**PIPE:**

1. It is a communication between two or more related or interlated process.

2. Communication can be achieved by one process writing into pipe and oher reading from the pipe.

3. To achieve a pipe system call, create two file, one to write into file and other to read from the file.

*#include<unistd.h>*

*int pipe(int pipedes[2]);*

The above system call create a pipe for one way communication, it create two file descriptor, first one is connected to read from pipe and other one is connected to write from pipe.

Descriptor pipedes[0] -> For reading

Descriptor pipedes[1] -> For witing

These descriptor calls would return 0 on success, -1 on failure.

*#include<sys/types.h>*

*#incllude<sys/stat.h>*

*#include<fcntl.h>*

*int open(const char\* pathName, int flags);*

*int open(const char\* pathName, int flags, mode\_t mode);*

Note:

It is essential to open the file before performing any operation on it and closing the file after completing the required operation.

By default there are three descriptor opened for every process,

standard input - 0

standard output - 1

standard error - 2

The arguments passed to open system call are pathname, flag (purpose of opening the file like O\_RDONLY for read, O\_WRONLY for write, O\_RDWR to read and write, O\_APPEND to append to existing data, O\_CREAT to create if not exist) and mode provides permission of read/ write/ execute for user or owner/ group/ others.

*#include<unistd.h>*

*int close(int fd);*

The above system call closes the file descriptor that has been opened, means the file associated is no longer in use and resource can be reused by other process. This system call return 0 on success and -1 on failure.

*#include<unistd.h>*

*size\_t read(int fd, void \*buf, size\_t count);*

The above system call is to read from the specified file with arguments of file descriptor, proper buffer with allocated memory(either static and dynamic) and the size of buffer.

Why file descriptor?

To identify the respctive file, which is returned after calling open or pipe system call.

The file needs to be opened before reading it.

This call (read) would return the number of bytes read or 0 on success and -1 in case of failure.

*#include<unistd.h>*

*ssize\_t write(int fd, void \*buf, size\_t count);*

The above call is to write to a specific file with the file descriptor as arguments and buffer size to allocate the memory and size of the buffer.

This call would return the number of bytes written (or zero if nothing is written) and -1 incase of failure.

Pgm 1:

#include<stdio.h>

#include<unistd.h>

int main()

{

int pipefds[2];

// p[0] = read

// p[1] = write

//To check the pipe return value

int returnstatus;

int pid;

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

//Buffer

char readmessage[20];

returnstatus = pipe(pipefds);

if(returnstatus == -1){

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

return 1;

}

//To get the fork return value

pid = fork();

if(pid == 0) {

//Child process

//read(file descriptor, buffer, sizeof(buffer))

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

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

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

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

} else {

//Parent process

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

//write(file description, buffer, sizeof(buffer))

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

printf("Parent Process - Writing from pipe - Message 2 is %s\n", writemessages[1]);

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

}

return 0;

}

Pgm 2:

#include<stdio.h>

#include<unistd.h>

int main()

{

int pipefds[2];

// p[0] = read

// pipe[1] = write

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(file descriptor, buffer size, size of buffer)

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

//read(file descriptor, buffer size, size of buffer)

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[1]);

//write(file descriptor, buffer size, size of buffer)

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

//read(file descriptor, buffer size, size of buffer)

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

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

return 0;

}

Pgm3:

#include<stdio.h>

#include<unistd.h>

int main()

{

int pipefds1[2], pipefds2[2];

int returnstatus1, returnstatus2;

int pid;

char pipe1writemessage[20] = "HiUniverse!";

char pipe2writemessage[20] = "HelloWorld!";

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){ //Parent process

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 parebnt: 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;

}