Sistemas Operativos

Operaciones sobre procesos

Departamento de Ingeniería en Sistemas y Computación
Universidad Católica del Norte, Antofagasta.

Operaciones sobre procesos

- El SO debe proveer un mecanismo para crear y terminar un proceso
- La mayoría de los SO permiten la ejecución (pseudo) concurrente y pueden ser creados y eliminados dinámicamente

Operaciones sobre procesos

- El SO debe proveer un mecanismo para crear y terminar un proceso
- La mayoría de los SO permiten la ejecución (pseudo) concurrente y pueden ser creados y eliminados dinámicamente

Creación de procesos

- Cada proceso tiene un identificador que se denomina **pid**
- El pid permite realizar diversas operaciones sobre procesos
- Un proceso padre puede crear procesos hijos, los cuales a su vez pueden crear otros procesos creando un árbol de procesos.

Creación de procesos

- Cada proceso tiene un identificador que se denomina **pid**
- El pid permite realizar diversas operaciones sobre procesos
- Un proceso padre puede crear procesos hijos, los cuales a su vez pueden crear otros procesos creando un árbol de procesos.

Creación de procesos

- Cada proceso tiene un identificador que se denomina **pid**
- El pid permite realizar diversas operaciones sobre procesos
- Un proceso padre puede crear procesos hijos, los cuales a su vez pueden crear otros procesos creando un árbol de procesos.

Creación de procesos

Existen tres opciones para la compartición de recursos:

- Padres e hijos comparten todos sus recursos
- Los hijos comparten un subconjunto de los recursos del padre
- Padres e hijos no comparten recursos

Creación de procesos

Existen tres opciones para la compartición de recursos:

- Padres e hijos comparten todos sus recursos
- Los hijos comparten un subconjunto de los recursos del padre
- Padres e hijos no comparten recursos

Creación de procesos

Existen tres opciones para la compartición de recursos:

- Padres e hijos comparten todos sus recursos
- Los hijos comparten un subconjunto de los recursos del padre
- Padres e hijos no comparten recursos

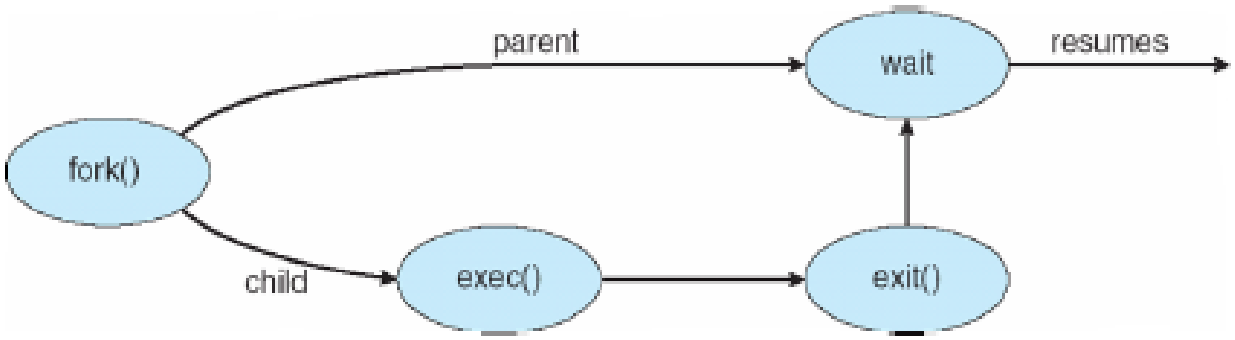
Ejecución de procesos

Dos alternativas para la ejecución:

- Padres e hijos se ejecutan concurrentemente
- El padre espera hasta que el hijo termine

Ejemplo UNIX

La llamada al sistema **fork** crea un nuevo proceso



Ejemplo en C

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
        printf("padre pid= %d \n", getpid()); // pid del padre
        fork();
        printf("pid= %d \n", getpid());
        fork();
        printf("pid= %d \n", getpid());
        fork();
        printf("pid= %d \n", getpid());
        fork();
        printf("pid= %d \n", getpid());
        return 0;
}
```

Terminación de procesos

- Un proceso ejecuta su última sentencia y pide al SO que lo elimine (**exit**)
- El hijo entrega datos de salida al padre (**wait**)
- Los recursos del proceso son liberados por el SO

Terminación de procesos

- Un proceso ejecuta su última sentencia y pide al SO que lo elimine (**exit**)
- El hijo entrega datos de salida al padre (**wait**)
- Los recursos del proceso son liberados por el SO

Terminación de procesos

- Un proceso ejecuta su última sentencia y pide al SO que lo elimine (**exit**)
- El hijo entrega datos de salida al padre (**wait**)
- Los recursos del proceso son liberados por el SO

Terminación de procesos

El padre puede terminar la ejecución de sus hijos (**abort**). La razón puede ser:

- El hijo ha excedido sus recursos asignados
- La tarea asignada al hijo ya no es necesaria
- Si el padre termina, algunos SO generan un término en cascada

Terminación de procesos

El padre puede terminar la ejecución de sus hijos (**abort**). La razón puede ser:

- El hijo ha excedido sus recursos asignados
- La tarea asignada al hijo ya no es necesaria
- Si el padre termina, algunos SO generan un término en cascada

Terminación de procesos

El padre puede terminar la ejecución de sus hijos (**abort**). La razón puede ser:

- El hijo ha excedido sus recursos asignados
- La tarea asignada al hijo ya no es necesaria
- Si el padre termina, algunos SO generan un término en cascada

Terminación de procesos

El padre puede terminar la ejecución de sus hijos (**abort**). La razón puede ser:

- El hijo ha excedido sus recursos asignados
- La tarea asignada al hijo ya no es necesaria
- Si el padre termina, algunos SO generan un término en cascada

Comunicación Inter-Proceso

- Los procesos en un SO pueden ser independientes o cooperativos
- Procesos cooperativos pueden ser afectados o afectar otros procesos incluidos los datos compartidos

Comunicación Inter-Proceso

- Los procesos en un SO pueden ser independientes o cooperativos
- Procesos cooperativos pueden ser afectados o afectar otros procesos incluidos los datos compartidos

Comunicación Inter-Proceso

Existen varias razones para que los procesos cooperen:

- Compartir información
- Modularidad
- Si el computador tiene más de un procesador, se logra mayor rapidez de ejecución

Comunicación Inter-Proceso

Existen varias razones para que los procesos cooperen:

- Compartir información
- Modularidad
- Si el computador tiene más de un procesador, se logra mayor rapidez de ejecución

Comunicación Inter-Proceso

Existen varias razones para que los procesos cooperen:

- Compartir información
- Modularidad
- Si el computador tiene más de un procesador, se logra mayor rapidez de ejecución

Comunicación Inter-Proceso

Hay dos modelos de IPC:

- Memoria compartida
- Paso de mensajes

Memoria compartida

- Se establece una región de memoria que se comparte por los procesos que cooperan entre sí
- Pueden intercambiar información escribiendo y leyendo
- Se requieren llamadas al sistema sólo para establecer las regiones compartidas de memoria

Memoria compartida

- Se establece una región de memoria que se comparte por los procesos que cooperan entre sí
- Pueden intercambiar información escribiendo y leyendo
- Se requieren llamadas al sistema sólo para establecer las regiones compartidas de memoria

Memoria compartida

- Se establece una región de memoria que se comparte por los procesos que cooperan entre sí
- Pueden intercambiar información escribiendo y leyendo
- Se requieren llamadas al sistema sólo para establecer las regiones compartidas de memoria

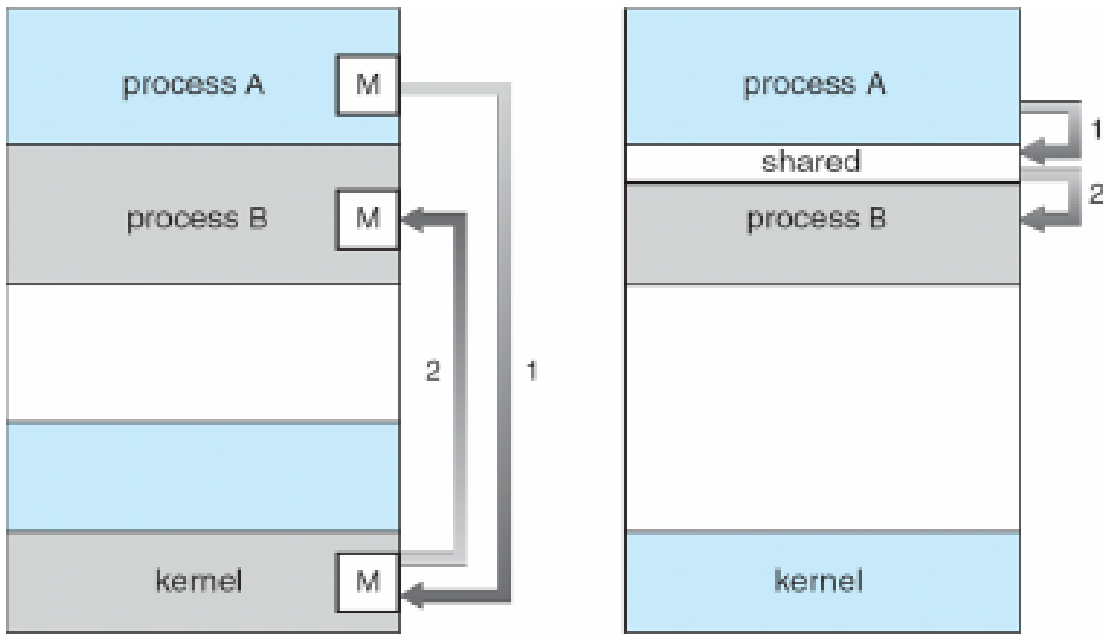
Paso de mensajes

- La comunicación se establece por medio de intercambios de mensajes
- Resulta útil para intercambiar cantidades pequeñas de datos

Paso de mensajes

- La comunicación se establece por medio de intercambios de mensajes
- Resulta útil para intercambiar cantidades pequeñas de datos

Modelos de IPC



Problema del Productor-Consumidor

- Ambos comparten un buffer de tamaño finito
- El productor no debe añadir más productos que la capacidad del buffer
- El consumidor no debe intentar tomar un producto si el buffer está vacío

Problema del Productor-Consumidor

- Ambos comparten un buffer de tamaño finito
- El productor no debe añadir más productos que la capacidad del buffer
- El consumidor no debe intentar tomar un producto si el buffer está vacío

Problema del Productor-Consumidor

- Ambos comparten un buffer de tamaño finito
- El productor no debe añadir más productos que la capacidad del buffer
- El consumidor no debe intentar tomar un producto si el buffer está vacío

Problema del Productor-Consumidor

Los datos compartidos son:

```
#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 + 1) % BUFFER SIZE )  == out)
      ;   /* do nothing -- no free buffers */
     buffer[in] = item;
     in = (in + 1) % BUFFER SIZE;
   }
```

Consumidor

```
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;
   }
```

Paso de mensajes

- El paso de mensajes es un mecanismo para comunicar y sincronizar las acciones de un proceso
- Los procesos se comunican sin necesidad de utilizar variables compartidas

Paso de mensajes

- El paso de mensajes es un mecanismo para comunicar y sincronizar las acciones de un proceso
- Los procesos se comunican sin necesidad de utilizar variables compartidas

Paso de mensajes

IPC provee de dos mecanismos:

- send(msg): msg puede tener tamaño fijo o variable
- receive(msg)

Modelos para send y receive

Los métodos para implementación lógica de operaciones *send()* y *receive()* permiten:

- Comunicación directa o indirecta
- Comunicación sincrónica o asincrónica

Modelos para send y receive

Los métodos para implementación lógica de operaciones *send()* y *receive()* permiten:

- Comunicación directa o indirecta
- Comunicación sincrónica o asincrónica

Comunicación directa

Los procesos deben nombrarse explícitamente:

- `send(P,msg)` : envía mensaje a P
- `receive(Q,msg)` : recibe un mensaje desde Q

Comunicación directa

Los procesos deben nombrarse explícitamente:

- `send(P,msg)` : envía mensaje a P
- `receive(Q,msg)` : recibe un mensaje desde Q

Comunicación directa

- Los enlaces se establecen automáticamente
- Un enlace se asocia con exactamente un par de procesos que se comunican
- Los enlaces pueden ser bidireccionales o unidireccionales

Comunicación directa

- Los enlaces se establecen automáticamente
- Un enlace se asocia con exactamente un par de procesos que se comunican
- Los enlaces pueden ser bidireccionales o unidireccionales

Comunicación directa

- Los enlaces se establecen automáticamente
- Un enlace se asocia con exactamente un par de procesos que se comunican
- Los enlaces pueden ser bidireccionales o unidireccionales

Comunicación indirecta

- Los mensajes son dirigidos y recibidos por *mailboxes*
- Cada mailbox tiene un **id** único
- Los procesos pueden comunicarse sólo si comparten un mailbox

Comunicación indirecta

Operaciones:

- Crear un mailbox
- Enviar y recibir mensajes a través de un mailbox
- Destruir mailbox

Comunicación indirecta

Operaciones:

- Crear un mailbox
- Enviar y recibir mensajes a través de un mailbox
- Destruir mailbox

Comunicación indirecta

Operaciones:

- Crear un mailbox
- Enviar y recibir mensajes a través de un mailbox
- Destruir mailbox

Comunicación indirecta

Operaciones primitivas:

- `send(A,msg)` : envía mensaje al mailbox A
- `receive(A,msg)` : recibe mensaje desde mailbox A

Comunicación indirecta

Operaciones primitivas:

- `send(A,msg)` : envía mensaje al mailbox A
- `receive(A,msg)` : recibe mensaje desde mailbox A

Sincronización

- El sistema de comunicación por mensajes puede ser sincrónico (bloqueante) o asincrónico (no bloqueante)
- El send bloqueante deja al proceso bloqueado hasta que el mensaje es recibido
- El receive bloqueante deja al receptor bloqueado hasta que el mensaje está disponible

Sincronización

- El sistema de comunicación por mensajes puede ser sincrónico (bloqueante) o asincrónico (no bloqueante)
- El send bloqueante deja al proceso bloqueado hasta que el mensaje es recibido
- El receive bloqueante deja al receptor bloqueado hasta que el mensaje está disponible

Sincronización

- El sistema de comunicación por mensajes puede ser sincrónico (bloqueante) o asincrónico (no bloqueante)
- El send bloqueante deja al proceso bloqueado hasta que el mensaje es recibido
- El receive bloqueante deja al receptor bloqueado hasta que el mensaje está disponible

Sincronización

- En el send no bloqueante, el proceso envía y continúa
- En el receive no bloqueante, el receptor recibe un mensaje válido o nulo

Ejemplo

Se tienen 3 procesos cuyo código es (para $i = 1, 2, 3$):

```
Pi()
{
printf("Soy el proceso %d\n", i);
}
```

Que funciones deberíamos insertar para sincronizar utilizando mensajes directos y sincrónicos, generando la salida en el orden Proceso 1, Proceso 2 y Proceso 3 ?

Ejemplo

```
P1()
{
printf("Soy %d\n", 1);
send(P2, --);
}
```

```
P2
{
receive(P1, --);
printf("Soy %d\n", 2);
send(P3, --);
}
```

```
P3()
{
receive(P2, --);
printf("Soy %d\n", 3);
}
```