Shared memory

Shared memory

Multiple processes can access (read/write) to a common memory which is backed by a swap space using shared memory.

There are two variants of shared memory implementations

1. The old SVR4 shmget(), shmat(), shmdt(), shmctl():
2. The new POSIX shm_open(), shm_unlink().

Advantages

- Fastest method of inter process communication

Disadvantages

- Changes do not persist in-case of system crash.

Shared memory helps multiple process to communicate with each other using common memory. This memory is backed by swap space. The changes done to the shared memory does not persist across reboot.

shm_open

```
#include <sys/mman.h>
#include <sys/stat.h> /* For mode constants */
#include <fcntl.h> /* For O_* constants */

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

Link with `-lrt`

Parameter	Direction	Description
name	in	*name* specifies the shared memory object to be created or opened. Eg */somename*
oflag	in	*oflag* is a bit mask created by ORing together exactly one of **O_RDONLY** or **O_RDWR** with one or more of O_CREAT, O_EXCL, O_TRUNC
mode	in	ORing of **S_IRWXU, S_IRWXG, S_IRWXO,** etc listed in ***open****(2)*

Return Value:
`shm_open()` returns a nonnegative file descriptor. On failure, `shm_open()` returns -1.

Description:

shm_open() creates and opens a new, or opens an existing, POSIX shared memory object. A POSIX shared memory object is in effect a handle which can be used by unrelated processes to ***mmap****(2)* the same region of shared memory.

shm_open

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

int shm_unlink(const char *name);
```

Link with `-lrt`

Parameter	Direction	Description
name	in	*name* specifies the shared memory object created by shm_open(3). Eg */somename*

Return Value:
`shm_unlink()` returns 0 on success, or -1 on error.

Description:
The **shm_unlink**() function performs the converse operation, removing an object previously created by **shm_open**().

Hands on lab:

Open a terminal and write the below program in mmap_shared.c using your preferred editor.

```
$ vi shm.c
#include <sys/mman.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

#define SHMSZ     4096

int main()
{
  char ch;
  int i, fd;
  char *shm;

  if ((fd = shm_open("/myshm", O_CREAT|O_RDWR, 0600)) < 0) {
    ...
  }

  if (ftruncate(fd, SHMSZ) != 0) {
    ...
  }

  shm = mmap(NULL, SHMSZ, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
  if (shm == MAP_FAILED) {
    ...
  }

  for (i=0; i<SHMSZ; i++) {
     shm[i] = 65 + i%26;
  }

  munmap(shm, SHMSZ);

  close(fd);

  return 0;
}
```

Compile and run the program :

```
$ gcc -o shm -lrt shm.c
$ ./shm
```

Memory mapped region in the virtual address space

In the second terminal, run cat /proc/<pid>/maps command. Where <pid> is the above process id.

```
$ cat /proc/`pidof shm`/maps

00400000-00401000 r-xp 00000000 08:08 537602894 /home/maruthisi/shm
00600000-00601000 r--p 00000000 08:08 537602894 /home/maruthisi/shm
00601000-00602000 rw-p 00001000 08:08 537602894 /home/maruthisi/shm
00956000-00977000 rw-p 00000000 00:00 0         [heap]
7fe7ebfb2000-7fe7ebfcb000 r-xp 00000000 08:06 101499267 /usr/lib64/libpthread-
2.25.so
7fe7ebfcb000-7fe7ec1cb000 ---p 00019000 08:06 101499267 /usr/lib64/libpthread-
2.25.so
7fe7ec1cb000-7fe7ec1cc000 r--p 00019000 08:06 101499267 /usr/lib64/libpthread-
2.25.so
7fe7ec1cc000-7fe7ec1cd000 rw-p 0001a000 08:06 101499267 /usr/lib64/libpthread-
2.25.so
7fe7ec1cd000-7fe7ec1d1000 rw-p 00000000 00:00 0
7fe7ec1d1000-7fe7ec398000 r-xp 00000000 08:06 101130799 /usr/lib64/libc-2.25.so
7fe7ec398000-7fe7ec598000 ---p 001c7000 08:06 101130799 /usr/lib64/libc-2.25.so
7fe7ec598000-7fe7ec59c000 r--p 001c7000 08:06 101130799 /usr/lib64/libc-2.25.so
7fe7ec59c000-7fe7ec59e000 rw-p 001cb000 08:06 101130799 /usr/lib64/libc-2.25.so
7fe7ec59e000-7fe7ec5a2000 rw-p 00000000 00:00 0
7fe7ec5a2000-7fe7ec5a9000 r-xp 00000000 08:06 101795188 /usr/lib64/librt-2.25.so
7fe7ec5a9000-7fe7ec7a8000 ---p 00007000 08:06 101795188 /usr/lib64/librt-2.25.so
7fe7ec7a8000-7fe7ec7a9000 r--p 00006000 08:06 101795188 /usr/lib64/librt-2.25.so
7fe7ec7a9000-7fe7ec7aa000 rw-p 00007000 08:06 101795188 /usr/lib64/librt-2.25.so
7fe7ec7aa000-7fe7ec7d1000 r-xp 00000000 08:06 101130590 /usr/lib64/ld-2.25.so
7fe7ec9b0000-7fe7ec9b2000 rw-p 00000000 00:00 0
7fe7ec9cd000-7fe7ec9ce000 rw-s 00000000 00:14 67391     /dev/shm/myshm
7fe7ec9ce000-7fe7ec9d0000 rw-p 00000000 00:00 0
7fe7ec9d0000-7fe7ec9d1000 r--p 00026000 08:06 101130590 /usr/lib64/ld-2.25.so
7fe7ec9d1000-7fe7ec9d3000 rw-p 00027000 08:06 101130590 /usr/lib64/ld-2.25.so
7ffc4fa9c000-7ffc4fabd000 rw-p 00000000 00:00 0         [stack]
7ffc4fbea000-7ffc4fbec000 r--p 00000000 00:00 0         [vvar]
7ffc4fbec000-7ffc4fbee000 r-xp 00000000 00:00 0         [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

In the above output, notice the `rw-s`. The `rw` are due to *prot*. `s` is due to flags. The `67391` is inode number of the /myshm file on /dev/shm pseudo file-system.

Exercise:

1. Write a program which implements producer consumer problem with bounded buffer using POSIX shared APIs.