Universidad Nacional de Rio Cuarto
Fac. Cs. Exa. Fco-Qcas y Nat.
Departamento de Computación
Sistemas Operativos

Informe Trabajo Final

"Shared Memory"

Estudiantes:

Cornejo, César
Politano, Mariano
Raverta, Fernando

- Año 2015 -

DESCRIPCIÓN DE LA SOLUCIÓN IMPLEMENTADA

Para brindar al Sistema Operativo Xv6 la capacidad de compartir bloques de memoria entre varios procesos, pensamos en disponer de las siguientes estructuras:
Un arreglo de bloques del Sistema Operativo (shmtable) de tamaño MAXSHMBLOCK que contenga la siguiente información de cada uno de los bloques de memoria compartida: Cantidad de procesos que lo referencian, cantidad de páginas físicas que contiene el bloque y direcciones de las páginas físicas de RAM (en realidad direcciones virtuales del Kernel) a las que hace referencia.
Por otro lado, cada proceso debe conocer que bloques de memoria compartidas ha adquirido y por tanto incorporamos esta información al struct proc, mas adelante detallaremos qué información debe mantener el proceso.
El módulo sharedmem.c posee la implementación de las llamadas al sistema que permiten al usuario hacer uso de bloques de memoria compartida. Este módulo implementa las siguientes abstracciones, provistas al proceso usuario como llamadas al sistema:

int shm_create(int size): permite crear un bloque de memoria compartido, retornando el id del bloque creado. Internamente, este proceso reserva un slot en la struct shmtable y aloca las páginas físicas necesarias para cubrir el tamaño expresado en bytes por el argumento size; pero no hace el mapeo de direcciones virtuales del proceso con las direcciones físicas adquiridas.

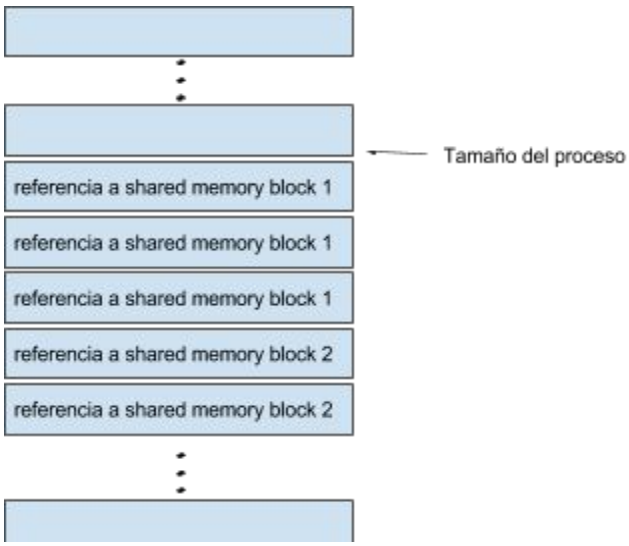
int shm_get(int key, void **addr): escribe en *addr la dirección de memoria virtual asociada al bloque compartido con id = key. Internamente, es este método es quien eventualmente mappea las direcciones virtuales del proceso con las físicas adquiridas por la ejecución de shm_create.

int shm_close(int key): dado un id de un bloque compartido (key), desmapea el bloque de memoria compartida del proceso que la ejecuta, y además, si es el único que referencia al bloque compartido, libera el mismo.

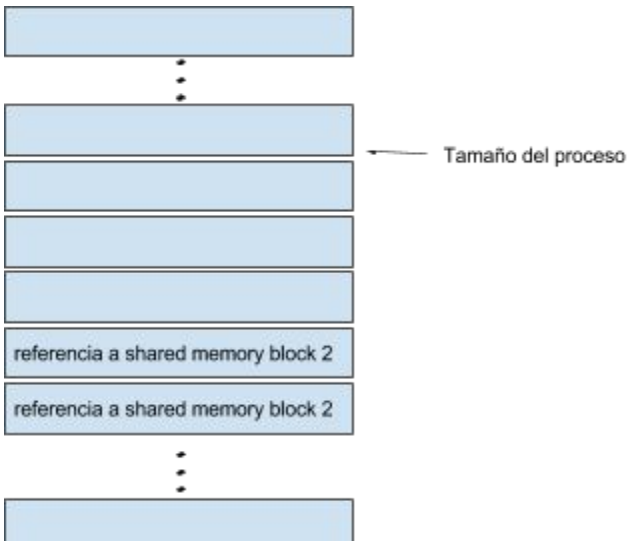
Notar que se hizo una descripción no rigurosa, particularmente debemos mencionar que es necesario realizar varios chequeos sobre los argumentos de cada uno de dichos métodos y también es importante informar al usuario a través de diferentes valores de retorno los errores producidos en la invocación de la llamada al sistema. Esto último está detallado en el código.

Otro aspecto importante de la implementación es la desición que hemos tomado a la hora de realizar el mapping de direcciones virtuales del proceso con direcciones físicas. En este sentido hemos decidido que cuando un proceso obtiene un bloque de memoria compartida(shm_get), las direcciones físicas sean mapeadas en direcciones virtuales a partir de la primera página siguiente a la última dirección del proceso donde además ocurra que

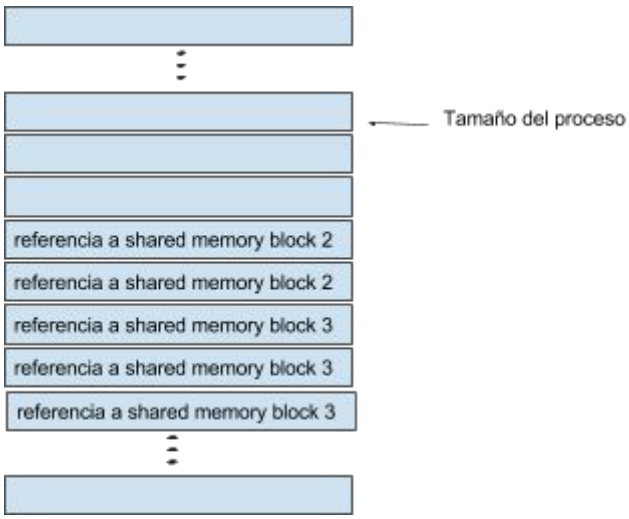
existe un rango de direcciones virtuales consecutivas que alcancen para cubrir el direccionamiento de todo el bloque. Para aclarar lo mencionado, supongamos que tenemos un proceso A que hace 2 shm_get de dos bloques diferentes creados por otro proceso donde el primero tiene identificador 1 y ocupa 3 páginas de RAM y el segundo tiene identificador 2 y ocupa 2 páginas. El escenario puede verse en la siguiente imagen:



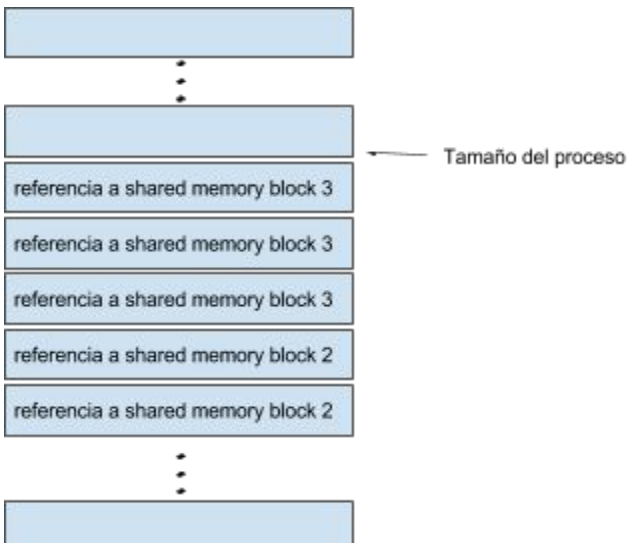
Es importante mencionar, que esta descripción es meramente descriptiva ya que no se visualiza la estructura real del mapa de direcciones virtuales.
Ahora supongamos que el proceso hace un shm_close pasando como argumento el descriptor 1, el nuevo escenario puede ser visto como:



Ahora supongamos que el proceso hace un create y un get de un bloque de 4 páginas cuyo identificador es 3, estas serían mapeado después del mapping al bloque de memoria compartida 2, veamos:



Pero, si hubiese echo create y get de mapping de un bloque de hasta 3 páginas este sería mapeado arriba del mapping del bloque 2 cómo se puede ver en la siguiente imagen:



Visto este detalle de implementación, es necesario almacenar en el struct de cada proceso información de cuál es la dirección virtual de comienzo para cada bloque de memoria compartida obtenido.

A modo de cierre, el sistema Xv6 puede soportar hasta MAXSHM cantidad de bloques de memoria compartida, mientras que cada proceso puede tener hasta MAXSHMPROC bloques de memoria compartida. Cada bloque puede tener a lo sumo MAXSHMPBLOCK páginas de memorias.

PRODUCTOR-CONSUMIDOR

Se implementó una versioń del problema del productor-consumidor en espacio de usuario. Para ello se desarrollaron 3 programas diferentes:

shmmaincp (amount_of_pages)

Este programa es el encargado de preparar un escenario para la interacción entre los productores y consumidores (procesos que instancia shmproducer y shmconsumer respectivamente), el cual crea un bloque de memoria compartida con tantas páginas como indique el argumento amount_of_pages e informa por salida standard los identificadores del bloque de memoria compartida y de los semáforos full empty y shmblock_sem. Donde full es el semáforo que controla la producción de elementos, empty el consumo y shmblock_sem el acceso al bloque de memoria compartida.

shmproducer (shmblock_descriptor, full, empty, shmblock_sem, value)

Programa que representa un productor el cual escribe elementos en el bloque de memoria compartida cuyo descriptor es shmblock_descriptor. Para su correcto funcionamiento debe ser llamado luego de la ejecución de shmmaincp, usando la información provista por ese programa. En ese sentido sus argumentos son shmblock_descriptor que es el id del bloque de memoria compartida, full, empty y shmblock_sem son los descriptores de semáforos creados por el maincp y por último, value es el carácter que el productor escribe en el bloque de memoria compartida.

shmconsumer(shmblock_descriptor, full, empty, shmblock_sem)

Programa que representa un consumidor el cual lee un caracter del bloque de memoria compartido cuyo descriptor es shmblock_descriptor. Para su correcto funcionamiento debe ser llamado luego de la ejecución de shmmaincp, usando la información provista por ese programa. En ese sentido sus argumentos son shmblock_descriptor que es el id del bloque de memoria compartida, full, empty y shmblock_sem son los descriptores de semáforos creados por el maincp.

Ejemplo de Uso:

Ejecutamos: **shmmaincp 3&**
Esto construye un escenario para productores-consumidores y supongamos que nos da la siguiente información:

id shared memory block = 0, full = 0, empty = 1, shmblock_sem=2

Con esta información podemos crear productores y consumidores que escriban y lean sincronizadamente en el bloque de memoria creado.
Por ejemplo para agregar un productor ejecutamos: **shmproducer 0 0 1 2 a&**

el cual en el bloque de memoria compartido con id = 0, escribe el caracter a, tantas veces como indique la constante del programa shmproducer element_that_produce.

Para agregar un consumidor ejecutamos: **shmconsumer 0 0 1 2** el cual lee tantas veces como indique la constante del programa shmconsumer element_that_consume, los caracteres escritos por el productor anterior.

Puede crear tantos consumidor y productores como le permita la configuración de Xv6.