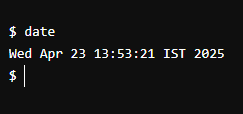
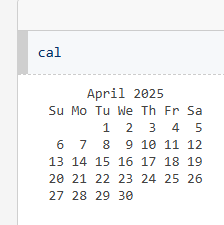
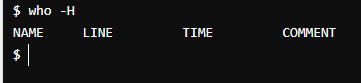
**Practical – 01**

**General UNIX Commands-**

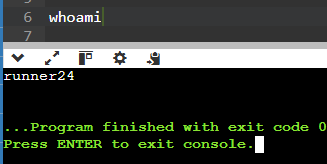
**1.Date Command-** It writes the current date and time to standard output if called with no flags or with a flag list that begins with a + (plus sign).

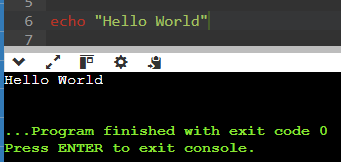


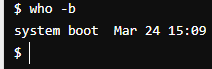
**2.Calendar command-** The cal command is used to display a calendar, either for the current month, a specific month, or a specific year, on a command-line interface.

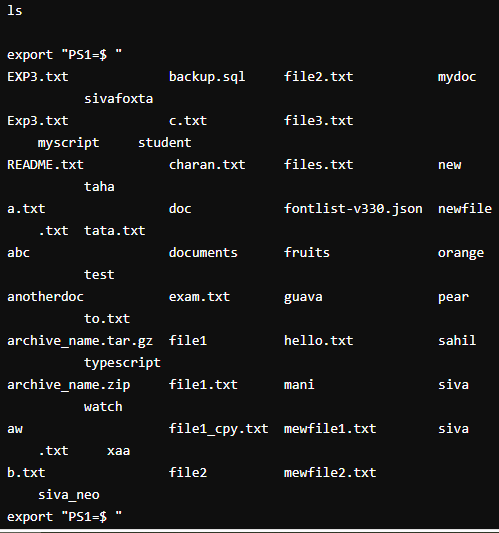


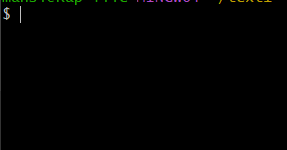
**3.who -H command-** It is a tool that Displays a header (title).

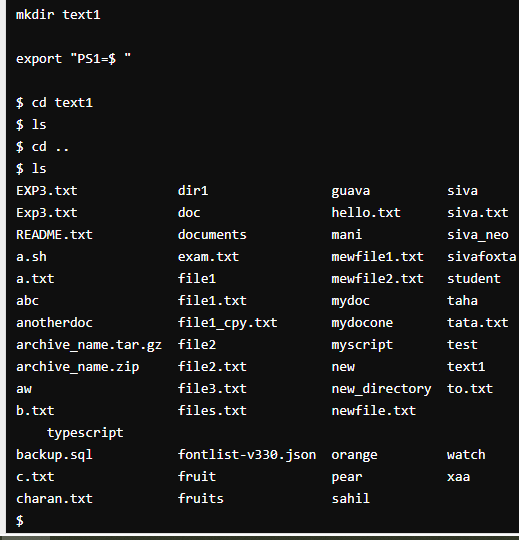
**4.’whoami’ command-** The whoami command displays the username of the currently logged-in user.

**5.’echo’ command-** The echo command is like a messenger that delivers your words to the terminal like printing text, variables and special characters to the standard output.

**6.who -b command-** Indicates the most recent system startup time and date.

**7.List command(ls)-** The ls command is used to list files. "ls" on its own lists all files in the current directory except for hidden files.

**8.Clear command-** The clear command in Linux is primarily used to clear the terminal screen, making it a clean slate for new output.

**9.Manipulation commands:-**

**Mkdir-** creates new directory

**cd-** changes directory

**cd ..** -used to move out of the current directory.

**Rmdir-** used to delete a directory

**Practical - 02**

**Implementation of CPU scheduling algorithm – FCFS**

#include<stdio.h>

void findWaitingTime(int processes[], int n,

int bt[], int wt[])

{

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

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

wt[i] = bt[i-1] + wt[i-1] ;

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n,

int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = bt[i] + wt[i];

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

printf("Processes Burst time Waiting time Turn around time\n");

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

printf(" %d ",(i+1));

printf(" %d ", bt[i] );

printf(" %d",wt[i] );

printf(" %d\n",tat[i] );

}

float s=(float)total\_wt / (float)n;

float t=(float)total\_tat / (float)n;

printf("Average waiting time = %f",s);

printf("\n");

printf("Average turn around time = %f ",t);

}

// Driver code

int main()

{

//process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

//Burst time of all processes

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

**Implementation of CPU scheduling algorithm – SJF**

#include <stdio.h>

int main()

{

// Matrix for storing Process Id, Burst

// Time, Average Waiting Time & Average

// Turn Around Time.

int A[100][4];

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of process: ");

scanf("%d", &n);

printf("Enter Burst Time:\n");

// User Input Burst Time and alloting Process Id.

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

printf("P%d: ", i + 1);

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

// Sorting process according to their Burst Time.

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

index = i;

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

if (A[j][1] < A[index][1])

index = j;

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

// Calculation of Waiting Times

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

A[i][2] = 0;

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

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

printf("P BT WT TAT\n");

// Calculation of Turn Around Time and printing the

// data.

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

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],

A[i][1], A[i][2], A[i][3]);

}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt);

printf("\nAverage Turnaround Time= %f", avg\_tat);

}

**Practical - 03**

**Implementation of CPU scheduling algorithm– Priority(Non-Preemptive)**

#include <stdio.h>

#include <stdlib.h>

// Structure to represent a process

struct Process {

int pid; // Process ID

int bt; // CPU Burst time required

int priority; // Priority of this process

};

// Comparison function to sort processes by priority (descending)

int compare(const void\* a, const void\* b) {

struct Process\* p1 = (struct Process\*)a;

struct Process\* p2 = (struct Process\*)b;

return p2->priority - p1->priority;

}

// Function to find the waiting time for all processes

void findWaitingTime(struct Process proc[], int n, int wt[]) {

wt[0] = 0;

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

wt[i] = proc[i - 1].bt + wt[i - 1];

}

// Function to calculate turnaround time

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {

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

tat[i] = proc[i].bt + wt[i];

}

// Function to calculate and display average times

void findavgTime(struct Process proc[], int n) {

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(proc, n, wt);

findTurnAroundTime(proc, n, wt, tat);

printf("\nProcesses Burst time Waiting time Turn around time\n");

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

total\_wt += wt[i];

total\_tat += tat[i];

printf(" %d\t\t%d\t %d\t\t %d\n", proc[i].pid, proc[i].bt, wt[i], tat[i]);

}

printf("\nAverage waiting time = %.2f", (float)total\_wt / n);

printf("\nAverage turn around time = %.2f\n", (float)total\_tat / n);

}

void priorityScheduling(struct Process proc[], int n) {

qsort(proc, n, sizeof(struct Process), compare);

printf("Order in which processes get executed:\n");

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

printf("%d ", proc[i].pid);

printf("\n");

findavgTime(proc, n);

}

int main() {

struct Process proc[] = { {1, 10, 2}, {2, 5, 0}, {3, 8, 1} };

int n = sizeof(proc) / sizeof(proc[0]);

priorityScheduling(proc, n);

return 0;

}

**Practical - 04**

**Implementation of Resource Allocation Graph:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

// Node types

#define PROCESS 0

#define RESOURCE 1

typedef struct {

char name[10];

int type; // 0: process, 1: resource

} Node;

int numNodes = 0;

Node nodes[MAX];

int adjMatrix[MAX][MAX]; // Adjacency matrix for graph

// Find index of a node by name

int findIndex(char \*name) {

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

if (strcmp(nodes[i].name, name) == 0) return i;

}

return -1;

}

// Add a node if it doesn't exist

int addNode(char \*name, int type) {

int idx = findIndex(name);

if (idx != -1) return idx;

strcpy(nodes[numNodes].name, name);

nodes[numNodes].type = type;

return numNodes++;

}

// Add an edge: from → to

void addEdge(char \*from, char \*to) {

int u = findIndex(from);

int v = findIndex(to);

if (u == -1 || v == -1) {

printf("Error: Invalid node name.\n");

return;

}

adjMatrix[u][v] = 1;

}

// Display the adjacency matrix

void displayGraph() {

printf("\nResource Allocation Graph (Adjacency Matrix):\n");

printf(" ");

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

printf("%4s", nodes[i].name);

}

printf("\n");

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

printf("%4s", nodes[i].name);

for (int j = 0; j < numNodes; j++) {

printf("%4d", adjMatrix[i][j]);

}

printf("\n");

}

}

int main() {

// Sample usage

addNode("P1", PROCESS);

addNode("P2", PROCESS);

addNode("R1", RESOURCE);

addNode("R2", RESOURCE);

addEdge("P1", "R1"); // Request edge

addEdge("R1", "P2"); // Assignment edge

addEdge("P2", "R2"); // Request edge

addEdge("R2", "P1"); // Assignment edge (circular dependency)

displayGraph();

return 0;

}