1. PROGRAM TO FIND GREATEST OF TWO NUMBERS

Aim:

To write a program to find the greatest of two numbers

Algorithm:

1. Enter the two numbers which are to be compared.
2. Check for the greatest number among the two.
3. If A is greater, then print the result as “A is the greatest number”.
4. If B is greater, then print the result as “B is the greatest number”.

Program:

```
echo “enter 2 numbers”

read a b

if[$a-$b]

then

echo “$a is greater”

else

echo “$b is greater”

fi
```

Output:

```
enter 2 numbers 10 20

20 is greater
```

Result:

Thus the program to find the greater of two numbers was successfully executed.

2. Find the cube

Aim:

To write a program to find the cube

Algorithm:

1. Enter the value of a number
2. Perform the product of the given number thrice
3. Display the result.

Program:

```
echo “enter a number”

read a

let b=$a\*$a\*$a

echo “the cube is $b”
```

Output:

```
Finding the queue:

Enter a number

3

The cube is 27
```

Result:

Thus the above program finding the cube has been executed successfully.

3.Sum of N numbers

Aim:

To write a program for sum of N numbers

Algorithm:

1. Enter the limit as n.
2. Initialize I as 1 and sum as 0.
3. Compute the value of sum and I until I is less than or equal to n.
4. Sum is computed by adding the value of sum and i.
5. I is computed by adding one to i. and display the result of sum.

Program:

```
echo “enter limit”

read n

i=1

sum=0

while[$i –le $n]

do

let sum=$sum+$i

let i=$i+1

done

echo”the sum of $n numbers is $sum”
```

Output:

```
 Sum of N numbers

Enter limit 8

The sum of 8 numbers is 36
```

Result: Thus the above program sum of N numbers has been executed successfully.

4. Swapping of two numbers

Aim:

To write a program to swapping of two numbers

Algorithm:

1. Enter the 2 numbers such as a and b to be swapped.
2. Assign the temp value as one.
3. Assign the temp variable as a.
4. Assign the value of b to a.
5. Assign the temp value to b.
6. Display the value of a and b.

Program:

```
echo “enter two numbers”

read a b

t=$a

a=$b

b=$t

b=$t

echo “a=$a b=$b”
```

Output:

Swapping of two numbers:

Enter two numbers 5 9 a=9 b=5

Result:

Thus the above program swapping of two numbers has been executed successfully.

5. Checking the number is positive or negative

Aim:

To write a program to find the given number is positive or negative

Algorithm:

1. Enter the number
2. If the number is greater than zero, display that the number is positive.
3. If the number is less than zero, display that the number is negative.
4. If the number is neither greater than zero nor less than zero, display that the number is zero.

Program:

```
echo" enter the number"

read a

if[$a –ge 0]

then

echo "$a is positive"

else

echo "$a is negative"

fi
```

Output:

```
Checking the number is positive or negative

7 is positive

Enter a number

-8

-8 is negative
```

Result: Thus the above program has been executed successfully.

Ex No: 6. Basic Calculator Using Switch Case

Aim:

To develop a Basic Math calculator Using case Statement.

Algorithm:

1) Create a new file.

2) Read the operands.

3) Select any one operation from the list.

4) Perform the operation. 5) Print the result.

Program:

```
# Implementation of Calculator application

#!bin/bash

# Prompt user to enter two numbers

echo "Enter two numbers:"

read -p "Number 1: " a

read -p "Number 2: " b

# Prompt user to select an operation

echo "Enter Choice :"

echo "1. Addition"

echo "2. Subtraction"

echo "3. Multiplication"

echo "4. Division"
```

```
read -p "Your choice: " ch

# Switch Case to perform calculator operations

case $ch in

        1) res=$(echo "$a + $b" | bc) ;;

        2) res=$(echo "$a - $b" | bc) ;;

        3) res=$(echo "$a * $b" | bc) ;;

        4) res=$(echo "scale=2; $a / $b" | bc) ;;

        *) echo "Invalid choice"

        exit 1 ;;

esac

# Display the result

echo "Result : $res"
```

OUTPUT:

```
enter two no

number1:20

number2:30

enter choice

1. addition

2. subtraction

3. multiplication

4. division

your choice3

result: 600
```

enter two no

number1:20

number2:30

enter choice

1. addition

2. subtraction

3. multiplication

4. division

your choice4

result: .66

Result:

Thus the above program to develop a calculator application was executed successfully.

7. First come First Serve

Aim:

To write a c program for FCFS

Algorithm:

1. Get the number of processes and burst time.
2. The process is executed in the order given by the user.
3. Calculate the waiting time and turnaround time.
4. Display the gantt chart, avg waiting time and turnaround time.

Program:

```
#include <stdio.h>

int main() {

   int n, bt[20], wt[20], tat[20], avwt = 0, avtat = 0, i, j;


   printf("Enter total number of processes (maximum 20): ");

   scanf("%d", &n);

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

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

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

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

   }

   wt[0] = 0;

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

      wt[i] = 0;

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

         wt[i] += bt[j];
```

```
    }

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avwt += wt[i];
        avtat += tat[i];
        printf("P[%d]\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }
    avwt /= i;
    avtat /= i;
    printf("Average Waiting Time: %d\n", avwt);
    printf("Average Turnaround Time: %d\n", avtat);
    return 0;
}
```

Output:

Enter total number of processes (maximum 20):3

Enter Process Burst Time

P[1]:1

P[2]:2

P[3]:3

ProcessttBurst TimetWaiting TimetTurnaround Time

P[1]tt1tt0tt1

P[2]tt2tt1tt3

P[3]tt3tt3tt6

Average Waiting Time:1

Average Turnaround Time:3

Result:

Thus the above program has been executed successfully.

8. Shortest Job First

Aim:

To write a c program for Shortest Job First

Algorithm:

1. Get the number of processes and burst time.
2. Sort the process based on the burst time in ascending order.
3. Calculate the waiting time and turnaround time.
4. Display the gantt chart, avg waiting time and turnaround time.

Program:

```
#include <stdio.h>

int main() {
    int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("\nEnter number of processes: ");
    scanf("%d", &n);
    printf("\nEnter Burst Time:\n");
    for (i = 0; i < n; i++) {
        printf("p%d: ", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1;
    }

    // Sorting of burst times
    for (i = 0; i < n; i++) {
        pos = i;
        for (j = i + 1; j < n; j++) {
```

```
        if (bt[j] < bt[pos])
            pos = j;
    }
    temp = bt[i];
    bt[i] = bt[pos];
    bt[pos] = temp;
    temp = p[i];
    p[i] = p[pos];
    p[pos] = temp;
}
wt[0] = 0;
for (i = 1; i < n; i++) {
    wt[i] = 0;
    for (j = 0; j < i; j++)
        wt[i] += bt[j];
    total += wt[i];
}
avg_wt = (float)total / n;
total = 0;
printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
for (i = 0; i < n; i++) {
    tat[i] = bt[i] + wt[i];
    total += tat[i];
    printf("p%d\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);
}
```

```
    avg_tat = (float)total / n;
    printf("\nAverage Waiting Time = %f", avg_wt);
    printf("\nAverage Turnaround Time = %f\n", avg_tat);
    return 0;
}
```

Output:

SJF:

Enter the number of process 3

Enter the burst time 2 1 3

Gantt chart

P1|p2|p3|

0 1 3 6

Average waiting time is 1.33

Average turnaround time is 3.33

Result:

Thus the above program has been executed successfully.

9.Round Robin(pre-emptive)

Aim:

To write a c program for round robin algorithm

Algorithm:

1. Get the number of processes and burst time.
2. Sort the process based on the burst time in ascending order.
3. Calculate the waiting time and turnaround time.
4. Display the gantt chart, avg waiting time and turnaround time.

Program

```
#include<stdio.h>

int main() {

   int i, NOP, sum = 0, count = 0, y, quant, wt = 0, tat = 0, at[10], bt[10], temp[10];

   float avg_wt, avg_tat;

      printf("Total number of processes in the system: ");

   scanf("%d", &NOP);

   y = NOP;

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

      printf("\nEnter the Arrival and Burst time of Process[%d]\n", i + 1);

      printf("Enter Arrival time: ");

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

      printf("Enter Burst time: ");

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

      temp[i] = bt[i];

   }

      printf("Enter the Time Quantum for the process: ");

   scanf("%d", &quant);
```

```
    printf("\nProcess No \t\t Burst Time \t\t TAT \t\t Waiting Time");


  for (sum = 0, i = 0; y != 0;) {
    if (temp[i] <= quant && temp[i] > 0) {
      sum = sum + temp[i];
      temp[i] = 0;
      count = 1;
    } else if (temp[i] > 0) {
      temp[i] = temp[i] - quant;
      sum = sum + quant;
    }
        if (temp[i] == 0 && count == 1) {
      y--;
      printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i + 1, bt[i], sum - at[i], sum -
at[i] - bt[i]);
      wt = wt + sum - at[i] - bt[i];
      tat = tat + sum - at[i];
      count = 0;
    }


    if (i == NOP - 1) {
      i = 0;
    } else if (at[i + 1] <= sum) {
      i++;
    } else {
      i = 0;
```

```
        }
    }

    avg_wt = (float)wt / NOP;
    avg_tat = (float)tat / NOP;
        printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);
}
```

Output:

Total number of process in the system: 2

Enter the Arrival and Burst time of the Process [1]

Enter Arrival time: 22

Enter Burst time: 3

Enter the Arrival and Burst time of the Process [2]

Enter Arrival time: 2

Enter Burst time: 2

Enter the Time Quantum for the process: 3

Process No Burst Time TAT Waiting

Time

Process No[1] 3 -19 -22

Process No[2] 2 3 1

Average waiting time: -10.50

Average Turnaround time: -8

Result:

Thus the above program has been executed successfully.

10. Priority

Aim:

To write a program for priority scheduling

Algorithm:

1. Get the number of processes and burst time.
2. Sort the process based on the burst time in ascending order.
3. Calculate the waiting time and turnaround time.
4. Display the gantt chart, avg waiting time and turnaround time.

Program

```
#include<stdio.h>

void swap(int *a, int *b) {

   int temp = *a;

   *a = *b;

   *b = temp;

}

int main() {

   int n, i, j;

   printf("Enter Number of Processes: ");

   scanf("%d", &n);

   // Arrays to store burst times, priorities, and process IDs

   int burst_time[n], priority[n], process_id[n];

   // Input burst times and priorities for each process

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

      printf("Enter Burst Time and Priority Value for Process %d: ", i + 1);

      scanf("%d %d", &burst_time[i], &priority[i]);

      process_id[i] = i + 1; // Assign process IDs

   }
```

```
// Sort processes based on priority (higher priority first)
for(i = 0; i < n - 1; i++) {
    for(j = 0; j < n - i - 1; j++) {
        if(priority[j] < priority[j + 1]) {
            swap(&priority[j], &priority[j + 1]);
            swap(&burst_time[j], &burst_time[j + 1]);
            swap(&process_id[j], &process_id[j + 1]);
        }
    }
}
// Calculate completion time, turnaround time, and waiting time
int waiting_time[n], turnaround_time[n], completion_time = 0;
float avg_waiting_time = 0, avg_turnaround_time = 0;
for(i = 0; i < n; i++) {
    completion_time += burst_time[i];
    turnaround_time[i] = completion_time;
    waiting_time[i] = turnaround_time[i] - burst_time[i];
    avg_waiting_time += waiting_time[i];
    avg_turnaround_time += turnaround_time[i];
}
avg_waiting_time /= n;
avg_turnaround_time /= n;
// Print schedule and process details
printf("\nOrder of Process Execution:\n");
printf("Process ID\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTur
```

```
naround Time\n");
    for(i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", process_id[i], burst_time[i
], priority[i], turnaround_time[i], waiting_time[i], completion_time);
    }
    printf("\nAverage Waiting Time: %.2f\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);

    return 0;
}
```

OUTPUT:

```
Enter Number of Processes: 4
Enter Burst Time and Priority Value for Process 1: 1 2
Enter Burst Time and Priority Value for Process 2: 2 2
Enter Burst Time and Priority Value for Process 3: 2 3
Enter Burst Time and Priority Value for Process 4: 3 3

Order of Process Execution:
Process ID      Burst Time      Priority        Completion Time Waiting Time
Turnaround Time
3               2               3               2               0
8
4               3               3               5               2
8
1               1               2               6               5
8
```

2 2 2 8 6

8

Average Waiting Time: 3.25

Average Turnaround Time: 5.25

Result:

Thus the above program has been executed successfully.

11. Implement the page replacement algorithms FIFO

Aim:

To write a program for page replacement algorithms FIFO

Algorithm:

1. Maintain a queue to store the pages in the order they were loaded.
2. When a page is requested:
- If the page is already in the queue → it's a hit.
- If the page is not in the queue:
    - If the queue is full → remove the oldest page (first in).
    - Add the new page to the back of the queue.
3. Count the number of hits and misses.

Program:

```
#include <stdio.h>

#include <stdlib.h>

int fifo(int pages[], int n, int capacity) {

        int *memory = (int *)malloc(capacity * sizeof(int));

        int page_faults = 0;

        int index = 0;

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

        memory[i] = -1;

        }

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

        int page = pages[i];

        int found = 0;

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

        if (memory[j] == page) {

                found = 1;
```

```
                break;
            }
            }
            if (!found) {
             memory[index] = page;
            index = (index + 1) % capacity;
             page_faults++;
            }
            }
            free(memory);
            return page_faults;
}
int main() {
            int capacity = 3;
            int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
            int n = sizeof(pages) / sizeof(pages[0]);
            printf("FIFO Page Faults: %d\n", fifo(pages, n, capacity));
            return 0;
}
```

OUTPUT:

FIFO Page Faults: 10