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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

LAB EXERCISE 5

Inter Process Communication

Submission Date:07-04-2022

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

System Calls

1. Name:shmget()

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

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

3. Name:shmdt()
    a. Description:detaches the shared memory segment located at the address specified by shmaddr from the address space of the calling process
    b. Header file:sys/types.h
    c. Syntax:int shmdt(const void *shmaddr)

d. 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.
e. Return type:
f. Success:returns 0
g. Failure:returns -1

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

Develop the following applications that uses interprocess 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.

Algorithm:

1) Fork() is called and the children id is stored in pid
2) If pid is equal to zero then
   a) Created a unique key for a project using ftok() and stored in key
   b) For the key, shared memory is allotted using shmget and returned id is stored in shmid
   c) Get input from the user which to be stored in shared memory
   d) Detach from the memory
3) Else then
   a) Created a unique key for a project using ftok() and stored in key
   b) For the key, shared memory is allotted using shmget and returned id is stored in shmid
   c) Read the data in shared memory and convert it to uppercase then display it
   d) Detach from the memory
   e) Erase the shared memory

Code:

```
#include <sys/ipc.h>
#define NULL 0
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <sys/wait.h>
#include <stdio_ext.h>

int main()
{
    int pid = fork();
    if (pid == 0)
    {
        int shmid = shmget(111, 1024, 0666 | IPC_CREAT);
        char *str = (char *)shmat(shmid, (void *)0, 0);
        printf("Data to be written in memory:");

        fgets(str, 100, stdin);
        // printf("Data written in memory: %s\n", str);
        shmdt(str);
    }
    else
    {
        wait(NULL);
        int shmid = shmget(111, 1024, 0666 | IPC_CREAT);
        char *str = shmat(shmid, (void *)0, 0);
        printf("\nActual Data read from memory: %s\n", str);

        for (int i = 0; str[i] != '\0'; i++)
        {
            if (str[i] >= 'a' && str[i] <= 'z')
            {
                str[i] = str[i] - 32;
            }
        }

        printf("Data to be displayed from memory: %s\n", str);
        shmdt(str);
        shmctl(shmid, IPC_RMID, NULL);
    }
}
```

Output:

```
jayannthan_hakr@jayannthan-Ubuntu:~/OS LAB/Assignment5$ ./1
Data to be written in memory:hi! how are you?

Actual Data read from memory: hi! how are you?

Data to be displayed from memory: HI! HOW ARE YOU?
```

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

Algorithm for server:

1) Created a unique key for a project using ftok() and stored in key

2) For the key, shared memory is allotted using shmget and returned id is stored in shmid
3) read input from the shared memory and store it in str
4) open file of name str and store file pointer in fp
5) read the file and write it into the shared memory using same str
6) close the file

Algorithm for client:

1) Created a unique key for a project using ftok() and stored in key
2) For the key, shared memory is allotted using shmget and returned id is stored in shmid
3) read input filename from the user and store it in the shared memory
4) After the server writes the file contents into the shared memory, display it and write it into a new file
5) Detach from the memory
6) Erase the memory

Code:

```c
/*Server Code*/

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

int main()
{

    int shmid = shmget(1, 50, 666 | IPC_CREAT);
    char *str = (char *)shmat(shmid, (void *)0, 0);

    while (str[0] == '\0')
        ;

    printf("File name received\n");

    FILE *fp;
    fp = fopen(str, "r");
    if (fp == NULL)
    {
        strcpy(str, "File not found\n");
    }
    else
    {
        printf("Reading the file...\n");
        char c;
```

```
        int i = 0;
        while ((c = fgetc(fp)) != EOF)
        {
            str[i] = c;
            i++;
        }
        str[i] = '\0';
        printf("File content fetched successfully!\n");
        fclose(fp);
    }
}
```

```
/*Client Code*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main()
{

    int shmid = shmget(1, 50, 666 | IPC_CREAT);
    char *str = (char *)shmat(shmid, (void *)0, 0);

    printf("Enter file name : ");
    scanf("%s", str);

    printf("Waiting for file content from server...\n");
    sleep(1);

    if (strcmp(str, "File not found") == 0)
    {
        printf("File not found\n");
    }
    else
    {
        printf("Content received\n");
        printf("File Contents :\n%s", str);
    }
    shmdt(str);

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

Output:

```
Enter filename: main.c

File: main.c
File Request Sent...
Contents of File:
#include <stdio.h>

int main(int argc, char const *argv[])
{
        printf("Hello World!!!\n");
        return 0;
}
```

3. Develop an client/server chat application using shared memory.

Algorithm for server:

1) Get process id using getpid() and store it in pid
2) Create a shared memory using shmget and returned id is stored in shmid
3) Attach the pointer of message structure (memory) into the shared memory
4) Set pid2 in the memory as pid
5) Set status as -1
6) Call signal(). Send SIGUSR2, handler function as parameters
7) Loop until exit
    a) If status is equal to 1 then wait for the other user to give input
    b) Else then get input from the user to chat
    c) Then set status as 1
    d) Signal all the process using kill. Send pid1, SIGUSR1 as parameters
8) Detach from the memory
9) Destroy the memory

Algorithm for client:

1) Get process id using getpid() and store it in pid
2) Create a shared memory using shmget and returned id is stored in shmid
3) Attach the pointer of message structure (memory) into the shared memory
4) Set pid2 in the memory as pid
5) Set status as -1
6) Call signal(). Send SIGUSR1, handler function as parameters
7) Loop until exit
    a) If status is equal to 1 then wait for the other user to give input
    b) Else then get input from the user to chat
    c) Then set status as 0
    d) Signal all the process using kill. Send pid2, SIGUSR2 as parameters
8) Detach from the memory
9) Destroy the memory

Code:

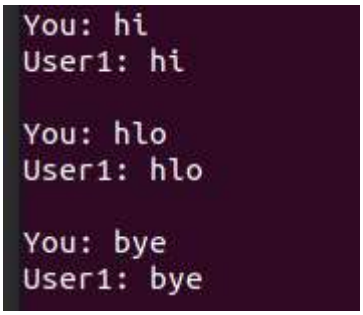
```
/*Server Code*/
```

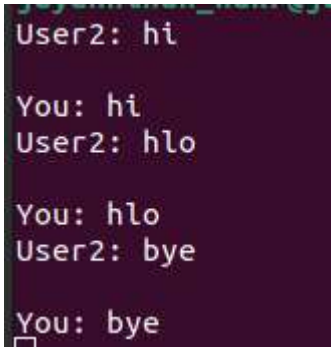
```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>

struct memory
{
    char buff[100];
    int status, pid1, pid2;
};

struct memory *shmptr;
void handler(int signum)
{
    if (signum == SIGUSR2)
    {
        printf("User1: ");
        puts(shmptr->buff);
    }
}

int main()
{
    int pid = getpid();
    int shmid = shmget(111, sizeof(struct memory), IPC_CREAT | 0666);
    shmptr = (struct memory *)shmat(shmid, NULL, 0);
    shmptr->pid2 = pid;
    shmptr->status = -1;
    signal(SIGUSR2, handler);
    while (1)
    {
        while (shmptr->status == 1)
            continue;
        sleep(1);
        printf("You: ");
        fgets(shmptr->buff, 100, stdin);
        shmptr->status = 1;
        kill(shmptr->pid1, SIGUSR1);
    }
    shmdt((void *)shmptr);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

```c
/*Client Code*/

#include <signal.h>
#include <stdio.h>
```

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

struct memory
{
    char buff[100];
    int status, pid1, pid2;
};

struct memory *shmptr;
void handler(int signum)
{
    if (signum == SIGUSR1)
    {
        printf("User2: ");
        puts(shmptr->buff);
    }
}

int main()
{
    int pid = getpid();
    int shmid = shmget(111, sizeof(struct memory), IPC_CREAT | 0666);
    shmptr = (struct memory *)shmat(shmid, NULL, 0);
    shmptr->pid1 = pid;
    shmptr->status = -1;
    signal(SIGUSR1, handler);
    while (1)
    {
        while (shmptr->status != 1)
            continue;
        sleep(1);
        printf("You: ");
        fgets(shmptr->buff, 100, stdin);
        shmptr->status = 0;
        kill(shmptr->pid2, SIGUSR2);
    }
    shmdt((void *)shmptr);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

Output:





Learning Outcome:

- Executed shared memory functions and system calls
- Executed server-side and client-side program using shared memory