SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
II Year CSE – A & B Sections (IV Semester)

Academic Year 2022-23

UCS2412 - OPERATING SYSTEMS LAB

Lab Exercise 5 Inter-process Communications using Shared Memory

Study the following system calls.
Shared memory - shmget, shmat, shmdt, shmctl

Additional optional system calls
Pipe - pipe, mkfifo, mknod, open, read, write, close
Message Queue – msgget, msgsnd, msgrcv, msgctl

Aim:
Develop the following applications that use inter-process communication concepts using shared memory.

1.Develop an application for getting a name in parent and convert it into uppercase in child using shared memory.
2.Develop a client / server application for file transfer using shared memory.
3. Develop a client/server chat application using shared memory.

Comment:

This program should be run as a single program in which parent is one process and child is another process.

Client / server:

Keep common code and parent code in one program as server.c

Keep common code and child code in another program as client.c

Run server.c in one terminal and client.c in another terminal. Communicate between the two processes.

Algorithm for parent and child process communication using shared memory

1.Generate shared memory identifier using shmget() with IPC_PRIVATE as its key_t
2. Generate the pid using fork()
3. If pid is greater than 0, then
    a. Attach a variable, name to the shared memory using shmat()
    b. Get name from the user
    c. Detach the variable, name from the shared memory using shmdt()
4. Else if pid is 0, then
    a. Attach a variable, str to the shared memory using shmat()
    b. Read the name from the shared memory and convert the string to uppercase
    c. Print the converted name
    d. Detach the variable, str from the shared memory using shmdt()
5. Remove the shared memory identifier and destroy the shared segment using shmctl()

SAMPLE I/O:
Parent Process:
Enter a name to convert into uppercase:

Child process:
Name in Uppercase

Algorithm for client / server application for file transfer using shared memory.

Client:

1. Generate key for the shared memory using ftok()
2. Generate shared memory identifier using shmget()
3. Attach the structure pointer to the shared memory using shmat() to store the file name and the contents of the file
4. Set the status to 0
5. Get the file name from the user
6. Wait until the status is not 0 using continue in while loop
7. if status is -1, then
    a. print error message
    b. exit
8. Open a new file in write mode using file pointer
9. Read the content from buff and store it in the file - character wise
10. Close the file
11. Set status to 0
12. Detach the pointer from the shared memory using shmdt()
13. Remove the shared memory identifier and destroy the shared segment using shmctl()

Server:

1. Generate key for the shared memory using ftok()
2. Generate shared memory identifier using shmget()
3. Attach the structure pointer to the shared memory using shmat() to store the file name and the contents of the file
4. Read the file name from the shared memory
5. Open the file in read mode using file pointer
6. if file pointer is null, then
    a. print error message
    b. set status to -1
    c. exit
7. Read the contents of the file character wise
8. Print and store it in buff
9. Close the file
10. Set status to 1
11. If status is 1, then detach the pointer from the shared memory using shmdt()

SAMPLE I/O:

Client:
Enter file to read data from:

Contents of the file:

Server:
File name from client:

Algorithm for client/server chat application using shared memory

client.c:

1. Get pid using getpid()
2. Generate shared memory identifier using shmget()
3. Set key to some value
4. Attach the structure pointer to the shared memory using shmat()
5. Assign pid2 to pid in step 1 and set status to NOTREADY
6. Signal the handler function to receive message using signal()
7. Get message from the user
8. Set status to READY
9. Send the message using kill()
10. Wait until the status is not READY using continue in while loop
11. Repeat until the user exits explicitly
12. Detach the pointer from the shared memory using shmdt()

server.c

1. Get pid using getpid()
2. Generate shared memory identifier using shmget()
3. Set key to some value

4. Attach the structure pointer to the shared memory using shmat()
5. Assign pid1 to pid in step 1 and set status to NOTREADY
6. Signal the handler function to receive message using signal()
7. Wait until the status is READY using continue in while loop
8. Get message from the user
9. Set status to FILLED
10. Send the message using kill()
11. Repeat until the user exits explicitly
12. Detach the pointer from the shared memory using shmdt()
13. Remove the shared memory identifier and destroy the shared segment using shmctl()

SAMPLE I/O:

Client:
Hello

Received from server: Hi
This is a chat application

Bye

Server:
Received from client: Hello
Hi

Received from client: This is a chat application

Bye

Example :
Shared memory :

```
#include <sys/ipc.h>
# define NULL 0
#include <sys/shm.h>
#include <sys/types.h>
# include<unistd.h>
# include<stdio.h>
# include<stdlib.h>
# include<string.h>
#include <sys/wait.h>
#include <stdio_ext.h>
// parent writing a char in shared memory and child reads it and prints it.
int main()
```

```
{
	int pid;
	char *a,*b,c;
	int id,n,i;
	// you can create a shared memory between parent and child here or you can //create
	inside them separately.
	id=shmget(111,50,IPC_CREAT | 00666);
	pid=fork();
	if(pid>0) //parent
		{
			// id=shmget(111,50,IPC_CREAT | 00666);
			a=shmat(id,NULL,0);
			a[0]='d';
			wait(NULL);
			shmdt(a);
		}
	else //child
		{
			sleep(3);
			//id=shmget(111,50,0);
			b=shmat(id,NULL,0);
			printf("\n child %c\n",b[0]);
			shmdt(b);
		}
	shmctl(id, IPC_RMID,NULL);
}
```