System calls simulation using C Programs

EX.NO.3 **SYSTEM CALLS OF UNIX OPERATING SYSTEM**

(a) Stat:

AIM :

To Execute a Unix Command in a ‘C’ program using stat() system call.

ALGORITHM:

1. Start the program
2. Declare the variables for the structure stat
3. Allocate the size for the file by using malloc function
4. Get the input of the file whose statistics want to be founded
5. Repeat the above step until statistics of the files are listed
6. Stop the

program. 7.

PROGRAM:

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

int main(void)

{
```

```
char *path,path1[10];
struct stat *nfile;

nfile=(struct stat *) malloc (sizeof(struct stat)); printf("enter
name of file whose stsistics has to"); scanf("%s",path1);

stat(path1,nfile);

printf("user id %d\n",nfile->st_uid); printf("block
size :%d\n",nfile->st_blksize); printf("last access
time %d\n",nfile->st_atime);

printf("time of last modification %d\n",nfile->st_atime);
printf("porduction mode %d \n",nfile->st_mode);
printf("size of file %d\n",nfile->st_size); printf("nu,mber of
links:%d\n",nfile->st_nlink);

}
```

OUTPUT:

```
enter name of file whose stsistics has to
stat.c user id 621

block size :4096

last access time 1145148485

time of last modification 1145148485
porduction mode 33204

size of file 654
nu,mber of
links:1
```

Result:

Thus the program for stat system call has been executed successfully.

(b) Wait:

AIM :

To Execute a Unix Command in a ‘C’ program using wait() system call.

AlLGORITHM :

1. Start the program
2. Initialize the necessary variables
3. Use wait() to return the parent id of the child else return -1 for an error
4. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
int main(void)

{
int pid,status,exitch;
if((pid=fork())==-1)

{
perror("error");
exit (0);

}

if(pid==0)

{

sleep(1);

printf("child process");
exit (0);

}
else

{

printf("parent process\n");
if((exitch=wait(&status))==-1)

{

perror("during wait()");
exit (0);

}

printf("parent existing\n");
exit (0);

}

}
```

OUTPUT :

```
parent process
child processparent existing
```

(c) GETPID:

AIM:

To Execute a Unix Command in a ‘C’ program using getpid() system call.

ALGORITHM:

1. Start the program
2. Declare the necessary variables
3. The getpid() system call returns the process ID of the parent of the
4. calling process
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()

{

int pid;
pid=getpid();

printf("process ID is %d\
n",pid); pid=getppid();

printf("parent process ID id %d\n",pid);

}
```

OUTPUT:

```
process ID is 2848
parent process ID id 2770
```

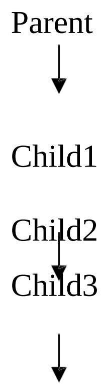
RESULT:

Thus the program for getpid system call has been executed successfully.

(d) Fork:

AIM:

To create a process in the following hierarchy



ALGORITHM:

1. Declare the necessary variables.
2. Parent process is the process of the program which is running.
3. Create the child1 process using fork() When parent is active.
4. Create the child2 process using fork() when child1 is active.
5. Create the child3 process using fork() when child2 is active.

PROGRAM:

```
#include<stdio.h>
int main(void)
{
int fork(void),value;
value=fork();
printf("main:value =%d\n",value);
return 0;
}
```

Output:

```
main:value =0
```

```
main:value =2860
```

Result:

Thus the program for fork system call has been executed successfully.

(e) Exec:

To Execute a Unix Command in a ‘C’ program using exec() system call.

AIM:

ALGORITHM:

1. Start the program
2. Declare the necessary variables
3. Use the prototype execv (filename,argv) to transform an executable binary file into process
4. Repeat this until all executed files are displayed
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
main()
{
     int pid;

     char *args[]={"/bin/ls","-l",0};
     printf("\nParent Process");
     pid=fork();

     if(pid==0)

     {



        }
            }
```

Out put:

```
            else
            {
```

```
m

            e      bin/ls",args); printf("\
            x      nChild process");
   }        e
            c
            v
            (
            "      wait();
            /      printf("\nParent process");
                   exit(0);
```

```
total 440
-rwxrwxr-x 1 skec25 skec25 5210 Apr 16 06:25 a.out
-rw-rw-r-- 1 skec25 skec25  775 Apr  9 08:36 bestfit.c
-rw-rw-r-- 1 skec25 skec25 1669 Apr 10 09:19 correctpipe.c
-rw-rw-r-- 1 skec25 skec25  977 Apr 16 06:15 correctprio.c
-rw------- 1 skec25 skec25  13 Apr 10 08:14 datafile.dat
-rw------- 1 skec25 skec25  13 Apr 10 08:15 example.dat
-rw-rw-r-- 1 skec25 skec25  166 Apr 16 06:25 exec.c
-rw-rw-r-- 1 skec25 skec25  490 Apr 10 09:43 exit.c
Parent Process
```

Result:

Thus the program for exec system call has been executed successfully.

(f) Opendir,readdir:

AIM:

To write a C program to display the files in the given directory

ALGORITHM:

1. Start the program
2. Declare the variable to the structure dirent (defines the file system-independent directory) and also for DIR
3. Specify the directory path to be displayed using the opendir system call
4. Check for the existence of the directory and read the contents of the directory using readdir system call (returns a pointer to the next active directory entry)
5. Repeat the above step until all the files in the directory are listed
6. Stop the program

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc,char *argv[])
{
    char buff[256];
    DIR *dirp;
    printf("\n\nEnter directory name");
    scanf("%s",buff);
    if((dirp=opendir(buff))==NULL)
    {
        printf("Error");
        exit(1);
```

```
          }
          while(dptr=readdir(dirp))
          {
                printf("%s\n",dptr->d_name);
          }
          closedir(dirp);
    }
```

Output:

```
Enter directory name
oslab
openreaddir.c
a.out
..vidhya.c
vidhya.
```

(h)Open:

AIM:

To Execute a Unix Command in a ‘C’ program using open() system call.

ALGORITHM:

1. Start the program
2. Declare the necessary variables
3. Open file1.dat to read or write access
4. Create file1.dat if it doesn’t exist
5. Return error if file already exist
6. Permit read or write access to the file
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
int main()
{
    int fd; fd=creat("file1.dat",S_IREAD|
    S_IWRITE); if(fd==-1)
        printf("Error in opening file1.dat\n");
```

111

```
        els
        e
        {
            printf("\nfile1.dat opened for read/write access\n"); printf("\
            nfile1.dat is currently empty");

        }
        close(fd);

    }
```

OUTPUT:

```
 file1.dat opened for read/write access.
```

RESULT:

Thus the program for open system call has been executed successfully.

Ex.No:4 I/O SYSTEM CALLS OF UNIX OPERATING SYSTEM (OPEN, READ, WRITE, ETC)

OPEN,READ,WRITE:

AIM:

To implement UNIX I/O system calls open, read , write etc.

ALGORITHM:

1. Create a new file using creat command (Not using FILE pointer).
2. Open the source file and copy its content to new file using read and write command.
3. Find size of the new file before and after closing the file using stat command.

PROGRAM:

```
#include<fcntl.h>

//file control #include<sys/types.h>
#include<sys/stat.h>
static char message[]="hai Hello world";

int main()
{
    int fd;
    char buffer[80]; fd=open("new2file.txt",O_RDWR|O_CREAT|
    O_EXCL,S_IREAD|S_IWRITE); if(fd!=-1)
    {
          printf("new2file.txt opened for read/write access\n");
          write(fd,message,sizeof(message)); lseek(fd,0l,0);
          if(read(fd,buffer,sizeof(message))==sizeof(message))

              printf("\"%s\" was written to new2file.txt\n",buffer);

          else

              printf("***Error readind new2file.txt***\n");

          close(fd);

      }

      else

          printf("***new2file.txt  already  exists***\n");
     exit(0);

    }
```

OUTPUT:

```
 new2file.txt opened for read/write access
"hai Hello world" was written to new2file.txt
```

Ex.No:5 C PROGRAMS TO SIMULATE UNIX COMMANDS LIKE LS, GREP.

(a) LS:

AIM:

To implement ls command in c.

ALGORITHM:

1. Include a dirent.h header file.
2. Create a variable DIR as pointer
3. Create a structure pointer of dirent
4. Using opendir function,open the current directory
5. Read the directory for files using readdir function
6. Display it till the end of the file.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
#include<errno.h>
#include<sys/stat.h>
int main(int argc,char ** argv)
{
     DIR *dir;
     struct dirent *dirent; char * where=NULL;
     if(argc==1)where=get_current_dir_name();
     else
```

111

```
        where=argv[1];
if(NULL==(dir=opendir(where))){

fprintf(stderr,"%d(%s)opendir %s failed\n",errno,strerror(errno),where);
return 2;

}

while(NULL!=(dirent=readdir(dir)))

{

        printf("%s\n",dirent->d_name);

}

closedir(dir);
return 0;

}
```

OUTPUT:

```
openreaddir.c
file.txt
openclose.c
staffrr.c fifo.c
example.dat
newgetpid.c
```

RESULT :

Thus the system call program has been executed successfully.

(b) GREP:

AIM:

To implement the grep command in c

ALGORITHM:

1. Obtain the required pattern to be searched and file name from the user
2. Open the file and read the constants used by word till the end of the file.
3. Match the given pattern with the read word and if it matches,display the line of occurance
4. Do this till the end of the file is reached.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int main(int argv,char * args[])
{
```

```
FILE * f;

char str[100];
char c;

int i,flag,j,m,k;

char arg[]="HI";
char temp[30];
if(argv<3)

{

        printf("usage grep<s> <val.txt>\
        n"); return;

}

f=fopen(args[2],"r");
while(!feof(f))

{
   i=0;
   while(1)
   {
         fscanf(f,"%c",&c);
if(feof(f))
         {
               str[i++]='\0';
               break;
         }
         if(c=='\n')
         {
               str[i++]='\0';
               break;
         }
         str[i++]=c;
   }
   if(strlen(str)>=strlen(args[1]))
   for(k=0;k<=strlen(str)-strlen(args[1]);k++)
   {
    for(m=0;m<strlen(args[1]);m++)
         temp[m]=str[k+m];
         temp[m]='\0';
         if(strcmp(temp,args[1])==0)
         {
               printf("%s\n",str);
        break;
```

```
            }
        }
        }
        return 0;
    }
```

OUTPUT:

```
[skec25@localhost ~]$ ./a.out print stat.c
printf("enter name of file whose stsistics has to");
printf("user id %d\n",nfile->st_uid);
printf("block size :%d\n",nfile->st_blksize); printf("last
access time %d\n",nfile->st_atime); printf("time of last
modification %d\n",nfile->st_atime); printf("porduction
mode %d \n",nfile->st_mode); printf("size of file
%d\n",nfile->st_size);
printf("nu,mber of links:%d\n",nfile->st_nlink);
```

RESULT:

Thus the program has been executed successfully.