Operating System Lab Manuals

Do's

1. Maintain regularity in the Lab work
2. Update the Lab work in Practical Copy
3. Get it Signed by Instructor
4. Learn and Practice
5. Update your knowledge

Don't Do's

1. Absentee from Lab Work
2. Shoes/sleeper in the lab

Subject: Operating Systems Lab L T P: 0-0-3

Subject Code: AMC15206

Objective: 1) To inculcate the Shell programming skill and its application
2) To understand the concept of processes, threads, Scheduling and Deadlocks via programming

Sl. No.	Name of Experiment/Lab	Learning Outcomes
1.	Introduction to Shell Programming: Syntax, various commands etc.	Students will Come to Know about the basics of shell Programming
2.	Algorithm and coding for Shell Programming	They will Come to Know about the logic and coding skill of Shell programming
3.	Execution of Shell Programming	They will Come to Know about the execution of Shell programming
4.	Shell Programming continued	Finally, They will Come to Know about Shell programming skill
5.	Programming based on Processes	They will Come to Know about the programming for process creation
6.	Programming based on Threads	They will Come to Know about the programming for Threads creation
7.	CPU Scheduling algorithms-FCFS	They will Come to Know about the programming for FCFS
8.	CPU Scheduling algorithms-SJF	They will Come to Know about the programming for SJF
9.	CPU Scheduling algorithms-RR	They will Come to Know about the programming for RR
10.	CPU Scheduling algorithms- Priority	They will Come to Know about the programming for Priority
11.	Programming based on Deadlock	They will Come to Know about the programming for Deadlock
12.	Lab Exam	

The computer program that allocates the system resources and coordinates all the details of the computer's internal is called the operating system or kernel. Users communicate with the kernel through a program known as the shell. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel. Otherwise, we can also say that a shell is simply a program that is used to start other programs. All operating systems like UNIX, Linux, Windows etc., have shells. UNIX is multiuser system and multitasking. UNIX was originally developed in 1969 by a group of AT&T employees at Bell labs, including Ken Thompson, Dennis Ritchie, Douglas Mcllory, and Joe Osssanna. There are various UNIX variants available in the market. Solaris UNIX, AIX, HP UNIX and BSD are few examples. Linux is also a flavour of UNIX which is freely available. There are a variety of UNIX shells to choose from. The original and most widely supported shell is called the Bourne shell (after S. R. Bourne). Its program file name is sh. Overall, approx 95% of the world's shell scripts are created for use with the Bourne shell. There are a number of Bourne shell derivatives, each offering a variety of extra features including Korn shell (ksh) (after Devid Korn), Bourne again shell (bash), c shell (csh). All data in UNIX is organized into files and all the files are organized into directories. These directories are organized into a tree-like structure called the file system. We can use "ls" command to list all the files or directories available in a directory. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options. A few commands are given in the following table:

Some Unix Commands	**Description**
ls	Lists all files or directories in a directory
ls -l	Lists all files in long format, one file per line
ls -a	Lists all files including hidden files
cd	Change directory
mkdir	Make directory
vi	Opens vi text editor
chmod	Changing permissions
find	Find files
cat	Displays file contents
pwd	To know about present working directory
mv	To rename the existing file
cp	To copy one or more files
man	Displays the manual pages for a chosen Unix command
rm	Removes files or directories
date	Displays current date and time
clear	Clear the screen
passwd	Change user password
who	Displays data about all the user have logged into the system currently

SHELL DECISION MAKING

The if…fi statement

```
if [expression]
then
  Statement(s) to be executed if expression is true
fi
```

The if…else…fi statement

```
if [expression]
then
  Statement(s) to be executed if expression is true
else
   Statement(s) to be executed if expression is not true
fi
```

The if…elif…fi statement

```
if [expression 1]
then
   Statement(s) to be executed if expression 1 is true
elif [expression 2]
then
    Statement(s) to be executed if expression 2 is true
elif [expression 3]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
fi
```

SHELL LOOP TYPES

The while Loop

```
while command
do
    Staement(s) to be executed if command is true
done
```

The for loop

```
for var in word1 word2 … wordN
do
   Statement(s) to be executed for every word
done
```

Exercise: Enter these commands at the UNIX prompt, and try to interpret the output:

i) echo hello world
ii) passwd
iii) date

iv) hostname
v) uname -a
vi) uptime
vii) who am i
viii) who
ix) id
x) last
xi) finger
xii) top (you may need to press q to quit)
xiii) echo $SHELL
xiv) man "automatic door"
xv) man ls (you may need to press q to quit)
xvi) man who (you may need to press q to quit)
xvii) lost
xviii) clear
xix) cal 2000
xx) bc -l (type quit or press Ctrl-d to quit)
xxi) echo 5+4 | bc -l
xxii) history

ALGORITHM FOR SHELL PROGRAMMING

1. Write a shell script program to read two numbers and perform basic arithmetic operations (+ , - , * , / , %)

Algorithm:
Step 1: Start
Step 2: Read two integers a, b
Step 3: Calculate Sum= a + b
Sub= a – b
Product= a * b
Div=a / b
Rem=a % b
Step 4: Display Sum, Sub, Product, Div and Rem
Step 5: Stop

2. Write a shell script to read three integer numbers and print the largest among three numbers.

Algorithm:
Step 1: Start
Step 2: Declare variables a, b and c.
Step 3: Read variables a, b and c.
Step 4: if a>b
if a>c
Display a is the largest number.
else
Display c is the largest number.
else
if b>c

Display b is the largest number.
else
Display c is the greatest number.
Step 5: Stop

3. Write a shell script program to read a character from keyboard and check whether it is vowel or not.

Algorithm:
Step1: Start
Step2: Declare variable ch.
Step3: Read the value of ch.
Step4: if (ch=='A' || ch=='a' || ch=='E' || ch=='e'|| ch=='I' || ch=='i'|| ch=='O' || ch=='o' || ch=='U' || ch == 'u') then
Display "Entered character is Vowel"
goto Step 6
else
Step5: Display "Entered character is not Vowel"
goto Step 6
Step 6: Stop

4. Write a shell script to print out the Fibonacci series up to a limit.

Algorithm:
Step 1: Start
Step 2: Declare variables n, a ← 0, b ← 1, c, i
Step 3: Read values of n
Step 4: Display a, b
Step 5: Assign i←2
Step 6: if i < n then goto step 7 otherwise goto step10
Step 7: calculate c ← a+b,
i ← i+1
a ← b, b ← c
Display the value of c
goto step 6
Step 10: Stop

5. To write a shell script to check whether the given number is prime or not.

Algorithm:
Step 1: Start
Step 2: Read an integer n
Step 3: Assign i=2, j=0
Step 4: Is i < n then r =n % i. otherwise goto step 8
Step 5: Is r=0 then increment i and j value by i. otherwise go to step 6
Step 6: Increment i value by one
Step 7: Is j=0 then print number is prime and goto step 10
Step 8: Is j != 0 then print number is not a prime number
Step 9: Stop

6. To write a shell script to find the Armstrong numbers between 1 to N.

Algorithm:
Step 1: Start
Step 2: When i equal to 0 and less than or equal to N, calculate increment value of i.
Step 3: Assign value of i to temp and n.
Step 4: Assign value of ams equal to zero.
Step 5: When n not equal to zero calculate
rem←n%10;
ams=ams+rem*rem*rem
n←n/10
Step 6: If temp equal to ams then print the value of ams.
Step 7: Thus for each value of i, values of ams is printed.
Step 8: Stop the program.

7. Write a shell script to perform Conversion of temperature in Celsius to Fahrenheit and Fahrenheit to Celsius.

Algorithm:
Step 1: Start
Step 2: Input the choice as 1 or 2
Step 3: Is choice is 1 then goto step 4 otherwise goto step 7
Step 4: Input temperature in Celsius
Step 5: Calculate Fahrenheit F = ((9/5)*c) +32
Step 6: Print Fahrenheit F and goto step 10
Step 7: Input temperature in Fahrenheit
Step 8: Calculate Celsius C = ((5/9)*(f-32))
Step 9: Print Celsius C
Step 10: Stop

8. Write a shell script to read an integer find out the reverse of the integer using function and check whether integer is palindrome or not.

Algorithm:
Step 1: Start
Step 2: read n
Step 3: copy n into m for later use. Also, initialize rn;
Step 5: while n is not zero
1. r = n % 10
2. n = n/10
3. rn = rn*10 + r;
Step 6: if m equal rn then the number is palindrome.
Step 7: Else Print number is not palindrome
Step 8: Stop

9. Write a shell script to read an integer find out the factorial of the integer.

Algorithm:
Step1: Start

Step2: Read a number ‘n’ and fact=1
Step3: if n==1 then
Return (1)
Step4: else
For i=0 to i<n
Factorial=fact*fact(n-1)
Return(fact)
Step4: Stop

10. Write a shell script program to read an array of ‘n’ integers and perform linear search operation.

Algorithm:
Step 1: Start
Step 2: Read the array A of ‘n’ elements, f=0
Step 3: Read the element ‘x’ to be searched in A
Step 4: Set i to 0
Step 5: if i > n then go to step 10
Step 6: if A[i] = x then f=1 and go to step 9
Step 7: Set i to i + 1
Step 8: Go to Step 5
Step 9: Print Element x Found at index i+1 and go to step 11
Step 10: if f=0 then Print element not found
Step 11: Stop

11. Write a shell script program to read an array of ‘n’ integers and sort number in ascending order using bubble sort technique.

Algorithm:
Step1: Start
Step2: Read the number of array elements
Step3: for i = 0 to n-1
Read array[i]
Step4: for i = 0 to n-1
for j = 0 to n-i-1
if (array[i]>array[j+1]) then
Temp=array[j]
Array[j]=array[j+1]
Array[j+1]=temp
Step7: Display array elements
Step8: Stop

12. Write a shell script program to read an array of ‘n’ integers and perform binary search.

Algorithm:
Step 1: Start
Step 2: Read the array a of n elements, f=0
Step 3: Sort using any algorithm
Step 4: Read the element to be searched in x

Step 5: Set L=0 the lower limit and u=n-1 the upper limit
Step 6: Repeat the steps 7, 8, 9, 10 until u>=L
Step 7: mid = (L+u)/2
Step 8: when a[mid]==x f=1 print the search is successful, display the position goto step 12
Step 9: when a[mid]<x L=mid+1
Step 10: when a[mid]>x u=mid-1
Step 11: if f==0 print search is unsuccessful
Step 12: Stop

PROCESS CREATION

AIM: To write a C program to perform process creation using fork() system call.

Algorithm:
Step1: Start program.
Step 2: Assign fork() system call to pid.
Step 3: if pid is equal to -1, child process not created.
Step 4: if pid is equal to 0, child process will be created.
Step 5: Print the id of parent process and child process.
Step 6: Create another one child process in same loop.
Step 7: Print id of parent process and the child process.
Step 8: Print grand parent id.
Step 9: Stop the program.

Note:- The fork() is a system which will create a new child process. The child process created is an identical process to the parent except that has a new system process ID. The process is copied in memory from its parent process, then a new process structure is assigned by the kernel. The return value of the function is which discriminates the two threads of execution. A 0 is returned by the fork function in the child's process, while the PID of the child process is returned in the parent's process.
In brief, fork() creates a child process that differs from the parent process only in its PID and PPID (the Parent PID, the creator of the process), and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited. Under Linux, fork() is implemented using copy-on-write pages, so the only penalty that it incurs is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.

THREADING

AIM: Thread creation:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_message_function( void *ptr );

main()
{
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
```

```
    char *message2 = "Thread 2";
    int  iret1, iret2;

   /* Create independent threads each of which will execute function */

    iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
    iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

    /* Wait till threads are complete before main continues. Unless we  */
    /* wait we run the risk of executing an exit which will terminate   */
    /* the process and all threads before the threads have completed.   */

    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

    printf("Thread 1 returns: %d\n",iret1);
    printf("Thread 2 returns: %d\n",iret2);
    exit(0);
}

void *print_message_function( void *ptr )
{
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
```

SCHEDULING ALGORITHMS

1. First Come First Serve Scheduling (FCFS Scheduling)

i) Jobs are executed on first come and first serve basis
ii) It is a non pre-emptive scheduling algorithm
iii) It is easy to understand and implement
iv) Its implementation is based on first in first out (FIFO) queue
v) It is poor in performance as average waiting time is high

AIM: To write a program to implement CPU & scheduling algorithm for first come first serve scheduling.

Algorithm:
1. Start the program.
2. Get the number of processes and their burst time.
3. Initialize the waiting time for process 1 and 0.
4. Process for (i=2; i<=n; i++), wt.p[i] = p[i-1]+bt.p[i-1].
5. The waiting time of all the processes is summed then average value time is calculated.
6. The waiting time of each process and average times are displayed
7. Stop the program

2. Shortest Job First Scheduling (SJF Scheduling)

i) It is a non pre-emptive scheduling algorithm
ii) It is best approach to minimize waiting time
iii) It is easy to implement in batch systems where required CPU time is known in advance
iv) Impossible to implement in interactive systems where required CPU time is not known
v) The processor should know in advance how much time process will take

AIM: To write a program to implement CPU and scheduling algorithm for shortest job first scheduling.

Algorithm:
1. Start the program. Get the number of processes and their burst time.
2. Initialize the waiting time for process 1 as 0.
3. The processes are stored according to their burst time.
4. The waiting time for the processes are calculated as follows:
   for(i=2; i<=n; i++), wt.p[i] = p[i=1]+bt.p[i-1].
5. The waiting time of all the processes summed and then the average time is calculate
6. The waiting time of each processes and average time are displayed.
7. Stop the program.

3. Priority Scheduling

i) SJF scheduling is special case of priority scheduling
ii) Priority is associated with each process
iii) CPU is allotted to the process with the highest priority
iv) For the case of equal priority, processes are scheduled on the basis of FCFS
v) It is a non pre-emptive scheduling
vi) Priority can be decided based on memory or time requirements or any other resource requirements

AIM: To write a program to perform priority scheduling.

Algorithm:
1. Start the program.
2. Read burst time, waiting time, turn the around time and priority.
3. Initialize the waiting time for process 1 and 0.
4. Based up on the priority process are arranged
5. The waiting time of all the processes is summed and then the average waiting time
6. The waiting time of each process and average waiting time are displayed based on the priority.
7. Stop the program.

4. Round Robin Scheduling

i) It is pre-emptive process scheduling algorithm
ii) Each process is provided a fix time to execute, it is called a quantum
iii) Once a process is executed for a given time period, it will be pre-empted at that given time and other process will execute for a given time period

AIM: To write a program to implement CPU and scheduling for Round Robin Scheduling.

Algorithm:

1. Get the number of process and their burst time.
2. Initialize the array for Round Robin circular queue as '0'.
3. The burst time of each process is divided and the quotients are stored on the round Robin array.
4. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling.
5. The waiting time for each process and average times are displayed.
6. Stop the program.

SIMULATE ALGORITHM FOR DEADLOCK AVOIDANCE

The Banker's algorithm is a resource allocation & deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of pre-determined maximum possible amounts of all resources, and then makes a "safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

The basic data structures used to implement this algorithm are given below

Let n be the total number of processes and m be the total number of resource types in the system.

Available: A vector of length m. It shows number of available resources of each type. If Available[i] = k, then k instances of resource Ri are available.

Max: An n×m matrix that contain maximum demand of each process. If Max[i,j] = k, then process Pi can request maximum k instances of resource type Rj.

Allocation: An n×m matrix that contain number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then Pi is currently allocated k instances of resource type Rj.

Need: An n×m matrix that shows the remaining resource need of each process. If Need[i,j] = k, then process Pi may need k more instances of resource type Rj to complete the task.

Working Rule

Banker's Algorithm is executed whenever any process puts forward the request for allocating the resources. It involves the following steps.

Step-01:

Banker's Algorithm checks whether the request made by the process is valid or not.

If the request is invalid, it aborts the request.
If the request is valid, it follows step-02.

Step-02:

Banker's Algorithm checks whether the request made by the process is valid or not.
If the request is invalid, it aborts the request.
If the request is valid, it follows step-03.

Step-03:

Banker's Algorithm makes an assumption that the requested resources have been allocated to the process.
Then, it modifies its data structures accordingly and moves from one state to the other state.
Now, Banker's Algorithm follows the safety algorithm to check whether the resulting state it has entered in is a safe state or not.
If it is a safe state, then it allocates the requested resources to the process in actual.
If it is an unsafe state, then it rollbacks to its previous state and asks the process to wait longer.