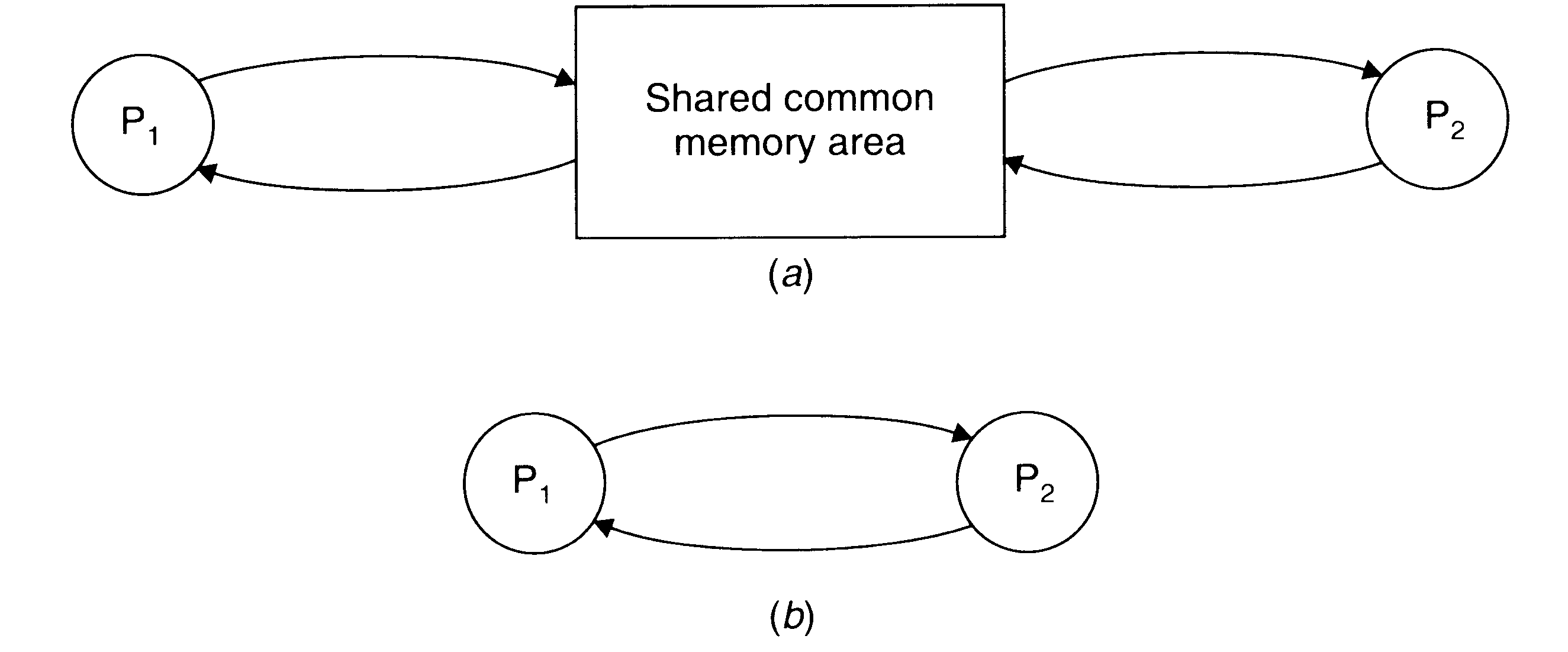
**Practical No. 2**

**Aim:** To implement message passing.

**Introduction:**

Inter process communication (IPC) basically requires information sharing among two or more processes. Two basic methods for information sharing are as follows:

* Original sharing, or shared-data approach;
* Copy sharing, or message-passing approach.

Two basic inter process communication paradigms: the shared data approach and message passing approach. In the shared-data approach, the information to be shared is placed in a common memory area that is accessible to all processes involved in an IPC. In the message-passing approach, the information to be shared is physically copied from the sender process’s space to the address space of all the receiver processes, and this is done by transmitting the data to be copied in the form of messages (message is a block of information).

**Message-passing system**: It is a subsystem of distributed operating system that provides a set of message-based IPC protocols, and does so by shielding the details of complex network protocols and multiple heterogeneous platforms from programmers. It enables processes to communicate by exchanging messages and allows programs to be written by using simple communication primitives, such as send and receive. In computer science, message passing sends a message to a process and relies on the process and the supporting infrastructure to select and invoke the actual code to run. Message passing differs from conventional programming where a process, subroutine, or function is directly invoked by name. Message passing is key to some models of concurrency and object oriented programming. Message passing is used ubiquitously in modern computer software. It is used as a way for the objects that make up a program to work with each other and as a means for objects and systems running on different computers (e.g., the Internet) to interact. Message passing may be implemented by various mechanisms, including channels.

Message passing is a technique for invoking behavior (i.e., running a program) on a computer. In contrast to the traditional technique of calling a program by name, message passing uses an object model to distinguish the general function from the specific implementations. The invoking program sends a message and relies on the object to select and execute the appropriate code. The justifications for using an intermediate layer essentially falls into two categories: encapsulation and distribution

**Implementation:**

1. Write the client and server .java files.
2. Place the client file on the client system and the server file on the server system.
3. Note the server’s IP address and enter the same in the client.java file to facilitate remote connection to the server.
4. Compile both of the files.
5. Run the server.java file on server system first, and then run the client.java file on the client system. Make sure both of the systems are connected to the internet.
6. Now the user can enter the name of the file needed, on the client system, which is then sent to the server system for checking if it exists.
7. The server sends the requested file to the client if it exists.

**Program:**

1. **Client**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class GGFileClient {public static void main(String srgs[])throws IOException

{

Socket s=null;

BufferedReader get=null;

PrintWriter put=null;

try

{

s=new Socket("192.168.1.6",8081);

get=new BufferedReader(new InputStreamReader(s.getInputStream()));

put=new PrintWriter(s.getOutputStream(),true);

}

catch(Exception e)

{

System.exit(0);

}

InputStreamReader get2=new InputStreamReader(s.getInputStream());

String u,f;

System.out.println("Enter the file name to transfer from server:");

DataInputStream dis=new DataInputStream(System.in);

f=dis.readLine();

put.println(f);

File f1=new File(f);

FileOutputStream fs=new FileOutputStream(f1);

BufferedInputStream d=new BufferedInputStream(s.getInputStream());

BufferedOutputStream outStream = new BufferedOutputStream(new FileOutputStream(f1));

byte buffer[] = new byte[1024];

int read;

while((read = d.read(buffer))!=-1)

{

outStream.write(buffer, 0, read);

outStream.flush();

}

fs.close();

System.out.println("File received");

s.close();

}

}

1. **Server**

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class GGFileServer {

public static void main(String args[])throws IOException

{

ServerSocket ss=null;

try

{

ss=new ServerSocket(8081);

}

catch(IOException e)

{

System.out.println("couldn't listen");

System.exit(0);

}

Socket cs=null;

try

{

cs=ss.accept();

System.out.println("Connection established"+cs);

}

catch(Exception e)

{

System.out.println("Accept failed");

System.exit(1);

}

PrintWriter put=new PrintWriter(cs.getOutputStream(),true);

BufferedReader st=new BufferedReader(new InputStreamReader(cs.getInputStream()));

String s=st.readLine();

System.out.println("The requested file is : "+s);

File f=new File(s);

if(f.exists())

{

BufferedInputStream d=new BufferedInputStream(new FileInputStream(s));

BufferedOutputStream outStream = new BufferedOutputStream(cs.getOutputStream());

byte buffer[] = new byte[1024];

int read;

while((read = d.read(buffer))!=-1)

{

outStream.write(buffer, 0, read);

outStream.flush();

}

d.close();

System.out.println("File transfered");

cs.close();

ss.close();

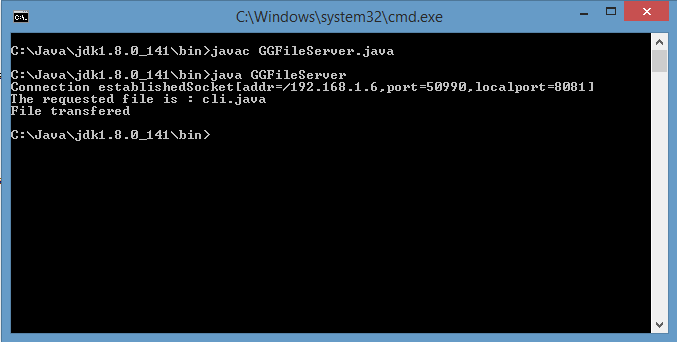
}

}

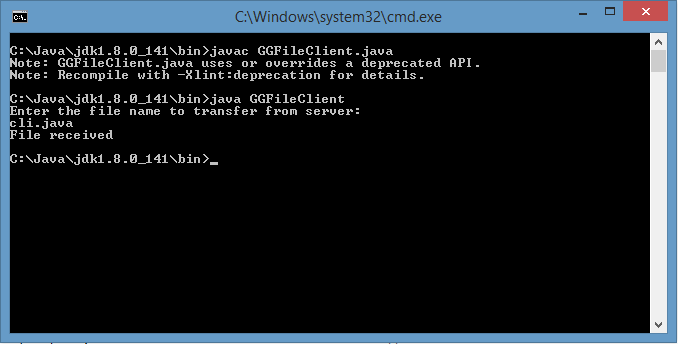
}

**Output:**

**Server**

****

**Client**

****

**Conclusion:** Message passing is a form of communication used in parallel programming and object-oriented programming. Communications are completed by the sending of **messages** (functions, signals and data packets) to recipients. Message passing simply means that (at a very abstract level) the fundamental mechanism of program execution is objects sending each other message. The important point is that the name and structure of these messages is not necessarily fixed beforehand in the source code and can itself be additional information.