Name : V.P.Abishek **Roll no: 205001002**
Ex -5

System calls:

1. Name:shmget()
   Description: returns the identifier for the shared memory segment associated with the value of the argument key
   Header file:sys/shm.h
   Syntax:int shmget(key_t key,size_t size,int shmflg);
   Arguments:
   Key - it identifies the shared memory segment
   Size - size of the shared segment
   Shmflg - specifies the required shared memory flag(s). Need to pass permissions as well.
   Return type:
   Success:returns valid shared memory identifier
   Failure:returns -1 and errno is set to indicate the error

2. Name:shmat()
   Description: attaches the shared memory segment identified by shmid to the address space of the calling process
   Header file:sys/shm.h
   Syntax:void *shmat(int shmid, const void *shmaddr, int shmflg);
   int shmdt(const void *shmaddr);
   Arguments:
   Shmid - shared memory identifier, which is the return value of shmget() system call.
   Shmaddr - specifies the address that attaches to the calling process.
   Shmflg - specifies the required shared memory flag/s.
   Return type:
   Success: returns address of the attached shared memory segment
   Failure:returns -1

3. Name:shmdt()
   Description:detaches the shared memory segment located at the address specified by shmaddr from the address space of the calling process
   Header file:sys/types.h
   Syntax:int shmdt(const void *shmaddr);
   Arguments:Shmaddr - the address of the shared memory segment to be detached. The to-be-detached segment must be the address returned by the shmat() system call.
   Return type:
   Success:returns 0

Failure:returns -1

4. Name:shmctl()
   Description: performs the control option specified by cmd on the system shared memory segment whose identifier is given by shmid
   Header file:sys/shm.h
   Syntax:int shmctl(int shmid,int cmd,struct shmid_ds *buf);
   Arguments:
   Shmid - shared memory identifier, which is the return value of shmget() system call.
   Cmd - command to perform the required control operation on the shared memory segment.
   Buf - pointer to the shared memory structure named struct shmid_ds.
   Return type:
   Success:returns 0
   Failure:returns -1

1)Develop an application for getting a name in parent and convert it into uppercase in child using
shared memory.

Aim :
To develop an application for getting a name in parent and convert it into uppercase in child using shared memory.

Algorithm :Conversion to uppercase:

1.1. Create an interface of the shared memory.
1.2. Create the child process using fork()
1.3. While parent is executed
    1.3.1. Attach a variable ‘a’ to shared memory using shmat.
    1.3.2. Read the string from the user and store it in ‘a’.
    1.3.3. Print the string ‘a’.
    1.3.4. Detach ‘a’ from memory.
1.4. While the child process is executed.
    1.4.1. Attach a variable ‘b’ to shared memory using shmat.
    1.4.2. Wait until the parent reads the string from the user.
    1.4.3. Convert it to uppercase.
    1.4.4. Detach ‘b’ from shared memory

Source code:

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

```c
#include<unistd.h>
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
#include<sys/wait.h>

int main() {
  int pid, id,key=27;
  char * a, * b;
  id = shmget(key,1024, IPC_CREAT | 00666);
  pid = fork();

  if (pid > 0) {
      printf("Parent Process\n");
      a = shmat(id, NULL, 0);
      a[0] = '\0';
      printf("Enter a string: ");
      scanf("%[^\n]", a);
      wait(NULL);
      shmdt(a);
  }

  else {
      b = shmat(id, NULL, 0);
      while (b[0] == '\0');
      printf("\nChild Process:\n");
      for(int i=0;i<strlen(b);i++){
      if(islower(b[i])){
            b[i]=b[i]-'a'+'A';
      }
      }
      printf("Uppercase: %s\n", b);
      shmdt(b);

  }

  shmctl(id, IPC_RMID, NULL);
}
```

Output:

```
root@spl2:~/Desktop/OS_Abi/ex5# gcc -o p1 p1.c
root@spl2:~/Desktop/OS_Abi/ex5# ./p1
Parent Process
Enter a string: Hello

Child Process:
Uppercase: HELLO
root@spl2:~/Desktop/OS_Abi/ex5#
```

2) Develop an client / server application for file transfer using shared memory.

Aim :
To develop a client / server application for file transfer using shared memory.

Algorithm :

1. We create a structure for memory with the elements as 2 file names, a buffer to hold the content, pid1,pid2 and a flag called status.
2. We create an id for the memory and set status as 0
3. Read the source filename from the user as file1 and destination file as file2
4. Set status as 1 and send signal to server
5. In the server when the signal is received, it opens the source file with read only permissions
6. It copies the content from file1 and stores it in a buffer
7. The flag is then set 0 again and the client receives the signal
8. The client copies the contents from the buffer and pastes it in the destination file
9. Print the contents of file2
10. Detach and delete the shared memory

Source code:
Client:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <unistd.h>

int main(){
      key_t key = ftok("file1",100);
      int shmid = shmget(key,1024,0666|IPC_CREAT);
      char *str = (char*) shmat(shmid,(void*)0,0);

      printf("Enter the file name : ");
      int s=scanf("%s",str);
```

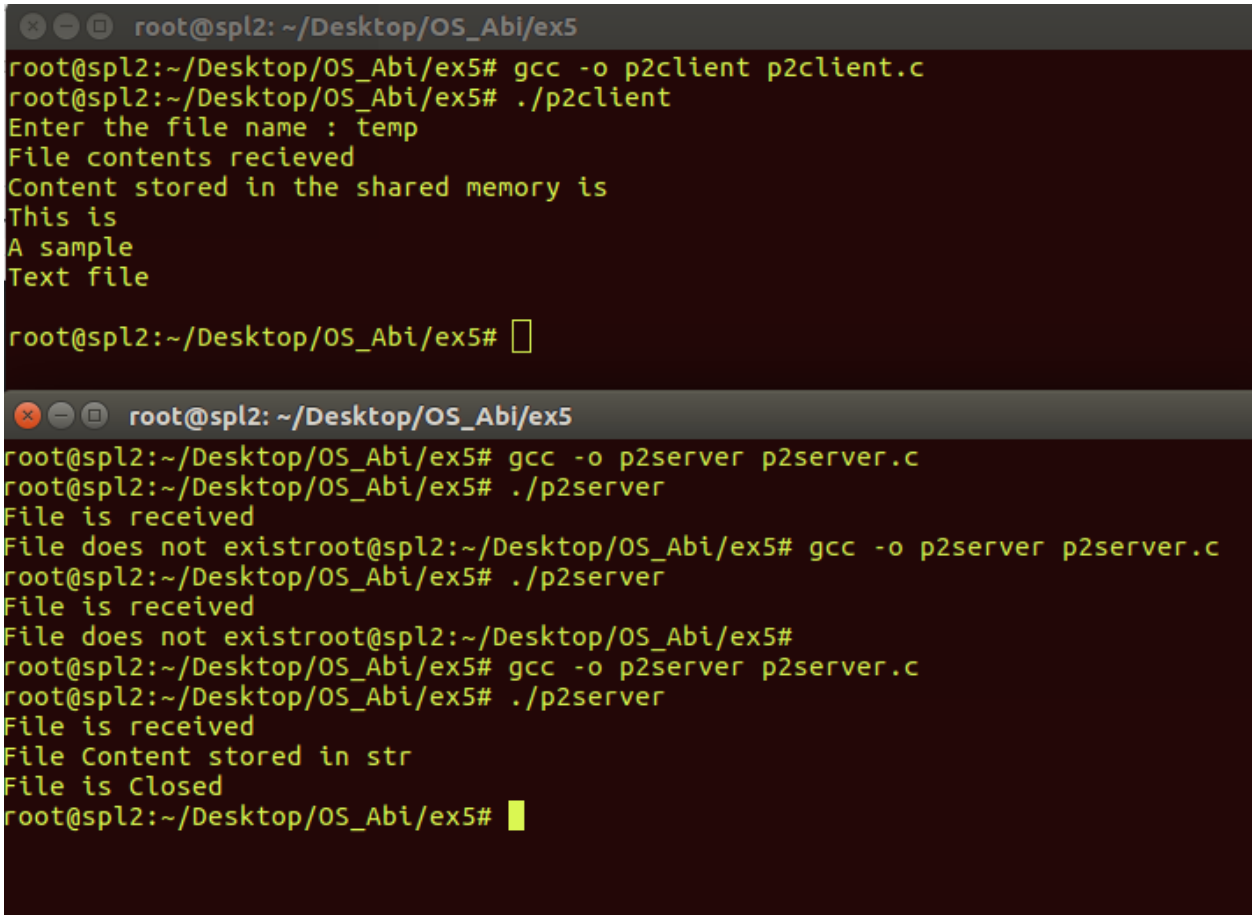
```
	sleep(1);
	if(strcmp(str,"EMPTY")==0){
	printf("File does not exist !!!!");
	return 0;
	}
	else{
	printf("File contents recieved\n");
	printf("Content stored in the shared memory is\n%s\n",str);
	FILE *fp=fopen("new_file.txt","w");
	fputs(str,fp);
	}
	shmdt(str);
	shmctl(shmid,IPC_RMID,NULL);
	return 0;
}
```

Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <unistd.h>

int main()
{

	key_t key = ftok("file1",100);

	int shmid = shmget(key,1024,0666|IPC_CREAT);

	char *str = (char*) shmat(shmid,(void*)0,0);

	while(str[0]=='\0');
	FILE *fd;
	printf("File is received\n");
	fd = fopen(str, "r");
	if(fd==NULL){
	printf("File does not exist");
	str[0]='E';str[1]='m';str[2]='p';str[3]='t';str[4]='y';str[5]='\0';
	return 0;
	}
	char ch;
```

```
        int i=0;
        while((ch=fgetc(fd))!=EOF){
        *(str+i)=ch;
        i++;
        }
        *(str+i)='\0';
        printf("File Content stored in str\n");
        fclose(fd);

        printf("File is Closed\n");
        shmdt(str);
        return 0;
}
```

Output:



3) Develop a client/server chat application using shared memory.

Aim : To develop a client/server chat application using shared memory.

Algorithm :

1.1.

Client.c

1.1.1. Use ftok() to generate a unique key.
1.1.2. Obtain a shared memory identifier for the key using shmget().
1.1.3. Attach str to it using shmat()
1.1.4. Read the client to it using shmat()
1.1.5. Wait for the reply from server/
1.1.6. Print the server reply.
1.1.7. Clear the content of the string str.
1.1.8. Go to step 2

1.2. Server.c

1.2.1. Use ftok() to generate a unique key.
1.2.2. Obtain a shared memory identifier for the key using shmget().
1.2.3. Attach str to it using shmat().
1.2.4. Wait for the message from the client.
1.2.5. Print the client message
1.2.6. Clear the content of the string str.1.2.7.
1.2.7.Read the reply from the user.
1.2.8Go to step 4.

Source code:

Client:

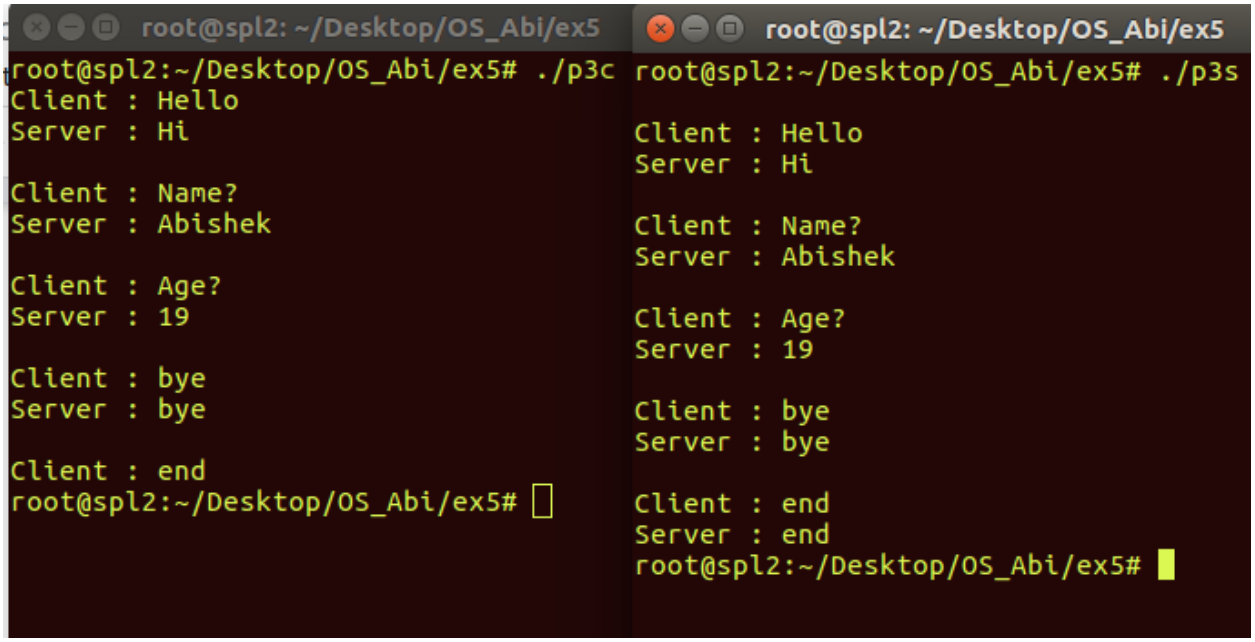
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <unistd.h>

int main() {

      key_t key =27;
      int shmid = shmget(key,1024,0666|IPC_CREAT);
      char *str = (char*)shmat(shmid,(void*)0,0);

      while (1) {
      while (str[0] == '\0');
      printf("Client : %s\n",str);
      str[0]='\0';
      printf("Server : ");
      scanf("%s",str);
```

```
	if(strcmp(str,"end")==0){
		return 0;
	}
	printf("\n");
	sleep(1);
	}
}
```

Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <unistd.h>
#include<sys/wait.h>

int main() {

	key_t key = 27;
	int shmid = shmget(key,1024,0666|IPC_CREAT);
	char *str = (char*)shmat(shmid,(void*)0,0);

	while (1) {
	str[0] = '\0';
	printf("Client : ");
	scanf("%s",str);
	if(strcmp(str,"end")==0){
		return 0;
	}
	sleep(1);
	while(str[0]=='\0');

	printf("Server : %s\n\n",str);
	}
}
```

Output:
Left side - client, right side- server



Learning Outcome :

1. Learned interprocess communication using shared memory.
2. Learned to implement file transfer through shared memory.
3. Learned to implement a chat application using shared memory.