Lab No. 07
Creating Multithreaded Applications

Objective
The objective of this lab is to familiarize students with the implementation of inter-process communication using shared memory and message passing.

Activity Outcomes
On completion of this lab students will be able to

- Write cooperating processes that share data using shared memory
- Write cooperating processes that share data using message passing

Instructor Notes
As pre-lab activity, read the content from the following (or some other) internet source: https://www.geeksforgeeks.org

1) Useful Concepts

Cooperating Processes

A process can be of two type: Independent or Cooperating. An independent process is not affected by the execution of other processes while a cooperating process can affect or be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilized for increasing computational speed, convenience and modularity. Cooperating processes need to share data with each other. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of cooperation between them. The most common methods to share data between processes include: Shared Memory and Message passing.

Shared Memory
Shared memory is a common way to share data between processes. In shared memory, the communicating processes request the OS to declare some memory that can be shared between them. OS declares this shared memory in user space and it is under the control of user processes. Once the shared memory is declared then it is attached with all of the communicating processes. Now, the writer process can write data in the shared memory that can be read by the reader processes.
Communication using shared memory takes place in the following steps

Writer Process	Reader Process
1. Declare shared memory if it is not already declared 2. Attach shared memory with the process 3. Write data in shared memory 4. Detach shared memory from the processes	1. Declare shared memory if it is not already declared 2. Attach shared memory with the process 3. Read data from shared memory 4. Detach shared memory from the processes

	5. Destroy shared memory if shared data is no more required

Now, we discuss the routines that can be used to implement shared memory.

Routine	Description
ftok( )	To generate a unique key
shmget(key, memory_size, shmflag)	Used to declare shared memory. Upon successful completion, *shmget()* returns an identifier for the shared memory segment.
*shmat(int shm_id, void *shmaddr, int shmflag)*	Before you can use a shared memory segment, you have to attach yourself to it using *shmat()*.shmid is shared memory id. shmaddr specifies specific address to use but we should set it to zero and OS will automatically choose the address.
*shmdt(void *shmaddr)*	Used to detach the process from the shared memory
shmctl(int shm_id, IPC_RMID, NULL)	It is used to destroy the shared memory

Message Passing

Message passing is another way to share data between processes. Communication between processes takes place in form of messages. This communication remains under the control of OS. OS that support message passing, provide facility/Services to send and receive messages. Inter-process communication using message passing takes place in the following steps:

Sender Process	**Receiver Process**
1. Request the OS to declare message queue if it is not already declared 2. Send message in the queue	1. Request the OS to declare message queue if it is not already declared 2. Receive message from the queue 3. Destroy message queue if message/shared data is no more required

Now, we discuss the routines that can be used to implement shared memory.

Routine	Description
ftok( )	To generate a unique key
msgget(key, shmflag)	Used to declare message queue. Upon successful completion, *shmget()* returns an identifier for the shared memory segment.
*msgsnd(int msg_id, void *msgaddr, size_t msgsize, int msgflag)*	Used to send message in the message queue
*msgsnd(int msg_id, void *msgaddr, size_t msgsize, int msgtype, int msgflag)*	Used to receive message from message queue
msgctl(int msg_id, IPC_RMID, NULL)	It is used to destroy the message queue

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*25*	*Medium*	*CLO-7*
2	*25*	*Medium*	*CLO-7*

Activity 1:

In this activity, we implement the shared memory. We have written two programs. The first program is writer.cpp. This program declares a shared memory and writes data in it. The second program; reader.cpp reads the data from shared memory written by the writer.

Code writer.cpp

*writer.cpp (~/) - gedit

```
#include<iostream>
#include<sys/ipc.h> //required for shared memory
#include<sys/shm.h> //required for shared memory
using namespace std;
int main()
{
key_t key=ftok("shmfile",100); //unique key is generated
int shm_id=shmget(key,128,0666|IPC_CREAT);//shared memory of szie 128 bytes is declared
char *sh_data=(char*)shmat(shm_id,0,0);// shared memory is attahed with process
cout<<"Write YOur Message"<<endl;
cin.getline(sh_data,100);// data is written to shared memory
shmdt(sh_data); // process is detached from shared memory
return 0;
}
```

Code reader.cpp

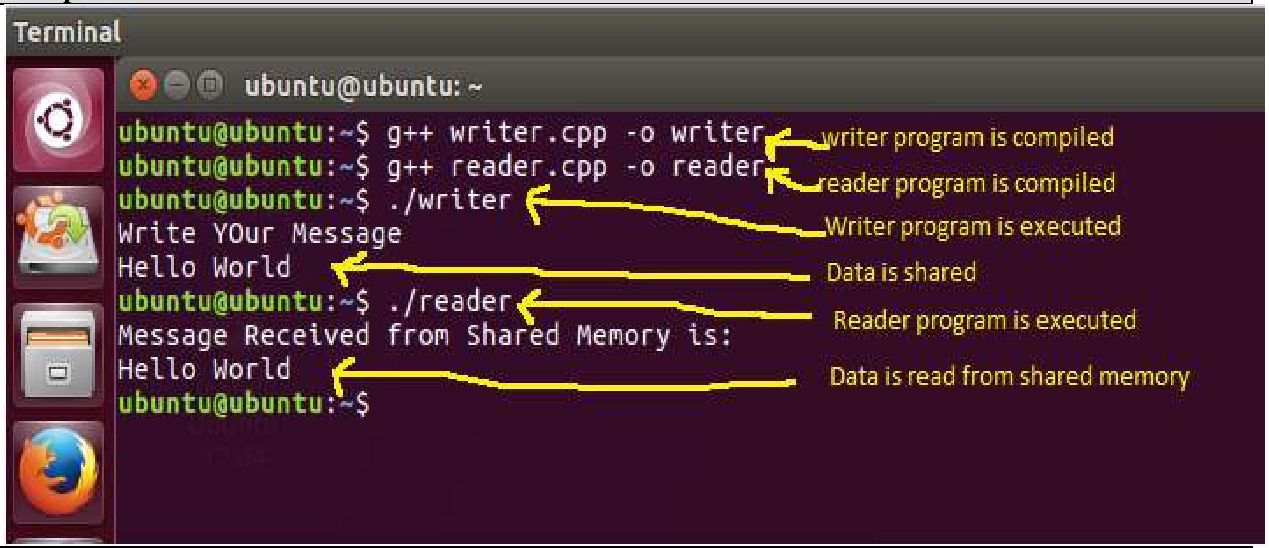
OS test [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

reader.cpp (~/) - gedit

```
#include<iostream>
#include<sys/ipc.h> //required for shared memory
#include<sys/shm.h> //required for shared memory
using namespace std;
int main()
{
key_t key=ftok("shmfile",100); //unique key is generated
int shm_id=shmget(key,128,0666|IPC_CREAT);//shared memory of szie 128 bytes is declared
char *sh_data=(char*)shmat(shm_id,0,0);// shared memory is attahed with process
cout<<"Message Received from Shared Memory is:"<<endl;
cout<<sh_data<<endl;// data is read from shared memory
shmdt(sh_data); // process is detached from shared memory
shmctl(shm_id,IPC_RMID,NULL); //shared memory is destroyed
return 0;
}
```

Out-put



Activity 2:

In this activity, we show how the IPC is implemented using the message passing. First, we have written a program sender.cpp that declares a message queue and sends a message in it. The second program receiver.cpp receives the message from the message queue.

Code sender.cpp

```
#include<iostream>
#include<sys/ipc.h> //required for IPC
#include<sys/msg.h> //required for message passing
using namespace std;
struct message  // structure for message is defined
{
long message_type;
char message_text[100];
};
int main()
{
message msg; // variable of type message is declared
msg.message_type=1;
cout<<"Please Write Your Message"<<endl;
cin.getline(msg.message_text, 100); //message is typed
key_t key=ftok("shmfile",100); //unique key is generated
int msg_id=msgget(key,0666|IPC_CREAT);//message queue is declared
msgsnd(msg_id,&msg,sizeof(msg.message_text),0);// mesage is sent to message queue
return 0;
}
```

Code receiver.cpp

receiver.cpp (~/) - gedit

```cpp
#include<iostream>
#include<sys/ipc.h> //required for IPC
#include<sys/msg.h> //required for message passing
using namespace std;
struct message  // structure for message is defined
{
long message_type;
char message_text[100];
};
int main()
{
message msg; // variable of type message is declared
key_t key=ftok("shmfile",100); //unique key is generated
int msg_id=msgget(key,0666|IPC_CREAT);//message queue is declared
msgrcv(msg_id,&msg,sizeof(msg.message_text),1,0);// mesage is received from message queue
cout<<"Message Received From Message Queue is;"<<endl;
cout<<msg.message_text<<endl;// message is displayed
return 0;
}
```

Out-put

ubuntu@ubuntu: ~

```
ubuntu@ubuntu:~$ g++ sender.cpp -o sender        sender program is compiled
ubuntu@ubuntu:~$ g++ receiver.cpp -o receiver    receiver program is compiled
ubuntu@ubuntu:~$ ./sender
Please Write Your Message                        sender is executed
Hello Pakistan                                   message is typed
ubuntu@ubuntu:~$ ./receiver                      receiver is executed
Message Received From Message Queue is;
Hello Pakistan                                   received message is displayed
ubuntu@ubuntu:~$
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Task 1:

Write a C++ program that creates 3 child processes. Each process creates a random number between 0 and 10. Now, each of the child process takes a value between 0 and 10 from user and compares it with the random number. If the user's guess is equal to the random number then user is declared winner. This process continues as long as the user wins. In the end, each child process shares the number of turns taken by each user to win. The parent process then decides the winner i.e. the user that takes minimum number of turns to win.