```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
main()
{
      int  pipefd [2] = {0,0}, n;
      char buff[100] ;

#ifdef OPEN
      open(__FILE__ , O_RDONLY);
#endif

      if( pipe( pipefd) == -1)
      {
        printf("can not create pipe \n");
      return 1;
      }
      printf(" %d %d \n",pipefd[0] , pipefd[1]);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
      int  pipefd [2] = {0,0}, n;
      if( pipe( pipefd) == -1)
      {
        printf("can not create pipe \n");
      return 1;
      }

      char output[] = "hello world\n";
      char input[20]= "SIKANDER";

      if (write (pipefd[1],output, sizeof(output))!= sizeof(output))
      {
        printf("pipe write error \n");
      }
      if(  ( n = read ( pipefd[0] , input, sizeof (input)  ) ) <= 0 )
      {
        printf("pipe read error \n");
      }

      puts(input);
}
```

Broken pipe

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
void myhandler(int signo)
{
      printf("SIGNAL HANDLER \n");
      printf("Signal number = %d \n" , signo);
}

main()
{
      int  pipefd [2] = {0,0}, n;
      if( pipe( pipefd) == -1)
      {
        printf("can not create pipe \n");
      return 1;
      }

      signal(SIGPIPE , myhandler);

#ifdef CLOSEREAD
      close(pipefd[0]);
#endif

      if(write (pipefd[1],"hello world\n", 12)!= 12)
      {
        printf("pipe write error \n");
      }
}
```

```
Pipe parent child.c
/* CHILD WRITES AND PARENT READS */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
      int  fd [2] = {0,0}, n;
      if( pipe( fd) == -1)
      {
        printf("can not create pipe \n");
      return 1;
      }


      if(fork() == 0)
      {
            printf("CHILD WRITES \n");
      //    close(fd[0]);     //Close read

            char output[] = "hello world\n";
```

```
            write(fd[1], output , sizeof(output));
      }
      else
      {
            char input[20]= "SIKANDER";
            printf("PARENT READS \n");
      //    close(fd[1]);

      if(  ( n = read ( fd[0] , input, sizeof (input)  ) ) > 0 )
      printf("Parent read the data %s \n" , input);

      }


}
```

```
Pipe 2 way communication
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
      int  p1[2] = {0,0}, p2[2];

       pipe(p1);
       pipe(p2);


      if(fork() == 0)
      {
            printf("CHILD PROCESS \n");
            close(p1[0]);
            close(p2[1]);
            char cbuf[20] = "";

            write(p1[1], "HAI FROM CHILD" , 20);
            read(p2[0] , cbuf , 20);
            printf("child read %s \n",cbuf);
      }
      else
      {
            printf("PARENT PROCESS \n");
            close(p1[1]);
            close(p2[0]);
            char buffer[20];

            buffer[read(p1[0] , buffer , 20)] = '\0';
            printf("parent read : %s \n",buffer);
            write(p2[1] ,"BYE FROM PARENT \n", 20);
      }
      return 0;
}
```

Command $ls | wc

```
/* PARENT WRITES AND CHILD READS */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
      int  fd [2] = {0,0}, n;
      if( pipe( fd) == -1)
      {
        printf("can not create pipe \n");
      return 1;
      }


      if(fork() == 0)
      {
            close(fd[0]);
            dup2(fd[1] , 1);
            execlp("ls","ls",0);
      /*    close(fd[1]);
            dup2(fd[0] , 0);
            execlp("wc","wc",0);
      */
      }
      else
      {
      /*    close(fd[0]);
            dup2(fd[1] , 1);
            execlp("ls","ls",0); */
            close(fd[1]);
            dup2(fd[0] , 0);
            execlp("wc","wc",0);
      }
}
```

```
/* IMPLEMENT ls -l | head -4 | tail -1 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
      int  fd [2] = {0,0}, n;
      if( pipe( fd) == -1)
      {
        printf("can not create pipe \n");
      return 1;
      }
```

```
	if(fork() == 0)
	{
		close(fd[0]);
		dup2(fd[1] , 1);
		execlp("ls","ls","-l",0);
	}
	else
	{
		int fd1[2] = {0};

		if(pipe(fd1) == -1)
			printf("Cannot create 2nd pipe ");
		if(fork() == 0)
		{
			close(fd[1]);
			close(fd1[0]);

			dup2(fd[0] , 0);
			dup2(fd1[1] , 1);
			execlp("head","head","-4",0);
		}
		else{
		close(fd[0]);	close(fd[1]);
		close(fd1[1]);
		dup2(fd1[0] , 0);
		execlp("tail","tail","-1",0);
		}
	}
}
```

Named Pipe

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>

int main()
{
	int ret , fd , r;

	char buff[10];

	ret = mknod("mypipe",S_IFIFO | 0666 , 0);

	fd = open("mypipe",O_RDONLY);
	printf("fd = %d \n",fd);
	r = read(fd , buff , sizeof(buff));
	buff[r] = '\0';
	printf("content from pipe : %s " , buff);
}

#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
```

```
int main()
{
      int  fd , r;

      char buff[10] = "CRANES";


      fd = open("mypipe",O_WRONLY);
      printf("fd = %d \n",fd);
      r = write(fd , buff , strlen(buff));

      printf("Number of bytes written = %d" , r);
}
```