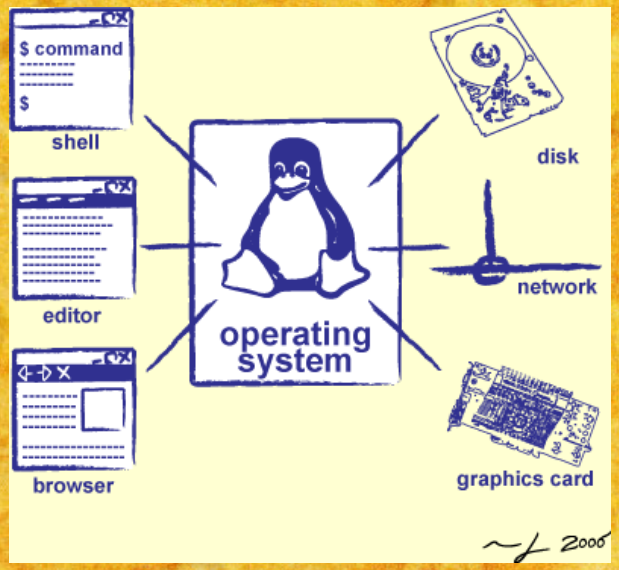
3.- Procesos


$ command
$
shell
editor
browser
operating system
disk
network
graphics card
2006

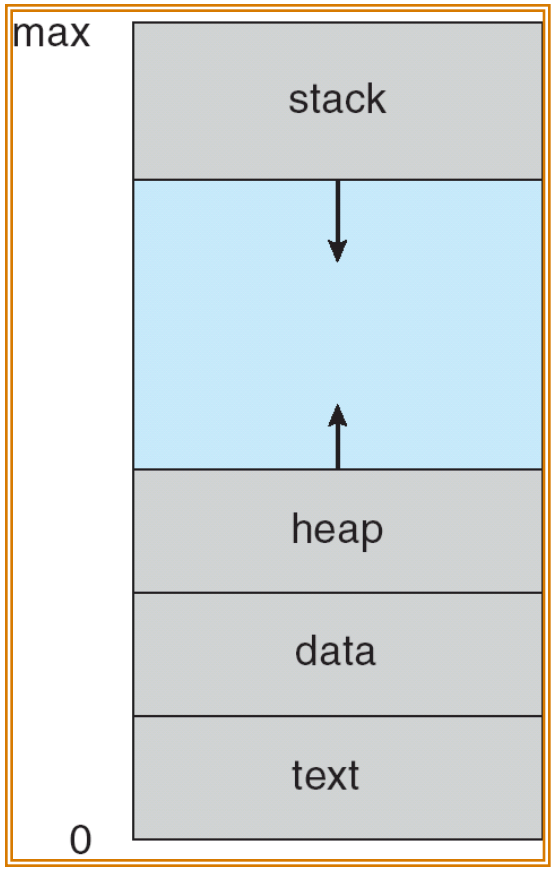
Capítulo 3: Procesos

- Concepto de Proceso
- Despacho (calendarización) de Procesos
- Operaciones en Procesos
- Procesos en cooperación
- Compunicación Interprocesos
- Communicación en sistemas Cliente-Servidor

Concepto de Proceso

- El SO ejecuta diversos programas:
  - Sistema de Batch – jobs
  - Sistema de tiempo compartido (Time-sharing) – programas o tareas de los usuarios
- Los términos *job* y *proceso* son casi sinónimos
- Proceso – programa en ejecución (secuencial)
- Un proceso incluye:
  - Contador de programa (PC)
  - stack
  - Sección de datos

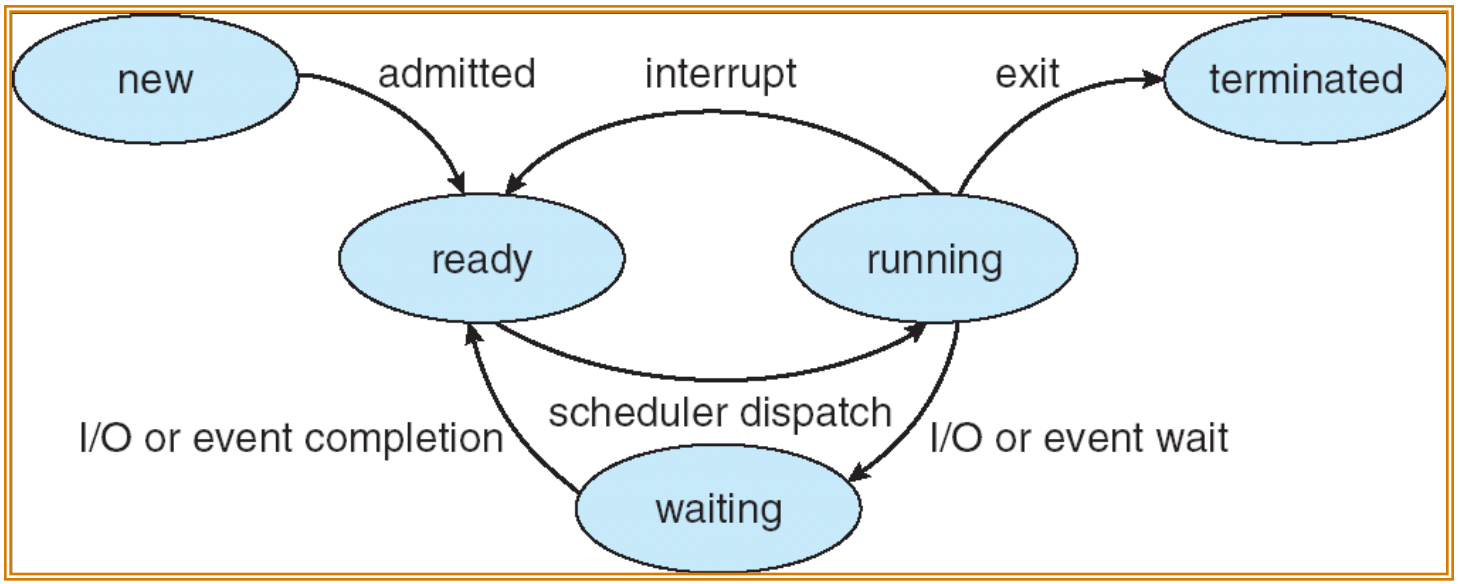
Proceso en Memoria



Estado de un Proceso

- Un proceso cambia de *estado* durante su "vida"
  - **new**: Se está creando
  - **running**: En ejecución
  - **waiting**: En espera de que ocurra algún evento
  - **ready**: Listo para ue se le asigne a un procesador
  - **terminated**: Ha terminado ejecución

Diagram de Estados de un Proceso

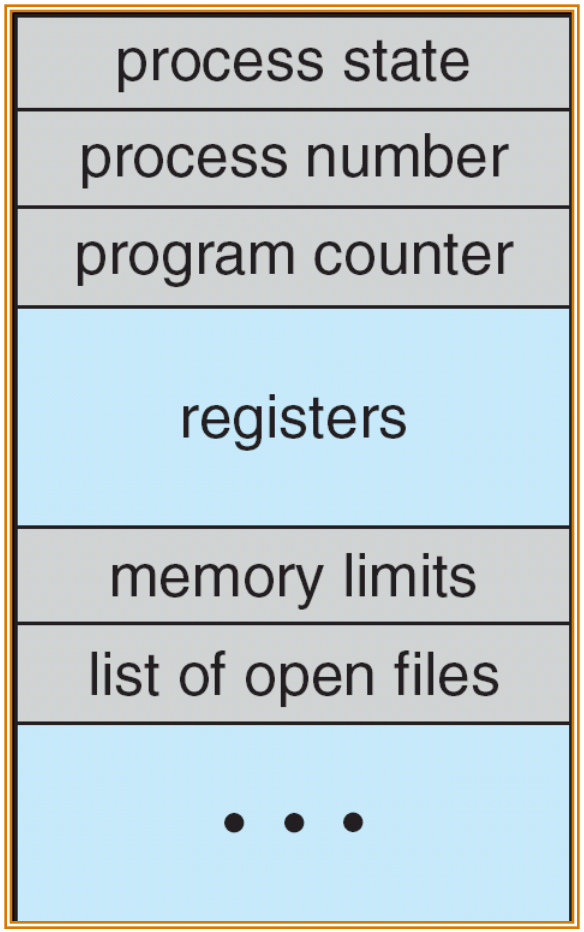


Process Control Block (PCB)

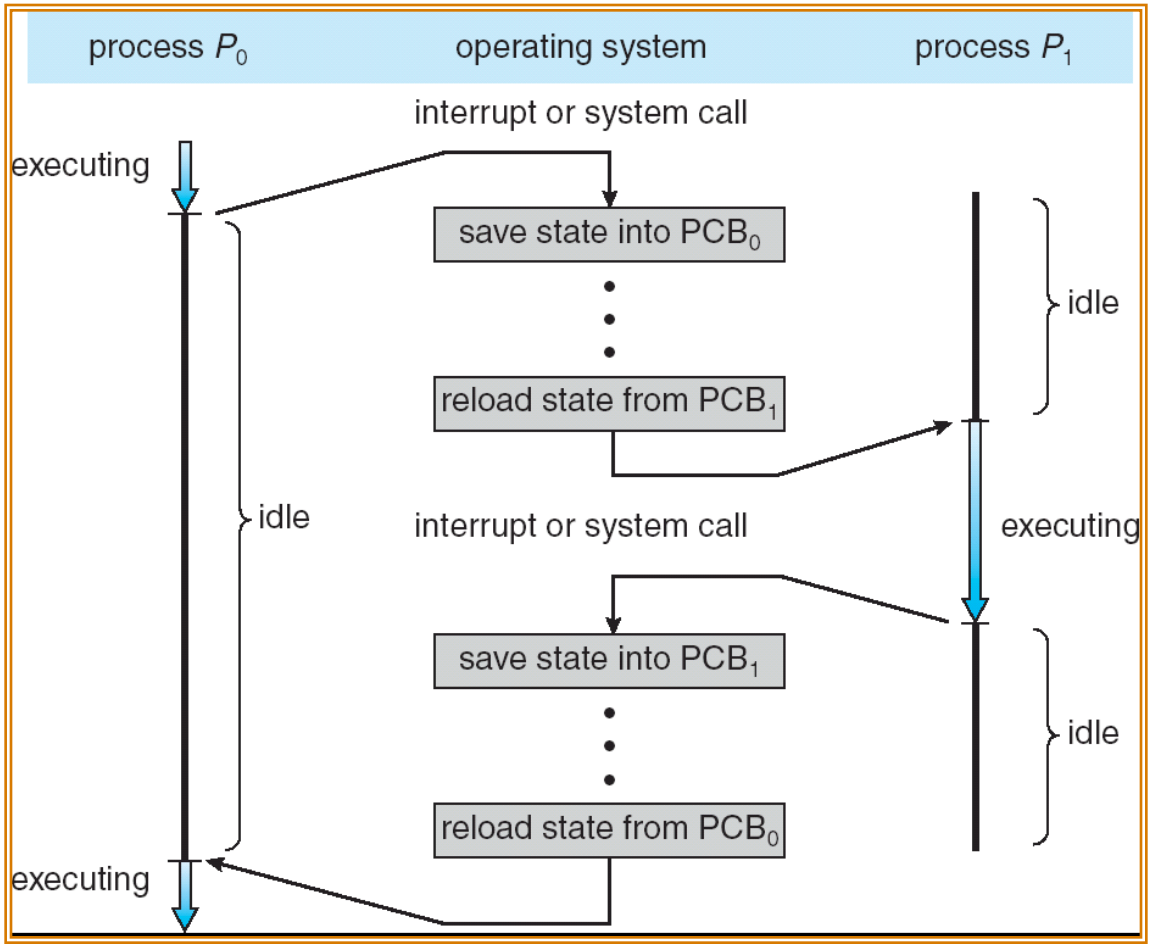
Información asociada con cada proceso

- Estado
- PC
- Registros del CPU
- Información de CPU scheduling
- Información de Memory-management
- Información de Accounting
- Información de status de I/O

Process Control Block (PCB)



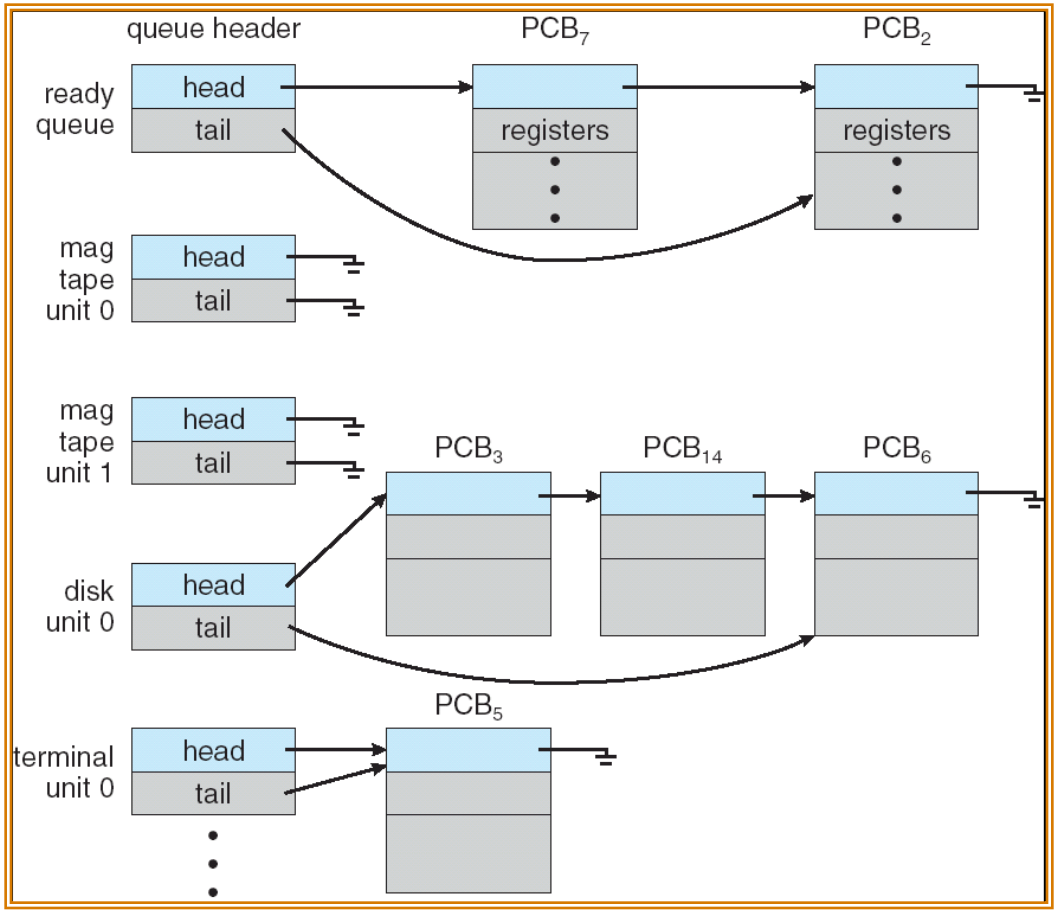
Cambio de CPU de Proceso a Proceso



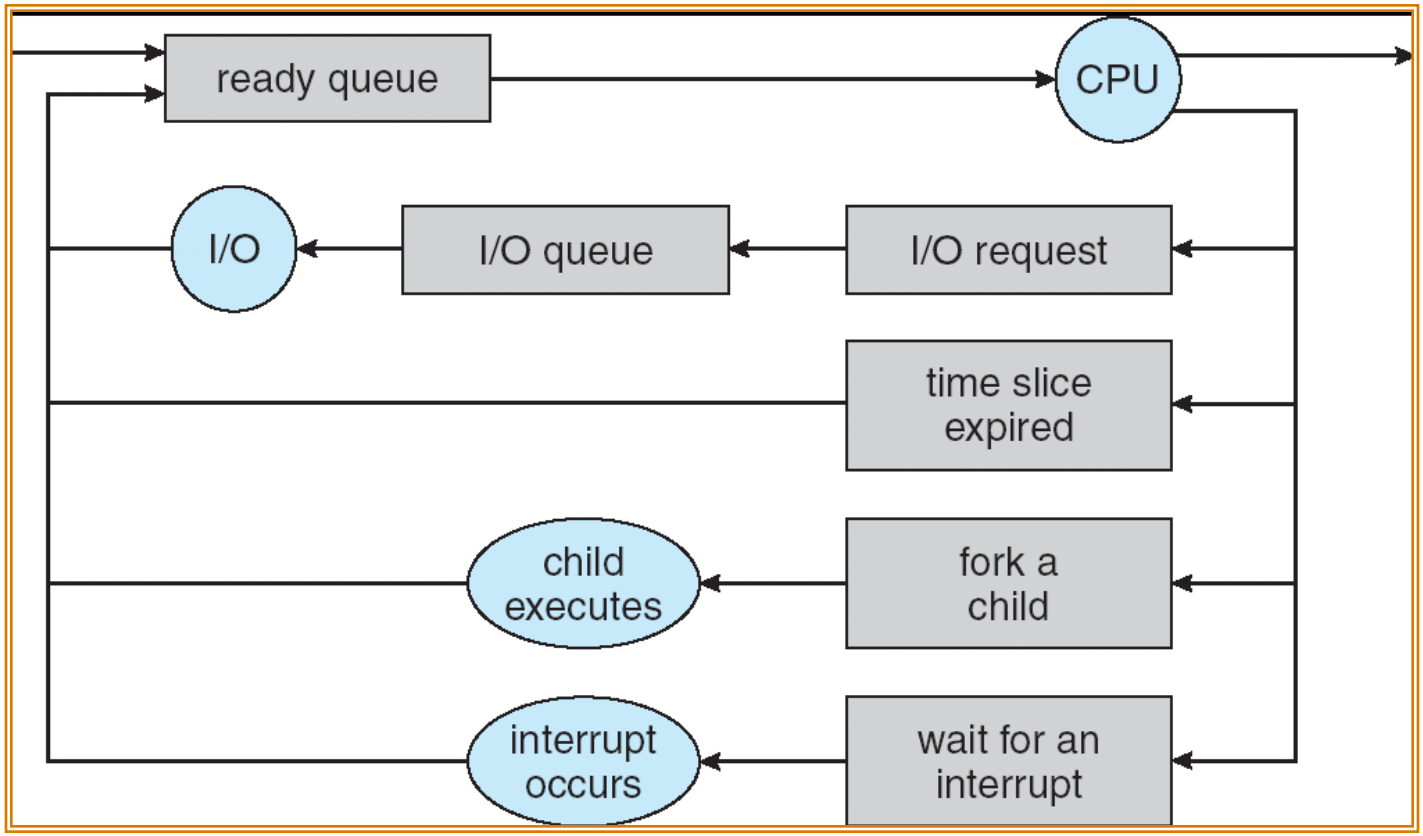
Colas del Despachador de Procesos

- **Job queue** – todos los procesos del sistema
- **Ready queue** –procesos residentes en memoria principal, listos y esperando ejecución
- **Device queues** – procesos en espera de un dispositivo I/O
- Los procesos migran entre las colas

Colas del Sistema (Queues)



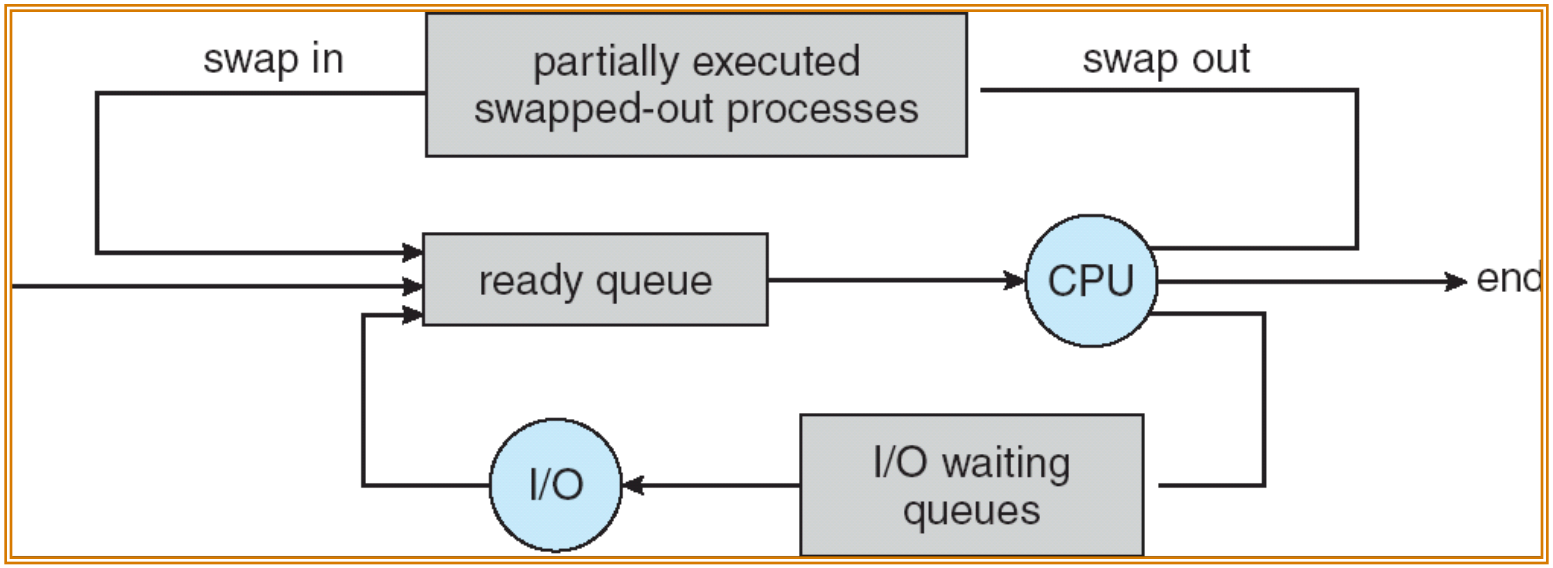
Despacho de Procesos



Despachadores (Schedulers)

- **Long-term scheduler** (job scheduler) – selecciona que procesos deben ser puestos en la cola *ready queue*
- **Short-term scheduler** (CPU scheduler) – selecciona que proceso ejecutar y asigna el CPU

Despachador de Mediano Plazo



Despachadores (Cont.)

- El de corto plazo (Short-term scheduler) se invoca frecuentemente (milisegundos) ⇒ (debe ser rápido)
- El de largo plazo (Long-term scheduler) no se invoca tan frecuentemente (segundos, minutos) ⇒ (puede ser lento)
- El de largo plazo controla el *grado de multiprogramación*
- Los proceso pueden clasificarse como:
  - **I/O-bound process** – pasa más tiempo en I/O que en cómputo (muchos CPU bursts cortos)
  - **CPU-bound process** – pasa más tiempo en cómputo (pocos CPU bursts largos)

Context Switch

- Cuando el CPU cambia a otro proceso, el sistema debe guardar el estado del proceso anterior y cargar el estado del nuevo proceso
- Context-switch genera overhead (costo) – el sistema no hace trabajo útil mientras cambia procesos
- El tiempo de cambio depende del soporte de hardware disponible

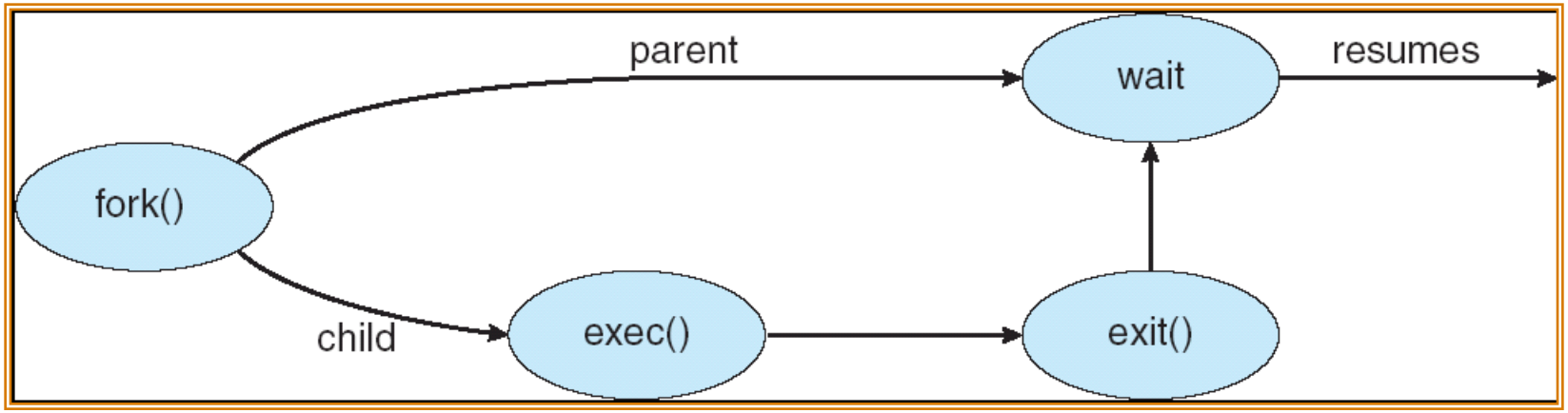
Creación de Procesos

- Procesos *padres* crean procesos *hijos*, lo cuales crean otros procesos, los cuales, a su vez, crean otros procesos, formando un árbol de procesos
- Compartir recursos
  - Los padres comparten todos los recursos
  - Los hijos comparten un subconjunto de los recursos de los padres
  - Padres e hijos no comparten recursos
- Ejecución
  - Padres e hijos ejecutan concurrentemente
  - Los padres esperan a que los hijos terminen

Creación de Procesos (Cont.)

- Espacio de direcciones
  - El hijo duplica el del padre
  - El hijo tiene un programa cargado en su espacio de direcciones
- Ejemplos UNIX
  - **fork** (system call) crea un nuevo proceso
  - **exec** (system call) se usa después de **fork** para reemplzar el espacio de memoria del proceso con un programa nuevo

Creación de Procesos (Cont.)



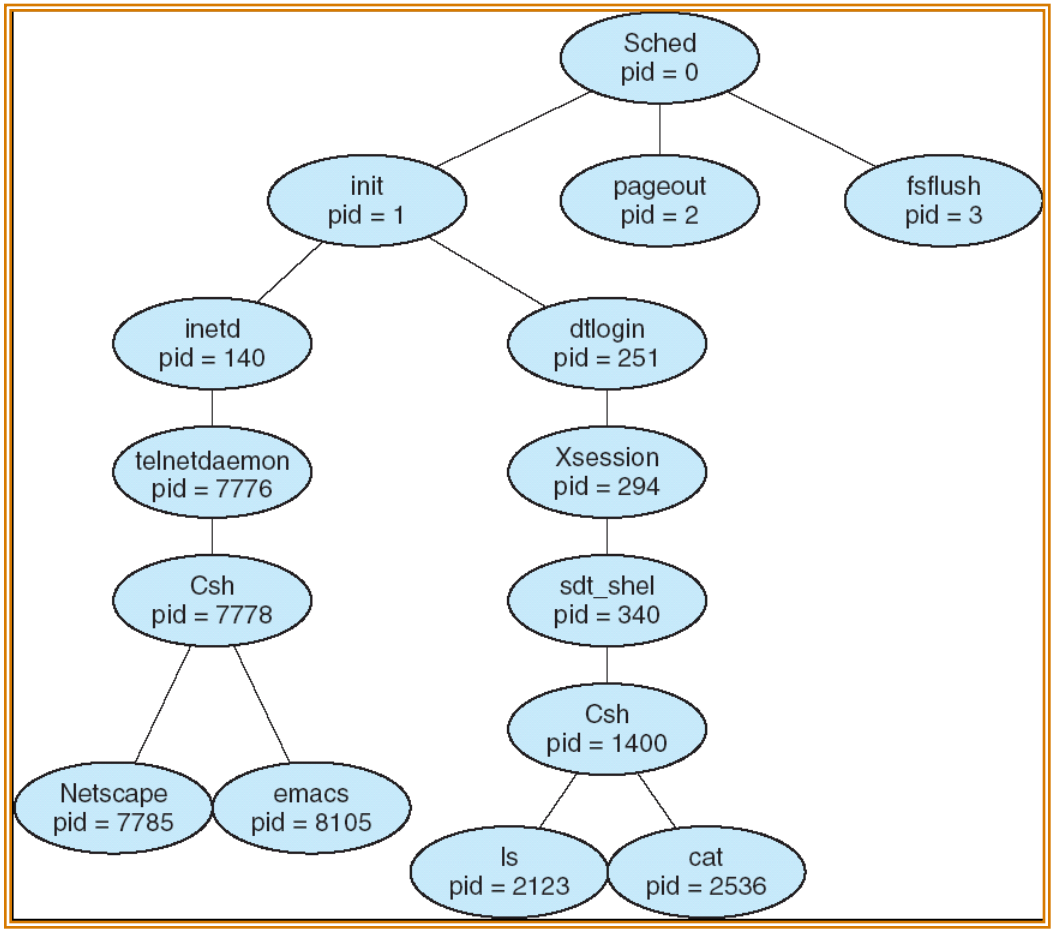
Programa C Creando Procesos (Forking)

```
int main() {
pid_t  pid;
  pid = fork();
  if (pid < 0) { /* error */
       fprintf(stderr, "Fork
  Failed");
       exit(-1);}
  else if (pid == 0) { /* child */
       execlp("/bin/ls", "ls", NULL);
  }
```

Programa C Creando Procesos (Forking)

```
    else { /* parent */
         /* parent will wait for the
    child to complete */
         wait (NULL);
         printf ("Child Complete");
         exit(0);
    }
}
```

Árbol de Procesos en Solaris



Terminación de Procesos

- Un proceso ejecuta la última instrucción y le pide al SO que lo borre (**exit**)
  - Envía datos del hijo al padre (via **wait**)
  - Los recursos del proceso son desasignados por el SO
- Un padre puede terminar la ejecución de un proceso hijo (**abort**)
  - Hijo se ha excedido en los recursos asignados
  - La tarea asignada al hijo ya no se necesita
  - Si el padre termina
    - Algunos SOs no permiten al hijo que continue
      - Todos los hijos terminan – *terminación en cascada*

Procesos en Cooperación

- Procesos **Independientes** no pueden afectar o ser afectados por otros procesos
- Procesos en **Cooperación** pueden afectar o ser afectados por otros procesos
- Ventajas de cooperación
  - Compartir información
  - Rapidez de ejecución (speed-up)
  - Modularidad
  - Conveniencia

Problema Productor-Consumidor

- Paradigma para procesos cooperantes, *productor* produce información usada por el *consumidor*
  - *unbounded-buffer* no establece límite en el tamaño del buffer
  - *bounded-buffer* asume que el buffer está limitado en tamaño

Bounded-Buffer – Memoria Compartida

- Datos compartidos

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

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

- Solución correcta, pero solo se pueden usar BUFFER_SIZE-1 elementos

Bounded-Buffer – Método Insert()

```
while (true) {
   /* Produce an item */

      while ((((in+1) % BUFFER_SIZE) == out);
                  /* do nothing -- no free buffers */

    buffer[in] = item;

    in = (in + 1) % BUFFER_SIZE;

   }
```

Bounded Buffer – Método Remove()

```
while (true) {
        while (in == out);
                // do nothing -- nothing to
consume
     // remove an item from the buffer
     item = buffer[out];
     out = (out + 1) % BUFFER_SIZE;
return item;
   }
```

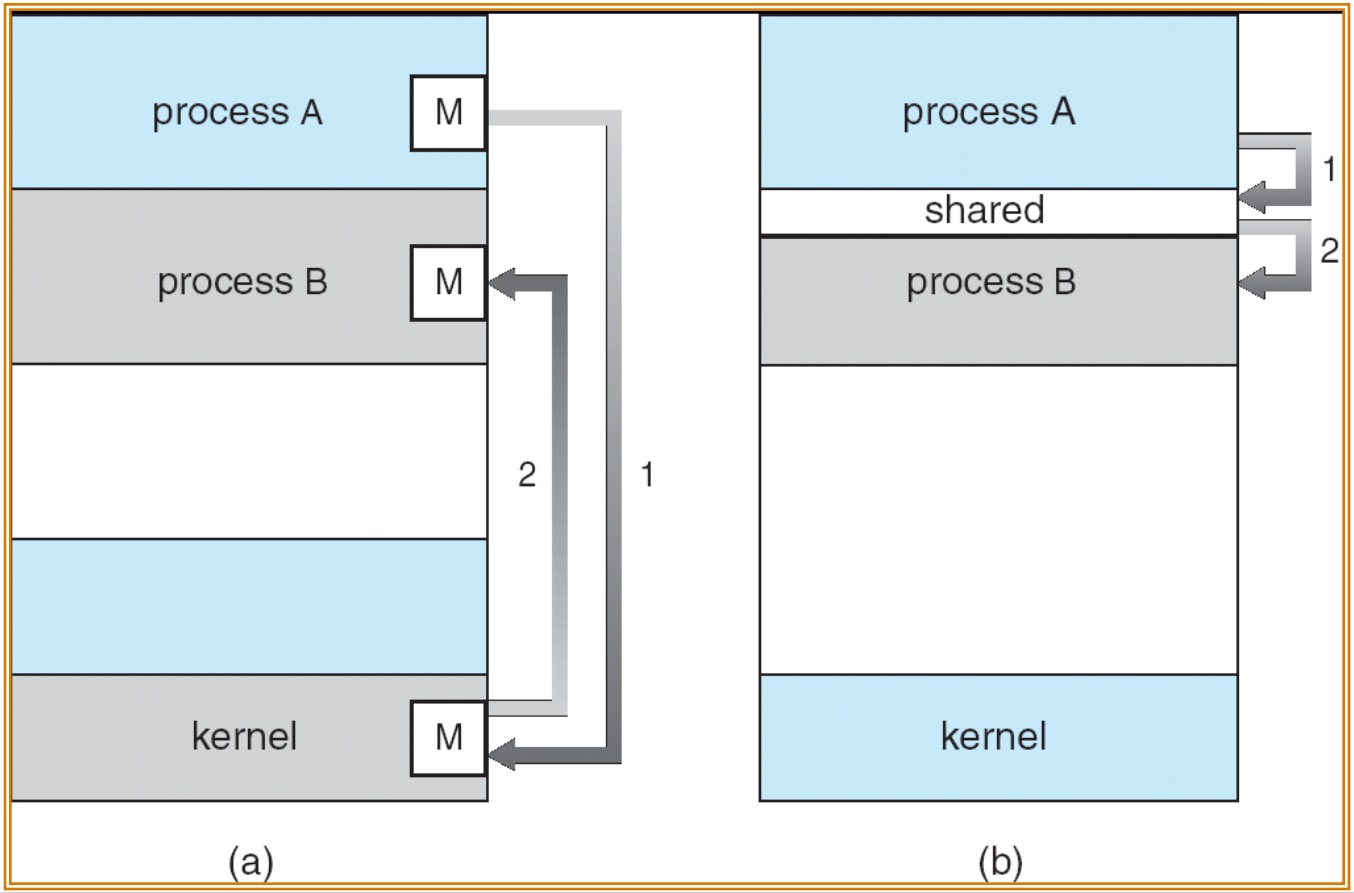
Comunicación Interprocesos (IPC)

- Mecanismo para comunicar procesos y sincronizar acciones
- Systema de mensajes – los procesos se comunican sin usar variables compartidas
- IPC proporciona dos operaciones:
  - **send**(*message*) – tamaño de mensaje fijo o variable
  - **receive**(*message*)
- Si *P* y *Q* necesitan comunicarse, necesitan:
  - Establecer un *canal de comunicación*
  - Intercambiar mensajes usando send/receive
- Implementación de un canal de comunicación
  - física (v.g., memoria compartida, bus del sistema)
  - lógica (v.g., propiedades lógicas)

Detalles de Implementación

- Como se establecen los canales de comunicación?
- Se puede asociar un canal con más de dos procesos?
- Cuántos canales pueden haber entre cada par de procesos?
- Cuál es la capacidad de un canal?
- Tamaño de mensaje fijo o variable?
- Canales uni/bi-direccionles?

Modelos de Comunicación



Comunicación Directa

- Los procesos se refieren de manera explícita:
  - **send** (*P, message*) – envía un mensaje al proceso P
  - **receive**(*Q, message*) – recibe un mensaje del proceso Q
- Propiedades de un canal de comunicación
  - Los canales se establecen automáticamente
  - Un canal se asocia con exactamente un par de procesos comunicantes
  - Entre cada par existe exactamente un canal
  - El canal puede ser unidireccional, pero usualmente es bi-direccional

Comunicación Indirecta

- Los mensajes se envían y son recibidos en buzones (puertos)
  - Cada buzón tiene un ID único
  - Los procesos se pueden comunicar solo si comparten un buzón
- Propiedades de un canal de comunicación
  - Se establece solo si dos procesos comparten un buzón
  - Un canal puede asociarse con muchos procesos
  - Cada par de procesos puede compartir varios canales de comunicación
  - El canal puede ser unidireccional o bi-direccional

Comunicación Indirecta

- Operaciones
  - Crear un buzón
  - Enviar y recibir mensajes por medio de un buzón
  - Destruir un buzón
- Primitivas:

  send(*A, message*) – enviar un mensaje al buzón A

  receive(*A, message*) – receibir un mensaje del buzón A

Comunicación Indirecta

- Buzones compartidos (Mailbox sharing)
  - P_1, P_2, y P_3 comparten el buzón A
  - P_1, envía; P_2 y P_3 reciben
  - ¿Quién recibe el mensaje?
- Soluciones
  - Canal asociado con dos procesos máximo
  - Solo un proceso puede ejecutar una operación *receive* en un instante dado
  - El sistema selecciona arbitrariamente al receptor. El remitente es notificado de quién recibió el mensaje

Sincronización

- Envío de mensajes con/sin bloqueo (blocking/non-blocking)
- **Blocking** es **síncrono**
  - **Blocking send** bloquea al remitente hasta que el mensaje es recibido
  - **Blocking receive** bloquea al receptor hasta que llega un mensaje
- **Non-blocking** es **asíncrono**
  - **Non-blocking** send – el remitente envía el mensaje y continúa
  - **Non-blocking** receive – el receptor recibe un mensaje válido o null

Buffering

- Se asocia una cola de mensajes al canal.
- Implementación de la cola
  1. 1. Capacidad cero – 0 mensajes
     El remitente espera al receptor (rendezvous)
  2. Capacidad limitada – longitud finita de *n* mensajes; el remitente debe esperar si el canal se llena
  3. Capacidad ilimitada – longitud infinita; el remitente nunca espera

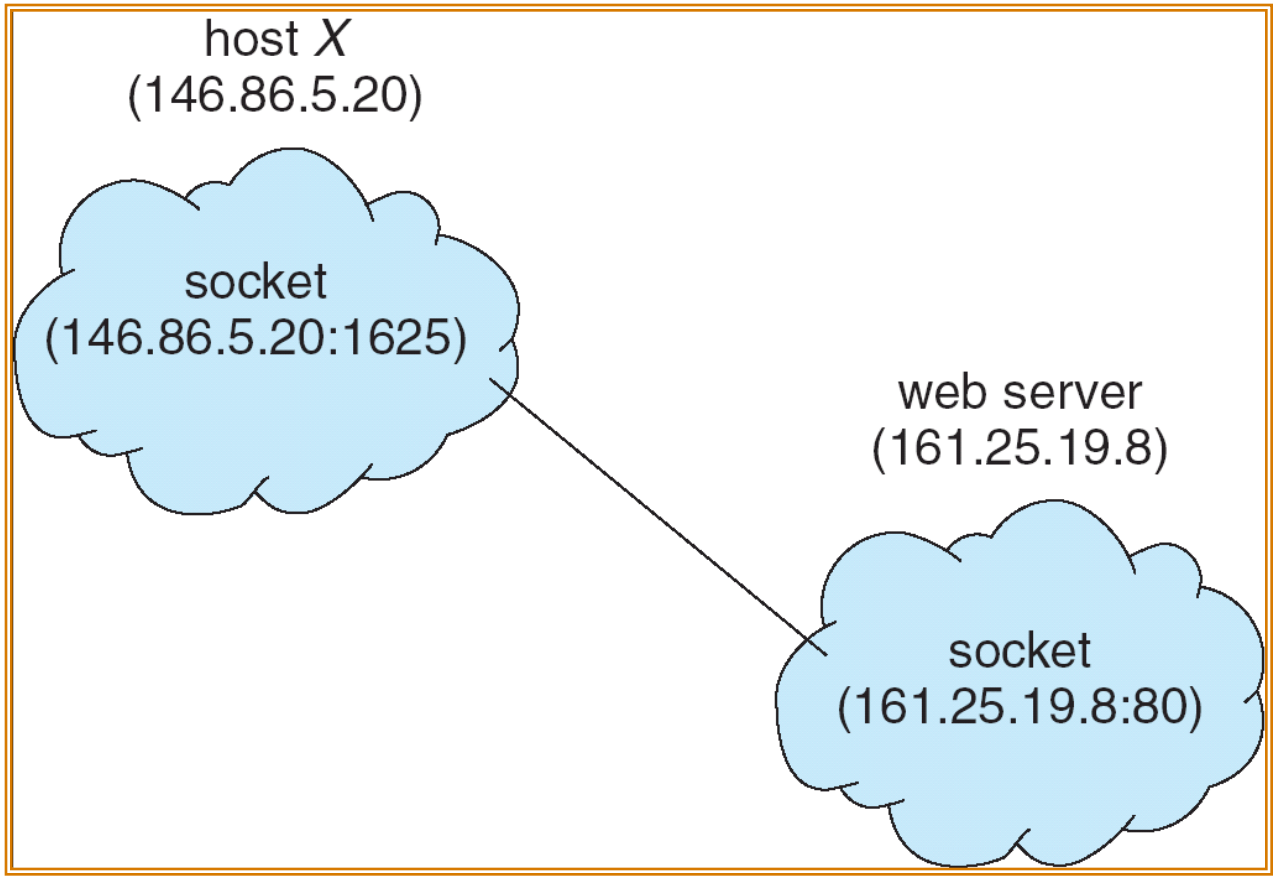
Comunicación Cliente-Servidor

- Sockets
- Remote Procedure Calls
- Remote Method Invocation (Java)

Sockets

- Socket: *punto terminal de comunicación*
- Concatenación de dirección IP y puerto
- El socket **161.25.19.8:1625** se refiere al puerto **1625** del host **161.25.19.8**
- Comunicación – pares de sockets

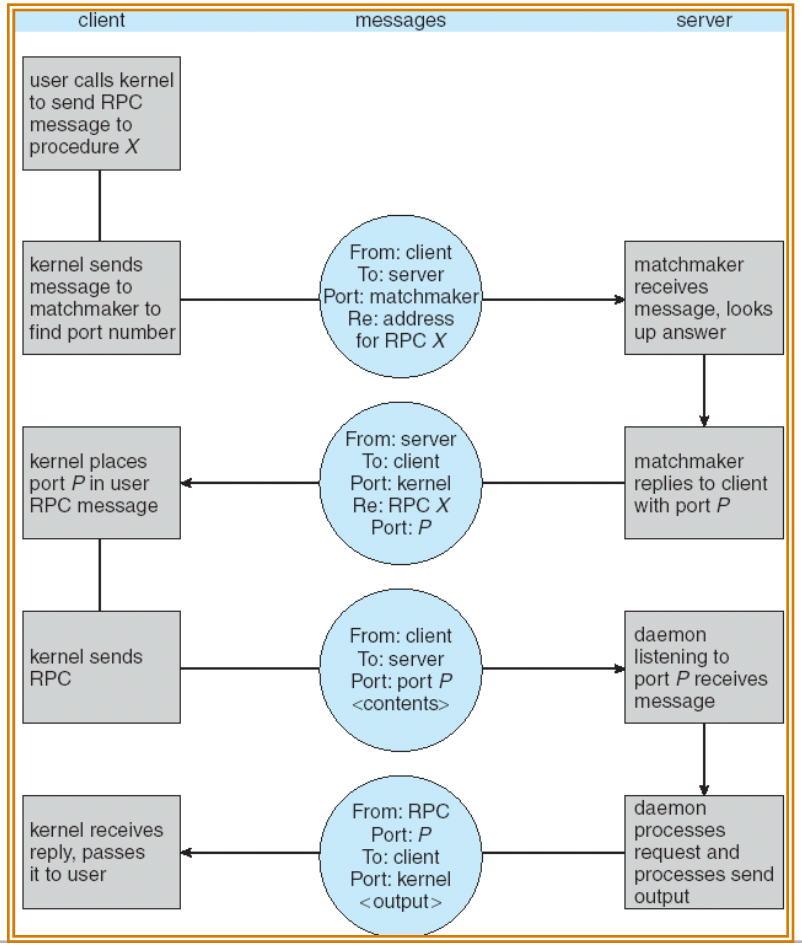
Communicación entre Sockets



Remote Procedure Calls

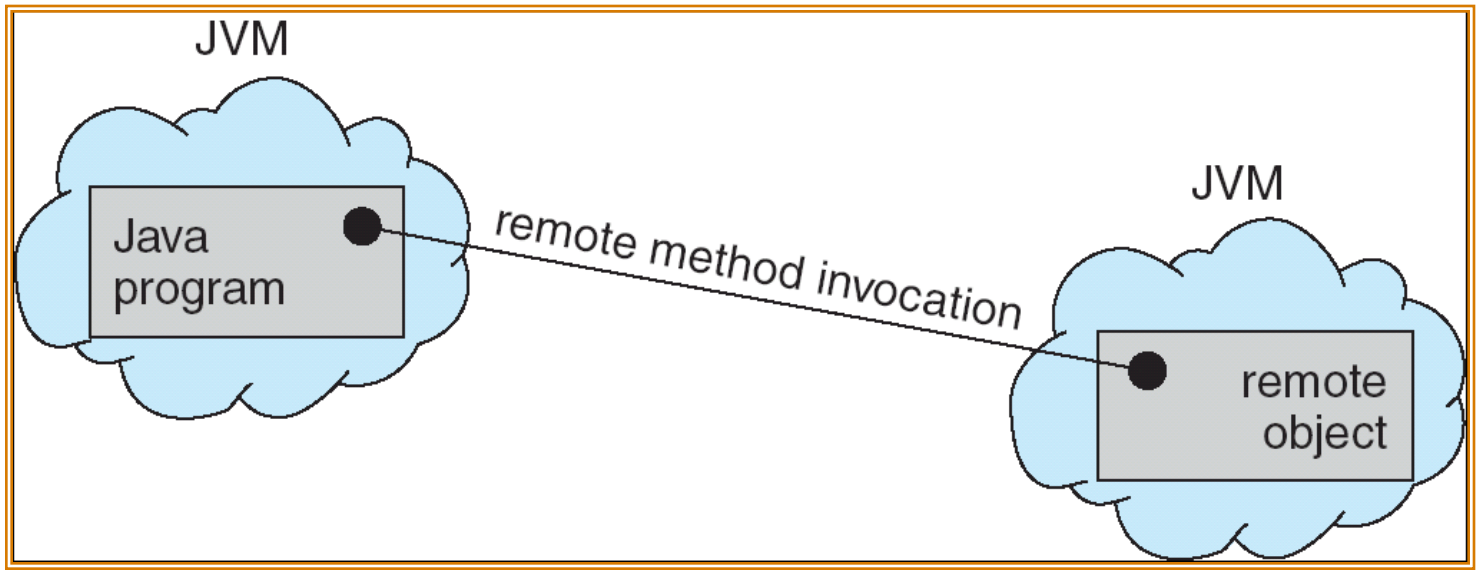
- Remote procedure call (RPC) abstrae las llamadas a procedimientos entre procesos en sistemas en red
- **Stubs** – proxy en el cliente para el procedimiento en el servidor
- El stub del cliente localiza al servidor y *envía* los parámetros
- El stub del servidor recibe el mensaje, desempaca los parámetros y ejecuta el procedimiento en el servidor

Ejecución de RPCs



Remote Method Invocation

- Remote Method Invocation (RMI) es un mecanismo de similar a los RPCs.



Envío de Parámetros