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

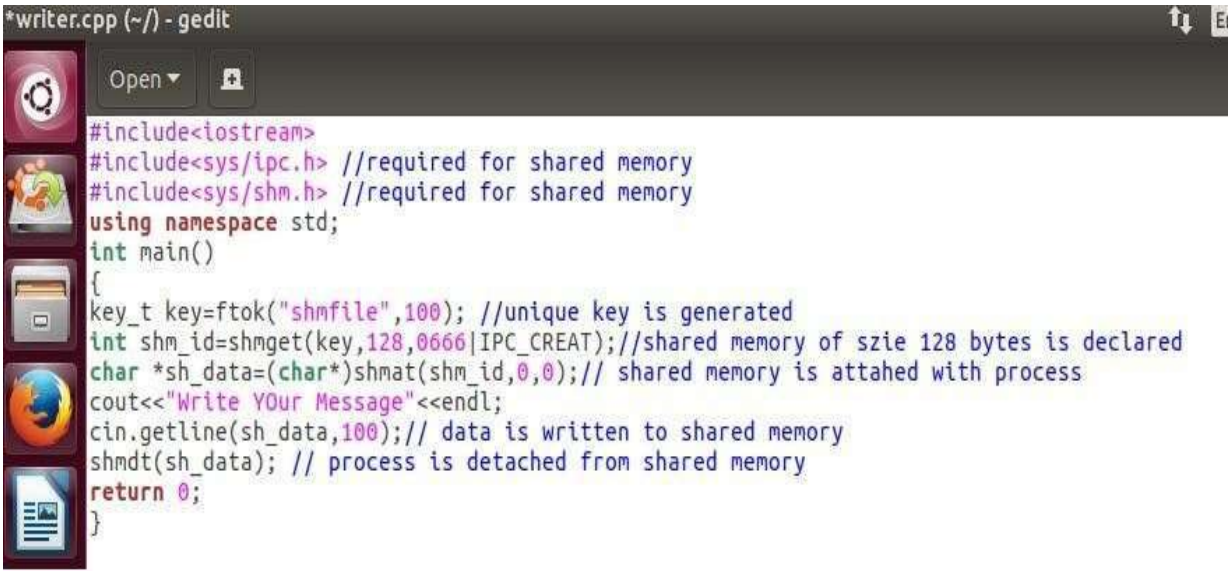
2) Solved Lab Activities

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

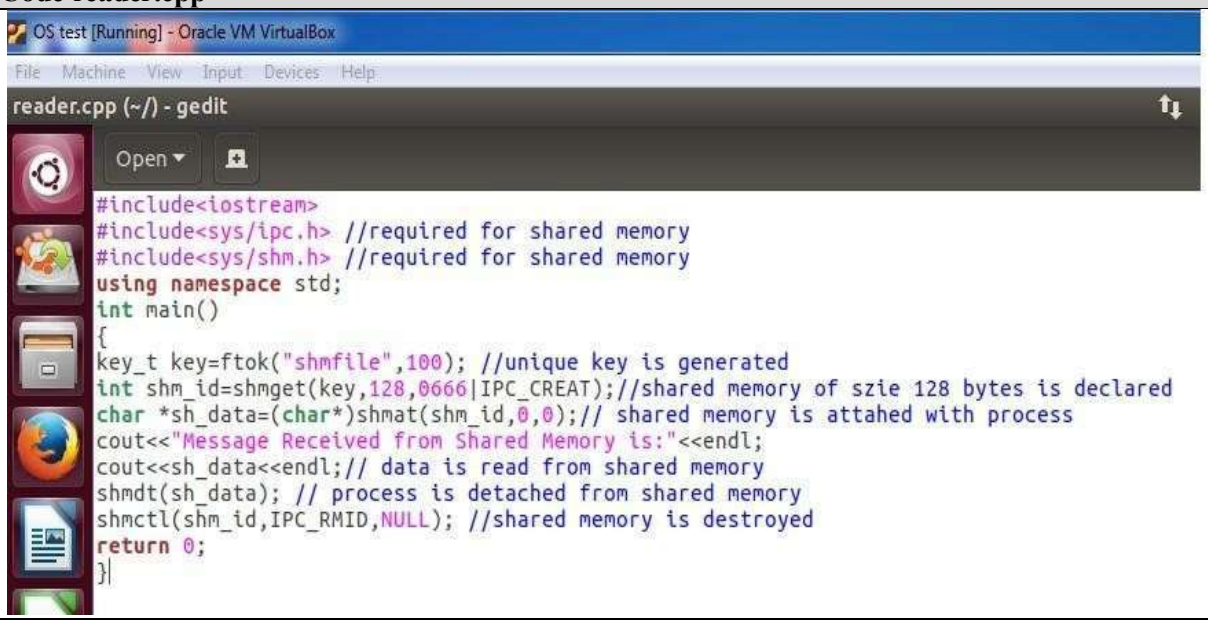
Activity 1:

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.

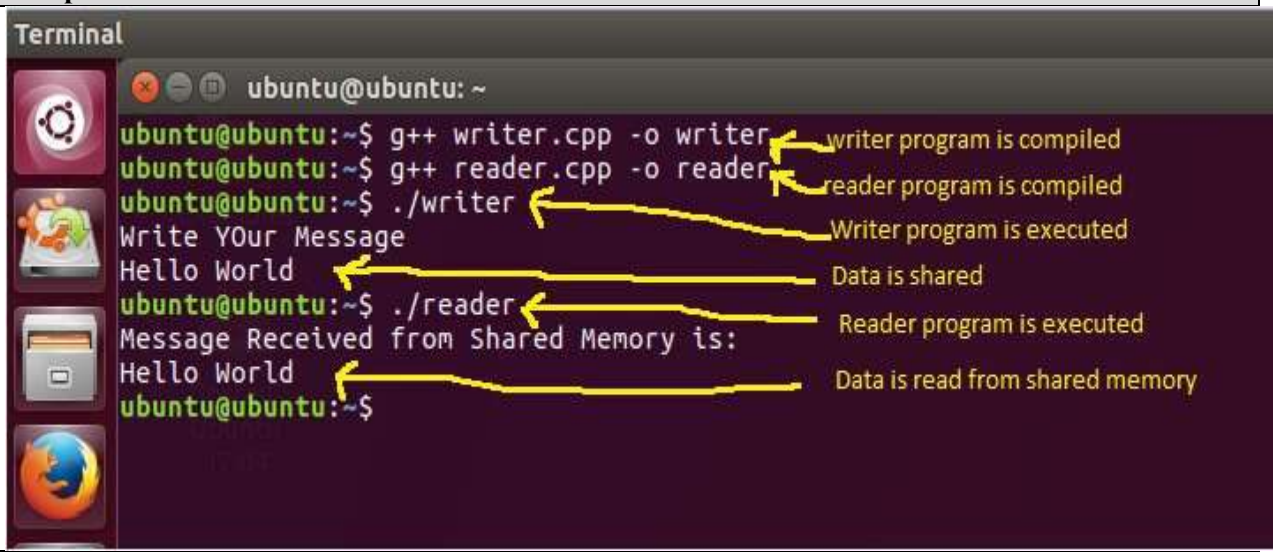
Code writer.cpp



Code reader.cpp



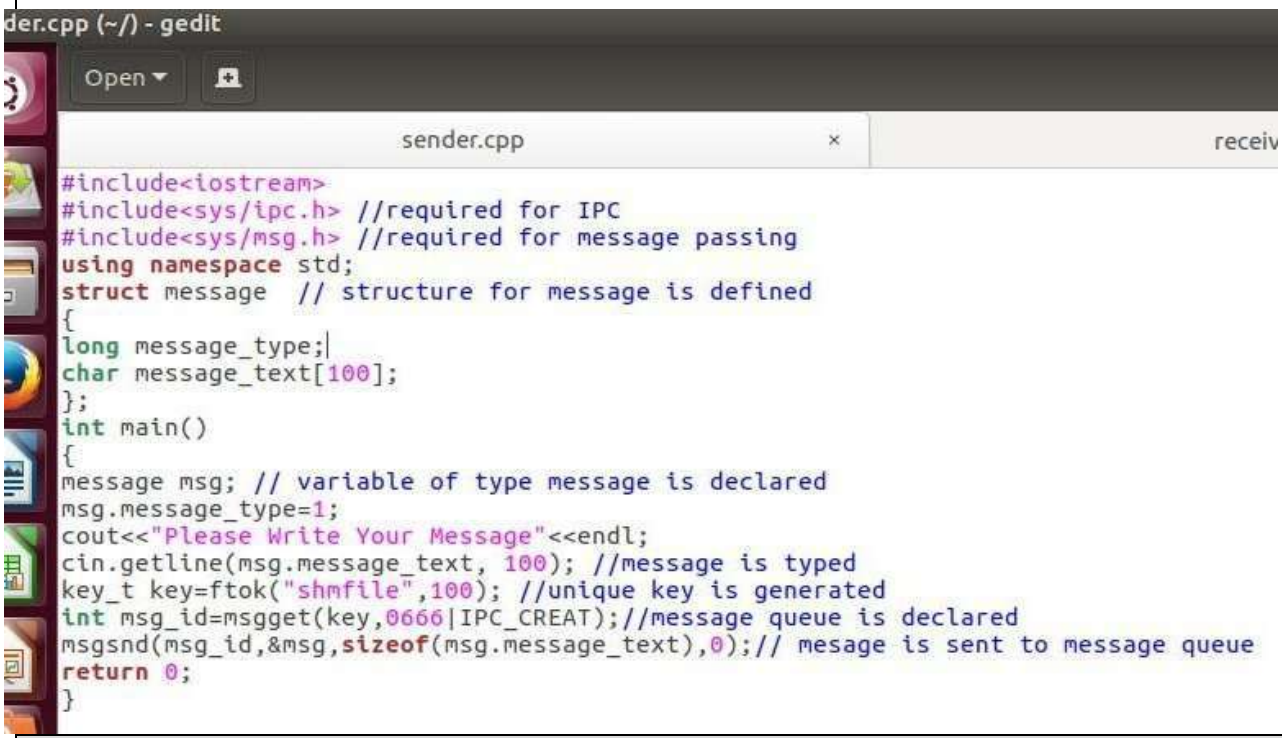
Out-put



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



Code receiver.cpp

receiver.cpp (~/) - gedit

Open

```
#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



3) Graded Lab Tasks

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.

Task 2:

Write two C++ programs using shared memory:

- The writer process takes two integers from user and stores them in shared memory.
- The reader process reads those integers, calculates their sum, and displays the result.

Task 3:

Write two C++ programs using message queue:

- The sender sends a single message: "Hello"
- The receiver receives and displays the message
- Then delete the message queue

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation

LABORATORY SKILLS ASSESSMENT (Psychomotor)
Total Marks: 100

(Max Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S< 70%	Level 3 70% ≤ S< G0%	Level 4 G0%≤ S ≤100%	Score (S)
Procedural Awareness (20)	Selects inappropriate Linux commands, shell scripting techniques, or process management methods.	Selects and applies partially appropriate Linux commands and techniques	Selects and applies considerably appropriate Linux commands and techniques.	Selects and applies completely appropriate Linux commands and techniques	
Practical Implementation (20)	Makes major critical errors in executing Linux commands, scripting, and system processes.	Makes numerous critical errors in executing commands and process management.	Makes minor non-critical errors in executing Linux commands and system operations.	Executes Linux commands and manages processes correctly with no errors.	
Process Management and Shell Scripting (20)	Program logic contains major errors with incorrect or contradictory script flow.	Program logic has some errors with occasional contradictions in process execution.	Program logic is mostly correct but may contain occasional redundancy or minor errors.	Program logic is completely correct with no contradictions or redundant processes.	
Syntax Correctness and Results (20)	Program does not follow proper syntax for Linux commands and shell scripting, leading to incorrect outputs	Program partially follows proper syntax, producing correct results for few inputs.	Program adequately follows proper syntax, producing correct results for most inputs.	Program fully follows proper syntax, producing accurate results for all inputs.	
Use of OS Tools (10)	Uses OS tools (like terminal, process manager) with limited competence.	Uses OS tools with some competence.	Uses OS tools with considerable competence.	Uses OS tools proficiently with a high degree of competence.	
Safety (10)	Requires constant reminders to follow system safety procedures (e.g., file permissions, process handling).	Requires some reminders to follow system safety procedures.	Follows system safety procedures with minimal reminders.	Routinely follows system safety procedures.	
				Marks Obtained	

LABORATORY SKILLS ASSESSMENT (Affective)
Total Marks: 40

Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S < 70%	Level 3 70% ≤ S < G0%	Level 4 G0% ≤ S ≤ 100%	Score (S)
Attitude s Engagement (5)	Shows little interest in lab activities; does not participate actively.	Participates occasionally but lacks enthusiasm and consistency.	Engages actively in most lab activities with interest.	Highly motivated, participates enthusiastically, and shows a proactive approach	
Responsibility s Punctuality (5)	Frequently misses deadlines and is often late to lab sessions.	Occasionally late or misses deadlines but tries to catch up.	Submits work on time and attends lab sessions regularly.	Always punctual, meets deadlines, and takes full responsibility for assigned tasks.	
Collaboration s Teamwork (10)	Rarely collaborates, struggles to work in a team, and does not contribute effectively.	Works with team members occasionally but struggles with communication.	Cooperates well, contributes effectively, and maintains professional interactions.	Actively engages in teamwork, supports peers, and demonstrates excellent collaboration.	
Communicatio n s Presentation Skills (10)	Struggles to explain concepts, unclear verbal/written communication.	Communicates ideas with some clarity but lacks confidence or coherence.	Presents ideas effectively with minor issues in clarity or structure.	Communicates clearly, confidently, and effectively in all aspects of lab work.	
Report Quality (10)	Report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains some grammatical errors.	Report is well organized and cohesive and contains no grammatical errors. Presentation seems polished.	
				Marks Obtained	