**Bahria University, Lahore Campus**

Department of Computer Sciences

Lab Journal 06

**(Spring 2024)**

|  |  |  |
| --- | --- | --- |
| Course: | **Operating System Lab** | Date: 03-28-2024 |
| Course Code: | CSL – 320 | Max Marks: 20 |
| Faculty’s Name: | ABDULLAH |  |

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Enroll No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Objective(s) :

To understand how the processes will cooperate by communicating with each other using the approach called as shared memory approach.

## Lab Tasks :

**Task 01:** Write the steps for sharing a common memory segment.

**Task 02:** Write the functions required for creating, attaching, detaching and removing the memory segment.

**Task 03:** Write a program in C to perform communication between parent and child through shared memory.

• Parent should be the server and child should be the client.

• Differentiate between parent and child through conditional statements after the fork system call.

• Parent:

Should create the shared memory and write Capital English “HELLO” in it, through those 5 steps taught.

• Child:

Should access that memory and read whatever is written by parent in it.

**Task 04:** Write a program that creates a shared memory segment and waits until three other separate processes writes something into that shared memory segment after which it prints what is written in shared memory.

For the communication between the processes to take place assume that the process 1 writes 1 in first position of shared memory and waits;

Process 2 writes 2 in first position of shared memory and goes on to write 'hello' and then process 3 writes 3 in first position of shared memory and goes on to write 'memory' and Process 4 writes “OS” and finally the process 1 prints what is in shared memory written by two other processes

**Lab Grading Sheet :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Max Marks** | **Obtained Marks** | **Comments(*if any*)** |
| 1. | 5 |  |  |
| 2. | 5 |  |  |
| 3. | 5 |  |  |
| 4. | 5 |  |  |
| **Total** | **20** |  | **Signature** |

**Note : Attempt all tasks and get them checked by your Lab. Instructor**.

# Lab 06: Inter Process Communication

**Objective(s):**

To understand how the processes will cooperate by communicating with each other using the approach called as shared memory approach.

**Tool(s) used:**

Ubuntu, VIM Editor

**Shared Memory:** Shared memory is the area (segment) of memory that is shared by morethan one process. It is basically a method of passing data between programs. This is by far the fastest form of IPC, because there is no intermediate link (i.e. a pipe, a message queue, etc). Instead, information is mapped directly from a memory segment, and into the addressing space of the calling process.

**Definition-Shared memory segment**: is described by a control structure with a uniqueinteger ID that points to an area of physical memory. A segment can be created by one process, and subsequently written to and read from by any number of processes.

**Critical Region:** One program will create a memory region, which is also called as acritical section and other process upon permission can access that region.

**Creating a Shared Memory Segment**

**Server:**

It should start before any client and should do thee following steps.

1. Ask for a shared memory with a memory key and memorize the returned shared memory ID. This is performed by system call shmget( ).
2. Attach shared memory to the server's address space with system call shmat( ).
3. Initialize the shared memory, if necessary.
4. Do whatever you want in shared memory (write something) and wait for all clients' completion.
5. Detach the shared memory with system call shmdt( ).
6. Remove the shared memory with system call shmctl( ).

**Client:**

1. Ask for a shared memory with the same memory key and memorize the returned shared memory ID. Do not write the flag IPC\_CREAT in the shmget() call from client.
2. Attach this shared memory to the client's address space with system call shmget().
3. Use the Shared memory (Read what was written by server).
4. Detach all shared memory segments, if necessary.
5. **Requesting for a Shared Memory Segment - shmget( ):**

Shmget( ) returns value

* 0 successfully get the requested shared memory < 0 failed to get the requested shared memory

**Key:**

A key is simply an integer of type key\_t; however, you should not use int or long, since the length of a key is system dependent.

There are three different ways of using keys, namely:

1. a specific integer value (e.g., 123456)
2. a key generated with function ftok( )
3. a uniquely generated key using IPC\_PRIVATE (i.e., a private key).

The system will generate a unique key and guarantee that no other process will have the same key. If a resource is requested with IPC\_PRIVATE in a place where a key is required, that process will receive a unique key for that resource. Since that resource is identified with a unique key unknown to the outsiders, other processes will not be able to share that resource and, as a result, the requesting process is guaranteed that it owns and accesses that resource exclusively.

**2) Attaching a Shared Memory Segment to an Address Space - shmat( ) :**

Suppose process 1, a server, uses shmget( ) to request a shared memory segment successfully. That shared memory segment exists somewhere in the memory, but is not yet part of the address space of process

Similarly, if process 2 requests the same shared memory segment with the same key value, process 2 will be granted the right to use the shared memory segment; but it is not yet part of the address space of process

So to make a requested shared memory segment part of the address space of a process, we use shmat( ).



System call shmat( ) accepts a shared memory ID, shm\_id, and attaches the indicated shared memory to the program's address space. The returned value is a pointer of type (void \*) to the attached shared memory. Thus, casting is usually necessary. If this call is unsuccessful, the return value is -1. Normally, the second parameter is NULL. If the flag is SHM\_RDONLY, this shared memory is attached as a read-only memory; otherwise, it is readable and writable.

**3) Writing/Reading the Shared Memory Segment:**

The pointer shm\_ptr returned returned by shmat() call points to the start of shared memory. To write or read from the shared memory we use this pointer.

**4) Detaching a Shared Memory Segment -shmdt( ):**

System call shmdt( ) is used to detach a shared memory.

After a shared memory is detached, it cannot be used. However, it is still there and can be re-attached back to a process's address space, perhaps at a different address.

where shm\_ptr is the pointer to the shared memory. This pointer is returned by shmat( ) when the shared memory is attached.

## 

**5) Removing a Shared Memory Segment -shmctl( ):**

To remove a shared memory, use shmctl( ).

where shm\_id is the shared memory ID. IPC\_RMID indicates this is a remove operation. This step is performed only from the sever side.

**Example Task 1:** Write a program in C to perform communication between parent and child through shared memory.

* Parent should be the server and child should be the client.
* Differentiate between parent and child through conditional statements after the fork system call.
* Parent:

Should create the shared memory and write Capital English alphabets from Z to A in it, through those 5 steps taught.

* Child:

Should access that memory and read whatever is written by parent in it.

**Algorithm:**

* Create parent and child through the fork system call.
* Create Shared memory in parent through private key.
* Attach the memory in address space
* Write from Z to A in it through the pointer to that shared memory.
* Access shared memory through child/client.
* Read whatever parent has written in shared memory.
* Compile by typing “gcc –o filename filename.c” in shell.
* Then execute it by “./filename”

**Code:**

#include <stdio.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/sem.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<stdio.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<sys/stat.h>

#define SIZE 100

int main(){

int shm\_id; key\_t key; char \*shm\_ptr, \*s;

int id\_=fork();

if(id\_==0)

{

//Child

key=9292;

shm\_id=shmget(key,SIZE,0666);

if(shm\_id<0)

{

perror("Erorr");

}

shm\_ptr=shmat(shm\_id,0,0);

if(shm\_ptr == (char\*)-1)

{

printf("Shmat Error");

}

//Start Reading

printf("Reading.....");

for(s=shm\_ptr; \*s!='\0'; s++)

printf("%c",\*s);

\*shm\_ptr='\*';

printf("\nI am Done");

}

else

{

//Parent

int shm\_id; key\_t key; char \*shm\_ptr, \*s;

key=9292;

shm\_id=shmget(key,SIZE,IPC\_CREAT | 0666);

if(shm\_id<0)

{

printf("Server Eror");

}

shm\_ptr=shmat(shm\_id,0,0);

if(shm\_ptr==(char\*)-1)

{

printf("SHMAT Eror");

}

s=shm\_ptr;

printf("Parent Writing..... \n");

for(char c='Z'; c>='A'; c--)

{

\*s=c;

s++;

}

\*s=’\0’;

printf("Waiting for Child\n");

while(\*shm\_ptr!='\*')

sleep(1);

shmdt(shm\_ptr);

if(shmctl(shm\_id,IPC\_RMID,NULL)==-1)

{

perror("shmctl");

exit(-1);

}

}

return 0;

}

**Example Task # 2:** Write a program that creates a shared memory segment and waits until two other separate processes writes something into that shared memory segment after which it prints what is written in shared memory.

For the communication between the processes to take place assume that the process 1 writes 1 in first position of shared memory and waits;

Process 2 writes 2 in first position of shared memory and goes on to write 'hello' and then process 3 writes 3 in first position of shared memory and goes on to write 'memory' and finally the process 1 prints what is in shared memory written by two other processes

**Code:**

**PROCESS 1**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define SHMSIZE 27

int main(){

char c;

int shmid;

char \*shm, \*s;

key\_t key;

key = 5678;

//Create Segment

if ((shmid= shmget(key, SHMSIZE, IPC\_CREAT | 0666))<0){

perror("shmget");

exit(1);

}

//Now attach the segment to our dataspace

shm=shmat(shmid, NULL,0);

//Now put 1 in first place

s=shm;

\*s++='1';

printf("Process 1 : I have put the message %s\n",shm);

//Finally, we waitt until the other process changes the first character of memory

while(\*shm !='2')

sleep(1);

printf("Process 1: Process 2 has put the messages %s\n",shm);

while(\*shm !='3')

sleep(1);

printf("Process 1: Process 3 has put the message %s\n",shm);

printf("Process 1: I am quiting\n");

shmdt(shm);

shmctl(shmid, IPC\_RMID,NULL);

return 0;

}

**PROCESS 2**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

# define SHMSIZE 27

int main(){

char c, \*shm, \*s;

int shmid;

key\_t key;

key=5678;

//Create the segment

if((shmid=shmget(key,SHMSIZE, 0666))<0){

perror("shmget");

exit(1);

}

//Now attach the segment to our database

shm= shmat(shmid,NULL,0);

//Process 2 writing hello in memory

s=shm;

s++;

\*s++=' ';

\*s++='h';

\*s++='e';

\*s++='l';

\*s++='l';

\*s++='o';

//Now put 2 in first place

\*shm='2';

shmdt(shm);

return 0;

}

**PROCESS 3**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

# define SHMSIZE 27

int main(){

char c, \*shm, \*s;

int shmid;

key\_t key;

key=5678;

//Create the segment

if((shmid=shmget(key,SHMSIZE, 0666))<0){

perror("shmget");

exit(1);

}

//Now attach the segment to our database

(shm= shmat(shmid,NULL,0);

//Process 3 is writing memory in memory

s=shm;

s++;

\*s++=' ';

\*s++='m';

\*s++='e';

\*s++='m';

\*s++='o';

\*s++='r';

\*s++='y';

//Now put 3 in first place

\*shm='3';

shmdt(shm);

return 0;

}

**OUTPUT**