PART B

1. Write a C/C++ program to implement the cat command using general file API’s.

#include<sys/types.h>

#include<sys/stat.h>

#include<stdio.h>

#include<fcntl.h>

main( int argc,char \*argv[3] )

{

int fd,i;

char buf[2];

fd=open(argv[1],O\_RDONLY,0777);

if(fd==-argc)

{

printf("file open error");

}

else

{

while((i=read(fd,buf,1))>0)

{

printf("%c",buf[0]);

}

close(fd);

}

}

Output: [UNIXLAB@localhost ~]$ gcc -0 p.out prgm1.c

UNIXLAB@localhost ~]$cat >h.txt /\*creating a text file\*/

hello

UNIXLAB@localhost ~]$./p.out h.txt /\*appending text file to display contents on the terminal without using cat command\*/

hello

2. Write a C/C++ program to implement the cp (copy) command using general file API’s.

#include <stdio.h>

#include <stdlib.h

#include <fcntl.h>

#include <errno.h>

#define BUFF\_SIZE 1024

int main(int argc, char\* argv[])

{

int srcFD,destFD,nbread,nbwrite;

char \*buff[BUFF\_SIZE];

/\*Check if both src & dest files are received\*/

if(argc != 3)

{

printf("\nUsage: cpcmd source\_file destination\_file\n");

exit(EXIT\_FAILURE);

}

/\*Open source file\*/

srcFD = open(argv[1],O\_RDONLY);

if(srcFD == -1)

{

printf("\nError opening file %s errno = %d\n",argv[1],errno);

exit(EXIT\_FAILURE);

}

/\*Open destination file with respective flags & modes

O\_CREAT & O\_TRUNC is to truncate existing file or create a new file

S\_IXXXX are file permissions for the user,groups & others\*/

destFD = open(argv[2],O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH);

if(destFD == -1)

{

printf("\nError opening file %s errno = %d\n",argv[2],errno);

exit(EXIT\_FAILURE);

}

/\*Start data transfer from src file to dest file till it reaches EOF\*/

while((nbread = read(srcFD,buff,BUFF\_SIZE)) > 0)

{

if(write(destFD,buff,nbread) != nbread)

printf("\nError in writing data to %s\n",argv[2]);

}

if(nbread == -1)

printf("\nError in reading data from %s\n",argv[1]);

if(close(srcFD) == -1)

printf("\nError in closing file %s\n",argv[1]);

if(close(destFD) == -1)

printf("\nError in closing file %s\n",argv[2]);

exit(EXIT\_SUCCESS);

}

Output: Output: gcc prgm2.c

[UNIXLAB@localhost ~]$ ./a.out q.txt j.txt

UNIXLAB@localhost ~]$cat q.txt

hello

UNIXLAB@localhost ~]$cat j.txt

hello

3. Write a C/C++ program to implement the ln/rename command using general file API’s.

#include <stdio.h>

#include<unistd.h>

int main(int argc, char \*argv[])

{

if(argc!=3)

{

printf(“usage: %s <src\_file><dest\_file>\n”,argv[0]);

Return 0;

}

if(link(argv[1],argv[2])==-1)

{

printf(“link error\n”);

return 1;

}

printf(“files linked\n”);

printf(“Inode number of linked files\n”);

char str[100];

sprint(str,”ls –i %s %s \n”,argv[1],argv[2]);

system(str);

return 0;

}

Output: gcc prgm1.c

[UNIXLAB@localhost ~]$ ./a.out j.txt q.txt

Files linked

Inode number of linked files

416529 j.txt 416529 q.txt

4. Write a C/C++ program to create a file called file1 in blocking read-write mode and show how you can use fcntl api to modify its access control flags to non-blocking read-write mode.

#include<stdio.h>

#include<sys/types.h>

#include<fcntl.h>

#include<stdlib.h>

int main(int argc,char \*argv[])

{

int accmode,val;

if(argc!=2)

{

fprintf(stderr,"usage:%s <description>",argv[0]);

exit(1);

}

val=fcntl(atoi(argv[1]),F\_GETFL,0); /\* F\_GETFL (void)

Return (as the function result) the file access mode and the

file status flags;\*/

if(val<0)

{

perror("fcntl error for fd");

exit(1);

}

accmode=val & O\_ACCMODE; /\*This macro stands for a mask that can be bitwise-ANDed with the file status flag value to produce a value representing the file access mode. The mode will be O\_RDONLY, O\_WRONLY, or O\_RDWR.\*/

if(accmode==O\_RDONLY)

printf("read only");

else if(accmode==O\_WRONLY)

printf("Write only");

else if(accmode==O\_RDWR)

printf("read write");

else

{

fprintf(stderr,"unknown access mode");

exit(1);

}

if(val & O\_APPEND)

printf(",append");

if(val & O\_NONBLOCK)

printf(",nonblocking");

if(val & O\_SYNC)

printf(",synchronous write"); /\*wait for writes to complete (data and attributes)\*/

putchar('\n');

exit(0);

}

Output:

gcc p22.c

[UNIXLAB@localhost ~]$ ./a.out p22.c

Read-Write

5.Write a C/C++ program to duplicate the file descriptor of a file Foo to standard input file descriptor.

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<stdio.h>

int main(int argc,char \*\*argv)

{

int fd,nfd;

if(argc<2){

printf("usage:%s pathname\n",argv);

exit(1);

}

if((fd=open(argv[1],O\_WRONLY))<0)

{

perror("Problem in opening the file");

exit(1);

}

if((nfd=fcntl(fd,F\_DUPFD,0))==-1)

{

perror("Problem in duplicating fd");

exit(1);

}

printf("Fd %d duplicated with %d\n",fd,nfd);

close(fd);

close(nfd);

}

Output: gcc prgm.c

[UNIXLAB@localhost ~]$ ./a.out ll.txt

Fd 3 duplicated with 4

gcc prgm.c

[UNIXLAB@localhost ~]$ ./a.out ll.txt

Fd 3 duplicated with 4

6.Write a C/C++ program to query and display the different attributes associated with a file.

#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<time.h>

#include<stdlib.h>

int main(int argc, char \*argv[])

{

struct stat sb;

if(argc!=2)

{

fprintf(stderr,"usage: %s <pathname>\n", argv[0]);

exit(EXIT\_FAILURE);

}

if(stat(argv[1],&sb)==-1)

{

perror("stat");

exit(EXIT\_FAILURE);

}

printf("file type: ");

switch(sb.st\_mode & S\_IFMT)

{

case S\_IFBLK: printf("block device file\n");

break;

case S\_IFCHR: printf("character device file\n");

break;

case S\_IFDIR: printf("directory\n");

break;

case S\_IFIFO: printf("FIFO/pipe\n");

break;

case S\_IFLNK: printf("symlink\n");

break;

case S\_IFREG: printf("regular file\n");

break;

case S\_IFSOCK: printf("socket\n");

break;

default: printf("regular file\n");

break;

}

printf("Inode number: %ld\n", (long) sb.st\_ino);

printf("Mode: %lo(octal)\n", (unsigned long) sb.st\_mode);

printf("Blocks allocated: %lld\n", (long long) sb.st\_blocks);

exit(EXIT\_SUCCESS);

}

Output: gcc prgm.c

[UNIXLAB@localhost ~]$ ./a.out prgm.c

File type: regular

Inode number: 1067168

Mode: 100664(octal)

Blocks allocated: 8

7.Write C/C++ program to read and display the last 10 character’s of the input file.

#include<stdio.h>

#include<stdlib.h>

int main()

{

FILE \*fp;

char ch;

int num;

long length;

printf("Enter the value of num:");

scanf("%d",&num);

fp=fopen("file.txt","r");

if(fp==NULL)

{

puts("Cannot open this file");

exit(1);

}

fseek(fp,-1,SEEK\_END);

length=ftell(fp);

fseek(fp,(length-num),SEEK\_SET);

do

{

ch=fgetc(fp);

putchar(ch);

}while(ch!=EOF);

fclose(fp);

return(0);

}

Output

cat > l.txt

hello world hai beautiful girl

gcc p55.c

[UNIXLAB@localhost ~]$ ./a.out p55.c

Enter the value of num:4

Girl

8.Write a C/C++ program to demonstrate masking of read/write/execute permission of a specified input file for user group and others category.

#include<sys/types.h>

#include<sys/stat.h>

#include<stdio.h>

int main()

{

mode\_t oldMask,newMask;

oldMask=umask((mode\_t)0);

printf("\n Old mask = %on",(int)oldMask);

if(oldMask & S\_IRGRP){

printf("\nChanging group read permission from Masked to unmasked.n");

oldMask=(oldMask ^ S\_IRGRP);/\* ^ Operator is binar XOR operator, copies the bit if it is set in one operand but not in both. Exclusive or is a logical operator that outputs true only when inputs differ\*/

}

newMask=(oldMask|S\_IWGRP|S\_IXGRP);

umask(newMask);

printf("\nNew MAsk = %onn",(int)newMask);

printf("\nThe file mode creation mask now specifies:nn");

printf("\n Group read permission UNMASKEDn");

printf("\n Group write permission MASKEDn");

printf("\n Group execute permission MASKEDn");

oldMask=umask((mode\_t)0);

printf("\n Old mask = %on",(int)oldMask);

if(oldMask & S\_IRUSR){

printf("\nChanging user read permission from Masked to unmasked.n");

oldMask=(oldMask ^ S\_IRUSR);

}

newMask=(oldMask|S\_IWUSR|S\_IXUSR);

umask(newMask);

printf("\nNew MAsk = %onn",(int)newMask);

printf("\nThe file mode creation mask now specifies:nn");

printf("\n User read permission UNMASKEDn");

printf("\n User write permission MASKEDn");

printf("\n User execute permission MASKEDn");

oldMask=umask((mode\_t)0);

printf("\n Old mask = %on",(int)oldMask);

if(oldMask & S\_IROTH){

printf("\nChanging Other read permission from Masked to unmasked.n");

oldMask=(oldMask ^ S\_IROTH);

}

newMask=(oldMask|S\_IWOTH|S\_IXOTH);

umask(newMask);

printf("\nNew MAsk = %onn",(int)newMask);

printf("\nThe file mode creation mask now specifies:nn");

printf("\n Other read permission UNMASKEDn");

printf("\n Other write permission MASKEDn");

printf("\n Other execute permission MASKEDn");

}

Output:

gcc p77.c

[UNIXLAB@localhost ~]$ ./a.out p77.c

Old mask = 2n

New MAsk = 32nn

The file mode creation mask now specifies:nn

Group read permission UNMASKEDn

Group write permission MASKEDn

Group execute permission MASKEDn

Old mask = 32n

New MAsk = 332nn

The file mode creation mask now specifies:nn

User read permission UNMASKEDn

User write permission MASKEDn

User execute permission MASKEDn

Old mask = 332n

New MAsk = 333nn

The file mode creation mask now specifies:nn

Other read permission UNMASKEDn

Other write permission MASKEDn

Other execute permission sMASKEDnss