**OPERATING**

**SYSTEM**

**LAB – 11**

**PIPE INTER PROCESS COMMUNICATION**

**Submitted by: ~**

**Ayushi Kumari**

**RA1911003010865.**

**Objective:**

Inter process communication between the processes using pipe concept.

1. **Uni directional pipe:**

**Algorithm**

Step 1 − Create a pipe using pipe() system call.

Step 2 − Send a message to the one end of the pipe.

Step 3 − Retrieve the message from the other end of the pipe and write it to the standard output.

**Expected Output:**

hello, world #1

hello, world #2

hello, world #3

**Code: -**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#define size 16

char\*msg1 = "hello world 1";

char\*msg2 = "hello world 2";

char\*msg3 = "hello world 3";

int main(){

char inbuf[size];

int p[2],i;

if(pipe(p)<0){

exit(1);

}

write(p[1],msg1,size);

write(p[1],msg2,size);

write(p[1],msg3,size);

for(i=0;i<3;i++){

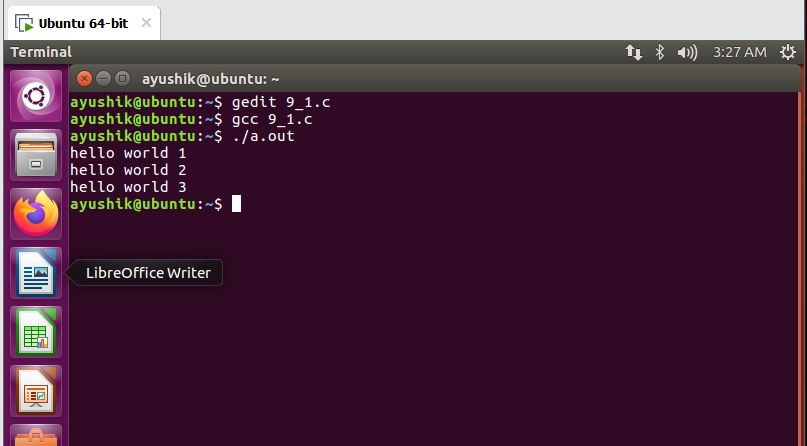
read(p[0] , inbuf , size);

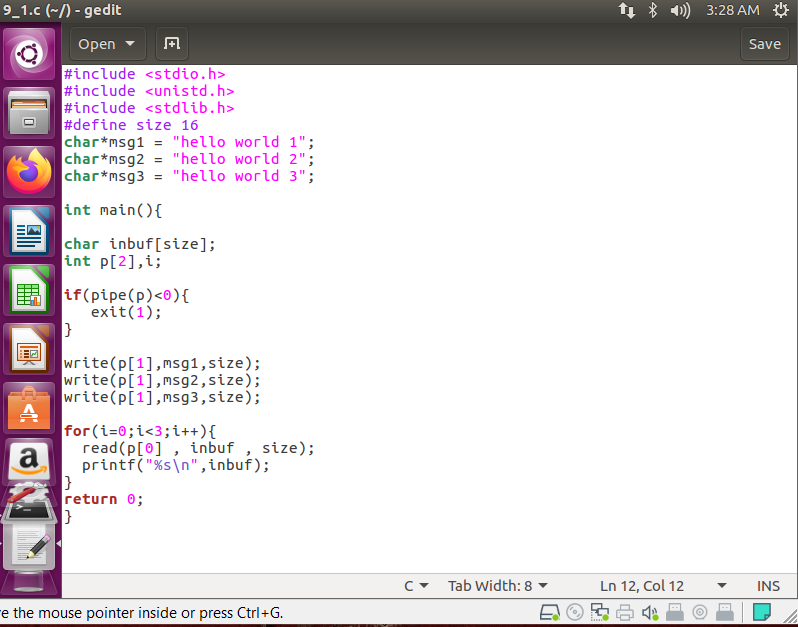
printf("%s\n",inbuf);

}

return 0;

}





1. **Write and read two messages using pipe.**

**Algorithm**

Step 1 − Create a pipe.

Step 2 − Send a message to the pipe.

Step 3 − Retrieve the message from the pipe and write it to the standard output.

Step 4 − Send another message to the pipe.

Step 5 − Retrieve the message from the pipe and write it to the standard output.

Note − Retrieving messages can also be done after sending all messages.

**Expected Output:**

Writing to pipe - Message 1 is Hi

Reading from pipe – Message 1 is Hi

Writing to pipe - Message 2 is Hello

Reading from pipe – Message 2 is Hello

**Code: -**

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds[2];

int returnstatus;

char writemessages[2][20]={"Hi", "Hello"};

char readmessage[20];

returnstatus=pipe(pipefds);

if(returnstatus==-1)

{

printf("unable to create pipe\n");

return 1;

}

printf("Writing to pipe - Message 1 is %s\n",writemessages[0]);

write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

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

printf("Reading from pipe - Message 1 is %s\n", readmessage);

printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);

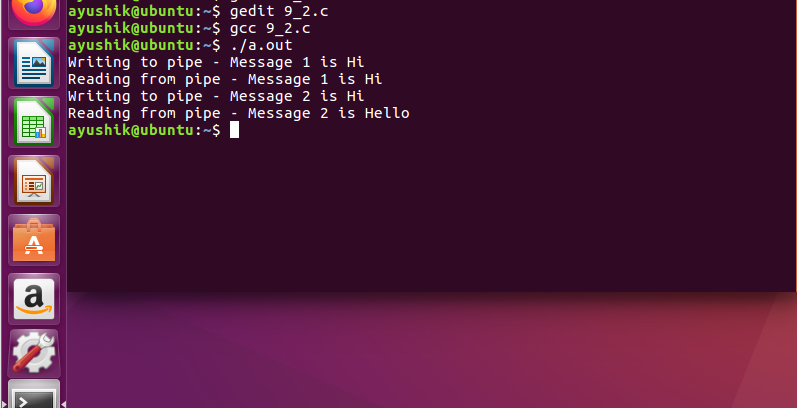
write(pipefds[1], writemessages[1], sizeof(writemessages[0]));

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

printf("Reading from pipe - Message 2 is %s\n", readmessage);

return 0;

}



1. **Program to write and read two messages through the pipe using the parent and the child processes.**

**Algorithm**

Step 1 − Create a pipe.

Step 2 − Create a child process.

Step 3 − Parent process writes to the pipe.

Step 4 − Child process retrieves the message from the pipe and writes it to the standard output.

Step 5 − Repeat step 3 and step 4 once again.

**Expected Output:**

Parent Process - Writing to pipe - Message 1 is Hi

Parent Process - Writing to pipe - Message 2 is Hello

Child Process - Reading from pipe – Message 1 is Hi

Child Process - Reading from pipe – Message 2 is Hello

**Code: -**

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds1[2], pipefds2[2];

int returnstatus1, returnstatus2;

int pid;

char pipe1writemessage[20] = "Hi";

char pipe2writemessage[20] = "Hello";

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);

write(pipefds1[1], pipe1writemessage, sizeof(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);

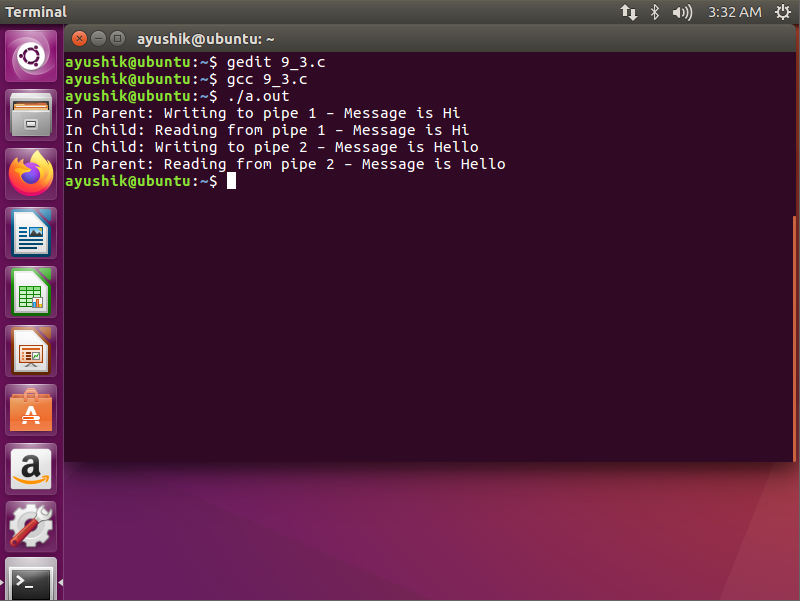
printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);

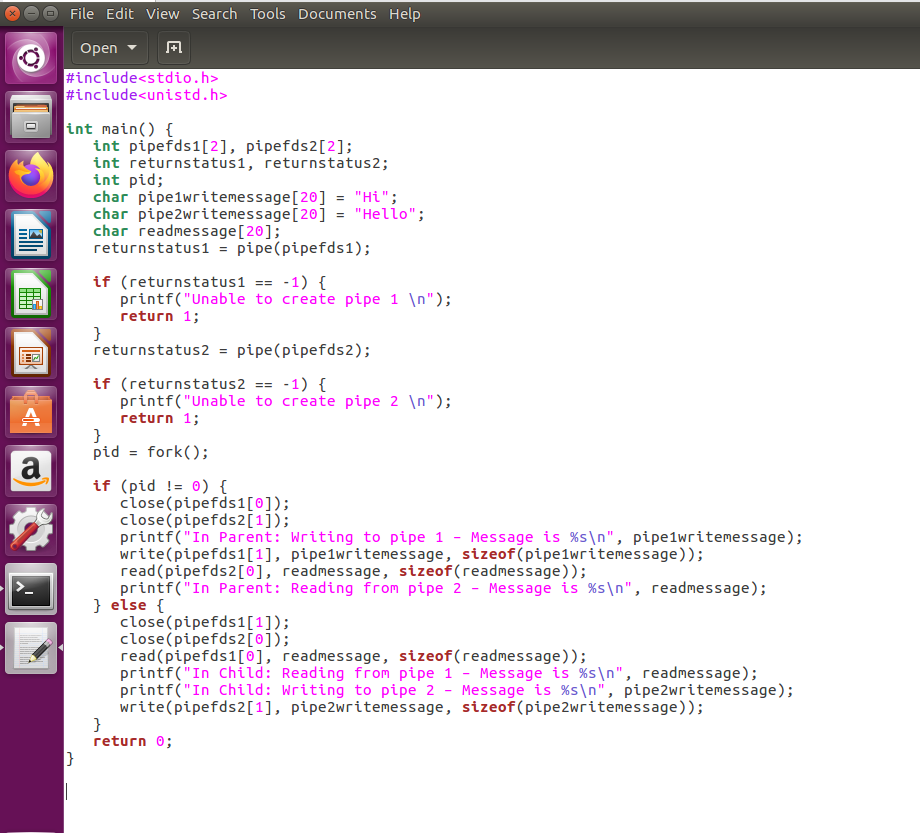
write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));

}

return 0;

}





1. **Write the output of the following program.**

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

int p[2];

char buff[25];

if(fork()==0)

{

printf(“Child : Writing to pipe \n”);

write(p[1],”Welcome”,8);

printf(“Child Exiting\n”);

}

else

{

wait(NULL);

printf(“Parent : Reading from pipe \n”);

read(p[0],buff,8);

printf(“Pipe content is : %s \n”,buff);

}

return 0;

}

=

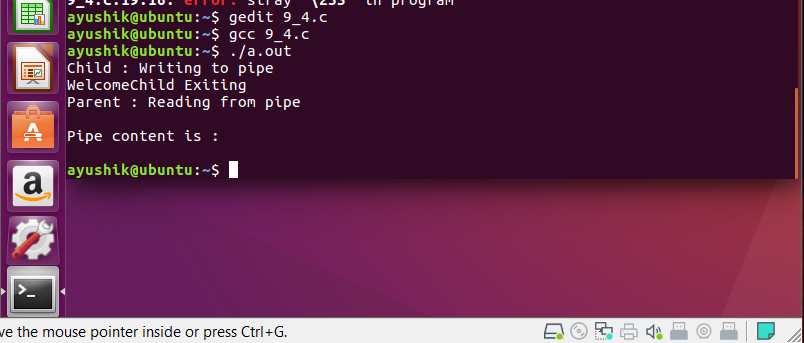
Output-

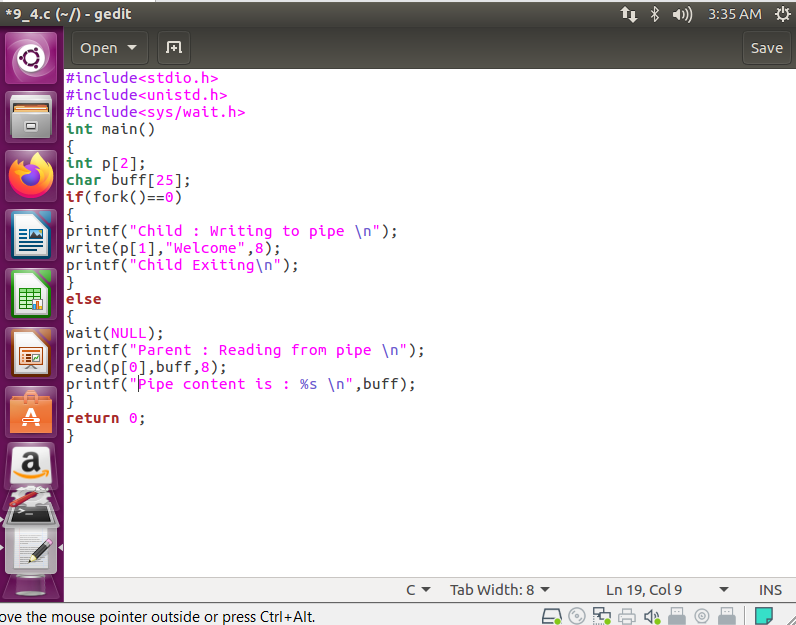
Child : Writing to pipe

Child Exiting

Parent : Reading from pipe

Pipe content is : ��\_





**Outcome:**

Interprocess communication using pipes concept learned and implemented.