DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIX OPERATING SYSTEM LABORATORY

B20CS0402

for

Fourth Semester

B.Tech in Computer Science and Engineering

Name	
SRN	
Branch	
Semester	
Section	
Academic Year	

Vision of the Department

Department of Computer Science and Engineering aspires to create a pool of high-calibre technologists and researchers in the field of computer science and engineering who have potential to contribute for development of the nation and society with their expertise, skills, innovative problem-solving abilities and strong ethical values.

Mission of the Department

MD1: To create a center of excellence where new ideas flourish and from which emerge tomorrow's researchers, scholars, leaders, and innovators.

MD2: Provide quality education in both theoretical and applied foundations of computer science and related inter-disciplinary areas and to train students to effectively apply the education to solve real-world problems.

MD3: Amplify student's potential for life-long high-quality careers and give them a competitive in the ever-changing and challenging global work environment.

MD4: Forge research and academic collaboration with industries and top global universities in order to provide students with greater opportunities.

MD5: Support the society by encouraging and participating in technology transfer.

COURSE DESCRIPTION

UNIX operating system provides a practical case of operating systems for the user to understand and master deeply and tangibly the theory and algorithms in operating systems. It gives deeper insights into the hierarchical structure, principles, applications, shells, development, and management of the UNIX operation system multi-dimensionally, systematically and from the elementary to the profound. It makes the user to understand about how UNIX operating system functions.

Course Objectives:

The objectives of this course are to:
Explain the history, basics and structure of UNIX Operating System

Describe UNIX process concepts and scheduling techniques

Illustrate the use of different memory management techniques of UNIX.

Describe UNIX kernel, data structures and internal representation of files in UNIX operating

Course Outcomes (Cos):

On successful completion of this course; student shall be able to:

CO#	Course Outcomes	Pos	PSOs
CO1	Outline the history of Operating System and UNIX environment.	1,2,5,9,10,12	1,3
CO2	Develop the programs to implement the different process states, attributes and control the process in foreground and	1,4,5, 9,10,12	1,3
CO3	Compare and analyze the performance of different memory management techniques.	1,4,5, 9,10,12	2,3
CO4	Make use of UNIX file types, file structure and file system implementation.	1,2,4, 5, 9,10,12	1,3
CO5	Summarize the issues of Inter-process Communication.	1,2,4,5, 9,10,12	1,3
CO6	Make use of system calls for managing processes, memory and the file system.	1,4,5, 9,10,12	1,3

BLOOMS LEVEL OF THE COURSE OUTCOMES

	Bloom's Level					
	Remember (L1)	Understand (L2)	Apply (L3)	Analyze (L4)	Evaluate (L5)	Create (L6)
CO1		√				
CO2			√			
CO3				√		
CO4			√			
CO5		√				
CO6			√			

COURSE ARTICULATION MATRIX

CO#/	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3			1				3	3		2	3		3
CO2	3			3	3				3	3		2	3		3
CO3	3			3	3				3	3		2		3	3
CO4	3	3		3	2				3	3		2	3		3
CO5	3	3		2	3				3	3		2	3		3
CO6	3			3	3				3	3		2	3		3

INDEX

1.Lab Requirements

Following are the required hardware and software for this lab, which is available in the laboratory.

- **Hardware:** Desktop system or Virtual machine in a cloud with OS installed. Presently in the Lab, Pentium IV Processor having 1 GB RAM and 250 GB Hard Disk is available. Desktop systems are dual boot having Windows as well as Linux OS installed on them.

- **Software:** Editor Tool included in the Linux package or Code Blocks in Widows operating environment used to execute the programs written in C Programming language.

Guidelines to Students

- ➢ Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
- ➢ Students are required to carry their observation / programs book with completed exercises while entering the lab.
- ➢ Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- ➢ Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- ➢ Lab records need to be submitted on or before date of submission.
- ➢ Students are not supposed to use flash drives.
- ➢ In SEE exam one Data base application will be asked and the set of some queries will be given in the final exam and evaluated for 50 marks and scale down to 25 marks.

Introduction to Unix and Operating system

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

Features of UNIX

- Multiuser System : Unix provides multiple programs to run and compete for the attention of the CPU
- Multitask System : A single user may run multiple tasks concurrently.
- The Building-Block Approach
- The UNIX Toolkit :
- Pattern Matching :
- Programming Facility :
- Documentation :

No	Title of the Experiment	Tools and Techniques	Expected Skill /Ability
	Part-A		
1.	a) Execute at least ten UNIX shell commands on the terminal and the use of the shell commands.	Linux OS	shell commands.
	b) Write a C/Java program to display the output of any UNIX shell command.	Linux OS	
2.	a) Write a C/Java program to create a sub process by printing its pid and the main process pid value.	Linux OS	Process Control.
	b) Write a C/Java program to show the process is an orphan process and print its parent pid value.		
3.	a) Write a C/Java program that creates a zombie and then calls system to execute the ps command to verify that the process is zombie.	Linux OS	Zombie Process.
	b) Write a C/Java program to avoid zombie process by forking twice.		
4.	a) Write a C/Java program that outputs the contents of its Environment list.	Linux OS	Process Control.
	b) Write a C/Java program to illustrate the race		
5.	a) Write a C/Java to create hard link and soft link and display the hard link count with other attributes of the created file within the sample code.	Linux OS	File Types and File attributes
	b) Consider the last 100 bytes as a region. Write a C/Java program to check whether the region is locked or not. If the region is locked, print pid of the process which has locked. If the region is not locked, lock the region with an exclusive lock, read the last 50 bytes and unlock the region.		
6.	a) Write a C/Java program which demonstrates Interprocesss communication between a reader process and a writer process of a FIFO file	Linux OS	Inter Process Communication.
	b) Write a C/Java program which demonstrates the signal handler function to handle the signal sent by the process.		UNIX signals.
	Part-B (Mini Project: Bank Management System)		

	Bank System is based on the concept of recording customer's account details. The system contains only the admin section. Here the admin can perform all the tasks like creating an account, deposit and withdraw amount, check balance, view all account holder. It contains the following modules of account creation. 1. Customer Module 2. Transaction Module	Linux OS	Modules of Bank Management System
1	Write a C/Java program to create account of a customer of Bank Management System and display the	Linux OS	Create a class bank to create account and display the
2	Write a C/Java program to close or delete an account of a created customer accounts of Bank Management	Linux OS	To close an existing account.
3	Write a C/Java program to display all account holders of a created customer accounts of Bank Management	Linux OS	To display all account holders.
4	Write a C/Java program to modify an account of a created customer accounts of Bank Management	Linux OS	To modify account and display the contents.
5	Write a C/Java program to deposit amount of a created customer account of bank Management System and display the contents	Linux OS	To deposit amount and display the contents.
6	Write a C/Java program to deposit and withdraw amount of a created customer account of bank	Linux OS	To withdraw amount and display the contents.
7	Write a C/Java program to check the balance amount of a created customer account of bank Management System and display the contents.	Linux OS	To check balance and display the contents.
8	Write a C/Java program to intergrate the above modules scenario and display each module contents.	Linux OS	Complete knowledge of the project.

	EXTRA PROGRAMS		
1	Write a C/C++ POSIX compliant program to check the following limits: (i) No. of clock ticks (ii) Max. no. of child processes (iii) Max. path length (iv) Max. no. of characters in a file name (v) Max. no. of open files/ process	LINUX OS	Commands
2	Write a C/C++ POSIX compliant program that prints the POSIX defined configuration options supported on any given system using feature test macros.	LINUX OS	System functions
3	Write a C/C++ program to implement the system function.	LINUX OS	System functions
4	Write a C/C++ program to set up a real-time clock interval timer using the alarm API.	LINUX OS	System functions

Name of the Faculty In charge:

Program/ Experiment No	**Title of the Program /Experiment**	**Date of Execution**	**Max Marks**	**Marks Obtained**	**Signature of the Faculty I/c**
1			10		
2			10		
3			10		
4			10		
5			10		
Internal Assessment-1 (IA-1) Consolidated Marks			50/5		

Program/ Experiment No	**Title of the Program /Experiment**	**Date of Execution**	**Max Marks**	**Marks Obtained**	**Signature of the Faculty I/c**
6			10		
7			10		
8			10		
9			10		
Additional Session (1-5)			10		
Internal Assessment-2 (IA-2) Consolidated Marks			50/5		
		IA	Max Marks	Marks Obtained	Signature of the Faculty
		IA1			
		IA2			
		IA1+IA2	20		

Signature of the Faculty-in-charge with date

Semester End Lab Examination Evaluation Procedure 2021-22

Name of the Lab: Unix Operating System Lab

Question	**Parameters to be considered**	**Marks distribution**
1-10	Write-up	**10**
	Conduction & Results	**15**
	Viva	**5**
	Total	**30**

Note:

1. Lab course is conducted for a total of 50 Marks:
1. 20 Marks –Continuous Evaluation
2. 30 Marks Semester End Examination

3. Passing Criterion:
1. 08/30 in Semester End Examination
2. 20/50 overall

1A) Execute atleast ten UNIX shell commands on the terminal and the use of the shell commands.

Explanation:

A shell is a program whose primary purpose is to read commands and run other programs. The default shell in many implementations of UNIX is bash. The shell's main advantages are its high action-to-keystroke ratio, its support for automating repetitive tasks, and its capacity to access networked machines.

Program:

- mkdir is Used to create a new file directory.
- $ mkdir thesis
-
- ls To list all the folders present in the directory
- $ ls **-F**
-
- pwd to print current working directory
- $ pwd
-
- cd is used for changing current directory
- $ cd thesis
-
- Touch and nano are used To create a text file
- $ cd thesis $ nano draft.txt or $ touch draft.txt
-
- mv is used for Renaming a file
- $ mv thesis/draft.txt thesis/quotes.txt
-
- cp is for copying text file (where the new file is named quotations)
- cp quotes.txt thesis/quotations.txt
-
- Removing file
- $ rm quotes.txt
-
- To invoke custom help
- $ ls –help
-
- wc is used to count the number of words in the files
- $ wc quotations.txt

1b) Write a C/Java program to display the output of any UNIX shell command

The UNIX shell commands can also be used through other programming languages. This enables us to easily handle system programming, automate tasks, enhance the features of programs etc.

Algorithm:

1.Get the desired command from the user.

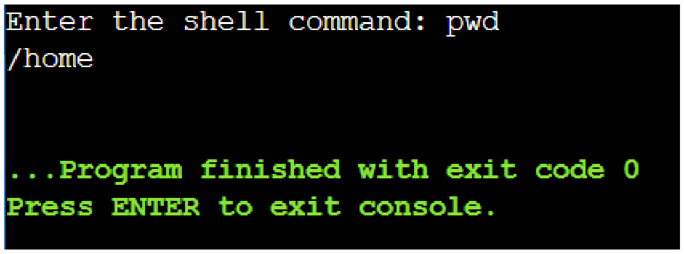
2. if the received command is valid, then execute the command via “system” library function.

3. else, print the error message.

Program

```
#include<stdio.h>

#include<stdlib.h>

int main()

{

char cmd[50];

printf("Enter the UNIX shell command: ");

fgets(cmd, 50, stdin);

system(cmd);

return 0;

}
```

Output:



```
Enter the shell command: pwd
/home


...Program finished with exit code 0
Press ENTER to exit console.
```

2 a) Write a program to create one parent with three child using fork() function where each process find its Id. using getpid()

```
// CPP code to create three child process of a parent
 #include <stdio.h>
 #include <stdlib.h>
 #include <unistd.h>

// Driver code
int main()
{
        int pid, pid1, pid2;
        pid = fork();   // variable pid will store the value returned from fork() system call
                        // If fork() returns zero then it means it is child process.
         if (pid == 0)
        {
                // First child needs to be printed  later hence this process is made  to sleep for 3
                seconds. sleep(3);
                // This is first child process  getpid() gives the process  id and getppid() gives the
                // parent id of that process.
                printf("child[1] --> pid = %d and ppid = %d\n", getpid(), getppid());

         }
        else
        {
                pid1 = fork();
                        if (pid1 == 0)

                        {

                            sleep(2);

                             printf("child[2] --> pid = %d and ppid = %d\n", getpid(), getppid());

                          }
                     else
```

```
                        {
                                pid2 = fork();

                                if (pid2 == 0)

                                {

                                // This is third child which is  needed to be printed first.
                                printf("child[3]  -->  pid  =  %d  and  ppid  =  %d\n",

                                        getpid(), getppid());

                                 }
                // If value returned from fork()  in not zero and >0 that means   this is parent
                process.
                else

                 {

                // This is asked to be printed at last  hence made to sleep for 3 seconds.
                 sleep(3);

              printf("parent --> pid = %d\n", getpid());


                  }


                      }

           }
       return 0;
}
```

OUTPUT:

```
thanmaisai@THANMAIs-MacBook-Air unixlab % cc prgm2.c
thanmaisai@THANMAIs-MacBook-Air unixlab % ./a.out child[3] --> pid =
50773 and ppid = 50770

child[2] --> pid = 50772 and ppid = 50770 child[1] --> pid =
50771 and ppid = 50770 parent --> pid = 50770
```

2b) **Write a C/Java program to show the process is an orphan process and print its parent pid value.**

Explanation

An orphan process is a computer process whose parent process has finished or terminated, though it remains running itself.

Program:

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

int main()

{

   // fork() Create a child process


   int pid =
   fork();

    if (pid> 0)

    {

      //getpid() returns process id  while getppid() will return parent process id

      printf("Parent process\n");

      printf("ID : %d\n\n", getpid());

     }

   else if (pid == 0)

    {
      printf("Childprocess\n
      ");

      // getpid() will return process id of child process

      printf("ID: %d\n", getpid());

      // getppid() will return parent process id of child process

      printf("Parent -ID: %d\n\n", getppid());

      sleep(10);

      // At this time parent process has finished. So if u will check parent process id it will
      show different process id

       printf("\nChild process \n");
```

```
        printf("ID: %d\n", getpid());
         printf("Parent-ID: %d\n",getppid());
       }
    else {
        printf("Failed to create child process");
           }
    return 0;
  }
```

OUTPUT:

```
Parent process ID :
        51418
Child process ID:
51419
Parent -ID: 51418


thanmaisai@THANMAIs-MacBook-Air unixlab % Child
process
ID: 51419
Parent -ID: 1
```

3a)Write a C/C++ program that creates a zombie and then calls system to execute the ps command to verify that the process is zombie.

Explanation:

A zombie process or defunct process is a process that has completed execution but still has an entry in the process table. This entry is still needed to allow the parent process to read its child's exit status. The term zombie process derives from the common definition of zombie - an undead person.

Algorithm:

Step 1: Start

Step 2: Create a process

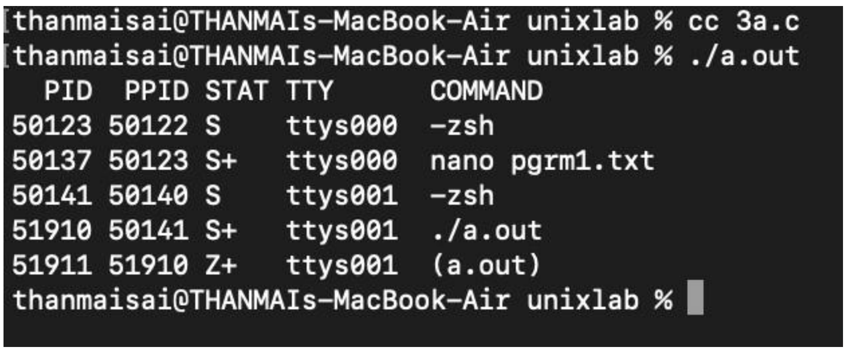
Step 3: If process is child, exit and go to Step 6 Step 4: If process is parent, display process status Step 5: Illustrate zombie state

Step 6: End

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
        int main()
        {
        int pid;
        if((pid=fork())<0)
        printf("fork error\n");
        else if(pid==0)
        _exit(0);
        sleep(2);
        system( "ps -o pid,ppid,state,tty,command");
        _exit (0);
        }
```

OUTPUT:



```
[thanmaisai@THANMAIs-MacBook-Air unixlab % cc 3a.c
[thanmaisai@THANMAIs-MacBook-Air unixlab % ./a.out
  PID  PPID STAT TTY      COMMAND
50123 50122 S    ttys000  -zsh
50137 50123 S+   ttys000  nano pgrm1.txt
50141 50140 S    ttys001  -zsh
51910 50141 S+   ttys001  ./a.out
51911 51910 Z+   ttys001  (a.out)
thanmaisai@THANMAIs-MacBook-Air unixlab %
```

3b) Write a C/C++ program to avoid zombie process by forking twice

Explanation

When a process terminates, either normally or abnormally, the kernel notifies the parent by sending the SIGCHLD signal to the parent. Because the termination of a child is an asynchronous event - it can happen at any time while the parent is running - this signal is the asynchronous notification from the kernel to the parent. The parent can choose to ignore this signal, or it can provide a function that is called when the signal occurs: a signal handler. A process that calls wait or waitpid can:

- Block, if all of its children are still running
- Return immediately with the termination status of a child, if a child has terminated and is waiting for its termination status to be fetched

Return immediately with an error, if it doesn't have any child processes.

Algorithm:

```
Step 1: Start
Step 2: Create a process
Step 3: If child, create another process Go to Step 5
Step 4: If parent, Wait for the death of the child. Go to Step 8 Step 5: If parent, exit and go to
Step 8
Step 6: If child, Sleep for 20 secs
Step 7: print the process status (zombie not created) Step 8: End
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
int pid;
pid = fork();

if (pid == 0)
{
```

```
// First child
pid = fork();
if (pid == 0)
 {
// Second child
sleep(1);
printf("Second child: My parent PID is %d\n", getppid());
}
}
Else
 {
// Parent process
wait(NULL);
sleep(2);
system("ps -o pid,ppid,state,tty,command");
}
return 0;
}
```

OutPut:

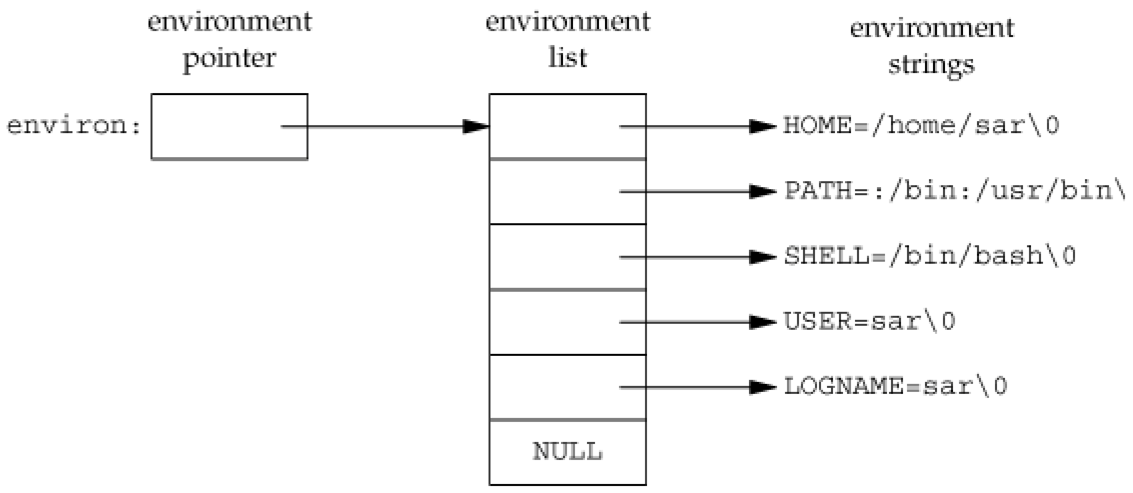
```
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ gcc lab2.c
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ ./a.out
Second child: My parent PID is 1576
    PID    PPID S TT       COMMAND
   3177    3168 S pts/0    bash
   9736    3177 S pts/0    ./a.out
   9740    9736 S pts/0    sh -c ps -o pid,ppid,state,tty,command
   9741    9740 R pts/0    ps -o pid,ppid,state,tty,command
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$
```

4 a) Write a C/C++ program that outputs the contents of its Environment list.

Explanation:

Each program is also passed an environment list. Like the argument list, the environment list is an array of character pointers, with each pointer containing the address of a null-terminated C string. The address of the array of pointers is contained in the global variable environ:

extern char **environ;



By convention the environment consists of *name=value* strings as shown in above figure. Historically, most Unix systems have provided a third argument to the main function that is the address of the environment list:

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

Since ANSI C specifies that the main function be written with two arguments, and since this third argument provides no benefit over the global variable environ, POSIX.1 specifies that environ should be used instead of the third argument. Access to specific environment variables is normally through the getenv and putenv functions instead of through the environ variable.

The environment strings are usually of the form: ***name=value.*** The UNIX kernel never looks at these strings; their interpretation is up to the various applications. The shells, for example, use numerous environment variables. Some, such as HOME and USER, are set automatically at login, and others are for us to set.

Algorithm: Main

Step 1: Start

Step 2: Create an environment list (name=value format)

Step 3: Execute a program that can display all environment variables

Step 4: End

Algorithm: Environment variables display

Step 1: Start

Step 2: Loop over global variable “environ” or third argument in main function

Step 3: Display each environment variable name=value format Step 4: End

Program:

```
#include <stdio.h>

void main(int argc, char *argv[], char * envp[])

{

int i;

for (i = 0; envp[i] != NULL; i++)

{

printf("\n%s", envp[i]);

}

}
```

OutPut:

```
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ gcc lab2.c
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ ./a.out

SHELL=/bin/bash
SESSION_MANAGER=local/candywood-Lenovo-IdeaPad-S540-15IML-
D:@/tmp/.ICE-unix/1835,unix/candywood-Lenovo-IdeaPad-S540-15IML-D:/tmp/.ICE-
unix/1835
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LANGUAGE=en_IN:en
GNOME_SHELL_SESSION_MODE=ubuntu
```

```
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1801
GTK_MODULES=gail:atk-bridge
PWD=/home/candywood/Documents
LOGNAME=candywood
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/candywood
USERNAME=candywood
IM_CONFIG_PHASE=1
LANG=en_IN
XDG_CURRENT_DESKTOP=ubuntu:GNOME
VTE_VERSION=6003
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/add717af_1f37_4251_9e
00_1b12b0a6a836
INVOCATION_ID=68322f05151f441b86f9466ea57b35f1
MANAGERPID=1576
GJS_DEBUG_OUTPUT=stderr
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=candywood
GNOME_TERMINAL_SERVICE=:1.97
DISPLAY=:1
SHLVL=1
QT_IM_MODULE=ibus
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=8:40619
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
PATH=/home/candywood/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin:/usr/games:/usr/local/games:/snap/bin
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
_=./a.outcandywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$
```

4b)Write a C/C++ program to avoid zombie process by forking twice.

Explanation

When a process terminates, either normally or abnormally, the kernel notifies the parent by sending the SIGCHLD signal to the parent. Because the termination of a child is an asynchronous event - it can happen at any time while the parent is running - this signal is the asynchronous notification from the kernel to the parent. The parent can choose to ignore this signal, or it can provide a function that is called when the signal occurs: a signal handler. A process that calls wait or waitpid can:

- Block, if all of its children are still running
- Return immediately with the termination status of a child, if a child has terminated and is waiting for its termination status to be fetched

Return immediately with an error, if it doesn't have any child processes.

Algorithm:

Step 1: Start

Step 2: Create a process

Step 3: If child, create another process Go to Step 5

Step 4: If parent, Wait for the death of the child. Go to Step 8 Step 5: If parent, exit and go to Step 8

Step 6: If child, Sleep for 20 secs

Step 7: print the process status (zombie not created) Step 8: End

Program:

```
#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

static void charatatime(char *);

int main(void)

{

pid_t pid;

if ((pid = fork()) < 0)

{

printf("fork error");

}

else if (pid == 0)
{
charatatime("output from child\n");
}
else
{
charatatime("output from parent\n");
}
return 0;
}
static void charatatime(char *str)
{
char *ptr;
int c;
setbuf(stdout, NULL); /* set unbuffered */
for (ptr = str; (c = *ptr++) != 0; )
putc(c, stdout);
}
```

Output:

```
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ gcc lab2.c
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ ./a.out
output from parent
output from child
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$
```

5A) Write a C/Java to create hard link and soft link and display the hard link count with other attributes of the created file within the sample code.

Explanation

1. Hard Links

- Each hard linked file is assigned the same Inode value as the original, therefore they reference the same physical file location. Hard links more flexible and remain linked even

if the original or linked files are moved throughout the file system, although hard links are unable to cross different file systems.
- ls -l command shows all the links with the link column shows number of links.
- Links have actual file contents
- Removing any link, just reduces the link count, but doesn't affect other links.
- Even if we change the filename of the original file then also the hard links properly work.

2. Soft Links

- A soft link is similar to the file shortcut feature which is used in Windows Operating systems. Each soft linked file contains a separate Inode value that points to the original file. As similar to hard links, any changes to the data in either file is reflected in the other. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).
- ls -l command shows all links with first column value l? and the link points to original file.
- Soft Link contains the path for original file and not the contents.
- Removing soft link doesn't affect anything but removing original file, the link becomes "dangling" link which points to nonexistent file.
- A soft link can link to a directory.
- Size of a soft link is equal to the name of the file for which the soft link is created. E.g If name of file is file1 then size of it's soft link will be 5 bytes which is equal to size of name of original file.
- If we change the name of the original file then all the soft links for that file become dangling i.e. they are worthless no

Algorithm:

```
Step 1: Start
Step 2: Create a file for modification
Step 3: create another file, and create a hard link between the two using the link() function in the c library.
Step 4: Create two more files. Pass the files as arguments to the symlink function.
Step 6: Assign the result of the symlink function to a variable , which serves as a link from a file to an existing file.
Step 7: Display the number of hard links in the code
Step 8: End
```

Program:

```
// C program to create an Hard Link  to the existing file

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>
```

```
// Driver Code
int main(int argc, char* argv[])
{
// Link function
int l = link(argv[1], argv[2]);
// argv[1] is existing file name
// argv[2] is link file name
if (l == 0)
 {
printf("Hard Link created    succuessfuly\n");
}
// Symlink function
int sl = symlink(argv[3], argv[4]);
// argv[1] is existing file name
// argv[2] is soft link file name
if (sl == 0)
 {
printf("Soft Link created"  succuessfuly\n");
}
int mode;
/*check a files attributes*/
mode = access("/home/candywood/Documents/f1.txt",0);
if(mode)
printf("File does not exist.\n");
else
/*check if the file can be written to*/
mode = access("/home/candywood/Documents/f1.txt",2);
if(mode)
printf("File cannot be written.\n");
else
printf("file can be written to.\n");
/*check if file can be read*/
mode = access("/home/candywood/Documents/f1.txt",4);
if(mode)
printf("File cannot be read.\n");
else
printf("File can be read.\n");

/*check if afile can be read/written*/
mode = access("/home/candywood/Documents/f1.txt",6);
if(mode)
printf("File cannot be read/written to.\n");
else
printf("File can be read/written to.\n");
system("ls -qAi | awk '{print $1}' | sort | uniq -d | wc -l");
}
```

IMPORTANT NOTE:

To run the file, compile it.
Then provide the file names as arguments.
1st arguemnt file should be an existing file in your directory.
3rd argument file should be an existing file in your directory.
2nd and 4th arguments should preferably be the new hard link file (.txt file) and new soft link file (.txt file) respectively.
So, the example input would be ,
./a.out f1.txt hlink.txt f2.txt slink.txt

while giving access to file, replace the given program line with the address of the file you are giving as argument.
Example, if f1 is the file whose properties u want to read, put the address of that file in the access() function.

Output :

```
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ gcc lab2.c
```

```
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ ./a.out f1.txt
hl.txt f2.txt sl.txt
Hard Link created succuessfuly
Soft Link created succuessfuly
file can be written to.
File can be read.
File can be read/written to.
1
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$
```

5B) Consider **the last 100 bytes as a region. Write a C/Java program to check whether the region is locked or not. If the region is locked, print pid of the process which has locked. If the region is not locked, lock the region with an exclusive lock, read the last 50 bytes and unlock the region**.

Explanation

- Multiple processes performs read and write operation on the same file concurrently.
- This provides a means for data sharing among processes, but it also renders difficulty for any process in determining when the other process can override data in a file.
- So, in order to overcome this drawback UNIX and POSIX standard support file locking mechanism.
- File locking is applicable for regular files.

UNIX systems provide fcntl function to support file locking. By using fcntl it is possible to impose read or write locks on either a region or an entire file

The prototype of fcntl is:

```
#include<fcntl.h>

int fcntl(int fdesc, int cmd_flag, ....);
```

- **The first argument specifies the file descriptor for a file to be processed.**
- The second argument cmd_flag specifies what operation has to be performed.
- The possible *cmd_flag* values are defined in the <fcntl.h> header. The specific values for file locking and their uses are:

cmd_flag	**Use**
F_SETLK	**sets a file lock, do not block if this cannot succeed immediately.**
F_SETLKW	**sets a file lock and blocks the process until the lock is acquired.**
F_GETLK	**queries as to which process locked a specified region of file.**

- For file locking purpose, the third argument to fctnl is an address of a *struct flock* type variable.

- This variable specifies a region of a file where lock is to be set, unset or queried. The *struct flock* is declared in the <fcntl.h> as:

```
struct flock

{

        short l_type;           /* what lock to be set or to unlock file */ short
        l_whence;               /* Reference address for the next field */ off_t
        l_start ;               /*offset from the l_whence reference addr*/

        off_t l_len ;           /*how many bytes in the locked region */ pid_t
        l_pid ;                 /*pid of a process which has locked the file */

};
```

- The l_type field specifies the lock type to be set or unset.
- The possible values, which are defined in the <fcntl.h> header, and their uses are

***l_type* value**	**Use**
F_RDLCK	**Set a read lock on a specified region**
F_WRLCK	**Set a write lock on a specified region**
F_UNLCK	**Unlock a specified region**

- The l_whence, l_start & l_len define a region of a file to be locked or unlocked.
- The l_whence field defines a reference address to which the l_start byte offset value is added. The possible values of *l_whence* and their uses are:

***l_whence* value**	**Use**
SEEK_CUR	**The *l_start* value is added to current file pointer address**
SEEK_SET	The *l_start* value is added to byte 0 of the file
SEEK_END	The *l_start* value is added to the end of the file

Algorithm

Step 1: Start

Step 2: Open a file for reading and writing

Step 3: Try to obtain an exclusive lock for the last 100 bytes of the file

Step 4: Check, if some process has put a read lock

Step 5: If yes, print the message “process.. has put a read lock”, wait until process unlocks

Step 6: If no, check if some process has put a write lock

Step 7: If yes, print the message "process .. has put a write lock", wait until process unlocks

Step 8: Put an exclusive lock with waiting option for last 100 bytes

Step 9: Simulate some 10 sec delay for processing

Step 10: Read last 50 bytes of data from the file into a memory buffer

Step 11: Display the buffer Step 12: Unlock the file Step 13: End

Program

```
 #include <stdio.h>
 #include<stdlib.h>
  #include<unistd.h>
 #include <fcntl.h>
  #include <errno.h>
   int main(int argc,char *argv[])
   {
              int fd;
              char buffer[255];
               struct flockfvar;
              if(argc==1)
              {
                         printf("usage: %sfilename\n",argv[0]);
                          return -1;
              }
              if((fd=open(argv[1],O_RDWR))==-1)
              {
                         perror("open");
                         exit(1);
              }
              fvar.l_type=F_WRLCK;
              fvar.l_whence=SEEK_END;
              fvar.l_start=SEEK_END-100;
              fvar.l_len=100;
              printf("press enter to set lock\n");
              getchar();
```

```
        printf("trying to getlock..\n");
        if((fcntl(fd,F_SETLK,&fvar))==-1)
        {       fcntl(fd,F_GETLK,&fvar);
                printf("\nFile already locked by process(pid):
\t%d\n",fvar.l_pid);
                return -1;
        }
        printf("locked\n"); if((lseek(fd,SEEK_END-
        50,SEEK_END))==-1)
        {
                  perror("lseek"); exit(1);
        }
        if((read(fd,buffer,100))==-1)
        {
                  perror("read");
                  exit(1);
        }
        printf("data read fromfile..\n");
        puts(buffer);
        printf("press enter to release lock\n"); getchar();
        fvar.l_type = F_UNLCK;
        fvar.l_whence =SEEK_SET;
        fvar.l_start = 0;
        fvar.l_len = 0;
        if((fcntl(fd,F_UNLCK,&fvar))==-1)
        {
                  perror("fcntl"); exit(0);
        }
        printf("Unlocked\n");
        close(fd);
```

```
                return 0;
        }
```

```
thanmaisai@THANMAIs-MacBook-Air unixlab % ./a.out new.c
press enter to set lock

trying to get lock..
locked
data read from file..
ked\n");
                close(fd);
                return 0;
  }V?
press enter to release lock
Unlocked

```

6A. Write a C/Java program which demonstrates Interprocess communication between a reader process and a writer process of a FIFO file by using the corresponding API's.

EXPLAINATION:

Fifo file:

It is a special pipe device file which provides a temporary buffer for two or more processes to communicate by writing data to and reading data from the buffer. The size of the buffer is fixed to PIPE_BUF. Data in the buffer is accessed in a first-in-first-out manner. The buffer is allocated when the first process opens the FIFO file for read or write. The buffer is discarded when all processes close their references (stream pointers) to the FIFO file. Data stored in a FIFO buffer is temporary.

The prototype of mkfifo is:

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

open:

This is used to establish a connection between a process and a file i.e. it is used to open an existing file for data transfer function or else it may be also be used to create a new file. The returned value of the open system call is the file descriptor (row number of the file table), which contains the inode information.

The prototype of open function is,

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

read:

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

The prototype of read function is:

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

write:

The write system call is used to write data into a file. The write function puts data to a file in the form of fixed block size referred by a given file descriptor.

The prototype of write is

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

close:

The close system call is used to terminate the connection to a file from a process.

The prototype of the close is

```
#include<unistd.h>
int close(int fdesc);
```

PROGRAM:

Program 1(Writes first)

```
// C program to implement one side of FIFO
// This side writes first, then reads
 #include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>


int main()
{
        int fd;
        // FIFO file path
        char * myfifo = "/tmp/myfifo";


        // Creating the named file(FIFO)
        // mkfifo(<pathname>, <permission>)
        mkfifo(myfifo, 0666);
        char arr1[80], arr2[80];
        while (1)
        {
                // Open FIFO for write only fd
                = open(myfifo, O_WRONLY);


                // Take an input arr2ing from user.
                // 80 is maximum length
                fgets(arr2, 80, stdin);


                // Write the input arr2ing on FIFO
                // and close it
                write(fd, arr2, strlen(arr2)+1);
                close(fd);
                // Open FIFO for Read only
                fd
                =open(myfifo,O_RDONLY);


                // Read from FIFO
                read(fd, arr1, sizeof(arr1));


                // Print the read message
                printf("User2:%s\n", arr1); close(fd);
            }


            return 0;
        }
```

Program 2(Reads First)

```
    // C program to implement one side of FIFO
    // This side reads first, then reads
    #include <stdio.h>
    #include <string.h>
    #include <fcntl.h>
    #include <sys/stat.h>
    #include <sys/types.h>
    #include <unistd.h>


    int main()
    {
        int fd1;
        // FIFO file path
        char * myfifo = "/tmp/myfifo";
        // Creating the named file(FIFO)
        // mkfifo(<pathname>,<permission>)
        mkfifo(myfifo, 0666);
        char str1[80], str2[80];
        while (1)
        {
                // First open in read only and read
                fd1 = open(myfifo,O_RDONLY);
                read(fd1, str1, 80);

                // Print the read string and close
                printf("User1: %s\n", str1);
                close(fd1);

                // Now open in write mode and write
                // string taken from user.fd1
                =open(myfifo,O_WRONLY);
                fgets(str2, 80, stdin);
                write(fd1, str2, strlen(str2)+1);
                close(fd1);
        }
        return 0;

}
```

OUTPUT:

Note: we have to open two sub terminals and run the files in the sub terminal's

```
Terminal1: $ cc filename.c
                $ ./a.out

Terminal2: $ cc filename2.c
                 $ ./a.out
```

The enter the command in any of the terminal and we can see the entered message in the another terminal.

6B) Write a C/ java program which demonstrates the signal handler function to handle the signal sent by the process.

EXPLAINATION:

If a process wishes to handle certain signals then in the code, the process has to register a signal handling function to the kernel. The signal handler function has void return type and accepts a signal number corresponding to the signal that needs to be handled.

In UNIX based systems, whenever ctrl+c is pressed, a signal SIGINT is sent to the process. The default action of this signal is to terminate the process.

The C library function **void (*signal(int sig, void (*func)(int)))(int)** sets a function to handle signal i.e. a signal handler with signal number **sig**.

This function returns the previous value of the signal handler, or SIG_ERR on error.

Compiling and running the function will cause program to go in infinite loop. To come out of the program we used CTRL + C keys.

If a process wishes to handle certain signals then in the code, the process has to register a signal handling function to the kernel. The signal handler function has void return type and accepts a signal number corresponding to the signal that needs to be handled.

In UNIX based systems, whenever ctrl+c is pressed, a signal SIGINT is sent to the process. The default action of this signal is to terminate the process.

The C library function void (*signal(int sig, void (*func)(int)))(int) sets a function to handle signal i.e. a signal handler with signal number sig.

This function returns the previous value of the signal handler, or SIG_ERR on error.

Compiling and running the function will cause program to go in infinite loop. To come out of the program we used CTRL + C keys.

Program:

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

void sighandler(int);

int main ()

{

signal(SIGINT, sighandler);

while(1)
```

```
 {
printf("Going to sleep for a second...\n");
sleep(1);
}
return(0);
}
void sighandler(int signum)
 {
printf("Caught signal %d, coming out...\n", signum);
exit(1);
}
```

Output:

```
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$ gcc
lab2.c
candywood@candywood-Lenovo-IdeaPad-S540-15IML-
D:~/Documents$ ./a.out
Going to sleep for a second...
Going to sleep for a second...
Going to sleep for a second...
Going to sleep for a second...
Going to sleep for a second...
Going to sleep for a second...
^CCaught signal 2, coming out...
candywood@candywood-Lenovo-IdeaPad-S540-15IML-D:~/Documents$
```