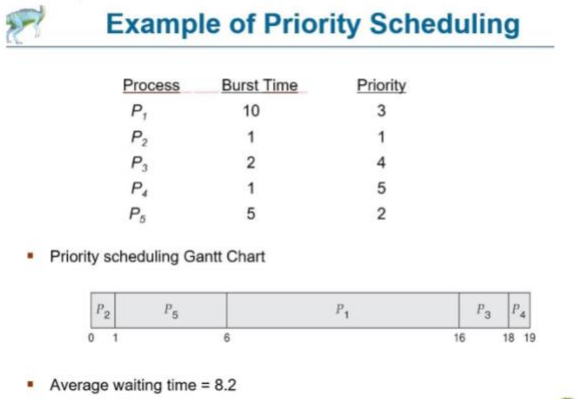
OPERATING SYSTEM LAB 4b

Roll No.: K041	Name: Anish Sudhan Nair
Batch No.: A2/K2	Date: 17/01/2021

Aim: To familiarise and implement the priority scheduling algorithm.

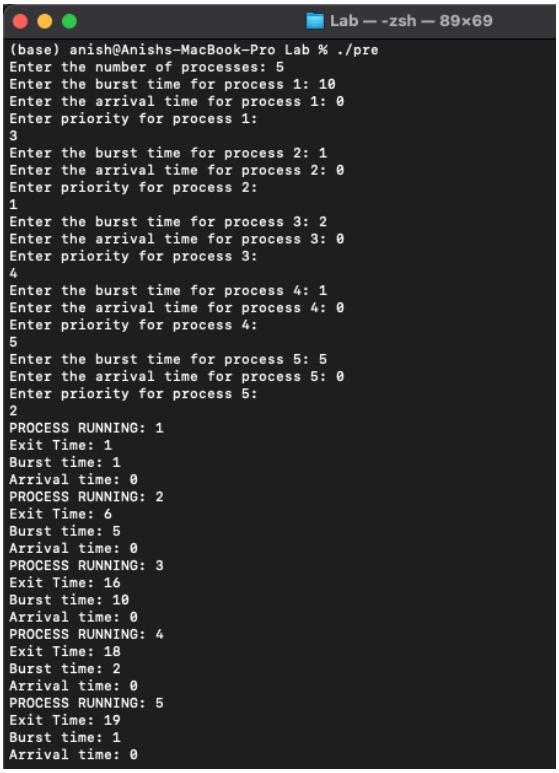
1. Example 1

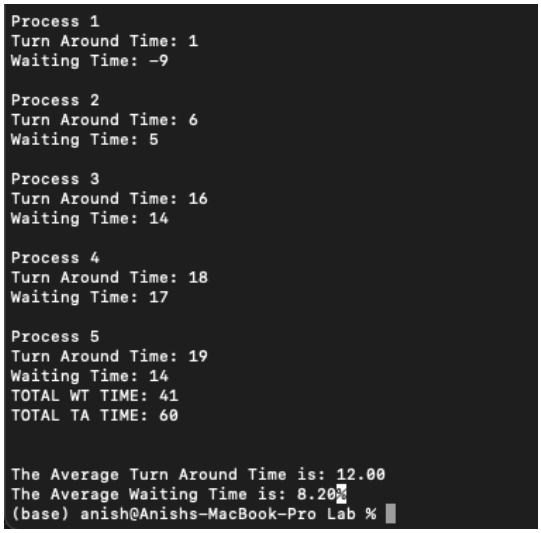
Example 1:



Output:

Pre-emptive





Non Pre-emptive

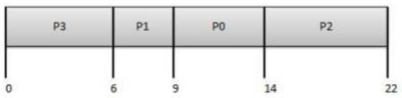
```
Lab — -zsh — 89×68
(base) anish@Anishs-MacBook-Pro Lab % clang priority.c -o p
(base) anish@Anishs-MacBook-Pro Lab % ./p
Enter the number of processes: 5
Enter the burst time for process 1: 10
Enter the arrival time for process 1: 0
Enter priority for process 1: 3
Enter the burst time for process 2: 1
Enter the arrival time for process 2: 0
Enter priority for process 2: 1
Enter the burst time for process 3: 2
Enter the arrival time for process 3: 0
Enter priority for process 3: 4
Enter the burst time for process 4: 1
Enter the arrival time for process 4: 0
Enter priority for process 4: 5
Enter the burst time for process 5: 5
Enter the arrival time for process 5: 0
Enter priority for process 5: 2

PROCESS RUNNING: 2
Exit Time: 1
Burst time: 1
Arrival time: 0

PROCESS RUNNING: 5
Exit Time: 6
Burst time: 5
Arrival time: 0

PROCESS RUNNING: 1
Exit Time: 16
Burst time: 10
Arrival time: 0

PROCESS RUNNING: 3
Exit Time: 18
Burst time: 2
Arrival time: 0

PROCESS RUNNING: 4
Exit Time: 19
Burst time: 1
Arrival time: 0
```

```
Process 1
Turn Around Time: 1
Waiting Time: 0

Process 2
Turn Around Time: 6
Waiting Time: 1

Process 3
Turn Around Time: 16
Waiting Time: 6

Process 4
Turn Around Time: 18
Waiting Time: 16

Process 5
Turn Around Time: 19
Waiting Time: 18


The Average Turn Around Time is: 12.00
The Average Waiting Time is: 8.20
(base) anish@Anishs-MacBook-Pro Lab %
```

2. Example 2

Example 2:

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	3
P2	2	8	1	8
P3	3	6	3	16



Wait time of each process is following

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

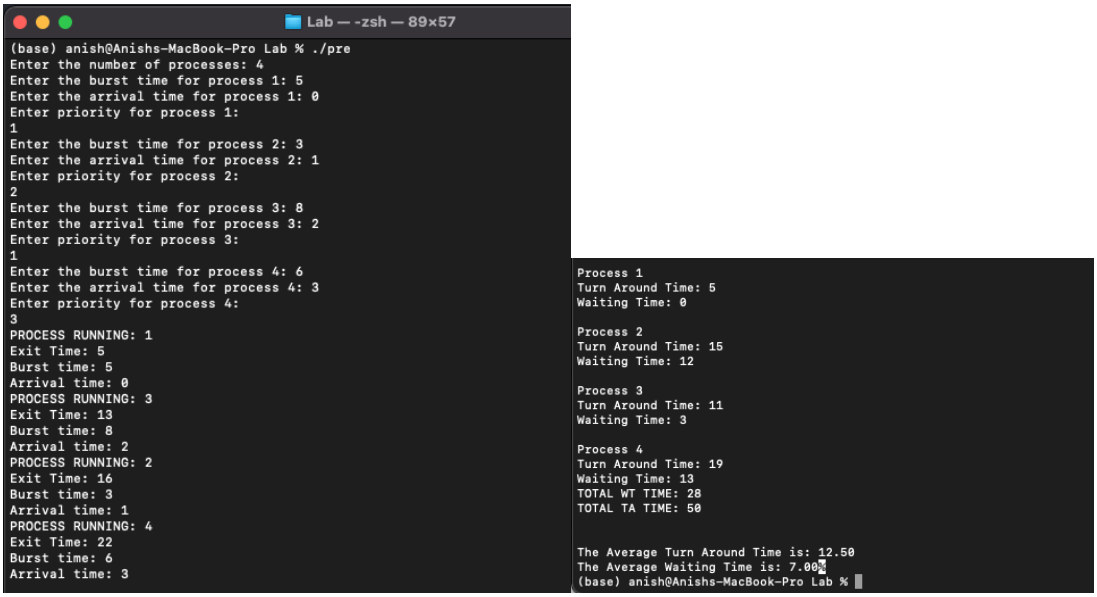
Formula Taught:

Turnaround Time: Exit Time(prev) + Arrival Time

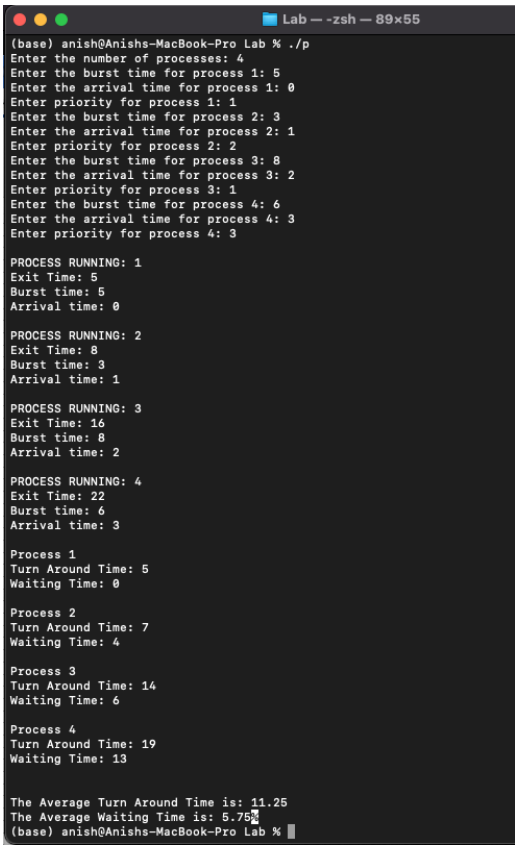
Waiting Time: Turnaround Time – Burst Time

Output:

Pre-emptive



Non Pre-emptive



CODE:

Pre-emptive

```c
#include <stdio.h>
#include <stdbool.h>
int num_process, processes[10], exit_times[10], arrival_times[20], waiting_times[20];
int burst_times[20], turnAround_times[20], priority[20], new_priority[20], process_id[20];
int temp, temp2, og_burst_times[20], final_exit_times[20], og_arrival_times[20], iterator=0;;
//pre-emtive priority scheduking

int exitTime(int i){
  if (i==0)
    return (arrival_times[i] + burst_times[i]);
  else{
    return (exit_times[i-1] + burst_times[i]);}
}

int turnAroundTime(int i){
  return final_exit_times[i]-og_arrival_times[i];
}

int waitingTime(int i){
  return turnAround_times[i] - og_burst_times[i];
}

void avgTime(){
  int totalWaitingTime=0, totalTurnAroundTime=0;
  for (int i = 0; i < num_process; i++) {
    totalWaitingTime+=waiting_times[i];
    totalTurnAroundTime+=turnAround_times[i];
  }
  printf("TOTAL WT TIME: %d\n", totalWaitingTime);
  printf("TOTAL TA TIME: %d\n", totalTurnAroundTime );
  float avgWaitingTime= (float)totalWaitingTime/(float)num_process;
  float avgTurnAroundTime=(float)totalTurnAroundTime/(float)num_process;

  printf("\n\nThe Average Turn Around Time is: %.2f\n", avgTurnAroundTime );
  printf("The Average Waiting Time is: %.2f",avgWaitingTime );
}

void makeSpace(int* array, int count){

  int temp=array[count], temp2=0;

  for (int i = count+1; i < 20; i++) {
    temp2=array[i];
    array[i]=temp;
    temp=temp2;
  }
}

void removeSpace(int* array, int count){

  for (int i = count; i < (20-count); i++) {
    array[i]=array[i+1];
  }
}

bool zeroes(int* array, int posn){
  int count=0;
  for (int i = 0; i < 3; i++) {
    if (array[i]==0) {
      count++;
```

```
    }
  }
  if (count==3) {
    return false;
  }
  return true;
}

int main(){

  printf("Enter the number of processes: " );
  scanf("%d",&num_process);


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

    printf("Enter the burst time for process %d: ",i+1 );
    scanf("%d", &burst_times[i] );

    og_burst_times[i]=burst_times[i];

    printf("Enter the arrival time for process %d: ",i+1 );
    scanf("%d", &arrival_times[i] );

    og_arrival_times[i]=arrival_times[i];

    printf("Enter priority for process %d: \n",i+1 );
    scanf("%d", &priority[i] );

    process_id[i]=i+1;

  }

for (int j = 0; j < num_process; j++) {
  for (int i = j; i < num_process; i++) {
    if (arrival_times[i]<arrival_times[j]) {
      temp=arrival_times[i];
      arrival_times[i]=arrival_times[j];
      arrival_times[j]=temp;

      temp2=burst_times[i];
      burst_times[i]=burst_times[j];
      burst_times[j]=temp2;
    }
    if ((arrival_times[i]==arrival_times[j])&&priority[i]<priority[j]) {
      temp=arrival_times[i];
      arrival_times[i]=arrival_times[j];
      arrival_times[j]=temp;

      temp2=burst_times[i];
      burst_times[i]=burst_times[j];
      burst_times[j]=temp2;

      temp2=priority[i];
      priority[i]=priority[j];
      priority[j]=temp2;
    }
  }
}
```

```
121   for (int i = 0; i < num_process; i++) {
122     exit_times[i]=exitTime(i);
123   }
124 
125   int og_burst_time=0;
126 
127   for (int i = 0; i < 20; i++) {
128       if ((exit_times[i]>arrival_times[i+1])&&(priority[i]>priority[i+1])&&(burst_times[i]!=0)&&(burst_times[i+1]!=0)) {
129 
130         makeSpace(&burst_times, i+2);
131         makeSpace(&exit_times, i+2);
132         makeSpace(&arrival_times, i+2);
133         makeSpace(&priority, i+2);
134         makeSpace(&process_id, i+2);
135 
136         og_burst_time=burst_times[i];
137         if ((arrival_times[i+1]-exit_times[i-1])<0) {
138           burst_times[i]=0;
139         } else {
140           burst_times[i]=arrival_times[i+1]-exit_times[i-1];
141         }
142         if (burst_times[i]==0) {
143           burst_times[i+2]=og_burst_time;
144         } else {
145           burst_times[i+2]=og_burst_time-burst_times[i];
146         }
147         arrival_times[i+2]=arrival_times[i];
148         priority[i+2]=priority[i];
149         process_id[i+2]=process_id[i];
150         exit_times[i]=exitTime(i);
151 
152         if (burst_times[i]==0) {
153           removeSpace(&burst_times, i);
154           removeSpace(&exit_times, i);
155           removeSpace(&arrival_times, i);
156           removeSpace(&priority, i);
157           removeSpace(&process_id, i);
158 
159           exit_times[i]=exitTime(i);
160         }
161         iterator++;
162     }
163     else if((burst_times[i]!=0)){
164       exit_times[i]=exitTime(i);
165       iterator++;
166     }
167   }
168 
169   for (int i = 0; i < num_process; i++) {
170     for (int j = 0; j < 6; j++) {
171       if (process_id[j]==(i+1)) {
172         final_exit_times[i]=exit_times[j];
173       }
174     }
175   }
176 
177   for (int i = 0; i < iterator; i++) {
178 
179     printf("PROCESS RUNNING: %d\n",process_id[i] );
180 
```

```
    printf("Exit Time: %d\n", exit_times[i]);

    printf("Burst time: %d\n",burst_times[i]);

    printf("Arrival time: %d\n",arrival_times[i]);

  }

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

      turnAround_times[i]=turnAroundTime(i);

      waiting_times[i]=waitingTime(i);

      printf("\nProcess %d\n",i+1);

      printf("Turn Around Time: %d\n", turnAround_times[i]);

      printf("Waiting Time: %d\n",waiting_times[i]);

    }

    avgTime();

    return 0;
  }

```

Non Pre-emptive

```c
#include <stdio.h>
int num_process, processes[10], exit_times[10], arrival_times[10], waiting_times[10];
int burst_times[10], turnAround_times[10], priority[10];
int temp, temp2, process_id[10];
//non preemptive priority

int exitTime(int i){
  if ((i==0) || (arrival_times[i]>exit_times[i-1]))
    return (arrival_times[i] + burst_times[i]);
  else
    return (exit_times[i-1] + burst_times[i]);
}

int turnAroundTime(int i){
  return exit_times[i]-arrival_times[i];
}

int waitingTime(int i){
  return turnAround_times[i] - burst_times[i];
}


void avgTime(){
  int totalWaitingTime=0, totalTurnAroundTime=0;
  for (int i = 0; i < num_process; i++) {
    totalWaitingTime+=waiting_times[i];
    totalTurnAroundTime+=turnAround_times[i];
  }
  float avgWaitingTime= (float)totalWaitingTime/(float)num_process;
  float avgTurnAroundTime=(float)totalTurnAroundTime/(float)num_process;

  printf("\n\nThe Average Turn Around Time is: %.2f\n", avgTurnAroundTime );
  printf("The Average Waiting Time is: %.2f",avgWaitingTime );
}

int main(){

  printf("Enter the number of processes: " );
  scanf("%d",&num_process);


  for (int i = 0; i < num_process; i++) {
    printf("Enter the burst time for process %d: ",i+1 );
    scanf("%d", &burst_times[i] );

    printf("Enter the arrival time for process %d: ",i+1 );
    scanf("%d", &arrival_times[i] );

    printf("Enter priority for process %d: ",i+1 );
    scanf("%d", &priority[i] );

    process_id[i]=i+1;

  }

for (int j = 0; j < num_process; j++) {
  for (int i = j; i < num_process; i++) {
    if (arrival_times[i]<arrival_times[j]) {
      temp=arrival_times[i];
      arrival_times[i]=arrival_times[j];
```

```
        arrival_times[j]=temp;

        temp2=burst_times[i];
        burst_times[i]=burst_times[j];
        burst_times[j]=temp2;

        temp2=priority[i];
        priority[i]=priority[j];
        priority[j]=temp2;

        temp2=process_id[i];
        process_id[i]=process_id[j];
        process_id[j]=temp2;
      }
      if ((arrival_times[i]==arrival_times[j])&&priority[i]<priority[j]) {
        temp=arrival_times[i];
        arrival_times[i]=arrival_times[j];
        arrival_times[j]=temp;

        temp2=burst_times[i];
        burst_times[i]=burst_times[j];
        burst_times[j]=temp2;

        temp2=priority[i];
        priority[i]=priority[j];
        priority[j]=temp2;

        temp2=process_id[i];
        process_id[i]=process_id[j];
        process_id[j]=temp2;
      }
      if ((arrival_times[i]==arrival_times[j])&&(priority[i]==priority[j])&&(process_id[i]<process_id[j])) {
        temp=arrival_times[i];
        arrival_times[i]=arrival_times[j];
        arrival_times[j]=temp;

        temp2=burst_times[i];
        burst_times[i]=burst_times[j];
        burst_times[j]=temp2;

        temp2=priority[i];
        priority[i]=priority[j];
        priority[j]=temp2;

        temp2=process_id[i];
        process_id[i]=process_id[j];
        process_id[j]=temp2;
      }
    }
  }

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

      exit_times[i]=exitTime(i);

      turnAround_times[i]=turnAroundTime(i);

      waiting_times[i]=waitingTime(i);
    }

```

```
121     for (int i = 0; i < num_process; i++) {
122
123       printf("\nPROCESS RUNNING: %d\n",process_id[i] );
124
125       printf("Exit Time: %d\n", exit_times[i]);
126
127       printf("Burst time: %d\n",burst_times[i]);
128
129       printf("Arrival time: %d\n",arrival_times[i]);
130
131     }
132
133       for (int i = 0; i < num_process; i++) {
134
135         printf("\nProcess %d\n",i+1);
136
137         printf("Turn Around Time: %d\n", turnAround_times[i]);
138
139         printf("Waiting Time: %d\n",waiting_times[i]);
140
141       }
142
143     avgTime();
144
145     return 0;
146   }
147
```

CONCLUSION:

In this lab, we were to implement and demonstrate the working of the priority scheduling algorithm (pre-emptive and non pre-emptive). By actually coding the algorithm, it helped to reinforce the working of this process and the manner in which it schedules the processes in a CPU.