**Operating System & Virtualization (3141601)**

**Practical Lab INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Date** | **Sign** |
|  | **Practical 1** |  |  |
| 1.1 | Introduction & Study of various features of O.S. |  |  |
| 1.2 | Study of basic Linux commands. |  |  |
| 1.3 | Study of File, Directory & Process related system calls in Linux. |  |  |
|  | **Practical 2** |  |  |
| 2.1 | Study of Shell scripting & it’s features. |  |  |
| 2.2 | Write a shell script to find whether given no is odd or even. |  |  |
| 2.3 | Write a shell script to find sum & factorial of given number. |  |  |
|  | **Practical 3** |  |  |
| 3.1 | Write a shell script to find given number is prime or not? |  |  |
| 3.2 | Write a shell script to find given string is Palindrom or not? |  |  |
| 3.3 | Write a shell script to make a sum of digits of given number. |  |  |
|  | **Practical 4** |  |  |
| 4.1 | Write a C program to create exactly n processes in Linux at 2nd level of Tree. |  |  |
| 4.2 | Write a C program to distribute logic/work among n processes created by single program. |  |  |
|  | **Practical 5** |  |  |
| 5.1 | Write a C Program two create two threads for addtion and subtraction of two large size array. |  |  |
| 5.2 | Write a C program to implement producer/consumer problem using POSIX threads in Linux. |  |  |
|  | **Practical 6** |  |  |
| 6.1 | Write a C program to create and use shared memory. |  |  |
| 6.2 | Write a C program to create and use semaphores in Linux. |  |  |
|  | **Practical 7** |  |  |
| 7.1 | Write a shell script to find max, min , average of given set of numbers. |  |  |
| 7.2 | Write a shell script to read a list of numbers as input from command line and sort them. |  |  |
| 7.3 | Write a shell script to move upward n levels from current dir. |  |  |
|  | **Practical 8** |  |  |
| 8.1 | Write a shell script to mail all users who are currently logged in. |  |  |
| 8.2 | Write a shell script to display 5 largest files in the current directory. |  |  |
| 8.3 | Write a shell script to count how many files in a given directory. |  |  |
|  | **Practical 9** |  |  |
| 9.1 | Write a C program to implement “ls” command in Linux. |  |  |
| 9.2 | Write a C program which will run as background process and check when a particular user logged in the system and it should pop up message on the screen. |  |  |
|  | **Practical 10** |  |  |
| 10.1 | Write a shell script to translate all characters in File to upper case. |  |  |
| 10.2 | Write a shell script to change file extension (.txt) to (.doc) using for loop. |  |  |
| 10.3 | Write a shell script to compare two files to check whether they are same or not. |  |  |

**Practical Set – 1**

**Program 1.1: Introduction & Study of various features of O.S.**

**ANS:**

* Operating System is a program that controls the execution of application programs. It is an interface between applications and hardware.
* OS provides different types of views. For user, it is abstract view because it provide features which are implemented for users. OS is intermediary between user and computer system.
* The major design goals/ functions of an OS are:

1. Efficient use of a computer system
2. User Convenience
3. Ability to evolve

* An OS is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs for interfering with the paper operation of the system.
* Efficiency is the one of the parameter for use of computer system. OS consumes some resources during its on operation. For example, it uses CPU and Memory. CPU is busy with scheduling and memory is occupied by the instruction and required data.
* This is one type of overhead and because of this lesser resources are available to the user.
* An OS makes a computer more convenient to use. If the OS can’t allocates the free available resources to program or it over allocates the resources then efficiency is poor.

**Efficient use**

* For efficient use of resources, it must be monitored by operating system. Proper scheduling of resources is also required.
* Computer contains different type’s resources like CPU, memory and I/O device etc. Proper Monitoring is required on these resources to avoid the overhead. As per the resource, scheduling is required.
* Special attention to be given for CPU and memory. If memory is not free then user can’t load new program into memory. Then CPU will be busy with memory management.

**User convenience**

* User convenience is affected by computing environment of the computer system. The computer environment is comprised of computer system, its interfaces with other systems and nature of computation performed by its users.
* Computer architecture and use change the computing environment of the system. Following factors are considered while considering the user convenience:

1. Good service
2. Ease of use
3. New programming model
4. Evolution
5. User friendly OS

**Ability to evolve**

* An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfering with service.
* Task performed by operating systems:

1. Maintain the list of resources in the system
2. Maintain the list of authorized users
3. Initiate the execution of programs and process
4. Maintain resource usage list
5. Maintain the resource allocated list
6. Scheduling of resources (CPU, Secondary storage etc.)
7. Also maintain the protection information

**Program 1.2: Study of basic Linux commands.**

**Ans:**

1. cat [filename]

Display file’s contents to the standard output device (usually your monitor).

-Eg

* cat file1.txt – display the content of file on terminal.
* cat file1.txt file2.txt – concatenate both file and show content on terminal.

2. cd /directorypath

Change to directory.

-Eg

* cd dir1 – change t directory dir1
* cd ..- go one level up in directory hierarchy
* cd - take you to your home directory.

3. chmod [options] mode filename

Change a file’s permissions.

-Eg

* chmod 777 file1.txt – give read, write and execute permission to all three groups(Owner,group,other) –can be check using - l command

4. chown [options] filename

Change who owns a file.

-Eg

* chown remo file1.txt – makes remo as owner of file1.txt can be check using ‘stat file1.txt’

5. Clear

Clear a command line screen/window for a fresh start.

6. cp [options] source destination

Copy files and directories.

-Eg

* cp file1.txt fille2.txt – make copy of file1.txt and name it ass file
* 2.txt
* cp fille1.txt dir1 – make copy of file1.txt in directory dir1

7. date [options]

Display or set the system date and time.

8. df [options]

Display used and available disk space.

9. du [options]

Show how much space each file takes up

-Eg

* du – prints the each file’s size in the current directory.
* du file1.txt – prints only for file1.txt
* du –time - gives last modification time of each file.

10. file [options] filename

Determine what type of data is within a file.

11. find [pathname] [expression]

Search for files matching a provided pattern.

12. grep [options] pattern [filesname]

Search files or output for a particular pattern.

-Eg

* grep dog file1.txt – display lines containing dogs in file1.txt
* grep – i dog file1.txt –case insensitive search displays all lines containing dog/Dog/dOg etc.

13. kill [options] pid

Stop a process. If the process refuses to stop, use kill -9 pid.

-Eg

* kill pid – kill process having id pid.

14. less [options] [filename]

View the contents of a file one page at a time.

15. ln [options] source [destination]

Create a shortcut.

16. locate filename

Search a copy of your filesystem for the specified filename.

16. lpr [options]

Send a print job.

17. ls [options]

List directory contents.

-Eg

* dir1 –ls – show content of directory dir1.

18. man [command]

Display the help information for the specified command.-

-Eg

* man rm - shows all the information and options about command rm

19. mkdir [options] directory

Create a new directory.

-Eg

* mkdir dir1 – will create a new directory as dir1.

20. mv [options] source destination

Rename or move file(s) or directories.

-Eg

* mv file1.txt file2.txt – rename file1 with file2 and delete the file1.txt
* mv file1.txt dir1 – move file1.txt into directory dir1.

21. passwd [name [password]]

Change the password or allow (for the system administrator) to change any password.

-Eg

* passwd –s –a - allow user s to change any password as admin.

22. ps [options]

Display a snapshot of the currently running processes.

23. Pwd

Display the pathname for the current directory.

-Eg

* pwd - ./home/stud\_24/parth(whichi is current level in directory hierarchy)

24. rm [options] directory

Remove (delete) file(s) and/or directories.

-Eg

* rm file1.txt – remove file file1.txt
* rm –rf dir1 – remove all the content of directory dir1.

25. rmdir [options] directory

Delete empty directories.

-Eg

* rmdir dir1 – remove empty directories only, if directory has content it won’t work.

26. ssh [options] user@machine

Remotely log in to another Linux machine, over the network. Leave an ssh session by typing exit.

27. su [options] [user [arguments]]

Switch to another user account.

-Eg

* su remo – will ask the password for remo account and changes the user.
* su - will as for root password and change user as root.

28. tail [options] [filename]

Display the last *n* lines of a file (the default is 10).

29. tar [options] filename

Store and extract files from a tarfile (.tar) or tarball (.tar.gz or .tgz).

-Eg

* tar –zcvf dir1.tar.gz dir1 – to compress dir1
* tar –zxvf archive.tar.gz - to decompress archive.tar

30. Top

Displays the resources being used on your system. Press q to exit.

31. touch filename

Create an empty file with the specified name.

32. who [options]

Display who is logged on.

**Program 1.3: Study of File, Directory & Process related system calls in Linux.**

**Ans:**

**System calls are functions that a programmer can call to perform the services of the operating system. There are several online books that describe them at some length, for example Programming For C.**

**System calls can be roughly grouped into five major categories process control, file management, device management, information maintenance and communication.**

**We will reproduce Wikipedia classification with some modifications:**

1. **Process Control** 
   * **load**
   * **execute**
   * **fork**
   * **terminate process**
   * **get/set process attributes**
   * **allocate, free memory**
2. **File management.**
   * **create file, delete file**
   * **open, close, check status of file descriptors**
   * **read, write, reposition**
   * **get/set file attributes; get information about inode**
   * **Commit data to disk**
3. **Device Management.**
   * **request device, release device**
   * **read, write, reposition**
   * **get/set device attributes**
   * **logically attach or detach devices**
   * **execute device specific operation**
4. **Information Maintenance.**
   * **get/set time or date**
   * **get/set system data**
   * **get/set process, file, or device attributes**
5. **Communication.**
   * **create, delete communication connection**
   * **send, receive messages**
   * **transfer status information**
   * **attach or detach remote devices**

**Man pages should be used as a reference when you study Unix system calls.  The manual pages are divided into eight sections, with section 2 devoted to Unix system calls. They are organized as follows:**

**1. Commands This section provides information about user-level commands, such as ps and ls.**

**2. UNIX System Calls This section gives information about the library calls that interface with the UNIX operating system, such as open for opening a file, and exec for executing a program file. These are often accessed by C programmers.**

**3. Libraries This section contains the library routines that come with the system. An example library that comes with each system is the math library, containing such functions as fabs for absolute value. Like the system call section, this is relevant to programmers.**

**4. File Formats This section contains information on the file formats of system files, such as init, group, and passwd. This is useful for system administrators.**

**5. File Formats This section contains information on various system characteristics. For example, a manual page exists here to display the complete ASCII character set (ascii).**

**6. Games This section usually contains directions for games that came with the system.**

**7. Device Drivers This section contains information on UNIX device drivers, such scsi and floppy. These are usually pertinent to someone implementing a device driver, as well as the system administrator.**

**8. System Maintenance This section contains information on commands that are useful for the system administrator, such as how to format a dis.**

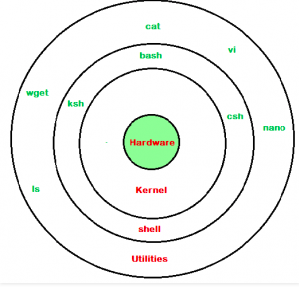
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | | |
| |  |  |  |  | | --- | --- | --- | --- | | **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** | |  |  |  |  | | | | |  |  |  |
|  | | | |  |  |  |

**Practical Set – 2**

**Program 2.1: Study of Shell scripting & it’s features.**

**A shell is special user program which provide an interface to user to use operating system services. It is a command language interpreter that execute commands read from input devices such as keyboards or from files.**

**As shell can also take commands as input from file we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called Shell Scripts or Shell Programs. It has .sh extension.**

****

A shell script comprises following elements –

● Shell Keywords – if, else, break etc.

● Shell commands – cd, ls, echo, pwd, touch etc.

● Functions

● Control flow – if...then...else, case and shell loops etc**.**

**Program 2.2: Write a shell script to find whether given no is odd or even.**

#!/bin/bash

echo "Enter a number to find whether it is odd or even : "

read n

if [ $((n%2)) -eq 0 ]

then

echo "The number is Even."

else

echo "The number is Odd."

fi

**Output**

A screenshot of a computer program

AI-generated content may be incorrect.

**Program 2.3: Write a shell script to find sum & factorial of given number.**

#!/bin/bash

echo “enter number to find sum and factorial”

read num

i=1

sum=0

facto=1

while [ $i -le $num ]

do

sum=`expr $sum + $i`

facto=`expr $facto \\* $i`

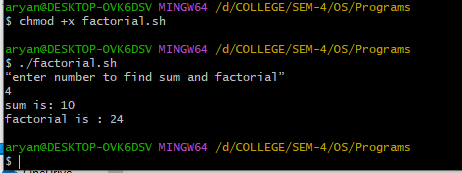
i=`expr $i + 1`

done

echo sum is: $sum

echo factorial is : $facto

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 3**

**3.1 Write a shell script to find given number is prime or not?**

#!/bin/bash

echo "Enter a number to check whether it is prime or not: "

read n

if [ $n -le 1 ]; then

echo "The number is not Prime."

exit

fi

i=2

flag=1

while [ $i -le $((n/2)) ]

do

if [ $((n % i)) -eq 0 ]

then

flag=0

break

fi

i=$((i + 1))

done

if [ $flag -eq 1 ]

then

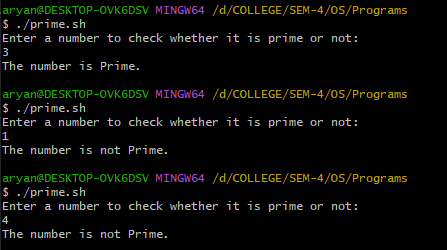
echo "The number is Prime."

else

echo "The number is not Prime."

fi

**Output:**

****

**3.2 Write a shell script to find given string is Palindrome or not?**

#!/bin/bash

echo "Enter a string to check whether it is palindrome or not: "

read str

# Reverse the string manually

len=${#str}

reverse=""

for (( i=$len-1; i>=0; i-- ))

do

reverse="$reverse${str:$i:1}"

done

echo "Reversed: $reverse"

if [ "$str" = "$reverse" ]

then

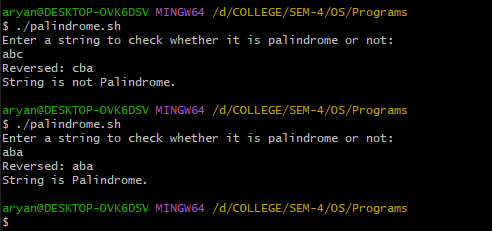
echo "String is Palindrome."

else

echo "String is not Palindrome."

fi

**Output**:



**3.3 Write a shell script to make a sum of digits of given number.**

#!/bin/bash

echo "Enter number to find sum of its digit: "

read num

sum=0

m=0

while [ $num -gt 0 ]

do

m=$((num%10))

sum=$((sum+m))

num=$((num/10))

done

echo $sum

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 4**

**4.1 Write a C program to create exactly n processes in Linux at 2nd level of Tree.**

#include <stdio.h>

int main()

{

int id;

id = process\_fork(3);

//printf("Hello I am %d\n", id);

if(id==0)

{

// parent process

printf("Hello I am parent %d\n", id);

}

else if(id==1)

{

// First Child

//printf("Hello I am child %d\n", id);

execl("/usr/bin/who","who", (char\*)0);

}

else if(id==2)

{

// Second Child

//printf("Hello I am child %d\n", id);

execl("/bin/ls","ls","-l" ,(char\*)0);

}

else if(id==3)

{

// Third Child

//printf("Hello I am child %d\n", id);

execl("/bin/pwd","pwd", (char\*)0);

}

}

int process\_fork(int n)

{

int i;

for(i=1;i<=n;i++)

{

if(fork()==0)

return i;

//else

//return 0;

}

return 0;

}

**Output:**

**4.2 Write a C program to distribute logic/work among n processes created by single program.**

#include<stdio.h>

#include<unistd.h>

main()

{

int i=0,k=0;

int pid[4],pidt;

float a=4,b=5;

for(i=0;i<4;i++)

{

k++;

pidt = fork();

if(pidt == 0)

break;

}

if(pidt == 0)

{

if(k==1)

{ sleep(20);

printf("Addition is :%f\n",a+b);

}

else if(k==2)

{ sleep(15);

printf("Multiplication is :%f\n",a\*b);

}

else if(k==3)

{ sleep(10);

printf("substraction is :%f\n",a-b);

}

else if(k==4)

{ sleep(9);

printf("Divison is :%f\n",a/b);

}

else

{ printf("Invalid k\n");

}

}

else

{

wait(0);

exit(1);

}

/\*

if(pidt == 0)

{

if(getpid()==pid[0])

printf("The addition is :%f\n",a+b);

else if(getpid==pid[1])

printf("The substraction is: %f\n",a-b);

else if(getpid == pid[2])

printf("The multiplication is %f\n:",a\*b);

else

printf("The divison is %f",a/b);

}

else

{

wait(0);

exit(1);

}

\*/

}

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 5**

**5.1 Write a C Program two create two threads for addtion and subtraction of two large size array.**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define SIZE 100000

int A[SIZE], B[SIZE], sum[SIZE], diff[SIZE];

void\* add(void\* arg) {

for (int i = 0; i < SIZE; i++) {

sum[i] = A[i] + B[i];

}

pthread\_exit(NULL);

}

void\* subtract(void\* arg) {

for (int i = 0; i < SIZE; i++) {

diff[i] = A[i] - B[i];

}

pthread\_exit(NULL);

}

int main() {

pthread\_t thread1, thread2;

// Initialize arrays

for (int i = 0; i < SIZE; i++) {

A[i] = i;

B[i] = SIZE - i;

}

pthread\_create(&thread1, NULL, add, NULL);

pthread\_create(&thread2, NULL, subtract, NULL);

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

// Print first 5 results for verification

printf("Index\tA\tB\tSum\tDiff\n");

for (int i = 0; i < 5; i++) {

printf("%d\t%d\t%d\t%d\t%d\n", i, A[i], B[i], sum[i], diff[i]);

}

return 0;

}

**Output:**

**5.2 Write a C program to implement producer/consumer problem using POSIX threads in Linux.**

#include <stdio.h>

#include <pthread.h>

#define MAX 10000000

pthread\_mutex\_t the\_mutex;

pthread\_cond\_t condc, condp;

int buffer[MAX] , count=0;

long k=0,j=0;

void \*producer(void \*ptr)

{

while(1)

{

pthread\_mutex\_lock(&the\_mutex);

if(count == MAX)

{

printf("Buffer FULL on %d\n",count);

pthread\_cond\_wait(&condp, &the\_mutex);

}

buffer[k]=k; k++; if(k==MAX+1) k=0;

count++;

if(count == 1)

pthread\_cond\_signal(&condc);

pthread\_mutex\_unlock(&the\_mutex);

}

pthread\_exit(0);

}

void \*consumer(void \*ptr)

{

int p;

while(1)

{

pthread\_mutex\_lock(&the\_mutex);

if(count == 0)

{

printf("BUffer is Empty on %d \n",count);

pthread\_cond\_wait(&condc,&the\_mutex);

}

p = buffer[j]; j++; if(j==MAX+1) j=0;

count--;

if(count == MAX-1)

pthread\_cond\_signal(&condp);

pthread\_mutex\_unlock(&the\_mutex);

}

pthread\_exit(0);

}

int main(int argc, char \*\*argv)

{

pthread\_t pro, con;

pthread\_mutex\_init(&the\_mutex, 0);

pthread\_cond\_init(&condc, 0);

pthread\_cond\_init(&condp, 0);

// pthread\_create(&con,0,consumer,0);

pthread\_create(&pro,0,producer,0);

pthread\_create(&con,0,consumer,0);

pthread\_join (pro,0);

pthread\_join (con,0);

printf("Count is :%d\n", count);

pthread\_cond\_destroy(&condc);

pthread\_cond\_destroy(&condp);

pthread\_mutex\_destroy(&the\_mutex);

}

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 6**

**6.1 Write a C program to create and use shared memory.**

#include <stdio.h>

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/ipc.h>

#include <unistd.h>

main()

{

int sid,sid1;

int \*sidp;

char \*msg,hel1[5]="hello";

sid = shmget(0x2,10,IPC\_CREAT | 0666);

// sid1 = shmget(0x3,5,IPC\_CREAT | 0666);

// printf("%d",sid);

msg = shmat(sid,0,0);

sidp = shmat(sid,0,0);

// \*sidp = 99;

strcpy(msg,hel1);

strcpy(hel1,"NOTOK");

\*sidp = 99;

printf("%s\n",msg);

printf("%s\n",hel1);

printf("%d\n",\*sidp);

}

**Output:**

**6.2 Write a C program to create and use semaphores in Linux.**

#include <stdio.h>

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/ipc.h>

#include <unistd.h>

main()

{

int sid,j;

double i;

int \*sidp;

int semaid;

sid = shmget(0x2,10,IPC\_CREAT | 0666);

sidp = shmat(sid,0,0);

\*sidp = 100;

semaid= semget(0x5,1,IPC\_CREAT | 0666);

semctl(semaid,0,SETVAL,0);

for(j=0;j<15;j++)

{

if((semctl(semaid,0,GETVAL,0)) == 0)

{

semctl(semaid,0,SETVAL,1);

for(i=0;i<10;i++)

\*sidp = \*sidp + 1;

printf("%d\n",\*sidp);

semctl(semaid,0,SETVAL,0);

}

for(i=0;i<1000000000/3;i++) { }

}

}

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 7**

**7.1 Write a shell script to find max, min , average of given set of numbers.**

#!/bin/bash

echo "Enter Size of array: "

read n

i=0

total=0

for((i=0;i<n;i++))

do

echo "Enter value s[$i] : "

read s[$i]

total=$((total+s[i]))

done

avg=$((total/n))

max=${s[0]}

min=${s[0]}

for((i=1;i<n;i++))

do

if [ ${s[$i]} -gt $max ]

then

max=${s[$i]}

elif [ ${s[$i]} -lt $min ]

then

min=${s[$i]}

fi

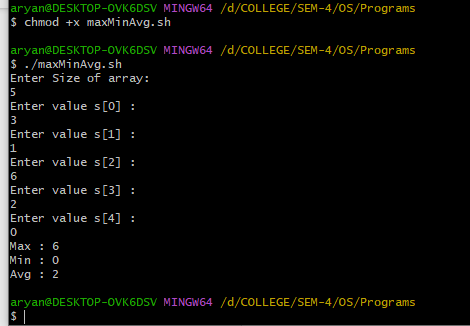
done

echo "Max : $max"

echo "Min : $min"

echo "Avg : $avg"

**Output:**

****

**7.2 Write a shell script to read a list of numbers as input from command line and sort them.**

#!/bin/bash

for i in "$@"

do

echo $i>>srt.txt

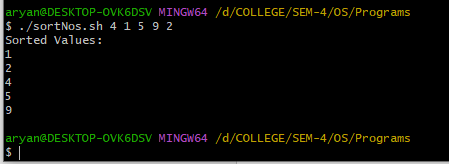
done

echo "Sorted Values:"

sort -n srt.txt

rm srt.txt

**Output:**

****

**7.3 Write a shell script to move upward n level from current dir.**

#!/bin/bash

echo "Enter number of levels to move upwards: "

read n

echo "Origin : $(pwd)"

while [ $n -gt 0 ]

do

cd ..

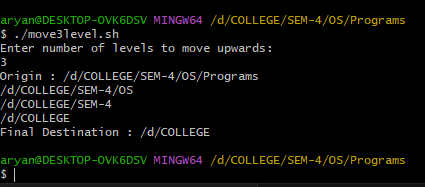
echo $(pwd)

n=$((n-1))

done

echo "Final Destination : $(pwd)"

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 8**

**8.1 Write a shell script to mail all users who are currently logged in.**

#!/bin/bash

echo "Warning : Please install mailutils first if you're using mail command first time."

email=`users`

echo "Enter body of message : "

read body

echo $body>body.txt

mail -s "Programming Exercise" $email < body.txt

echo "email sent to $email."

sudo mail -f /var/spool/mail/user

**Output:**

A close up of a text

AI-generated content may be incorrect.

**8.2 Write a shell script to display 5 largest files in the current directory.**

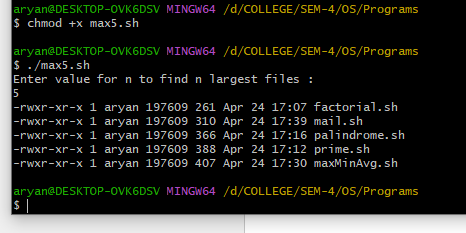
#!/bin/bash

echo "Enter value for n to find n largest files : "

read n

ls -lS -r | grep '^-' | tail -$n

**Output:**

****

**8.3 Write a shell script to count how many files in a given directory.**

#!/bin/bash

echo "No. of Total files : "

ls -l | wc -l

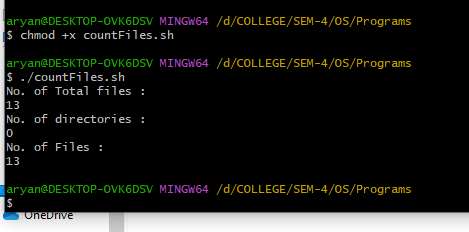
echo "No. of directories : "

ls -l | grep '^d' |wc -l

echo "No. of Files : "

ls -l | grep -v '^d' | wc -l

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 9**

**9.1 Write a C program to implement “ls” command in Linux.**

#include <stdio.h>

#include <sys/types.h>

#include <dirent.h>

#include <unistd.h>

// #include <linux/unistd.h>

// #include <linux/dirent.h>

main(int argc,char \*argv[])

{

DIR \*dirname;

struct dirent \*dir;

dirname = opendir(argv[1]);

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

{

if(strcmp(dir->d\_name,".") == 0 || strcmp(dir->d\_name,"..")==0)

continue;

printf("%-10s",dir->d\_name);

}

printf("\n");

}

**9.2 Write a C program which will run as background process and check when a particular user logged in the system and it should pop up message on the screen.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <utmp.h>

#define UTMP "/var/run/utmp"

int main (int argc, char \*argv[])

{

FILE \*fp;

struct utmp u;

for(;;)

{

fp = fopen(UTMP,"r");

while(!feof(fp))

{

fread(&u, sizeof(u), 1,fp);

if(u.ut\_name == NULL)

continue;

if(strcmp(argv[1], u.ut\_name)==0)

{

printf("\n \7 \7 \7 \7 %s has login\n", argv[1]);

exit(0);

}

}

fclose(fp);

sleep(5);

}

}

Output:

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |

**Practical Set – 10**

**10.1 Write a shell script to translate all characters in File to upper case.**

echo "Enter filename to translate all charaters to Uppercase : "

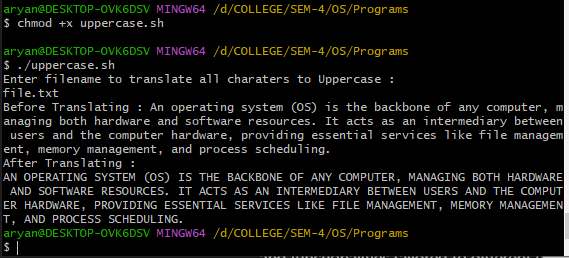
read file

echo "Before Translating : $(cat $file)"

echo "After Translating : "

cat $file | tr '[a-z]' '[A-Z]'

**Output:**

****

**10.2 Write a shell script to change file extension (.txt) to (.doc) using for loop.**

#!/bin/bash

echo "Converting all '.txt' files to '.doc'."

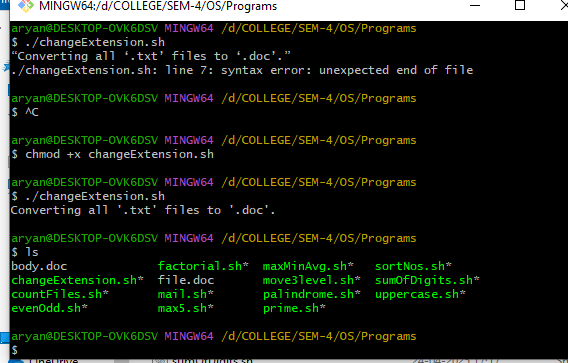
for fname in \*.txt

do

mv "$fname" "${fname%.txt}.doc"

done

**Output:**

****

**10.3 Write a shell script to compare two files to check whether they are same or not**.

#!/bin/bash

echo "Disclaimer : Enter File names with extension."

echo "Enter name of first file : "

read file1

echo "Enter name of second file : "

read file2

# Normalize files to compare line by line

if diff -q "$file1" "$file2" > /dev/null; then

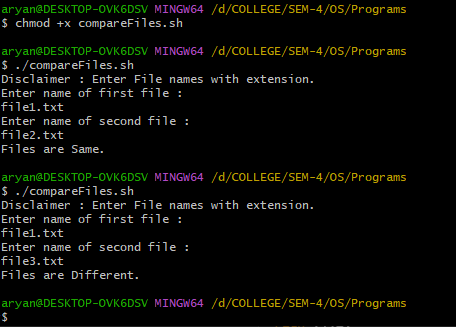
echo "Files are Same."

else

echo "Files are Different."

fi

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Understanding of Problem(2)** | **Implementation of Problem(6)** | **Alternative/Improved solution of Lab Experiment(2)** | **Total (10)** |
|  |  |  |  |