Q. Program to send a message from parent process to child process.

Note: To create a simple pipe with C, we make use of the pipe() system call. It takes a single argument, which is an array of two integers.

fd[0] is set up for reading, fd[1] is set up for writing.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main()

{

      int fd[2],n;

      char buffer[100];

      pid\_t p;

      pipe(fd);

      p=fork();

      if(p>0)  //parent

      {

                  close(fd[0]);

                  printf("Passing value to child\n");

                  write(fd[1],"hello\n",6);

                  wait();

      }

      else // child

      {

                  close(fd[1]);

                  n=read(fd[0],buffer,100);

                  write(1,buffer,n);

      }

}

Program for inter-process communication using popen and pclose

//This is the structure stored in 'stdio.h' under the name FILE.This file pointer is just to store the composite information about a file subject to manipulation. C has a special "data type" for handling files which is defined in the standard library 'stdio.h '.  
It is called the file pointer and has the syntax FILE\*.

Q. Program to write into a pipe  
   
#include<stdio.h>   
#include<stdlib.h>   
#include<unistd.h>   
#include<string.h>   
int main()

{

       FILE \*rd;

       char buffer[50];

       sprintf(buffer,"name first");

/\*The C library function int **sprintf**(char \*str, const char \*format, ...) sends formatted output to a string pointed to, by str. ... int **sprintf**(char \*str, const char \*format, ...) str − This is the pointer to an array of char elements where the resulting C string is stored. \*/

       rd=popen("wc -c","w"); // wc -c -> is the process which counts the number of characters

                                              //passed. 2nd parameter is "w" which means pipe is opened in writing

                                             //mode

       fwrite(buffer,sizeof(char),strlen(buffer),rd); // to write the data into the pipe

/\* The second argument of **fwrite**() is the size of each object, and the third argument is ...**fwrite**(**buffer**, **sizeof**(**char**), sizeof(**buffer**), file**);**. is a simple .... functions, such as **strlen** , strcpy . printf("%s", **buffer); \*/**

      pclose(rd);

}  
  
 Q. Program to read from a pipe  
  
#include<stdio.h>   
#include<stdlib.h>   
#include<unistd.h>   
#include<string.h>   
int main()

{

       FILE \*rd;

       char buffer[50];

       rd=popen("ls","r");

       fread(buffer, 1, 50, rd);

       printf("%s\n", buffer);

      pclose(rd);

}

Program for inter process communication using named pipes

In [computing](https://en.wikipedia.org/wiki/Computing), a **named pipe** (also known as a [**FIFO**](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)) for its behavior) is an extension to the traditional [pipe](https://en.wikipedia.org/wiki/Pipeline_(Unix)) concept on [Unix](https://en.wikipedia.org/wiki/Unix) and [Unix-like](https://en.wikipedia.org/wiki/Unix-like) systems, and is one of the methods of [inter-process communication](https://en.wikipedia.org/wiki/Inter-process_communication) (IPC). The concept is also found in [OS/2](https://en.wikipedia.org/wiki/OS/2) and [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), although the semantics differ substantially. A traditional pipe is "[unnamed](https://en.wikipedia.org/wiki/Anonymous_pipe)" and lasts only as long as the process. A named pipe, however, can last as long as the system is up, beyond the life of the process. It can be deleted if no longer used. Usually a named pipe appears as a file, and generally processes attach to it for [inter-process communication](https://en.wikipedia.org/wiki/Inter-process_communication).

This will require three different programs to work

**Program 1: Creating fifo/named pipe ( 1.c )**

            #include<unistd.h>

            #include<stdlib.h>

            #include<stdio.h>

            int main()

            {

                        int res;

**res = mkfifo("fifo1",0777**); //creates a named pipe with the name fifo1

                        printf("named pipe created\n");

            }

Now compile and run this program.

**Program 2: Writing to a fifo/named pipe ( 2.c )**

            #include<stdlib.h>

            #include<stdio.h>

            #include<fcntl.h>

            int main()

            {

                        int res,n;

**res=open("fifo1",O\_WRONLY);**

**write(res,"written",7);**

                        printf("Process %d finished\n",getpid());

            }

 Compile this program as

     gcc -o 2 2.c

Note: If you run this you will not see any output

**Program 3: Reading from the named pipe ( 3.c )**

            #include<stdlib.h>

            #include<stdio.h>

            #include<fcntl.h>

            int main()

            {

                        int res,n;

                        char buffer[100];

**res=open("fifo1",O\_RDONLY);**

**n=read(res,buffer,100);**

                        printf("Total bytes read: %d\n",n);

                        printf("%s was passed to me \n",buffer);

                        printf("Process %d finished\n",getpid());

            }

Compile the program as  
     gcc  -o  3  3.c  
  
Now run both the object files simultaneously as  
    ./2  &  ./3

# IPC:Shared Memory

**Shared Memory** is an efficeint means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access.

A process creates a shared memory segment using shmget()

Once created, a shared segment can be attached to a process address space using shmat(). It can be detached using shmdt()

1. Program 1: This program creates a shared memory segment, attaches itself to it and then writes some content into the shared memory segment.

* shm\_id = shmget(
* key\_t k, /\* the key for the segment \*/
* int size, /\* the size of the segment \*/
* int flag); /\* create/use flag \*/
* In the above definition, **k** is of type **key\_t** or **IPC\_PRIVATE**. It is the numeric key to be assigned to the returned shared memory segment. **size** is the size of the requested shared memory. The purpose of **flag**is to specify the way that the shared memory will be used

#### NAME

shmat - shared memory attach operation

#### SYNOPSIS

#include <[sys/shm.h](http://pubs.opengroup.org/onlinepubs/7908799/xsh/sysshm.h.html)>

void \*shmat(int *shmid*, const void \**shmaddr*, int *shmflg*);

#### DESCRIPTION

The *shmat()* function attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the address space of the calling process. The segment is attached at the address specified by one of the following criteria:

* If *shmaddr* is a null pointer, the segment is attached at the first available address as selected by the system.
* If *shmaddr* is not a null pointer and (*shmflg*&SHM\_RND) is non-zero, the segment is attached at the address given by (*shmaddr*-((*uintptr\_t*)*shmaddr*%SHMLBA)) The character % is the C-language remainder operator.
* If *shmaddr* is not a null pointer and (*shmflg*&SHM\_RND) is 0, the segment is attached at the address given by *shmaddr*.
* The segment is attached for reading if (*shmflg*&SHM\_RDONLY) is non-zero and the calling process has read permission; otherwise, if it is 0 and the calling process has read and write permission, the segment is attached for reading and writing.

#include<unistd.h>  
#include<stdlib.h>  
#include<stdio.h>  
#include<string.h>  
#include<sys/shm.h>  
int main()  
{  
    int i;  
    void \*shared\_memory;  
    char buff[100];  
    int shmid;  
    shmid=shmget((key\_t)2345, 1000,0666|IPC\_CREAT); \\creates shared memory segment \\**IPC\_CREAT | 0666** for a server (*i.e.*, creating and granting read and write access to the server)

    printf("Key of shared memory is %d\n",shmid);  
    shared\_memory=shmat(shmid,NULL,0); \\process attached to shared memory segment  
    printf("Process attached at %X\n",(int)shared\_memory);  
    printf("Enter some data to write to shared memory\n");  
    read(0,buff,100); \\get some input from user  
    strcpy(shared\_memory,buff); \\data written to shared memory  
    printf("You wrote : %s\n",shared\_memory);  
}  
  
  
Program 2: This program attaches itself to the shared memory segment created in Program 1 and then reads whatever was written in the shared memory via Program 1  
  
#include<unistd.h>  
#include<stdlib.h>  
#include<stdio.h>  
#include<string.h>  
#include<sys/shm.h>  
int main()  
{  
    int i;  
    void \*shared\_memory;  
    char buff[100];  
    int shmid;  
    shmid=shmget((key\_t)2345, 1000,0666);  
    printf("Key of shared memory is %d\n",shmid);  
    shared\_memory=shmat(shmid,NULL,0); \\process attached to shared memory segment  
    printf("Process attached at %X\n",(int)shared\_memory);  
    printf("Data read from shared memory is : %s\n",shared\_memory);  
}