INTERNALS-3		
Course Code : 20CS5PCUSP	**Course Title : Unix Shell and System Programming**	
Semester : 5 A/B/C/D	**Maximum Marks: 40**	**Date: 18-01-2022**
SCHEME AND SOLUTIONS		

1. Explain the concept of pipes with suitable figure(s).

Solution:

Pipes are the oldest form of UNIX System IPC and are provided by all UNIX systems. Pipes have two limitations.

1. Historically, they have been half duplex (i.e., data flows in only one direction). Some systems now provide full-duplex pipes, but for maximum portability, we should never assume that this is the case.

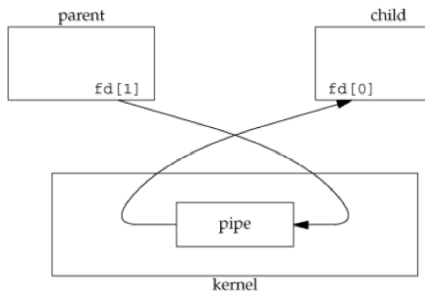
2. Pipes can be used only between processes that have a common ancestor. Normally, a pipe is created by a process, that process calls fork, and the pipe is used between the parent and the child.

Despite these limitations, half-duplex pipes are still the most commonly used form of IPC. Every time you type a sequence of commands in a pipeline for the shell to execute, the shell creates a separate process for each command and links the standard output of one process to the standard input of the next using a pipe. A pipe is created by calling the pipe function.

```
#include <unistd.h>

int pipe(int fd[2]);
```

Two file descriptors are returned through the fd argument: fd[0] is open for reading, and fd[1] is open for writing. The output of fd[1] is the input for fd[0].

Explanation of figures below



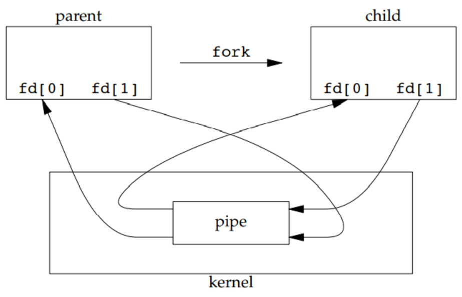


Figure 15.3 Half-duplex pipe after a fork

Figures: 2 Marks **Explanation: 3 Marks** **Total : 5 Marks**

2. a. Analyze the given program. Find the errors. Write the corrected program. Underline the places where errors are corrected.

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
 char buf[256];
 int fdesc, len;
 while(--argc>0)
 {
  if(access(*++argv,F_OK))
  {fdesc =open(*argv, O_WRONLY|O_CREAT,0744);
   write(fdesc,"HelloWorld\n",12);
  }
  close(fdesc);
 }
}
```

5 Errors- 1 Mark each **Total : 5 Marks**

2.b. Demonstrate the use of major and minor device number and mknod function with an example program.

- When a process reads or writes to a device file, the kernel uses the major and minor device numbers of a file to select a device driver function to carry out the actual data transfer.
- Device file support is implementation dependent. UNIX System defines the mknod API to create device files.
- Major Device number – index to kernel table used to locate and invoke a device driver function that does the actual data transmission with physical device.
- Minor Device number – argument which specifies parameters like buffer size to be used for a particular device type. This argument is passed as a parameter to a device driver function when it is invoked
- The prototype of mknod is

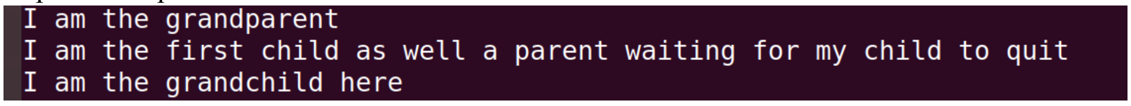
```
#include <sys/stat.h>

 #include<unistd.h>

int mknod(const char* path_name, mode_t mode, int device_id);
```

- The first argument pathname is the pathname of a device file to be created.
- The second argument mode specifies the access permission, for the owner, group and others, also S_IFCHR or S_IBLK flag to be assigned to the file.
- The third argument device_id contains the major and minor device number.
- Example mknod("SCSI5",S_IFBLK | S_IRWXU | S_IRWXG | S_IRWXO,(15<<8) | 3);
- The above function creates a block device file "SCSI5", to which all the three i.e. read, write and execute permission is granted for user, group and others with major number as 8 and minor number 3. On success mknod API returns 0 , else it returns -1
- Major device number uses fourteen bits and minor device number is of 18 bits.

Total: 5 Marks

2.c. Analyze the given program and the expected output. Fill in the set of statements in the place mentioned to get the output shown.

Expected output:



```
I am the grandparent
I am the first child as well a parent waiting for my child to quit
I am the grandchild here
```

Solution:

```
#include "apue.h"
#include <sys/wait.h>
int main(void)
{
pid_t pid;
if ((pid = fork()) < 0)
{
  printf("fork error");
}
else if (pid == 0)
{ /* first child */
  if ((pid = fork()) < 0)
      printf("fork error");
  else if (pid > 0)
    {
    printf("I am the first child as well a parent waiting for my child to quit\n");
    if (waitpid(pid, NULL, 0) != pid) /* wait for first child */
      printf("waitpid error");
    }
    else
    {printf("I am the grandchild here\n"); exit(0);}
}
else
{printf("I am the grandparent\n");
if (waitpid(pid, NULL, 0) != pid) /* wait for first child */
   printf("waitpid error");
exit(0);
}
}
```

5 Marks

3.a. Write a C/C++ program to accept an existing filename from the user. Check for the existing of the file and reaccept the filename if it is not existing. Logically divide the file contents into 5 equal (approximately) pieces. Print the first, third and fifth set of file contents. 10 Marks

Solution:

```
#include<fcntl.h>
#include<unistd.h>
#include <stdio.h>
int main(int argc, char **argv) {
```

```
char buf; int size,fd,ptr,bsize, count=0;
if(access(argv[1],F_OK))
 {  printf("Enter an existing file name");
   exit(0);
 }
 else
   { printf("file name-%s",argv[1]);
    fd=open(argv[1],O_RDONLY);

    size=lseek(fd,-1,SEEK_END);      /*Pointer taken to EOF -1 …..*/
    printf("Size:%d",size);
    bsize = size/5;
    printf("Block size:%d",bsize);                          5 Marks

    while(count<=4)
    {
    ptr=bsize;
    lseek(fd, count*bsize,SEEK_SET);
    printf("\n\nPrinting %d block\n", count+1);
    while(--ptr >=0) {
      read(fd,&buf,1);                        /*  Read 1 char  at a time  */
      write(STDOUT_FILENO,&buf,1);
      }
    printf("\nPrinted %d block",count+1);
    count+=2;
    //lseek(fd, count*bsize,SEEK_SET);
    }                                                        5 Marks
  }}                                                               Total: 10 Marks
```

3.b. Write a C/C++ program that demonstrates setjmp and longjmp by creating three user-defined functions say f1, f2, and f3. The function calls are as main() calls f1 which inturn calls f2 which calls f3. Let the user decide if f2 returns back to main or calls f3 which returns back to main. The main function displays which function as called last (f2 or f3) 10 Marks

Solution:

```
#include <setjmp.h>
#include <stdio.h>
#include <stdlib.h>
static void f1();
static void f2(void);
static void f3(void);
static jmp_buf jmpbuffer;
int ch;
int main(void)
{

if (setjmp(jmpbuffer) != 0)
   {printf("error"); exit(0);}
if (setjmp(jmpbuffer)==1)
  printf("longjmp from f3");
else if(setjmp(jmpbuffer)==2)
{
printf("longjmp from f2");
exit(0);
}
f1();
exit(0);
}
/*
* Change variables after setjmp, but before longjmp.
*/


static void f1()
{
printf("in f1():\n");
f2();
}
static void f2(void)
{
printf("in f2()\n");
printf("Enter 1 to go to f3 or 0 to go back to main");
scanf("%d",&ch);
if (ch==0)
longjmp(jmpbuffer, 2);
else
 f3();
}


static void f3(void)
{ printf("in f3()\n");
 longjmp(jmpbuffer,1);
}
```

Correct usage of setjmp – 5 marks correct usage of longjmp – 5 Marks   Total:10 Marks

4.A. Write a C/C++ program to accept five filenames from the user. Check for the existing of the files and if not existing, create the files. Print the contents of the files based on ascending order of their sizes (file with smallest size to be printed first)

Solution:

```
#include<fcntl.h>
#include<unistd.h>
#include <stdio.h>
int main(int argc, char **argv) {
char buf[80],*o1,*o2,*o3; int s1,sp1,s2,sp2,s3,sp3,fd,fd1,fd2,fd3; struct stat s11,s12,s13;
```

```
if(access(*argv[1],F_OK))
 {  printf("creating one\n");
    fd1 = open(argv[1], O_WRONLY|O_CREAT|O_TRUNC, 0777);
    strcpy(buf,"Hi welcome");    s1=strlen(buf);     write(fd1,&buf,s1);
    printf("Size of %s is %d\n",argv[1],s1);
}
else
    {
     printf("existing one\n");
     if(stat(argv[1],&s11)<0)
        printf("File error\n");
     else
       {
        s1=s11.st_size;        printf("Size of %s is %d\n",s11,s1);
       }
      }
if(access(*argv[2],F_OK))
 {  printf("creating two\n");
    fd2 = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC, 0777);
    strcpy(buf, "Hi welcome to bms");    s2=strlen(buf);    write(fd2,&buf,s2);
    printf("Size of %s is %d\n",argv[2],s2);
   }
else
    {printf("existing two\n");
     if(stat(argv[2],&s12)<0)
        printf("File error\n");
     else
       {
        s2=s12.st_size;
        printf("Size of %s is %d\n",s12,s2);
       }
    }
if(access(*argv[3],F_OK))
 {  printf("creating three\n");
    fd3 = open(argv[3], O_WRONLY|O_CREAT|O_TRUNC, 0777);
    strcpy(buf, "Hi");
    s3=strlen(buf);
    write(fd3,&buf,s3);
 printf("Size of %s is %d\n",argv[3],s3);
}
else
    {printf("existing three\n");
    if(stat(argv[3],&s13)<0)
        printf("File error\n");
     else
       {
        s3=s13.st_size;
        printf("Size of %s is %d\n",s13,s3);
       }
    }
printf("\n3 file sizes: %d %d %d\n ",s1,s2,s3);
```

5 Marks

```
if(s1<s2)
{
 if(s1<s3)
  { o1=argv[1];sp1=s1;
    if (s2<s3)
    {o2=argv[2];sp2=s2;}
    else
    {o3=argv[3];sp3=s3;}
  }
  else
   {
   o1=argv[3];sp1=s3;      o2=argv[1];sp2=s1;      o3=argv[2];sp3=s2;
   }
}
else
{  if(s2<s3)
   {
    o1=argv[2];sp1=s2;
    if(s3<s1)
    {
     o2=argv[3];sp2=s3;    o3=argv[1];sp3=s1;
    }
    else
      {
      o2=argv[1];sp2=s1;          o3=argv[3];sp3=s3;
      }
   }
   else
   { o1=argv[3];sp1=s3;        o2=argv[2];sp2=s2;        o3=argv[1];sp3=s1;
   }
}
   printf("file name-%s",o1);       fd=open(o1,O_RDONLY);      strcpy(buf," ");
   read(fd,&buf,sp1);      write(STDOUT_FILENO,&buf,sp1);

   printf("file name-%s",o2);
   fd=open(o2,O_RDONLY);       strcpy(buf," ");      read(fd,&buf,sp2);
   write(STDOUT_FILENO,&buf,sp2);

   printf("file name-%s",o3);      fd=open(o3,O_RDONLY);      strcpy(buf," ");
   read(fd,&buf,sp3);       write(STDOUT_FILENO,&buf,sp3);
}
```

5 Marks

Total: 10 Marks

4.b. Write a C/C++ program to create a FIFO. Read from a file and write into another file. Skip digits while writing as well convert all lowercase charcters to uppercase characters. 10 Marks

Solution:

```
#include<sys/types.h>
#include<unistd.h>
#include<fcntl.h>
```

```
#include<sys/stat.h>
#include<string.h>
#include<errno.h>
#include<stdio.h>
int main(int argc, char* argv[])
{
int fd,fd1,size,s;
char buf[256],buf1;
if(argc != 2 && argc != 3)
{
printf("USAGE %s <file> [<arg>]\n",argv[0]);
return 0;
}
mkfifo(argv[1],S_IFIFO | S_IRWXU | S_IRWXG | S_IRWXO );
if(argc == 3)   //reader process
{
fd = open(argv[1], O_RDONLY|O_NONBLOCK);
 size=lseek(fd,-1,SEEK_END);     /*Pointer taken to EOF -1 …..*/
   printf("Size:%d",size);
  s=lseek(fd,0,SEEK_SET);
  printf("s=%d",s);
   fd1 = open(argv[2], O_RDONLY|O_CREAT|O_TRUNC, 0777);               5 Marks

    while(--size >=0)
    {
     read(fd,&buf1,1);

      if(!isdigit(buf1))
        if(islower(buf1))
            {
                buf1=toupper(buf1);
             write(fd1,&buf1,1);
             write(STDOUT_FILENO,&buf1,1);
             //printf("\n");
            }
      }

close(fd);                                              5 Marks
}
}                                                       Total : 10 Marks
```