



Memoria compartida

POSIX



Memoria compartida

Memoria Compartida POSIX

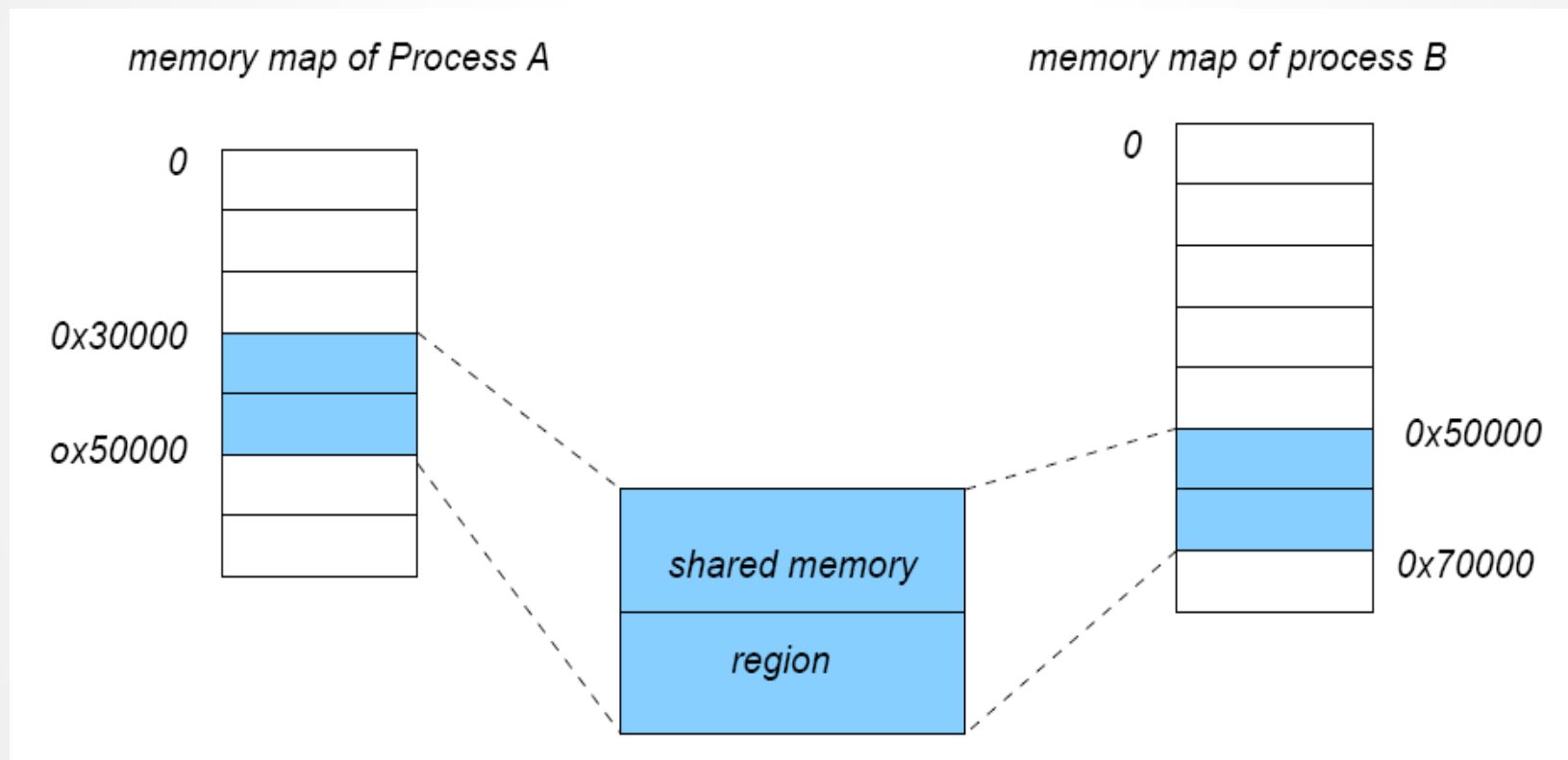
En sistemas Unix es posible hacer que dos procesos distintos sean capaces de compartir una zona de memoria común y, de esta manera, compartir o comunicar datos.

- Permite compartir una región de memoria sin necesidad de crear un archivo.
- Linux utiliza un sistema de archivo dedicado montado en el directorio `/dev/shm`.
- El sistema de archivo tiene persistencia de Kernel, es decir que los objetos de memoria compartida que contiene, permanecerán mientras el proceso los tenga abiertos, pero se perderán al reiniciar el sistema.



Memoria compartida

Una región de memoria compartida es una porción física de memoria, que es compartida por múltiples procesos





Memoria compartida

Memoria Compartida POSIX

Para utilizar un objeto de memoria compartida POSIX, llevamos a cabo dos pasos:

1. Abrir (o crear y abrir) un objeto con nombre específico con la función `shm_open()`. La función, `shm_open()` devuelve un descriptor de archivo hace referencia al objeto.
2. Pasar el descriptor obtenido, en una llamada a `mmap()` con la bandera `MAP_SHARED` en el argumento de banderas. Esto mapea el objeto de memoria compartida en el espacio virtual de direcciones del proceso.



Memoria compartida

Creación de objetos de memoria compartida

shm_open: crea y abre (o solo abre si ya existe) un nuevo objeto POSIX de memoria compartida

```
#include <fcntl.h>           // define constantes O_  
#include <sys/stat.h>        // definición de contantes mode  
#include <sys/mman.h>
```

```
int shm_open(const char *name, int oflag, mode_t mode);  
int shm_open (nombre, flags, modo)
```

Devuelve descriptor de archivo en caso de éxito, o -1 si hay error



Memoria compartida

El argumento ***name*** identifica el objeto de memoria compartida que se crea o se abre.

El argumento ***Oflag*** es una máscara de bits que modifican el comportamiento de la llamada.

Modo se hace por medio de una máscara de permisos de modo, como en los permisos de archivos. Si no se crea el objeto el modo debe ir en cero.

Flag	Descripción
O_CREAT	Crea un objeto si no existe
O_EXCL con O_CREAT	Crea un objeto exclusivamente, si existe falla
O_RDONLY	Abre para lectura solamente
O_RDWR	Abre para lectura y escritura
O_TRUNC	Truncar objeto a la longitud cero



Memoria compartida

Eliminación de objetos de memoria compartida

```
#include <sys/mman.h>  
int shm_unlink(const char *name);
```

Devuelve 0 en caso de éxito, o -1 en caso de error.

Elimina un nombre de objeto de memoria compartida, y una vez que todos los procesos han eliminado el objeto, libera y destruye el contenido de la zona de memoria asociada



Memoria compartida

Función ftruncate()

La llamada a sistema ftruncate() trunca a un fichero a una longitud específica.

```
#include <unistd.h>
int ftruncate(int fd, off_t length);
```

Devuelve 0 si tuvo éxito, o -1 en caso de error

Fd: descriptor de fichero

Length: tamaño de archivo

Si el fichero era mayor que la nueva longitud, los bytes extras se pierden y si la nueva longitud del fichero es mayor se rellena con bytes 0.

El puntero del fichero no se modifica. Con ftruncate el archivo debe estar abierto como escritura.



Memoria compartida

Función mmap()

La función mmap ubica una cantidad de bytes(length) comenzando en el desplazamiento (offset) desde el fichero (u otro objeto especificado por el descriptor de fichero fd) en memoria, en la dirección addr.

```
#include <sys/mman.h>
```

```
void *addr= mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);  
mmap(start, longitud,Prot,flags, fd, offset);
```

Devuelve un puntero al área reservada o MAP_FAILED (-1) en caso de error.

El argumento addr indica la dirección virtual en el que el mapeo se va a ubicar. Si especificamos start como NULL, el kernel la dirección de la asignación.



Memoria compartida

El argumento `prot` describe la protección de memoria deseada, y no debe entrar en conflicto con el modo de apertura del fichero.

Valor	PROT	Descripción
	<code>PROT_NONE</code>	Las páginas no pueden ser accedidas
	<code>PROT_READ</code>	El contenido de las páginas pueden ser leído
	<code>PROT_WRITE</code>	El contenido de las páginas pueden ser escrito
	<code>PROT_EXEC</code>	Las páginas deben ser ejecutadas

FLAGS

`MAP_PRIVATE` Crear un área privada "copy-on-write"

`MAP_SHARED` Comparte este área con todos los otros objetos que señalan a este objeto



Memoria compartida

Función memcpy()

Copia áreas de memoria

#include <string.h>

void *memcpy(void *destino, const void *origen, size_t longitud);

Destino: es el lugar donde queremos copiar

Origen: es de donde copiamos

Longitud: Es la cantidad de bytes a copiar

Devuelve un puntero a destino

Las áreas de memoria destino y origen no deben tener ningún punto de intersección, no deben solaparse.