Q.1)Take multiple files as Command Line Arguments and print their inode numbers and file types

```
#include<stdio.h> #include<sys/stat.h>  #include<unistd.h>  #include<fcntl.h>
int main(int argc, char *argv[]) {
struct stat fileStat; if(argc!=3)
{          printf("Invalid number of arguments ");
return 1; }
int file1 = open(argv[1], O_RDONLY); if(file1 < 0)
{
fprintf(stderr, "error opening file1\n"); return 1;
}
int file2 = open(argv[2], O_RDONLY); if(file2 < 0)
{
fprintf(stderr, "error opening file2\n"); return 1;
}
if(fstat(file1,&fileStat)<0) return 1;
printf("File1 is:%s and Inode:%ld\n",argv[1],fileStat.st_ino);
if(fstat(file2,&fileStat)<0) return 1;
printf("File2 is:%s and Inode:%ld\n",argv[2],fileStat.st_ino); }
```

Q.2) Write a C program to send SIGALRM signal by child process to parent process and parent process make a provision to catch the signal and display alarm is fired.(Use Kill, fork, signal and sleep system call)

```
#include <signal.h> #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/wait.h>
void signalHandler(int signal)    {
if (signal == SIGALRM) { printf("Ding!\n");
wait(NULL);
} }
int main(int argc, char *argv[]) {
signal(SIGALRM, signalHandler); if (argc != 2)
{
printf("Invalid arguments\n"); return 0;
}
printf("Alarm application starting\n"); int delay;
sscanf(argv[1], "%d", &delay);   // compute delay if (fork() == 0)
{
printf("Waiting for alarm to go off\n"); sleep(delay);
kill(getppid(), SIGALRM); exit(0);
}
wait(NULL); printf("done\n");
}
```

Q.4) Write a C program that catches the ctrl-c (SIGINT) signal for the first time and display the appropriate message and exits on pressing ctrl-c again.

```
#include<stdio.h> #include<stdlib.h> #include<signal.h> #include<unistd.h> void sigint()
{
write(STDOUT_FILENO, "Press Ctrl + C once again to exit",1); signal(SIGINT, SIG_DFL);
}
void main() {
signal(SIGINT, sigint); while(1){
printf("Hello"); }
```

Q.3) Write a C program to find file properties such as inode number, number of hard link, File permissions, File size, File access and modification time and so on of a given file using stat() system call.

```
#include<stdio.h> #include<stdlib.h> #include<sys/stat.h> #include<sys/types.h> #include<time.h>
#include<fcntl.h>
int main(int argc, char const *argv[]) {
if(argc != 2) {
fprintf(stderr, "usage : %s <filepath>\n", argv[0]); return 1; }
int file = open(argv[1], O_RDONLY); if(file < 0)
{
fprintf(stderr, "error opening file\n");return 1; }
struct stat st; if(fstat(file, &st) < 0) {
fprintf(stderr, "error reading file info\n"); return 1;
}
printf("File Name is : %s \n", argv[1]);
printf("File size : %ld\n", st.st_size);
printf("Number of hard links : %d\n", st.st_nlink);
printf("File inode : %ld\n", st.st_ino);
printf("File Permissions : ");
printf(S_ISDIR(st.st_mode) ? "d" : "-");
printf((st.st_mode & S_IRUSR) ? "r" : "-");
printf((st.st_mode & S_IWUSR) ? "w" : "-");
printf((st.st_mode & S_IXUSR) ? "x" : "-");
printf((st.st_mode & S_IRGRP) ? "r" : "-");
printf((st.st_mode & S_IWGRP) ? "w" : "-");
printf((st.st_mode & S_IXGRP) ? "x" : "-");
printf((st.st_mode & S_IROTH) ? "r" : "-");
printf((st.st_mode & S_IWOTH) ? "w" : "-");
printf((st.st_mode & S_IXOTH) ? "x" : "-");
printf("\n");
char timestr[50];
struct tm *modified_time = localtime(&st.st_mtime); strftime(timestr, 80, "%b %d %I:%M %p",
modified_time); printf("Modified time : %s\n", timestr);struct tm *access_time =
localtime(&st.st_atime); strftime(timestr, 80, "%b %d %I:%M %p", access_time); printf("Access time :
%s\n", timestr);
return 0; }
```

Q.8) Write a C program that redirects standard output to a file output.txt. (use of dup and open system call).

```
#include<stdio.h> #include<stdlib.h> #include<fcntl.h> #include<unistd.h> void main()
{ int fd;
fd = open("output.txt",O_CREAT| O_WRONLY, 0777); close(STDOUT_FILENO);
dup(fd);
printf("this is some text to be printed on the screen\n"); printf("but it will be written to the file
output.txt\n"); }
```

Q.5) Print the type of file and inode number where file name accepted through Command Line

```
#include<stdio.h> #include<stdlib.h> #include<sys/stat.h> #include<sys/types.h> #include<time.h>
#include<fcntl.h>
int main(int argc, char const *argv[]) {
if(argc != 2){
fprintf(stderr, "usage : %s <filepath>\n", argv[0]); return 1;
}
int file = open(argv[1], O_RDONLY); if(file < 0)
{
fprintf(stderr, "error opening file\n"); return 1; }
struct stat st; if(fstat(file, &st) < 0) {
fprintf(stderr, "error reading file info\n"); return 1;
}
printf("File Name is %s  and ", argv[1]);
if( S_ISREG(st.st_mode) ) printf("This is Regular file\n");
if( S_ISDIR(st.st_mode) ) printf("This is Directory file\n");
if( S_ISCHR(st.st_mode) )
printf("This is Chracter Special file\n"); if( S_ISBLK(st.st_mode) )
printf("This is Block Special file\n");
if( S_ISFIFO(st.st_mode) )printf("This is Pipe or FIFO file\n");
 if( S_ISLNK(st.st_mode) )
printf("This is Symbolic file\n"); if( S_ISSOCK(st.st_mode) )
printf("This is Socket file\n"); return 0;
}
```

Q.7) Write a C program to get and set the resource limits such as files, memory associated with a Process

```
#include <stdio.h>#include <sys/resource.h>#include <string.h>#include <errno.h> #include
<unistd.h>#include <sys/types.h>#include <sys/stat.h>#include <fcntl.h>
int main() {
struct rlimit old_lim, lim, new_lim; // Get old limits
if( getrlimit(RLIMIT_NOFILE, &old_lim) == 0)
printf("Old limits -> soft limit= %ld \t" " hard limit= %ld \n",old_lim.rlim_cur,
old_lim.rlim_max);
else
fprintf(stderr, "%s\n", strerror(errno));// Set new value
lim.rlim_cur = 5;
lim.rlim_max = 1024;
// Set limits
if(setrlimit(RLIMIT_NOFILE, &lim) == -1)
fprintf(stderr, "%s\n", strerror(errno)); // Get new limits
if( getrlimit(RLIMIT_NOFILE, &new_lim) == 0)
printf("New limits -> soft limit= %ld " "\t hard limit= %ld \n",
new_lim.rlim_cur,new_lim.rlim_max);
else
fprintf(stderr, "%s\n", strerror(errno)); return 0;
}
```

Q.6) Write a C program which creates a child process to run linux/ unix command or any user defined program. The parent process set the signal handler for death of child signal and Alarm signal. If a child process does not complete its execution in 5 second then parent process kills child process.

```
#include <stdio.h> #include <signal.h> #include <stdlib.h> #include <unistd.h> #include <sys/types.h>
// function declaration of sighup, sigint and sigquit functions
void sighup();
void sigint();
void sigquit();
// main function or driver code void main()
{
int pid;
// pid variable, which will be used later to identify the process, whether it is child process or parent process
// to get the child process if ((pid = fork()) < 0)
{perror("fork"); exit(1);
}
if (pid == 0)
{           /* child process, since pid equals to zero for child process */
signal(SIGHUP, sighup);
signal(SIGINT, sigint);
signal(SIGQUIT, sigquit);
for (;;);   /* infinite loop i.e. loop for ever  */ }
else      /* parent process*/
{ // pid hold the process id of child process printf("\nPARENT: sending SIGHUP\n\n"); kill(pid, SIGHUP);
sleep(3); // pause for 3 seconds printf("\nPARENT: sending SIGINT\n\n"); kill(pid, SIGINT);
sleep(3); // pause for 3 seconds
printf("\nPARENT: Waiting for 5 Second then kill child\n\n"); printf("\nPARENT: sending SIGQUIT\n\n");
kill(pid, SIGQUIT);
sleep(5); // pause for 5 seconds }
}
// function definition of sighup() void sighup()
{
signal(SIGHUP, sighup); /* reset signal */ printf("CHILD: I have received a SIGHUP\n");}
// function definition of sigint() void sigint()
{
signal(SIGINT, sigint); /* reset signal */ printf("CHILD: I have received a SIGINT\n");
}
// function definition of sigquit() void sigquit()
{
printf("My Papa has Killed me!!!\n"); exit(0);
}
```

Q.9) Write a C program to find whether a given files passed through command line arguments are present in current directory or not.

```
#include <stdio.h> #include <unistd.h>
int main(int argc, char *argv[]) {
if(access(argv[1],F_OK)==0)
printf("File %s exists in current directory \n", argv[1]); else
printf("File %s doesn't exist in current directory \n", argv[1]); return 0;}
```

Q.10) Write a C program which creates a child process and child process catches a signal SIGHUP, SIGINT and SIGQUIT. The Parent process send a SIGHUP or SIGINT signal after every 3 seconds, at the end of 15 second parent send SIGQUIT signal to child and child terminates by displaying message "My Papa has Killed me!!!".

```
#include <stdio.h>
#include <signal.h> #include <stdlib.h> #include <unistd.h> #include <sys/types.h>
// function declaration of sighup, sigint and sigquit functions
void sighup();
void sigint();
void sigquit();
// main function or driver code void main()
{
int pid;
// pid variable, which will be used later to identify the process, whether it is child process or parent process
// to get the child process if ((pid = fork()) < 0)
{perror("fork"); exit(1);
}
if (pid == 0)
{           /* child process, since pid equals to zero for child process */
signal(SIGHUP, sighup);
signal(SIGINT, sigint);
signal(SIGQUIT, sigquit);
for (;;);   /* infinite loop i.e. loop for ever  */ }
else      /* parent process*/
{ // pid hold the process id of child process printf("\nPARENT: sending SIGHUP\n\n"); kill(pid, SIGHUP);
sleep(3); // pause for 3 seconds printf("\nPARENT: sending SIGINT\n\n");
kill(pid, SIGINT);
sleep(3); // pause for 3 seconds
printf("\nPARENT: sending SIGQUIT\n\n"); kill(pid, SIGQUIT);
sleep(3); // pause for 3 seconds }
}
// function definition of sighup() void sighup()
{signal(SIGHUP, sighup); /* reset signal */ printf("CHILD: I have received a SIGHUP\n");
}
// function definition of sigint() void sigint(){
signal(SIGINT, sigint); /* reset signal */ printf("CHILD: I have received a SIGINT\n");
}
// function definition of sigquit() void sigquit()
{
printf("My Papa has Killed me!!!\n"); exit(0);}
```

Q.11) Read the current directory and display the name of the files, no of files in current directory

```
#include<stdio.h> #include<dirent.h> int main()
{
DIR *d; int cnt=0;
struct dirent *dir; // pointer for directory entry d=opendir(".");
if(d==NULL) {
printf("Could not open the current directory"); return(0);
}
while((dir=readdir(d))!=NULL)
{
printf("%s\n",dir->d_name); cnt++;
}printf("\nTotal no. of files in the current directory=%d\n",cnt); closedir(d);
return 0;
}
```

Q.12) Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it.
Message1 = "Hello World" Message2 = "Hello SPPU"
Message3 = "Linux is Funny"

```
#include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <string.h> #include <sys/wait.h>
int main() {
int pipe_fd[2];
38pid_t child_pid;
if (pipe(pipe_fd) == -1) { perror("Pipe creation failed"); exit(EXIT_FAILURE);
}
child_pid = fork();
if (child_pid == -1) { perror("Fork failed"); exit(EXIT_FAILURE);
}
if (child_pid == 0) {  // Child process
close(pipe_fd[0]); // Close the read end of the pipe
char *message1 = "Hello World";
char *message2 = "Hello SPPU";
char *message3 = "Linux is Funny";
write(pipe_fd[1], message1, strlen(message1) + 1);
write(pipe_fd[1], message2, strlen(message2) + 1);
write(pipe_fd[1], message3, strlen(message3) + 1);
close(pipe_fd[1]); // Close the write end of the pipe
39exit(EXIT_SUCCESS); } else {  // Parent process
close(pipe_fd[1]); // Close the write end of the pipe char buffer[100];
while (read(pipe_fd[0], buffer, sizeof(buffer)) > 0) {
printf("Received message: %s\n", buffer); }
close(pipe_fd[0]); // Close the read end of the pipe wait(NULL); // Wait for the child process to
complete
} return 0;
}
```

Q.13) Display all the files from current directory which are created in particular month

```
#include<stdio.h>#include<dirent.h> #include<string.h> #include<sys/stat.h> #include<time.h>
#include<stdlib.h>
int main(int argc, char *argv[]) {
char in[100],st[100],*ch,*ch1,c,buff[512]; DIR *dp;
int i;
struct dirent *ep; struct stat sb; char mon[100]; dp=opendir("./"); if (dp != NULL) {
while(ep =readdir(dp)) {
if(stat(ep->d_name,&sb) == -1) {
perror("stat");exit(EXIT_SUCCESS); }
strcpy(mon,ctime(&sb.st_ctime)); ch=strtok(mon," ");
ch=strtok(NULL,","); ch1=strtok(ch," ");
if((strcmp(ch1,argv[1]))==0) {
printf("%s\t\t%s",ep->d_name,ctime(&sb.st_ctime)); }}
(void)closedir(dp); }
return 0; }
```

Q.14) Write a C program to create n child processes. When all n child processes terminates, Display total cumulative time children spent in user and kernel mode

```
#include<sys/types.h> #include<sys/wait.h> #include<unistd.h> #include<time.h>
#include<sys/times.h> #include<stdio.h>#include<stdlib.h> int main(void) {
int i, status; //pid_t data type is signed interger type repesenting process ID pid_t pid; //time_ t data
type used to storeing system time value
time_t currentTime;
//times() stores the current process time in the struct tms that buffer points to. struct tms cpuTime;
if((pid = fork())==-1) //start child process {
perror("\nfork error"); exit(EXIT_FAILURE); }
else if(pid==0) //child process {
time(&currentTime); // gives normal time
printf("\nChild process started at %s",ctime(&currentTime)); for(i=0;i<5;i++) {
printf("\nCounting= %dn",i); //count for 5 seconds sleep(1); }
time(&currentTime);
printf("\nChild process ended at %s",ctime(&currentTime)); exit(EXIT_SUCCESS);
} else
{ //Parent process time(&currentTime);
printf("\nParent process started at %s ",ctime(&currentTime)); if(wait(&status)== -1) //wait for child
processperror("\n wait error"); if(WIFEXITED(status))
printf("\nChild process ended normally.....\n");
else
printf("\nChild process did not end normally"); if(times(&cpuTime)<0) //Get process time
perror("\nTimes error");
else
{ // _SC_CLK_TCK: system configuration time: seconds clock tick
printf("\nParent process user time= %fn",((double) cpuTime.tms_utime));
printf("\nParent process system time = %fn",((double) cpuTime.tms_stime));
printf("\nChild process user time = %fn",((double) cpuTime.tms_cutime));
printf("\nChild process system time = %fn",((double) cpuTime.tms_cstime));}
time(&currentTime);
printf("\nParent process ended at %s",ctime(&currentTime)); exit(EXIT_SUCCESS);
} }
```

Q.15) Implement the following unix/linux command (use fork, pipe and exec system call)
ls –l | wc –l

```
#include<stdio.h>
#include<stdlib.h> #include<unistd.h> #include<fcntl.h> #include<sys/wait.h> #include<errno.h>
void main(){
int filedes[2];
if (pipe(filedes) == -1) {
perror("pipe"); exit(1);
}
if(fork() == 0) {
while ((dup2(filedes[1], STDOUT_FILENO) == -1)) {} char *args[] = {"ls","-l", NULL};
int ret = execvp("ls",args); if(ret <0)
{
printf("Program can't be executed\n"); }
exit(0); }
close(filedes[1]); if(fork() == 0) {
while((dup2(filedes[0], STDIN_FILENO) == -1)){} char *args[] = {"wc","-l", NULL};
int ret = execvp("wc",args); if(ret <0)
{
printf("Program can't be executed\n"); }
exit(0); }
char output[100];read(filedes[0], output, 100); printf("%s", output); close(filedes[0]);
exit(0); }
```

Q.16) Write a program that illustrates how to execute two commands concurrently with a pipe.

```
#include <stdio.h> #include <unistd.h> #include <sys/types.h> #include <stdlib.h> int main()
{
int pfds[2]; char buf[80]; if(pipe(pfds)==-1){
perror("pipe failed"); exit(1);
}
if(!fork())
{
close(1); dup(pfds[1]); system ("ls -l"); }
else {
printf("parent reading from pipe \n"); while(read(pfds[0],buf,80))
printf("%s \n" ,buf); }
}
```

Q.20) Write a C program that print the exit status of a terminated child process

```
#include<stdio.h>
#include<unistd.h> #include<sys/types.h> #include<stdlib.h> int main()
{
int pid; pid=fork(); if (pid<0) {
printf("Fork Failed \n");exit(1); }
else if(pid==0) {
execlp("/bin/ls","ls","-l",NULL); // Execute ls }
else {
wait(NULL);
printf("\nChild Complete"); exit(0);
}
}
```

Q.18) Generate parent process to write unnamed pipe and will read from it

```
#include<stdio.h>
#include<unistd.h> int main() {
int pipefds[2];
int returnstatus;
int pid;
char writemessages[1][20]={"Hello"}; char readmessage[20];
returnstatus = pipe(pipefds); if (returnstatus == -1)
{
printf("Unable to create pipe\n"); return 1;
}
pid = fork();
// Child process if (pid == 0)
{
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Child Process - Reading from pipe – Message is %s\n", readmessage); }
else
{ //Parent process
printf("Parent Process - Writing to pipe - Message is %s\n", writemessages[0]);write(pipefds[1],
writemessages[0], sizeof(writemessages[0])); }
return 0;
}
```

Q.19) Write a C program to Identify the type (Directory, character device, Block device, Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call.

```
#include<stdio.h> #include<stdlib.h> #include<sys/stat.h> #include<sys/types.h>#include<time.h>
#include<fcntl.h>
int main(int argc, char const *argv[]) {
if(argc != 2){
fprintf(stderr, "usage : %s <filepath>\n", argv[0]); return 1;
}
int file = open(argv[1], O_RDONLY); if(file < 0){
fprintf(stderr, "error opening file\n"); return 1;
}
struct stat st; if(fstat(file, &st) < 0) {
fprintf(stderr, "error reading file info\n"); return 1;
}
printf("File Name is %s : \n", argv[1]); printf("File Type: ");
switch (st.st_mode & S_IFMT) {
case S_IFBLK:  printf("this block device\n");          break;
case S_IFCHR:  printf("this character device\n");      break;
case S_IFDIR:  printf("this directory\n");             break;
case S_IFIFO:  printf("this FIFO/pipe\n");             break;
case S_IFLNK:  printf("this symlink\n");               break;
case S_IFREG:  printf("this is regular file\n");       break;
case S_IFSOCK: printf("this socket\n");                break;
default:       printf("unknown?\n");
break;}
return 0; }
```

Q.21) Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes. I) (e.g $ a.out a.txt b.txt c.txt, …)

```
#include <stdio.h>
#include <dirent.h>
#include<string.h>
#include<unistd.h>
#include<time.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<stdlib.h>
typedef struct file_info {
char *name; size_t size; }fileinfo;
void insertionSort(fileinfo info[], int n) {
int i, j;
fileinfo key;
for (i = 1; i < n; i++) {
key = info[i]; j = i - 1;
while (j >= 0 && info[j].size > key.size) {
info[j + 1] = info[j]; j = j - 1;
}
info[j + 1] = key; }}
void main(int argc, char **argv) {
struct stat fstat; if(argc < 3)
{
printf("No files passed\n"); exit(1);
}
int fileCount = argc -1; fileinfo info[fileCount]; int i;
printf("Display all filenames in ascending order according to their sizes.\n"); for(i =1;i<argc;i++)
{
info[i-1].name = argv[i]; stat(argv[i],&fstat); info[i-1].size = fstat.st_size; }
insertionSort(info, fileCount); for(i=0;i<fileCount;i++)
{
printf("%s -> %ld\n",info[i].name, info[i].size); }
}
```

Q.23) Write a C program that a string as an argument and return all the files that begins with that name in the current directory. For example > ./a.out foo will return all file names that begins with foo

```
#include<stdio.h> #include<dirent.h> #include<string.h>
int main(int argc, char* argv[]) {DIR *d;
char *position; struct dirent *dir; int i=0;
if(argc!=2) {
printf("Provide suffiecient args"); }
else {
d = opendir("."); if (d)
{
while ((dir = readdir(d)) != NULL) {
position=strstr(dir->d_name,argv[1]); i=position-dir->d_name;
if(i==0)
printf("%s\n",dir->d_name); }
closedir(d); return(0); }
} }
```

Q.22) Write a C program that illustrates suspending and resuming processes using signals

```
#include <signal.h>
55#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
volatile int child_suspended = 0; void suspend_handler(int signum) {
if (signum == SIGUSR1) { kill(getpid(), SIGSTOP); child_suspended = 1; }
}
void resume_handler(int signum) { if (signum == SIGUSR2) { kill(getpid(), SIGCONT);
child_suspended = 0; }
}
int main() {
pid_t child_pid;
signal(SIGUSR1, suspend_handler); signal(SIGUSR2, resume_handler);
56child_pid = fork(); if (child_pid == -1) {
perror("Fork failed"); exit(EXIT_FAILURE); }
if (child_pid == 0) { while (1) {
if (!child_suspended) { printf("Child: Running...\n"); sleep(1);
} else {
printf("Child: Suspended.\n"); pause();
}
}
} else { sleep(2);
kill(child_pid, SIGUSR1); sleep(3);
kill(child_pid, SIGUSR2); wait(NULL);
printf("Parent: Exiting.\n"); }
57return 0; }
```

Q.26) Display all the files from current directory whose size is greater that n Bytes Where n is accept from user

```
#include <stdio.h>#include <dirent.h>#include<string.h>#include<unistd.h>#include<time.h>
#include<sys/stat.h>#include<sys/types.h>#include<stdlib.h>
void main(int argc, char **argv) {
struct dirent *de;
struct stat fstat;
struct tm *timeinfo; if(argc != 2){
printf("no size value passed\n"); exit(1);
}
int size = atoi(argv[1]); if(size <0){
printf("invalid size value : size should be non negative\n"); exit(1);
}
DIR *directory = opendir("."); char **filenames;
if (directory == NULL) {
printf("Could not open current directory" ); return;
}while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,"..")) {
stat(de->d_name,&fstat); if(fstat.st_size > size) {
printf("%s\t %ld\n",de->d_name,fstat.st_size ); }
}
closedir(directory); }
```

Q.24) Write a C program to find file properties such as inode number, number of hard link, File permissions, File size, File access and modification time and so on of a given file using stat() system call.

```
#include<stdio.h> #include<stdlib.h> #include<sys/stat.h> #include<sys/types.h> #include<time.h>
#include<fcntl.h>
int main(int argc, char const *argv[]) {
if(argc != 2) {
fprintf(stderr, "usage : %s <filepath>\n", argv[0]); return 1;
}
int file = open(argv[1], O_RDONLY); if(file < 0)
{
fprintf(stderr, "error opening file\n");return 1; }
struct stat st; if(fstat(file, &st) < 0) {
fprintf(stderr, "error reading file info\n"); return 1;
}
printf("File Name is : %s \n", argv[1]);
printf("File size : %ld\n", st.st_size);
printf("Number of hard links : %d\n", st.st_nlink);
printf("File inode : %ld\n", st.st_ino);
printf("File Permissions : ");
printf(S_ISDIR(st.st_mode) ? "d" : "-");
printf((st.st_mode & S_IRUSR) ? "r" : "-");
printf((st.st_mode & S_IWUSR) ? "w" : "-");
printf((st.st_mode & S_IXUSR) ? "x" : "-");
printf((st.st_mode & S_IRGRP) ? "r" : "-");
printf((st.st_mode & S_IWGRP) ? "w" : "-");
printf((st.st_mode & S_IXGRP) ? "x" : "-");
printf((st.st_mode & S_IROTH) ? "r" : "-");
printf((st.st_mode & S_IWOTH) ? "w" : "-");
printf((st.st_mode & S_IXOTH) ? "x" : "-");
printf("\n");
char timestr[50];
struct tm *modified_time = localtime(&st.st_mtime); strftime(timestr, 80, "%b %d %l:%M %p",
modified_time); printf("Modified time : %s\n", timestr);struct tm *access_time =
localtime(&st.st_atime); strftime(timestr, 80, "%b %d %l:%M %p", access_time); printf("Access time :
%s\n", timestr);
return 0; }
```

Q.29) Write a C Program that demonstrates redirection of standard output to a file

```
#include<stdlib.h> #include<stdio.h> #include<string.h>
main(int argc, char *argv[]) {
char d[50]; if(argc==2) {
bzero(d,sizeof(d));
strcat(d,"ls ");
strcat(d,"> ");
strcat(d,argv[1]); system(d);
}
else
printf("\nInvalid No. of inputs"); }
```

Q.25) Write a C program which create a child process which catch a signal sighup, sigint and sigquit. The Parent process send a sighup or sigint signal after every 3 seconds, at the end of 30 second parent send sigquit signal to child and child terminates my displaying message "My DADDY has Killed me!!!".

```
#include <stdio.h>
#include <signal.h> #include <stdlib.h> #include <unistd.h> #include <sys/types.h>
// function declaration of sighup, sigint and sigquit functions
void sighup();
void sigint();
void sigquit();
// main function or driver code void main()
{
int pid;
// pid variable, which will be used later to identify the process, whether it is child process or parent process
// to get the child process if ((pid = fork()) < 0)
{perror("fork"); exit(1);
}
if (pid == 0)
{           /* child process, since pid equals to zero for child process */
signal(SIGHUP, sighup);
signal(SIGINT, sigint);
signal(SIGQUIT, sigquit);
for (;;);   /* infinite loop i.e. loop for ever  */ }
else      /* parent process*/
{ // pid hold the process id of child process printf("\nPARENT: sending SIGHUP\n\n"); kill(pid, SIGHUP);
sleep(3); // pause for 3 seconds printf("\nPARENT: sending SIGINT\n\n");
kill(pid, SIGINT);
sleep(3); // pause for 3 seconds
printf("\nPARENT: sending SIGQUIT\n\n"); kill(pid, SIGQUIT);
sleep(3); // pause for 3 seconds }
}
// function definition of sighup() void sighup()
{signal(SIGHUP, sighup); /* reset signal */ printf("CHILD: I have received a SIGHUP\n");
}
// function definition of sigint() void sigint()
{
signal(SIGINT, sigint); /* reset signal */ printf("CHILD: I have received a SIGINT\n");
}
// function definition of sigquit() void sigquit()
{
printf("My Papa has Killed me!!!\n"); exit(0);
}
```

Q.30) Write a C program to find whether a given file is present in current directory or not

```
#include<stdio.h> #include<unistd.h> int main(int argc, char *argv[]) {
 if(access(argv[1],F_OK)==0)
printf("File %s exists in current directory \n", argv[1]);
 else printf("File %s doesn't exist in current directory \n", argv[1]); return 0; }
```

Q.17) Generate parent process to write unnamed pipe and will write into it. Also generate child process which will read from pipe

```
#include <stdio.h>#include <stdlib.h> #include <unistd.h>#define BUFFER_SIZE 25
int main() {
   int pipe_fd[2]; // File descriptors for the pipe
   if (pipe(pipe_fd) == -1) {
      perror("Pipe creation failed");
      exit(EXIT_FAILURE);  }
  pid_t pid = fork();
if (pid < 0) {
 perror("Fork failed");
 exit(EXIT_FAILURE);
} if (pid > 0) {
      // Parent process
 close(pipe_fd[0]); // Close the read end of the pipe in the parent
char message[] = "Hello from the parent!";
  write(pipe_fd[1], message, sizeof(message));
close(pipe_fd[1]); // Close the write end of the pipe in the parent
   } else { // Child process
      close(pipe_fd[1]); // Close the write end of the pipe in the child
char buffer[BUFFER_SIZE];
      ssize_t bytesRead = read(pipe_fd[0], buffer, sizeof(buffer));
 if (bytesRead > 0) {
         printf("Child received: %s\n", buffer);
      } else { perror("Read failed in the child process");
      } close(pipe_fd[0]); // Close the read end of the pipe in the child
   } return 0;}
```

Q.27) Write a C program to implement the following unix/linux command (use fork, pipe and exec system call). Your program should block the signal Ctrl-C and Ctrl-\ signal during the execution.
i. Ls –l | wc –l

```
#include <stdio.h> #include <stdlib.h>#include <unistd.h>#include <signal.h>
void sigint_handler(int signum) {
   printf("Ctrl-C is blocked during execution.\n"); }
void sigquit_handler(int signum) {
   printf("Ctrl-\\ is blocked during execution.\n");}     int main() {
 // Block Ctrl-C and Ctrl-\
 signal(SIGINT, sigint_handler);  signal(SIGQUIT, sigquit_handler);
int pipe_fd[2]; if (pipe(pipe_fd) == -1) {
perror("Pipe creation failed"); exit(EXIT_FAILURE); }
pid_t pid = fork(); if (pid < 0) {
perror("Fork failed");exit(EXIT_FAILURE);  }          if (pid > 0) {
close(pipe_fd[0]); // Close the read end of the pipe in the parent
dup2(pipe_fd[1], STDOUT_FILENO);
close(pipe_fd[1]);          execlp("ls", "ls", "-l", NULL);
perror("Exec failed in the parent process");         exit(EXIT_FAILURE);       } else {
close(pipe_fd[1]); // Close the write end of the pipe in the child
dup2(pipe_fd[0], STDIN_FILENO);          close(pipe_fd[0]);
execlp("wc", "wc", "-l", NULL);    perror("Exec failed in the child process");
exit(EXIT_FAILURE);} return 0; }
```

28) Write a C program to Identify the type (Directory, character device, Block device, Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call

```
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>

void identifyFileType(const char *filename) {
   struct stat fileStat;

   if (stat(filename, &fileStat) == -1) {
      perror("Error in stat");
      return;
   }

   if (S_ISDIR(fileStat.st_mode)) {
      printf("%s is a directory.\n", filename);
   } else if (S_ISCHR(fileStat.st_mode)) {
      printf("%s is a character device.\n", filename);
   } else if (S_ISBLK(fileStat.st_mode)) {
      printf("%s is a block device.\n", filename);
   } else if (S_ISREG(fileStat.st_mode)) {
      printf("%s is a regular file.\n", filename);
   } else if (S_ISFIFO(fileStat.st_mode)) {
      printf("%s is a FIFO or pipe.\n", filename);
   } else if (S_ISLNK(fileStat.st_mode)) {
      printf("%s is a symbolic link.\n", filename);
   } else if (S_ISSOCK(fileStat.st_mode)) {
      printf("%s is a socket.\n", filename);
   } else {
      printf("%s is of unknown type.\n", filename);
   }
}

int main(int argc, char *argv[]) {
   if (argc != 2) {
      printf("Usage: %s <filename>\n", argv[0]);
      return 1;
   }

   identifyFileType(argv[1]);

   return 0; }
```