Assignment no:08

Title: Shared Memory Segment.

Objectives:

To understand the Shared Memory,its functions and Client-Server communication using System V Shared Memory.

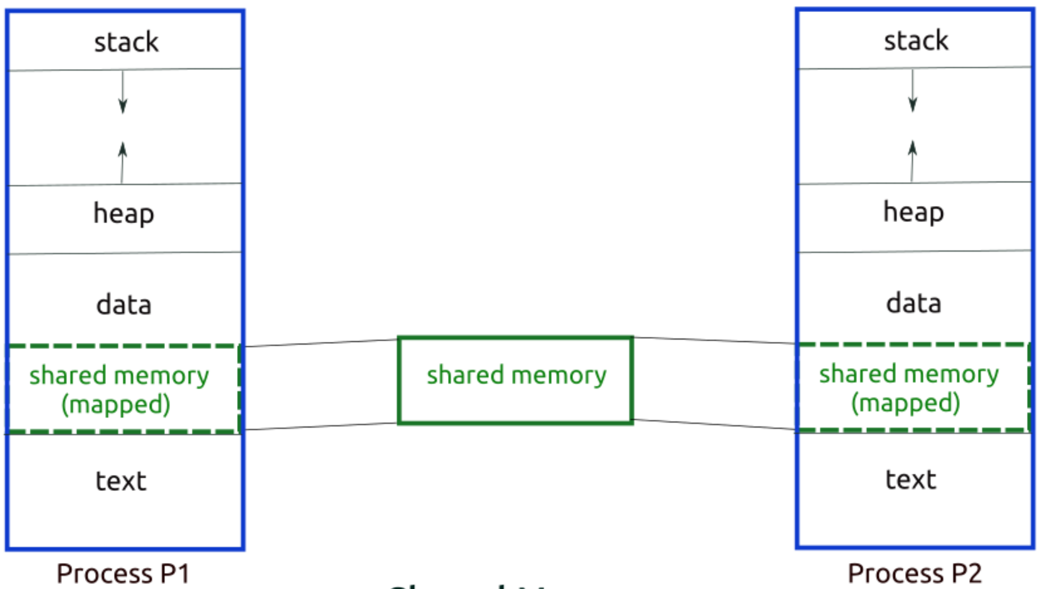
Problem Statement:

Inter Process Communication Using Shared Memory Using System V. Application to Demonstrate Client and Server Programs In Which Server Process Creates a Shared memory Segment and Writes the Message to the Shared Memory Segment.Client Process Reads The Message From the Shared Memory Segment and Displays it to the Screen

Theory:

Shared Memory

Shared memory is one of the three interprocess communication (IPC) mechanisms available under Linux and other Unix-like systems. The other two IPC mechanisms are the message queues and semaphores. In case of shared memory, a shared memory segment is created by the kernel and mapped to the data segment of the address space of a requesting process. A process can use the shared memory just like any other global variable in its address space.



Shared Memory

In the interprocess communication mechanisms like the pipes, fifos and message queues, the work involved in sending data from one process to another is like this. Process *P1* makes a system call to send data to Process *P2*. The message is copied from the address space of the first process to the kernel space during the system call for sending the message. Then, the second process makes a system call to receive the message. The message is copied from the kernel space to the address space of the second process. The shared memory mechanism does away with this copying overhead. The first process simply writes data into the shared memory segment. As soon as it is written, the data becomes available to the second process. Shared memory is the fastest mechanism for interprocess communication.

Functions:

1. shmget:

#include <sys/ipc.h>

#include <sys/shm.h>

int shmget (key_t key, size_t size, int shmflg);

As the name suggests, shmget gets you a shared memory segment associated with the given *key*. The key is obtained earlier using the ftok function. If there is no existing shared memory segment corresponding to the given *key* and IPC_CREAT flag is specified in *shmflg*, a new shared memory segment is created. Also, the *key* value could be IPC_PRIVATE, in which case a new shared memory segment is created. *size* specifies the size of the shared memory segment to be created; it is rounded up to a multiple of PAGE_SIZE. If *shmflg* has IPC_CREAT | IPC_EXCL specified and a shared memory segment for the given key exists, shmget fails and returns -1, with errno set to EEXIST. The last nine bits of *shmflg* specify the permissions granted to owner, group and others. The execute permissions are not used. If shmget succeeds, a shared memory identifier is returned. On error, -1 is returned and errno is set to the relevant error.

2. shmat:

#include <sys/types.h>

#include <sys/shm.h>

void *shmat (int shmid, const void *shmaddr, int shmflg);

With shmat, the calling process can attach the shared memory segment identified by shmid. The process can specify the address at which the shared memory segment should be attached with shmaddr. However, in most cases, we do not care at what address system attaches the shared memory segment and shmaddr can conveniently be specified as NULL. shmflg specifies the flags for attaching the shared memory segment. If shmaddr is not null and SHM_RND is specified in shmflg, the shared memory segment is attached at address rounded down to the nearest multiple of SHMLBA, where SHMLBA stands for Segment low boundary address. The idea is to attach at an address which is a multiple of SHMLBA. On most Linux systems, SHMLBA is the same as PAGE_SIZE. Another flag is SHM_RDONLY, which means the shared memory segment should be attached with read only access.

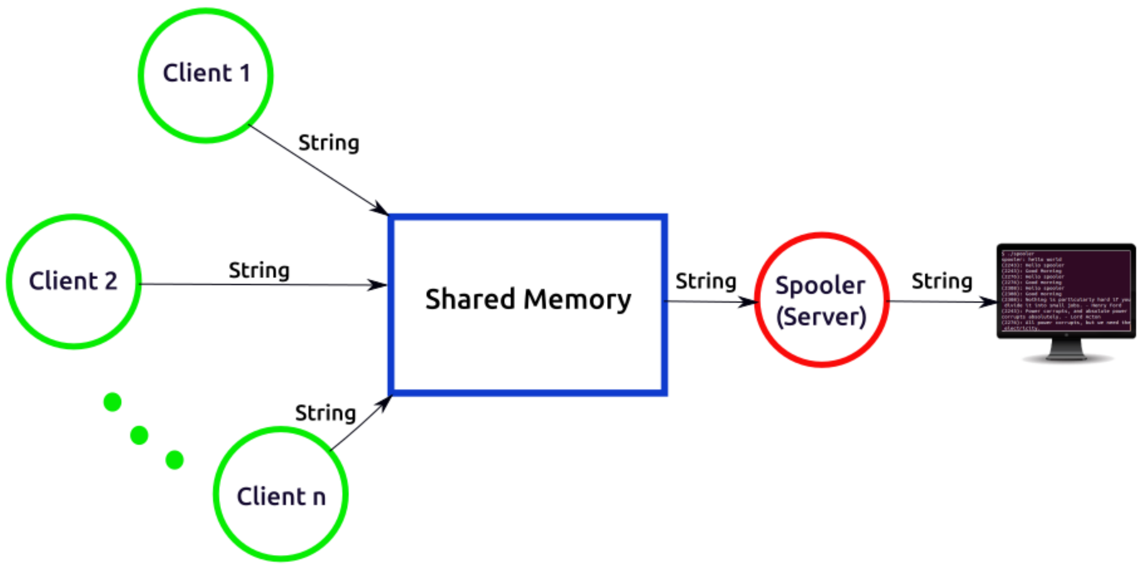
3. shmdt:

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

 #include <sys/shm.h>

 int shmdt (const void *shmaddr);
```

shmdt detaches a shared memory segment from the address space of the calling process. *shmaddr* is the address at which the shared memory segment was attached, being the value returned by an earlier shmat call. On success, shmdt returns 0. On error, shmdt returns -1 and errno is set to indicate the reason of error.

An example: Client-Server communication using System V Shared Memory

The example has a server process called spooler which prints strings received from clients. The spooler is a kind of consumer process which consumes strings. The spooler creates a shared memory segment and attaches it to its address space. The client processes attach the shared memory segment and write strings in it. The client processes are producers; they produce strings. The spooler takes strings from the shared memory segment and writes the strings on the terminal. The spooler prints strings in the order they are written in the shared memory segment by the clients. So, effectively, there are *n* concurrent processes producing strings at random times and the spooler prints them nicely in the chronological order. The software architecture is like this.



Software Architecture for Clients and Server using Shared Memory

Header Files:

#include <sys/ipc.h>

The <sys/ipc.h> header is used by three mechanisms for interprocess communication (IPC): messages, semaphores and shared memory. All use a common structure type, ipc_perm to pass information used in determining permission to perform an IPC operation.

#include <sys/shm.h>

time_t shm_ctime Time of last change by shmctl (). The pid_t, time_t, key_t, and size_t types shall be defined as described in <*sys*/types.*h*>. In addition, all of the symbols from <*sys*/*ipc*.*h*> shall be defined when this header is included.

Conclusion:

In this practical we studied shared memory management.how to manage memory via server to client.