Operating System Lab

Exp 8 - part 1 : Shared Memory

1. Objectives:

- Be familiar with the concept of shared memory.
- Learn the implementation of shared memory using bounded buffer shared memory.

2. Introduction:

1- A general scheme of using shared memory between producer and consumer processes:

For a producer, it should be started before any consumer. The producer should perform the following tasks:

Ask for a shared memory with a memory key and memorize the returned shared memory ID.

This is performed by system call **shmget( ).**

Attach this shared memory to the producer's address space with system call **shmat().**

Initialize the shared memory, if necessary.

Do something and wait for all consumer's completion.

Detach the shared memory with system call **shmdt( ).**

Remove the shared memory with system call **shmctl( ).**

For the consumer part, the procedure is almost the same:

Ask for a shared memory with the same memory key and memorize the returned shared memory ID.

Attach this shared memory to the consumer's address space.

Use the memory.

Detach all shared memory segments, if necessary.

Exit.

2- Asking for a Shared Memory Segment - shmget( ) :

The system call that requests a shared memory segment is shmget( ). It is defined as follows:

```
shm_id = shmget (
key_t k, /* the key for the segment */ int size, /* the size of the
segment */ int flag
); /* create/use flag */
```

In the above definition, **k** is of type **key_t** or **int**. It is the numeric key to be assigned to the returned shared memory segment. **size** is the size of the requested shared memory. The purpose of flag is to specify the way that the shared memory will be used. For our purpose, only the following two values are important:
IPC_CREAT | 0666 for a producer (i.e., creating and granting read and write access to the producer) 0666 for any consumer (i.e., granting read and write access to the consumer)

If **shmget( )** can successfully get the requested shared memory, its function value is a non-negative integer, the shared memory ID; otherwise, the function value is negative.

3- Attaching a Shared Memory Segment - shmat( ) :
Suppose process 1, a producer, uses **shmget( )** to request a shared memory segment successfully. That shared memory segment exists somewhere in the memory, but is not yet part of the address space of process 1. To make a requested shared memory segment part of the address space of a process, use **shmat( ).**
After a shared memory ID is returned, the next step is to attach it to the address space of a process. This is done with system call **shmat().** The use of **shmat( )** is as follows:

```
shm_ptr = shmat (
int shm_id, /* shared memory ID */
char *ptr, /* a character pointer */
int flag
); /* access flag */
```

System call **shmat( )** accepts a shared memory ID, **shm_id**, and attaches the indicated shared memory to the program's address space. The returned value is a pointer of type **(void *)** to the attached shared memory. Thus, casting is usually necessary. If this call is unsuccessful, the return value is -1. Normally, the second parameter is NULL. If the flag **is SHM_RDONLY**, this shared memory is attached as a read-only memory; otherwise, it is readable and writable.

4- Detaching and Removing a Shared Memory Segment - shmdt( ) and shmctl( ):
System call **shmdt( )** is used to detach a shared memory. After a shared memory is detached, it cannot be used. However, it is still there and can be re-attached back to a process's address space, perhaps at a different address. The only argument to **shmdt( )** is the shared memory address returned by **shmat( )**. Thus, the following code detaches the shared memory from a program:

```
shmdt (shm_ptr);
```

where **shm_ptr** is the pointer to the shared memory. This pointer is returned by **shmat()** when the shared memory is attached. If the detach operation fails, the returned function value is non-zero. To remove a shared memory segment, use the following code:

```
shmctl (shm_id, IPC_RMID, NULL);
```

Where **shm_id** is the shared memory ID. **IPC_RMID** indicates this is a remove operation. Note that after the removal of a shared memory segment, if you want to use it again, you should use **shmget( )** followed by **shmat( ).**

5- Example

```
 //The producer process
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
void main()
{
      key_t key= 1234;
      int shm_id;
      char *shm;
      /* Get shared memory ID */
      shm_id = shmget(key,10*sizeof(char),0644|IPC_CREAT);
      /* Attach to shared memory */
      shm = (char*)shmat(shm_id,(void*)0,0);
      strcpy(shm,"Hello");
}
```

```
 //Consumer process
 #include <sys/ipc.h>
 #include <sys/shm.h>
 #include <stdio.h>
 void main() {
       key_t key=1234;
       int shm_id;
       char *shm;
       char buffer[10];
       /* Get shared memory ID */
       shm_id = shmget(key,10*sizeof(char),0);
       /* Attach to shared memory */
       shm = (char*)shmat(shm_id,(void*)0,0);
       /* Print the message from Shm to buffer */
       strcpy(buffer,shm);
       /* Print buffer to the screen */
       printf("Read from shared memory: %s", buffer);
 }
```