**Midterm**

**Lab-1**

Date: 15/12/2020

**Topic:** File create/remove, directory create/remove

**Key words:**

In Linux Folder = Directory

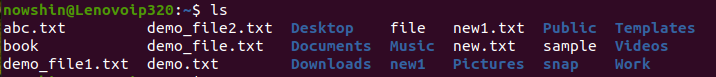
1. To show the current working directory.

pwd



1. To see the files on a directory (folder) as a list. (It will show the visible files only).

ls



1. To change directory (go to another).

cd



1. To back from one directory to previous directory.

cd ..



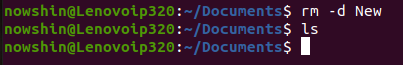
1. To make a new directory.

mkdir



1. To remove a directory.

rm –d



1. To create a file.

touch new.txt (file name and type: ex- newfile.txt)

or, cat > new.txt

To



1. To remove a file.

rm (file name: newfile.txt)



1. To edit writing inside a file/ edit a file.

cat (file name: ex- newfile.txt)

1. To enter into vi editor.

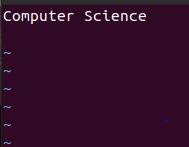
vi (file name: ex- newfile.txt)

press “i” to start writing.

Press “:w” to save the writing.

Press “:q” to go back.



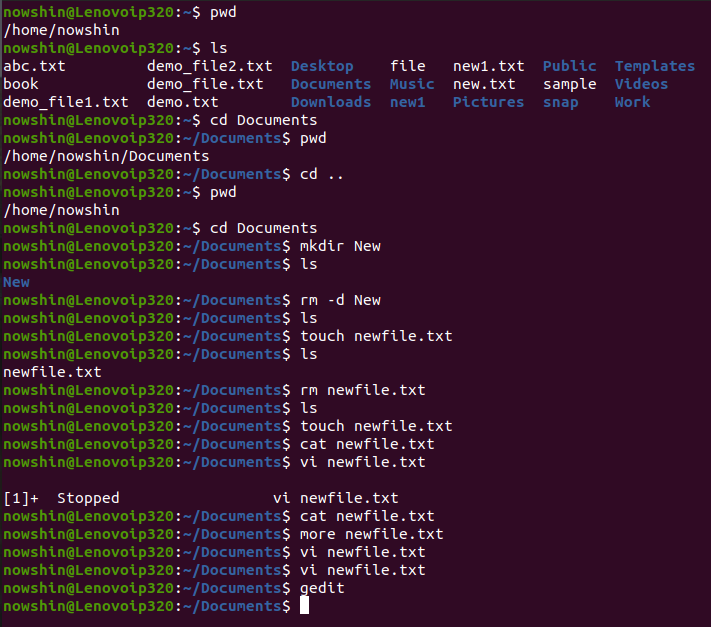


1. To open gedit.

gedit



**All in one**



**Lab-2**

Date: 22/12/2020

**Topics:** ls, chmod(file permission), head,tail

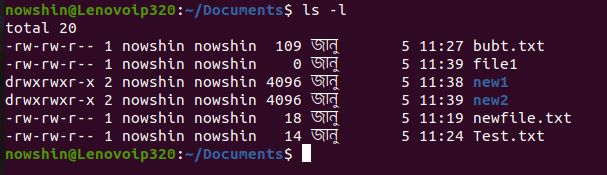
1. To show the hidden files.

ls –a



1. To see the details information of a file.

ls –l



The -l ( lowercase L) option tells ls to print files in a long listing format.

When the long listing format is used, you can see the following file information:

* The file type.
* The file permissions.
* Number of hard links to the file.
* File owner.
* File group.
* File size.
* Date and Time.
* File name.

When will see the information in details (we will find 10 characters/ values).

There can be 3 types of user in Linux:

i)Owner / User (that’s me)

ii) Group (when people works in work station. Work station = different people can get access in the same computer (they will not see each other’s content, but some content will be common for all).

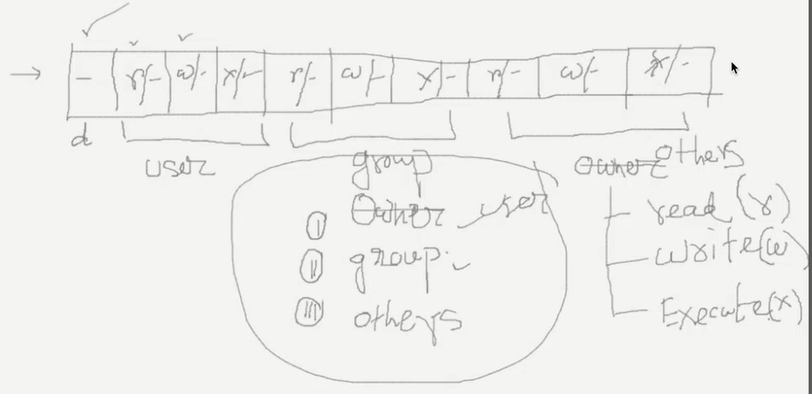
ii) Others

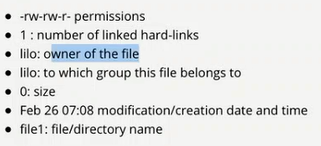
Normally we can do this tasks for a file:

* Read (r)
* Write (w)
* Execute (x)
* 🡪 File

d 🡪 directory

if user can write = “w” otherwise “– “





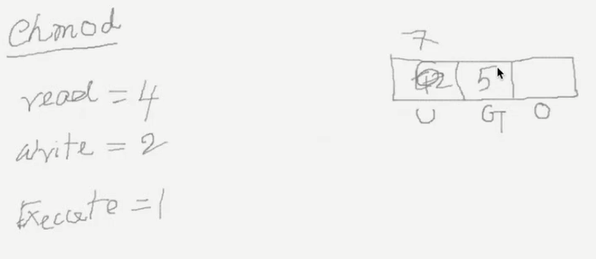
1. To add execute permission for User.

Command to modify mode: **chmod**

We can update mode by giving number.

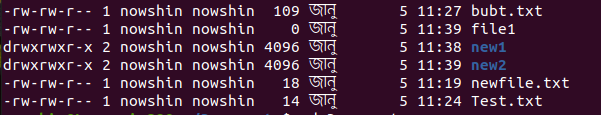
* Read = 4
* Write = 2
* Execute = 1

We can control the whole thing using a 3-digit number:



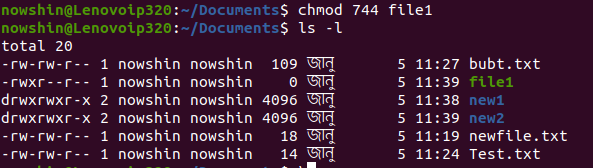
**Change permission for files**

Before:

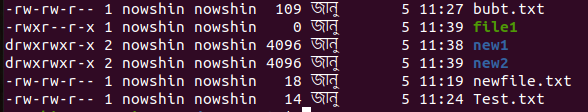


After:

To give permission to write for group and others (here, for file1):



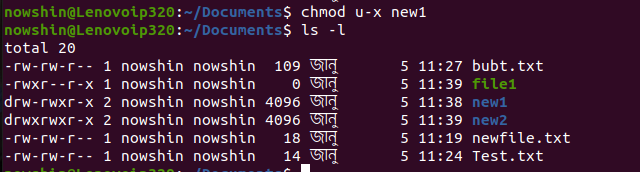
To give permission to write and execute for others (here, for file1):



**Change permission for directory**

1. Owner / User 🡪 u
2. Group 🡪 g
3. Others 🡪 o

To remove execute permission for user from directory **new1:**



**To change permission for multiple users:**

Ex: u+w,g+w new1.txt (have to use comma(,))

Before:



After:





It is same as for directory to change permission.

Ex: u+w,g+w new1

**To change permission for all kinds of users:**

Before:



After:





To write something on a file:



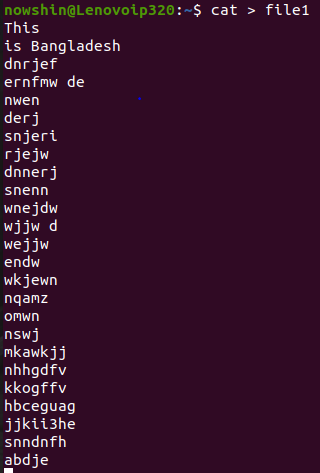
To see it:



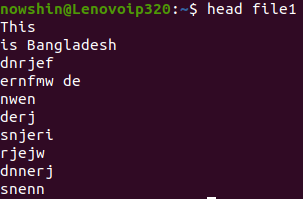
To see the first 10 lines of a file:

**head file1**

Before:

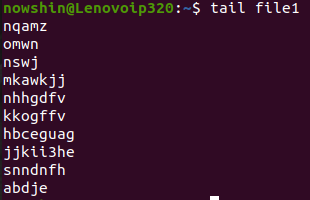


After:



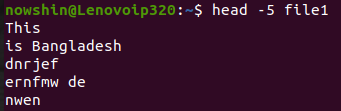
To see the last 10 lines of a file:

**tail file1**

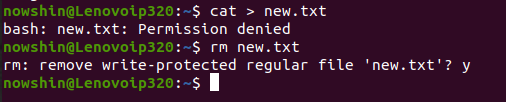


To see the lines of a file as per our wish:

head -5 file1 (here, if we want to see first 3 lines then we will write head -3 file1)



If permission is not given, then we will see this message and have to choose y/n (yes/no).

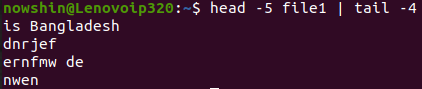


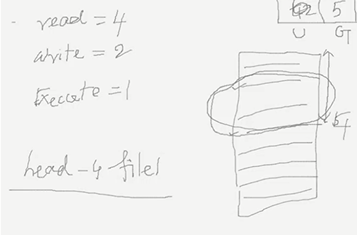
**Pipelining**

One pipes output is input of another pipe. It is same for command (One commands output is input of another command).

Symbol: |

To see 2nd,3rd,4th and 5th line (2-5th line) of the first 5 lines (without line 1):

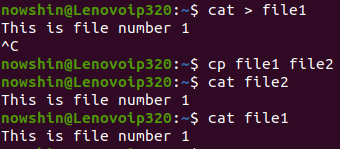




To copy something from one file to another (ex: file1 to file2):

cp file1 file2 (source folder destination folder)



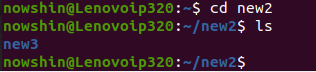


To move one directory to another:

mv new3 new2 (dic to move dic in where we will move)



Directory new3 moved into new2



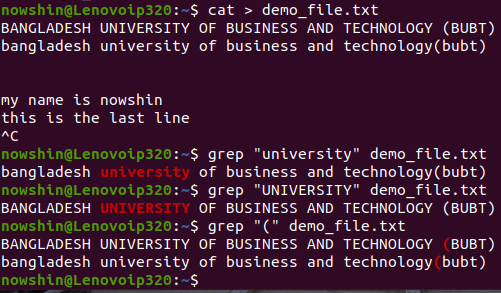
**Lab-3**

Date: 29/12/2020

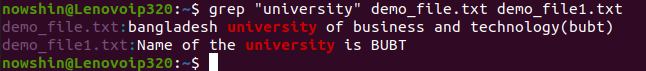
**Topics:** Grep, Sort, Pipe

To see/search string from a file.

grep “string” file name (ex: grep “university” demo\_file.txt)

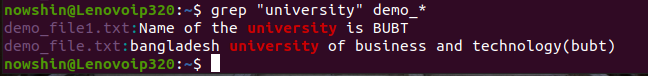


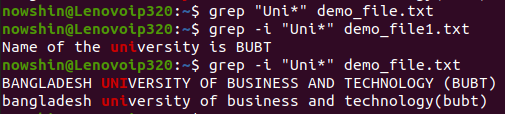
To see the string for multiple files:



To see/find the string using regular expression:

grep “string” file\*





For case insensitive:



To see/find how many lines match:

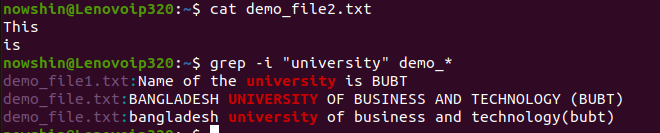
Here “c” = count.



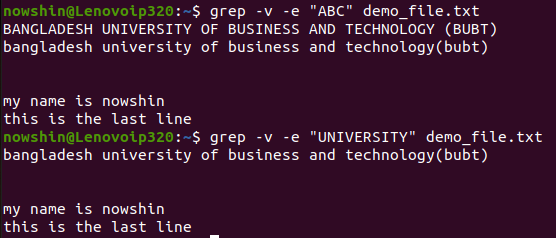
For case insensitive:



Here, demo\_file2.txt does not consist with the word “university” so demo\_file2.txt is not showing here.



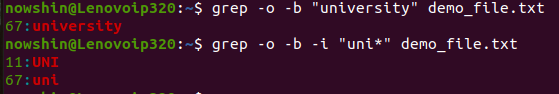
To see/find how many lines does not match:



To avoid details info and to show the file name only:



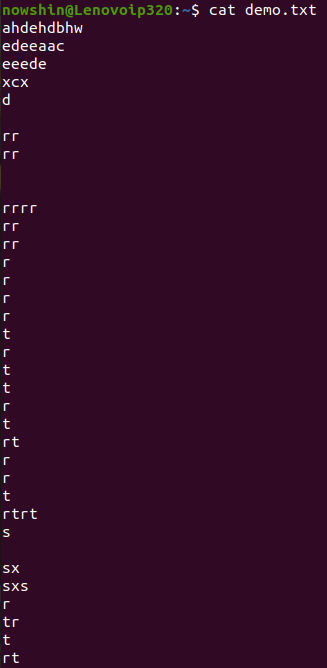
To see/find the string position on a file:



Class task1: Do the lab task: Run the “System info” commands.

To see some contents from a lot of contents:

Before:

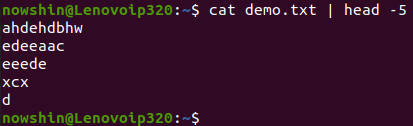


After:

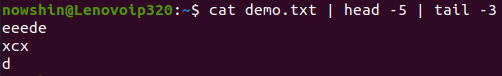
Here, space is also counted as a line. So, it showed first 10 lines (as we used head).



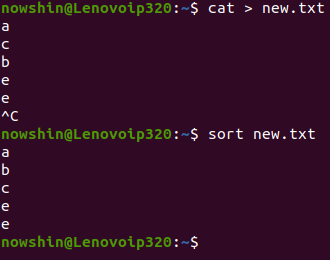
First 5 lines:



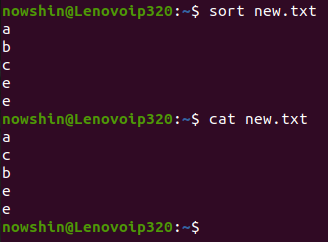
To see last 3 lines:



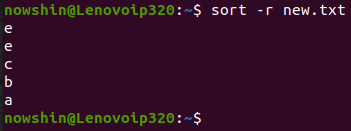
To see the data of file sorted:



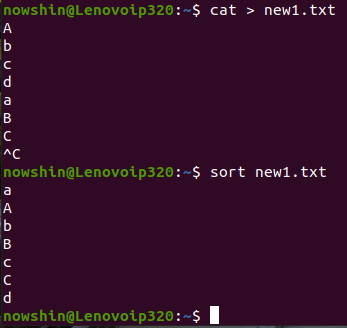
Here, data of the main file is not sorted:



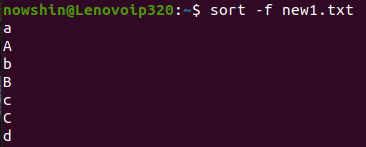
To see the data of file reversed:

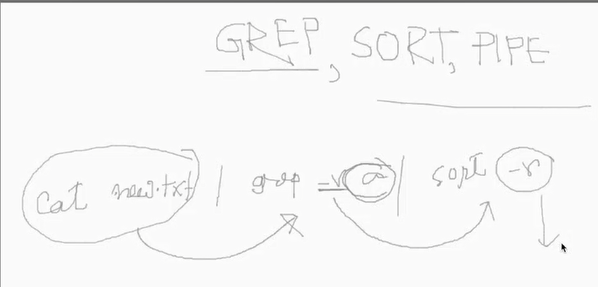


If there are multiple character (small+capital):



For case insensitive





Create regular expressions using meta character.

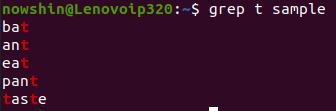
To find the words started with a(a specific character).

grep ^a sample



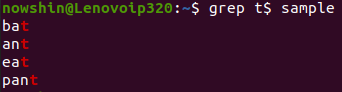
To find the words consist with t

grep t sample



To find the words ends with t (a specific letter)

grep t$ sample



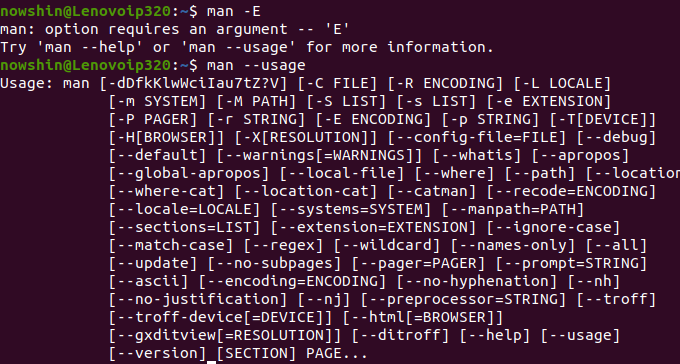
To see one character after another



To see two similar letter one together



To know the Meaning of something



**Lab-4**

Date: 29/12/2020

**Topics:** Grep, Sort, Pipe

**Final Term**

**Lab-1**

Date: 09/02/2021

**Topic:** Producer Consumer Problem. Process Synchronization Technique Using Semaphore we have to do this problem.

Lab Task: FCFS scheduling algorithm (Implement using Unix C or Shell Scripting).

**Question:** Write down a shell script program that implements the FCFS scheduling algorithm. Your program will take n number processes as input having arrival time and burst time and produces the waiting time, turnaround time, and response time of each process. Finally calculate the average waiting time, average response time and average turnaround time.

**Solution:**

#include<stdio.h>

int main () {

int burst\_time [30], waiting\_time [30], turnaround\_time [30];

float average\_waiting\_time = 0.0, average\_turnaround\_time = 0.0;

int count, j, n;

printf("Enter The Number of Processes To Execute:\t");

scanf("%d", &n);

printf("\nEnter The Burst Time of Processes:\n\n");

for(count = 0; count < n; count++) {

printf("Process [%d]:", count + 1);

scanf("%f", &burst\_time[count]); }

waiting\_time [0] = 0;

for(count = 1; count < n; count++) {

waiting\_time [count] = 0;

for(j = 0; j < count; j++) {

waiting\_time[count] = waiting\_time[count] + burst\_time[j]; } }

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time\n");

for(count = 0; count < n; count++) {

turnaround\_time [count] = burst\_time [count] + waiting\_time [count];

average\_waiting\_time = average\_waiting\_time + waiting\_time [count];

average\_turnaround\_time = average\_turnaround\_time + turnaround\_time [count];

printf("\nProcess [%d]\t\t%.2f\t\t%.2f\t\t%.2f", count + 1, burst\_time [count], waiting\_time

[count], turnaround\_time [count]); }

printf("\n");

average\_waiting\_time = average\_waiting\_time / count;

average\_turnaround\_time = average\_turnaround\_time / count;

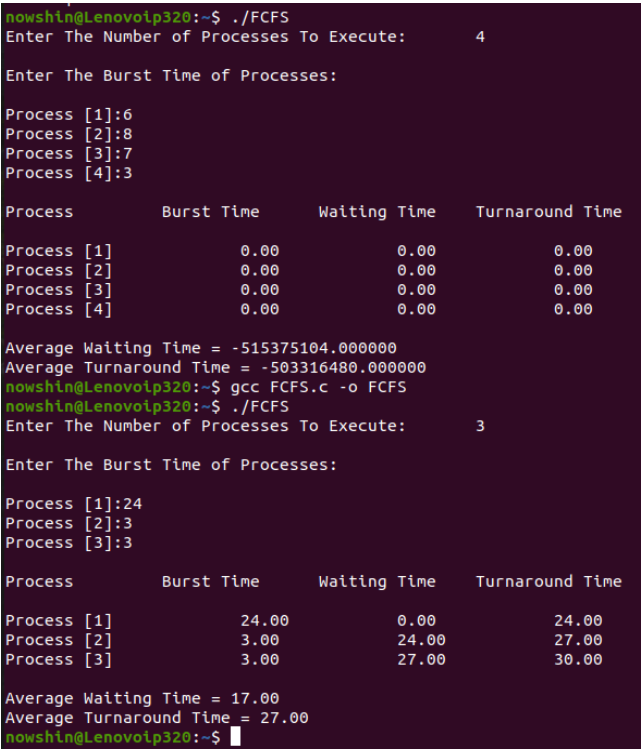
printf("\nAverage Waiting Time = %f", average\_waiting\_time);

printf("\nAverage Turnaround Time = %f", average\_turnaround\_time);

printf("\n");

return 0; }

**Output:**



**Lab-2**

Date: 23/02/2021

**Lab work:** SJF algorithm(preemptive).

**Topic:** Banker’s Algorithm for Deadlock detection.

**H.W:** Resource request algorithm for deadlock detection.

* There are 2 types of deadlock avoidance algorithm:

1. Resource-allocation Algorithm. (For single instance 🡪 single instance per resource).
2. The Banker’s Algorithm. (For single instance).

* There are 2 approaches for deadlock detection (For Banker’s Algorithm).

1. Safety Algorithm
2. Resource Request Algorithm
3. */\*\*\**


7. *\*\*\*/*
9. #include <stdio.h>
10. #include <string.h>
11. #include <math.h>
12. #include <stdlib.h>
13. #include <ctype.h>
15. #include <algorithm>
16. #include <iostream>
17. #include <vector>
18. #include <map>
19. #include <set>
20. #include <string>
21. #include <sstream>
22. #include <queue>
23. #include <list>
24. #include <string>
25. #include <stack>
27. #define ll long long
28. #define smallestnum 1e18
29. using namespace std;
30. #define MAX 1000

33. int  main()
34. {
36. //return 0;
37. int memsize=150;
38. float pagesize;
39. int nofpage;
40. int p[10];
41. int frameno,offset;
42. int logadd,phyadd;
43. int i;
44. int choice =0;
45. printf("Memsize is %d**\n**",memsize);
46. printf("Enter page size: ");
47. scanf("%f",&pagesize);
48. nofpage= ceil((memsize\*1.0)/pagesize);
49. printf("Number of total pages are: %d**\n**", nofpage);
50. for(i=0; i<nofpage; i++)
51. {
52. printf("Enter the frame of page%d:",i+1);
53. scanf("%d",&p[i]);
54. }

57. do
58. {
59. printf("**\n**Enter a logical address:");
60. scanf("%d",&logadd);
62. int div=10;
64. if(logadd<1000  & logadd>100){
65. div = 100;
66. }
67. frameno = logadd/div;
68. cout << "Frameno: " << frameno << endl;
70. offset=logadd%div;
71. cout << "Ofset: "  << offset << endl;
72. phyadd=(p[frameno]\*pagesize)+offset;
73. printf("**\n**Physical address is: %d",phyadd);
74. printf("**\n**Do you wat to continue(1/0)?:");
75. scanf("%d",&choice);
76. printf("**\n\n**");
77. }
78. while(choice==1);

81. return 0;
82. }

**Lab-3**

Date: 05/03/2021

**Topic:** 1) Memory allocation: First fit, Best fit and Worst fit algorithm implementation.

2) Paging: Page replacement algorithm.

**Lab-4**

Date: 09/03/2021

**Topic:** 1) Paging: Page replacement algorithm for demand paging.

2) Page replacement algorithm:

* FIFO (sir taught it)
* Optimal Page replacement algorithm (H.W)
* LRU Algorithm (H.W)

**Lab-5**

Date: 16/03/2021

**Topic: Readers-Writers problem.**

**For lab final exam:**

* Deadlock (Banker’s algorithm for deadlock detection).
* Memory management (
* Process Synchronization
* Virtual memory

1. #include<bits/stdc++.h>
2. using namespace std;
4. int sum=0;
5. void bestFit(int blockSize[], int m, int processSize[], int n)
6. {

9. int allocation[n];
11. memset(allocation, -1, sizeof(allocation));
13. for (int i=0; i<n; i++)
14. {
15. int bestIdx = -1;
16. for (int j=0; j<m; j++)
17. {
18. if (blockSize[j] >= processSize[i])
19. {
20. if (bestIdx == -1){
21. bestIdx = j;
22. }
24. else if (blockSize[bestIdx] > blockSize[j])
25. {
26. bestIdx = j;
27. }
28. }
29. }
31. if (bestIdx != -1)
32. {
33. allocation[i] = bestIdx;
35. blockSize[bestIdx] -= processSize[i];
37. sum-=processSize[i];
39. }
40. }
42. cout << "**\n**Process No.**\t**Process Size**\t**Block no.**\n**";
43. for (int i = 0; i < n; i++)
44. {
45. cout << " " << i+1 << "**\t\t**" << processSize[i] << "**\t\t**";
46. if (allocation[i] != -1)
47. cout << allocation[i] + 1;
48. else
49. cout << "Not Allocated";
50. cout << endl;
51. }
52. cout << "External Fragmentation is : " << sum << endl;
53. }

56. int main()
57. {
58. int blockSize[] = {100, 500, 200, 300, 600};
59. int processSize[] = {212, 417, 112, 426, 174};
61. int m = sizeof(blockSize)/sizeof(blockSize[0]);
62. int n = sizeof(processSize)/sizeof(processSize[0]);
63. for(int i=0; i<m; i++){
64. sum+=blockSize[i];
65. }
66. bestFit(blockSize, m, processSize, n);
68. return 0 ;
69. }