**EXP NO :7 INTERPROCESS COMMUNICATION**

1. **PIPES**

**AIM:**

To write a C program to communicate over two process using pipes in inter process communication.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Declare two pipes pipefds1[2], pipefds1[2] and two returnstatuses returnstatus1, returnstatus2 for checking and process id(pid) are of integer data type and also declare write and read messages of pipe as char data type.

**STEP 3:** Create pipe1 for parent process to write and child process to read and create pipe2 for child process to write and parent process to read through pipe () system call and assign it to return status.

**STEP 4:** Check the condition for two pipes that if return status is -1 then it is unable to create pipes.

**STEP 5:** Then the fork () system call is assigned to pid and it returns the value to pid whether the child process is created or not.

**STEP 6:** If the pid is not equal to zero then the parent process executes after closing the unwanted pipe of pipe1 and pipe2 and write and read messages are done by write and read system call.

**STEP 7:** After the parent process executes, due to fork system call child process has been executed and read, write messages has been done by read and write system call.

**STEP 8:** Thus, the communication is made between parent and child process by the concept of pipes.

**STEP 9:** Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<unistd.h>

int main ()

{

int pipefds1[2], pipefds2[2];

int returnstatus1, returnstatus2;

int pid;

char pipe1writemessage[20] = "GCT";

char pipe2writemessage[20] = "CSE";

char readmessage[20];

returnstatus1 = pipe(pipefds1);

if (returnstatus1 == -1)

{

printf("Unable to create pipe 1 \n");

return 1;

}

returnstatus2 = pipe(pipefds2);

if (returnstatus2 == -1)

{

printf("Unable to create pipe 2 \n");

return 1;

}

pid = fork();

if (pid != 0)

{

close(pipefds1[0]);

close(pipefds2[1]);

printf("In Parent: Writing to pipe 1 – Message is %s\n", pipe1writemessage));

read(pipefds2[0], readmessage, sizeof(readmessage));

printf("In Parent: Reading from pipe 2 – Message is %s\n", readmessage);

}

else

{

close(pipefds1[1]);

close(pipefds2[0]);

read(pipefds1[0], readmessage, sizeof(readmessage));

printf("In Child: Reading from pipe 1 – Message is %s\n", readmessage);

printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);

write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));

}

return 0;

}

**OUTPUT:**

In Parent: Writing to pipe 1 -Message is GCT

In Child: Reading from pipe 1 – Message is GCT

In Child: Writing to pipe 2 -Message is CSE

In Parent: Reading from pipe 2- Message is CSE

**RESULT:**

Thus, the C program on implementation of inter process communication through pipes has been executed and the required output has been obtained.

**b) SOCKETS**

**(1) Server-side socket program:**

**AIM:**

To write a C program to communicate over two process (server and client) using sockets in inter process communication.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Declare the two file descriptors server\_fd, new\_socket and valread, addlen are of integer data type and sockaddr\_in address are of struct data type.

**STEP 3:** Here the arguments are passedas command line arguments in server and client program. In server, port number is sent as argument.

**STEP 4:** Socket is created using socket system call with parameters domain, type, protocol. Then the respective system calls setsockopt, bind, listen, accept are made to make a connection from the server.

**STEP 5:** The mentioned system calls are given under if condition to check whether the system call is not working. If yes, then it throws an error and exits the program. If not, the system calls have been executed to create a communication to the client.

**STEP 6:** If the system calls have no error in it, then the connection is successfully made through send and read system calls.

**STEP 7:** Once the server accepts the connection, server and client starts communicating with each other and the message has been sent and received.

**STEP 8:** Stop the program.

**PROGRAM:**

**Server-side socket program:**

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

int main(int argc, char const \*argv[])

{

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = {0};

char \*hello = "Hello from server”;

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEADDR | SO\_REUSEPORT,

&opt, sizeof(opt)))

{

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons( PORT );

if (bind(server\_fd, (struct sockaddr \*)&address,

sizeof(address))<0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0)

{

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address,

(socklen\_t\*)&addrlen))<0)

{

perror("accept");

exit(EXIT\_FAILURE);

}

valread = read( new\_socket , buffer, 1024);

printf("%s\n",buffer );

send(new\_socket , hello , strlen(hello) , 0 );

printf("Hello message sent\n");

return 0;

}

**OUTPUT:**

Server side:

Hello from client

Server: Hello message sent

**(2) Client-side socket programming:**

**AIM:**

To write a C program to communicate over two process (server and client) using sockets in inter process communication.

**ALGORITHM:**

**STEP 1:** Start the program.

**STEP 2:** Declare the valread, sock are of integer data type and sockaddr\_in serv\_addr are of structure data type and buffer is of character data type and initialise the sock to zero.

**STEP 3:** Here the arguments are passedas command line arguments in server and client program. In client, name of the port and respective port number is sent as argument.

**STEP 4:** Socket is created using socket system call with parameters domain, type, protocol. Then the respective system calls socket, connect are made to make a connection from the client.

**STEP 5:** The mentioned system calls are given under if condition to check whether the system call is not working. If yes, then it throws an error and exits the program. If not, the system calls have been executed to create a communication to the server.

**STEP 6:** The address is also converted from text to binary form. If the system calls have no error in it, then the connection is successfully made.

**STEP 7:** Once the server accepts the connection, server and client starts communicating with each other and the message has been sent and received using send and read system calls.

**STEP 8:** Stop the program

**PROGRAM:**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

int main(int argc, char const \*argv[])

{

int sock = 0, valread;

struct sockaddr\_in serv\_addr;

char \*hello = "Hello from client";

char buffer[1024] = {0};

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

{

printf("\nInvalid address not supported \n");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

{

printf("\nConnection Failed \n");

return -1;

}

send(sock , hello , strlen(hello) , 0 );

printf("Hello message sent\n");

valread = read( sock , buffer, 1024);

printf("%s\n",buffer );

return 0;

}

**OUTPUT:**

Client side:

Client: Hello message sent

Hello from server

**RESULT:**

Thus, the C program on the implementation of inter process communication through sockets has been executed and obtained the required result.