Chhatrapati Shahu Maharaj Shikshan Sanstha's
CHH. SHAHU COLLEGE OF ENGINEERING
Kanchanwadi, Paithan Road, Aurangabad.

Date :

Experiment no. 1.

Title:- Hands on UNIX Commands

Objective:-

- To understand basic utilities of unix
- To compare basic unix shell with popular shell and to learn the basic components in constructing shell script.

Analyzing the problem:-

- Start the Linux and enter the user name and password.
- Now write start x and after that open the terminal.
- At the terminal try the different commands and see the output.

Designing the solution:-

- At the terminal first perform the command without and with the different options available for it.

The exercises in this ~~job~~ lab cover the usage of some of the most basic system utilities that users and administrators alike need to be familiar with. Most of the commands are used in navagating and manipulating the file system. The file system is made up of files and directories.

Theory / Description :-

1] pwd COMMAND :-
pwd - print working Directory. pwd command prints the full file name of the current working directory.
SYNTAX :-
pwd [options]

2] cd COMMAND :-
cd command is used to change the directory.
SYNTAX :-
cd [directory /~/ ./ /../ /-]

3] ls COMMAND :-
ls COMMAND lists the files and directories under current working directory.
SYNTAX :-
ls [OPTIONS] ... [FILE]
OPTIONS :-

-l Lists all the files, directories and their mode, Number of links, owner of the file, file size, modified date and time and filename.
-t Lists all in order of last modification time.
-a Lists all entries including hidden files.
-d Lists directory files instead of contents.

-p puts slash at the end of each directories.
-u Lists in order of last access time.
-i Display inode information.

4] rm COMMAND:-
rm linux command is used to remove/delete the file from the directory.
SYNTAX:-
rm [options] [file] | directory]
OPTIONS:-

-f Remove all files in a directory without prompting the user.
-i Interactive with this option rm prompts for confirmation before removing any files.

5] cat COMMAND:-
cat command in linux concatenates files and print it on the standard output.
SYNTAX:-
cat [OPTIONS] [FILE]...

OPTIONS:-

-A Show all
-b omits line numbers for blank space in the output.

-E Displays a $ (dollar sign) at the end of each line.
-n Line numbers for all the output lines.

6] cmp COMMAND:-
cmp linux command compares two files and tells you which line numbers are different.
SYNTAX:-
cmp [OPTIONS...] file1 file2
OPTIONS:-

-e output differing bytes as characters.
-l print the byte number (decimal) and the differing byte values (octal) for each difference.
-s prints nothing for differing files, return exist status only.

7] cp COMMAND:-
cp command copy files from one location to another. If the destination is an existing file, then file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).
SYNTAX:-
cp [OPTIONS] SOURCE DEST.

8] echo COMMAND:-

echo command prints the given input string to standard output.

SYNTAX:-

echo [options] [string]

9] mkdir COMMAND:-

mkdir command is used to create one or more directories.

SYNTAX:-

mkdir [OPTIONS] directories

OPTIONS:-

-m set the access mode for the new directories
-p create intervening parent directories if they don't exist.
-v print help message for each directory created.

10] paste COMMAND:-

paste command is used to paste the content from one file to another file. It is also used to set column format for each line.

SYNTAX:-

paste [OPTIONS]

OPTIONS:-

-s paste one file at a time instead of in parallel.
-d Reuse characters from LIST instead of TABS.

11] rmdir COMMAND:-

rmdir command is used to delete / remove a directory and its subdirectories.

SYNTAX:-

rmdir [options...] Directory

OPTIONS:-

-p Allows users to remove the directory dir name and its parent directories which become empty.

12] head COMMAND:-

It is the complementary of Tail command. The head command, as the name implies, print the top N numbers of ~~data~~ dates of the given input. By default it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceeds by its file name.

SYNTAX:-

head [OPTION]... [FILE]...

13] tail COMMAND:-

It is complementary of head command. The tail command, as the name implies, print the last N Numbers of data of the given input. By default it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

SYNTAX:-

tail [OPTION]... [FILE]...

14] date command :-

date command is used to display the system date and time. date command is also used to set date and time of the system. By default the date command displays the date in ~~which~~ the time zone on which Unix / linux operating system is configured. You must be the super-user (root) to change the date and time.

SYNTAX:-

date [OPTION] ... [+FORMAT]

Conclusion :- Hence we have studied Hands on Unix commands.

Experiment no. 2.

Title:- Basics of Shell Programming.

Objective :-

- To understand shell decision-making in UNIX.
- To understand shell loops controls in UNIX.

Theory / Description :-

Shell Descision Making

Unix shell ~~sp~~ supports conditional statements which are used to perform different actions based on different conditions. We will now understand two decision-making statements here-

- The if... else statement
- The case... esac statement.

The if... else statements.

If else statements are useful decision-making statements which can be used to select an option from a given set of options.

UNIX shell supports following forms of if... else statements-

- if... fi statement.
- if... else... fi statement.
- if... elif... else... fi statement.

Most of the if statements checks relations using relational operators discussed in the previous chapter.

The case... esac Statement

You can use multiple if... elif statements to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Unix shell supports case... esac statement which handles exactly this situation, and it does so more efficiently than repeated if... elif statements.

The case... esac statements in the unix shell is very similar to the switch... case statements we have in other programming languages like C or C++ and PERL, etc.

Shell Loop types.

A loop is a powerful programming tool that enables you to execute a set of commands repeatedly. In this chapter, we will examine the following type of loops available to shell programmers.

- The while loop.
- The for loop.
- The until loop.
- The select loop.

You will use different loops based on the situation for example, the while loop executes the given commands until the given condition remains true; the until loop executes until a given condition becomes true.

Once you have good programming practice you will gain the expertise and thereby start using appropriate loops based on the situation. Here, while and for loops are available in most of the other programming languages like C, C++ and PERL etc.

Nesting loops

All the loops support nesting concept which mean you can put one loop inside another similar one or different loops. This nesting can go upto unlimited number of times based on your requirement.

Here is an example of nesting while loop. The other loops can be nested based on the programming requirement in a similar way-

Nesting while loops

It is possible to use a while loop as part of the body of another loop.

Syntax

```
while command 1; # this is loop 1, the outer loop.

do
   Statement(s) to be executed if command 1 is true

   while command 2; # this is loop 2, the inner loop

   do
      Statement(s) to be executed if command 2 is true done..

   Statement(s) to be executed if command 1 is true done.
```

Shell loop Control.

we will learn following two statements that are used to control shell loops-

- The break statement
- The continue statement.

The infinite loop.

All the loops have a limited life and they come out once the condition is false or true depending on the loop.

A loop may continue forever if the required condition is not met. A loop that executes forever without terminating executes for an infinite number of times for this reason, such loops are called infinite loops.

Here Example

Here is a simple example that uses the while loop to display the numbers zero to nine.—

```
#!/bin/sh

a=10

until [$a -1+10]

do
   echo $a
a= 'expr $a+1
done.
```

This loop continues forever because a is always greater than or equal to 10 and it is never less than 10.

The break statement

The break statement is used to terminate the execution of the entire loop, after completing the excution of all of the lines of code up to break statement. It then steps down to the code following the end of the loop.

Syntax

The following break statement is used to come out of a loop-

break

The break command can also be used to exit from a nested loop using this format-

break n

Here n specifies the n^{th} enclosing loop to the exit from.

Example

Here is a simple example which shows that loop terminates as soon as a becomes 5 -

```
#!/bin/sh

a=0

while [$a -lt 10]

do

  echo $a

if [$a -eq 5]
```

```
then
    break
fi
 a= 'expt $a +1'
done.
```

upon execution, you will recieve the following result-

```
0
1
2
3
$4
5
```

Here is a simple example of nested for loop. This script breaks out of both loops if var1 equals 2 and var2 equals 0-

```
#!/bin/sh

for var1 in 123

do
  for var2 in 05
do
   if [ $var1 -eq2 - a $var2 -eq 0]
```

```
then
    break 2

else
    echo "$var1 $var2"

fi
 done

done.
```

upon execution, you will receive the following result. In the inner loop, you have a break command with the argument 2. This indicates that if a condition is met you should break.. Out of outer loop and ultimately from the inner loop as well

```
1  0
1  5
```

The Continue statement

The Continue statement is similar to the break command, except that it causes the current iteration of the loop to exit, rather than the entire loop.

This statement is useful when an error has occured but you want to try to execute the next iteration of the loop.

Syntax

Continue

Like with the break statement, an integer argument can be given to the continue command to skip commands from nested loops.

Continue n.

Here n specifies the nth enclosing loop to continue from.

Example

The following loop makes use of the continue statement which returns from the continue statement and starts processing the next statement:-

```
#!/bin/sh

NUMS = " 1  2  3  4  5  6  7 "

for NUM in $NUMS

do
  ϕ = ' expr $NUM % 2 '

if [$ϕ -eq 0]
then
   echo " Number is an even number!! "
```

```
continue
fi
  echo "found odd number"

done.
```

Conclusion:- Hence we have studied Basics of shell programming.

Exp. 3. Implementation of Scheduling Algorithm. : FCFS.

Aim:- Implementation of Scheduling algorithm : FCFS.

Objective:-

- To understand the difference between preemptive and Non-preemptive Scheduling.
- To understand Scheduling Criteria and implement FCFS ~~steduli~~ Scheduling algorithm.

Theory / Description:-

Study FCFS Scheduling algorithms.

1.] Explain Scheduling Criteria.

Different CPU scheduling algorithms have different properties and the choice of a particular algorithm depends on the various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.

The criteria include the following:

1. CPU Utilisation.
2. Throughput
3. Turaround time.

4. Waiting Time.
5. Response Time.

2.] What is throughput, Turnaround time, waiting Time and Response Time?

→

a.) Throughput :-
A measure of the work done by the CPU is the number of processes being executed and completed per unit time. This is called Throughput.

b.) Turnaround Time :-
For a particular process an important criteria is how long it takes to execute that process. This is called Turnaround Time.

c.) Response Time :-
In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue.

d.) Waiting time :-
A scheduling algorithm does not affect the time required to complete the process once it starts execution.

Algorithm for FCFS Scheduling

Step 1 : Start the process
Step 2 : Accept the number of processor in the reddy queue.
Step 3: For each process in the ready Queue, assign the process id and accept the CPU burst time.
Step 4: Set the waiting of the first process as '0' and its burst time as its turnaround time.
Step 5: For each process in the ready and calculate.
(a) Waiting time for process (n) : waiting time of process (n-1) + Burst time of process (n-1)
(b) Turnaround time for process (n) : waiting time of process (n) of Burst time for process (n)
Step 6:- Calculate (c) average waiting time : total waiting time / Number of process.
(d) Average Turnaround time = Total Turnaround time / Number of Process.
Step 7: Stop the process.

Conclusion:-
Thus we have understood implementation of scheduling algorithm FCFS.

Experiment No : 4

Title:- Shell programming for the file handling.

Objective:-

- To understand shell programming for file handling in Unix.
- To learn and perform shell programming for file handling in Unix.

Theory / Description:-

1. head command:

It is the complementary of Tail command. The head command as the name implies, print the top N number of data of the given input. By default it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

Syntax:-
head [OPTION] : .... [File] ....

Option:

1: -n num: prints the first 'num' lines instead of first 10 lines. num is mandatory to be specified in command otherwise it displays an error.

```
$ head -n5 state.txt.
Andra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh.
```

Applications of head command.

1. Print line between M and N lines: For this purpose we use head, tail and pipeline commands.
Command is: head -M file name | tail - (M-N);
Since first time line takes first M lines and tail command cuts (M-N) lines starting from end. Let say from state.txt file we have to print lines between 10 and 20.

```
$ head -n 20 state.txt 1-10
Jharkand
Karnataka
Kerala
Madhya Pradesh
```

Maharashtra
Manipur
Meghalaya
Mizoram
Nagaland
Odisha

2. How to use head with pipeline (1): The head command can be piped with other commands. in the following example the output of the ls command is piped to head to show only the three most recently modified files or folder.

Display all recently modified or recently used files

```
$ ls -t
e.txt
d.txt
c.txt
b.txt
a.txt
```

cut three most-recently used file

```
ls -t | head -n3
e.txt
d.txt
c.txt.
```

It can also be piped with one or more filters for additional procesiing. For example, the sort filter could be used to sort the three most recently used files or folders in the alphabetic order.

```
$ ls -t | head -n3 | sort
c.txt
d.txt
e.txt
```

There are number of other filters or commands along which we use head command. Mainly, it can be used for viewing huge log files in Unix.

2. tail command

It is the complementary of head command. The tail command, as the name implies, print the last N number of data of the given input. By default it prints the last 10 lines of the specified files. If more than one files name is provided then data from each file is preceded by its file name.

Syntax:
tail [OPTION].... [FILE]....

Option:

1] -n num:

$ tail -n file-name

Prints the last 'num' lines instead of last 10 lines num is mandatory to be specified in command otherwise it displays an error. This command can otherwise it displays an error. This command can also be written as without symbolizing 'n' character but '-' sign is mandatory. Tail command also comes with an '+' option which is not present in the head command. With this option tail command prints the data starting from specified line number of the file instead of end.

for command : tail +n file-name, data will start printing from line number 'n' till the end of the file specified.

$ tail -n3 state.txt.
Uttar Pradesh
Uttarakhand
West Bengal
OR

$ tail -3 state.txt
Uttar Pradesh
Uttarakhad
West Bengal.

Applications of tail command

1. How to use tail with pipes (|) : The tail command can be piped with many other commands of the unix. In the following example output of the tail command is given as input to the sort command with -r option to sort the last 7 state names coming from file state.txt in the reverse order.

```
$ tail -n7 state.txt
Sikkim
Tamil Nadu
Telangana
Tripura
Uttar Pradesh
Uttarakhand
West Bengal.

$ tail -n 7 state.txt | sort -r

West Bengal
Uttarakhand
Uttar Pradesh
Tripura
Telangana
Tamil Nadu
Sikkim.
```

It can also be piped with one or more filters for additional processing. Like in the following example, we are using cat, head and tail command and whose output is stored in the file name list.txt using directive (>).

```
$ cat state.txt | head -n 20 | tail -n 5 > list.txt

$ cat list.txt
 Manipur
Meghalaya
Mizoram
Nagaland
 Odisha
```

First cat command gives all the data present in the file state.txt and after that pipe transfer all the output coming from cat command to the head command. Head command gives all the data from start (line number 1) output coming from head command to tail command. Now, tail command gives last 5 lines of the data and the output goes to the file name list.txt via directive operator.

3. Cut Command:

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the results to standard output. It can be used to cut parts of a line by byte position, character and field. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error. If more than one file name is provided then data from each file is not precedes by its file name.

Syntax:-
cut [OPTION] ..... [FILE]....

Let us consider the files having name state.txt and capital.txt contains 5 names of the Indian states and capitals respectively.

$ cat state.txt
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chattisgarh.

OPTION:

1] -c (column):

To cut by character use the -c option. This selects the characters given to the -c option. This can be a list of numbers seperated comma or a range of numbers seperated by hyphen (-). Tabs and backspace are treated as a character. It is necessary to ~~specify~~ specify list of characters numbers otherwise it gives error with this option.

Syntax:

$ cut -c [(k) - (n) / (k), (n) / (n)] file name

Here, k denotes the starting position of the character and n denotes the ending position of the character in each lines, if k and n are seperated by "-" otherwise they are only the position of character in each line from the file taken as an input.

$ cut -c , 2,5,7 state.txt
nr
rah
sn
ir
nti

2) -f(field):
-c option is useful for fixed-length lines. Most unix files doesn't have fixed-length lines. To extract the useful information you need to cut by fields rather than columns. List of the fields number specified must be seperated by comma, Ranges are not described with -f option. cut uses tab as a default field delimiter but can also work with other delimiter by using -d option.

Note: Space is not considered as delimiter in Unix.

Syntax:
$ cut -d "delimiter" -f (field number) file.txt

Like in the file state.txt fields are seperated by space if -d option is not used then it prints whole line.

```
$ cut -f1 state.txt
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh.
```

If -d option is used then it considered space as a field separator or delimiter:

```
$ cut -d " " -; f | state.txt.
```

Andra
Arunachal
Assam
Bihar
Chhattisgarh

Command prints field from first to fourth of each line from the file.

Command:

```
$ cut -d " " -f |-4 state.txt
```

Output:

Andra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh.

3. Paste command

Paste command is one of the useful commands in Unix or linux operating system. It is used to join files horizontally (parallel merging) by outputing lines consisting of lines from each files specified, seperated by tab or delimiter, to the standard output. When no file is specified, or put dash ("-") instead of file name, paste reads from standard input and gives output as it's until a interrupt command [ctrl.-c] is given.

Syntax:
paste [OPTION] ... [FILE] ...

4. Sort command

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts files assuming the contents are ASCII. Using the options in sort command, it can also be used to sort numerically.

Syntax:
$ sort filename.txt

OPTIONS:

1) -n option:
To sort a file numerically used -n option. -n option is also predefined in unix as the above options are: This option is used to sort the file with numeric data present inside.

Syntax:
$ sort -n filename.txt

2) -nr option:
To sort a file with numeric data in reverse order we can use the combination of two options as stated below.

Syntax:
$ sort -nr filename.txt

3) -k option: Unix provides the features of sorting a table on the basis of any column number by using -k option. Use the -k option to sort on a certain column.

Syntax:
$ sort -k filename.txt

4) -c option:
This option is used to check if the file given is already sorted or not and checks if a file is already sorted pass the -c option to sort. The sort tool can be used to understand if this file is sorted and which lines are out of order.

Syntax:
$ sort -c filename.txt

5) -u option:
To sort and remove duplicates pass the -u option to sort. This will write a sorted list to standard output and remove duplicates.
This option is helpful as the duplicates being removed gives us an redulant file.

Syntax:
$ sort -u filename.txt

Conclusion:-

Thus we have understood and implemented shell programming for file handling.

Experiment no. 5

Title :- Implementation of CPU scheduling algorithm: SJF.

Objective:-

- To understand the difference b/w preemptive and non-preemptive scheduling.
- To understand scheduling criteria and implement SJF scheduling algorithm.

Theory / Description:-

1] Explain Scheduling Criteria.

Different CPU scheduling algorithm have different properties and the choices of a particular algorithms depends on the various factors. Many criteria have been suggested for comparing CPU scheduling algorithm.

The criteria include the following:

1. CPU utilisation
2. Throughput
3. Turnaround Time.
4. Waiting Time.
5. Response Time.

2] What is throughput, Turnaround time, waiting time and response time?

→

a] Throughput:-
A measure of the work done by CPU is the number of processes being executed and complete per time. This is called Throughput.

b] Turnaround Time:-
For a particular process an important criteria is how long it takes to execute that process. This is called Turnaround Time.

c] Response Time:-
In an interactive system, turn-around time is not the best criteria. A process may produce some fairly early and continue.

d] Waiting Time:-
A scheduling algorithm does not affect the time required to complete the process once it starts execution.

Algorithm:-

Step 1: Start the process

Step 2: Accept the number of processes in ready queue.

Step 3: For each process in the ready Q, assign the process id and accept CPU burst time.

Step 4: Start the ready Q according the shortest Burst time by sorting according to lowest highest burst time.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate
(a) waiting time for process (n) = waiting time of process (n-1) + Burst time of process (n)

Step 7:- Calculate (c) ~~Ava~~ Average waiting time = Total waiting time / Number of process
(d) Average Turnaround Time = Total Turnaround Time / Number of process.

Step 8 :- Stop the process.

Program:-

```
#include <stdio.h>
int main ()
{
~~bt[20]~~ int
bt[20], p[20], wt[20], tat[20], i, j, n, total=0, pos, temp;
float avg-wt, avg_tat;
printf ("Enter number of proces:");
scanf ("%d", &n);

printf ("\n Enter Burst Time: ");
for (i=0; i<n; i++)
{
 printf ("p%d ", i+1);
 scanf ("%d", &bt[i]);
 p[i] = i+1;
}

// sorting of burst times
 for (i=0; i<n; i++)
{
  pos = i;
 for (j= i+1; j<n; j++)
  {
    if (bt[j] < bt[pos])
   pos = j;
  }
```

```
temp = bt[i];
bt[i] = bt[pos];
bt[pos] = temp;

temp = p[i];
p[i] = p[pos];
p[pos] = temp;
}
 wt[0] = 0;

for(i=1; i<n ; i++)
{
  wt[i]=0;
for(j=0; j<n ; j++)
  wt[i] += bt[j];
}
total += wt[i]
}
avg_wt = (float) total /n;
total=0;

printf("\n Proces Burst time + waiting time +
       Turnaround time");
for(i=0; i<n ; i++)
{
  tat[i] = bt[i] + wt[i];
  total += tat[i];
```

```
printf ("\n p%d \t\t %d \t\t %d \t\t\t %d", p[i],
        bt[i], wt[i], tat[i]);
}
avg_tat = (float)total/n;
printf ("\n\n Average waiting Time = %f", avg_wt);
printf ("\n Average Turnaround Time = %f\n", avg_tat);
}
```

Output:-

Enter Burst Time:

P_1: 4

P_2: 3

P_3: 7

P_4: 1

P_5: 2

Process	Burst Time	Waiting Time
P_4	1	0
P_5	2	1
P_2	3	3
P_1	4	6
P_3	7	10

Average waiting Time = 4

Average Turnaround Time = 7.4

Conclusion:- Thus we have implemented SJF scheduling algorithm.

Experiment no. 6

Title:- To implement Quick Sort and Merge Sort Algorithm.

Objective:-
- To understand and implement Quicksort.
- To understand and implement Merge Sort.

Theory / Description:-

Sorting Algorithm:-

A sorting algorithm is used to rearrange a given array or list elements according to a comparison operator on the element. The comparison operator is used to decide the new order of element in the respective data structure.

Quick sort:-

Quick sort is a divide and conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

Merge sort:-

Merge sort is a divide and conquer algorithm. It divides the input array in two halves, calls itself for the two halves, and then it merges the two sorted halves.

Algorithm:-

Merge sort:-

Step I :- Start

Step 2 :- declare array and left, right, mid variable

Step 3 :- Perform merge functions.

```
mergesort (array, left, right)
mergesort (array, left, right)
if left > right
return
mid = (left + right)/2
mergesort (array, left, right)
mergesort (array, mid+1, mid)
mergesort (array, left, mid, right)
```

Step 4 :- Stop.

Quick Sort:-

Step 1 :- Choose the highest index value has pivot

Step 2 :- Take two variables to point left and right of the list excluding pivot.

step 3 :- left points to the low index.

Step 4 :- right points to the high

Step 5 :- while value at left is less than pivot move right.

Step 6 :- while value at right is greater than pivot move left.

Step 7:- If both step 5 and step 6 does not match swap left and right.

Step 8:- If left >= right, the point where they met is new pivot.

Program:-

Quick Sort:-

```
# include <stdio.h>
int partition (int a[], int start, int end)
{
 int pivot = a [end];
 int i = (start - 1);

for ( int j = start ; j <= end-1 ; j++)
 {
  if (a[j] < pivot)
  {
    i++;
   int t = a[i];
    a[i] = a[j];
    a[j] = t;
  }
}
int t = a[i+1];
a[i+1] = a[end];
```

```
a[end] = t;
return (i+1);
}
void quick (int a[], int start, int end)
{
   if (start < end)
   {
    int p = partition (a, start, end)
       quick (a, start, p-1)
       quick (a, p+1, end)
   }
}

void print Arr(int a[], int n)
{
   int i;
   for (i=0; i<n; i++)
      printf ("%d", a[i]);
}
int main()
{
  int a[] = {24, 9, 29, 14, 19, 27};
    int n = sizeof(a) / size of (a[0]);
  printf ("Before sorting array elements are - \n")
  printf Arr (a,n);
  quick (a, 0, n-1);
```

```
printf("\n After sorting array element are-\n");
print Arr(a,n);

return 0;
}
```

Output:-

Before sorting array elements are:-
24 9 29 14 19 27

After sorting array elements are:-
9 14 19 24 27 29

Conclusion:-

We have understood and ~~implented~~ implemented quick sort and merge sort algorithm.

Experiment no. :- 7.

Aim :- Implementation of various memory allocation algorithms (First fit, Best fit and Worst fit).

Objective :- To understand and implement various memory allocation algorithms :-
- First fit
- Best fit
- Worst fit.

Theory :-

1. First fit :-

In the first fit, approach is to allocate to the first free partition or hole large enough, which can accomodate the process. It finishes after finding the first suitable free partition.

Advantage :-
- Fastest algorithms because it searches as little as possible.

Disadvantage :-
- The remaining unused memory area left after allocation become waste if it too smaller. Thus request for larger memory requirement cannot be accomplished.

2. Best fit:-

The best fit deals with allocating the smallest free partition which meets the requirement of the ~~entire list~~ requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Advantage:-
- Memory ~~utti~~ utilization is much better than first fit as it searches the smallest free partition ~~fee~~ first available.

Disadvantage:-
- Is is slower and may even tend to fill up memory with tiny useless holes.

3. Worst fit:-

In wont fit approach is to locate largest available free partition so that the portion left will be big enough to be useful. It is the reverse of best fit.

Advantage:-
- Reduces the rate of production of small gaps.

Disadvantage:-

- If a process requiring larger memory arrives at a later stage it cannot be accomodated as the largest hole is already split and occupied.

Programs:-

C implementation of first-fit algo.

```
# include <stdio.h>
void firstfit (int blocksize[] , int m,
        int processSize[] , int n)
{
  int i, j;
  int allocation [n];

  for (i=0 ; i<n ; i++)
  {
    allocation [i] = -1;
  }

  for (i=0; i<n ; i++)
  {
    for (j=0; j<m ; j++)
    {
      if (blocksize [j] >= processsize [i])
      {
```

```
            allocation[i] = j;
            blocksize[j] == processsize[i];
            break;
          }
        }
      }

printf("\n Process no. \t Process size \t Block no. \n");

for (int i=0; i<n; i++)
 {
   printf("%i \t\t\t", i+1);
   printf("%i \t\t\t", processsize[i]);

  if (allocation[i] != -1)
    printf("%i", allocation[i]+1);
  else
    printf("Not allocated");
   printf("\n");
  }
}

int main()
 {
   int m, n;
   int blockSize[] = {100, 500, 200, 300, 600};
   int processSize[] = {212, 417, 112, 426};
```

```
m = sizeof (blocksize) / size of (blocksize[0]);
n = size of (processSize) / sizeof (processsize [0]);

first fit (blocksize, m, process size, n);

return 0;
}
```

Output:-

Process no.	Process Size	Block no.
1	212	2
2	417	5
3	112	2
4	426	Not Allocated

Conclusion:-

Thus, we have understood and implemented various memory allocation algorithms.