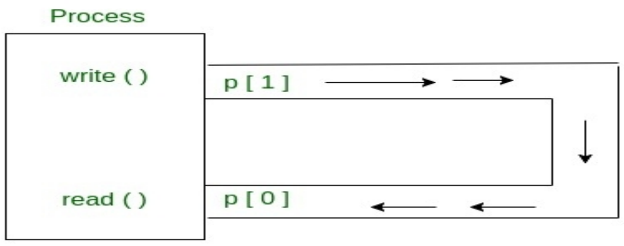
What is a pipe() system call [I/O system call]?

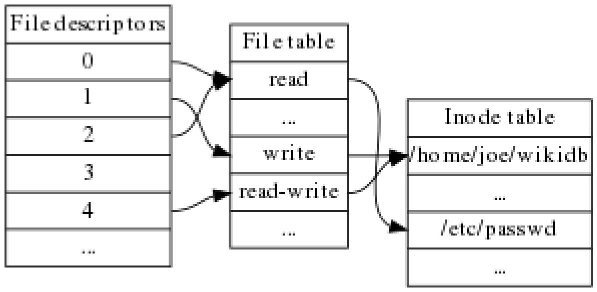
A pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In UNIX Operating System, Pipes are useful for communication between related processes (inter-process communication).

1. Pipe is one-way communication only i.e we can use a pipe such that One process write to the pipe, and the other process reads from the pipe.
2. The pipe can be used by the creating process, as well as all its child processes, for reading and writing.
3. If a process tries to read before something is written to the pipe, the process is suspended until something is written.
4. The pipe system call finds the first two available positions in the process's open file table and allocates them for the read and write ends of the pipe.



pipe(file-descriptor); int fd[2]; fd[1] → write(); fd[0] → read()

In Unix and related computer operating systems, a file descriptor (FD, less frequently fildes) is an abstract indicator (handle) used to access a file or other input/output resource, such as a pipe or network socket. File descriptors form part of the POSIX application programming interface. A file descriptor is a non-negative integer, generally represented in the C programming language as the type int (negative values being reserved to indicate "no value" or an error condition).



In modern POSIX compliant operating systems, a program that needs to access data from a file stored in a file system uses the **read system call**. The file is identified by a file descriptor that is normally obtained from a previous call to open. This system call reads in data in bytes, the number of which is specified by the caller, from the file and stores then into a buffer supplied by the calling process.

The read system call takes three arguments:

The file descriptor of the file.
the buffer where the read data is to be stored and
the number of bytes to be read from the file.

The write is one of the most basic routines provided by a Unix-like operating system kernel. It writes data from a buffer declared by the user to a given device, maybe a file. This is the primary way to output data from a program by directly using **write system call**. The destination is identified by a numeric code. The data to be written, for instance a piece of text, is defined by a pointer and a size, given in number of bytes.

write thus takes three arguments:

The file code (file descriptor or fd).
The pointer to a buffer where the data is stored (buf).
The number of bytes to write from the buffer (nbytes).

Syntax in C language: int pipe(int fd[2]);
With Parameters :
fd[0] will be the fd(file descriptor) for the read end of pipe.
fd[1] will be the fd for the write end of pipe.
Returns : 0 on Success; -1 on error.

Pipes behave FIFO (First in First out), Pipe behave like a queue data structure. Size of read and write don't have to match here. We can write 512 bytes at a time but we can read only 1 byte at a time in a pipe.

```
// C program to illustrate pipe system call in C
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";
```

```
int main(){
    char inbuf[MSGSIZE];
    int fd[2], i;
    pipe(fd);
    /* Replacing standard Input/output -> First write in pipe */
    write(fd[1], msg1, MSGSIZE);
    write(fd[1], msg2, MSGSIZE);
    write(fd[1], msg3, MSGSIZE);
    /* Then read written data from the pipe */
    for (i = 0; i < 3; i++) {
        read(fd[0], inbuf, MSGSIZE);
        printf("Retrieve from instance %d, data -> %s \n", (i+1),inbuf);
    }
    return 0;
}
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ gcc -o pipe0 pipe0.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ./pipe0
Retreive from instance 1, data -> hello, world #1
Retreive from instance 2, data -> hello, world #2
Retreive from instance 3, data -> hello, world #3
```

1. Write a code in C to implement how a process communicates with another process using the **pipe** function.

SYNOPSIS

```
    #include <unistd.h>
    [XSI] [Option Start] ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
           off_t offset); [Option End]
    ssize_t  write(int fildes, const void *buf, size_t nbyte);
    ssize_t  read(int fildes, void *buf, size_t nbyte);
```

DESCRIPTION

The write() function shall attempt to write nbyte bytes from the buffer pointed to by buf to the file associated with the open file descriptor, fildes.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <stdlib.h>
#define MSGSIZE 100
char* msg = "hello, this is LAB3113\n";
int main(){
    char inbuf[MSGSIZE]; pid_t childpid; int fd[2], i, nbytes;
    //char msg[] = "hello, this is LAB3113";
    pipe(fd); // pipe system call
    if ((childpid = fork()) == 0) { // Child creation
        close(fd[0]); // close the read-end of the pipe before writing
                printf("CHILD (%d) : preparing to write message in the pipe -> %s with length %d \
n",getpid(),msg,strlen(msg));
        write(fd[1],msg,strlen(msg)+1);
        sleep(2);
        exit(0);
    }
    else if (childpid > 0){ // PARENT
       close(fd[1]); // close the write-end of the pipe before reading
       nbytes= read(fd[0],inbuf,sizeof(inbuf));
       printf("\n Parent %d of child %d retrieves %d bytes string : %s \n ",getpid(), childpid, nbytes,inbuf);
       wait();
       return 0;    }
}
```

```
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ gcc -o pipe1 pipe1.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ./pipe1
CHILD (2488) : preparing to write message in the pipe -> hello, this is LAB3113  with length 23
 Parent 2487 of child 2488 retrieves 24 bytes string : hello, this is LAB3113
```

2. Write a code in C to implement how a process communicates with another process using the **popen** function.

```
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ls -l|tr -s ' '|cut -d ' ' -f 5,9|sort -n
```

//popen ,pclose

Function: FILE * popen (const char *command, const char *mode)

Function: int pclose (FILE *stream)

The popen function is closely related to the system function; It executes the shell command command as a subprocess. However, instead of waiting for the command to complete, it creates a pipe to the subprocess and returns a stream that corresponds to that pipe.

If you specify a mode argument of **"r"**, you can read from the stream to retrieve data from the standard output channel of the subprocess. The subprocess inherits its standard input channel from the parent process.

Similarly, if you specify a mode argument of "**w"**, you can write to the stream to send data to the standard input channel of the subprocess. The subprocess inherits its standard output channel from the parent process.

```
#include<stdlib.h>
#include <stdio.h>
int main() {
          FILE  *fdin,*fdout;
          char readbuf[80];
          if((fdin = popen("ls -l|tr -s ' '|cut -d ' ' -f 5,9","r"))==NULL)
          {printf("Pipe open rd error\n");
           exit(1);          }
          if((fdout=popen("sort -n","w"))==NULL)
           {printf("Pipe open wr error\n");
        exit(1);          }
          while(fgets(readbuf,80,fdin))
               fputs(readbuf,fdout);
           //close(fdin);
          //close(fdout);
       if (pclose (fdin) != 0)
 { fprintf (stderr,"Could not run more or other error.\n"); }
       if (pclose (fdout) != 0)
 { fprintf (stderr,"Could not run more or other error.\n"); }
         // check the system("command") function
        system("ls -l|tr -s ' '|cut -d ' ' -f 5,9|sort -n");
         return 0;    }
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ls -l|tr -s ' '|cut -d ' ' -f5,9|sort –n
429 popen2.c
662 pipe0.c
878 parentchld.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ gcc popen2.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ./a.out
429 popen2.c
662 pipe0.c
878 parentchld.c
```

3. Write a program that creates a **one-way pipe between a parent and child process**. The parent process gets a **string** from **standard input** and sends the string to the child. The **child** prints the **reverse** of the string. Both parent and child terminates when the string “**quit**” is input.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>

int flag=0; pid_t childpid;

void func(int signum) {printf("CHILD(%d) process exiting with signum = %d\
n",childpid,signum);}

int main (void)
{
  int fd[2], nbytes,i,strsize,j;
  //pid_t childpid;
  char str[40];
  char readbuffer[80], rev[80];
  pipe(fd);   //creating pipe by pipe system call
  signal(SIGCHLD, func);

  if ((childpid = fork()) == 0)
  {
     /* child process closes the input side of the pipe*/
       close(fd[1]);
       /* receive string through the input fd[0] side of pipe*/
       do {
               nbytes= read(fd[0],readbuffer,sizeof(readbuffer));
              printf("child:read %d bytes of string = %s\n",nbytes,readbuffer);
               if (strcmp(readbuffer,"quit")==0)
               exit(0);
             else flag = 1;

               i = strlen(readbuffer)-1;
               j=0;
               while (i>=0)
               {
                 rev[j]= readbuffer[i--];
```

```
                    printf("%c ",rev[j++]);
                }
                 printf("\n");
          } while (flag);  //
    }
    else if (childpid > 0)
    {
      /*Parent process closes input side of the pipe*/
       close(fd[0]);
       // read in a string from standard input
         do
        {
           printf("Parent:enter string of 30 or less char,enter quit to end:");
           scanf("%s",str);
           printf("Result of strcmp(%s, \"quit\")=%d \n",str, strcmp(str,"quit"));
           strsize=write(fd[1],str,strlen(str)+1);
                printf("Parent(%d):sent   bytes   %d   of   string   :   %s   to   child   (%d)   \n
",getpid(),strsize,str,childpid);
           sleep(2);
         } while (strcmp(str,"quit") != 0);
         return 0;
    }
} // main ended.....
```

```
nilna@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ gcc pipstdInp.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ./a.out
Parent:enter string of 30 or less char,enter quit to end:abcdefgh
Result of strcmp(abcdefgh, "quit")=-1
Parent:sent bytes 9 of string : abcdefgh
child:read 9 bytes abcdefgh
h g f e d c b a
 Parent:enter string of 30 or less char,enter quit to end:quit
Result of strcmp(quit, "quit")=0
Parent:sent bytes 5 of string : quit
child:read 5 bytes quit
```

4. Write a program that creates a two-way pipe between a parent and child process. **The parent process gets an integer from standard input and sends the integer to the child** and the **child computes the sum of all integers up to the input value received from parent and sends the result back to the parent process, which then prints it**. Both the parent and child terminates when the number 0 is input.

```
/*prob4 : integer send Two way pipes*/
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<wait.h>

int main()
{
  int flag =1;
  int data1,data2,i,cum=0; // cumulative sum
  int fd1[2];
  int fd2[2];
  char buf1[30];
  char buf2[30];
  char writedata[10];
  char readdata[10];
  int val;
  pid_t childpid;

  if(pipe(fd1)==0 && pipe(fd2)==0)
  {
    if((childpid=fork())==0)
    {   // CHILD.....
       do {
        cum =0; val=0;
        data1=read(fd1[0],buf1,sizeof(buf1));
        val= atoi(buf1);
        printf("\n CHILD (%d):: %d bytes read %s  %d.\n",getpid(),data1,buf1,val);
        if (val == 0) flag=0;
        printf("CHILD (%d):: flag = %d\n",getpid(),flag);

        for (i=1;i<=val;i++) cum+=i;
        printf("CHILD (%d):: Cum=%d\n",getpid(),cum);
        sprintf(writedata,"%d",cum);
        printf("CHILD (%d):: writedata= %s \n",getpid(),writedata);
        data2=write(fd2[1],writedata,strlen(writedata)+1);
            printf("\n    CHILD    (%d)::    Child    writes    %d    bytes    string        %s\
n",getpid(),data2,writedata);
       } while (flag !=0);
      exit(0);
  }
  else if (childpid > 0)
  {   //PARENT....
       do {
            printf("PARENT (%d):: flag=%d\n",getpid(),flag);
            printf("PARENT (%d):: Enter an integer:",getpid());
            scanf("%s",readdata);
            data2=write(fd1[1],readdata,strlen(readdata)+1);
            printf("\n PARENT (%d):: writes %d bytes %s \n",getpid(),data2,readdata);
            data1=read(fd2[0],buf2,sizeof(buf2));
            printf("\nPARENT (%d):: %d bytes read %s.\n",getpid(),data1,buf2);
            sleep(3);
       } while ((strcmp(readdata,"0"))!= 0);
      wait(NULL);
       return 0;
  }
  } // both pipe creation successful....
  printf("\n done.... \n");
  return(0);
}
```

```
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ gcc intInp6.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ./a.out
PARENT (3334):: flag=1
PARENT (3334):: Enter an integer:5
PARENT (3334):: writes 2 bytes 5
CHILD (3335):: 2 bytes read 5  5.
CHILD (3335):: flag = 1
CHILD (3335):: Cum=15
CHILD (3335):: some_data1= 15
CHILD (3335):: Child writes 3 bytes string  15
PARENT (3334):: 3 bytes read 15.
PARENT (3334):: flag=1
PARENT (3334):: Enter an integer:0
PARENT (3334):: writes 2 bytes 0
CHILD (3335):: 2 bytes read 0  0.
CHILD (3335):: flag = 0
CHILD (3335):: Cum=0
CHILD (3335):: some_data1= 0
CHILD (3335):: Child writes 2 bytes string  0
PARENT (3334):: 2 bytes read 0.
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$
```

5. Write a program that creates **one-way pipe** between a **parent** and **three child processes**. ***The parent process gets an integer number range from standard input and divides the entire range to three child processes***. Each of the child after receiving the sub-range*, searches for prime numbers in that range and prints them.*

/* Try sprintf and sscanf first */

```
#include<stdio.h>
int main()
{
   char str1[30],str2[30],str3[30];
   int firstnum, secondnum;
   int  start,end,mid1,mid2;
   printf("enter range low high :");
```

```
    scanf("%d %d",&start,&end);
    mid1=(start+end)/3;
    mid2=2*mid1;
    sprintf(str1,"%d %d",start,mid1);
    sprintf(str2,"%d %d",mid1+1,mid2);
    sprintf(str3,"%d %d",mid2+1,end);
    // sscanf(characterArray, “Conversion specifier”, address of variables);
    sscanf(str1,"%d %d",&firstnum,&secondnum);
    // The strings start, mid1, mid1+1, mid2, mid2+1, end are stored
    // into str1, str2 and str3 instead of printing on stdout
    printf("\n str1 = %s \n str2 = %s \n str3 = %s \n",str1,str2,str3);
    printf("\n sscanf => firstnum = %d, secondnum = %d \n",firstnum,secondnum);
    return 0;
}
```

```
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ gcc trysprintf.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ./a.out
enter range low high :0 30
 str1 = 0 10
 str2 = 11 20
 str3 = 21 30
 sscanf => firstnum = 0, secondnum = 10
```

/* Try the assignment of three child processes printing the prime numbers */

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int isPrime(int n) {
        int flag, i;
        for(i=2;i<n;i++)    {
          if(n%i==0)
          {   flag=0;  break;   }
          flag=1;
        }
        return flag;
}
void range(int low,int high, char *msg) {
        int j;
        for(j=low; j<=high;j++)   {
                if(isPrime(j))
                  printf("%s prime -> %d \n",msg,j);
        }
}
```

```
int main() {
        int pid1,pid2,pid3;
        int p1[2], p2[2], p3[2];
        char str1[30],str2[30],str3[30];
        int start,end,mid1,mid2;
      char buf1[30], buf2[30], buf3[30];
        char *mg1="child 1", *mg2="child2", *mg="child3";
        pipe(p1);
        pid1=fork();
        if (pid1==0)
        {  printf("CHILD1: my pid = %d, my parent-ID = %d \n",getpid(),getppid());
         read(p1[0],buf1,sizeof(buf1));
          sscanf(buf1,"%d%d",&start,&end);
          range(start,end,mg1); sleep(2);
        }
        else if (pid1>0)      {
                   printf("PARENT: my pid = %d and my child1 = %d \n",getpid(),pid1);
                  pipe(p2);
              pid2=fork();
              if (pid2==0)
                 {   printf("CHILD2: my pid = %d, my parent-ID = %d \n",getpid(),getppid());
                     sleep(2);
                     read(p2[0],buf2,sizeof(buf2));
                     sscanf(buf2,"%d%d",&start,&end);
                     range(start,end,mg2);
         }
           else if (pid2>0)  {
                    printf("PARENT: my pid = %d and my child2 = %d \n",getpid(),pid2);
                     pipe(p3);
                     pid3=fork();
                     if (pid3==0)
                     {  printf("CHILD3: my pid = %d, my parent-ID = %d \n",getpid(),getppid());
                         sleep(4);
                        read(p3[0],buf3,sizeof(buf3));
                 sscanf(buf3,"%d%d",&start,&end);
```

```
                   range(start,end,mg);
                     }
                  else if (pid3>0)   {
                    //main program....................................................
                    printf("PARENT: my pid = %d and my child3 = %d \n",getpid(),pid3); sleep(2);

                  printf("enter range low high :");
                  scanf("%d %d",&start,&end);
                  mid1=(start+end)/3;
                  mid2=2*mid1;
                        printf("start = %d, mid1 = %d\n",start,mid1);
                  printf("mid1+1 = %d, mid2 = %d\n",(mid1+1),mid2);
                         printf("mid2+1 = %d, end = %d\n",(mid2+1),end);
                  sprintf(str1,"%d %d",start,mid1);
                  sprintf(str2,"%d %d",mid1+1,mid2);
                  sprintf(str3,"%d %d",mid2+1,end);
                  write(p1[1],str1,strlen(str1)+1); sleep(2);
                  write(p2[1],str2,strlen(str2)+1); sleep(2);
                  write(p3[1],str3,strlen(str3)+1); sleep(2);
                  wait();wait();wait();        // parent waits for three children.... You can comment
out....
               }
               else printf("error fork3\n");
             }
             else  printf("error fork2\n");
           }
           else printf("error fork1\n");
    return 0;
}
```

```
nilina@nilina-HP-Pro-3330-MT:~/Desktop/oslab/pipe$ ./a.out
PARENT: my pid = 3308 and my child1 = 3309
PARENT: my pid = 3308 and my child2 = 3310
CHILD1: my pid = 3309, my parent-ID = 3308
PARENT: my pid = 3308 and my child3 = 3311
CHILD2: my pid = 3310, my parent-ID = 3308
CHILD3: my pid = 3311, my parent-ID = 3308
enter range low high :0 10
start = 0, mid1 = 3
```

```
mid1+1 = 4, mid2 = 6
mid2+1 = 7, end = 10
child 1 prime -> 3
child2 prime -> 5
child3 prime -> 7
```