**Practical – 1**

# ls (List all the files and directories)

**Description: -** Display the list information about files. **Example:** -



# pwd (Print working directory)

**Description: -** Print name of the working/current directory. **Example: -**



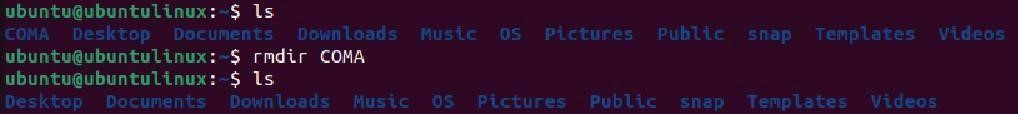
# mkdir (make directories)

**Description: -** Create directory, if they do not already exist. **Example: -**



# rmdir (Remove empty directories)

**Description: -** Remove the directories, if they are empty. **Example: -**



# cat (Create files)

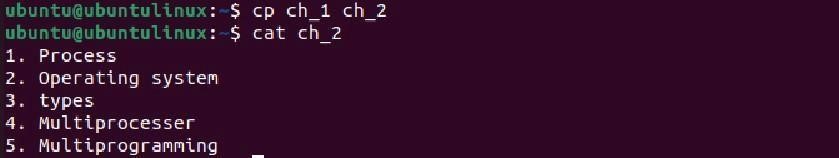
**Description: -** Concatenate files and print on the standard output. **Example: -**





# cp (Copy files and directories)

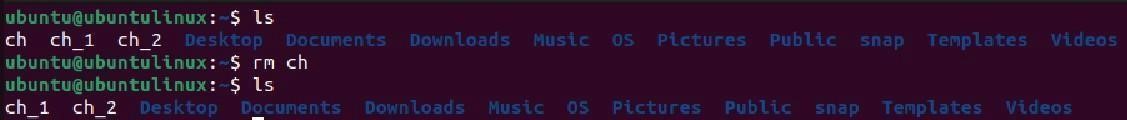
**Description: -** Copy one file data to another file. **Example: -**



# rm (Remove files or directory)

**Description: -** Removes each specified file.

- By default, it does not remove directory. **Example: -**



1. **mv (move (rename) files)**

**Description: -** Rename source or move source to directory. **Example: -**



1. **uname (Print system information) Description:**

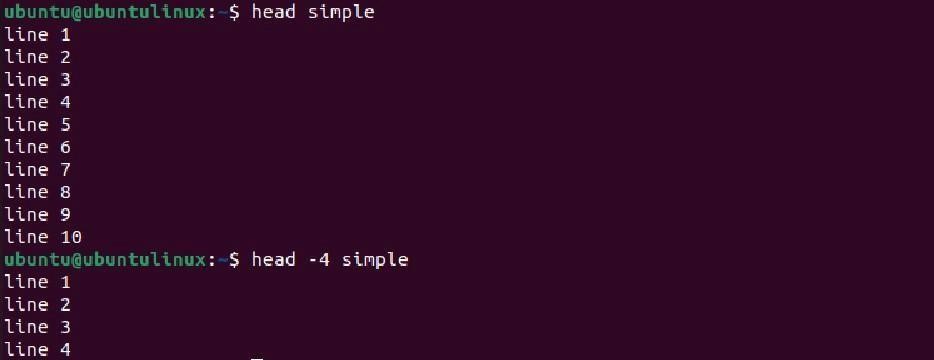
**-** Print certain system information. **Example: -**

1. **head (Output the first part of files)**

**Description: -** Print the first 10 lines of each file to standard output.

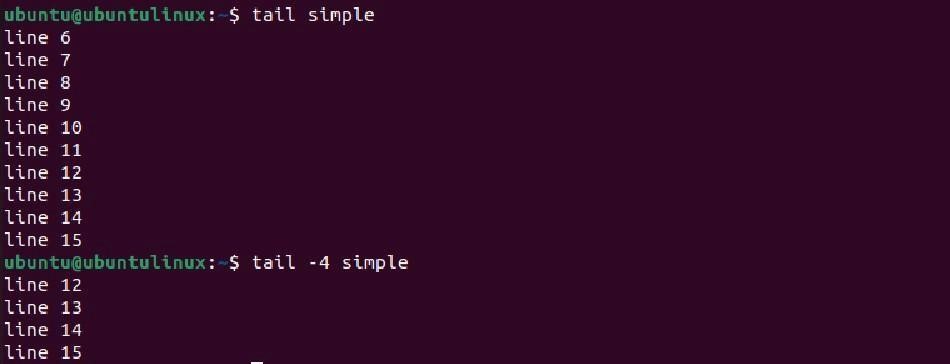
**Example: -**





1. **tail (Output the last part of files)**

**Description: -** Print the last 10 lines of each file to standard output. **Example: -**



1. **grep (Print lines matching a pattern)**

**Description: -** grep searches for the pattern in each file. Use to find specific string in series of output. **Example: -**



1. **history (history library)**

**Description:** - Shows a list of previously executed commands, including their line numbers. **Example: -**



1. **clear (Clear the terminal screen)**

**Description: -** Clear the terminal screen, providing a clean slate for new commands or output. **Example: -**



1. **Echo (Display a line of text)**

**Description: -** echo the string to standard output. **Example: -**



1. **man (manuals)**

**Description: -** an interface to the online reference manuals. **Example: -**

A computer screen with white text

Description automatically generated

1. **cd (Change directory)**

**Description: -** The cd command is used to change the current working directory. Without any arguments, it takes you to your home directory. **Example: -**



1. **Whoami (Print effective user I’d)**

**Description: -** Print the user name associated with the current effective use I’d. **Example: -**



1. **Sort (Sort lines of text files)**

**Description: -** Sorts the lines in unsorted text alphabetically and prints the result. **Example: -**



1. **hostname (System hostname)**

**Description: -** Show or set the system’s hostname. **Example:** -



1. **ps (Process status)**

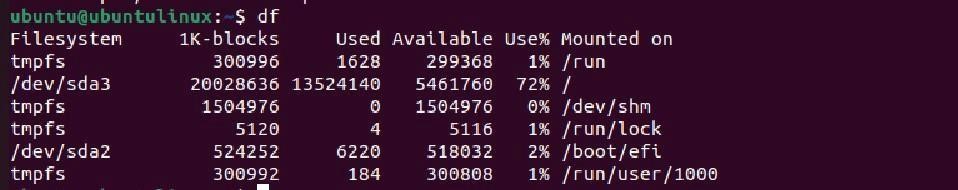
**Description: -** Display information about active processes. **Example:** -



1. **df (Disk free)**

**Description: -** Report file system disk space usage.

**Example:** -



1. **find (Find files)**

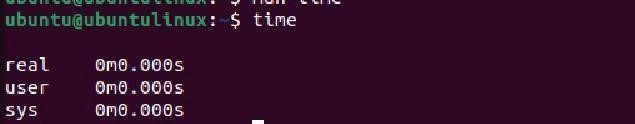
**Description: -** Search for all files with a .txt extension in the specified directory and its sub-directories. **Example: -**



1. **time (execution time)**

**Description: -** Display the information about resources used by command. If command exits with non-zero status that’s means time display a warning message and the exit status.

**Example: -**



**25.chmod (Change mode)**

**Description: -** Change file or directory mode.

**Example: -**

A screenshot of a computer

Description automatically generated

**Practical – 2**

**AIM:** Study the basics of shell programming.

**THEORY:** Basics programs of shell scripting are as follows:

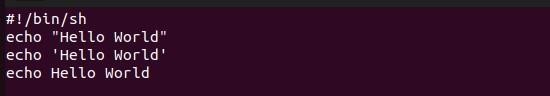
The steps in creating a shell script:

1. Create a file using a vi editor name script file with extension .sh Ex. Addition.sh
2. Then press i (i=insert mode)
3. Start the script with #!/bin/bash
4. Write your code
5. Then press esc
6. :x and Enter or :wq! (! = save and exit)
7. Change the mode execution
8. Run – bash filename.sh or ./filename.sh

**Programs:**

**Program 1:** Write Hello World using shell script.

**Code:**

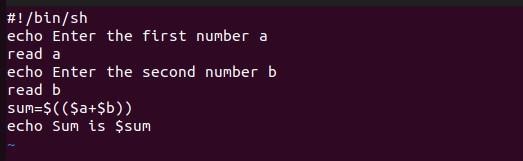


**Output:**

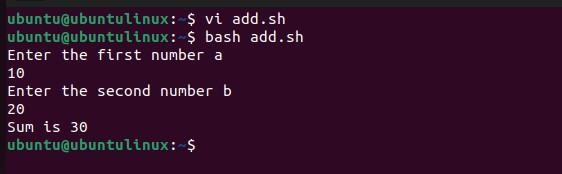


**Program 2:** Write a shell script program to add two numbers.

**Code:**

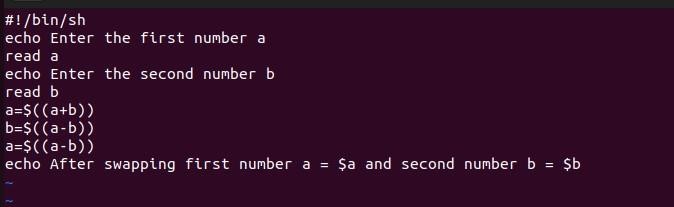


**Output:**

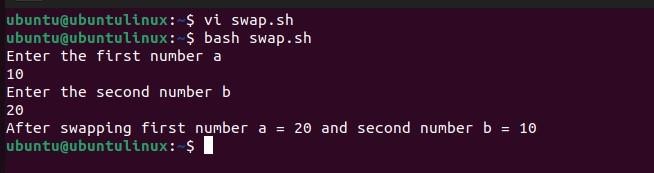


**Program 3:** Swap two variables without using third variable.

**Code:**

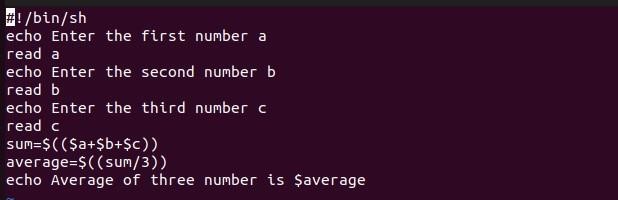


**Output:**

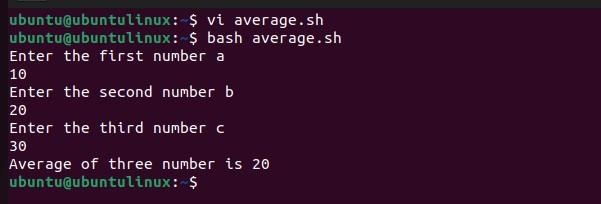


**Program 4:** Find average of three numbers.

**Code:**

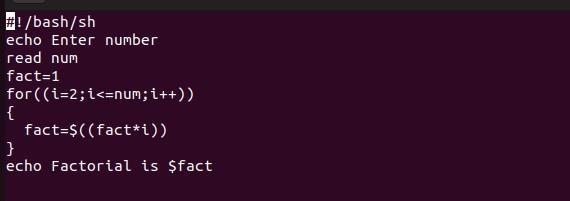


**Output:**

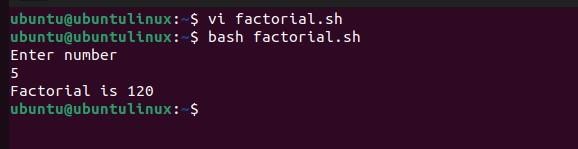


**Program 5:** Calculate factorial of given number.

**Code:**



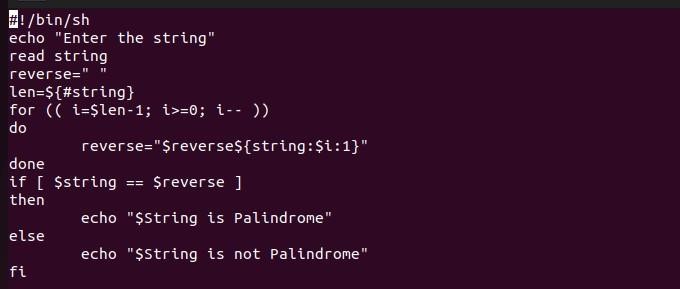
**Output:**



# Practical – 3

**Program 1:** Write a shell script to check entered string is palindrome or not.

**Code:**



**Output:**

A computer screen with white text

Description automatically generated

**Program 2:** Write a shell script to Print an Array.

**Code:**

A computer screen with white text

Description automatically generated

**Output:**

A black background with white text

Description automatically generated

# Practical – 4

1. **Write a c program to create a child process.**

A screen shot of a computer screen

Description automatically generated

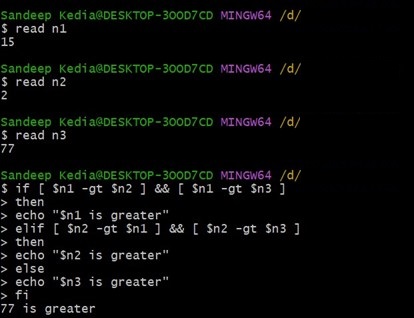
**Output** :-

A screen shot of a computer

Description automatically generated

# Practical – 5

**AIM:** Finding out biggest number from given three numbers supplied as command line arguments



**Practical** – 6

**AIM: Printing the patterns using a for loop.**

* **Write Shell Script to Print Pyramid.**

A computer screen with text on it

Description automatically generated

**Practical** – 7

1. **Shell script to determine whether given file exist or not.**

A screen shot of a computer

Description automatically generated

1. **Write a Shell Script that prints the even number up to the number given by the user.**

A computer screen with text on it

Description automatically generated

**Practical** – 8

**AIM: Write a program for process creation using C. (Use of gcc compiler).**

## THEORY

GCC COMPILER:

GCC, which stands for GNU Compiler Collection, is a compiler system produced by the GNU Project. It is used to compile programs written in various programming languages, including C, C++, Objective-C, FORTRAN, Ada, and others. GCC is widely used in both academia and industry and is available for many platforms.

KEY FEATURES OF GCC COMPILER:

* Multi-Language Support: GCC supports several programming languages, including C, C++, Objective-C, FORTRAN, Ada, and others. This makes it versatile for various software development needs.
* Optimization Capabilities: GCC provides a plethora of optimization options to improve the performance of compiled code. It includes optimizations for speed, size, and speci ic processor architectures.
* Portability: GCC is highly portable and is available on a wide range of platforms, including Unix-like systems (Linux, BSD, macOS), Windows, and others. Its portability enables developers to write code on one platform and compile it for another without signi icant modi ications.
* Open Source: GCC is open-source software, released under the GNU General Public License (GPL). Its source code is freely available, allowing developers to study, modify, and redistribute it according to the terms of the license.
* Standards Compliance: GCC strives to comply with various language standards such as ANSI C, ISO C99, ISO C++, and others. It ensures that the compiled code adheres to the speci ied language standards, promoting code portability and interoperability.
* Extensibility: GCC supports plugins and has a modular architecture, allowing developers to extend its functionality. This extensibility enables the integration of custom tools, analysis frameworks, and language extensions into the compiler.

WORKING OF GCC COMPILER:

GCC is a toolchain used for compiling code, linking it with library dependencies, converting code to assembly, and creating executable iles. It adheres to the UNIX philosophy of employing simple tools for speci ic tasks.

* When GCC processes a source code ile, it begins by using a preprocessor to handle header iles and remove comments. Then, it tokenizes the code, expands macros, and identi ies compile-time issues before preparing it for compilation.
* The code is then passed to the compiler, which constructs syntax trees for program objects and control low and generates assembly code based on these trees.
* The assembler converts the generated assembly code into the binary executable format of the system.
* Finally, the linker integrates references to external libraries, if necessary, producing a inished executable that can run on the target system.

SIGNIFICANCE OF GCC COMPILER:

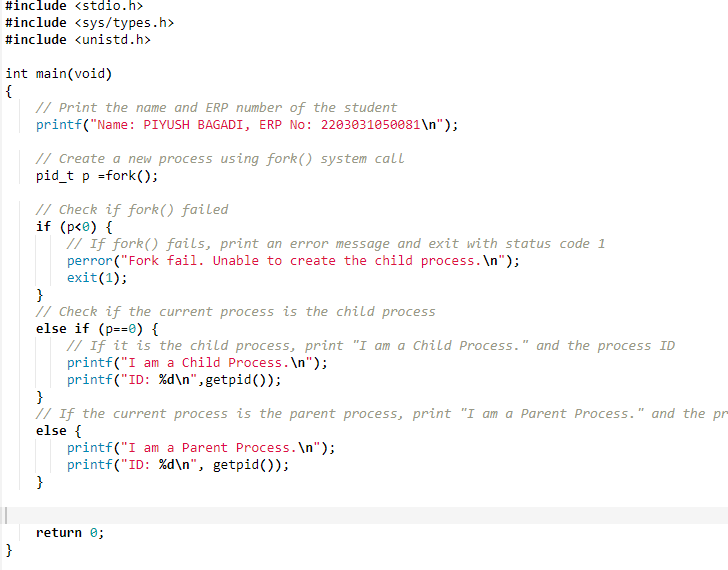
* Industry Standard: GCC is widely regarded as an industry-standard compiler suite for various programming languages. Its reliability, performance, and extensive feature set have made it a preferred choice for both academic and commercial software development projects.
* Ecosystem Support: The GCC compiler ecosystem includes a vast array of libraries, development tools, and documentation, fostering a supportive environment for software developers. This ecosystem enhances productivity and accelerates the development process.
* Community Collaboration: Being an open-source project, GCC bene its from the contributions and collaboration of a diverse community of developers worldwide. This collaborative effort ensures continuous improvement, bug ixes, and adaptation to emerging technologies and standards.
* Empowerment of Developers: GCC empowers developers by providing them with a powerful and lexible toolchain for software development. Its rich feature set, combined with extensive documentation and community support enables developers to create high-quality, ef icient, and portable software solutions.

SYNTAX:

gcc [-c|-S|-E] [-std=standard]

PROGRAM TO CREATE A CHILD PROCESS:

Code:



Output:

A screenshot of a computer program

Description automatically generated

**Practical – 9**

**AIM: Implementation of FCFS &Round Robin**

**Algorithm.**

* // TO CALCULATE AVERAGE WAITING TIME USING FIRST COME FIRST SERVE SCHEDULING
* **CODE :**

1. #include <stdio.h>
2. **int** main()
3. {
4. **int** p[10], at[10], bt[10], ct[10], tat[10], wt[10], i, j, temp = 0, n;
5. **float** awt = 0, atat = 0;
6. printf("ENTER THE NO. OF PROCESSES:");
7. scanf("%d", &n);
8. printf("ENTER %d
9. PROCESS:", n); 11 **for** (i =
10. 0; i < n; i++)
11. {
12. scanf("%d",
13. &p[i]); 14 }
14. 15 printf("ENTER %d ARRIVAL TIME:", n); 16 **for** (i = 0; i < n; i++)
15. {
16. scanf("%d",
17. &at[i]); 19 }
18. 20 printf("ENTER %d BURST
19. TIME:", n); 21 **for** (i = 0; i < n; i++)
20. {
21. scanf("%d",
22. &bt[i]); 24 }
23. 25 // sorting at,bt, and process according to at 26 **for** (i = 0; i < n; i++)
24. {
25. **for** (j = 0; j < (n - i); j++)
26. {
27. **if** (at[j] > at[j + 1])
28. {
29. temp = p[j + 1]; 33 p[j + 1] = p[j];
30. p[j] = temp;
31. temp = at[j + 1];
32. at[j + 1] = at[j];
33. at[j] = temp;
34. temp = bt[j + 1];
35. bt[j + 1] = bt[j];
36. bt[j] = temp;
37. }
38. }
39. }
40. /\* calculating 1st ct \*/ 45 ct[0] = at[0] + bt[0];
41. 46 /\* calculating 2 to n ct \*/ 47 **for** (i = 1; i < n; i++)
42. {
43. // when proess is ideal in between i and i+1
44. temp = 0;
45. **if** (ct[i - 1] < at[i])
46. {
47. temp = at[i] -
48. ct[i - 1]; 54 }
49. 55 ct[i] = ct[i - 1] + bt[i]
50. + temp; 56 }
51. /\* calculating tat and wt \*/
52. printf("\nP\t A.T\t B.T\t C.T\t
53. TAT\t WT"); 59 **for** (i = 0; i < n; i++)
54. {
55. tat[i] = ct[i] - at[i];
56. wt[i] = tat[i] - bt[i];
57. atat += tat[i]; 64 awt +=
58. wt[i]; 65 } 66 atat = atat / n;
59. awt = awt / n;
60. **for** (i = 0; i < n; i++)
61. {
62. printf("\nP%d\t %d\t %d\t %d \t %d \t %d", p[i], at[i], bt[i], ct[i],
63. tat[i], wt[i]);
64. }
65. printf("\n\nAVERAGE TURN AROUND TIME IS : %f", atat);
66. 74
67. printf("\nAVERAGE WAITING TIME IS : %f", awt);
68. **return** 0;
69. }

A screenshot of a computer screen

Description automatically generated

* // TO CALCULATE AVERAGE WAITING TIME USING ROUND ROBIN SCHEDULING
* CODE:

1. #include<stdio.h>
2. #include<conio.h>
3. **void** main() 7 {
4. // initlialize the variable name
5. **int** i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
6. **float** avg\_wt, avg\_tat;
7. printf("TOTAL NO. OF PROCESSES: ");
8. scanf("%d", &NOP);
9. y = NOP; // Assign the number of process to variable y 14
10. // Use loop to enter the details of the process like Arrival ime and the Burst Time
11. **for**(i=0; i<NOP; i++)
12. {
13. printf("\nENTER THE ARRIVAL TIME AND BURST TIME : [%d]\n", i+1);
14. printf("ARRIVAL TIME IS : "); // Accept arrival time
15. scanf("%d", &at[i]);
16. printf("\nBURST TIME IS : "); // Accept the Burst time
17. scanf("%d", &bt[i]);
18. temp[i] = bt[i]; // store the burst time in temp array
19. }
20. // Accept the Time qunat
21. printf("ENTER QUANTUM TIME(MAX. TIME) : ");
22. scanf("%d", &quant);
23. // Display the process No, burst time, Turn Around Time and the waiting time
24. printf("\n P \t\t BT \t\t TAT \t\t WT ");
25. **for**(sum=0, i = 0; y!=0; )
26. {
27. **if**(temp[i] <= quant && temp[i] > 0) // define the conditions
28. {
29. sum = sum + temp[i];
30. temp[i] = 0;
31. count=1;
32. }
33. **else if**(temp[i] > 0)
34. {
35. temp[i] = temp[i] - quant;
36. sum = sum + quant;
37. **if**(temp[i]==0 && count==1)
38. {
39. y--; //decrement the process no.
40. printf("\nP[%d]\t\t %d\t\t %d\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i ]);
41. wt = wt+sum-at[i]-bt[i];
42. tat = tat+sum-at[i];
43. count =0;
44. **if**(i==NOP-1)
45. {
46. i=0;
47. }
48. **else if**(at[i+1]<=sum)
49. {
50. i++;
51. }
52. **else** {
53. i=0;
54. }
55. }
56. // represents the average waiting time
57. avg\_wt = wt \* 1.0/NOP;
58. printf("\n\nAVERAGE WAITING TIME : \t%f", avg\_wt);
59. getch();
60. }

A computer screen with white text

Description automatically generated

**Practical – 10**

# AIM: Implementation of Banker’s Algorithm

* Banker's Algorithm
* CODE :

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k;

n = 5; // Number of processes

m = 3; // Number of resources

int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } }; // P4

int avail[3] = { 3, 3, 2 }; // Available Resources

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][m];

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

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

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

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

if (f[i] == 0) {

int flag = 0;

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

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

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

{

if(f[i]==0)

{

flag=0;

printf("The following system is not safe");

break;

}

}

if(flag==1)

{

printf("Following is the SAFE Sequence\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

}

return (0);

}

A black background with white text

Description automatically generated