**Institute of Computer Technology**

**B. Tech Computer Science and Engineering**

**Sub: Operating Systems**

**Practical 1: System Call**

**File handling and directory management system calls (read, write, open create, close opendir,**

**mkdir, changedir, rmdir)**

**p11.c: To display directory contents using readdir system call**

**Input**:

#include <stdio.h>

#include <dirent.h>

#include <stdlib.h>

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

{

struct dirent \*dptr; DIR \*dname;

if (argc != 2)

{

printf("Usage: ./a.out <dirname>\n"); exit(-1);

}

if((dname = opendir(argv[1])) == NULL)

{perror(argv[1]);

exit(-1);

}

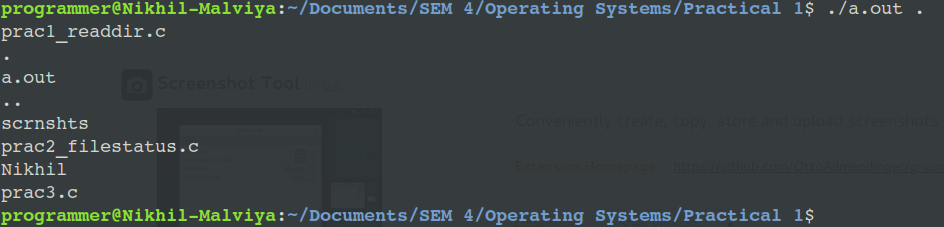
while(dptr=readdir(dname))

printf("%s\n", dptr->d\_name);

closedir(dname);

}

**Screen Shot:**

****

**p12.c: To display file status using stat system call**

**Input:**

#include<sys/types.h>

#include<sys/stat.h>

#include<time.h>

#include<stdio.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\n");

break;

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

break;

}

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

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

printf("Link count:%ld\n", (long) sb.st\_nlink));

printf("Ownership:UID=%ldGID=%ld\n", (long) sb.st\_uid,(long) sb.st\_gid);

printf("Preferred I/O block size: %ld bytes\n",(long)sb.st\_blksize);

printf("File size:%lld bytes\n", (long long)sb.st\_size);

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

printf("Last status change:%s", ctime(&sb.st\_ctime));

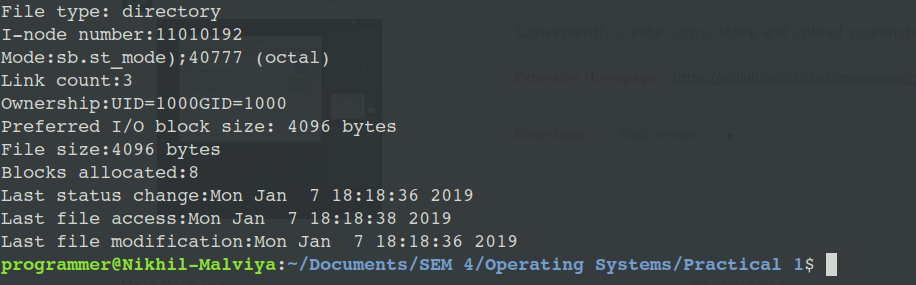
printf("Last file access:%s", ctime(&sb.st\_atime));

printf("Last file modification:%s", ctime(&sb.st\_mtime));

exit(EXIT\_SUCCESS);

}

**Screen Shot:**

****

**p13.c: To create a file and to write contents.**

**Input:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

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

{

int fd, n, len;

char buf[100];

if (argc != 2)

{

printf("Usage: ./a.out <filename>\n"); exit(-1);

}

fd = open(argv[1], O\_WRONLY|O\_CREAT|O\_TRUNC, 0644);

if(fd < 0)

{

printf("File creation problem\n"); exit(-1);

}

printf("Press Ctrl+D at end in a new line:\n");

while((n = read(0, buf, sizeof(buf))) > 0)

{

len = strlen(buf);

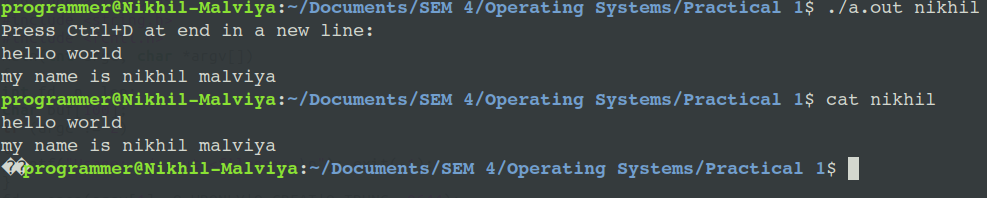
write(fd, buf, len);

}

close(fd);

}

**Screen Shot:**



**p14.c: To read the given file and to display file contents.**

**Input:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

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

{

int fd,i;

char buf[100];

if (argc < 2)

{

printf("Usage: ./a.out <filename>\n"); exit(-1);

}

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

{

printf("%s file does not exist\n", argv[1]);

exit(-1);

}

printf("Contents of the file %s is : \n", argv[1]);

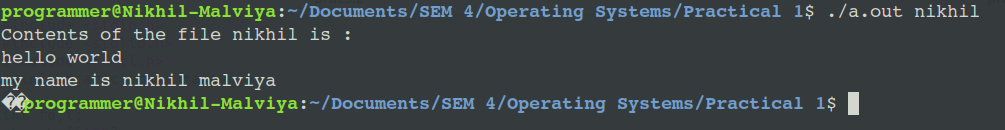
while(read(fd, buf, sizeof(buf)) > 0)

printf("%s", buf);

close(fd);

}

**Screen Shot:**



**p15.c: To append content to an existing file.**

**Input:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <fcntl.h>

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

{

int fd, n, len;

char buf[100];

if (argc != 2)

{

printf("Usage: ./a.out <filename>\n"); exit(-1);

}

fd = open(argv[1], O\_APPEND|O\_WRONLY|O\_CREAT, 0644); if (fd < 0)

{

perror(argv[1]);

exit(-1);

}

while((n = read(0, buf, sizeof(buf))) > 0)

{

len = strlen(buf);

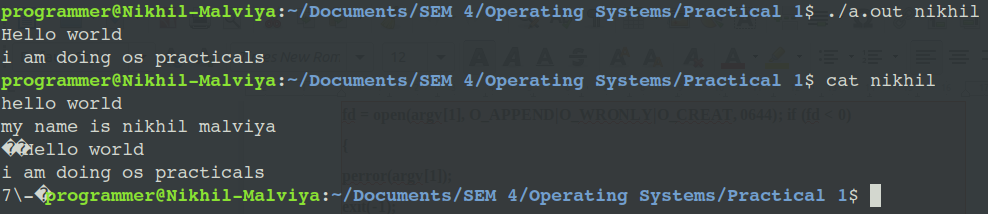
write(fd, buf, len);

}

close(fd);

}

**Screen Shot:**



**p16.c: To create a new child process using fork system call.**

**Input:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

main()

{

pid\_t pid;

int x = 5;

pid = fork(); x++;

if (pid < 0)

{

printf("Process creation error");

exit(-1);

}

else if (pid == 0)

{

printf("Child process:");

printf("\nProcess id is %d", getpid());

printf("\nValue of x is %d", x);

printf("\nProcess id of parent is %d\n", getppid());

}

else

{

printf("\nParent process:");

printf("\nProcess id is %d", getpid());

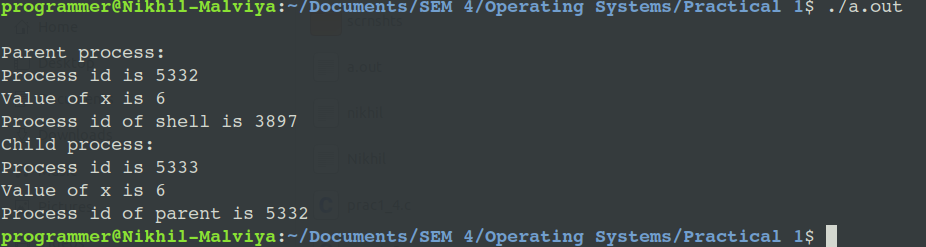
printf("\nValue of x is %d", x);

printf("\nProcess id of shell is %d\n", getppid());

}

}

**Screen Shot:**



**p17.c: To block a parent process until child completes using wait system call.**

**Input:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

main()

{

int i, status;

pid\_t pid;

pid = fork();

if (pid < 0)

{

printf("\nProcess creation failure\n"); exit(-1);

}

else if(pid > 0)

{

wait(NULL);

printf ("\nParent starts\nEven Nos:");

for (i=2;i<=10;i+=2)

printf ("%3d",i);

printf("\nParent ends\n");

}

else if (pid == 0)

{printf ("Child starts\nOdd Nos:");

for (i=1;i<10;i+=2)

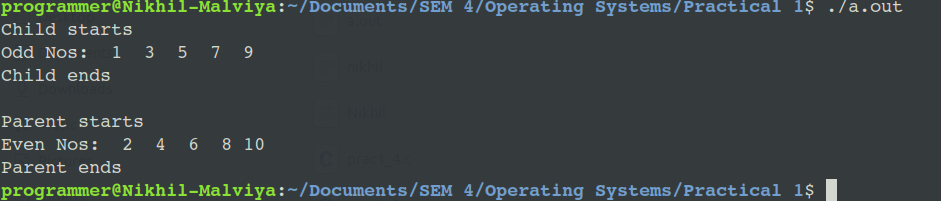
printf ("%3d",i);

printf ("\nChild ends\n");

}

}

**Screen Shot:**



**p19.c: To simulate ls command using UNIX system calls**

**Input:**

#include <stdio.h>

#include <dirent.h>

main()

{

struct dirent \*\*namelist; int n,i;

char pathname[100];

getcwd(pathname);

n = scandir(pathname, &namelist, 0, alphasort);

if(n < 0)

printf("Error\n");

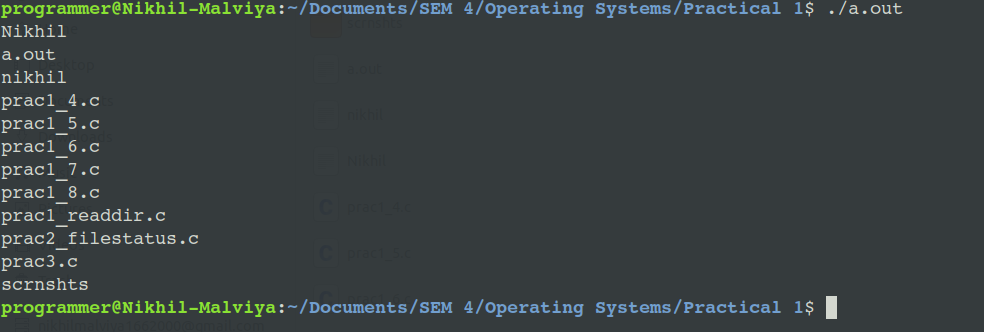
else

for(i=0; i<n; i++) if(namelist[i]->d\_name[0] != '.')

printf("%-20s", namelist[i]->d\_name);

}

**Screen Shot:**



**p111.c: To simulate grep command using UNIX system call**

**Input:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

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

{

FILE \*fd;

char str[100];

char c;

int i, flag, j, m, k;

char temp[30];

if(argc != 3)

{

printf("Usage: gcc mygrep.c –o mygrep\n");

printf("Usage: ./mygrep <search\_text><filename>\n"); exit(-1);

}

fd = fopen(argv[2],"r");

if(fd == NULL)

{

printf("%s is not exist\n",argv[2]);

exit(-1);

}

while(!feof(fd))

{

i = 0;

while(1)

{

c = fgetc(fd);if(feof(fd))

{

str[i++] ='\0';

break;

}

if(c == '\n')

{

str[i++] ='\0';

break;

}

str[i++] = c;

}

if(strlen(str) >= strlen(argv[1]))

for(k=0; k<=strlen(str)-strlen(argv[1]); k++)

{

for(m=0; m<strlen(argv[1]);m++)

temp[m] = str[k+m];

temp[m] = '\0';

if(strcmp(temp,argv[1]) == 0)

{

printf("%s\n",str);

break;

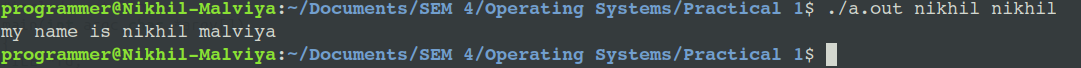
}

}

}

}

**Screenshot:**



**p112.c: To simulate rm command using UNIX system call.**

**Input:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

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

{

int fd;

if (argc != 2)

{

printf("Usage: gcc del.c –o del\n");

printf("Usage: ./del <filename>\n");

exit(-1);

}

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

if (fd != -1)

{

close(fd);

unlink(argv[1]);

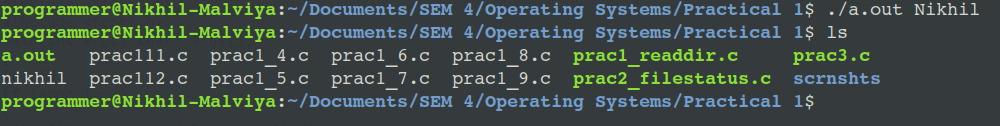
}

else

perror(argv[1]);

}

**Screenshot:**

****

**PRE-LAB QUESTIONS**

**1. Define System calls.**

**Ans:** A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system’s kernel.

**2. List out different Types of System Calls.**

**There are five types of system calls:**

1.Process control.

2.File management.

3.Device management.

4.Information maintenance.

5.Communications.

**3. Give suitable Examples of Windows and Unix System Calls.**

|  |  |  |
| --- | --- | --- |
| **Types of System Calls** | **Windows** | **Linux** |
| Process Control | CreateProcess()  ExitProcess()  WaitForSingleObject() | fork()  exit()  wait() |
| File Management | CreateFile()  ReadFile()  WriteFile()  CloseHandle() | open()  read()  write()  close() |
| Device Management | SetConsoleMode()  ReadConsole()  WriteConsole() | ioctl()  read()  write() |
| Information Maintenance | GetCurrentProcessID()  SetTimer()  Sleep() | getpid()  alarm()  sleep() |
| Communication | CreatePipe()  CreateFileMapping()  MapViewOfFile() | pipe()  shmget()  mmap() |

**POST-LAB QUESTIONS**

**1. System call routines of operating system are mostly written in** C and C++

**2. What is the total number of processes at the end of the execution of the following**

**program?**

**main() {**

**fork();**

**fork();**

**fork();**

**}**

**Answer :**

8

**3. Consider the following program:**

**main() {**

**int fd;**

**fd = open(“outfile”, O\_RDWR)**

**fork();**

**write(fd, “hello”, 5);**

**exit();**

**}**

**Assume all system calls finish successfully on a uniprocessor system. Also, assume that a**

**system call cannot be interrupted in the middle of its execution. What will be the contents**

**of the “outfile” file, after all processes have successfully exited? Explain brieflyAnswer:**

content of outfile should be hellohello because of the fork call. One time hello will be written by parent and other one time by child process

**4. Now, consider the following program:**

**main() {**

**int fd;**

**fork();**

**fd = open(“outfile”, O\_RDWR)**

**write(fd, “hello”, 5);**

**exit();**

**}**

**Assume all system calls finish successfully on a uniprocessor system. Also, assume that a**

**system call cannot be interrupted in the middle of its execution. Notice that open is now called after**

**fork (not before fork as in the previous question). What will be the contents of the “outfile” file, after all**

**processes have successfully exited? Explain briefly.**

**Answer:**

in outfile there will be only hello because there will be two processes so first one process will write in to one file and other process will overwrite that file.