3.4Comunicación interprocesos

Un proceso es cooperativo si puede afectar o verse afectado por los demás procesos que se ejecutan en el sistema.

Un proceso es independiente si no puede afectar o verse afectado por los restantes procesos que se ejecutan en el sistema

Compartir información: proporcionar un entorno que permita el acceso concurrente a dicha información.

Acelerar los cálculos: Ejecutar una tarea rápidamente se debe dividir en subtareas, ejecutándose cada una en paralelo.

Modularidad: Podemos querer construir el sistema de forma modular, dividiendo las funciones del sistema en diferentes procesos o hebras

Conveniencia: Incluso un solo usuario puede querer trabajar en muchas tareas al mismo tiempo.

3.4.1 Sistemas de memoria compartida

La comunicación interprocesos requiere que los procesos que se estén comunicando establezcan una región de memoria compartida.

Los procesos también son responsables de verificar que no escriben en la misma posición simultáneamente.

Un proceso productor genera información que consume un proceso consumidor

3.4.2 Sistemas de paso de mensajes

proporciona un mecanismo que permite a los procesos comunicarse y sincronizar sus acciones sin compartir el mismo espacio de direcciones.

Es útil en un entorno distribuido, en el que dichos procesos pueden residir en diferentes computadoras conectadas en red.

3.4.2.1 Nombrado

Los procesos comunicados deben disponer de un modo de referenciarse entre sí. Pueden usar comunicación directa o indirecta.

3.4.2.2 Sincronización

La comunicación entre procesos tiene lugar a través de llamadas a las primitivas send( ) y receive( ) .

Envío con bloqueo: El proceso se bloquea hasta que el proceso receptor o el buzón de correo reciben el mensaje.

Envío sin bloqueo: El proceso transmisor envía el mensaje y continúa operando.

Recepción con bloqueo: El receptor se bloquea hasta que hay un mensaje disponible.

Recepción sin bloqueo: El receptor extrae un mensaje válido o un mensaje nulo.

3.4.2.3 Almacenamiento en búfer

Sea la comunicación directa o indirecta, los mensajes intercambiados por los procesos que se están comunicando residen en una cola temporal.

Capacidad cero: La cola tiene una longitud máxima de cero; por tanto, no puede haber ningún mensaje esperando en el enlace.

Capacidad limitada: La cola tiene una longitud finita n; por tanto, puede haber en ella n mensajes como máximo.

Capacidad ilimitada: La longitud de la cola es potencialmente infinita; por tanto, puede haber cualquier cantidad de mensajes esperando en ella.

3.5 Ejemplos de sistemas IPC

3.5.1 Un ejemplo: memoria compartida en POSIX

El primer parámetro especifica la clave (o identificador) del segmento de memoria compartida.

El segundo parámetro especifica el tamaño (en bytes) del segmento.

El tercer parámetro identifica el modo, que indica cómo se va a usar el segmento de memoria compartida: para leer, para escribir o para ambas operaciones.

3.5.2 Un ejemplo: Mach

Permite la creación y destrucción de múltiples tareas, que son similares a los procesos, pero tienen múltiples hebras de control

La mayor parte de las comunicaciones en Mach, incluyendo la mayoría de las llamadas al sistema y toda la comunicación Inter-tareas, se realiza mediante mensajes.

Los mensajes se envían y se reciben mediante buzones de correo, que en Mach se denominan puertos.

3.5.3 Un ejemplo: Windows XP

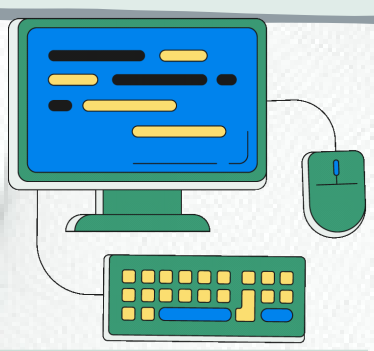
Proporciona soporte para varios entornos operativos, o subsistemas, con los que los programas de aplicación se comunican usando un mecanismo de paso de mensajes.

Los programas de aplicación se pueden considerar clientes del servidor de subsistemas de Windows XP.
Su facilidad de paso de mensajes se denomina llamada a procedimiento local (LPC, local procedure call).

Comunicación interprocesos

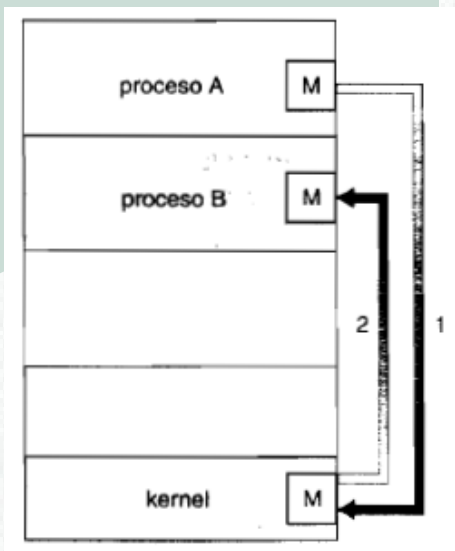
Los procesos que se ejecutan concurrentemente pueden ser procesos independientes o procesos cooperativos.
Un proceso es independiente si no puede afectar o verse afectado por los restantes procesos que se ejecutan en el sistema.
Un proceso es cooperativo si puede afectar o verse afectado por los demás procesos que se ejecutan en el sistema.

Hay varias razones para proporcionar un entorno que permita la cooperación entre procesos:
-> **Compartir información:** Dado que varios usuarios pueden estar interesados en la misma información.
-> **Acelerar los cálculos:** Si deseamos que una determinada tarea se ejecute rápidamente, debemos dividirla en subtareas, ejecutándose cada una de ellas en paralelo con las demás.
-> **Modularidad:** Podemos querer construir el sistema de forma modular, dividiendo las funciones del sistema en diferentes procesos o hebras.
-> **Conveniencia:** Incluso un solo usuario puede querer trabajar en muchas tareas al mismo tiempo.



La cooperación entre procesos requiere mecanismos de comunicación interprocesos(IPC, Interprocess Communication) que les permitan intercambiar datos e información. Existen dos modelos fundamentales de comunicación interprocesos:

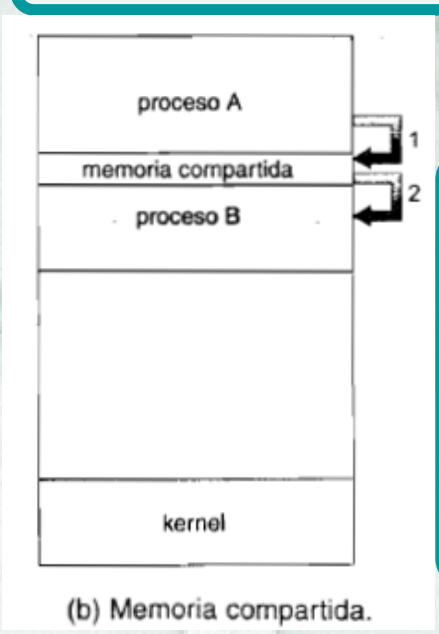
(1) memoria compartida
(2)paso de mensajes



(a) Paso de mensajes.

Sistemas de paso de mensajes

El paso de mensajes proporciona un mecanismo que permite a los procesos comunicarse y sincronizar sus acciones sin compartir el mismo espacio de direcciones, y es especialmente útil en un entorno distribuido, en el que los procesos que se comunican pueden residir en direntes computadoras conectadas en red.



(b) Memoria compartida.

Sistemas de memoria compartida

La comunicación interprocesos que emplea memoria compartida requiere que los procesos que se estén comunicando establezcan una región de memoria compartida. Normalmente, una región de memoria compartida reside en el espacio de direcciones del proceso que crea el segmento de memoria compartida. La memoria compartida requiere que dos o más procesos acuerden eliminar esta restricción.

Ejemplos de sistemas IPC

Un ejemplo: memoria compartida en POSIX

Para los sistemas POSIX hay disponibles varios mecanismos IPC, incluyendo los de memoria compartida y de paso de mensajes.
En primer lugar, un proceso tiene que crear un segmento de memoria comapartida usando la llamada al sistema *shmget().shmget()* se deriva de Shared Memory GET(obtención de datos a través de memoria compartida). El siguiente ejemplo ilustra el uso de *shmget()*.

segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);

Una llamada a *shmget()* que se ejecute con éxito devolverá un identificador entero para el segmento.
Los procesos que deseen acceder a un segmento de memoria compartida deben asociarlo a su espacio de direcciones usando la llamada al sistema *shmat()* [Shared Memory ATach].Si se ejecuta correctamente, *shmat()* devuelve un puntero a la posición inicial de memoria a la que se ha asociado la región de memoria compartida.
Una vez que la región de memoria compartida se ha asociado al espacio de direcciones de un proceso, éste puede acceder a la memoria compartida como en un acceso de memoria normal, usando el puntero devuelto por *shmat()*.
Para desconectar una región de memoria compartida, el proceso puede pasar el puntero de la región de memoria compartida a la llamada al sistema *shmdt()*.
Por ultimo un segmento de memoria compartida puede eliminarse del sistema mediante la llamada al sistema *shmctl()*, a la cual se pasa el identificador del segmento compartido junto con el indicador IPC_RMID.

POSIX stands for Portable Operating System Interface based on UNIX

Un ejemplo: Mach

El sistema operativo Mach, desarrollado en la Universidad Carnegie Mellon. La mayor parte de las comunicaciones en Mach, incluyen la mayoría de las llamadas al sistema y toda la comunicación inter-tareas, se realiza mediante mensajes. Los mensajes se envían y se reciben mediante buzones de correo, que en Mach se denominan puertos. Sólo son necesarias tres llamadas al sistema para la transferencia de mensajes, La llamada *msg_send()* envia un mensaje a un buzón de correo. Un mensaje se recibe mediante *msg_receive()*. Finalmente, las llamadas a procedimientos remotos (RPC) se ejecutan mediante *msg_rcp()*, que envía un mensaje y espera a recibir como contestación exactamente un mensaje. De esta forma, las llamadas RPC modelan una llamada típica a procedimiento, pero pueden trabajar entre sistemas distintos(de ahí el calificativo de remoto).

Un ejemplo: Windows XP

Windows XP proporciona soporte para varios entornos operativos, o subsistemas con los que los programas de aplicación se comunicacn usando un mecanismo de paso de mensajes. La facilidad de paso de mensajes en Windows XP se denomina llamada a procedimiento local (LPC, Local Procedure Call). Como Mach, Windows XP usa un objeto puerto para establecer y mantener una conexión entre dos procesos. Cada cliente que llama a un subsistencia necesita un canal de comunicación, que se proporciona mediante un objeto puerto y que nunca se hereda. La comunicación funciona del modo siguiente:
->El cliente abre un descriptor del objeto puerto de conexión del subsistema
->El cliente envía una solicitud de conexión
->El servidor crea dos puertos de comunicación privados y devuelve el descriptor de uno de ellos al cliente
->El cliente y el servidor usan el descriptor del puerto correspondiente para enviar mensajes o realizar retro llamadas y esperar las respuestas.