**Introducción**

La memoria compartida es la memoria física que se asigna a la agrupación de memoria compartida y se comparte entre varias particiones lógicas. El término agrupación de memoria compartida es una colección definida de bloques de memoria física que el hipervisor gestiona como una agrupación de memoria individual. Las particiones lógicas que configura para que utilicen memoria compartida comparten la memoria en la agrupación con otras particiones de memoria compartida.

**Memoria compartida**

Las utilidades de memoria compartida permiten crear segmentos de memoria a los que pueden acceder múltiples procesos, pudiendo definirse restricciones de acceso (sólo lectura).

Para trabajar con un segmento de memoria compartida, es necesario crear un vínculo (attachment) entre la memoria local del proceso interesado y el segmento compartido. Esto se realiza con la función shmat. El proceso que vincula un segmento de memoria compartida cree estar trabajando con ella como si fuera cierta área de memoria local. Para deshacer el vínculo está la función shmdt.

**Obtener un segmento de memoria**

shmget - obtiene un segmento de memoria compartida.

**Sintaxis**

#include <sys/shm.h>

int shmget( key\_t key, size\_t size, int shmflg);

**Descripción**

La función shmget retorna el identificador de memoria compartida asociada a key. Un identificador de memoria compartida y la estructura de datos asociada se crearán para key si una de las siguientes condiciones se cumple:

Si key es igual a IPC\_PRIVATE. Cuando esto ocurre se creará un nuevo identificador, si existen disponibilidades, este identificador no será devuelto por posteriores invocaciones a shmget mientras no se libere mediante la función shmctl. El identificador creado podrá ser utilizado por el proceso invocador y sus descendientes; sin embargo, esto no es un requerimiento. El segmento podrá ser accedido por cualquier proceso que posea los permisos adecuados.

Si key aún no tiene asociado un identificador de memoria compartida y además shmflg & IPC\_CREAT es verdadero.

Como consecuencia de la creación, la estructura de datos asociada al nuevo identificador se inicializa de la siguientes manera: shm\_perm.cuid y shm\_perm.uid al identificador de usuario efectivo del proceso invocador, shm\_perm.gcuid y shm\_perm.guid al identificador de grupo efectivo del proceso invocador, los 9 bits menos significativos de shm\_perm.mode se inicializan a los 9 bits menos significativos del parámetro shmflg, shm\_segsz al valor especificado por size, shm\_ctime a la fecha y hora que el sistema poseía en el momento de la invocación y por último se ponen a cero shm\_lpid, shm\_nattach, shm\_atime y shm\_dtime.

Si la ejecución se realiza con éxito, entonces retornará un valor no negativo denominado identificador de segmento compartido. En caso contrario, retornará -1 y la variable global errno tomará en código del error producido.

**Vincular un área de memoria compartida**

shmat, shmdt - funciones de operación sobre memoria compartida.

**Sintaxis**

#include <sys/shm.h>

char \*shmat( int shmid, void \*shmaddr, int shmflg );

int shmdt( void \*shmaddr )

**Descripción**

shmat asocia el segmento de memoria compartida especificado por shmid al segmento de datos del proceso invocador. Si el segmento de memoria compartida aún no había sido asociado al proceso invocador, entonces shmaddr debe tener el valor de cero y el segmento se asocia a una posición en memoria seleccionado por el sistema operativo. Dicha localización será la misma en todos los procesos que acceden al objeto de memoria compartida. Si el segmento de memoria compartida ya había sido asociado por el proceso invocador, shmaddr podrá tener un valor distinto de cero, en ese caso deberá tomar la dirección asociada actual del segmento referenciado por shmid. Un segmento se asocia en modo sólo lectura si shmflg & SHM\_RDONLY es verdadero; si no entonces se podrá acceder en modo lectura y escritura.. No es posible la asociación en modo sólo escritura. Si la función se ejecuta con éxito, entonces retornará la dirección de comienzo del segmento compartido, si ocurre un error devolverá -1 y la variable global errno tomará el código del error producido.

shmdt desasocia del segmento de datos del proceso invocador el segmento de memoria compartida ubicado en la localización de memoria especificada por shmaddr. Si la función se ejecuta sin error, entonces devolverá 0, en caso contrario retornará -1 y errno tomará el código del error producido.

**Operaciones de control**

shmctl - Realiza operaciones de control en una región de memoria compartida.

**Sintaxis**

#include <sys/shm.h>

int smctl(int shmid, int cmd, struct shmid\_ds \*buff);

**Descripción**

La función shmctl permite realizar un conjunto de operaciones de control sobre una zona de memoria compartida identificada por shmid. El argumento cmd se usa para codificar la operación solicitada. Los valores admisibles para este parámetro son:

IPC\_STAT: lee la estructura de control asociada a shmid y la deposita en la estructura apuntada por buff.

IPC\_SET: actualiza los campos shm\_perm.uid, shm\_perm.gid y shm\_perm.mode de la estructura de control asociada a shmid tomando los valores de la estructura apuntada por buff.

IPC\_RMID: elimina el identificador de memoria compartida especificado por shmid del sistema, destruyendo el segmento de memoria compartida y las estructuras de control asociadas. Si el segmento está siendo utilizado por más de un proceso, entonces la clave asociada toma el valor IPC\_PRIVATE y el segmento de memoria es eliminado, el segmento desaparecerá cuando el último proceso que lo utiliza notifique su desconexión del segmento. Esta operación sólo la podrán utilizar aquellos procesos que posean privilegios de acceso a recurso apropiados, para llevar a cabo esta comprobación el sistema considerará el identificador de usuario efectivo del proceso y lo comparará con los campos shm\_perm\_id y shm\_perm\_cuid de la estructura de control asociada a la región de memoria compartida.

SHM\_LOCK: bloquea la zona de memoria compartida especificada por shmid. Este comando sólo puede ejecutado por procesos con privilegios de acceso apropiados.

SHM\_UNLOCK: desbloquea la región de memoria compartida especificada por shmid. Esta operación, al igual que la anterior, sólo la podrán ejecutar aquellos procesos con privilegios de acceso apropiados.

La función shmctl retornará el valor 0 si se ejecuta con éxito, o -1 si se produce un error, tomando además la variable global errno el valor del código del error producido.

**Sockets**

Definición

Los sockets son mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando en distintas máquinas.

Una forma de conseguir que dos programas se transmitan datos. Un socket no es más que un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador. Desde el punto de vista de programación, un socket no es más que un "fichero" que se abre de una manera especial.

Una vez abierto se pueden escribir y leer datos de él con las funciones de read() y write(). La forma de referenciar un socket por los procesos implicados, es mediante un descriptor del mismo tipo que el utilizado para referenciar ficheros. Se podrá realizar redirecciones de los archivos de E/S estándar (descriptores 0,1 y 2) a los sockets y así combinar entre ellos aplicaciones de la red.

**Propiedades**

Fiabilidad de la Transmisión. No se pierden los datos transmitidos.

Conservación del Orden de los Datos. Los datos llegan en el orden en que se emitieron.

No Duplicación de los Datos. El Dato sólo llega una vez.

Comunicación en modo conectado. La conexión está establecida antes de iniciar la comunicación. De este modo, la emisión desde un extremo va destinada al otro (implícitamente).

Conservación de los límites de los mensajes. Los límites de mensajes emitidos pueden encontrarse o conocerse en el destino.

Envío de Mensajes “urgentes”. Permite el envío de datos fuera de flujo o fuera de banda. Al enviar datos fuera del flujo normal, son accesibles de inmediato.

**Atributos**

Dominio

Especifica el medio de comunicación de la red que el socket utilizará.

AF\_UNIX: Sockets internos de UNIX (Sockets del sistema de archivos).

AF\_INET: Protocolos de internet ARPA (Sockets de redes de UNIX).

AF\_ISO: Protocolos estándar ISO.

AF\_NS: Protocolos de redes Xerox

**Protocolo**

Especifica que protocolo se va a usar.

Se usa donde el mecanismo de transporte permite más de un protocolo a ser usado.

En las redes de UNIX y en los sockets de sistema de archivos no necesitamos seleccionar otro protocolo diferente al default.

Tipo

Los protocolos de internet proveen dos niveles distintos de servicio: flujo y datagramas

SOCK\_STREAM: Para flujo

* Son implementados en el dominio AF\_INET por conexiones TCP/IP.
* Son el tipo usual en el dominio AF\_UNIX.

SOCK\_DGRAM : Para datagramas

* No establecen ni mantienen una conexión.
* También existe un límite en el tamaño del datagrama que se puede enviar.
* Se transmite como un solo mensaje en la red que se puede perder, duplicar o llegar fuera de secuencia.

Tipos de Sockets

Los no orientados a conexión

* El programa de aplicación da la fiabilidad

Los orientados a conexión.

* Comunicaciones fiables
* Circuito Virtual

Principales características de los Sockets no orientados a conexión

* Es el llamado protocolo UDP.
* No es necesario que los programas se conecten.
* Cualquiera de ellos puede transmitir datos en cualquier momento, independientemente de que el otro programa esté "escuchando" o no.
* Garantiza que los datos que lleguen son correctos, pero no garantiza que lleguen todos.
* Se utiliza cuando es muy importante que el programa no se quede bloqueado.
* No importa que se pierdan datos.

Principales características de los Sockets a conexión

* Primero hay que establecer correctamente la conexión.
* Ninguno de los dos puede transmitir datos.
* Se usa el protocolo TCP del protocolo TCP/IP, para gestionar la conexión.
* Se garantiza que todos los datos van a llegar de un programa al otro correctamente.
* Se utiliza cuando la información a transmitir es importante, no se puede perder ningún dato.
* No importa que los programas se queden "bloqueados" esperando o transmitiendo datos.

Arquitectura

Cliente – Servidor

* Servidor: es el programa que permanece pasivo a la espera de que alguien solicite conexión con él.
* Cliente: es el programa que solicita la conexión para pedir datos al servidor

Servidor:

* Está ejecutándose y esperando a que otro quiera conectarse a él.
* Nunca da "el primer paso" en la conexión.
* Es el que "sirve" información al que se la pida.

Cliente:

* Es el programa que da el “primer paso” en la conexión.
* En el momento de ejecutarlo o cuando lo necesite, intenta conectarse al servidor.
* Es el que solicita información al servidor.

Para poder realizar la conexión entre ambos programas (cliente y servidor) es necesario conocer la dirección IP del servidor y el servicio que queremos crear / utilizar.

Funciones del servidor

int socket ( int dominio, int tipo, int protocolo)

Crea un socket sin nombre de un dominio, tipo y protocolo específico

* dominio: AF\_INET, AF\_UNIX
* tipo: SOCK\_\_DGRAM, SOCK\_\_STREAM
* protocolo: 0 (protocolo por defecto)

int bind ( int dfServer, struct sockaddr\* direccServer, int longDirecc)

Nombra un socket: asocia el socket no nombrado de descriptor dfServer con la dirección del socket

* almacenado en direccServer.
* La dirección depende de si estamos en un dominio AF\_UNIX o AF\_INET.

int listen (int dfServer, int longCola)

Especifica el máximo número de peticiones de conexión pendientes.

int accept (int dfServer, struct sockaddr\* direccCliente, int\* longDireccCli)

Escucha al socket nombrado“servidor dfServer” y esperahasta que se reciba la petición de la conexión de un cliente.

Al ocurrir esta incidencia, crea un socket sin nombre con lasmismas características que el socket servidor original, lo conecta al socket cliente y devuelve un descriptor de fichero que puede ser utilizado para la comunicación con el cliente.

Funciones del cliente

int socket (int dominio, int tipo, int protocolo)

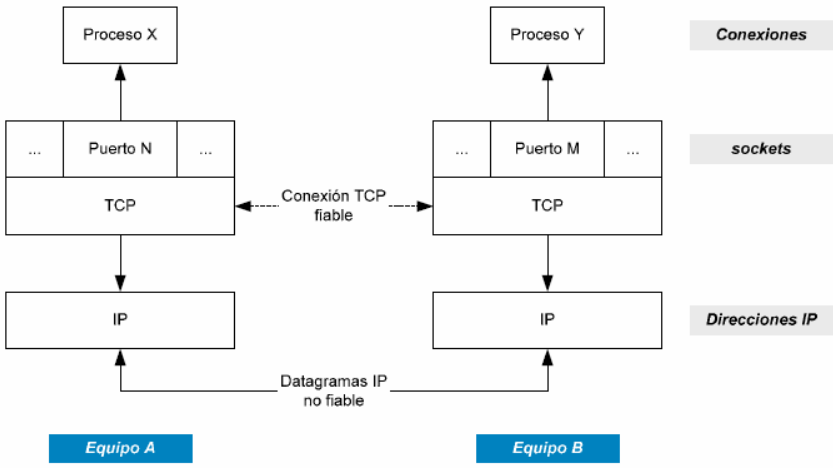
Crea un socket sin nombre de un dominio, tipo y protocolo específico

* dominio: AF\_INET, AF\_UNIX
* tipo: SOCK\_\_DGRAM, SOCK\_\_STREAM
* protocolo: 0 ( protocolo por defecto )

int connect (int dfCliente, struct sockaddr\* direccServer, int longDirecc)

Intenta conectar con un socket servidor cuya dirección se encuentra incluida en la estructura apuntada por direccServer.

* El descriptor dfCliente se utilizará para comunicar con el socket servidor.
* El tipo de estructura dependerá del dominio en que nos encontremos.



SEÑALES Y EXEPCIONES

Cuando un Sistema Operativo desea notificar a un proceso la ocurrencia de un determinado evento o error, recurre a 2 tipos de mecanismos: SEÑALES y EXEPCIONES, la primera se utiliza en POSIX (por ser familia UNIX) y la segunda en WINDOWS NT.

SEÑALES

Las señales tienen frente al proceso el mismo comportamiento que las interrupciones tienen frente al procesador, por lo que se puede decir que una señal es una interrupción al proceso; también se puede definir como una notificación por software a un proceso/thread de la ocurrencia de un evento que Permite asociar señales independientes a cada proceso o thread. Por ejemplo, una interrupción notificación por hardware, una señal es gestionada por el Sistema Operativo, interrupción de una tarea actualmente en curso y activación de la tarea asociada a la señal. Las señales son un mecanismo propio de los Sistemas UNIX.

Tipos de Señales: Dado que las señales se utilizan para indicarle al proceso muchas cosas diferentes, existe una gran variedad de ellas.

Excepciones hardware.

Comunicación.

E/S asíncrona.

ESTADOS DE UNA SEÑAL

Generada: Una señal está asociada a un evento, por lo que cuando dicho evento se produce se dice que la señal se ha generado.

Depositada: Se dice que la señal está depositada cuando el proceso asociado emprende una acción con base a ella.

Tiempo de vida: el tiempo de vida de una señal es el intervalo entre la generación y el depósito de ésta.

Pendiente: se dice que una señal está pendiente si ha sido generada pero todavía no está depositada.

Atrapada: un proceso atrapa una señal si éste ejecuta el manejador de señal cuando se deposita.

Ignorada: no es ejecutado ningún manejador al ser depositada.

Bloqueada: Si se genera una señal y está bloqueada no se pierde, queda pendiente de ser depositada hasta que sea desbloqueda. Se controla mediante la máscara de la señal.

COMPORTAMIENTO DEL PROCESO QUE RECIBE UNA SEÑAL

El proceso detiene su ejecución en la instrucción de máquina que está ejecutando.

Bifurca a ejecutar una rutina de tratamiento de la señal, cuyo código ha de formar parte del propio proceso.

Una vez ejecutada la rutina de tratamiento, sigue la ejecución del proceso en la instrucción en el que fue interrumpido.

EL ORIGEN DE UNA SEÑAL PUEDE SER UN PROCESO O EL SISTEMA OPERATIVO. SEÑAL PROCESO à PROCESO

Un proceso puede enviar una señal a otro proceso que tenga el mismo identificador de usuario (uid), pero no a los que lo tengas distinto, un proceso del superusuario puede mandar una señal a cualquier proceso, con independencia de su uid. Un proceso también puede mandar una señal a un grupo de procesos, que han de tener su mismo uid.

SEÑAL SISTEMA OPERATIVO à PROCESO

El sistema operativo también toma la decisión de enviar señales a los procesos cuando ocurren determinadas condiciones. Por ejemplo, las excepciones de ejecución programa (el desbordamiento en las operaciones aritméticas, la división por cero, el intento de ejecutar una instrucción con código de operación incorrecto o de direccionar una posición de memoria prohibida) las convierte el sistema operativo en señales al proceso que ha causado la excepción (evento que ocurre durante la ejecución de un programa y que requiere la ejecución de un fragmento de código situado fuera del flujo normal de ejecución).

IDENTIFICACIÓN DE UNA SEÑAL Algunas etiquetas definidas en <signal.h>

SIGHUP Hangup Colgar SIGKILL Eliminate the process Elimina el proceso SIGALRM Alarm clock Despertador

SIGTERM software termination signal from kill Señal de terminación de software de matanza

SIGUSR1 User defined signal 1 Definida señal 1 SIGUSR2 User defined signal 2 Definida señal 2

SIGRTMIN first (highest-priority) realtime signal Primero la señal de mayor prioridad

SIGRTMAX last (lowest-priority) realtime signal De ultimo la señal de menor prioridad

¿QUÉ GENERA UNA SEÑAL?

Interrupciones hardware:

Activación de pines (puertos) especiales de la CPU. Excepciones en la ejecución. Relojes.

¿QUIÉN RECIBE UNA SEÑAL?

Las señales van dirigidas a un proceso, pero pueden existir varios threads dentro del proceso:

Señales síncronas: son aquellas que son generadas por la ejecución del código, y por tanto son generadas por un thread concreto. En este caso la señal puede ser depositada únicamente por el thread que generó el evento.

Señales asíncronas: el resto de señales producidas por llamas explícitas (kill) o por eventos no asociados al código en ejecución. En este caso la señal va dirigida a todos los threads del proceso.

Una señal solo puede ser depositada por un thread y la recibirá uno cualquiera que cumpla una de las siguientes condiciones:

El thread está suspendido esperando la señal. Si existen varios habrá una cola de prioridades.

El thread no incluye en su máscara de bloqueo la señal en cuestión

EFECTO DE LA SEÑAL Y ARMADO DE LA MISMA

El efecto de la señal es ejecutar una rutina de tratamiento. Para que esto sea asi, el proceso debe tener armada ese tipo de señal, es decir, ha de estar preparado para recibir dicho tipo de señal.

Armar una señal significa indicar al sistema operativo el nombre de la rutina del proceso que ha de tratar ese tipo de señal, lo que se consigue en POSIX con el servicio sigaction. Algunas señales admiten que se las ignoren, lo cual ha de ser indicado por el proceso al sistema operativo. En este caso, el sistema operativo simplemente desecha las señales ignoradas por ese proceso. Un proceso también puede enmascarar diversos tipos de señales. El efecto es que las señales enmascaradas quedan bloqueadas (no se desechan), a las espera de que el proceso las desenmascare. Entonces, se debe asignar un Manejador.

Asignación de un Manejador

El manejador es una función que es ejecutada cuando se deposita una señal (si no es ignorada). Se gestiona mediante la función sigaction(), que permite:

Ignorar la señal: no se ejecutaría ningún manejador (a pesar de ello la señal si que llegaría al proceso que podría ejecutar una acción).

Asignar un manejador por defecto.

Asignar nuestro propio manejador.

IMPLEMENTACION DE SEÑALES (sigaction)

Con la llamada sigaction asociamos señales con acciones para indicar cómo responderá el proceso actual cuando reciba una señal. La declaración de sigactiones:

Donde sig es la señal cuya acción queremos programar, act es la nueva acción que se le asocia y oact es la que tenía asociada. Si la llamada se ejecuta satisfactoriamente devuelve 0, en caso contrario devuelve -1, en errno se almacena el código del error producido y no se le asocia ninguna acción a la señal.

PARA CONFIGURAR UNA SEÑAL DEBE ESTAR BLOQUEADA.

Para señales asíncronas van a existir dos aspectos diferentes de configuración y gestión de señales:

Aquellos aspectos que afectan únicamente al thread que las llama: irán ubicadas generalmente la función asociada al thread

Aquellos aspectos comunes a todos los threads: se ubican generalmente en el thread principal (main).

La configuración de una señal tiene dos partes:

Establecer el bloqueo de la señal (la señal queda pendiente de ser depositada). Es un aspecto de configuración independiente para cada thread (se hereda en la creación del thread)

Decidir entre atrapar la señal (asignar un manejador para la señal => función especial que se ejecuta cuando es depositada), o bien ignorar la señal. Se trata de un aspecto común para todos los threads

BLOQUEO DE UNA SEÑAL

Supone que dicha señal quede pendiente de depósito en el caso de que sea generada

Es específico de cada thread del proceso (se hereda en la creación del thread)

os threads que no hayan bloqueado la señal son los únicos que pueden depositar la señal.

La gestión del bloqueo de señales se realiza mediante lo que se denomina máscara de señal:

Se maneja mediante una estructura de tipo sigset\_t almacenada por el S.O. para cada thread.

La máscara de señal indica la configuración de cada señal (bloqueada o no).

Adicionalmente podemos definir nuevos conjuntos de señales de tipo sigset\_t que nos permitirán activar o desactivar grupos de señales en cada momento.

Importante: Cuando un proceso recibe una señal sin haberla armado o enmascarado previamente, se ejecuta la acción por defecto, que en la mayoría de los casos consiste en matar al proceso con el servicio kill, ya que es el más utilizado para matar procesos.

MECANISMOS DE COMUNICACIÓN Y SINCRONIZACION DE PROCESOS

SINCRONIZACIÓN MEDIANTE SEÑALES

Las señales pueden utilizarse para sincronizar procesos. Si se utilizan Señales Posix un proceso puede bloquearse en el servicio pause esperando la recepción de una señal. Esta señal puede ser enviada por otro proceso mediante el servicio kill. Con este mecanismo se consigue que unos procesos esperen a que se cumpla una determinada condición y que otros los despierten cuando se cumple la condición que les permite continuar su ejecución. El empleo de señales, sin embargo, no es un mecanismo muy apropiado para sincronizar procesos debido a las siguientes razones:

Las señales tienen un comportamiento asíncrono. Un proceso puede recibir una señal en cualquier punto de su ejecución, aunque no esté esperando su recepción.

Las señales no se encolan. Si hay una señal pendiente de entrega a un proceso y se recibe una señal del mismo tipo, la primera se perderá. Sólo se entregará la última. Esto hace que se puedan perder eventos de sincronización importantes.