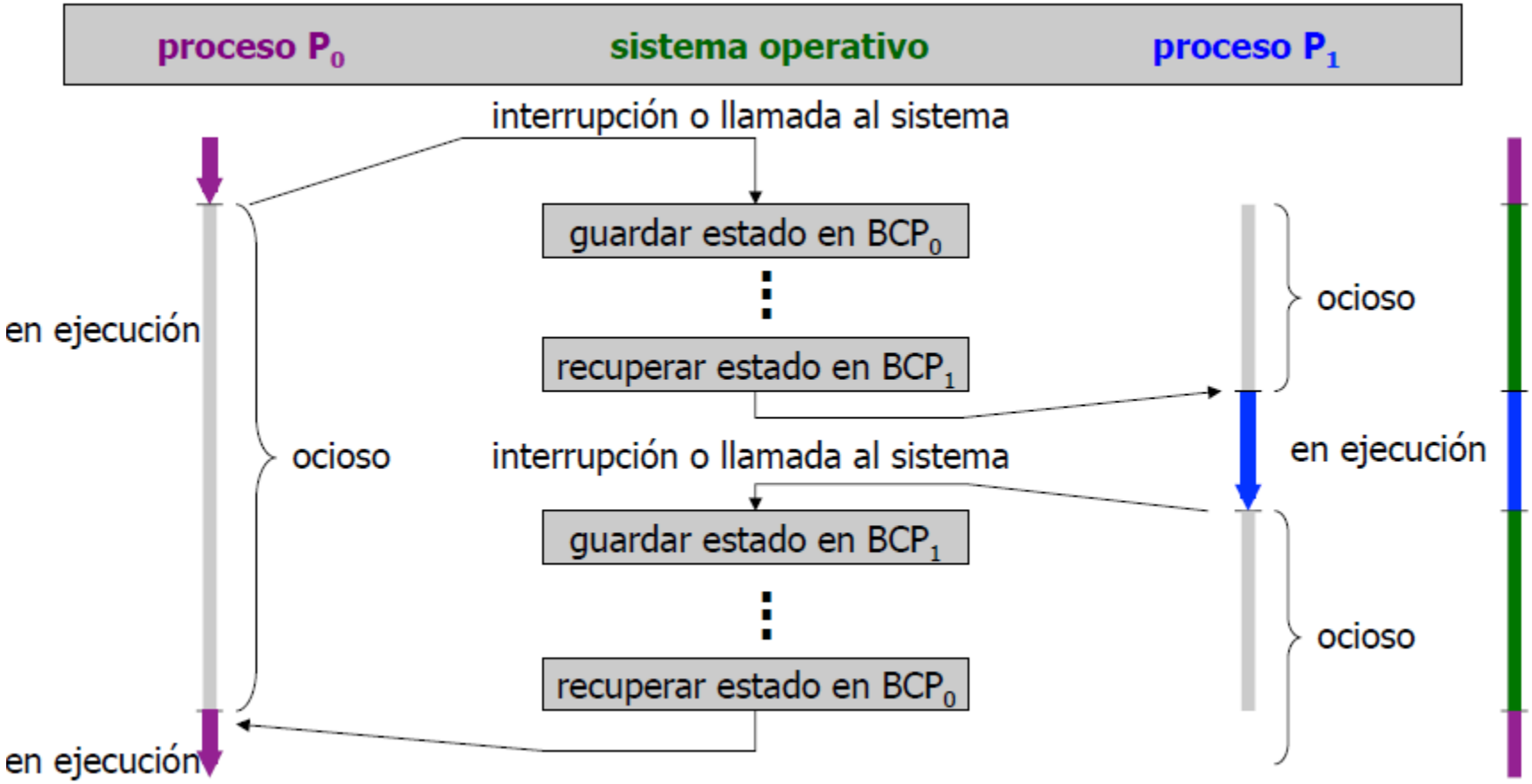
# Creación de un proceso

- Asignar un identificador de proceso único
- Asignar espacio para el proceso
- Inicializar bloque de control de proceso
- Establecer vínculos apropiados
  - Ejemplo: añadir un nuevo proceso a la lista vinculada utilizada para la cola de estados (Listos/Suspendidos)
- Crear o expandir otras estructuras de datos
  - Ejemplo: mantener un archivo de auditoría

# Cambio de contexto *(context switch)*

- Es la operación que consiste en desalojar a un proceso de la CPU y reanudar (o empezar) otro
- El módulo del SO encargado de esta acción es el *dispatcher*
- Hay que guardar el estado del proceso que **sale** en su BCP, y recuperar los registros del proceso que **entra**
  - Cada contexto se encuentra en cada BCP
- El cambio de contexto es **tiempo perdido**, así que debe ser lo más rápido posible
- El hardware a veces ofrece mecanismos para reducir el tiempo del cambio de contexto
  - En x86, la instrucción **PUSHA =** guarda todos los registros, bancos de registros alternativos, etc.

# Cambio de contexto *(context switch)*



# Cuando cambiar un proceso

- Interrupción del reloj
- El proceso se ha ejecutado para el intervalo de tiempo máximo permitido
- Interrupción de E/S
- Error de memoria
  - La dirección de memoria está en la memoria virtual por lo que debe ser introducido en la memoria principal
- Trap
  - Error o excepción
  - Puede hacer que el proceso se mueva al estado de salida
- Llamada de supervisor
  - Por ejemplo un archivo abierto

# Operaciones sobre procesos

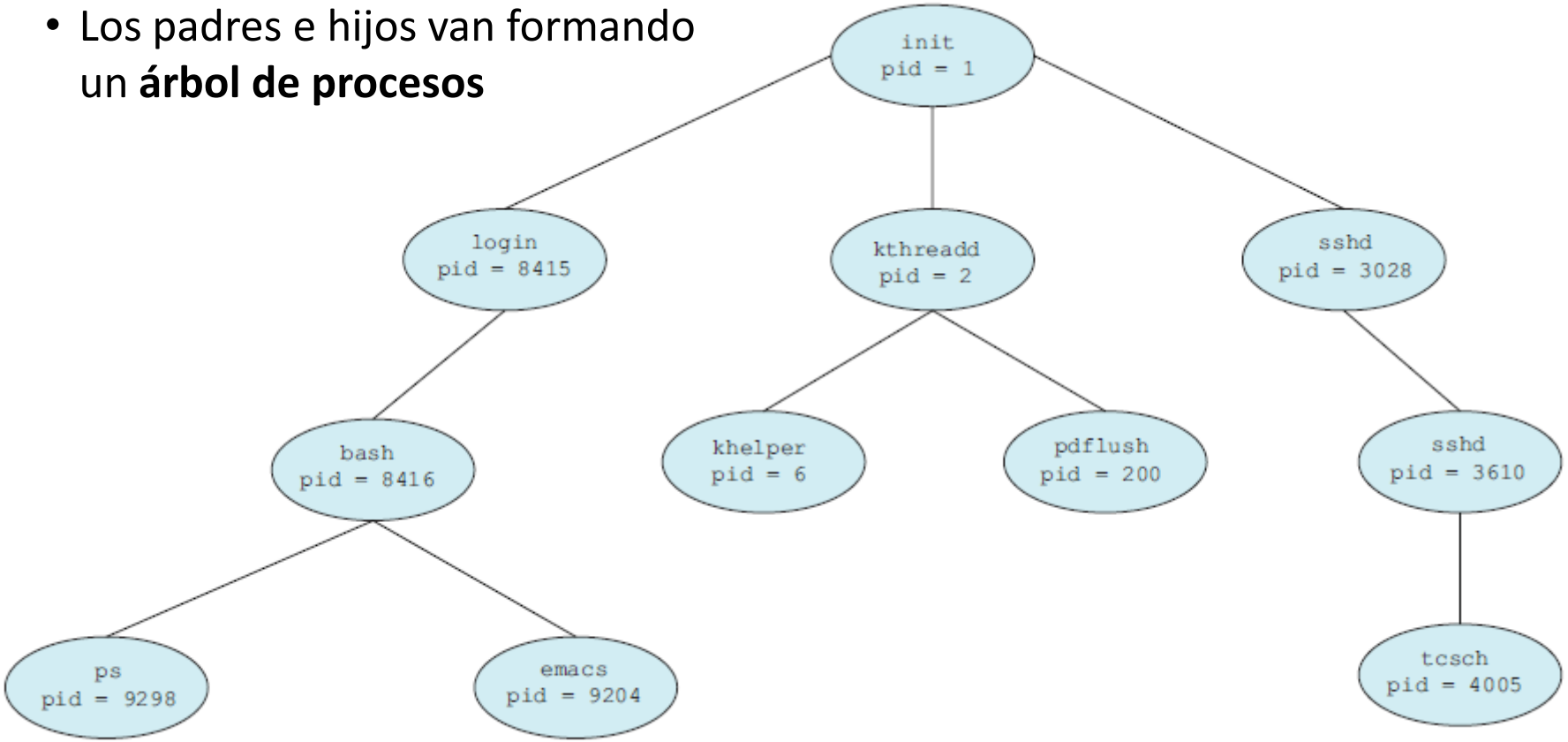
- Creación de nuevos procesos
- Terminación de procesos
- Comunicación entre procesos (IPC)

# Creación de nuevos procesos

- Un proceso puede crear otros nuevos procesos mientras se ejecuta
  - Para ello se utiliza una llamada al sistema específica para la creación de procesos
- El proceso creador se denomina “padre” y los nuevo procesos son los “hijos” de dicho proceso
- Cada uno de estos nuevos procesos pueden a su vez crear otros procesos, dando lugar a un “árbol” de procesos
- La mayoría de los SO identifican los procesos con un identificador de proceso unívoco *“pid”* que normalmente es un número entero

# Identificadores y árboles de procesos

- Los padres e hijos van formando un **árbol de procesos**




Árbol de procesos típico para el sistema operativo Linux

# Creación de nuevos procesos

Cuando un proceso crea otro proceso nuevo, en términos de ejecución, existen 2 posibilidades:

1. El padre continua ejecutándose concurrentemente con su hijo
2. El padre espera hasta que alguno o todos sus hijos hayan terminado de ejecutarse

También existen 2 posibilidades en función del espacio de direcciones del nuevo proceso:

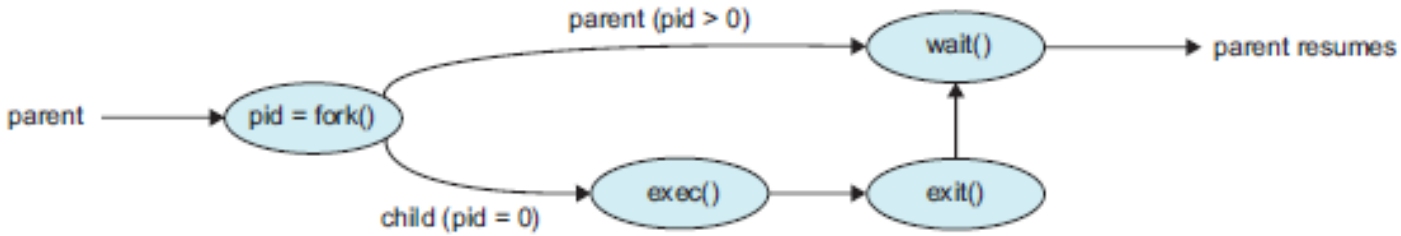
1. El proceso hijo es un duplicado del proceso padre (usa el mismo programa y los mismos datos del padre)
2. El proceso hijo carga un nuevo programa

# Creación de nuevos procesos

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid t pid;
    /* bifurca un proceso hijo */
    pid = fork();
    if (pid < 0) { /* se produjo un error */
           printf(stderr, "Fork Failed");
           return 1;
    }
    else if (pid == 0) { /* proceso hijo */
           execlp("/bin/ls","ls",NULL);
    }
    else { /* proceso padre*/
           /* el padre esperará hasta que el proceso hijo se complete */
           wait(NULL);
           printf("Child Complete");
    }
    return 0;
}
```

# Creación de nuevos procesos



# Terminación de procesos

- Un proceso termina
  - Cuando hace una llamada específica al sistema, por ejemplo ***exit()***
  - Cuando se genera una excepción y el SO decide abortarlo
  - Puede existir una llamada al sistema para abortar otro proceso, por ejemplo ***kill(), TerminateProcess()***
- Un padre puede terminar la ejecución de uno de sus hijos por distintas razones, como las siguientes:
  - El hijo ha excedido el uso de algunos de los recursos que ha sido asignado. Para determinar si esto ha ocurrido, el padre debe tener un mecanismo para inspeccionar el estado de sus hijos
  - La tarea asignada al hijo ya no es necesaria
  - El padre está saliendo y el sistema operativo no permite que un hijo continúe si su padre termina

# Comunicación entre procesos

Hay varias razones para permitir la cooperación entre procesos

- Compartir información
- Acelerar los cálculos
- Modularidad
- Conveniencia
- La cooperación entre procesos requiere mecanismos de comunicación interprocesos (IPC) que les permita intercambiar datos e información

# Comunicación entre procesos

- ¿Cómo intercambian datos los procesos?
  - Memoria compartida
  - Pase de mensajes
- Mecanismos clásicos (UNIX, Windows)
  - Archivos (ineficiente)
  - Zonas de memoria compartida
  - Tuberías (pipes)
  - Sockets
  - Llamas a procedimientos remotos (RPC)

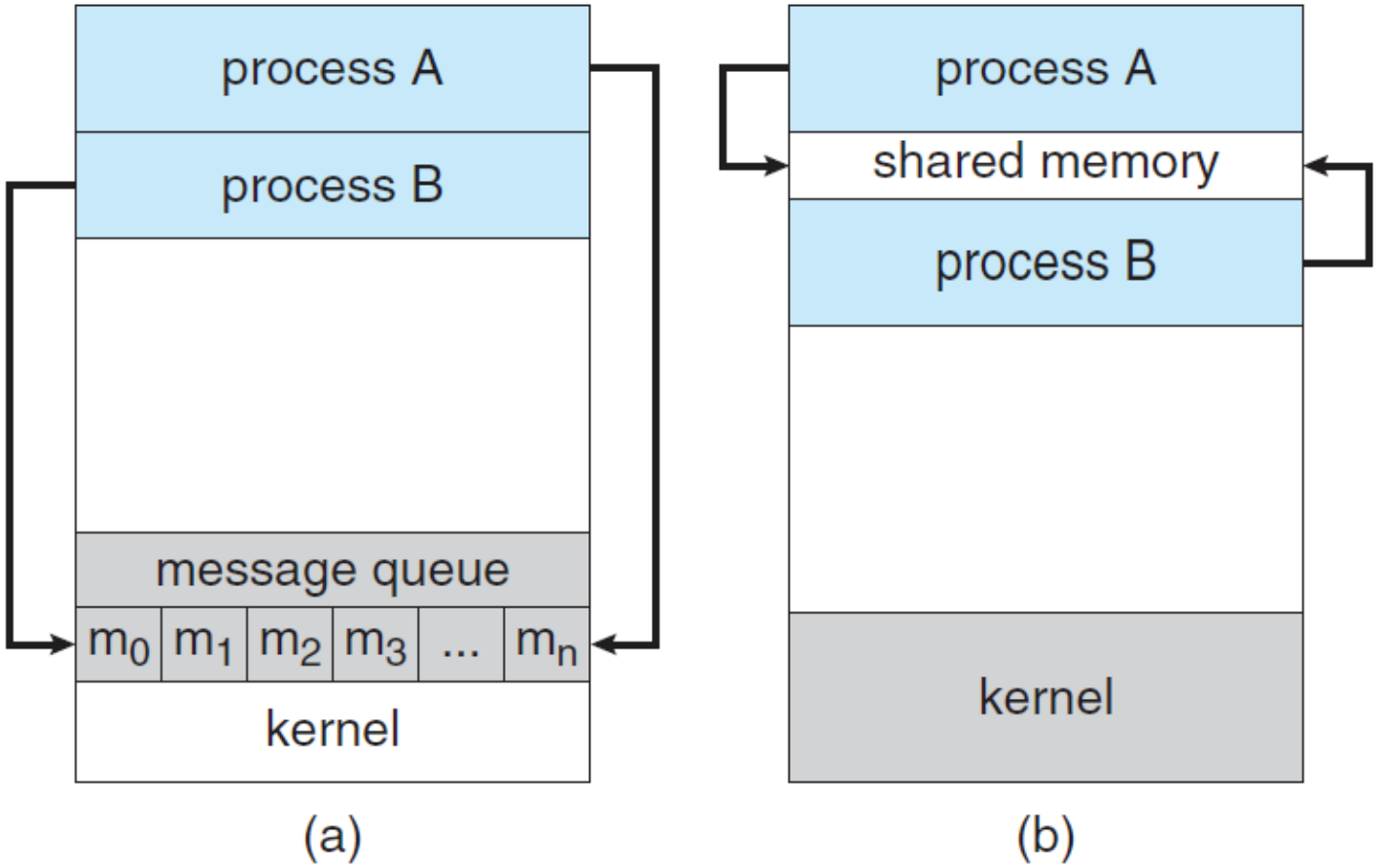
# IPC Memoria compartida

- Este método requiere que los procesos que se estén comunicando establezcan una **zona** de memoria compartida
- Normalmente esta zona compartida reside en el espacio de direcciones del proceso que crea el segmento de memoria compartida
- Otros procesos que deseen comunicarse usando este segmento de memoria compartida deben conectarse a su espacio de direcciones
- El formato de los datos y su ubicación están determinados por los procesos y no se encuentran bajo control del SO
- Los procesos también son responsables de verificar que no escriben simultáneamente en la misma posición

# IPC Paso de mensajes

- El SO proporciona los medios para que los procesos comuniquen y sincronicen sus acciones, sin compartir el espacio de direcciones
- Especialmente útil en entornos distribuidos
- Una facilidad de pase de mensajes proporciona al menos dos operaciones:
  - Envío de mensajes *(send)*
  - Recepción de mensajes *(receive)*

# IPC Modelos de comunicación




Communications models. (a) Message passing. (b) Shared memory.

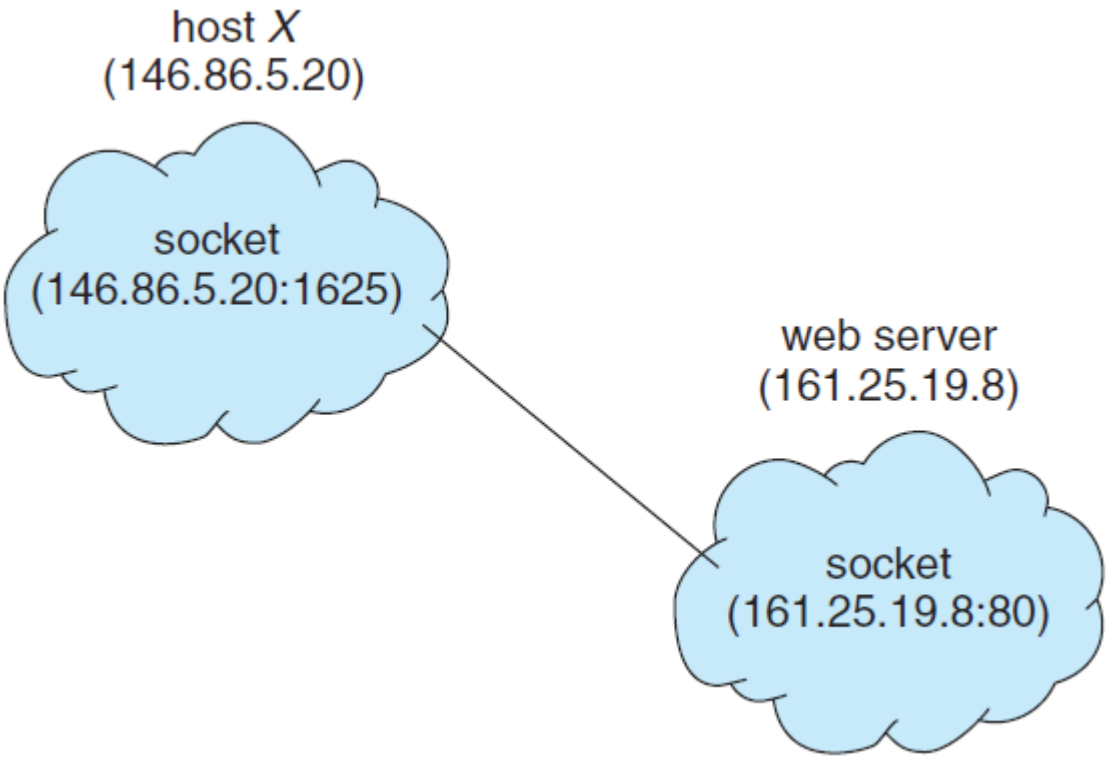
# Comunicación en sistemas Cliente-Servidor

- Los procesos pueden comunicarse utilizando la memoria compartida y el paso de mensajes
  - Estas técnicas también se pueden utilizar para la comunicación en sistemas cliente-servidor
- Tres estrategias de comunicación en sistemas cliente-servidor:
  1. *Sockets*
  2. Llamadas de procedimiento remoto *(Remote Procedure Call – RPC)*
  3. Tuberías *(pipes)*

# *Sockets*

- Un *socket* es un punto terminal de una comunicación
- Una pareja de procesos que se comunican a través de una red emplea una pareja de *sockets*, una para cada proceso
- Cada *socket* se identifica mediante una dirección IP junto a un número de puerto
- En general, los *sockets* se usan en una arquitectura cliente-servidor
- El servidor espera que entren solicitudes del cliente poniéndose a la escucha de un determinado puerto
- Una vez que recibe una solicitud, el servidor acepta una conexión del *socket* cliente y la conexión queda establecida

# Comunicación Cliente-Servidor usando *sockets*



# *Sockets*

- La comunicación mediante *sockets*, aunque común y eficiente, se considera una forma de comunicación de bajo nivel entre procesos distribuidos
- Una razón es que los *sockets* permiten que sólo un flujo no estructurado de bytes se intercambie entre los hilos de comunicación
  - Es la responsabilidad de la aplicación cliente o servidor imponer una estructura en los datos
- Métodos de comunicación de nivel superior:
  - Llamadas de procedimiento remoto (RPCs)
  - Tuberías

# Llamada a procedimiento remoto RPC

- Las RPC se diseñaron como un método para abstraer los mecanismos de llamada a procedimientos para ser utilizados en sistemas conectados en red
- La semántica de las llamadas a RPC permite a un cliente invocar un procedimiento de un remoto del mismo modo que lo haría con un local
- En contraste con los mensajes IPC, los mensajes intercambiados en la comunicación RPC están bien estructurados y, por lo tanto, ya no son sólo paquetes de datos

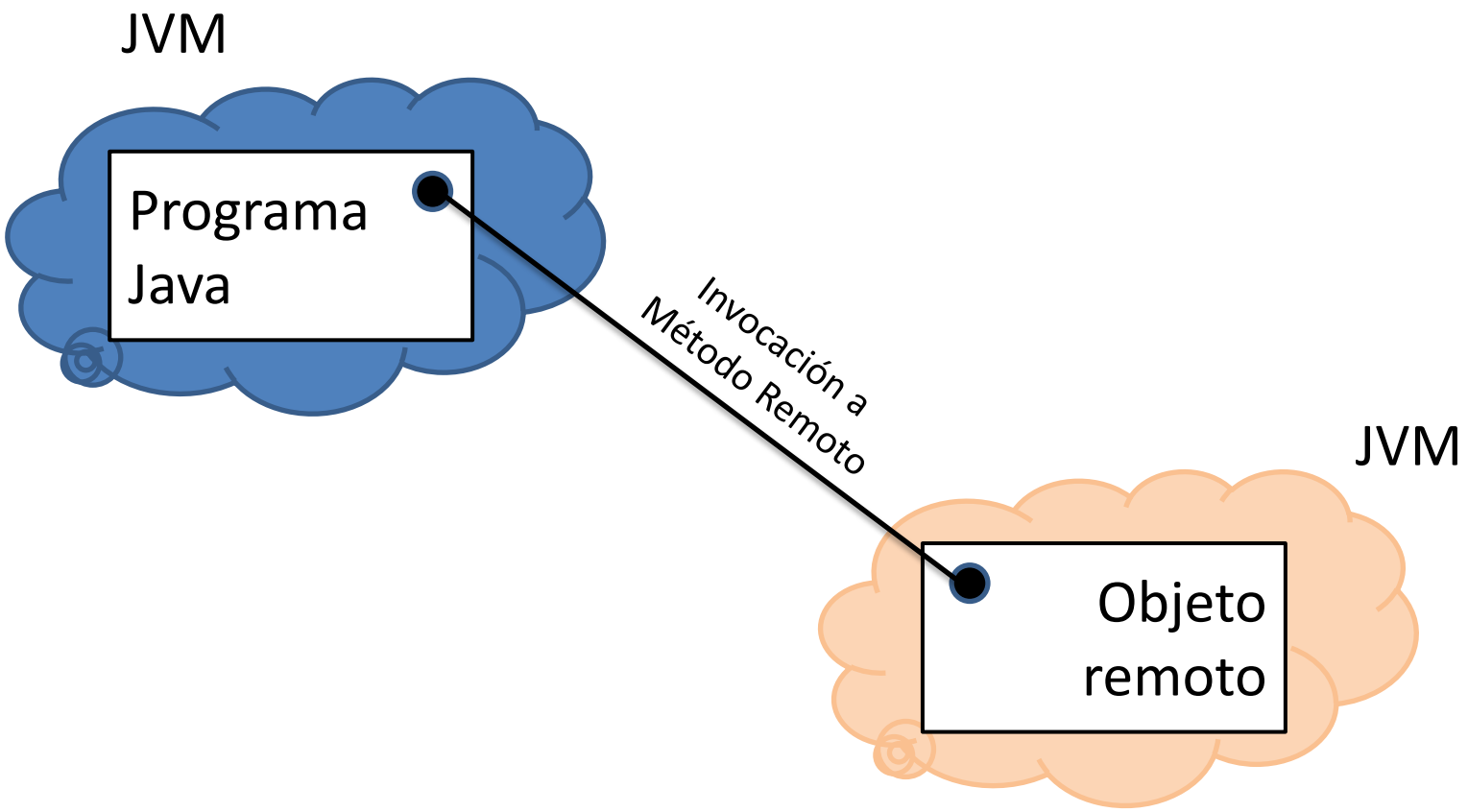
# Pipes

- Un tubería actúa como un conducto que permite que dos procesos se comuniquen
- Normalmente proporcionan una de las maneras más simples para que los procesos se comuniquen entre sí, aunque tienen algunas limitaciones
- En la implementación de una tubería, se deben considerar cuatro cuestiones:
  1. ¿La tubería permite la comunicación bidireccional, o es unidireccional?
  2. Si se permite la comunicación bidireccional, ¿es semi-dúplex (los datos pueden viajar sólo de una forma a la vez) o dúplex completo (los datos pueden desplazarse en ambas direcciones al mismo tiempo)?
  3. ¿Debe existir una relación (tal como padre-hijo) entre los procesos de comunicación?
  4. ¿Pueden las tuberías comunicarse a través de una red, o deben los procesos de comunicación residir en la misma máquina?

# Invocación a Métodos Remotos (RMI, Java)

- Funcionalidad Java similar a las RPC
- RMI permite a un programa Java invocar un método sobre un objeto remoto
  - Los objetos se consideran remotos si residen en una JVM diferente
  - RMI hace posible que los usuarios desarrollen aplicaciones Java distribuidas a través de una red

# Invocación a Métodos Remotos (RMI, Java)



# Bibliografía complementaria

- Sistemas Operativos: aspectos internos y principios de diseño (Stallings, 2005, 5ª edición)
  - Capítulo 3
- Fundamentos de los Sistemas Operativos (Silberschatz, Galvin, Gagne. 2006, 7ª edición)
  - Capítulos 3, 5