Student Name:	Aditya.Rajesh.Wanwade
Roll No:	C2-27
Practical No:	1
Aim:	a) Write a C programs to implement UNIX system calls and file management.

1.A) To write the program to implement fork () system call.

```c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main(){
int id,childid;
id=getpid();
if((childid=fork())>0){
printf("\n i am in the parent process %d",id);
printf("\n i am in the parent process %d",getpid());
printf("\n i am in the parent process %d\n",getppid());

}
else{
printf("\n i am in the child process %d",id);
printf("\n i am in the child process %d",getpid());

printf("\n i am in the child process %d",getppid());
}}
~
~
~
~
~
~
```

OUTPUT: (All the test cases are included)

```
┌──(klinux㉿kali)-[~]
└─$ cc fork.c

┌──(klinux㉿kali)-[~]
└─$ ./a.out

i am in the parent process 6580
i am in the parent process 6580
i am in the parent process 3930


i am in the child process 6580
i am in the child process 6581
┌──(klinux㉿kali)-[~]
└─$
```

1.B.1) To write the program to implement the system calls wait () and exit ().

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/wait.h>
int main(){
int i,pid;
pid=fork();
if(pid==-1){
printf("fork failed");
exit(0);
}
else if (pid==0){
printf("\n child process starts");
for(i=0;i<5;i++){
printf("\n child process %d is called ",i);
}
printf("\n child process ends");
}
else{
wait(0);
printf("\n Parent process ends");
}
exit(0);
}
~
~
~
~
```

OUTPUT: (All the test cases are included)

```
┌──(klinux㉿kali)-[~]
└─$ vi wait.c

┌──(klinux㉿kali)-[~]
└─$ cc wait.c

┌──(klinux㉿kali)-[~]
└─$ ./a.out

 child process starts
 child process 0 is called
 child process 1 is called
 child process 2 is called
 child process 3 is called
 child process 4 is called
 child process ends
 Parent process ends
```

1.B.2) To write the program to implement the system calls wait () and exit ().

```c
#include <stdlib.h>
#include <errno.h>
#include<stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
void main()
{
pid_t pid;
int rv;
switch(pid=fork())
{
 case -1:
perror("fork");
exit(1);
 case 0:
printf("\n CHILD: This is the child process!\n");
fflush(stdout);
printf("\n CHILD: My PID is %d\n", getpid());
printf("\n CHILD: My parent's PID is %d\n",getppid());
printf("\n CHILD: Enter my exit status (make it small):\n ");
printf("\n CHILD: I'm gone to end now!\n");
scanf(" %d", &rv);
exit(rv);
default:
printf("\nPARENT: This is the parent process!\n");
printf("\nPARENT: My PID is %d\n", getpid());
fflush(stdout);
wait(&rv);
fflush(stdout);
printf("\nPARENT: My child's PID is %d\n", pid);
printf("\nPARENT: I'm now waiting for my child to exit()...\n");
fflush(stdout);
printf("\nPARENT: My child's exit status is:%d\n",WEXITSTATUS(rv));
printf("\nPARENT: I'm gone to end now\n");
}

}
```

OUTPUT: (All the test cases are included)

```
┌──(kali㉿kali)-[~/C27]
└─$ vi 1b2.c

┌──(kali㉿kali)-[~/C27]
└─$ gcc 1b2.c

┌──(kali㉿kali)-[~/C27]
└─$ ./a.out

PARENT: This is the parent process!

PARENT: My PID is 37338

 CHILD: This is the child process!

 CHILD: My PID is 37339

 CHILD: My parent's PID is 37338

 CHILD: Enter my exit status (make it small):

 CHILD: I'm gone to end now!
^C
```

1.C)To write a program to implement the system call execl()

```
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
void main()
{
printf("Before execl \n");
execl("/bin/ls","ls",(char*)0);
printf("After Execl\n");
}
~
~
~
```

OUTPUT: (All the test cases are included)

```
┌──(kali㉿kali)-[~/C27]
└─$ vi 1c.c

┌──(kali㉿kali)-[~/C27]
└─$ gcc 1c.c

┌──(kali㉿kali)-[~/C27]
└─$ ./a.out
Before execl
1b2.c  1c.c  a.out  new2.c  new.c
```

1.D)To write a program to implement the system call execv()

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
void main(int argc,char *argv[])
{
printf("before execv\n");
execv("/bin/ls",argv);
printf("after execv\n");
}
```

OUTPUT: (All the test cases are included)

```
┌──(kali㉿kali)-[~/C27]
└─$ vi 1d.c

┌──(kali㉿kali)-[~/C27]
└─$ gcc 1d.c

┌──(kali㉿kali)-[~/C27]
└─$ ./a.out
before execv
1b2.c  1c.c  1d.c  a.out  new2.c  new.c
```

1.E)To write the program to implement the system calls opendir (),readdir (),closedir ().

```c
#include<stdio.h>
#include<dirent.h>
#include<errno.h>
#include<fcntl.h>
#include<unistd.h>
int main(int argc,char *argv[])
{
struct dirent *direntp; DIR *dirp; if(argc≠2)
{
printf("ussage %s directory name \n",argv[0]);
return 1;
}
if((dirp=opendir(argv[1]))==NULL)
{
perror("Failed to open directory \n");
return 1;
}
while((direntp=readdir(dirp))≠NULL)
printf("%s\n",direntp→d_name);
while((closedir(dirp)==-1)&&(errno==EINTR));
return 0;
}
```

OUTPUT: (All the test cases are included)

```
┌──(kali㉿kali)-[~/C27]
└─$ gcc 1e.c

┌──(kali㉿kali)-[~/C27]
└─$ ./a.out /lib32
libutil.so.1
libstdc++.so.6
libnss_compat.so.2
libBrokenLocale.so.1
libstdc++.so.6.0.32
libanl.so.1
librt.so.1
..
libnsl.so.1
libc_malloc_debug.so.0
gconv
libresolv.so.2
libnss_dns.so.2
libmemusage.so
libpcprofile.so
libpthread.so.0
libm.so.6
libthread_db.so.1
libnss_hesiod.so.2
.
libdl.so.2
libnss_files.so.2
libgcc_s.so.1
ld-linux.so.2
libc.so.6
```

1.F)To write the program to implement the system calls open ( ), read ( ), write ( ) & close ().

```c
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>


int main()
{
    int fd[2];
    char buf1[25] = "just a test \n";
    char buf2[50];
    fd[0] = open("file1.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    fd[1] = open("file2.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (fd[0] == -1 || fd[1] == -1)
    {
        perror("Open");
        exit(EXIT_FAILURE);
    }
    // Writing to file1.txt
    write(fd[0], buf1, strlen(buf1));
    printf("Enter the text now: ");
    fgets(buf1, sizeof(buf1), stdin);
    write(fd[0], buf1, strlen(buf1));
    // Rewind to the beginning of file1.txt
    lseek(fd[0], 0, SEEK_SET); // Fix: Replace Iseek with lseek
    // Reading from file1.txt
    ssize_t bytes_read = read(fd[0], buf2, sizeof(buf2));
    if (bytes_read == -1)
    {
        perror("read");
        exit(EXIT_FAILURE);
    }
    // Writing the read data to console
    printf("Data read from file1.txt: %s\n", buf2);

    // Rewind to the beginning of file2.txt
    lseek(fd[1], 0, SEEK_SET); // Fix: Replace Iseek with lseek
    // Reading from file2.txt
    bytes_read = read(fd[1], buf2, sizeof(buf2));
    if (bytes_read == -1)
    {
```

```
        perror("read");
        exit(EXIT_FAILURE);
    }
    // Writing the read data to console
    printf("Data read from file2.txt: %s\n", buf2);
    close(fd[0]);
    close(fd[1]);
    return 0;
}
```

OUTPUT: (All the test cases are included)

```
PS E:\coding p> cd "e:\coding p\" ; if ($?) { gcc 1f.c -o 1f } ; if ($?) { .\1f }
Enter the text now: Hello World
Data read from file1.txt: just a test
Hello World

a
Data read from file2.txt: just a test
Hello World

a
PS E:\coding p>
```

Result: Thus we have implemented UNIX system calls and file management in C.