Operating Systems LAB

PAPER CODE	**:**	**CIC-353**
Faculty Name	:	Ms. Kavita Saxena
Name	:	Amit Singhal
Enrollment No.	:	11614802722
Branch	:	Computer Science & Engg.
Semester | Group	:	5C6

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

PSP Area, Plot No. 1, Sector-22, Rohini, Delhi-110086

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

Computer Science & Engineering Department

VISION

"To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values."

MISSION

To foster an open, multidisciplinary and highly collaborative research environment for producing world-class engineers capable of providing innovative solutions to real-life problems and fulfil societal needs.

LAB Assessment Sheet

S.No.	Experiment Name	M A R K S					Total Marks	Date of Perf.	Date of Check.	Signature
		R1	R2	R3	R4	R5				

S.No.	Experiment Name	M	A	R	K	S	Total Marks	Date of Perf.	Date of Check.	Signature
		R1	R2	R3	R4	R5				

Lab Exercise - 1

❖ AIM :: Introduction to Linux & vi-Editor

1. Introduction to Linux

- **What is Linux?**: Linux is a powerful and versatile open-source operating system based on the Unix architecture. It was created by Linus Torvalds in 1991 and has since grown into a widely-used platform for both personal and professional computing.
- **Open Source Nature**: One of the defining characteristics of Linux is that its source code is freely available for anyone to view, modify, and distribute. This has led to a collaborative environment where developers worldwide contribute to its development.
- **Kernel and Distributions**: Linux is composed of a kernel, which is the core component of the OS, and various distributions (distros) that bundle the kernel with software and package management systems. Popular distributions include Ubuntu, Fedora, Debian, and CentOS.
- **Linux in Different Environments**: Linux is used in a variety of environments, including desktops, servers, mobile devices, and embedded systems. Its flexibility allows it to run on a wide range of hardware, from supercomputers to small IoT devices.

2. Overview of the `vi` Editor

The `vi` (Visual Editor) is a powerful text editor available on almost all Unix-like operating systems, including Linux. It's known for its efficiency and versatility, particularly in environments where only a terminal interface is available. Here is a detailed look at the `vi` editor and its commands, presented in informative points.

---

1. Basics of `vi` Editor

- **Launching** `vi`: To start `vi`, type `vi filename` in the terminal. If `filename` does not exist, `vi` will create it.
- **Modes in** `vi`:
  - **Normal Mode**: The default mode where you can navigate and manipulate text.
  - **Insert Mode**: Used for inserting text. Enter by pressing `i`, `a`, or `o`.
  - **Command Mode**: Enter by typing `:` in Normal Mode for commands like saving, quitting, etc.
  - **Visual Mode**: Used to highlight and manipulate blocks of text.

2. Basic Commands for Running a C File

To work with C files in the `vi` editor, you only need a few basic commands to edit, save, and compile the file. Here's a simplified guide:

- **Open a File**: `vi filename.c`
  - Launches `vi` and opens the file named `filename.c`. If it doesn't exist, `vi` will create it.
- **Insert Mode**:
  - `i`: Enter Insert Mode before the cursor position.
  - `I`: Enter Insert Mode at the beginning of the line.
  - `a`: Enter Insert Mode after the cursor position.
  - `A`: Enter Insert Mode at the end of the line.
  - `o`: Open a new line below the current line and enter Insert Mode.
  - `O`: Open a new line above the current line and enter Insert Mode.
- **Save and Exit**:
  - `:w`: Save the file without exiting.
  - `:w filename`: Save the file with a new name.
  - `:q`: Quit `vi` without saving.
  - `:wq` **or** `ZZ`: Save the file and quit `vi`.
  - `:q!`: Quit without saving changes.

Implementation

Writing and Running a basic "Hello, World!" program in C using the terminal on a Linux system.

```
1. cd ~/project

2. vi hello.c



/* Save and Exit vi:

    • Press Esc to exit Insert Mode.
    • Type :wq and press Enter to save the file and quit vi.

*/


3. gcc hello.c -o hello

4. ./hello
```

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}

~
~
~
~
:wq
```

```
amit@Toshiba-Satellite-C850:~$ cd Downloads/
amit@Toshiba-Satellite-C850:~/Downloads$ vi hello.c
amit@Toshiba-Satellite-C850:~/Downloads$ gcc hello.c -o hello
amit@Toshiba-Satellite-C850:~/Downloads$ ./hello
Hello, World!
amit@Toshiba-Satellite-C850:~/Downloads$
```

Lab Exercise – 2.1

- AIM :: WAP in C to implement basic operations in different functions on Linux using vi-Editor

Source_Code ::

```
#include <stdio.h>

// Function to find the greatest number among three numbers
int findGreatest(int a, int b, int c)
{
    if (a > b && a > c) {
        return a;
    } else if (b > c) {
        return b;
    } else {
        return c;
    }
}
// Function to check if a number is even or odd
void evenOdd(int num)
{
    if (num % 2 == 0) {
        printf("%d is Even\n", num);
    } else {
        printf("%d is Odd\n", num);
    }
}
```

```
// Function to check if a number is prime
void checkPrime(int num)
{
    int i, flag = 0;
    if (num <= 1) {
        printf("%d is not a Prime number\n", num);
        return;
    }
    for (i = 2; i <= num / 2; ++i) {
        if (num % i == 0) {
            flag = 1;
            break;
        }
    }
    if (flag == 0) {
        printf("%d is a Prime number\n", num);
    } else {
        printf("%d is not a Prime number\n", num);
    }
}
// Function to calculate the average of three numbers
double calculateAverage(int a, int b, int c) { return (a + b + c) / 3.0; }
int main()
{
    printf("\n5C6 - Amit Singhal (11614802722)\n");
    int num1, num2, num3;
    int choice;
    printf("\nChoose an operation:\n");
    printf("1. Find Greatest of Three Numbers\n");
    printf("2. Check Even or Odd\n");
```

```
printf("3. Check Prime Number\n");
printf("4. Calculate Average of Three Numbers\n");
printf("5. Exit\n");
while (1) {
    printf("\nEnter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
    case 1:
        printf("\nEnter three numbers: ");
        scanf("%d %d %d", &num1, &num2, &num3);
        printf("Greatest Number: %d\n", findGreatest(num1, num2, num3));
        break;
    case 2:
        printf("\nEnter a number: ");
        scanf("%d", &num1);
        evenOdd(num1);
        break;
    case 3:
        printf("\nEnter a number: ");
        scanf("%d", &num1);
        checkPrime(num1);
        break;
    case 4:
        printf("\nEnter three numbers: ");
        scanf("%d %d %d", &num1, &num2, &num3);
        printf("Average: %.2f\n", calculateAverage(num1, num2, num3));
        break;
    case 5:
        printf("\n");
        return 0;
    default:
```

```
        printf("\nInvalid choice! Please choose again.\n");
    }
  }
  return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~$ cd Desktop/Code/
amit@Toshiba-Satellite-C850:~/Desktop/Code$ vi basic_operations.c
amit@Toshiba-Satellite-C850:~/Desktop/Code$ gcc basic_operations.c -o basic_operations
amit@Toshiba-Satellite-C850:~/Desktop/Code$ ./basic_operations

5C6 - Amit Singhal (11614802722)

Choose an operation:
1. Find Greatest of Three Numbers
2. Check Even or Odd
3. Check Prime Number
4. Calculate Average of Three Numbers
5. Exit

Enter your choice: 1

Enter three numbers: 105 116 122
Greatest Number: 122

Enter your choice: 2

Enter a number: 13345
13345 is Odd

Enter your choice: 3

Enter a number: 5456527
5456527 is not a Prime number

Enter your choice: 4

Enter three numbers: 2234 4523 4355
Average: 3704.00

Enter your choice: 5

amit@Toshiba-Satellite-C850:~/Desktop/Code$
```

Lab Exercise – 2.2

- AIM :: WAP in C to implement basic operations in different functions on Linux using vi-Editor.

Source_Code ::

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

// Function to print the Fibonacci series up to n terms
void fibonacci(int n)
{
   int first = 0, second = 1, next;

   if (n <= 0) {
      printf("Please enter a positive integer.\n");
      return;
   }

   printf("Fibonacci Series: ");
   for (int i = 1; i <= n; i++) {
      if (i == 1) {
         printf("%d ", first);
         continue;
      }
      if (i == 2) {
         printf("%d ", second);
```

```
            continue;
        }
        next = first + second;
        first = second;
        second = next;
        printf("%d ", next);
    }
    printf("\n");
}

// Function to calculate the factorial of a number
int factorial(int n)
{
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}

// Function to calculate the sum of digits of a number
int digitsSum(int num)
{
    int sum = 0;
    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

// Function to check if a string is a palindrome
bool isPalindrome(char str[])
```

```
{
    int length = strlen(str);
    int start = 0;
    int end = length - 1;

    while (start < end) {
        if (str[start] != str[end]) {
            return false;
        }
        start++;
        end--;
    }
    return true;
}

// Function to count the occurrences of a character in a string
int countChar(char* str, char ch)
{
    int count = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            count++;
        }
    }
    return count;
}

int main()
{

    int choice, num1, num2, num3;
    char str[100], ch;
```

```
printf("\n5C6 - Amit Singhal (11614802722)\n");

// Display the menu
printf("\nMenu:\n");
printf("1. Print Fibonacci Series\n");
printf("2. Calculate Factorial\n");
printf("3. Calculate Sum of Digits\n");
printf("4. Check Palindrome\n");
printf("5. Count Character Occurrences\n");
printf("6. Exit\n");

while (1) {
   printf("\nEnter your choice (1-6): ");
   scanf("%d", &choice);

   switch (choice) {

   case 1:
      printf("\nEnter the number of terms for Fibonacci series: ");
      scanf("%d", &num1);
      fibonacci(num1);
      break;
   case 2:
      printf("\nEnter a number to calculate its factorial: ");
      scanf("%d", &num1);
      printf("Factorial: %d\n", factorial(num1));
      break;
   case 3:
      printf("\nEnter a number to calculate the sum of its digits: ");
      scanf("%d", &num1);
      printf("Sum of Digits: %d\n", digitsSum(num1));
```

```
            break;
        case 4:
            printf("Enter a string to check if it is a palindrome: ");
            scanf("%s", str);

            if (isPalindrome(str)) {
                printf("%s is a Palindrome\n", str);
            } else {
                printf("%s is not a Palindrome\n", str);
            }
            break;
        case 5:
            printf("\nEnter a string: ");
            scanf("%s", str);
            printf("Enter a character to count its occurrences: ");
            scanf(" %c", &ch);
            printf("Count of '%c': %d\n", ch, countChar(str, ch));
            break;
        case 6:
            printf("\nExiting the program. Have a great day!\n");
            return 0;
        default:
            printf(
                "\nInvalid choice! Please select a number between 1 and 6.\n");
        }
    }

    return 0;
}
```

Output ::

```
amit@Toshiba-Satellite-C850:~/Downloads/OS$ vi basic_operations_2.c

amit@Toshiba-Satellite-C850:~/Downloads/OS$ gcc basic_operations_2.c -o prg_2

amit@Toshiba-Satellite-C850:~/Downloads/OS$ ./prg_2

5C6 - Amit Singhal (11614802722)

Menu:
1. Print Fibonacci Series
2. Calculate Factorial
3. Calculate Sum of Digits
4. Check Palindrome
5. Count Character Occurrences
6. Exit

Enter your choice (1-6): 1

Enter the number of terms for Fibonacci series: 9
Fibonacci Series: 0 1 1 2 3 5 8 13 21

Enter your choice (1-6): 12

Invalid choice! Please select a number between 1 and 6.
Enter your choice (1-6): 2

Enter a number to calculate its factorial: 12
Factorial: 479001600

Enter your choice (1-6): 3

Enter a number to calculate the sum of its digits: 35544355
Sum of Digits: 34

Enter your choice (1-6): 4
Enter a string to check if it is a palindrome: madam
madam is a Palindrome

Enter your choice (1-6): 5

Enter a string: helloworld
Enter a character to count its occurrences: l
Count of 'l': 3

Enter your choice (1-6): 6

Exiting the program. Have a great day!
amit@Toshiba-Satellite-C850:~/Downloads/OS$
```

Lab Exercise – 3

- AIM :: WAP in C to implement CPU scheduling for `first come first serve` (fcfs).

Source_Code ::

```
#include <stdio.h>

typedef struct
{
  int pid;        // Process ID
  int arrival;    // Arrival time
  int burst;      // Burst time
  int completion; // Completion time
  int waiting;    // Waiting time
  int turnaround; // Turnaround time
} Process;

// Function to sort processes by arrival time
void sortByArrival(Process *p, int n)
{
  for (int i = 0; i < n - 1; i++)
  {
    for (int j = 0; j < n - i - 1; j++)
    {
      if (p[j].arrival > p[j + 1].arrival)
```

```
      {
        Process temp = p[j];
        p[j] = p[j + 1];
        p[j + 1] = temp;
      }
    }
  }
}

// Main FCFS logic
void fcfsScheduling(Process *p, int n)
{
  int time = 0;

  for (int i = 0; i < n; i++)
  {
    if (time < p[i].arrival)
      time = p[i].arrival; // Set time to the process arrival time if idle

    time += p[i].burst;
    p[i].completion = time;
    p[i].turnaround = p[i].completion - p[i].arrival;
    p[i].waiting = p[i].turnaround - p[i].burst;
  }
}

// Function to display the Gantt chart with idle times
void displayGanttChart(Process *p, int n)
{
```

```
  int currentTime = p[0].arrival; // Start from the first process arrival time
  printf("Gantt Chart:\n");

  // Print initial time
  printf("%d", currentTime);

  for (int i = 0; i < n; i++)
  {
    if (currentTime < p[i].arrival)
    {
      // Display idle time
      printf(" -- XX -- %d", p[i].arrival);
      currentTime = p[i].arrival; // Update current time to the arrival of the next process
    }
    // Display the process and its completion time
    printf(" -- P%d -- %d", p[i].pid, p[i].completion);
    currentTime = p[i].completion; // Update current time to the completion of the current process
  }

  printf("\n\n");
}

// Function to calculate and display average times
void calculateAverages(Process *p, int n)
{
  float totalTurnaround = 0, totalWaiting = 0;

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

```
  {
    totalTurnaround += p[i].turnaround;
    totalWaiting += p[i].waiting;
  }

  printf("\nAverage Turnaround Time: %.2f\n", totalTurnaround / n);
  printf("Average Waiting Time: %.2f\n", totalWaiting / n);
}

// Function to display process information
void displayResults(Process *p, int n) {
  printf("PID\tArrival\t Burst\t Completion\tTurnaround\tWaiting\n");
  for (int i = 0; i < n; i++) {
    printf("%d\t%d\t %d\t %d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst,
           p[i].completion, p[i].turnaround, p[i].waiting);
  }
}

int main() {
  int n;
  printf("\n5C6 - Amit Singhal (11614802722)\n");
  printf("\nEnter number of processes: ");
  scanf("%d", &n);
  Process p[n];

  for (int i = 0; i < n; i++) {
    printf("\nEnter Arrival Time and Burst Time for Process %d: ", i + 1);
    p[i].pid = i + 1;
    scanf("%d%d", &p[i].arrival, &p[i].burst);
```

```
    p[i].completion = 0; // Initially, no process is completed
  }
  printf("\n");

  sortByArrival(p, n);
  fcfsScheduling(p, n);
  displayGanttChart(p, n);
  displayResults(p, n);
  calculateAverages(p, n);

  printf("\n");

  return 0;
}
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ vi prg_3_fcfs.c
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ gcc prg_3_fcfs.c
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ ./a.out

5C6 - Amit Singhal (11614802722)

Enter number of processes: 4

Enter Arrival Time and Burst Time for Process 1: 0 2

Enter Arrival Time and Burst Time for Process 2: 1 2

Enter Arrival Time and Burst Time for Process 3: 5 3

Enter Arrival Time and Burst Time for Process 4: 6 4

Gantt Chart:
0 -- P1 -- 2 -- P2 -- 4 -- XX -- 5 -- P3 -- 8 -- P4 -- 12

PID     Arrival  Burst   Completion     Turnaround      Waiting
1       0        2       2              2               0
2       1        2       4              3               1
3       5        3       8              3               0
4       6        4       12             6               2

Average Turnaround Time: 3.50
Average Waiting Time: 0.75
```

Lab Exercise – 4

- AIM :: WAP in C to implement CPU scheduling for `shortest job first` (sjf).

Source_Code ::

```
#include <stdio.h>


typedef struct
{
  int pid;        // Process ID
  int arrival;    // Arrival time
  int burst;      // Burst time
  int completion; // Completion time
  int waiting;    // Waiting time
  int turnaround; // Turnaround time
} Process;


// Function to sort processes by arrival time, and by burst time in case of tie
void sortByArrival(Process *p, int n)
{
  for (int i = 0; i < n - 1; i++)
  {
    for (int j = 0; j < n - i - 1; j++)
    {
```

```
      if (p[j].arrival > p[j + 1].arrival ||
          (p[j].arrival == p[j + 1].arrival && p[j].burst > p[j + 1].burst))
      {
        Process temp = p[j];
        p[j] = p[j + 1];
        p[j + 1] = temp;
      }
    }
  }
}

// Main SJF logic
void sjfScheduling(Process *p, int n)
{
  int time = 0, completed = 0, minIndex;

  while (completed < n)
  {
    minIndex = -1;

    // Find process with min burst time from the pool of arrived processes
    for (int i = 0; i < n; i++)
    {
      if (p[i].arrival <= time && p[i].completion == 0)
      {
        if (minIndex == -1 || p[i].burst < p[minIndex].burst)
```

```
        {
          minIndex = i;
        }
      }
    }

    if (minIndex != -1)
    {
      if (time < p[minIndex].arrival)
        time = p[minIndex].arrival; // Set time to the process arrival time if idle

      time += p[minIndex].burst;
      p[minIndex].completion = time;
      p[minIndex].turnaround = p[minIndex].completion - p[minIndex].arrival;
      p[minIndex].waiting = p[minIndex].turnaround - p[minIndex].burst;
      completed++;
    }
    else
    {
      time++;
    }
  }
}

// Function to display the Gantt chart
void displayGanttChart(Process *p, int n)
```

```
{
  int startTime = p[0].arrival;
  printf("Gantt Chart:\n%d", startTime);

  for (int i = 0; i < n; i++)
  {
    printf(" -- P%d -- %d", p[i].pid, p[i].completion);
  }

  printf("\n\n");
}

// Function to calculate and display average times
void calculateAverages(Process *p, int n)
{
  float totalTurnaround = 0, totalWaiting = 0;

  for (int i = 0; i < n; i++)
  {
    totalTurnaround += p[i].turnaround;
    totalWaiting += p[i].waiting;
  }

  printf("\nAverage Turnaround Time: %.2f\n", totalTurnaround / n);
  printf("Average Waiting Time: %.2f\n", totalWaiting / n);
}
```

```
// Function to display process information
void displayResults(Process *p, int n)
{
  printf("PID\tArrival\t Burst\t Completion\tTurnaround\tWaiting\n");
  for (int i = 0; i < n; i++)
  {
    printf("%d\t%d\t %d\t %d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst,
          p[i].completion, p[i].turnaround, p[i].waiting);
  }
}

int main()
{
  int n;
  printf("\n5C6 - Amit Singhal (11614802722)\n");
  printf("\nEnter number of processes: ");
  scanf("%d", &n);
  Process p[n];

  for (int i = 0; i < n; i++) {
    printf("\nEnter Arrival Time and Burst Time for Process %d: ", i + 1);
    p[i].pid = i + 1;
    scanf("%d%d", &p[i].arrival, &p[i].burst);
    p[i].completion = 0; // Initially, no process is completed
  }
```

```
    printf("\n");

    sortByArrival(p, n);
    sjfScheduling(p, n);
    displayGanttChart(p, n);
    displayResults(p, n);
    calculateAverages(p, n);

    printf("\n");

    return 0;
}
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ vi prg_4_sjf.c
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ gcc prg_4_sjf.c
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ ./a.out

5C6 - Amit Singhal (11614802722)

Enter number of processes: 4

Enter Arrival Time and Burst Time for Process 1: 1 3

Enter Arrival Time and Burst Time for Process 2: 2 4

Enter Arrival Time and Burst Time for Process 3: 1 2

Enter Arrival Time and Burst Time for Process 4: 4 4

Gantt Chart:
1 -- P3 -- 3 -- P1 -- 6 -- P2 -- 10 -- P4 -- 14

PID     Arrival  Burst   Completion     Turnaround      Waiting
3       1        2       3              2               0
1       1        3       6              5               2
2       2        4       10             8               4
4       4        4       14             10              6

Average Turnaround Time: 6.25
Average Waiting Time: 3.00
```

2) Preemptive Mode

Source_Code ::

```
#include <stdio.h>


typedef struct
{
  int pid;        // Process ID
  int arrival;    // Arrival time
  int burst;      // Burst time
  int remaining;  // Remaining burst time (for preemption)
  int completion; // Completion time
  int waiting;    // Waiting time
  int turnaround; // Turnaround time
} Process;


// Function to find the process with the shortest remaining time at a given time
int findShortestRemaining(Process *p, int n, int time)
{
  int min_index = -1;
  int min_remaining = 99999;


  for (int i = 0; i < n; i++)
  {
    if (p[i].arrival <= time && p[i].remaining > 0 && p[i].remaining < min_remaining)
```

```
    {
      min_remaining = p[i].remaining;
      min_index = i;
    }
  }

  return min_index;
}

void sjfPreemptive(Process *p, int n)
{
  int time = 0;      // Current time
  int completed = 0; // Number of completed processes
  int gantt[100];    // Gantt chart sequence
  int gantt_index = 0;

  while (completed < n)
  {
    int shortest_job = findShortestRemaining(p, n, time);

    if (shortest_job == -1)
    {
      // If no process is ready, increment the time (idle)
      time++;
      gantt[gantt_index++] = -1;
    }
    else
    {
```

```
    // Execute the process for 1 unit of time
    p[shortest_job].remaining--;
    gantt[gantt_index++] = shortest_job;

    time++;

    // If the process is finished
    if (p[shortest_job].remaining == 0)
    {
      p[shortest_job].completion = time;
      p[shortest_job].turnaround = p[shortest_job].completion - p[shortest_job].arrival;
      p[shortest_job].waiting = p[shortest_job].turnaround - p[shortest_job].burst;
      completed++;
    }
  }
 }

 // Gantt chart display
 printf("\nGantt Chart:\n");
 printf("0"); // Start at time 0
 int current_time = 0;
 for (int i = 0; i < gantt_index; i++)
 {
  if (gantt[i] == -1)
  {
   printf(" -- XX -- %d", ++current_time); // Idle time
  }
```

```
    else
    {
      if (i == 0 || gantt[i] != gantt[i - 1])
      { // Only display if process changes
        printf(" -- P%d -- %d", p[gantt[i]].pid, ++current_time);
      }
      else
      {
        current_time++;
      }
    }
  }
  printf("\n");
}

// Function to display the process table
void displayResults(Process *p, int n)
{
  printf("\nPID\tArrival\t  Burst\t  Completion\tTurnaround\tWaiting\n");
  for (int i = 0; i < n; i++)
  {
    printf("%d\t%d\t  %d\t  %d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival,
p[i].burst,
          p[i].completion, p[i].turnaround, p[i].waiting);
  }
}

// Function to calculate and display average times
void calculateAverages(Process *p, int n)
```

```
{
  float total_waiting = 0, total_turnaround = 0;

  for (int i = 0; i < n; i++)
  {
    total_waiting += p[i].waiting;
    total_turnaround += p[i].turnaround;
  }

  printf("\nAverage Waiting Time: %.2f", total_waiting / n);
  printf("\nAverage Turnaround Time: %.2f\n", total_turnaround / n);
}

int main()
{
  int n;
  printf("\n5C6 - Amit Singhal (11614802722)\n");
  printf("\nEnter the number of processes: ");
  scanf("%d", &n);

  Process p[n];

  // Input the arrival and burst times for each process
  for (int i = 0; i < n; i++)
  {
    p[i].pid = i + 1;
    printf("\nEnter Arrival Time and Burst Time for Process %d: ", i + 1);
    scanf("%d%d", &p[i].arrival, &p[i].burst);
```

```
        p[i].remaining = p[i].burst; // Remaining burst time for preemption
        p[i].completion = 0;        // Initially no completion time
    }

    sjfPreemptive(p, n);
    displayResults(p, n);
    calculateAverages(p, n);

    return 0;
}
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ gcc prg_4.2_sjf.c
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./a.out

5C6 - Amit Singhal (11614802722)

Enter the number of processes: 4

Enter Arrival Time and Burst Time for Process 1: 0 7

Enter Arrival Time and Burst Time for Process 2: 2 4

Enter Arrival Time and Burst Time for Process 3: 4 1

Enter Arrival Time and Burst Time for Process 4: 5 4

Gantt Chart:
0 -- P1 -- 1 -- P2 -- 3 -- P3 -- 5 -- P2 -- 6 -- P4 -- 8 -- P1 -- 12

PID     Arrival   Burst   Completion    Turnaround      Waiting
1       0         7       16            16              9
2       2         4       7             5               1
3       4         1       5             1               0
4       5         4       11            6               2

Average Waiting Time: 3.00
Average Turnaround Time: 7.00
```

Lab Exercise – 5

AIM :: Introduction to Shell Scripting

Introduction

Shell scripting is a fundamental aspect of Unix-like operating systems (such as Linux and macOS) and serves as a bridge between users and the system kernel. A shell script is a sequence of commands written in a file, allowing users to automate tasks, run complex programs, and manipulate files and processes. The term "shell" refers to the command-line interpreter that facilitates interaction between the user and the operating system.

Shell scripts streamline routine system administration tasks, automate repetitive jobs, and enable the execution of multiple commands in sequence. This not only saves time but also reduces errors that could occur from manual execution. Shell scripts are often employed by system administrators, developers, and users to manage files, backups, network configurations, and more.

- **Purpose of Shell Scripting**

The primary purpose of shell scripting is automation. It enables users to create efficient workflows for repetitive tasks. For example, rather than executing several commands manually each time you need to back up data or clean a directory, a shell script can be used to automate these tasks. This reduces both time and effort and minimizes the chance of human error.

Another key purpose is system management. Shell scripts are used to configure servers, manage networks, and control system processes, making them an essential tool for system administrators. Moreover, shell scripting enhances task reproducibility, ensuring that procedures are consistently followed without variation.

Shell scripts also serve as a powerful tool for creating utilities and simple programs that can automate complex tasks. Developers and data scientists use them to preprocess data, compile code, or even manage and deploy software environments.

- ## How Shell Scripting Works

Shell scripting revolves around the use of commands that are interpreted by the shell, which can be thought of as a layer between the user and the operating system. The shell reads the script file line by line and executes each command in the order it appears. Shell scripts are typically written in plain text and can be created using any text editor. The most common shell interpreters are Bash (Bourne Again Shell), sh (Bourne Shell), and zsh (Z Shell).

Steps to Create and Execute a Shell Script

1. **Create a script file**: Use any text editor like `vim`, `nano`, or `gedit` to create a file with a `.sh` extension.
2. **Make the file executable**: After writing the script, you must grant it execution permissions. This can be done using the command:

   ```
   chmod +x script_name.sh
   ```

3. **Run the script**: Execute the script by typing:

   ```
   ./script_name.sh
   ```

When a script is executed, the shell runs each command within the script sequentially. The shell interpreter also supports variables, loops, conditional statements, and functions, making scripts flexible and powerful.

- ## Basic Shell Commands

Here are a few fundamental shell commands that form the basis of shell scripting:

1. **`echo`**: Prints output to the terminal.

   ```
   echo "Hello, World!"
   ```

2. **`ls`**: Lists the files and directories in the current directory.

   ```
   ls -l
   ```

3. **`cd`**: Changes the current working directory.

   ```
   cd /path/to/directory
   ```

4. **`pwd`**: Prints the current working directory.

   ```
   Pwd
   ```

5. **`cp`**: Copies files or directories.

```
cp source_file
destination_directory
```

6. **`mv`**: Moves or renames files and directories.

```
mv old_name new_name
```

7. **`rm`**: Removes files or directories.

```
rm file_name
```

8. **`cat`**: Displays the content of a file.

```
cat file_name
```

9. **`if` statements**: Used for conditional execution.

```
if [ condition ]; then
     # Commands
fi
```

10. **`for` loops**: Used for iterating over items.

```
for i in {1..5}; do
     echo "Number: $i"
done
```

Conclusion

Shell scripting is an essential skill for automating tasks in Unix-based systems, enabling users to execute sequences of commands, manage systems efficiently, and reduce manual effort. Its flexibility makes it ideal for a wide range of use cases, from simple file operations to complex system administration tasks.

Understanding basic commands and how to structure scripts empowers users to automate and streamline workflows, ultimately boosting productivity. Whether you're a developer, system administrator, or enthusiast, mastering shell scripting offers a powerful way to interact with and control your system more effectively.

Lab Exercise – 6

- AIM :: WAP in Shell Scripting to implement various Basic Operations

Source_Code ::

```
#!/bin/bash

# 1. Greatest of Three Numbers
echo "Program 1: Greatest of Three Numbers"
echo "Enter three numbers:"
read a b c

if [ $a -ge $b ] && [ $a -ge $c ]; then
  echo "$a is the greatest"
elif [ $b -ge $a ] && [ $b -ge $c ]; then
  echo "$b is the greatest"
else
  echo "$c is the greatest"
fi
echo

# 2. Even or Odd Number
echo "Program 2: Even or Odd Number"
```

```
echo "Enter a number:"
read num

if [ $((num % 2)) -eq 0 ]; then
  echo "$num is Even"
else
  echo "$num is Odd"
fi
echo

# 3. Average of Three Numbers
echo "Program 3: Average of Three Numbers"
echo "Enter three numbers:"
read a b c

avg=$(echo "scale=2; ($a + $b + $c) / 3" | bc)
echo "The average is $avg"
echo

# 4. Prime or Not
echo "Program 4: Prime or Not"
echo "Enter a number:"
read num
flag=0

for ((i=2; i<=$((num / 2)); i++)); do
  if [ $((num % i)) -eq 0 ]; then
    flag=1
```

```
    break
  fi
done

if [ $num -eq 1 ]; then
  echo "1 is neither prime nor composite"
elif [ $flag -eq 0 ]; then
  echo "$num is a prime number"
else
  echo "$num is not a prime number"
fi
echo

# 5. Factorial of a Number
echo "Program 5: Factorial of a Number"
echo "Enter a number:"
read num
fact=1

for ((i=1; i<=num; i++)); do
  fact=$((fact * i))
done

echo "Factorial of $num is $fact"
echo

# 6. Fibonacci Sequence
echo "Program 6: Fibonacci Sequence"
```

```
echo "Enter the number of terms:"
read terms
a=0
b=1

echo "Fibonacci sequence up to $terms terms:"
for ((i=0; i<terms; i++)); do
  echo -n "$a "
  fib=$((a + b))
  a=$b
  b=$fib
done
echo
echo

# 7. Sum of Digits
echo "Program 7: Sum of Digits"
echo "Enter a number:"
read num
sum=0

while [ $num -gt 0 ]; do
  digit=$((num % 10))
  sum=$((sum + digit))
  num=$((num / 10))
done

echo "Sum of digits is $sum"
```

```
echo

# 8. String Validation (Empty or Not)
echo "Program 8: String Validation (Empty or Not)"
echo "Enter a string:"
read str

if [ -z "$str" ]; then
  echo "String is not valid (empty)"
else
  echo "String is valid"
fi
echo

# 9. Count Number of Words and Characters in a String
echo "Program 9: Count Number of Words and Characters in a String"
echo "Enter a string:"
read str

word_count=$(echo $str | wc -w)
char_count=$(echo $str | wc -c)

echo "Number of words: $word_count"
echo "Number of characters: $char_count"
echo

# 10. Palindrome or Not (String)
echo "Program 10: Palindrome or Not (String)"
```

```
echo "Enter a string:"
read str
rev=$(echo $str | rev)

if [ "$str" == "$rev" ]; then
  echo "$str is a palindrome"
else
  echo "$str is not a palindrome"
fi
echo
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 1.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 1.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./1.sh
Enter three numbers:
34 67 12
67 is the greatest
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 2.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 2.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./2.sh
Enter a number:
573543
573543 is Odd
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 3.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 3.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./3.sh
Enter three numbers:
2 6 10
The average is 6.00
```

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 4.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 4.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./4.sh
Enter a number:
367531
367531 is a prime number
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 5.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 5.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./5.sh
Enter a number:
8
Factorial of 8 is 40320
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 6.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 6.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./6.sh
Enter the number of terms:
7
Fibonacci sequence up to 7 terms:
0 1 1 2 3 5 8
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 7.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 7.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./7.sh
Enter a number:
6565453
Sum of digits is 34
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 8.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 8.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./8.sh
Enter a string:
Amit Singhal
String is valid
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 9.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 9.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./9.sh
Enter a string:
Kavita Saxena
Number of words: 2
Number of characters: 14
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi 10.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x 10.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./10.sh
Enter a string:
Madam
Madam is not a palindrome
```

Lab Exercise – 7

- AIM :: WAP in shell script to implement CPU scheduling for `first come first serve` (fcfs).

Source_Code ::

```
echo $'\n' "5C6 - Amit Singhal (11614802722)" $'\n'

read -p "Enter the number of processes: " num_processes
echo $'\n' "Enter Arrival Time & Burst Time for $num_processes processes"

# Collect process details
for ((i=0;i<num_processes;i++)); do
    echo -n "P$((i+1)): "
    read arrival_time burst_time
    processes[$i]="$arrival_time $burst_time"
done

# Sort processes by arrival time
IFS=$'\n' sorted_processes=($(sort -n -k1 <<<"${processes[*]}"))
unset IFS

# Initialize variables
total_completion_time=0
total_waiting_time=0
total_turnaround_time=0
gantt_chart="0"  # Start Gantt chart at time 0

# Display table header
```

```
echo -e "\nProcess    Arrival Time    Burst Time    Completion Time    TurnAround Time    Waiting Time"

# Process all processes
for ((i=0;i<num_processes;i++)); do
    current_process=(${sorted_processes[$i]})
    current_arrival_time=${current_process[0]}
    current_burst_time=${current_process[1]}

    # If the process arrives after the last completion time, idle CPU
    if (( total_completion_time < current_arrival_time )); then
        idle_time=$((current_arrival_time - total_completion_time))
        total_completion_time=$current_arrival_time
        gantt_chart+=" -- XX -- $total_completion_time"
    fi

    # Calculate waiting time
    if (( total_completion_time >= current_arrival_time )); then
        waiting_time=$((total_completion_time - current_arrival_time))
    else
        waiting_time=0
    fi

    # Calculate completion time and turnaround time
    completion_time=$((total_completion_time + current_burst_time))
    turnaround_time=$((completion_time - current_arrival_time))

    # Update total values
    total_completion_time=$completion_time
    total_waiting_time=$((total_waiting_time + waiting_time))
    total_turnaround_time=$((total_turnaround_time + turnaround_time))

    # Display process details
    echo -e "P$((i+1))\t\t$current_arrival_time\t\t$current_burst_time\t\t$completion_time\t\t  $turnaround_time\t\t  $waiting_time"
```

```
    # Update Gantt chart
    gantt_chart+=" -- P$((i+1)) -- $completion_time"
done

# Calculate averages
avg_waiting_time=$(awk "BEGIN {printf \"%.2f\", $total_waiting_time/$num_processes}")
avg_turnaround_time=$(awk "BEGIN {printf \"%.2f\", $total_turnaround_time/$num_processes}")

# Display Gantt chart
echo -e "\nGantt Chart:"
echo -e "$gantt_chart"

# Display averages
echo ""
echo "Avg waiting time: $avg_waiting_time"
echo "Avg turnaround time: $avg_turnaround_time"
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ vi prg_5_fcfs.sh
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ chmod +x prg_5_fcfs.sh
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ ./prg_5_fcfs.sh

 5C6 - Amit Singhal (11614802722)

Enter the number of processes: 4

 Enter Arrival Time & Burst Time for 4 processes
P1: 0 2
P2: 1 2
P3: 5 3
P4: 6 4

Process    Arrival Time    Burst Time    Completion Time    TurnAround Time    Waiting Time
P1              0               2               2                2               0
P2              1               2               4                3               1
P3              5               3               8                3               0
P4              6               4               12               6               2

Gantt Chart:
0 -- P1 -- 2 -- P2 -- 4 -- XX -- 5 -- P3 -- 8 -- P4 -- 12

Avg waiting time: 0.75
Avg turnaround time: 3.50
```

Lab Exercise – 8

AIM :: WAP in shell script to implement CPU scheduling for `shortest job first` (sjf).

Source_Code ::

```
echo $'\n' "5C6 - Amit Singhal (11614802722)" $'\n'

read -p "Enter the number of processes: " num_processes
echo $'\n' "Enter Arrival Time & Burst Time for $num_processes processes"

# Collect process details
for ((i=0;i<num_processes;i++)); do
    echo -n "P$((i+1)): "
    read arrival_time burst_time
    processes[$i]="$arrival_time $burst_time"
done

# Initialize variables
total_completion_time=0
total_waiting_time=0
total_turnaround_time=0
completed_processes=0
gantt_chart="0"  # Start Gantt chart at time 0
time=0

# Create an array to store completion status of each process (0 = incomplete, 1 = complete)
for ((i=0;i<num_processes;i++)); do
```

```
    process_completed[$i]=0
done

# Function to find the process with the shortest burst time among those that have arrived
find_shortest_job() {
    local min_burst=-1
    local min_index=-1

    for ((i=0;i<num_processes;i++)); do
        current_process=(${processes[$i]})
        current_arrival_time=${current_process[0]}
        current_burst_time=${current_process[1]}

        if (( process_completed[$i] == 0 && current_arrival_time <= time )); then
            if (( min_burst == -1 || current_burst_time < min_burst )); then
                min_burst=$current_burst_time
                min_index=$i
            fi
        fi
    done

    echo $min_index
}

# Display table header
echo -e "\nProcess    Arrival Time    Burst Time    Completion Time    Turnaround Time    Waiting Time"

# Process all processes using SJF
while (( completed_processes < num_processes )); do
    shortest_job=$(find_shortest_job)

    if (( shortest_job == -1 )); then
```

```
        # No process available, increase time (idle)
        gantt_chart+=" -- XX -- $((++time))"
    else
        current_process=(${processes[$shortest_job]})
        current_arrival_time=${current_process[0]}
        current_burst_time=${current_process[1]}

        if (( time < current_arrival_time )); then
            time=$current_arrival_time
            gantt_chart+=" -- XX -- $time"
        fi

        completion_time=$((time + current_burst_time))
        turnaround_time=$((completion_time - current_arrival_time))
        waiting_time=$((turnaround_time - current_burst_time))

        # Update total values
        total_completion_time=$completion_time
        total_waiting_time=$((total_waiting_time + waiting_time))
        total_turnaround_time=$((total_turnaround_time + turnaround_time))

        # Mark the process as completed
        process_completed[$shortest_job]=1
        completed_processes=$((completed_processes + 1))

        # Display process details
        echo -e "P$((shortest_job+1))\t\t$current_arrival_time\t\
t$current_burst_time\t\t$completion_time\t\t $turnaround_time\t\t
$waiting_time"

        # Update Gantt chart
        gantt_chart+=" -- P$((shortest_job+1)) -- $completion_time"

        # Update current time
        time=$completion_time
```

```
    fi
done

# Calculate averages
avg_waiting_time=$(awk "BEGIN {printf \"%.2f\", $total_waiting_time/$num_processes}")
avg_turnaround_time=$(awk "BEGIN {printf \"%.2f\", $total_turnaround_time/$num_processes}")

# Display Gantt chart
echo -e "\nGantt Chart:"
echo -e "$gantt_chart"

# Display averages
echo ""
echo "Avg waiting time: $avg_waiting_time"
echo "Avg turnaround time: $avg_turnaround_time"
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ vi prg_6_sjf.sh
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ chmod +x prg_6_sjf.sh
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads/_LAB_Work/OS/Code$ ./prg_6_sjf.sh

 5C6 - Amit Singhal (11614802722)

Enter the number of processes: 4

 Enter Arrival Time & Burst Time for 4 processes
P1: 1 3
P2: 2 4
P3: 1 2
P4: 4 4

Process    Arrival Time    Burst Time    Completion Time    Turnaround Time    Waiting Time
P3              1              2              3                  2                 0
P1              1              3              6                  5                 2
P2              2              4              10                 8                 4
P4              4              4              14                 10                6

Gantt Chart:
0 -- XX -- 1 -- P3 -- 3 -- P1 -- 6 -- P2 -- 10 -- P4 -- 14

Avg waiting time: 3.00
Avg turnaround time: 6.25
```

2) Preemptive Mode

Source_Code ::

```
echo $'\n' "5C6 - Amit Singhal (11614802722)" $'\n'


read -p "Enter the number of processes: " num_processes

echo $'\n' "Enter Arrival Time & Burst Time for $num_processes processes"


# Collect process details

for ((i=0;i<num_processes;i++)); do

    echo -n "P$((i+1)): "

    read arrival_time burst_time

    processes[$i]="$arrival_time $burst_time"

    remaining_burst[$i]=$burst_time  # Track the remaining burst time for preemption

    process_completed[$i]=0          # Track if the process is completed

done


# Initialize variables

total_completion_time=0

total_waiting_time=0

total_turnaround_time=0

gantt_chart="0"  # Start Gantt chart at time 0

time=0           # Global time

completed_processes=0
```

```
prev_process=-1  # Track the previously executing process for Gantt chart


# Function to find the process with the shortest remaining burst time among those that have arrived
find_shortest_remaining() {
    local min_burst=-1
    local min_index=-1

    for ((i=0;i<num_processes;i++)); do
        current_process=(${processes[$i]})
        current_arrival_time=${current_process[0]}

        if (( process_completed[$i] == 0 && current_arrival_time <= time )); then
            if (( min_burst == -1 || remaining_burst[$i] < min_burst )); then
                min_burst=${remaining_burst[$i]}
                min_index=$i
            fi
        fi
    done

    echo $min_index
}

# Display table header
```

```
echo -e "\nProcess\t Arrival Time\t Burst Time\t Completion Time\t Turnaround Time\t Waiting Time"


# Process all processes using SRTF (Preemptive SJF)
while (( completed_processes < num_processes )); do
    shortest_job=$(find_shortest_remaining)

    if (( shortest_job == -1 )); then
        # No process available, increase time (idle)
        gantt_chart+=" -- XX -- $((++time))"
    else
        current_process=(${processes[$shortest_job]})
        current_arrival_time=${current_process[0]}
        current_burst_time=${current_process[1]}

        # If a new process is selected or time has changed
        if (( prev_process != shortest_job )); then
            if (( prev_process != -1 )); then
                gantt_chart+=" -- $time"
            fi
            gantt_chart+=" -- P$((shortest_job+1))"
            prev_process=$shortest_job
        fi

        # Execute the shortest job for one unit of time
        remaining_burst[$shortest_job]=$((remaining_burst[$shortest_job] - 1))
```

```
        time=$((time + 1))

        # If the process is completed, update its stats
        if (( remaining_burst[$shortest_job] == 0 )); then
            completion_time=$time
            turnaround_time=$((completion_time - current_arrival_time))
            waiting_time=$((turnaround_time - current_burst_time))

            # Update total values
            total_completion_time=$completion_time
            total_waiting_time=$((total_waiting_time + waiting_time))
            total_turnaround_time=$((total_turnaround_time + turnaround_time))

            # Mark the process as completed
            process_completed[$shortest_job]=1
            completed_processes=$((completed_processes + 1))

            # Display process details
            echo -e "P$((shortest_job+1))\t\t$current_arrival_time\t\t$current_burst_time\t\t$completion_time\t\t\t$turnaround_time\t\t\t$waiting_time"
        fi
    fi
done

# End Gantt chart with the last completion time
```

```
gantt_chart+=" -- $time"


# Calculate averages

avg_waiting_time=$(awk "BEGIN {printf \"%.2f\", $total_waiting_time/$num_processes}")

avg_turnaround_time=$(awk "BEGIN {printf \"%.2f\", $total_turnaround_time/$num_processes}")


# Display Gantt chart

echo -e "\nGantt Chart:"

echo -e "$gantt_chart"


# Display averages

echo ""

echo "Avg waiting time: $avg_waiting_time"

echo "Avg turnaround time: $avg_turnaround_time"
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi prg_7.2_sjf.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x prg_7.2_sjf.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./prg_7.2_sjf.sh

 5C6 - Amit Singhal (11614802722)

Enter the number of processes: 6

 Enter Arrival Time & Burst Time for 6 processes
P1: 5 9
P2: 4 8
P3: 3 7
P4: 2 7
P5: 5 8
P6: 6 9

Process  Arrival Time    Burst Time      Completion Time         Turnaround Time         Waiting Time
P4              2               7               9                       7                       0
P3              3               7               16                      13                      6
P2              4               8               24                      20                      12
P5              5               8               32                      27                      19
P1              5               9               41                      36                      27
P6              6               9               50                      44                      35

Gantt Chart:
0 -- XX -- 1 -- XX -- 2 -- P4 -- P4 -- 9 -- P3 -- P3 -- 16 -- P2 -- P2 -- 24 -- P5 -- P5 -- 32 -- P1 -- P1 -- 41 -- P6 -- 50

Avg waiting time: 16.50
Avg turnaround time: 24.50
```

Lab Exercise – 9

❖ AIM :: WAP to perform Priority Scheduling.

Source_Code ::

```
echo $'\n' "5C6 - Amit Singhal (11614802722)" $'\n'

# Read the number of processes
read -p "Enter the number of processes: " num_processes
echo $'\n'

# Declare arrays for storing process information
declare -a arrival
declare -a burst
declare -a priority
declare -a completion
declare -a waiting
declare -a turnaround
declare -a process_ids
declare -a remaining_burst

# Input arrival time, burst time, and priority for each process
for ((i=0; i<num_processes; i++))
do
    process_ids[$i]=$((i+1))
    echo -n "Enter Arrival Time, Burst Time, and Priority for Process $((i+1)): "
    read arrival[$i] burst[$i] priority[$i]
    remaining_burst[$i]=${burst[$i]}  # Initialize remaining burst time
    completion[$i]=0  # Initialize completion time to 0
done

# Priority scheduling with preemption
priority_scheduling() {
    time=0
    completed=0
    gantt_chart=""
    prev_process=-1
```

```
    while [ $completed -lt $num_processes ]; do
        # Find the process with the highest priority that has arrived and has remaining burst time
        highest_priority=-1
        current_process=-1

        for ((i=0; i<num_processes; i++)); do
            if [ ${arrival[$i]} -le $time ] && [ ${remaining_burst[$i]} -gt 0 ]; then
                if [ $highest_priority -eq -1 ] || [ ${priority[$i]} -lt $highest_priority ]; then
                    highest_priority=${priority[$i]}
                    current_process=$i
                fi
            fi
        done

        if [ $current_process -ne -1 ]; then
            if [ $current_process -ne $prev_process ]; then
                gantt_chart+="$time -- P${process_ids[$current_process]} -- "
            fi

            remaining_burst[$current_process]=$((remaining_burst[$current_process] - 1))
            time=$((time + 1))

            # If the process finishes, calculate its completion, turnaround, and waiting times
            if [ ${remaining_burst[$current_process]} -eq 0 ]; then
                completion[$current_process]=$time
                turnaround[$current_process]=$((completion[$current_process] -
                                                arrival[$current_process]))
                waiting[$current_process]=$((turnaround[$current_process] -
                                             burst[$current_process]))
                completed=$((completed + 1))
            fi

            prev_process=$current_process
        else
            gantt_chart+="$time -- XX -- "
            time=$((time + 1))
        fi
    done
    gantt_chart+="$time"  # Add the final time to Gantt chart
}
```

```
# Function to display the Gantt chart
display_gantt_chart() {
    echo $'\n'"Gantt Chart:"
    echo "$gantt_chart"
}

# Function to display the process table with calculated times
display_results() {
    echo $'\n'"PID |  AT  |  BT  | Priority |  CT  |  TAT  |  WT  |"
    echo "---------------------------------------------------"
    for ((i=0; i<num_processes; i++)); do
        printf "P%-3d |  %-3d |  %-2d |     %-4d |  %-3d |   %-3d |  %-3d |\n" \
            "${process_ids[$i]}" "${arrival[$i]}" "${burst[$i]}" "${priority[$i]}" \
            "${completion[$i]}" "${turnaround[$i]}" "${waiting[$i]}"
    done
    echo "---------------------------------------------------"
}

# Function to calculate and display the average waiting and turnaround times
calculate_averages() {
    total_waiting=0
    total_turnaround=0

    for ((i=0; i<num_processes; i++)); do
        total_waiting=$((total_waiting + waiting[$i]))
        total_turnaround=$((total_turnaround + turnaround[$i]))
    done

    avg_waiting=$(echo "scale=2; $total_waiting / $num_processes" | bc)
    avg_turnaround=$(echo "scale=2; $total_turnaround / $num_processes" | bc)

    echo $'\n'"Average Waiting Time <WT> :: $avg_waiting"
    echo "Average Turnaround Time <TAT> :: $avg_turnaround"
}

# Run the priority scheduling algorithm with preemption
priority_scheduling

# Display the Gantt chart
display_gantt_chart
```

```
# Display the process table
display_results

# Calculate and display the averages
calculate_averages
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads$ chmod +x prg9
singhal-amit@singhal-amit-ThinkPad-T430:~/Downloads$ ./prg9

 5C6 - Amit Singhal (11614802722)

Enter the number of processes: 4


Enter Arrival Time, Burst Time, and Priority for Process 1: 0 4 2
Enter Arrival Time, Burst Time, and Priority for Process 2: 1 3 1
Enter Arrival Time, Burst Time, and Priority for Process 3: 2 5 3
Enter Arrival Time, Burst Time, and Priority for Process 4: 3 2 4

Gantt Chart:
0 -- P1 -- 1 -- P2 -- 4 -- P1 -- 7 -- P3 -- 12 -- P4 -- 14

PID |  AT  |  BT  | Priority |  CT  |  TAT  |  WT  |
-------------------------------------------------------
P1   |  0   |  4  |     2    |  7   |   7   |  3   |
P2   |  1   |  3  |     1    |  4   |   3   |  0   |
P3   |  2   |  5  |     3    |  12  |   10  |  5   |
P4   |  3   |  2  |     4    |  14  |   11  |  9   |
-------------------------------------------------------

Average Waiting Time <WT> :: 4.25
Average Turnaround Time <TAT> :: 7.75
```