1

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
int main()
{
	int file=0, n;
	char buffer[25];
	if((file=open("testfile.txt",O_RDONLY)) < -1)
		printf("file open error\n");
	if(read(file,buffer,20) != 20)
		printf("file read operation failed\n");
	else
		write(STDOUT_FILENO, buffer, 20);
	printf("\n");
	if(lseek(file,10,SEEK_SET) < 0)
		printf("lseek operation to beginning of file failed\n");
	if(read(file,buffer,20) != 20)
		printf("file read operation failed\n");
	else
		write(STDOUT_FILENO, buffer, 20);
	printf("\n");
	if(lseek(file,10,SEEK_CUR) < 0)
		printf("lseek operation to beginning of file failed\n");
	if(read(file,buffer,20) != 20)
		printf("file read operation failed\n");
	else
		write(STDOUT_FILENO, buffer, 20);
	printf("\n");
	if((n = lseek(file,0,SEEK_END)) <0)
		printf(“lseek operation to end of file failed\n”);
	printf("size of file is %d bytes\n",n);
	close(file);
	return 0;
}
```

```
2. #include<stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include <time.h>
int main(int argc,char* argv[])
{
	struct dirent *dir;
	struct stat mystat;
	DIR *dp;
	dp = opendir(".");
	if(dp)
	{
		while(dir = readdir(dp))
		{
		stat(dir->d_name,&mystat);
		// inode mode uid guid access_time;
```

```
          printf("%ld %o %d %d %s %s\n",mystat.st_ino ,
      mystat.st_mode , mystat.st_uid, mystat.st_gid,
      ctime(&mystat.st_atime) , dir->d_name);
          }
      }
}
```

3.

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{char buf[100];
int fd1,fd2;
off_t size,ret,set;
ssize_t readdata,writedata;
if(argc<3)
      printf("TOO FEW ARGUMENTS");

fd1=open(argv[1],O_RDONLY); //Open file 1
if(fd1==-1)
      printf("ERROR IN OPENING FILE: FILE DOES NOT
EXIST \n");
else
      printf("FILE 1 OPENED SUCCESSFULLY \n");
```

```
fd2=open(argv[2],O_WRONLY | O_CREAT | O_TRUNC, 0666);
//open file 2 in read-write mode, truncate its length to 0, create the
file if it does not exist, 0666 is the access permission for the
created file. order is important.
if(fd2==-1)
	printf("ERROR IN OPENING FILE");
else
	printf("FILE 2 OPENED SUCCESSFULLY \n");

size=lseek(fd1,0L,SEEK_END); //obtain the size of file 1 using
lseek
if(size==-1)
	printf("ERROR: COULD NOT OBTAIN FILE SIZE \n");
else
	printf("FILE SIZE OF FILE 1 OBTAINED \n");

ret=lseek(fd1,0L,SEEK_SET); //change the current pointer to the
beginning of the file
if(ret==-1)
	printf("RETRACE FAILED \n");

readdata=read(fd1,buf,size); //read data equal to the size of the
first file
if(readdata==-1)
	printf("ERROR IN READING FILE CONTENTS \n");

writedata=write(fd2,buf,size);  //write the data to file 2 from buffer
after read
```

```
if(writedata!=size)
	printf("ERROR IN COPYING FILE");
else
	printf("FILE COPIED SUCCESSFULLY");
return 0;
}
```

4.
```
#include <stdio.h>
#include <unistd.h>
void print_unbuffered(const char* str)
{
	setbuf(stdout, NULL);
	while(true)
	{
		char ch = *str;
		if (ch == '\0') break;
		printf("%c", ch); // alternatively, use putc
		str++;
	}
}

int main()
{
	pid_t pid = fork();
	if (pid < 0)
	{
		return -1;
	}
	if (pid)
	{
		print_unbuffered("parent parent parent parent parent");
	}
	else
	{
		print_unbuffered("child child child child child");
	}
}
```

5.

```
/*program to create a daemon process */

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

void daemonize()
{
	pid_t pid = fork();

	if(pid < 0)
		fprintf(stderr, "Error Forking\n");
	else if(pid)
	{
		printf("PID of Child %d\n",pid);
		exit(0); // kill the parent process , child is orphanded and
runs in the bg
	}

	umask(0);
	if(chdir("/") < 0)
		printf("Error changing directory \n");
	if(setsid() < 0)  //make the child process the session leader
		printf("Error creating session\n");

	printf("Daemon Created! \n");

}
int main()
{
	deamonize();
	system("ps -axj");
	return 0;
}
```

6./*Write a program (use signal system call)
i. which calls a signal handler on SIGINT signal and then reset the default action of the
SIGINT signal

ii. Which ignores SIGINT signal and then reset the default action of SIGINT signal*/

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

void callback()
{
	printf("Interrupt Received !\n");
	(void)signal(SIGINT,SIG_DFL);
}

int main()
{
	int ch,i=0;
	printf("Enter choice 1: call handler and default\n2: ignore 
first time and default\n");
	scanf("%d",&ch);

	switch(ch)
	{
		case 1 :	(void)signal(SIGINT,callback);
				break;

		case 2 :	(void)signal(SIGINT,SIG_IGN);
				break;
```

```
        }
        while(1)
        {
                sleep(1);
                printf("Press CTRL+C ...\n");
                i++;
                if(i == 10 && ch == 2)
                        (void) signal(SIGINT,SIG_DFL);
        }
        return 0;
}
```