33231

CLASSMATE
Date :
Page :

ASSIGNMENT 8

Title: Inter-process communication using shared memory using System V

Problem Statement : 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 :-

Inter-process communication (IPC) through shared memory is a concept where two or more processes can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process. The problem with pipes, fifo and message queue - is that for two processes to exchange information, the information has to go through the kernel.

- Server reads from the input file
- The server writes this data in a message using either a pipe, fifo or message queue.
- The client reads the data from the IPC channel, again requiring the data to be copied from Kernel's IPC buffer to the client's buffer.
- Finally the data is copied from the client's buffer.

System Calls used :

- ftok() - is used to generate a unique key
- shmget() - int shmget (key_t size, int shmflag); upon successful

completion, shmget() returns an identifier for the shared memory segment.

- shmat() – Before you can use shared memory segment, you have to attach yourself to it using shmat().
  void *shmat(int shmid, void *shmaddr, int shmflg);
  shmid is shared memory id. shmaddr specifies address to use but we should set it to zero and OS will automatically choose the address
- shmdt() – When you're done with the shared memory segment, your program should detach itself from it using shmdt(). int shmdt(void *shmaddr);
- shmctl() – When you detach from shared memory, it is not destroyed. So, to destroy shmctl() is used.
  shmctl(int shmid, IPC_RMID, NULL);

Conclusion: I have successfully understood and implemented inter-process communication using shared memory using System V.