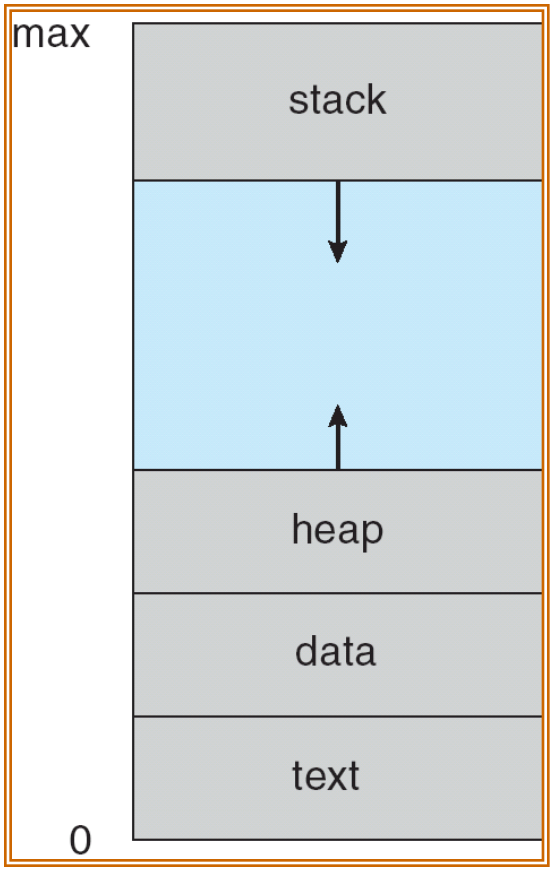
3: Procesos

Sistemas Operativos 1
Ing. Alejandro León Liu

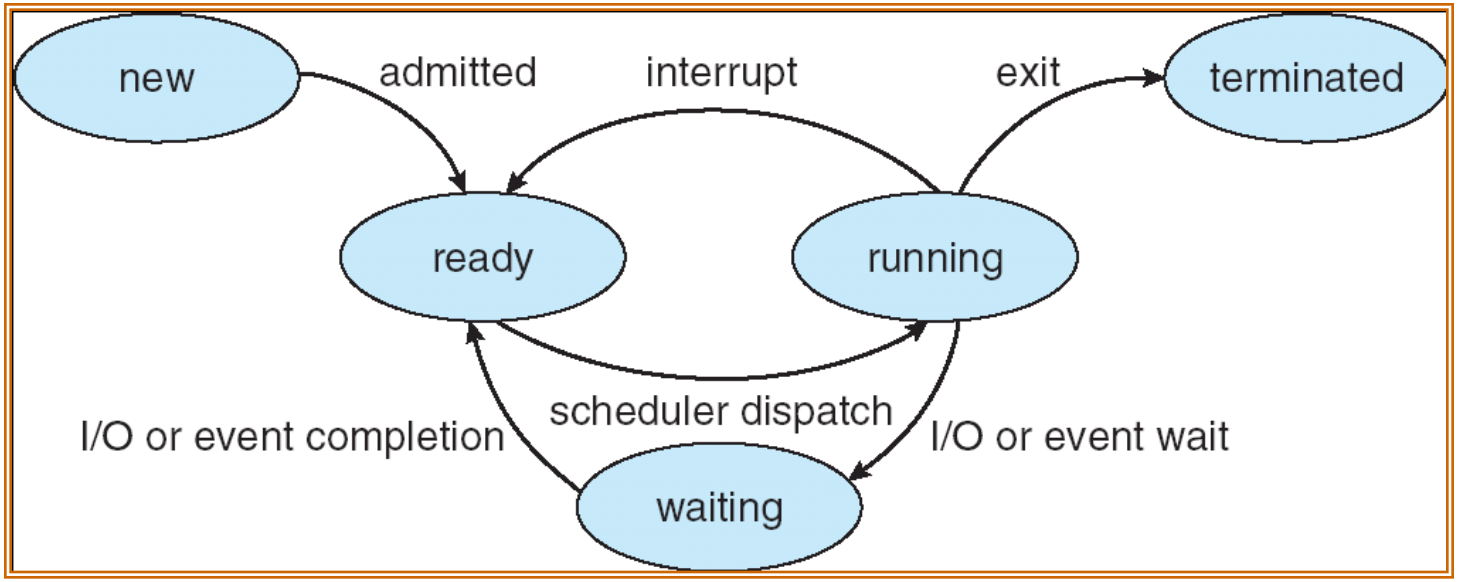
- ¿Qué son los procesos?
- Operaciones sobre procesos: calendarización, creación, terminación, comunicación.

PROCESOS

- Instancia de un programa.
  - Pueden existir múltiples instancias de un programa.
- Unidad de trabajo en sistemas multitasking.
- Compuesto de
  - Área o segmento de código
  - Datos
  - Registros
    - Program counter
  - Pila
  - Heap (Memoria asignada dinámicamente)


max
stack
heap
data
text
0

Estados

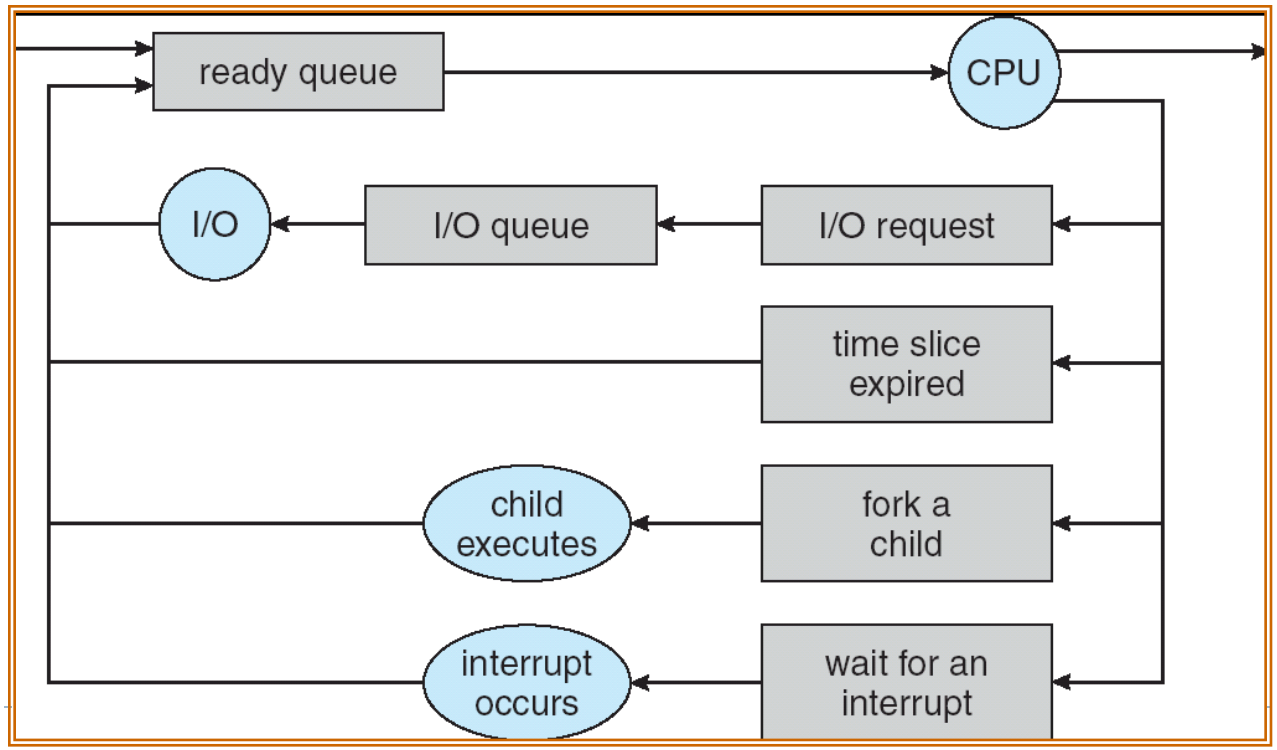

new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

Process Control Block

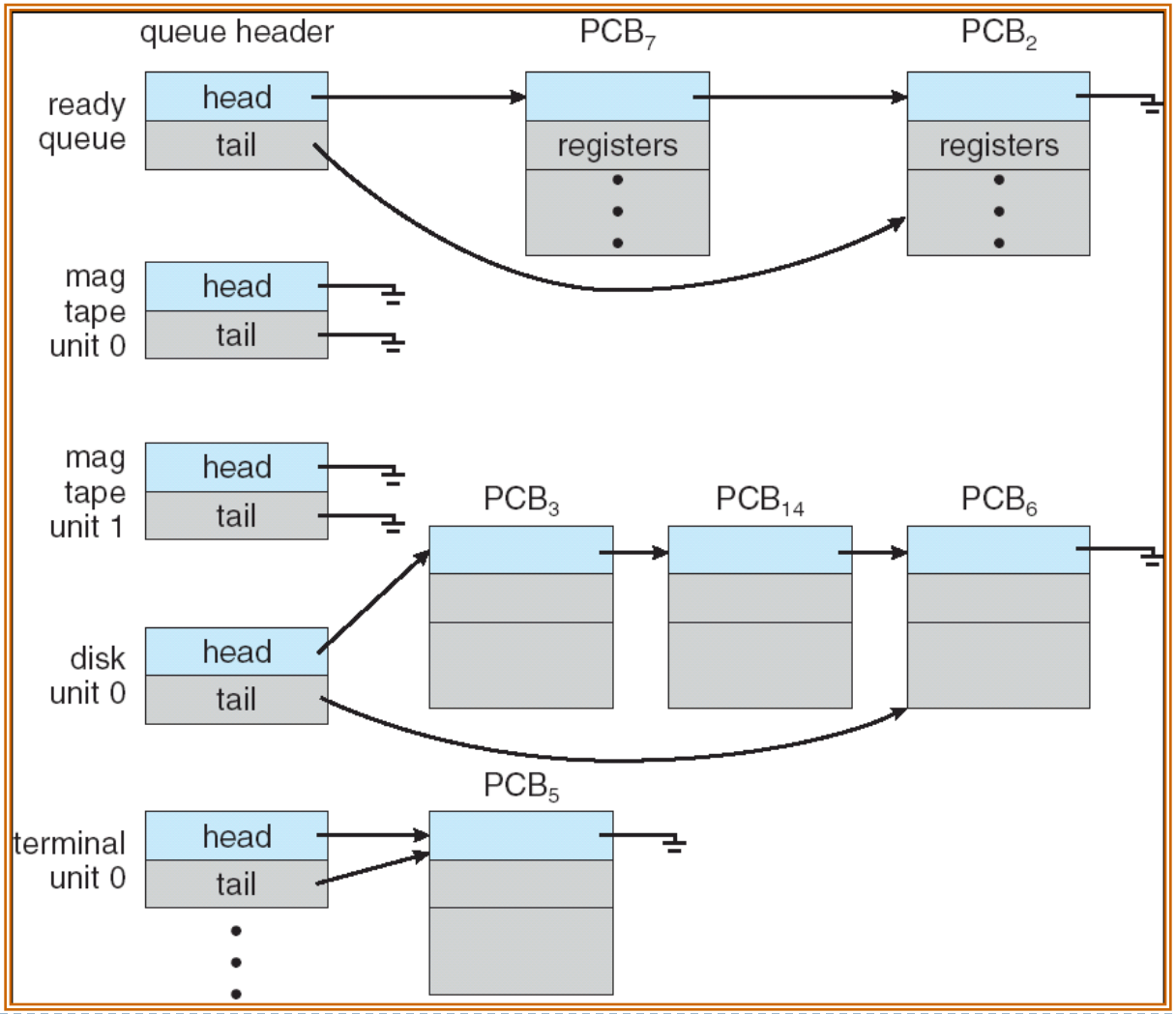
- Información acerca de un proceso
  - Estado
  - Registros
    - Program counter
  - Información de calendarización
    - Prioridad, colas
  - Información de manejo de memoria
    - Base y límite de memoria
    - Información de pagineo (Memoria virtual)
  - Información de rendimiento
    - Tiempos en CPU, tiempo ejecución
    - ID proceso
  - Información de I/O
    - Dispositivos y archivos en uso

CALENDARIZACIÓN

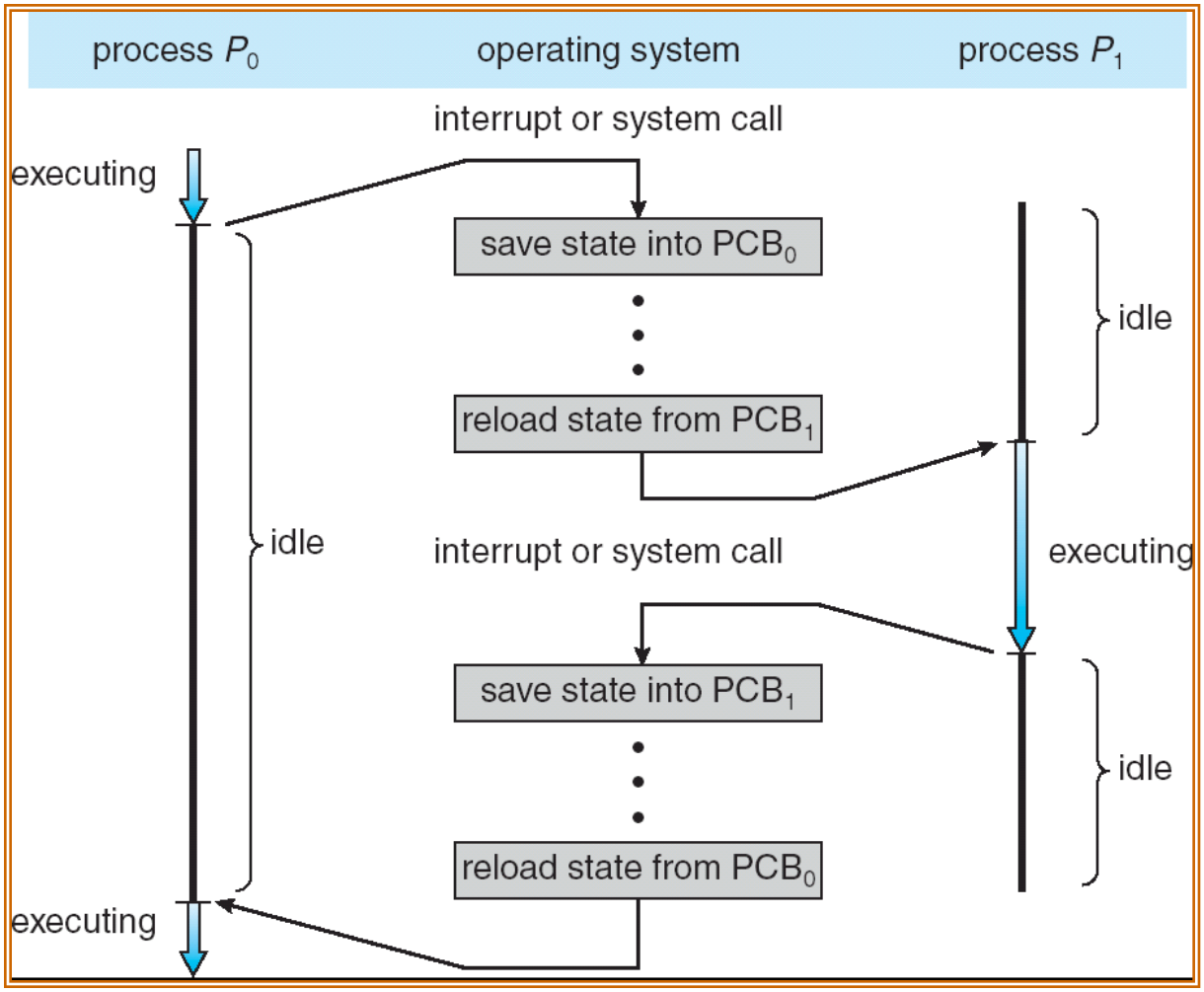
- Multiprogramación
  - Tener un proceso ejecutándose.
- Tiempo compartido (Multitasking)
  - Cambio frecuente: Interacción



- Colas
  - Nodos: PCB
  - Job queue: todos los procesos en el sistema
    - Job scheduler (largo plazo): selecciona procesos y los carga en memoria.
    - Frecuencia: grado de multiprogramación
  - Ready queue: procesos listos para ejecutarse
    - CPU scheduler (corto plazo): selecciona proceso (ready) y asigna CPU
    - Frecuencia: grado de multitasking
  - Device queue: procesos esperando por un dispositivo I/O
  - UNIX y Windows no tienen calendarizador a largo plazo.

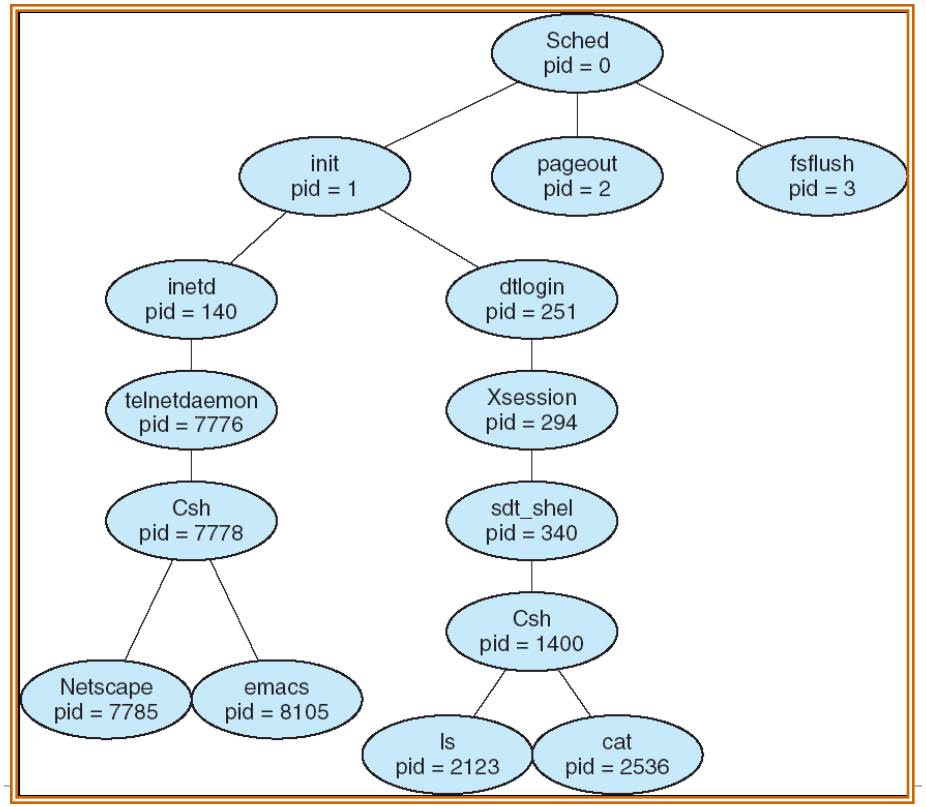

queue header
PCB7
PCB2
ready queue
head
tail
registers
registers
mag tape unit 0
head
tail
mag tape unit 1
head
tail
PCB3
PCB14
PCB6
disk unit 0
head
tail
PCB5
terminal unit 0
head
tail

- Context switch
  - Mecanismo para almacenar estado de proceso (PCB).
  - Permite restaurar procesos para continuar su ejecución.
  - Tiempo de context switch
    - Tiempo perdido para el usuario
    - Depende de
      - Arquitectura del procesador
        - Número de registros
        - PUSHA / POPA
        - Múltiples conjuntos de registros
      - Velocidad de memoria
      - Complejidad del S.O.
        - Manejo de memoria
        - Atributos de los procesos

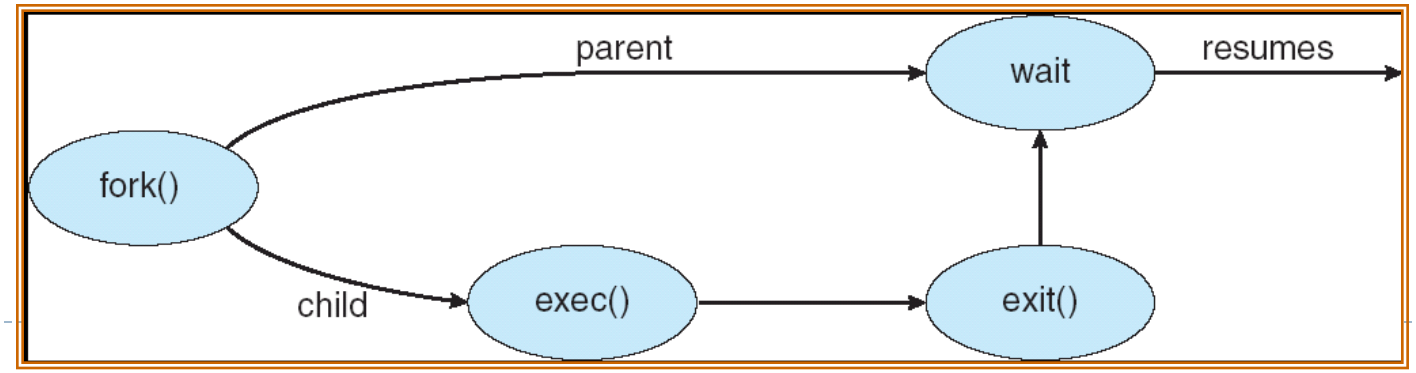

process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing

OPERACIONES SOBRE PROCESOS

- Creación de procesos
  - Proceso padre, crea proceso hijo: Árbol jerárquico
  - Identificador de proceso: pid



- Recursos del proceso hijo:
  - Obtener del S.O.
  - Restringido a recursos del padre
- Ejecución del proceso padre:
  - Ambos ejecutan concurrentemente
  - Padre espera a que algún o todos los hijos terminen
- Memoria del proceso hijo:
  - Hijo es un duplicado del padre (mismos datos y código)
  - Hijo tiene un nuevo programa



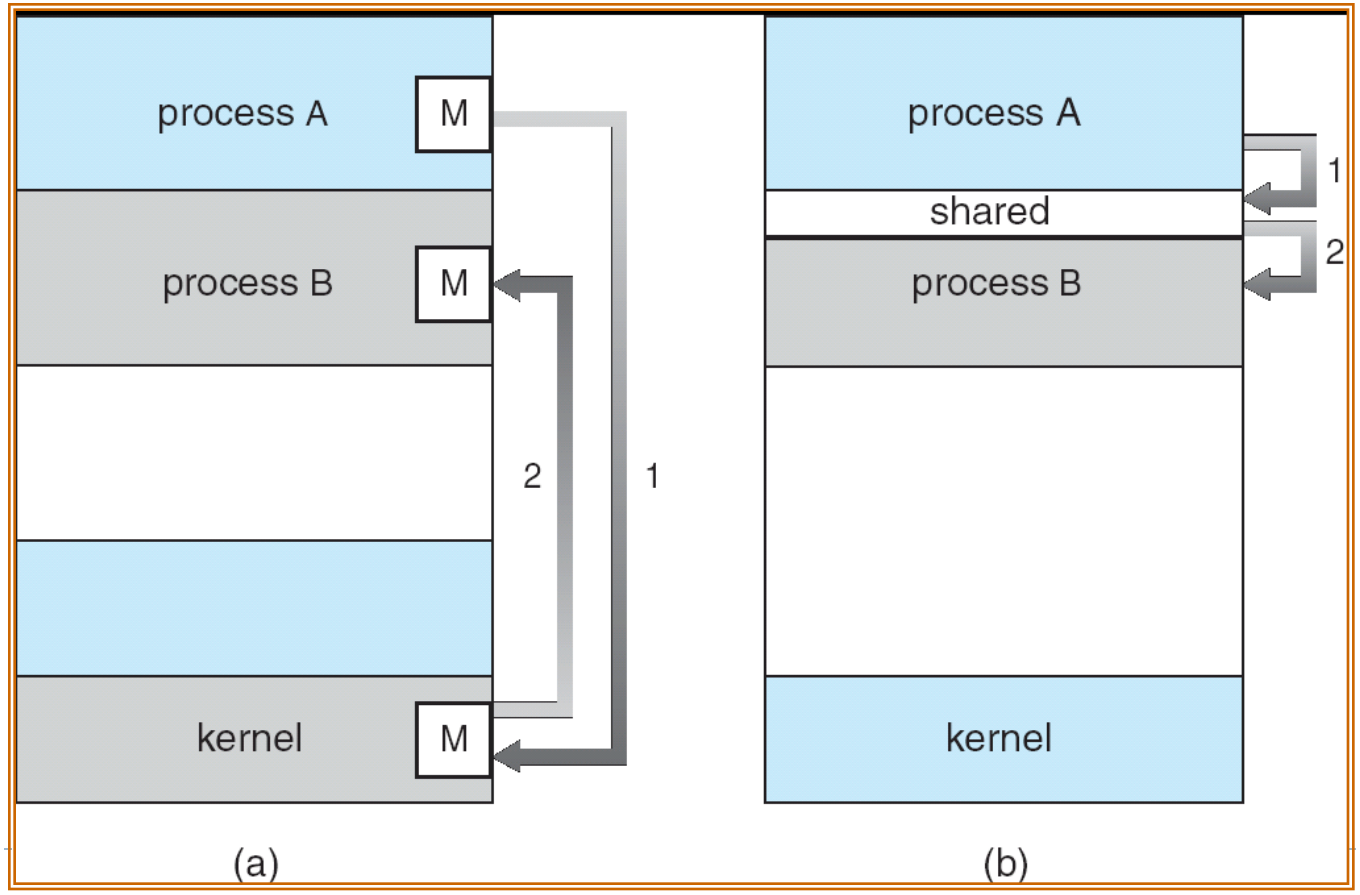
```
int main()
{
    Pid_t  pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
            fprintf(stderr, "Fork Failed");
            exit(-1);
    }
    else if (pid == 0) { /* child process */
            execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
            /* parent will wait for the child to complete */
            wait (NULL);
            printf ("Child Complete");
            exit(0);
    }
}
```

- Terminación de procesos
  - Exit(): retorna resultado
  - Recursos desasignados
  - Procesos padres terminan procesos hijos:
    - Proceso hijo sobrepasa uso de recursos compartidos
    - Tarea del proceso hijo ya no es necesaria
    - Proceso padre termina. S.O. no permite que procesos hijos ejecuten sin padre
      - Terminación en cascada

COMUNICACIÓN ENTRE PROCESOS

- Procesos independientes o cooperantes.
- Compartir información
- Separar en varios procesos
  - Eficiencia
  - Modularidad
- Conveniencia para el usuario
  - Multitasking: “ejecutar varios procesos a la vez”.
- Procesos cooperantes requieren de un mecanismo de IPC (Interprocess communication)

- a) Intercambio de mensajes
- b) Memoria compartida



- Memoria compartida
  - Mayor velocidad (simples accesos a memoria)
  - Problemas de protección
  - Problemas de sincronización
  - Establecer sección de memoria compartida con system calls
  - Comunicación a través de accesos a memoria

- Productor – Consumidor
  - Server – Cliente
  - Compilador – Ensamblador

- Buffers
  - Sin límite
    - Productor siempre puede producir elementos
  - Con límite.
    - Productor espera si está lleno
    - Consumidor espera si está vacío

- Contenidos de la memoria compartida
  - In: apunta al siguiente libre
  - Out: apunta al siguiente lleno

```
#define BUFFER_SIZE 10
Typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Productor

```
while (true) {
        /* Produce an item */
         while (((in = (in + 1) % BUFFER SIZE count)  == out) // FULL
          ;   /* do nothing -- no free buffers */
         buffer[in] = item;
         in = (in + 1) % BUFFER SIZE;
     {
```

- Consumidor

```
while (true) {
        while (in == out) // EMPTY
              ; // do nothing -- nothing to consume

      // remove an item from the buffer
      item = buffer[out];
      out = (out + 1) % BUFFER SIZE;
   return item;
    {
```

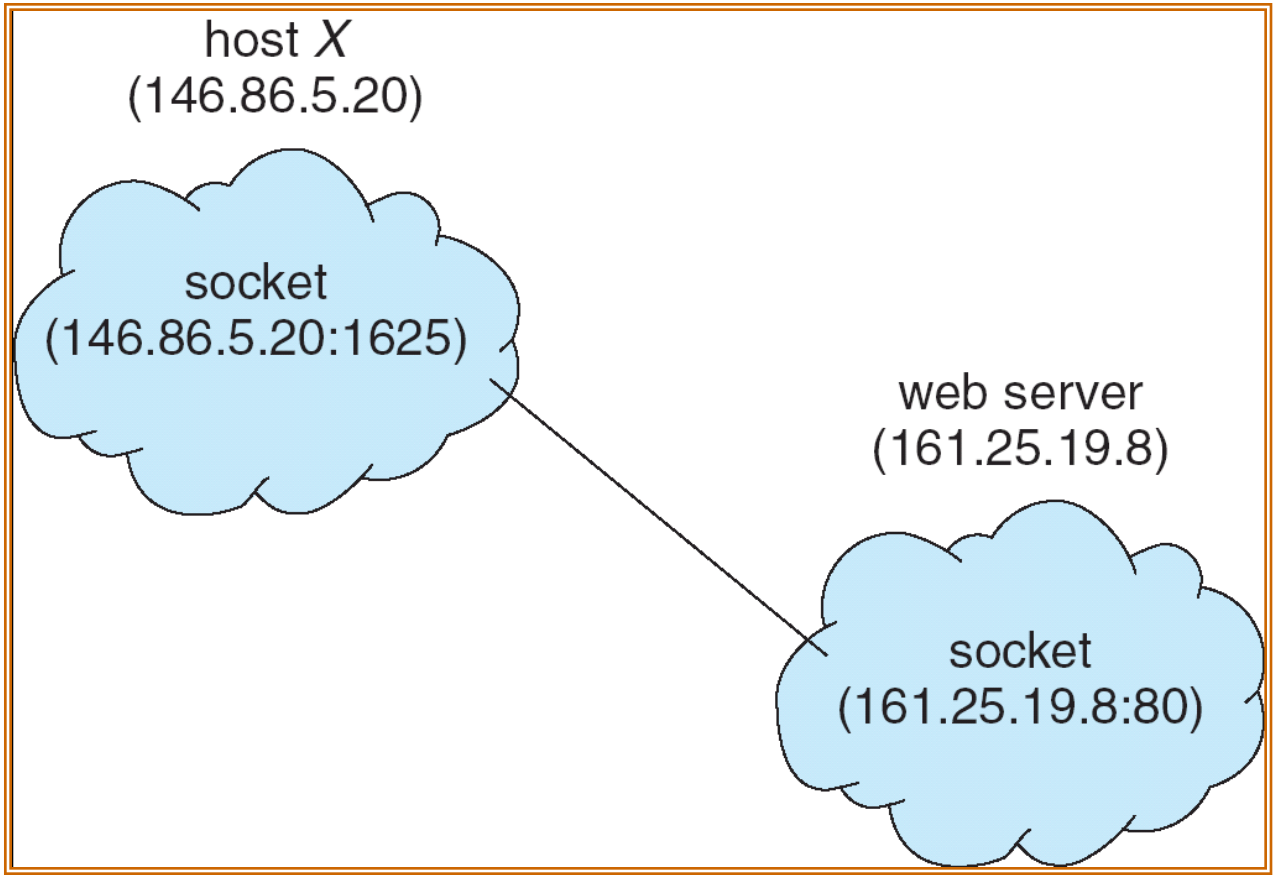
- Intercambio de Mensajes
  - Útil para poca información
  - Fácil implementación
  - Establecer enlace de comunicación
  - Send() y receive()
  - Formas de direccionamiento
    - Comunicación directa: Especificar nombre del emisor o receptor.
    - Comunicación indirecta. Mensajes recibidos y enviados a puertos.

- Sincronización
  - Sincrónica (blocking)
    - Send: el proceso se bloquea hasta que el mensaje es recibido.
    - Receive: el proceso se bloquea hasta que exista un mensaje.
  - Asincrónica (nonblocking)
    - Send: envía el mensaje y continúa.
    - Recieve: recibe un mensaje válido o null.
- Buffering
  - Sin capacidad. No hay cola. El mensaje debe ser recibido inmediatamente.
  - Con límite. Send() puede bloquearse hasta que haya espacio.
  - Sin límite. Send() nunca se bloquea.

EJEMPLOS DE IPC

- POSIX Shared memory
  - Crear sección de memoria compartida
    - segment_id = shmget(IPC_PRIVATE, size, read | write)
  - Adjuntar a sección de memoria
    - Shared_memory = (char *) shmat (id, NULL, read | write)
  - Escribir a memoria
    - Sprintf(shared_memory, “test”)

- Sockets
  - Cada extremo de una comunicación.
  - Compuesto de IP y puerto
  - Puertos comunes de servicios (Server)
    - HTTP: 80
    - FTP: 21
  - TCP: utilizando una conexión
  - UDP: sin conexión
  - Loopback (localhost): 127.0.0.1
  - Tipo de IPC más utilizado
    - Interfaz común sin importar S.O. ni lenguaje de programación.


host X
(146.86.5.20)
socket
(146.86.5.20:1625)
web server
(161.25.19.8)
socket
(161.25.19.8:80)

- Java Remote Method Invocation
  - Invocar métodos en objetos remotos (Otra JVM)
  - Parámetros
    - Locales (JVM local. Quien invoca): serialización de objetos (Enviar copia del objeto)
      - Interfaz java.io.Serializable
    - Remotos (JVM remota): por referencia