ASSIGNMENT-1

① Program to check and display _POSIX_VERS Constant of the system on which it is run.

Ans1
```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include <iostream.h>
#include <unistd.h>
int main()
{
#ifdef _POSIX_VERSION
cout<<"System supports to POSIX"<<"_POSIX_VERSION<<en
#else
cout<<"_POSIX_VERSION undefined\n";
#endif return 0;
}
```

Output

```
gedit version.cpp [To create a new file]
g++ version.cpp [To compile]
./a.out [To run].
System supports to POSIX 200809
```

---

② Program to print POSIX defined configuration options supported on any given system.

Ans2
```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include<stdio.h>
#include<unistd.h>
int main()
{
```

```
#ifdef _POSIX_JOB_CONTROL
    printf("System supports job control\n");
#else
    printf("System does not support job control\n");
#endif
#ifdef _POSIX_SAVED_IDS
    printf("System supports saved set-UID and saved
            set-GID\n");
#else
    printf("System does not support saved set-UID and
            saved set-GID\n");
#endif
#ifdef _POSIX_CHOWN_RESTRICTED
    printf("chown_restricted option is %d\n", _POSIX_CHOWN
            _RESTRICTED);
#else
    printf("System does not support chown_restricted option\n");
#endif
#ifdef _POSIX_NO_TRUNC
    printf("Pathname trunc option is %d\n", _POSIX_NO_TRUNC);
#else
    printf("System does not support system-wide pathname
            trunc option\n");
#endif
#ifdef _POSIX_VDISABLE
    printf("Disable character for terminal files is %d\n",
            _POSIX_VDISABLE);
#else
    printf("System does not support _POSIX_VDISABLE\n");
#endif
return 0;
}
```

Output

System supports job control
System supports saved set-UID and set-GID.
chown-restricted option is 1
Pathname trunc option is 1
Disable character for terminal files is 0.

③ Illustrate the use of sysconf, pathconf, fpathconf

Ans3 Prototypes :

```
#include<unistd.h>
Long sysconf(const int limit_name);
Long pathconf(const char *pathname, int flimit_name);
Long fpathconf(const int fd, int flimit_name);
```

// program to illustrate use of sysconf, pathconf & fpathconf

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include<stdio.h>
#include<iostream.h>
#include<unistd.h>
int main()
{
  int res;
  if((res=sysconf(_SC_OPEN_MAX))==-1)
      perror("sysconf");
  else
    cout<<"OPEN_MAX:"<<res<<endl;
if((res=pathconf("/",_PC_PATH_MAX))==-1)
      perror("pathconf");
```

```
else
cout<<"max path name :"<<(res+1)<<endl;
if((res=fpathconf(0, _PC_CHOWN_RESTRICTED))==-1)
    perror("fpathconf");
else
cout<<"chown_restricted for stdin:"<<res<<endl;
return 0;
}
```

Output

```
OPEN_MAX : 1024
Max pathname : 4097
chown_restricted for stdin : 1
```

---

④ Program to illustrate the System calls.

Ans 4 System calls

① open : It is used to open an existing file for data transfer function or else it may be also used to create a new file.

Prototype :

```
#include<sys/types.h>
#include<sys/fcntl.h>
int open(const char * pathname, int accessmode,
         mode_t permission);
```

Program to illustrate open API

```
#include<sys/types.h>
#include<stdio.h>
```

```
#include<fcntl.h>
#include<unistd.h>
#include<errno.h>
int main()
{
int fd = open("/home/vishal/sample.txt", O_RDONLY |
                                    O_CREAT);
printf("fd = %d\n", fd);
if(fd == -1)
{
printf("Error number %d\n", errno);
perror("program");
}
return 0;
}
```

// sample.txt is present in the directory

output

fd = 3.

---

② creat: It is used to create new regular files. creat function can be implemented using open function.

Prototype:

```
#include<sys/types.h>
#include<unistd.h>
int creat(const char *pathname, mode_t mode);
```

Program:

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
#include<errno.h>
int main()
{
// creat.txt is created in the directory
int fd = open("creat.txt", O_RONLY|O_CREAT);
printf("fd=%d\n", fd);
if(fd==-1)
{
printf("Error number %d\n", errno);
perror("program");
}
return 0;
}
```

Output

fd = 3

---

③ read : The read function fetches a fixed size of block of data from a file referenced by a given file descriptor.

Prototype :

```
#include<sys/types.h>
#include<unistd.h>
ssize_t read(int fdesc, void *buf, size_t
nbyte);
```

Program:

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
int main()
{
  int fd;
  char buffer[88];
  char msg[40]="Hello this is read system call";
  fd=open("read.txt", O_RDWR|O_CREAT);
  printf("fd=%d\n", fd);
  if(fd!=-1)
{
printf("File opened for read and write access\n");
read(fd, msg, sizeof(msg));
printf("Data read from the file is : %s\n", msg);
close(fd);
}
return 0;
}
```

Output

```
fd=3
File is opened for read and write access
Data read from the file is : Hello this is
read system call.
```

(4) write() : The write function puts a fixed size block of data to a file referenced by a file descriptor.

Prototype :
```
#include <sys/types.h>
#include <unistd.h>
size_t size_t write(int fdesc, const void* buf, size_t size);
```

Program :
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
int fd;
char buffer[80];
char msg[50]="hello this is file for write system call";
fd=open("write.txt", O_RDWR);
printf("fd=%d", fd);
if(fd!=-1)
{
printf("\n file is opened for read and write access\n");
write(fd, msg, sizeof(msg));
printf("Data written to the file is : %s", msg);
close(fd);
}
return 0;
}
```

Output

fd=3

file is opened for read & write access

Data written to the file is : hello this is file for write system call.

5) Lseek : The lseek system call can be used to change the file offset to a different value. It allows a process to perform random access of data on any opened file.

Prototype :
```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fdesc, off_t pos, int whence);
```

Program for copying contents of one file in reverse order in another file & displaying number of bytes.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
char buffer[1];
int main()
{
int fd1, fd2;
char file1[50], file2[50];
printf("Enter the file to reverse:\n");
gets(file1);
if((fd1=open("file1.txt", O_RDONLY))==-1)
{
printf("File does not exist\n");
return -1;
}
printf("Enter file to output in:\n");
gets(file2);
```

```
fd2 = open("file2.txt", O_RDWR|O_CREAT);
int size = lseek(fd1, 0, SEEK_END);
printf("The no of bytes read in file2 is: %d\n", size);
int i;
for(i=1; i<=size; i++)
{
if((lseek(fd1, -i, SEEK_END)) == -1)
{
printf("%s", strerror(errno));
}
int rd = read(fd1, buffer, 1);
int wr = write(fd2, buffer, 1);
if(wr == 0)
{
printf("Did not write to file\n");
return -1;
}
}
return 0;
}
```

file1.txt

→ HELLO

file2.txt(reverse)

→ OLLEH

output

```
Enter the file to reverse:
file1.txt
Enter the file to output in:
file2.txt
The number of bytes read in file2 is:
7
```

⑥ fcntl: The fcntl function helps a user to query or set flags & close on exec flag of any file descriptor.

Prototype: #include <fcntl.h>
int fcntl(int fdesc, int cmd, ---);

C program to Duplicate a file descriptor with fcntl().

```
→ #include<sys/types.h>
  #include<stdio.h>
  #include<fcntl.h>
int main(int argc, char **argv)
{
  int fd, nfd;
  if(argc<2)
{
printf("Usage: %s pathname\n", argv[0]);
  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("Fd %d duplicated with %d\n", fd, nfd);
 close(fd);
 close(nfd);
} return 0;
```

Output

$ gcc fcntl.c
$ ./a.out fcntl.txt
$ Fd 3 duplicated with 4.

(7) Link : The link function creates a new link for the existing file.

Prototype : #include <unistd.h>
int link(const char *cur_link, const char *new_link);

C program to demonstrate ln command using link API

```
#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];
sprintf(str, "ls -i %s %s\n", argv[1], argv[2]);
```

```
System(str);
return 0;
}
```

OUTPUT

```
$ gcc link.c
./a.out link1.txt link2.txt
Files linked
Inode number of linked files
799270 link1.txt 799270 link2.txt
```

(8) unlink : The unlink function deletes a link of an existing file. This function decreases the hard link count attributes of named file & removes the file name entry of link from directory file.

Prototype : `#include <unistd.h>`

`int unlink(const char* cur_link);`

C program to implement mv command using unlink API & rename a file.

→

```
#include <sys/types.h>
#include <Stdio.h>
#include <unistd.h>
#include <fcntl.h>.
int main(int argc, char **argv)
{
int fd1, fd2;
int n, count = 0;
fd1 = open(argv[1], O_RDONLY);
```

```
    fd2 = creat(argv[2], S_IWUSR);
    rename(fd1, fd2);
    unlink(argv[1]);
    close(fd1);
    close(fd2);
    printf("File renamed successfully\n");
    return 0;
}
```

unlink1.txt will be renamed to sample.txt

Output

```
./a.out unlink1.txt sample.txt
File renamed successfully.
```

⑨ Stat, fstat : This function is used to retrive file attributes of specifile file.

prototype : #include<unistd.h>

```
int stat(const char * path_name, struct
        Stat * stav);
Int fstat(int fdisc, struct stat* stav);
int Lstat(const char * pathname, struct stat*
        Stav);
```

```
Struct stat
{
 dev_ts t_dev;        time_t st_atime;
 ino_t  st_ino;       time_t st_mtime;
 mode_t st_mode;      time_t st_ctime;
 uid_t  st_uid;       }
 gid_t  st_gid;
 dev_t  st_rdev;
 off_t  st_size;
```

C program to know the type of file using Stat() & fstat() API:

```
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdio.h>
int main()
{
 int fd=0;
 struct stat st;
 fd=open("stat.txt", O_RDONLY);
  if(fd==-1)
{
 printf("Null file directory\n");
  return -1;
 }
 if(fstat(fd, &st))
{
 printf("fstat error\n");
  close(fd);
  return -1;
}
if(S_ISREG(st.st_mode))
{
printf("File is regular file\n");
}
```

```
else if(S_ISDIR(st.st_mode))
{
printf("File is directory file\n");
}
close(fd);
return 0;
}
```

→ stat.txt should be created.

Output

```
./a.out stat.txt
File is regular file.
```

(10) utime() : It is used to change access time & modification time of a file.

Prototype :

```
#include<sys/types.h>
#include<utime.h>
int utime(const char* path_name, Struct utimebuf
          * times);
Struct utimbuf
{
  time_t actime;
  time_t modtime;
};
```

C program to copy access & modification time of a file to another file using utime function.

```
#include<stdio.h>
#include<stdlib.h>
#include<utime.h>
#include<time.h>
```

```
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
int main(int argc, char *argv[])
{
 if(argc<3)
 {
  printf("Exiting");
  exit(0);
 }
printf("%s", argv[2]);
Struct stat mystat;
Stact utimbuf times;
Stat(argv[1], &mystat);
printf("\n atime = %s \n mtime = %s", ctime(&mystat.st_atime),
        ctime(&mystat.st_mtime));
Stat(argv[2], &mystat);
times.actime = mystat.st_atime;
times.modtime = mystat.st_mtime;
Utime(argv[1], &times);
Stat(argv[1], &mystat);
printf("\n atime = %s \n mtime = %s", ctime(&mystat.st_atime),
    ctime(&mystat.st_mtime));
 return 0;
}
```

OUTPUT

```
./a.out utime.c link.c
atime = Mon Apr 13 16:04:48 2020
mtime = Mon Apr 13 16:04:48 2020
atime = Sun Apr 12 21:20:24 2020
mtime = Sun Apr 12 21:20:24 2020
```