LAB 2: Process Scheduling Algorithms

OBJECTIVE:

The report aims to provide a clear understanding of how each process scheduling algorithm functions, examining their advantages and limitations. Through a comparative analysis, we will investigate the impact of these algorithms on system performance metrics such as turnaround time, waiting time, and throughput.

THEORY:

Process scheduling algorithms are integral components of operating systems that manage the execution of multiple processes, ensuring efficient resource utilization and system responsiveness. And they are:

- First-Come-First-Serve (FCFS) Algorithm:

  In the FCFS algorithm, processes are executed in the order they arrive, with the first process that enters the ready queue being the first one to be executed. This simplistic approach is easy to understand and implement. However, FCFS may lead to the "convoy effect," where shorter processes get delayed behind longer ones, impacting overall system efficiency. The primary advantage of FCFS lies in its simplicity and ease of implementation, making it suitable for scenarios with low computational overhead.

- Shortest Job First (SJF) Algorithm:
  Contrastingly, the SJF algorithm prioritizes the execution of the shortest job first, aiming to minimize the total processing time. This algorithm requires predicting the burst time of each process, which can be challenging in real-world scenarios. Preemptive SJF allows for dynamic adaptation to changing burst times, while non-preemptive SJF commits to a selected process until completion. SJF is known for its efficiency in minimizing waiting times and maximizing throughput, especially in scenarios where burst times are accurately known.

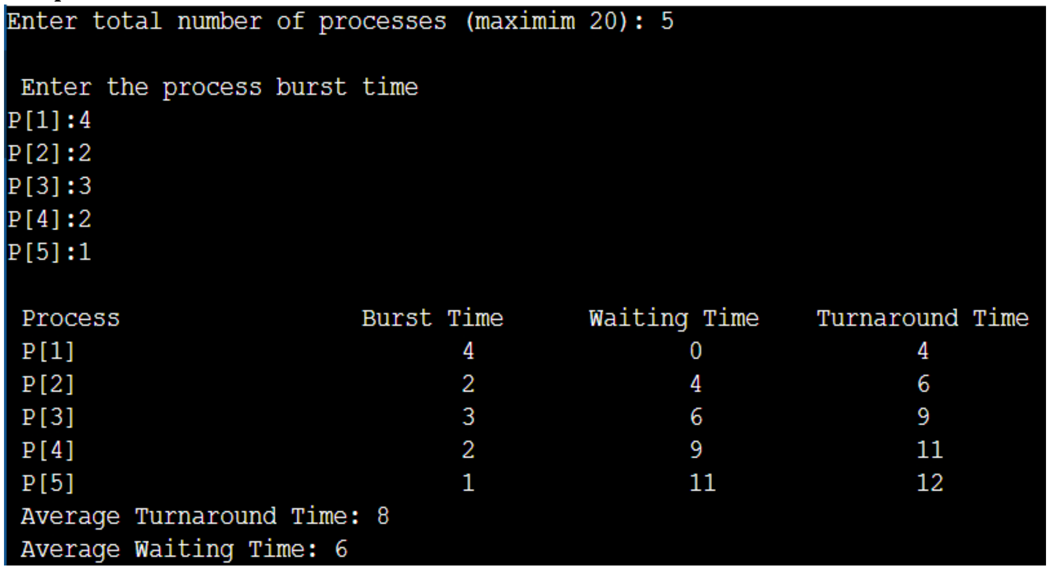
- Priority Algorithm:
  A priority algorithm is a computational approach used in various fields to determine the order or sequence in which tasks, processes, or elements should be executed or processed based on assigned priorities. The fundamental concept behind priority algorithms is to allocate resources or attention to higher-priority items before lower-priority ones.

Program 1: Write a program demonstrating use of FCFS Algorithm for programs having same arrival time

Solution:

```
#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 (maximim 20): ");
   scanf("%d", &n);
   printf("\n Enter the 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("\n Process \t\t Burst Time \t Waiting Time\t Turnaround Time");
   for(i=0;i<n;i++){
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+= tat[i];
      printf("\n P[%d] \t\t%d \t\t%d \t\t%d", i+1, bt[i], wt[i], tat[i]);    }
   avwt/=i;
   avtat/=i;
   printf("\n Average Turnaround Time: %d", avtat);
   printf("\n Average Waiting Time: %d", avwt);
   return 0;
}
```

Output:



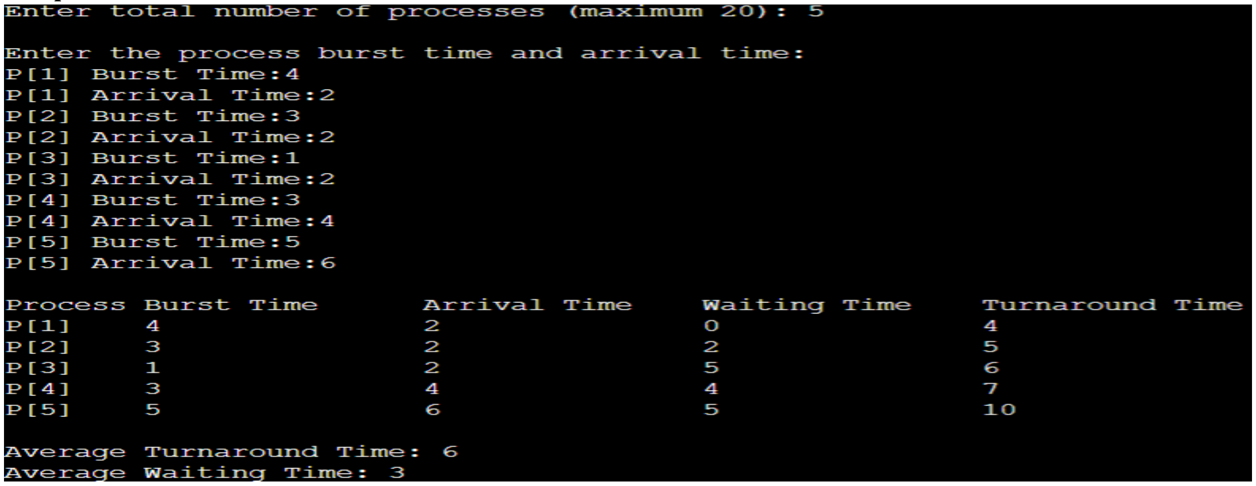
```
Enter total number of processes (maximim 20): 5

 Enter the process burst time
P[1]:4
P[2]:2
P[3]:3
P[4]:2
P[5]:1

 Process                Burst Time      Waiting Time    Turnaround Time
 P[1]                         4               0               4
 P[2]                         2               4               6
 P[3]                         3               6               9
 P[4]                         2               9               11
 P[5]                         1               11              12
 Average Turnaround Time: 8
 Average Waiting Time: 6
```

Program 2: Write a program demonstrating use of FCFS Algorithm for programs having different arrival time

Solution:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main(){
  char pn[10][10],t[10];
  int arr[10],bur[10], star[10],finish[10],tat[10],wt[10],i,j,n,temp;
  int totwt = 0, tottat = 0;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  for(i=0;i<n;i++){
    printf("Enter the Process Name Arrival Time $ Burst Time: ");
    scanf("%s%d%d", &pn[i], &arr[i], &bur[i]);
  }
  for(i=0;i<n;i++){
        for(j=0;j<n;j++){
                if(arr[i]<arr[j]){
                        temp = arr[i];
                        arr[i] = arr[j];
                        arr[j] = temp;
                        temp = bur[j];
                        bur[i] = bur[j];
                        bur[j] = temp;
                        strcpy(t,pn[i]);
                        strcpy(pn[i],pn[j]);
                        strcpy(pn[j],t);
                        }
        }
        }
        for(i=0;i<n;i++){
                if(i==0){
                        star[i] = arr[i];
                }else{
                        star[i] = finish[i-1];
                }
                star[i] = arr[i];
                wt[i] = star[i] - arr[i];
                finish[i] = star[i] + bur[i];
                tat[i] = finish[i] - arr[i];
        }

        printf("\n Pname Arrtime Burtime Waittime Start TAT Finish");
        for(i=0;i<n;i++){
                printf("\n%s \t%3d \t%3d \t%3d \t%3d \t%6d \t%6d", pn[i], bur[i], wt[i], star[i], tat[i],
finish[i]);
                totwt += wt[i];
                tottat += tat[i];
        }
        printf("\n Average Waiting Time: %f", (float)totwt);
        printf("\n Average Turn Around Time: %f", (float)tottat);
        return 0;
}
```

Output:



```
Enter total number of processes (maximum 20): 5

Enter the process burst time and arrival time:
P[1] Burst Time:4
P[1] Arrival Time:2
P[2] Burst Time:3
P[2] Arrival Time:2
P[3] Burst Time:1
P[3] Arrival Time:2
P[4] Burst Time:3
P[4] Arrival Time:4
P[5] Burst Time:5
P[5] Arrival Time:6

Process Burst Time       Arrival Time     Waiting Time     Turnaround Time
P[1]     4               2                0                4
P[2]     3               2                2                5
P[3]     1               2                5                6
P[4]     3               4                4                7
P[5]     5               6                5                10

Average Turnaround Time: 6
Average Waiting Time: 3
```

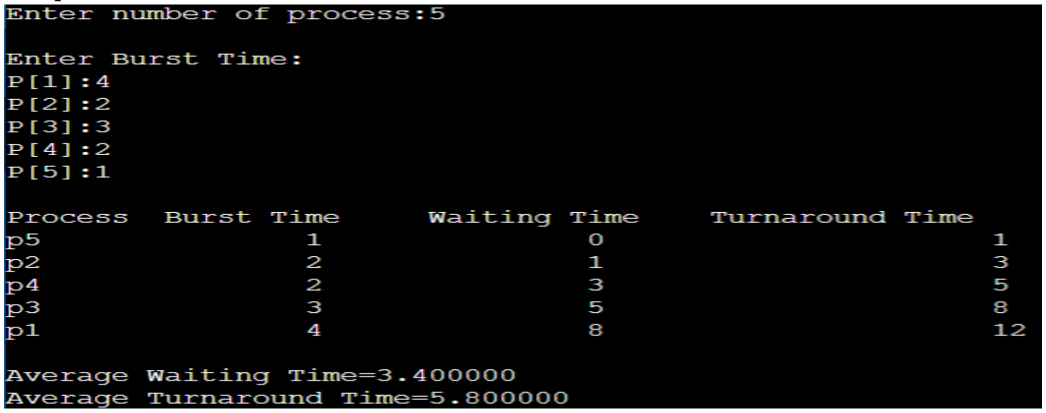
Program 3: Write a program demonstrating use of SJF Algorithm for programs having same arrival time

Solution:

```
#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("Enter 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; }
  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");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        printf("\nP%d\t\t%d\t\t%d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);}
    avg_tat = (float) total / n;
    printf("\n\nAverage Waiting Time = %f", avg_wt);
    printf("\nAverage Turnaround Time = %f\n", avg_tat);
    return 0;}
```

Output:



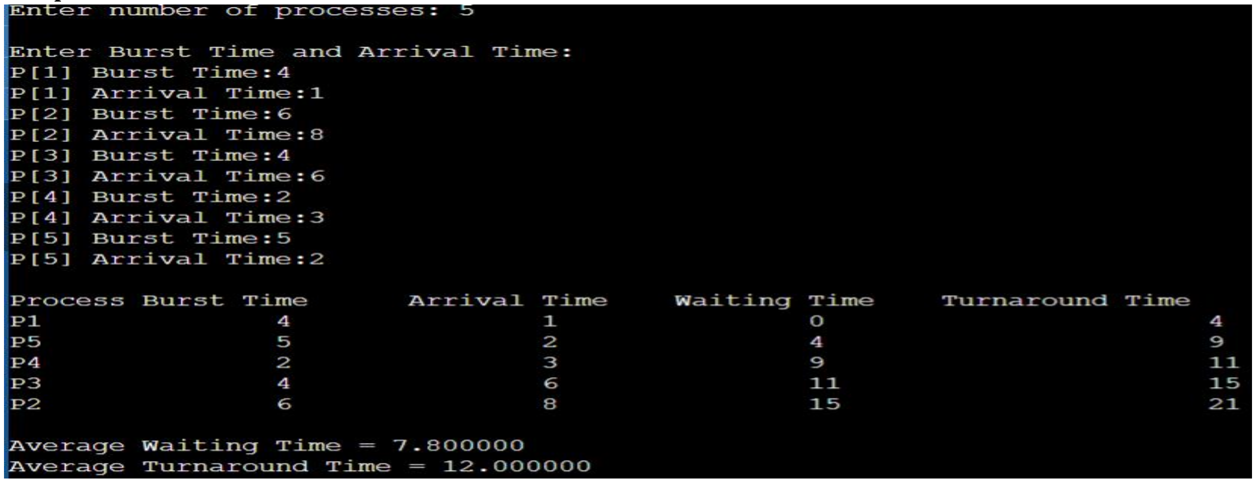
```
Enter number of process:5

Enter Burst Time:
P[1]:4
P[2]:2
P[3]:3
P[4]:2
P[5]:1

Process   Burst Time      Waiting Time     Turnaround Time
p5                1                0                         1
p2                2                1                         3
p4                2                3                         5
p3                3                5                         8
p1                4                8                         12

Average Waiting Time=3.400000
Average Turnaround Time=5.800000
```

Program 4: Write a program demonstrating use of SJF Algorithm for programs having different arrival time

Solution:

```
#include<stdio.h>
int main() {
    int bt[20], at[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("\nEnter Burst Time and Arrival Time:\n");
    for (i = 0; i < n; i++) {
        printf("P[%d] Burst Time:", i + 1);
        scanf("%d", &bt[i]);
        printf("P[%d] Arrival Time:", i + 1);
        scanf("%d", &at[i]);
        p[i] = i + 1;}
    for (i = 0; i < n; i++) {
        pos = i;
        for (j = i + 1; j < n; j++) {
            if (at[j] < at[pos] || (at[j] == at[pos] && bt[j] < bt[pos]))
                pos = j; }
        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;
        temp = at[i];
        at[i] = at[pos];
        at[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\tArrival Time\tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t\t%d", p[i], bt[i], at[i], wt[i], tat[i]);}
    avg_tat = (float) total / n;
    printf("\n\nAverage Waiting Time = %f", avg_wt);
    printf("\nAverage Turnaround Time = %f\n", avg_tat);
    return 0;}
```

Output:



```
Enter number of processes: 5

Enter Burst Time and Arrival Time:
P[1] Burst Time:4
P[1] Arrival Time:1
P[2] Burst Time:6
P[2] Arrival Time:8
P[3] Burst Time:4
P[3] Arrival Time:6
P[4] Burst Time:2
P[4] Arrival Time:3
P[5] Burst Time:5
P[5] Arrival Time:2

Process Burst Time       Arrival Time     Waiting Time     Turnaround Time
P1              4                1                0                        4
P5              5                2                4                        9
P4              2                3                9                       11
P3              4                6               11                       15
P2              6                8               15                       21

Average Waiting Time = 7.800000
Average Turnaround Time = 12.000000
```

Program 5: Write a program demonstrating use of SJF Algorithm for programs having different arrival time

Solution:

```
#include<stdio.h>
int main() {
   int bt[20], at[20], p[20], wt[20], tat[20], priority[20], i, j, n, total = 0, pos, temp;
   float avg_wt, avg_tat;
   printf("Enter number of processes: ");
   scanf("%d", &n);
   printf("\nEnter Burst Time, Arrival Time, and Priority:\n");
   for (i = 0; i < n; i++) {
      printf("P[%d] Burst Time:", i + 1);
      scanf("%d", &bt[i]);
      printf("P[%d] Arrival Time:", i + 1);
      scanf("%d", &at[i]);
      printf("P[%d] Priority:", i + 1);
      scanf("%d", &priority[i]);
      p[i] = i + 1;
   }
   // Sorting based on Arrival Time and Priority
   for (i = 0; i < n; i++) {
      pos = i;
      for (j = i + 1; j < n; j++) {
         if (at[j] < at[pos] || (at[j] == at[pos] && priority[j] < priority[pos]))
            pos = j;
      }
      temp = bt[i];
      bt[i] = bt[pos];
      bt[pos] = temp;
      temp = at[i];
      at[i] = at[pos];
      at[pos] = temp;
      temp = priority[i];
      priority[i] = priority[pos];
      priority[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\tArrival Time\tPriority\tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        printf("\nP%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t\t%d", p[i], bt[i], at[i], priority[i], wt[i], tat[i]);
    }
    avg_tat = (float) total / n;
    printf("\n\nAverage Waiting Time = %f", avg_wt);
    printf("\nAverage Turnaround Time = %f\n", avg_tat);
    return 0;
}
```

CONCLUSION:

In conclusion, this lab equipped us with a practical understanding of the trade-offs and nuances associated with different scheduling algorithms. The dynamic nature of real-world systems requires careful consideration of factors such as arrival times and priorities to design efficient and responsive systems. As we move forward in the study of operating systems, these fundamental concepts will serve as a solid foundation for tackling more advanced scheduling challenges.