Slip 1

//Takes multiple files as Command Line Arguments and print their inode //number

```
#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);
}

//OUTPUT
//scos@localhost:~/aos$ ./a.out ass1.c hole.txt
//File1 is:ass1.c and Inode:5117221
//File2 is:hole.txt and Inode:5117205
```

Slip 2

```
//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;
}

/*OUTPUT
scos@localhost:~/aos$ gcc ass3.c
scos@localhost:~/aos$ ./a.out ass5.c
File Name is : ass5.c
File size : 483
Number of hard links : 1
File inode : 5117212
File Permissions : -rw-r--r--
Modified time : Oct 13  4:45 PM
Access time : Oct 13  4:45 PM
*/
```

Slip 3

//Assignment 4 : 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;
}

/*
The ls command helps you to identify and classify all kind of the file types found on a Linux system.
scos@localhost:~$ ls -l
-rw-------.  Regular file
drwxr-xr-x.  Directory file
```

lrwxrwxrwx. symboilc link file

crw-rw----. Chracter Special file

brw-rw----. Block Special file

srw-rw-rw-   Socket file

prw-------. Pipe or FIFO file

OUTPUT

scos@localhost:~/aos$ gcc ass4.c

scos@localhost:~/aos$ ./a.out demo

File Name is demo and This is Directory file

scos@localhost:~/aos$ ./a.out ass4.c

File Name is ass4.c and This is Regular file

*/

Slip 4

//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;
```

```
}

/* OUTPUT
scos@localhost:~/aos$ gcc ass5.c
scos@localhost:~/aos$ ./a.out ass11.c
File ass11.c doesn't exist in current directory
*/
```

Slip 5

//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;
}



/* OUTPUT
scos@localhost:~/aos$ gcc ass7.c
scos@localhost:~/aos$ ./a.out
.
ass3.c
ass10.c
hole.txt
ass7.c
a.out
ass6.c
..
demo
ass9.c
ass1.c
ass4.c
ass2.c
ass5.c
ass8.c

Total no. of files in the current directory=15
*/
```

Slip 6

//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;
}

/*Output :
[root@localhostUnix]# cc month.c
[root@localhostUnix]# ./a.out Mar
a.out Fri Mar 20 22:15:23 2020
. Fri Mar 20 22:15:23 2020
.. Fri Mar 20 22:14:29 2020
*/
```

Slip 7

//Write a C Program that demonstrates redirection of standard output to a file.

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
int 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");
}

/* OUTPUT
scos@localhost:~/aos$ gcc ass11.c

scos@localhost:~/aos$ ls >f1
 create file f1 where list files in current directory
*/
```

Slip 8

//Write a C program that redirects standard output to a file output.txt.

```
#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");
}
```

Slip 9

//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 â€“ 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;
}
```

Slip 10

//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);
}
}
```

Slip 11

//Assignment 22 : 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;

}
```

Slip 12

// Assignment 24 : 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);
        }
}
```

Slip 13

// Assignment 28 : Write a C program that illustrates suspending and resuming processes using signals

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

int main ()
{
 int pid1;
 int pid2;

 pid1 = fork();
 if (pid1 == 0) /* First child */
 {
 while (1) /* Infinite loop */
 {
 printf ("P1 is alive\n");
 sleep (1);
 }
 }

 pid2 = fork (); /* Second child */
 if (pid2 == 0)
 {
 while (1) /* Infinite loop */
 {
 printf ("P2 is alive\n");
```

```
 sleep (1);
 }
 }

 sleep (3);
 kill (pid1, SIGSTOP); /* Suspend first child */

 sleep (3);
 kill (pid1, SIGCONT); /* Resume first child */

 sleep (3);
 kill (pid1, SIGINT); /* Kill first child */
 kill (pid2, SIGINT); /* Kill second child */
}
```

Slip 14

//Assignment 10: 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\n",de->d_name );
}
}
closedir(directory);
}


/*Output
[root@localhostUnix]# cc month.c
[root@localhostUnix]# ./a.out Mar
a.out Fri Mar 20 22:15:23 2020
. Fri Mar 20 22:15:23 2020
.. Fri Mar 20 22:14:29 2020
*/
```