PRELAB-1

Write the description for the following system calls: open, close, read, write, fork, wait, exit, sleep, File descriptor values for standard I/O channels.

Pre lab-1

190031187
N.V.Radhakrishna

System call :-

(i) open (filename, flags)

description - open a file ; the flags indicate Read / Write

(ii) close(fd)

description - close the file which pointed by the file descriptor and frees the file descriptor

(iii) read :-

used to read the data from the file associated with file descriptor

Syntax :-

size_t read (int fd, void* buf, size_t cnt)

parameters :-

fd : file descriptor
buf - buffer to write the data into
cnt :- length of buffer

(iv) write (fd, buf, n)

description :- write n bytes to an open file

(v) fork()

description :- create a process

(vi) wait()

description :- wait for a child process to exit

(vii) exit()

description :- Terminate current process

(viii) sleep(n)

description :- sleep for n clock ticks

(ix) fsat(fd)

description :- return info about an open file

(x) link(f_1, f_2)

description :- create another name f_2 for the file f_1

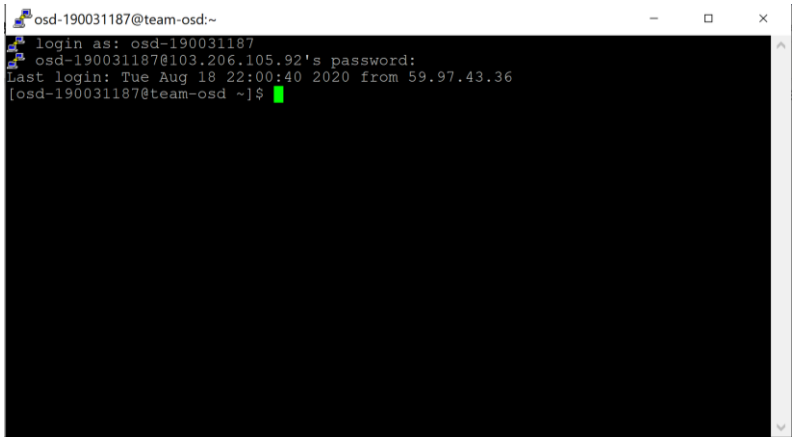
INLAB-1

Write a system program for implementing cat.c: which forms the essence of cat copies data from its standard input to its standard output. If an error occurs, it writes a message to the standard error. fork.c: A Simple Child Creation Program.

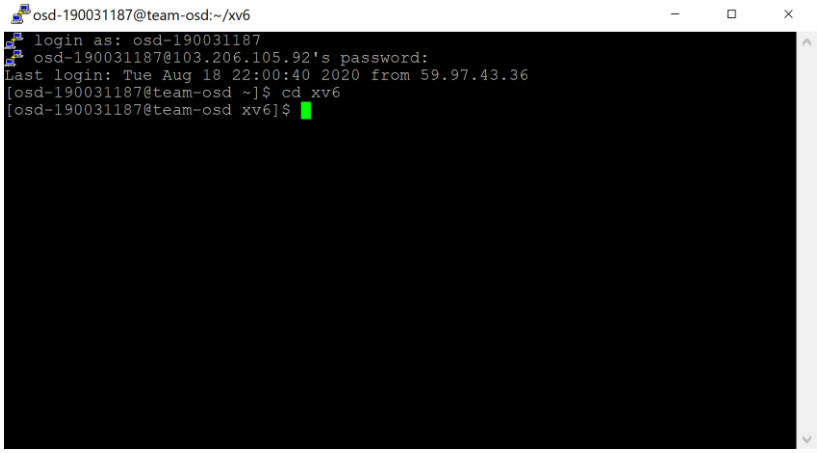
Solution)

Stepwise Procedure:

1) Login to putty and write cd xv6



2) next type cd xv6



3) Type nano fork1.c and enter the code and save

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1                File: fork1.c

#include "types.h"
#include "stat.h"
#include "user.h"

int main(void)
{
  int pid=fork();
  if(pid>0)
  {
    printf(1,"parent: child=%d\n",pid);
    pid=wait();
    printf(1,"child %d is done\n",pid);
  }
  else if(pid==0)
  {
    printf(1,"child: exiting\n");
    exit();
  }
  else
  {
    printf(1,"fork error\n");
  }
  exit();
}

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

4) next type nano Makefile Under UPROGS add the file

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1                File: Makefile

# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _square\
        _fork1\
        _prog2\

fs.img: mkfs README $(UPROGS)
        ./mkfs fs.img README $(UPROGS)

-include *.d

clean:
        rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
        *.o *.d *.asm *.sym vectors.S bootblock entryother \
        initcode initcode.out kernel xv6.img fs.img kernelmemfs \
        xv6memfs.img mkfs .gdbinit \

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

4.2) Also Under EXTRA add the file and SAVE

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1                File: Makefile

# check in that version.

EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
        ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
        printf.c umalloc.c fork1.c square.c prog2.c\
        README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
        .gdbinit.tmpl gdbutil\

dist:
        rm -rf dist
        mkdir dist
        for i in $(FILES); \
        do \
                grep -v PAGEBREAK $$i >dist/$$i; \
        done
        sed '/CUT HERE/,$$d' Makefile >dist/Makefile

^G Get Help ^O WriteOut ^R Read File^Y Prev Page^K Cut Text ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

5) write make clean

```
osd-190031187@team-osd:~/xv6
login as: osd-190031187
osd-190031187@103.206.105.92's password:
Last login: Tue Aug 18 22:00:40 2020 from 59.97.43.36
[osd-190031187@team-osd ~]$ cd xv6
[osd-190031187@team-osd xv6]$ nano fork1.c
[osd-190031187@team-osd xv6]$ nano Makefile
[osd-190031187@team-osd xv6]$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs
_usertests _wc _zombie _square _fork1 _prog2
[osd-190031187@team-osd xv6]$
```

7) next write make command and next write make qemu-nox

```
osd-190031187@team-osd:~/xv6
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C0



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 8
init: starting sh
$
```

9) type your file name the ouput will execute

```
osd-190031187@team-osd:~/xv6
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C0



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 8
init: starting sh
$ fork1
parent: child=4
child: exiting
child 4 is done
$
```

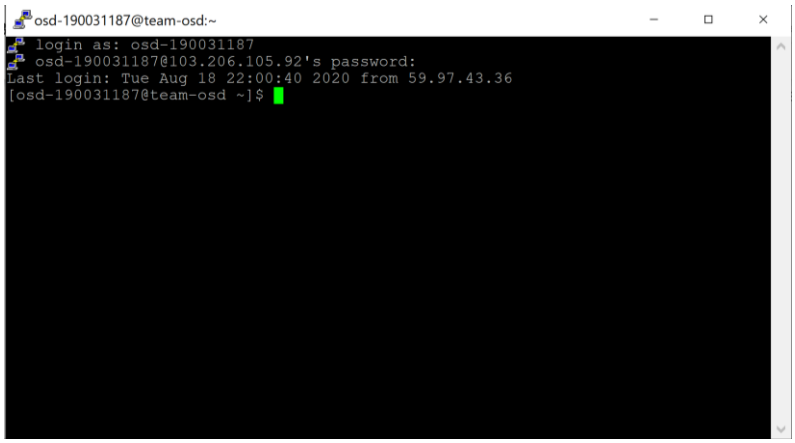
POSTLAB-1

Write a program using C library functions for file handling Standard I/O library that will copy data from one file to another file. io.c : A Program That Does I/O operations.

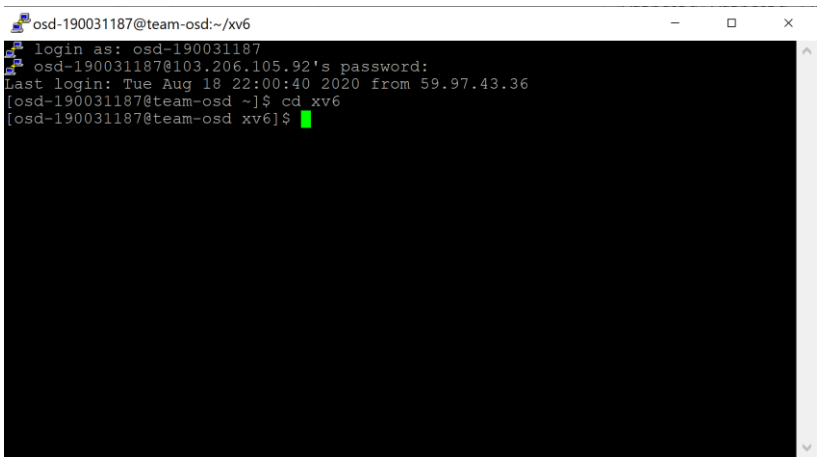
Solution)

Stepwise Procedure:

1) Login to putty and write cd xv6



2) next type cd xv6



3) Write code for i/o

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1                    File: prog2.c

#include "types.h"
#include "stat.h"
#include "user.h"

int main(void)
{
  char buf[512];
  int n;
  for(;;)
  {
   n = read(0,buf,sizeof(buf));
   if(n==0)
    break;
   if(n<0)
   {
    printf(2,"read error\n");
    exit();
   }
   if(write(1,buf,n)!=n)
   {
     printf(2,"write error\n");
     exit();
   }
  }
  exit();
}

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

4) next type nano Makefile Under UPROGS add the file

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1                    File: Makefile

# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _square\
        _fork1\
        _prog2\

fs.img: mkfs README $(UPROGS)
        ./mkfs fs.img README $(UPROGS)

-include *.d

clean:
        rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
        *.o *.d *.asm *.sym vectors.S bootblock entryother \
        initcode initcode.out kernel xv6.img fs.img kernelmemfs \
        xv6memfs.img mkfs .gdbinit \

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

4.2) Also Under EXTRA add the file and SAVE

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1            File: Makefile

# check in that version.

EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
        ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
        printf.c umalloc.c fork1.c square.c prog2.c\
        README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
        .gdbinit.tmpl gdbutil\

dist:
        rm -rf dist
        mkdir dist
        for i in $(FILES); \
        do \
                grep -v PAGEBREAK $$i >dist/$$i; \
        done
        sed '/CUT HERE/,$$d' Makefile >dist/Makefile

^G Get Help  ^O WriteOut  ^R Read File^Y Prev Page^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

5) write make clean

```
osd-190031187@team-osd:~/xv6
login as: osd-190031187
osd-190031187@103.206.105.92's password:
Last login: Tue Aug 18 22:00:40 2020 from 59.97.43.36
[osd-190031187@team-osd ~]$ cd xv6
[osd-190031187@team-osd xv6]$ nano fork1.c
[osd-190031187@team-osd xv6]$ nano Makefile
[osd-190031187@team-osd xv6]$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs
_usertests _wc _zombie _square _fork1 _prog2
[osd-190031187@team-osd xv6]$
```

7) next write make command and next write make qemu-nox

```
osd-190031187@team-osd:~/xv6
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C0



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 8
init: starting sh
$
```

9) type your file name

input:radha

output:radha

osd-190031187@team-osd:~/xv6

```
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32
init: starting sh
$ prog2
radha
radha
```

Operating System Design – 19CS2106S

Lab Experiment - 2

Pre-Lab 2:

- Write the description for the following
  getpid: Return the current process's pid
  getppid: Obtains a parent process's ID number getuid and getgid get the user or group ID of the process, respectively.
  getegid- get the effective group ID
  geteuid - get the effective user ID errno, a global variable that holds the numeric code of the last system call error perror (), a subroutine that describes system call errors.
  Moving in a File: lseek () dup:Duplicate fd

OSD
Practical-2

190031187
Radhakrishna

prelab

1. getpid –
returns the process id of the parent of the calling process was created by fork() function and the parent process still exists at the time of getpid function call, this returns the process id of the parent process.

getppid :-
returns the process id of the calling process This is often used by routines that generate unique temporary filenames.

getegid() :-
Finds the effective group id of the calling process. returns the effective group id It is always successful.

format
```
#define _POSIX_SOURCE
#include <unistd.h>
gid_t getegid (void);
```

geteuid()

finds the effective user ID of the calling process. Returns the effective userID of the calling process. It is always successful

format

```
# define _POSIX_SOURCE
# include <unistd.h>
euid_t geteuid(void);
```

Iseek()

For this we will be provided two txt files our task is to write the contents from one file to another file using Iseek() which is used to change the pointer of the file descriptor.

190031187
N.V.Radhakrishna

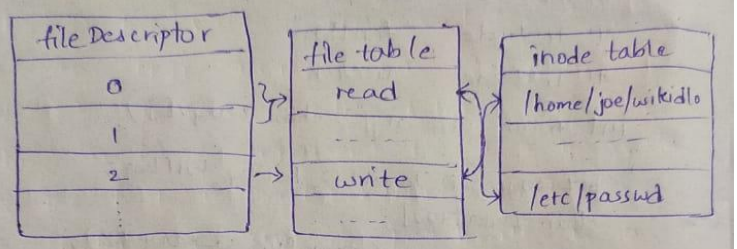
In-Lab 2:

- ccp.c: Copying a File. Show a pictorial arrangement of File Descriptor, File and Inode tables for a single process that has two different files open.

Inlab 190031187 Radhakrishna

```
1.  # include "types.h"
    # include "stat.h"
    # include "fcntl.h"
    # include "user.h"
    int main (int argc, char *argv[])
    {
      int SourceFD, TargetFD, RdFlag, WrFlag;
      char Data[100];
      SourceFD = open(argv[1], O_RDONLY);
      if(SourceFD<0)
      {  printf(1,"Error reading the sourcefile");
         exit();
      }
      RdFlag = read(SourceFD, Data, sizeof(Data));
      if( RdFlag<0)
      {  printf(1,"Error reading source file");
         exit();
      }
      TargetFD = open(argv[2], O_CREATE|O_WRONLY);
      if(TargetFD<0)
      {  printf(1,"Error opening target file");
         exit();
      }
      WrFlag = write(TargetFD, Data, sizeof(Data));
      if( WrFlag <0){
         printf(1,"Error writing target file");
         exit()
      }
      close(SourceFD);
      close(TargetFD);
    }
```

pictorial Arrangement of File descriptor, file table, inode table



File descriptor for a single process file table & inode table

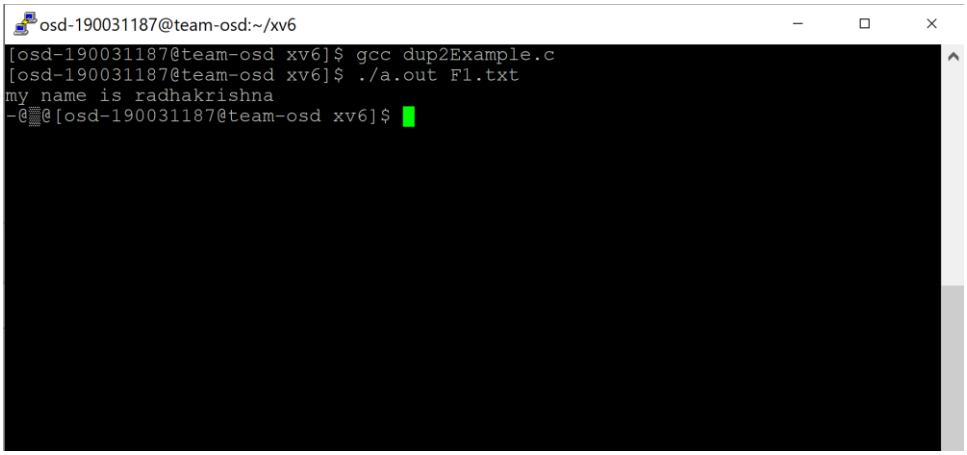
OUTPUT CP.C

```
osd-190031187@team-osd:~/xv6
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
190031187$ cat>F1.txt
my name is radhakrishna
my id no is 190031187
190031187$ cp F1.txt F2.txt
pid 4 cp: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
190031187$ cat F2.txt
my name is radhakrishna
my id no is 190031187
190031187$
```

- Write a system program which will opens files in the parent and uses dup2 in the child to reassign the descriptors .

```
2. #include <sys/types.h>
   #include <sys/stat.h>
   #include <fcntl.h>
   #include <stdio.h>
   int main() {
      int SrcFD, TrgFD, Flag, RdFlag, WrFlag;
      char Data[100];
      SrcFD = open("F1.txt", O_RDONLY);
      Flag = dup2(SrcFD, TrgFD);
      RdFlag = read(TrgFD, Data, sizeof(Data));
      WrFlag = write(1, Data, sizeof(Data));
      close(SrcFD);
      return 0;
   }
```

190031187
Radhakrishna



Post-Lab 2:

- show_error s.c: Displaying system call errors with perror Filename provided as argument

2. Display system call error:

It is represented as (errno). The program displays the value errno and the filename if an error occurs when using the open() system call the return probe sets the arg() & argument to -1 & the value of the buit in errno variable indicates the nature of error. A perdicate is used to test the value of arg. Alternatively you could test whether the value of errno is greater than zero.

when you have saved the script to a file and make the file executable you can then run it to display information about any failure of the open() system call that occurs on system. After you have started the script in a seperate terminal you can run commands that result in an error, such as running the ls command to the list the file that does not exist

190031187
Radhakrishna

- process.c: Looking Up Some Process Credentials Prints PID, PPID, real and effective UIDs and GIDs Also fetches and sets PATH.

Post Lab 190031187 Radhakrishna

1. process.c

Looking up some process credentials prints PID, PPID, read and effective UID's & GID's Also fetches and sets path.

Process ID (PID)

Each process has a unique non-negative integer identifier that is assigned when the process is created using fork() . A process can obtain its PID using getpid()

Parent Process ID (PPID)

A parent process ID identifies the created process using fork(). A process can obtain its PPID using getppid().

User and group identifiers

Each process have various associated user & group ID's. These ID's are respectively represented using the types uid_t , gid_t Real user ID , Real group ID These id's determines who owns the process. A process can obtain its real user group ID using get uid() , getgid()

Lab Experiment – 3

Pre-Lab 3:

The exec family(**execl, execv, execlp, execvp**) of library functions replace the calling process's code, data, and stack from the executable whose pathname is stored in path

practical-3

pre-Lab

1. Exec System call:

The exec system call replaces the current running process image with new one

That is, it replaces the current address space (the text, data & stack) with that of the new process image.

Since no new process is created the PID does not change across an exec.

Normally each exec is followed by one or more letters

e → An array of pointers to environment variables is explicitly passed to new process image

l → command line arguments are passed individually (as a list) to the function

p → uses the path environment variable to find the file named in the file argument to be executed.

v → command line arguments are passed to the function as an array (vector) of pointers.

190031187 Radhakrishna

These are exec family system calls

execl execv execlp execvp

Syntax

int execl (const char *path, const char *arg, ... /* (char*) NULL */);

int execv (const char *path, char *const argv[]);

int execlp (const char *[redacted], const char *arg, ... /* (char*) NULL */);

int execvp (const char *[redacted], char *const argv[]);

file

file

In-Lab 3:

- execl *and* execv, Gathering the exit Status using wait with standard input and output redirection.

190031187

In lab

execl :-

The execl function requires each component of the command line of the new program to be specified as individual arguments

Syntax

int execl (const char *path, const char *argv[], .../*(char*)NULL*/);

→ execl doesn't use the path.

→ The first argument (path) signifies the absolute or relative pathname of the program.

→ The other arguments represent each word of the command line beginning with the name of the command

→ The ellipsis representation in the syntax (..../*) points to varying no. of arguments

190031187 Radhakrishna

Example program on execl()

→ consider we have EXEC.c and execDemo.c

→ Now, we will Replace the execDemo.c with EXEC.c by calling execl() function in execDemo.c

→ EXEC.c code:-

```
#include <stdio.h>
#include<unistd.h>
int main()
{   int i;
    printf("[redacted] I am EXEC.C called by execl()");
    printf("\n");
    return 0;
}
```

Now create an executable file of EXEC.C using command gcc EXEC.C -OEXEC

execDemo.c

execDemo.c Code

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{   char *args="./EXEC";
```

```
    execl (args, args, NULL)
    printf("ending...");
    return 0;
}
```

→ create an executable file of execDemo.c using command gcc execDemo -o execDemo

Final Output

Now execute code by typing

./execDemo

output:- I am EXEC.c called by execl()

execv

syntax :-

int execv(const char *path, char *const argv[]);

Here path should point to the path of file being executed and argv[] is a null terminated [redacted] array of character pointers

Example program on execv();

Conside we have EXEC2.c, execDemo2.c

we will replace execDemo2.c with EXEC2.c by calling execv() in execDemo2.c

190031187 Radhakrishna

190031187 Radhakrishna

EXEC2.C code

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("I am in EXEC2.c called by execv")
    printf("\n");
    return 0;
}
```

Create an executable file of EXEC2.C using command gcc EXEC2.C -o EXEC2

execDemo2.c code

```
#include <stdio.h>
#include<stdlib.h>
#include <unistd.h>
int main()
{   char *args[]={"./EXEC2",NULL};
    execv(args[0], args)
    printf("ending...");
    return 0;
}
```

Create an executable file of execDemo2.C using command

execDemo2.C -o execDemo2

Final output:

190031187 Radha krishna

Now execute the code by typing

./exec2Demo

output:

I am in ([redacted]) EXEC2.C called by execv()

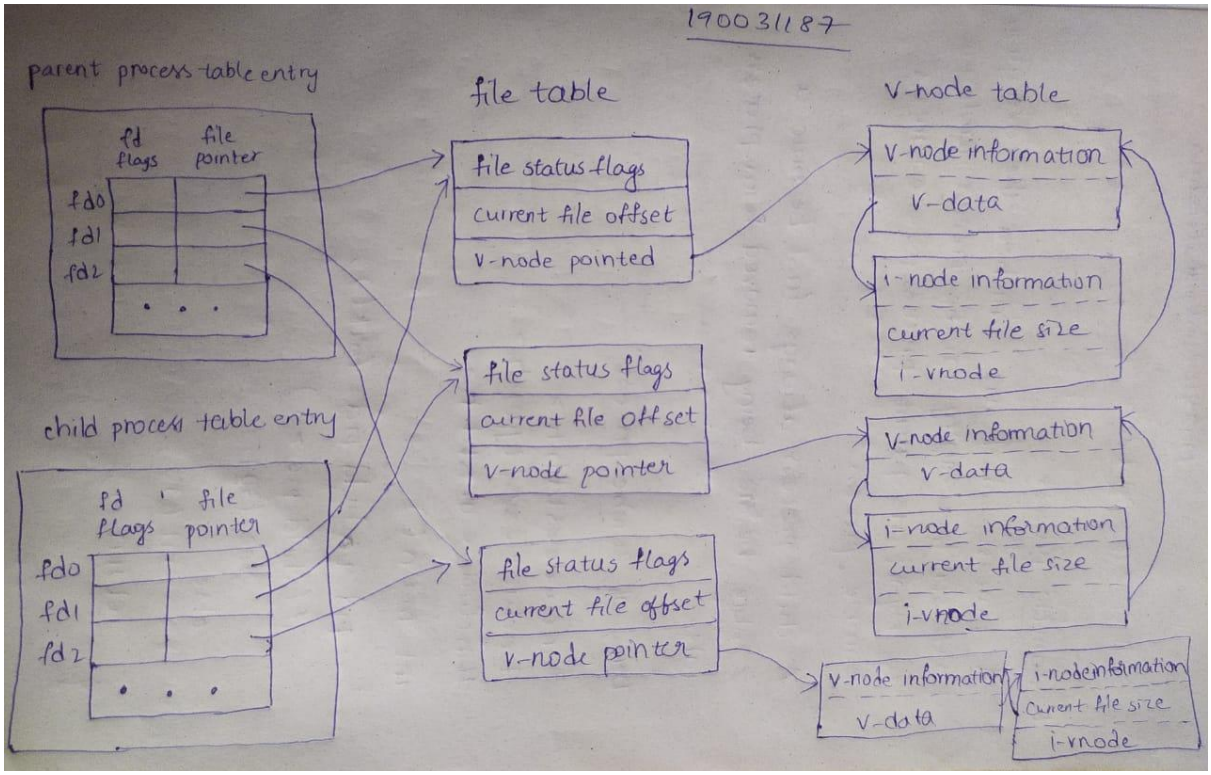
execl() OUTPUT

```
osd-190031187@team-osd:~
login as: osd-190031187
osd-190031187@103.206.105.92's password:
Last login: Sat Aug 29 15:33:03 2020 from 117.192.183.57
[osd-190031187@team-osd ~]$ ./execDemo
I am in EXEC.C called by execl
[osd-190031187@team-osd ~]$
```

execv() OUTPUT

```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ ./exec2Demo
I am in EXEC2.C called by execv()
[osd-190031187@team-osd ~]$
```

- Show a pictorial arrangement - Sharing of open files between parent and child after fork



Post-Lab 3:

- orphan.c, zombie.c: create orphan and processes

190031187 Radhakrishna

post lab

1. zombie process :-

A process which has finished the execution but still has entry in the process table to report its parent process is known as zombie process.

A child process always first become a zombie before being removed from process table

Demo of zombie process

```
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
    pid_t child_pid = fork();
    if (child_pid > 0)
        sleep(50);
    else
        exit(0);
    return 0;
}
```

190031187

Here the child finishes its execution using [redacted] exit() system call while the parent sleeps for 50 seconds, hence does n't call wait() & the child process entry still exists in the process table

orphan process

A process whose parent no more exists i.e either finished or terminated without waiting for its child process to terminate is called orphan process.

Demo of orphan process

```
# include< stdio.h>
# include < sys/types.h>
# include < unistd.h >
int main()
{ int pid= fork();
  if( pid >0)
    printf("In parent process");
  elseif ( pid ==0){
    sleep(30);
    printf("In child process");
  }
  return 0;
}
```

190031187

Here the parent process finishes execution & exits while the child process is still executing. This is known as orphan process.

However the orphan process is soon adopted by init process, once its parent dies

DEMO OF ORPHAN PROCESS OUTPUT

```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ nano DemoOrphan.c
[osd-190031187@team-osd ~]$ gcc DemoOrphan.c -o DemoOrphan
[osd-190031187@team-osd ~]$ nano DemoOrphan.c
[osd-190031187@team-osd ~]$ gcc DemoOrphan.c -o DemoOrphan
[osd-190031187@team-osd ~]$ ./DemoOrphan
In parent process
[osd-190031187@team-osd ~]$
```

- Program that creates a new Process to Copy File

postlab 3 190031187

```
#include "types.h"
#include "fcntl.h"
#include "stat.h"
#include "user.h"

int main(int argc, char *argv[])
{
    int pid=fork();
    if(pid>0)
    {
        pid=wait();
    }
    else if(pid==0)
    {
        int SourceFD, TargetFD, RdFlag, WrFlag;
        char Data[100];
        SourceFD=open(argv[1], O_RDONLY);
        if(SourceFD<0)
        {
            printf(1, "Error opening source file");
            exit();
        }
        RdFlag = read(SourceFD, Data, sizeof(Data));
        If(RdFlag<0)
        {
            printf(1, "Error reading source file");
            exit();
        } TargetFD=open(argv[2], O_CREATE|O_WRONLY)
        if(TargetFD<0)
        {
            printf(1, "Error opening target file");
            exit(); }
```

190031187

```
    wrFlag= write (TargetFD, Data, sizeof(Data));
    if(wrFlag <0)
    {  printf( 1," error writing target file");
       exit();
    }
    close (TargetFD);
    close (SourceFD);
  }
  return 0;
}
```

osd-190031187@team-osd:~/xv6

```
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
190031187$ cat>F1.txt
my name is radhakrishna
190031187$ postlab3 F1.txt F2.txt
pid 5 postlab3: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
pid 4 postlab3: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
190031187$ cat F2.txt
my name is radhakrishna
190031187$
```

Operating System and Design (19CS2106S)
Lab- 4
Pre-Lab:

Unnamed Pipes: pipe ()

OSD Lab-4 190031187
Radhaknishna

pre-Lab

1. PIPE: The UNIX system has a powerful Construct called pipe, which allows stdout of a command. The UNIX command line interpreter provides a pipe facility. we can use the pipe character to do so. Here There are two types of pipes: Named pipes
Ordinary pipes
(Also known as unnamed pipes)

unnamed pipes:- The ordinary pipes in OS allows the process to communicate in a standard way. The process writes on the one end (~~as a result~~) and reads on the other end. As a result, ordinary pipes are unidirectional, allowing only one-way communications.

Pipe() :- pipe() is a system call that facilitates inter process communication. It opens a pipe, which is an area of main memory treated as virtual file

190031187 Radhakrishna

* The pipe can be used by the creating process, as well as all its child processes for reading and writing
* we can use a pipe such that one process write to the pipe and the other process reads from the pipe. data flows from left to right through the pipeline
* If a process tries to read before something is written to the pipe, the process is suspended until something is written.

<u>System call</u> pipe()

```
int pipe(int fields[2]);
```

fields[0] will be fields for read end of file
fields[1] will be fields for write end of file

Return 0: on success
-1 on error

osd-190031187@team-osd:~

```
  GNU nano 2.3.1        File: unnamedpipe.c

#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#define WRITE 1
#define READ 0

char *phrase="Stuff this in your pipe and sent it";

int main()
{
 int fd[2],bytesRead;
 char message[100];
 pipe(fd);
 if(fork()==0){
  close(fd[READ]);
  write(fd[WRITE],phrase,strlen(phrase)+1);
  close(fd[WRITE]);
 }
 else{
  close(fd[WRITE]);
  bytesRead=read(fd[READ],message,100);
  printf("Read %d bytes:%s\n",bytesRead,message);
  close(fd[READ]);
 }
 return 0;
}

                        [ Read 28 lines ]
^G Get Help ^O WriteOut ^R Read Fil ^Y Prev Pag ^K Cut Text ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Pag ^U UnCut Te ^T To Spell
```

OUTPUT

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano unnamedpipe.c
[osd-190031187@team-osd ~]$ gcc unnamedpipe.c
[osd-190031187@team-osd ~]$ ./a.out
Read 36 bytes:Stuff this in your pipe and sent it
[osd-190031187@team-osd ~]$
```

mkfifo: creates a named pipe called fileName.

190031187 Radhakrishna

mkfifo :- make a FIFO [First in First Out]

create a named pipe called FileName

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

mkfifo() makes a FIFO special file with name pathname mode specifies the FIFO permissions. It is modified by the process unmask in the usual way

The permissions of the file are (mode of unmask)

mknod: Creates a special file.

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ mknod mypipe p
[osd-190031187@team-osd ~]$ ls -l mypipe
prw-rw-r--. 1 osd-190031187 osd-190031187 0 Sep  8 13:06 mypipe
[osd-190031187@team-osd ~]$
```

link: Creates a hard link.

190031187 Radhakrishna

Link: creates a hard link

create a source file and add some content to it using nano or vi command

nano abc.txt

create a hard link b/w source file and any other file using

ln abc.txt abcd.txt

verify hardlink is created or not using ls command

ls -l abc.txt abcd.txt

Remove the source file using rm command

rm abc.txt

verify link b/w them using ls command which shows source file is deleted

ls-l abc.txt abcd.txt

Although the source file is deleted we will have the same content of source file in linked file

cat abcd.txt

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano abc.txt
[osd-190031187@team-osd ~]$ ln abc.txt abcd.txt
[osd-190031187@team-osd ~]$ ls -l abc.txt abcd.txt
-rw-rw-r--. 2 osd-190031187 osd-190031187 43 Sep  8 12:01 abcd.txt
-rw-rw-r--. 2 osd-190031187 osd-190031187 43 Sep  8 12:01 abc.txt
[osd-190031187@team-osd ~]$ rm abc.txt
[osd-190031187@team-osd ~]$ ls -l abc.txt abcd.txt
ls: cannot access abc.txt: No such file or directory
-rw-rw-r--. 1 osd-190031187 osd-190031187 43 Sep  8 12:01 abcd.txt
[osd-190031187@team-osd ~]$ cat abcd.txt
my id is 190031187
my name is radhakrishna
[osd-190031187@team-osd ~]$
```

Deadlock scenario for link.

190031187 Radhakrishna

Deadlock scenario for links

Two processes called deadlock by executing the following system calls simultaneously

process A : link ("a/b/c/d", "e/f/g");
process B : link ("e/f", "a/b/c/d/e");

suppose process A finds the inode for the file "a/b/c/d" at the same time process B finds the inode for "e/f", the phase at the same time means that the system arrives at a state where each process has allocated its inode. when process A now attempts to find the inode for directory "e/f", it would sleep awaiting the event that the inode for directory "e/f", it would sleep awaiting the event that the inode for "f" becomes free. But when process B attempt to find the inode for directory "a/b/c/d", it would sleep awaiting the event that the inode for "d" becomes free. process A would be holding a locked inode that process B wants, and process B would be holding a locked inode that process A wants.

process A	process B
⋮	Try to get inode e
Get inode for a	SLEEP _ inode e locked
Release inode a	⋮
Get inode for b	⋮
Release b	⋮
Get inode c	⋮
Release c	⋮
Get inode d	⋮
Try to get inode e	⋮
SLEEP - inode e locked	⋮
⋮ wakeup - inode e unlocked	⋮
	Get inode c
Get inode e	Release e
Release e	Get inode f
Try to get inode f	Get inode a
SLEEP - proc Blocked inode	Release a
	⋮
	Try to get inode d
Dead Lock	SLEEP - proc A locked inode

In-Lab:

1. **pipe.c**: Runs two programs in a pipeline Child runs cat, parent runs tr

osd-190031187@team-osd:~

```
  GNU nano 2.3.1          File: pipe1.c

#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<stdarg.h>

void quit(char *message,int exit_Status)
{
 printf("%s",message);
 exit(exit_Status);
}

int main()
{
 int fd[2];
 if(pipe(fd)<0)
  printf("os lead");
 switch(fork())
 {
  case -1: quit("fork",2);
  case 0: close(fd[0]);
          dup2(fd[1],STDOUT_FILENO);
          close(fd[1]);
          execlp("cat","cat","os.txt",(char *) 0);
          quit("cat",3);
  default:close(fd[1]);
          dup2(fd[0],STDIN_FILENO);
          close(fd[0]);
          execlp("tr","tr","[a-z]","[A-Z]",(char *) 0);
          quit("tr",4);
 }
}


^G Get Help ^O WriteOut ^R Read Fil ^Y Prev Pag ^K Cut Text ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Pag ^U UnCut Te ^T To Spell
```

OUTPUT

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano pipe1.c
[osd-190031187@team-osd ~]$ gcc pipe1.c
[osd-190031187@team-osd ~]$ ./a.out
HI
WELCOME TO OSD COURSE
MY NAME IS RADHAKRISHNA
MY ID IS 190031187
[osd-190031187@team-osd ~]$ cat os.txt
hi
welcome to osd course
my name is radhakrishna
my id is 190031187
[osd-190031187@team-osd ~]$
```

2. A half-baked directory using mknod.

```
osd-190031187@team-osd:~
  GNU nano 2.3.1              File: half-bake.c

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<dirent.h>

int main(int argc, char *argv[])
{
 if(argc!=2)
 {
  printf("try:comand directory name\n");
  exit(0);
 }
 if(mknod(argv[1],040777,0)==-1)
  printf("mknod fails\n");
}

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

OUTPUT

```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ nano half-bake.c
[osd-190031187@team-osd ~]$ gcc half-bake.c
[osd-190031187@team-osd ~]$ ./a.out half-bake.c
mknod fails
[osd-190031187@team-osd ~]$
```

Post-Lab:

1. mylink.c -- create the filename "another.txt" and link it to the other file. Later delete it using unlink.

```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ cat >> a.txt
welcome to osd course
my name is radhakrishna
my id is 190031187
[osd-190031187@team-osd ~]$ cat b.txt
cat: b.txt: No such file or directory
[osd-190031187@team-osd ~]$ ln -s a.txt b.txt
[osd-190031187@team-osd ~]$ cat b.txt
welcome to osd course
my name is radhakrishna
my id is 190031187
[osd-190031187@team-osd ~]$ ls -l a.txt b.txt
-rw-rw-r--. 1 osd-190031187 osd-190031187 65 Sep  8 13:30 a.txt
lrwxrwxrwx. 1 osd-190031187 osd-190031187  5 Sep  8 13:30 b.txt ->
 a.txt
[osd-190031187@team-osd ~]$ unlink b.txt
[osd-190031187@team-osd ~]$ ls -l a.txt b.txt
ls: cannot access b.txt: No such file or directory
-rw-rw-r--. 1 osd-190031187 osd-190031187 65 Sep  8 13:30 a.txt
[osd-190031187@team-osd ~]$ cat b.txt
cat: b.txt: No such file or directory
[osd-190031187@team-osd ~]$
```

2. Unlinking an opened file

```
osd-190031187@team-osd:~
  GNU nano 2.3.1            File: unlink.c

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main(argc, argv)
int argc;
char *argv[];
{
int fd;
char buf[10241];
struct stat statbuf;
if(argc !=2) /* need a parameter */
exit(1);
fd = open(argv[1], O_RDONLY);
if(fd ==-1) /* open fails */
exit(1);
if(unlink(argv[1])==-1) /* unlink file just opened */
exit(1);
if(stat(argv[1], &statbuf)==-1) /* stat the file by name*/
printf("stat %s fails as it should\n", argv[1]);
else
printf("stat %s succeeded!!!!\n", argv[1]);
if(fstat(fd, &statbuf)==-1)
{ /* stat the file by fd */
printf("fstat %s fails!!!\n", argv[1]);
}
else
{
printf("fstat %s succeeds as it should\n", argv[1]);
}
while(read(fd,buf,sizeof(buf)>0))    /* read open/unlinked file */
{
printf("%1024s", buf); /* prints 1K byte field */
}
}
```

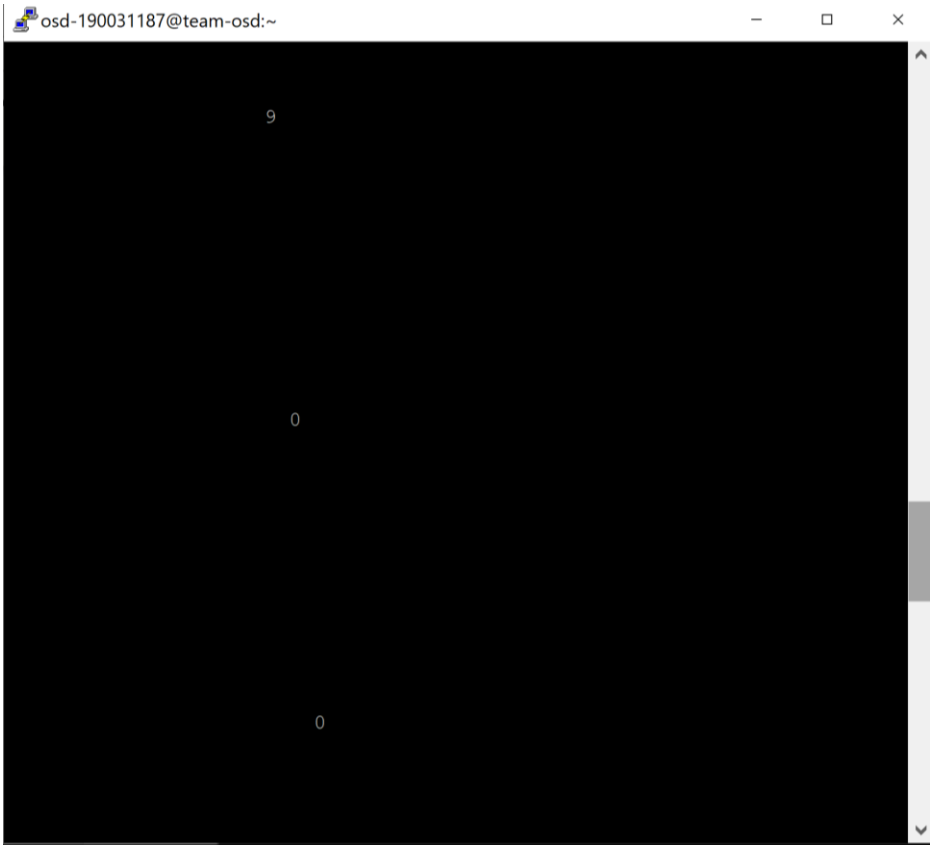
OUTPUT

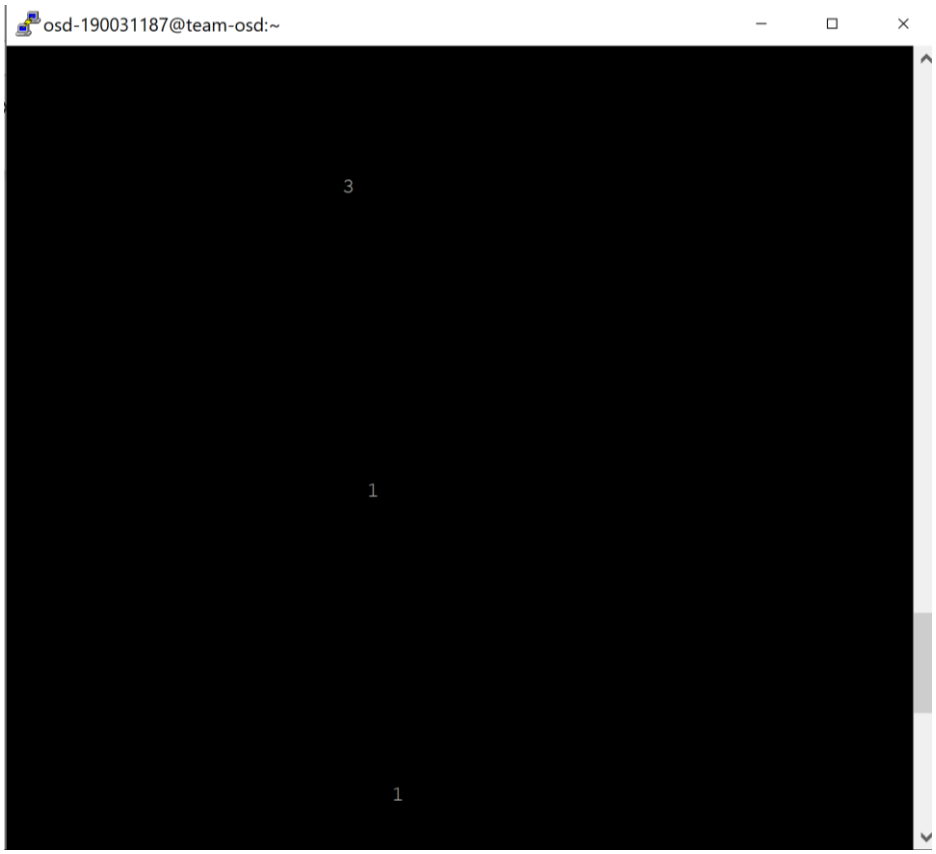
```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ nano unlink.c
[osd-190031187@team-osd ~]$ nano os.txt
[osd-190031187@team-osd ~]$ cleqar
bash: cleqar: command not found...
[osd-190031187@team-osd ~]$ clear
[osd-190031187@team-osd ~]$ nano unlink.c
[osd-190031187@team-osd ~]$ gcc unlink.c
[osd-190031187@team-osd ~]$ ./a.out os.txt
stat os.txt fails as it should
fstat os.txt succeeds as it should

 m
```

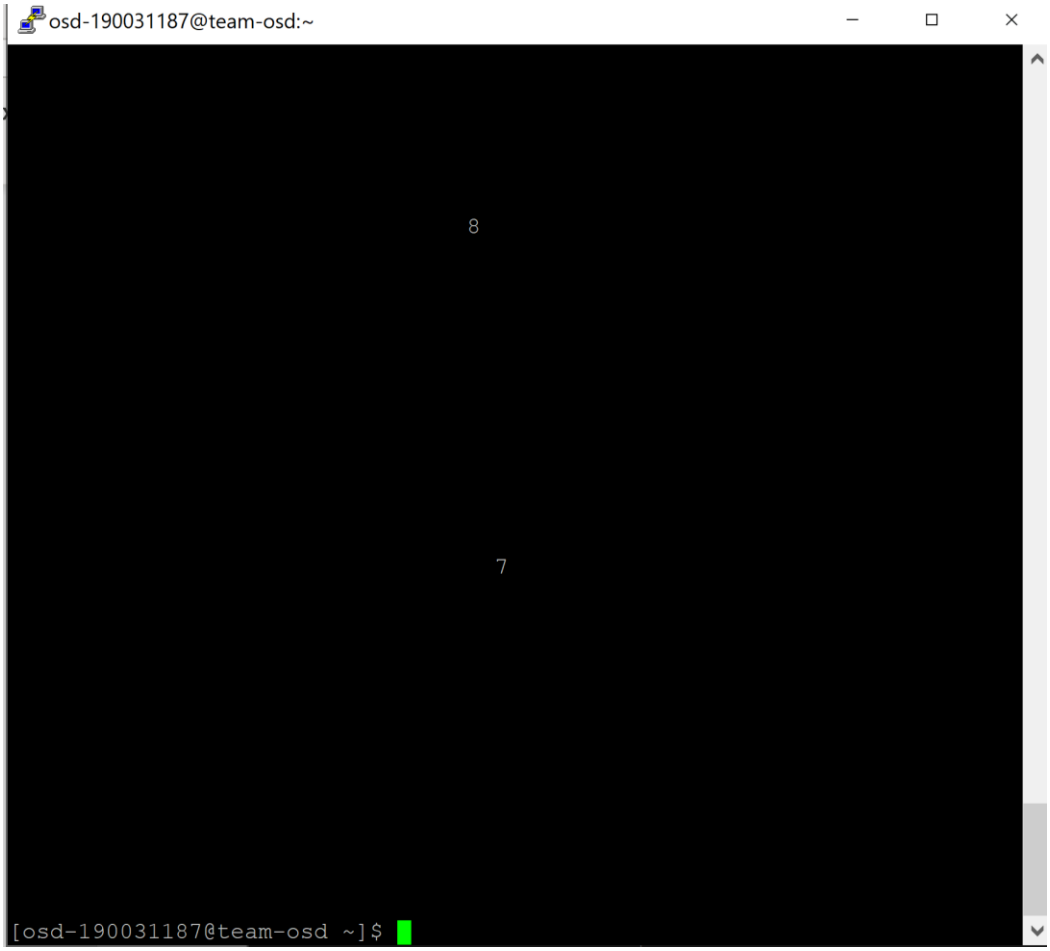
```
osd-190031187@team-osd:~

   y

       i
```

osd-190031187@team-osd:~
d
i

osd-190031187@team-osd:~
s
l


osd-190031187@team-osd:~
9
0
0


osd-190031187@team-osd:~
3
1
1


osd-190031187@team-osd:~
8
7
[osd-190031187@team-osd ~]$

Operating System and Design (19CS2106S)
Lab- 5

Pre-Lab

Reading Directory Information: **opendir (), readdir (), and closedir ()**

OSD Practical-5 190031187
Radhakrishna

prelab

1. opendir()

It shall open a directory stream corresponding to the directory named by the dirname argument. The directory stream is positioned at the first entry. If the type DIR is implemented using filedescriptor, applications shall only be able to open up to a total of files and directories.

readdir()

It returns a pointer to a dirent structure representing next directory entry in the directory stream pointed to by dirp. It returns NULL on reaching the end of the directory stream or if an error occurred

closedir()

It shall close the directory stream referred to by the argument dirp. Upon return, the value of dirp may no longer point to an accessible object of the type DIR. If a fileDescriptor is used to implement type DIR, that fileDescriptor shall be closed.

chown: Changes a file's owner and/or group.

190031187 Radhakrishna

3. chown: changes a file's owner / or group

To change both the owner and the group of a file use the chown command followed by the new owner and group seperated by a colon with no intervening spaces and the target file.

chmod: Changes a file's permission settings.

2. ch[redacted]mod : changes a file's permission settings.

To modify the permission flags on existing files and directories, use the chmod command. It can be used for individual files or it can be run recursively with the -R operation to change permissions for all of the subdirectories and files within directory.

lstat: returns file attributes about an inode

4. lstat : returns file attributes about an inode

These functions return information about a file. No permissions are required on the file itself, but in the case of stat() and lstat() - execute (search) permission is required on all of the directories in path and lead to the file.

stat() stats the file pointed to by path and fills in buf.

lstat() is identical to stat(), except that if path is a symbolic link, then the link itself is stat-ed, not the file that it refers to.

fstat() is identical to stat(), except that the file to be stat-ed is specified by the file descriptor fd.

IN-LAB

1. attributes.c -- Uses lstat call and struct stat to display file attributes.

CODE

osd-190031187@team-osd:~

```
  GNU nano 2.3.1                                    File: attribute.c

#include <stdio.h>
#include <sys/stat.h>   /* For struct stat */
#include <stdarg.h>
#include <stdlib.h>
#include <time.h>
void quit(char *message, int exit_status)
{
        printf(" %s",message);
        exit(exit_status);
}

void arg_check (long int args, long int argc, char *message, long int exit_status)
{
        if (argc != args)
        {
                printf("%s", message);
                exit(exit_status);
        }
}

int main(long int argc, char **argv)
{
        struct stat statbuf;  /* We'll use lstat to populate this */
        arg_check(2, argc, "Single filename required\n", 1) ;
        if(lstat(argv[1], &statbuf) == -1)
                quit("Couldn't stat file", 1);
        printf("File: %s\n", argv[1]);
        printf("Inode number: %ld \n", statbuf.st_ino);
        printf("UID: %ld   ", statbuf.st_uid);
        printf("GID: %ld\n", statbuf.st_gid);
        printf("Type and Permissions: %o\n",statbuf.st_mode);
        printf("Number of links: %ld \n", statbuf.st_nlink);
        printf("Size in bytes: %ld\n", statbuf.st_size);
        printf("Blocks allocated: %ld\n", statbuf.st_blocks);
        printf("Last Modification Time: %s", ctime(&statbuf.st_mtime));
        printf("Last Access Time: %s\n", ctime(&statbuf.st_atime));
        exit(0);
}
```

OUTPUT

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano attribute.c
[osd-190031187@team-osd ~]$ gcc attribute.c
[osd-190031187@team-osd ~]$ cat>f1.txt
THis is practical 5
my id is 190031187
[osd-190031187@team-osd ~]$ ./a.out f1.txt
File: f1.txt
Inode number: 1109966712
UID: 1779   GID: 1780
Type and Permissions: 100664
Number of links: 1
Size in bytes: 63
Blocks allocated: 8
Last Modification Time: Wed Sep  9 10:42:55 2020
Last Access Time: Wed Sep  9 10:42:20 2020

[osd-190031187@team-osd ~]$ ln f1.txt f2.txt
[osd-190031187@team-osd ~]$ ./a.out f1.txt
File: f1.txt
Inode number: 1109966712
UID: 1779   GID: 1780
Type and Permissions: 100664
Number of links: 2
Size in bytes: 63
Blocks allocated: 8
Last Modification Time: Wed Sep  9 10:42:55 2020
Last Access Time: Wed Sep  9 10:42:20 2020

[osd-190031187@team-osd ~]$ ./a.out f2.txt
File: f2.txt
Inode number: 1109966712
UID: 1779   GID: 1780
Type and Permissions: 100664
Number of links: 2
Size in bytes: 63
Blocks allocated: 8
Last Modification Time: Wed Sep  9 10:42:55 2020
Last Access Time: Wed Sep  9 10:42:20 2020

[osd-190031187@team-osd ~]$
```

2. lsdir.c -- Lists only directories - Uses S_IFMT and S_ISDIR macros

CODE

osd-190031187@team-osd:~/xv6

```
  GNU nano 2.3.1                                    File: lsdir.c

#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h>
#include <unistd.h>
void quit (char *message, int exit_status)
{
        printf(" %s",message);
        exit(exit_status);
}

void arg_check(int args, int argc, char *message, int exit_status)
{
        if(argc != args)
        {
                printf("%s", message);
                exit(exit_status);
        }
}

int main(int argc, char *argv[])
{
        DIR *dir;
        struct dirent *direntry; /* Returned by readdir() */
        struct stat statbuf; /* Address of statbuf used by lstat() */
        mode_t file_type,file_perm;
        arg_check(2, argc, "Directory not specified\n", 1) ;
        if((dir = opendir(argv[1])) == NULL)
                quit("Couldn't open directory", 1);
        if((chdir(argv[1]) == -1)) /* Change to the directory before */
                quit("chdir", 2); /* you starting reading its entries */
        while((direntry = readdir(dir)) != NULL)
        {
                /* Read each entry in directory*/
                if(lstat(direntry->d_name, &statbuf) < 0)
                {
                        /* dname must be in */
                        perror("lstat");
                        /* current directory */
                        continue;

                }
                if(S_ISDIR(statbuf.st_mode))
                {
                        /* If file is a directory */
                        file_type = statbuf.st_mode & S_IFMT;
                        file_perm = statbuf.st_mode & ~S_IFMT;
                        printf("%o %4o %s\n",file_type,file_perm,direntry->d_name);
                }
        }
        exit(0);
}
```

OUTPUT

osd-190031187@team-osd:~/xv6

```
[osd-190031187@team-osd xv6]$ nano lsdir.c
[osd-190031187@team-osd xv6]$ gcc lsdir.c
[osd-190031187@team-osd xv6]$ mkdir OSD
[osd-190031187@team-osd xv6]$ ./a.out OSD
40000  775 .
40000  775 ..
[osd-190031187@team-osd xv6]$
```

Post-Lab

1. mychown.c -- change the group of the file.

190031187 Radhakrishna

post-lab

1. change the group of the file

Steps :-

As we are going to change the group of file we need to add a user, so we should come from home to root directory in the terminal by pressing sudo su command

(1) Create a file having some content in it

(2) Check by using ls command

(3) now add a user

(4) now add password for that user

Now by using chown command change the group

(5) Once again check by using ls command

Finally the group will be changed.

```
osd-190031187@team-osd:~
login as: osd-190031187
osd-190031187@103.206.105.92's password:
Last login: Wed Sep  9 09:47:03 2020 from 117.192.181.40
[osd-190031187@team-osd ~]$ cat>>cse.txt
hi
[osd-190031187@team-osd ~]$ ls -l cse.txt
-rw-rw-r--. 1 osd-190031187 osd-190031187 3 Sep  9 13:04 cse.txt
[osd-190031187@team-osd ~]$ useradd cse
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
[osd-190031187@team-osd ~]$
```

The reason why it is not working here. I don't have the permission to add a new user

2. mychmod.c -- changed the permission flags of the file.

2. change the permission flags of the file:-

steps:-

(1) create a file having some content in it

(2) now check all the permissions by using ls command

(3) change the permissions of the file by using chmod command

(4) once check whether the permissions are changed or not by using ls command.

```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ ls -l cse.txt
-rw-rw-r--. 1 osd-190031187 osd-190031187 3 Sep  9 13:04 cse.txt
[osd-190031187@team-osd ~]$ chmod 600 cse.txt
[osd-190031187@team-osd ~]$ ls -l cse.txt
-rw-------. 1 osd-190031187 osd-190031187 3 Sep  9 13:04 cse.txt
[osd-190031187@team-osd ~]$
```

Operating System Design – 19CS2106S

Skill Experiment – 1

1) What Is the Shell, Navigation, Exploring the System, Compiling Programs.

OS SKill - 1

190031187
N.V Radhakrishna

1. What is the shell, Navigation, Exploring the system, compiling programs

Shell :- A shell is a user interface for access to an operating system's services In general, operating system shells use either command Line Interface (CLI) or Graphical user Interface (GUI) depending on a computer's role and particular operation

Navigation:-

It refers to act of opening and moving throu -gh computer

for example

cd - used to change directory

ls - used to list the files in that directory

Exploring the System

Exploring the system can be referred to checking the types of files present in a directory etc..

ls - lists the files in that directory

file - Determine file type

190031187
Radhakrishna

compiling programe :-

A compiler takes the program code (source code) and converts into machine language code.

For c this can be acheived by

Type gcc filename.c

If there are no errors

Type ./a.out

Output will be displayed

2) cat.c syscall.c syscall.h sysproc.c user.h usys.S

190031187
Radhakrishna

cat.c - list the content of the file.

syscall.c - system call handling / dispatch code

syscall.h - system call handling / dispatch code

sysproc.c - process related system call implementation

user.h - declaration of system call wrappers & standard library functions

usys.s

assembly code (generated by preprocessor) for system call wrapper.

3) 1. XV6 Installation 2. add-a-newsystem-call-inxv6

190031187 Radhakrishna

3. 1. XV6 Installation

open putty and login to your system

then type

git clone git://github.com/mit-pdos/xv6-public.git xv6

Type ls to check whether xv6 is cloned or not

type cd xv6
make
make qemu-nox

xv6 is running under QEMU

2. Add a new system call in xv6

First change to xv6 directory (cd xv6)

step:-1

create a file named square.c using nano

190031187 Radhakrishna

step-2

write the program as follows

```
#include "types.h"
#include "math.h"
#include "user.h"

int main(void)
{
  printf(1, "square value of 9 is %d\n",
                    square(9));
  exit();
}
```

step-3

open syscall.h and add #define

SYS_square 22

step-4

open syscall.c and add

extern int sys_square(void); and

[SYS_square] sys_square,

step-5

open sysproc.c and add following

190031187
Radhakrishna

```
int sys_square (void)
{
    int Num;
    argptr( 0, (void *) & Num, sizeof(Num));
    return Num * Num;
}
```

step-6

open user.h and add int square(int);

step-7

open usys.s and add
SYSCALL(square);

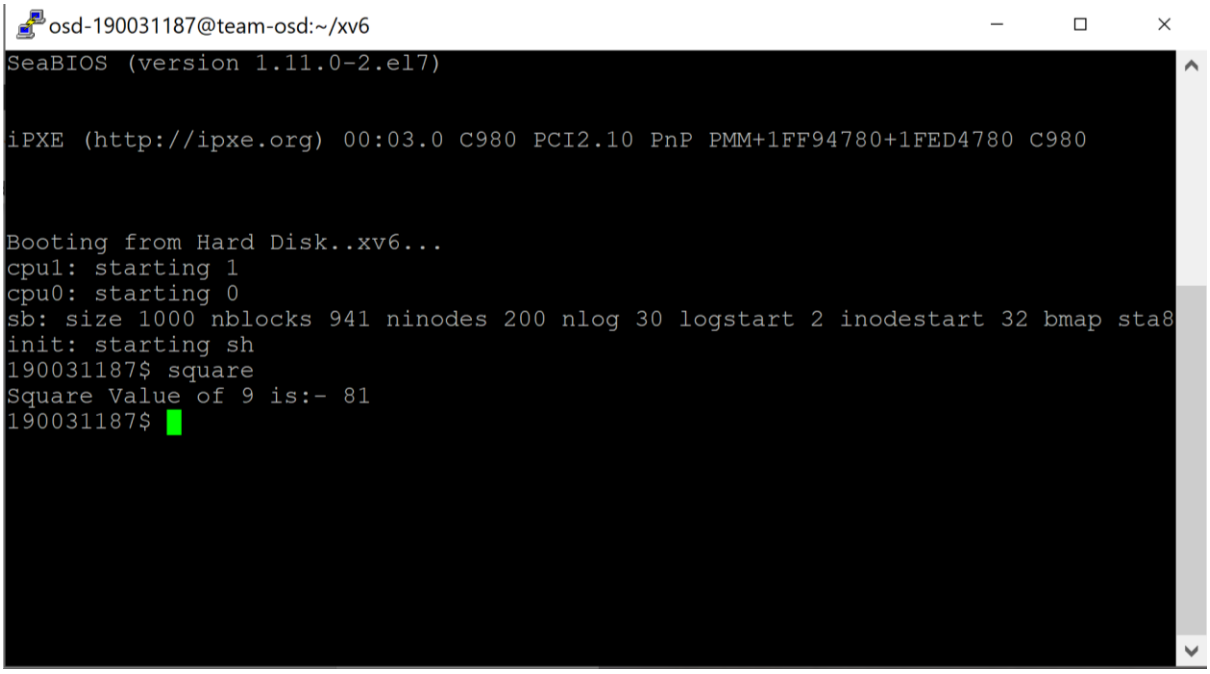
step-8

open nano Makefile and add in
UPROGS = \ _square\ and add
square.c\ in EXTRA = \

step-9

Type make qemu-nox
and square to get Output

OUTPUT


osd-190031187@team-osd:~/xv6
SeaBIOS (version 1.11.0-2.el7)
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980
Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
190031187$ square
Square Value of 9 is:- 81
190031187$

Operating System Design – 19CS2106S

Skill Experiment - 2

1) Manipulating Files and Directories.

OSD Skill-2 190031187
N.V.Radhakrishna

1. Manipulating Files and directories

These are the basic commands for manipulating files and directories

cp - copy files and directories
mv - move or rename files & directories
rm - remove file or directory
mkdir - create directory

2) echo.c(xv6 design implementation)

190031187
Radhakrishna

Explanation of
echo.c

1. purpose of this command is to print the data supplied to the echo command on the screen
2. In main funtion argc will store the count of arguments and argv will store the value of the argument passed
3. The loop repeats for the count of arguments
4. In line 9 printf syntax is used
5. In line 9 first %s will print value of argv[i] and second %s will print space or new line based on the condition.

2. code of echo.c file

```
#include "types.h"
#include "stat.h"
#include "user.h"
int main (int argc, char *argv[])
{   int i;
    for(i=1; i< argc; i++){
        printf(1, "%s%s", argv[i], i+1<argc?" ":
                                    "\n");
    }
    exit();
}
```

3) cp, wc.c (xv6 customization)

190031187
Radhakrishna

3.

cp

1. Type nano cp.c
2. write the below code

```
# include "types.h"
# include "fcntl.h"
# include "stat.h"
# include "user.h"

int main (int argc, char * argv[])
{
  int sourceFD, TargetFD, RDFlag, WrFlag;
  char Data[100];
  sourceFD = open (argv[1], O_RDONLY);
  if (sourceFD < 0)
  {  printf (1, "Error opening source file");
     exit();
  }
  RdFlag = read (sourceFD, Data, sizeof (Data));
  if ( RdFlag < 0)
  {  printf (1, "Error reading source file");
     exit();
  }
  TargetFD = open( argv[2], O_CREATE|O_WRONLY)
  if (TargetFD < 0)
  {
     printf(1, "Error opening target file");
     exit();
  }
```

```
WrFlag = write(TargetFD, Data, sizeof(Data));
if(WrFlag<0)
{
   printf(1,"Error writing target file");
   exit();
}
close(sourceFD);
close(TargetFD);
return 0;
}
```

wordcount.c

```
#include "types.h"
#include "stat.h"
#include "fcntl.h"
#include "user.h"
int main(int argc, char *argv[])
{ int FD, i, charCount=0, WordCount=0;
  int LineCount=0, RdFlag;
  char Data[500];
  FD = open(argv[1], ORDONLY);
  if(FD<0)
  { printf(1,"Error reading the file");
    exit();
  }
```

190031187
Radhakrishna

```
RdFlag = read (FD, Data, sizeof (Data));
if ( RdFlag<0)
{ printf (1, "Error reading the file");
  exit()
}
for( i=0; i<RdFlag ; i++)
{ charCount ++ ;
  if ( Data[i] == ' ' || Data [i] == '\n' )
   wordCount ++;
  if (Data[i]=='\n')
   LineCount ++;
}
close (FD);
printf ( 1, "%d \t %d \t %d \t, charcount,
                wordCount, LineCount);
return 0;
}
```

4. now type nano Makefile

5. Add -cp\
   -wordcount\
   under UPROGS= \

6. Add cp.c wordcount.c
   under EXTRA = \

190031187
Radhakrishna

190031187
Radhakrishna

7. Save
8. Type make
9. Type make qemu-nox
10. check whether the commands are working or not.

OUTPUT

osd-190031187@team-osd:~/xv6

```
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
190031187$ cat>F1.txt
my name is radhakrishna
my id no is 190031187
190031187$ cp F1.txt F2.txt
pid 4 cp: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
190031187$ cat F2.txt
my name is radhakrishna
my id no is 190031187
190031187$
```

osd-190031187@team-osd:~/xv6

```
grep           2 6 16020
init           2 7 14232
kill           2 8 13368
ln             2 9 13316
ls             2 10 16172
mkdir          2 11 13404
rm             2 12 13380
sh             2 13 24820
stressfs       2 14 14324
usertests      2 15 67228
wc             2 16 15152
zombie         2 17 13040
square         2 18 13164
fork1          2 19 13448
prog2          2 20 13588
cp             2 21 14268
wordcount      2 22 14408
console        3 23 0
F1.txt         2 24 46
F2.txt         2 25 100
190031187$ wordcount F1.txt
46      9       2
pid 4 wordcount: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill prc
190031187$
```

Operating System Design – 19CS2106S

Skill Experiment – 3

1. **Working with Commands, Redirection**

OSD 190031187
SKILL 3 Radhakrishna

1. Commands

type - indicate how a command name is interpreted

which - Display which executable program will be executed.

man - Display a commad's manual page

apropos - Display a list of appropriate commands

info - display command's info entry

what is - Display a very brief description of a command

alias - Create an alias for a command

Redirection

cat - concatenate files

sort - sort lines of text

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ type ls
ls is aliased to `ls --color=auto'
[osd-190031187@team-osd ~]$ type cp
cp is /usr/bin/cp
[osd-190031187@team-osd ~]$ which ls
alias ls='ls --color=auto'
        /usr/bin/ls
[osd-190031187@team-osd ~]$ man ls
[osd-190031187@team-osd ~]$ apropos floppy
fd (4)               - floppy disk device
fdformat (8)         - low-level format a floppy disk
mbadblocks (1)       - tests a floppy disk, and marks the bad blocks in the FAT
mformat (1)          - add an MSDOS filesystem to a low-level formatted flopp...
[osd-190031187@team-osd ~]$ whatis ls
ls (1)               - list directory contents
ls (1p)              - list directory contents
[osd-190031187@team-osd ~]$ info coreutils
[osd-190031187@team-osd ~]$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-ti
lde'
[osd-190031187@team-osd ~]$
```

2. exec.c (Xv6 design & implementation)

2. Step:- 1

Go to XV6 directory

step :- 2

Type vi exec-c (or) nano exec.c

Step -3

you will find the following code

```
#include "types.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "defs.h"
#include "x86.h"
#include "elf.h"
int exec(char *path, char **argv)
{
        char *s, *last; int i, off;
        uint argc, sz, sp, ustack[3+MAXARG+1];
        struct elfhdr elf;
        struct inode *ip;
        struct proghdr ph;
        pde_t *pgdir, *oldpgdir;
        struct proc *curproc = myproc();
        begin_op();
        if((ip = namei(path)) == 0)
        {
                end_op();
                cprintf("exec: fail\n");
                return -1;
        }
```

```
        ilock(ip);
        pgdir = 0;
// Check ELF header
        if(readi(ip, (char*)&elf, 0, sizeof(elf)) != sizeof(elf))
                goto bad;
        if(elf.magic != ELF_MAGIC)
                goto bad;
        if((pgdir = setupkvm()) == 0)
                goto bad;
        // Load program into memory.
        sz = 0;
        for(i=0, off=elf.phoff; i< ph.filesz)
                goto bad;
        sz = 0;
        for(i=0, off=elf.phoff; i< ph.filesz)
                goto bad;
        if(ph.vaddr + ph.memsz < ph.vaddr)
                goto bad;
        if((sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
                goto bad;
        if(ph.vaddr % PGSIZE != 0)
                goto bad;
        if(loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)
                goto bad;
        iunlockput(ip);
        end_op();
        ip = 0;
         // Allocate two pages at the next page boundary.
        // Make the first inaccessible. Use the second as the user stack.
        sz = PGROUNDUP(sz);
        if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
                goto bad;
```

```
clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
sp = sz;
// Push argument strings, prepare rest of stack in ustack.
for(argc = 0; argv[argc]; argc++)
{
        if(argc >= MAXARG)
                goto bad;
        sp = (sp - (strlen(argv[argc]) + 1)) & ~3;
        if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)
                goto bad;
        ustack[3+argc] = sp;
}
ustack[3+argc] = 0;
ustack[0] = 0xffffffff;
// fake return PC ustack[1] = argc;
ustack[2] = sp - (argc+1)*4;
// argv pointer sp -= (3+argc+1) * 4;
if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)
        goto bad;
// Save program name for debugging.
for(last=s=path; *s; s++)
        if(*s == '/')
                last = s+1;
safestrcpy(curproc->name, last, sizeof(curproc->name));
// Commit to the user image.
oldpgdir = curproc->pgdir; curproc->pgdir = pgdir;
curproc->sz = sz;
curproc->tf->eip = elf.entry;
// main
curproc->tf->esp = sp; switchuvm(curproc);
freevm(oldpgdir);
return 0;
```

```
    bad:
        if(pgdir) freevm(pgdir);
        if(ip){ iunlockput(ip);
        end_op();
        return -1;
}
```

3. pwd, cd, mv(xv6 customization)

190031187

3. pwd

step 1 :- cd XV6

step 2 :- rename sh.c to shold.c

step 3 :- make a new file sh.c & write the given code

step 4 :- create another new file pwd.c & type the code

step 5 :- Go to makefile and add _pwd\ under UPROGS and pwd.c under EXTRA

step 6 :- make

step 7 :- make qemu-nox

step 8 :- pwd

o/p :- /

step 9 :- mkdir OSD

step 10 :- pwd

o/p :- /OSD/

osd-190031187@team-osd:~/xv6

```
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap 8
init: starting sh
190031187$ pwd
/
190031187$ mkdir OSD
190031187$ cd OSD
190031187$ pwd
/OSD/
190031187$
```

190031187

mv

step 1 :- cd xv6

step 2 :- create a new file mv.c (vi mv.c)

step 3 :- Type the code and save

step 4 :- add _mv \ under UPROGS and mv.c under EXTRA in Makefile

step 5 :- make

step 6 :- make qemu-nox

step 7 :- cat >F1.txt

Welcome to OSD course

step 8 :- mv F1.txt F2.txt

step 9 :- ls

you can observe F1.txt has be renamed as F2.txt.

step 10 :- cat F2.txt

welcome to OSD course

osd-190031187@team-osd:~/xv6

```
Booting from Hard Disk...
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
190031187$ cat>F1.txt
Wlecome to OSD course
190031187$ mv F1.txt F2.txt
190031187$ cat F2.txt
Wlecome to OSD course
```

190031187

cd

step 1 :- cd XV6

step2 :- nano cd-c

step3 :- type code and save

step4 :- nano makefile

step 5 :- add cd-c\ under EXTRA
_cd\ under UPROGS

Step 6 :- make

step 7 :- make qemu-nox

step 8 :- mkdir OSD

step 9 :- cd OSD

step 10 :- pwd

o/p :- /OSD/

you can observe that you are inside OSD directory so cd is working

osd-190031187@team-osd:~/xv6

```
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap 8
init: starting sh
190031187$ pwd
/
190031187$ mkdir OSD
190031187$ cd OSD
190031187$ pwd
/OSD/
190031187$
```

Operating System Design Skill Experiment – 4

1. Seeing the World as The Shell Sees It, Advanced Keyboard Tricks, Permissions

190031187
Radhakrishna

OSD Skill - 4

1. Seeing the world as the shell sees it

* echo - Display a line of text

Ex:- echo hi o/p:- hi

* ls - list the contents of a directory

Advanced keyboard tricks
(mouse) cursor movement commands

ctrl+a : Move cursor to the beginning of the line
ctrl+e : Move curor to the end of the line
Alt+f : Move cursor forward one word
Alt+b : Move cursor backward one word
ctrl+f : Move cursor forward one character
ctrl+d : Delete the character at the cursor location
ctrl+k : kill text from the cursor location to end of line
ctrl+u : kill text from cursor location to beginning of line

permissions

* id - Display user identity
* chmod - change a file's mode
* passwd - change a user's password

File types

- - A regular file
d - directory
l - Symbolic link. The read file attributes are those of the file the symbolic link points to

190031187 Radhakrishna

e - A character special file. This file type refers to a device that handles data as a stream of bytes such as a terminal or modem.

Example for permissions

changing your password

$ passwd

(current) UNIX password:

NEW UNIX password:

The 'passwd' command will try to enforce use of strong passwords. This means it will refuse to accept passwords that are too short too similar to previous passwords.

2. ln.c , rm.c, mkdir.c (Xv6 design & implementation. (xv6 source code))

ln.c

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1                File: ln.c

#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  if(argc != 3){
    printf(2, "Usage: ln old new\n");
    exit();
  }
  if(link(argv[1], argv[2]) < 0)
    printf(2, "link %s %s: failed\n", argv[1], argv[2]);
  exit();
}

                              [ Read 15 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

OUTPUT

```
osd-190031187@team-osd:~/xv6
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
190031187$ cat>F1.txt
hi my id is 190031187
190031187$ ln F1.txt F2.txt
190031187$ cat F2.txt
hi my id is 190031187
190031187$
```

rm.c

```
osd-190031187@team-osd:~/xv6
  GNU nano 2.3.1              File: oldrm.c

#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int i;

  if(argc < 2){
    printf(2, "Usage: rm files...\n");
    exit();
  }

  for(i = 1; i < argc; i++){
    if(unlink(argv[i]) < 0){
      printf(2, "rm: %s failed to delete\n", argv[i]);
      break;
    }
  }

  exit();
}

^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

OUTPUT

```
osd-190031187@team-osd:~/xv6
square          2 18 13164
fork1           2 19 13448
prog2           2 20 13588
cp              2 21 14268
wordcount       2 22 14408
pwd             2 23 15152
mv              2 24 23752
cd              2 25 13064
postlab3        2 26 14472
ForkDemo        2 27 13112
console         3 28 0
temp.pwd        2 32 1
F1.txt          2 30 3
190031187$ rm F1.txt
190031187$ ls
.               1 1 512
..              1 1 512
README          2 2 2170
cat             2 3 14484
echo            2 4 13340
forktest        2 5 8172
grep            2 6 16020
init            2 7 14232
kill            2 8 13368
ln              2 9 13316
ls              2 10 16172
mkdir           2 11 13404
rm              2 12 16880
sh              2 13 27352
stressfs        2 14 14324
usertests       2 15 67228
wc              2 16 15152
zombie          2 17 13040
square          2 18 13164
fork1           2 19 13448
prog2           2 20 13588
cp              2 21 14268
wordcount       2 22 14408
pwd             2 23 15152
mv              2 24 23752
cd              2 25 13064
postlab3        2 26 14472
ForkDemo        2 27 13112
console         3 28 0
temp.pwd        2 32 1
190031187$
```

mkdir.c

osd-190031187@team-osd:~/xv6

```
  GNU nano 2.3.1                File: mkdir.c

#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int i;

  if(argc < 2){
    printf(2, "Usage: mkdir files...\n");
    exit();
  }

  for(i = 1; i < argc; i++){
    if(mkdir(argv[i]) < 0){
      printf(2, "mkdir: %s failed to create\n", argv[i]);
      break;
    }
  }

  exit();
}

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

OUTPUT

osd-190031187@team-osd:~/xv6

```
Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
190031187$ ls
.              1 1 512
..             1 1 512
README         2 2 2170
cat            2 3 14484
echo           2 4 13340
forktest       2 5 8172
grep           2 6 16020
init           2 7 14232
kill           2 8 13368
ln             2 9 13316
ls             2 10 16172
mkdir          2 11 13404
rm             2 12 16880
sh             2 13 27352
stressfs       2 14 14324
usertests      2 15 67228
wc             2 16 15152
zombie         2 17 13040
square         2 18 13164
fork1          2 19 13448
prog2          2 20 13588
cp             2 21 14268
wordcount      2 22 14408
pwd            2 23 15152
mv             2 24 23752
cd             2 25 13064
postlab3       2 26 14472
ForkDemo       2 27 13112
console        3 28 0
temp.pwd       2 32 1
190031187$ mkdir OSD
190031187$ ls
.              1 1 512
..             1 1 512
README         2 2 2170
cat            2 3 14484
echo           2 4 13340
forktest       2 5 8172
grep           2 6 16020
init           2 7 14232
kill           2 8 13368
ln             2 9 13316
ls             2 10 16172
mkdir          2 11 13404
rm             2 12 16880
sh             2 13 27352
stressfs       2 14 14324
usertests      2 15 67228
wc             2 16 15152
zombie         2 17 13040
square         2 18 13164
fork1          2 19 13448
prog2          2 20 13588
cp             2 21 14268
wordcount      2 22 14408
pwd            2 23 15152
mv             2 24 23752
cd             2 25 13064
postlab3       2 26 14472
ForkDemo       2 27 13112
console        3 28 0
temp.pwd       2 32 1
OSD            1 30 32
190031187$
```

Type here to search

3. rm, getpinfo (xv6 customization)

rm.c

osd-190031187@team-osd:~/xv6

GNU nano 2.3.1 File: rm.c Modified

```
#include "types.h" include "stat.h" include "user.h" include "fcntl.h" include "fs.h"

int
delete(char *path)
{
    return unlink(path);
}

char*
strcat(char *d,char *s)
{
    char *temp=d;
    while(*d) ++d;
    while(*s) *d++=*s++;
    *d=0;
    return temp;
}

void
rm_rf(char *path)
{
    char *buff;
    buff=(char*)malloc(512*sizeof(char));
    int fd0;
    struct dirent de;
    struct stat st;
    if(path[strlen(path)-1]=='/') path[strlen(path)-1]=0;
    if(((fd0=open(path,0))<0)
    {
        printf(2,"rm: cannot open %s No such file or directory\n",path);
        exit();
    }
    if(fstat(fd0,&st)<0)
    {
        printf(2,"rm: cannot stat %s No such file or directory\n",path);
        close(fd0);
        exit();
    }
    switch(st.type)
    {
        case T_FILE:
        {
            delete(path);
            break;
        }
        case T_DIR:
        {
            strcpy(buff,path);
            strcat(buff,"/");
            int len=strlen(buff);
            while(read(fd0,&de,sizeof(de))==sizeof(de))
            {
                if(de.inum==0 || de.name[0]=='.') continue;
                memmove(buff+len,de.name,strlen(de.name));
                rm_rf(buff);
                memset(buff+len,'\0',sizeof(buff)+len);
            }
            unlink(path);
            break;
        }
    }
    free(buff);
    close(fd0);
}

int main(int argc,char *argv[])
{
    int i;
    if(argc<2)
    {
        printf(2,"Usage: rm [OPTIONS] [files]...\n");
        printf(2,"Options:\n");
        printf(2,"  -rf : delete file secara recursive\n");
        exit();
    }
    if(strcmp(argv[1],"-rf")==0)
    {
        for(i=2;i<argc;i++)
        {
            rm_rf(argv[i]);
            delete(argv[i]);
        }
    }
    else
    {
        for(i=1;i<argc;i++)
        {
            delete(argv[i]);
        }
    }
    exit();
}
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Type here to search ENG IN 09:51 09-09-2020

OUTPUT

```
osd-190031187@team-osd:~/xv6
square          2 18 13164
fork1           2 19 13448
prog2           2 20 13588
cp              2 21 14268
wordcount       2 22 14408
pwd             2 23 15152
mv              2 24 23752
cd              2 25 13064
postlab3        2 26 14472
ForkDemo        2 27 13112
console         3 28 0
temp.pwd        2 32 1
F1.txt          2 30 3
190031187$ rm F1.txt
190031187$ ls
.               1 1 512
..              1 1 512
README          2 2 2170
cat             2 3 14484
echo            2 4 13340
forktest        2 5 8172
grep            2 6 16020
init            2 7 14232
kill            2 8 13368
ln              2 9 13316
ls              2 10 16172
mkdir           2 11 13404
rm              2 12 16880
sh              2 13 27352
stressfs        2 14 14324
usertests       2 15 67228
wc              2 16 15152
zombie          2 17 13040
square          2 18 13164
fork1           2 19 13448
prog2           2 20 13588
cp              2 21 14268
wordcount       2 22 14408
pwd             2 23 15152
mv              2 24 23752
cd              2 25 13064
postlab3        2 26 14472
ForkDemo        2 27 13112
console         3 28 0
temp.pwd        2 32 1
190031187$
```

getpinfo

1. Open syscall.h and add the following
   #define SYS_getpinfo 22
2. Open defs.h and add the following in //proc.c section
   int getpinfo (void);
3. open user.h file
   add in systemcall section
   int getpinfo (void);
4. open proc.c and add the following function:

```
osd-190031187@team-osd:~/xv6-getpinfo
  GNU nano 2.3.1              File: proc.c
int getpinfo()
{
  struct proc *p;
  //Enables interrupts on this processor.
  sti();

  //Loop over process table looking for process with pid.
  acquire(&ptable.lock);
  cprintf("name \t pid \t state \t \n");
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state == SLEEPING)
        cprintf("%s \t %d \t SLEEPING \t  \n ", p->name,p->pid);
    else if(p->state == RUNNING)
        cprintf("%s \t %d \t RUNNING \t \n ", p->name,p->pid);
    else if(p->state == RUNNABLE)
        cprintf("%s \t %d \t RUNNABLE \t  \n ", p->name,p->pid);
  }
  release(&ptable.lock);
  return 22;
}
```

5. open sysproc.c and add following function:

```
osd-190031187@team-osd:~/xv6-getpinfo
  GNU nano 2.3.1            File: sysproc.c

int
sys_getpinfo(void)
{
  return getpinfo();
}
```

6. open usys.S and add the following
   SYSCALL(getpinfo)
7. open the ***syscall.c*** file and add the following
   //Add this where the other system calls are defined in syscall.c
   extern int sys_getpinfo(void);
   //Add this inside static int (*syscalls[])(void)
   [SYS_getpinfo] sys_getpinfo,
8. Create user program getpinfo.c

```
osd-190031187@team-osd:~/xv6-getpinfo
  GNU nano 2.3.1          File: getpinfo.c

        #include "types.h"
        #include "stat.h"
        #include "user.h"
        #include "fcntl.h"

        int main(void){
          getpinfo ();
          exit();
        }

^G Get Help ^O WriteOut ^R Read Fil ^Y Prev Pag ^K Cut Text ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Pag ^U UnCut Te ^T To Spell
```

9. nano Makefile
   Add getpinfo\ to uprogs
   Add getpinfo.c to Extra
10. make
11. make qemu-nox
12. $ getpinfo

OUTPUT

```
osd-190031187@team-osd:~/xv6-getpinfo
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED0



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodesta8
init: starting sh
$ getpinfo
name     pid     state
init     1       SLEEPING
 sh      2       SLEEPING
 getpinfo        3       RUNNING
 $ sh
$ getpinfo
name     pid     state
init     1       SLEEPING
 sh      2       SLEEPING
 sh      4       SLEEPING
 getpinfo        5       RUNNING
 $
```