Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune-37

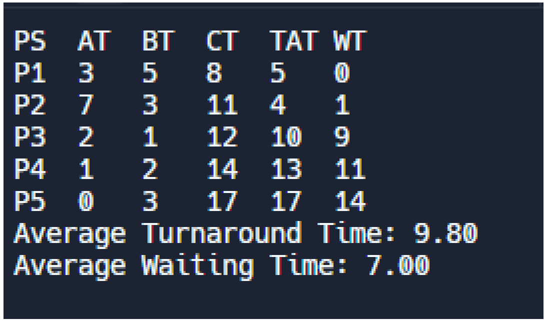
(Anautonomous Institute of Savitribai Phule Pune University)

Department of Multidisciplinary Engineering

Division	**CS-A**
Batch	**B1**
Roll no.	**90**
Name	**Aditya Shrinivas Kurapati**

1. FCFS :-

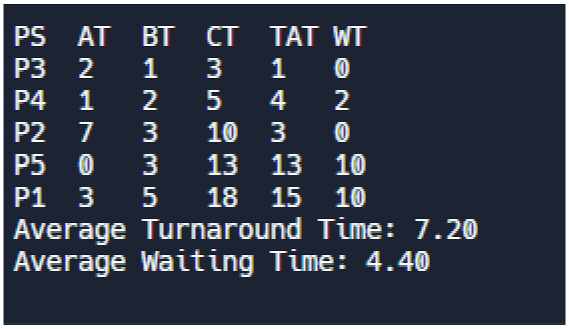
```
#include <stdio.h>

struct ps {
  char p_name[20];
  int at, bt, ct, tat, wt, flag;
};

struct queue {
  int front, rear;
  struct ps a[100];
};

int at[] = {3, 7, 2, 1, 0};
int bt[] = {5, 3, 1, 2, 3};

int n = 5;

int main() {
  struct ps p[n];
  float avg_tat, avg_wt;
  int done = 0;
  int time = 0;
  int ps = 0;

  for (int i = 0; i < n; i++) {
    sprintf(p[i].p_name, "P%d", i + 1);
    p[i].at = at[i];
    p[i].bt = bt[i];
    p[i].flag = 0;
  }

  while (!done) {
    done = 1;
    for (int i = 0; i < n; i++) {
      if (p[i].flag == 0) {
        done = 0;
        if (time >= p[i].at) {
          p[i].flag = 1;
          p[i].ct = time + p[i].bt;
          time = p[i].ct;
        } else {
          time = p[i].at;
          p[i].flag = 1;
          p[i].ct = time + p[i].bt;
          time = p[i].ct;
        }
      }
    }
  }
  for (int i = 0; i < n; i++) {
    p[i].tat = p[i].ct - p[i].at;
    p[i].wt = p[i].tat - p[i].bt;
  }

  // Calculate average turnaround time and average waiting time
  float total_tat = 0, total_wt = 0;
  for (int i = 0; i < n; i++) {
    total_tat += p[i].tat;
    total_wt += p[i].wt;
  }
  avg_tat = total_tat / n;
  avg_wt = total_wt / n;

  printf("PS\tAT\tBT\tCT\tTAT\tWT\n");
  for (int i = 0; i < n; i++) {
    printf("%s\t%d\t%d\t%d\t%d\t%d\n", p[i].p_name, p[i].at, p[i].bt, p[i].ct,
           p[i].tat, p[i].wt);
  }

  printf("Average Turnaround Time: %.2f\n", avg_tat);
  printf("Average Waiting Time: %.2f\n", avg_wt);

  return 0;
      }
```



```
PS  AT  BT  CT  TAT WT
P1  3   5   8   5   0
P2  7   3   11  4   1
P3  2   1   12  10  9
P4  1   2   14  13  11
P5  0   3   17  17  14
Average Turnaround Time: 9.80
Average Waiting Time: 7.00
```

2. SJF Non-Preemptive :-

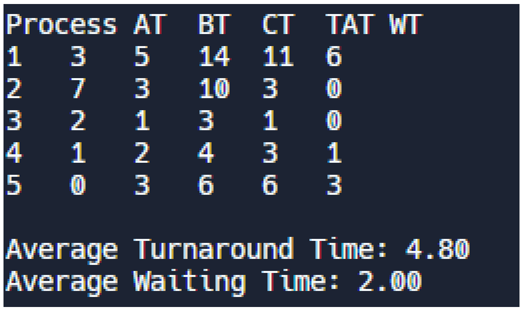
```
#include <stdio.h>

struct ps {
  char p_name[20];
  int at, bt, ct, tat, wt, flag;
};

struct queue {
  int front, rear;
  struct ps a[100];
};

int at[] = {3, 7, 2, 1, 0};
int bt[] = {5, 3, 1, 2, 3};
int n = 5;

void swap(struct ps *xp, struct ps *yp) {
  struct ps temp = *xp;
  *xp = *yp;
  *yp = temp;
}

void sort(struct ps p[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (p[j].bt > p[j + 1].bt) {
        swap(&p[j], &p[j + 1]);
      }
    }
  }
}

int main() {
  struct ps p[n];
  float avg_tat, avg_wt;
  int time = 0;

  for (int i = 0; i < n; i++) {
    sprintf(p[i].p_name, "P%d", i + 1);
    p[i].at = at[i];
    p[i].bt = bt[i];
    p[i].flag = 0;
  }

  sort(p, n);

  for (int i = 0; i < n; i++) {
    if (time < p[i].at) { // Process arrives after current time
      time = p[i].at;
    }
    p[i].ct = time + p[i].bt;
    time = p[i].ct;
  }

  for (int i = 0; i < n; i++) {
    p[i].tat = p[i].ct - p[i].at;
    p[i].wt = p[i].tat - p[i].bt;
    if (p[i].tat < 0) {
      p[i].tat = 0; // Turnaround time cannot be negative
    }
    if (p[i].wt < 0) {
      p[i].wt = 0; // Waiting time cannot be negative
    }
  }
  // Calculate average turnaround time and average waiting time
  float total_tat = 0, total_wt = 0;
  for (int i = 0; i < n; i++) {
    total_tat += p[i].tat;
    total_wt += p[i].wt;
  }
  avg_tat = total_tat / n;
  avg_wt = total_wt / n;

  printf("PS\tAT\tBT\tCT\tTAT\tWT\n");
  for (int i = 0; i < n; i++) {
    printf("%s\t%d\t%d\t%d\t%d\t%d\n", p[i].p_name, p[i].at, p[i].bt, p[i].ct,
           p[i].tat, p[i].wt);
  }

  printf("Average Turnaround Time: %.2f\n", avg_tat);
  printf("Average Waiting Time: %.2f\n", avg_wt);

  return 0;
      }
```



```
PS  AT  BT  CT  TAT WT
P3  2   1   3   1   0
P4  1   2   5   4   2
P2  7   3   10  3   0
P5  0   3   13  13  10
P1  3   5   18  15  10
Average Turnaround Time: 7.20
Average Waiting Time: 4.40
```

3. SJF Preemptive :-

```
#include <limits.h>
#include <stdio.h>
#define MAX 5

struct ps {
  int name;
  int arrival_time;
  int burst_time;
  int completion_time;
  int turnaround_time;
  int waiting_time;
};

int main() {
  struct ps ps[MAX];
  int n = 5; // Number of processes
  int at[] = {3, 7, 2, 1, 0};
  int bt[] = {5, 3, 1, 2, 3};
  int i, total_burst_time = 0;

  for (i = 0; i < n; i++) {
    ps[i].name = i + 1;
    ps[i].arrival_time = at[i];
    ps[i].burst_time = bt[i];
    total_burst_time += ps[i].burst_time;
    ps[i].completion_time = -1;
    ps[i].turnaround_time = -1;
    ps[i].waiting_time = -1;
  }

  int current_time = 0;
  int completed_ps = 0;

  while (completed_ps < n) {
    int smallest_burst_index = -1;
    int smallest_burst = INT_MAX;

    for (i = 0; i < n; i++) {
      if (ps[i].arrival_time <= current_time && ps[i].completion_time == -1 &&
          ps[i].burst_time < smallest_burst) {
        smallest_burst = ps[i].burst_time;
        smallest_burst_index = i;
      }
    }

    if (smallest_burst_index != -1) {
      current_time++;
      ps[smallest_burst_index].burst_time--;

      if (ps[smallest_burst_index].burst_time == 0) {
        ps[smallest_burst_index].completion_time = current_time;
        completed_ps++;
      }
    } else {
      current_time++;
    }
  }

  // Calculating turnaround time and waiting time
  for (i = 0; i < n; i++) {
    ps[i].turnaround_time = ps[i].completion_time - ps[i].arrival_time;
    ps[i].waiting_time = ps[i].turnaround_time - bt[i];
  }

  // Displaying process information
  printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
  for (i = 0; i < n; i++) {
    printf("%d\t%d\t%d\t%d\t%d\t%d\n", ps[i].name, ps[i].arrival_time, bt[i],
          ps[i].completion_time, ps[i].turnaround_time, ps[i].waiting_time);
  }

  // Calculating averages
  float avg_turnaround_time = 0, avg_waiting_time = 0;
  for (i = 0; i < n; i++) {
    avg_turnaround_time += ps[i].turnaround_time;
    avg_waiting_time += ps[i].waiting_time;
  }
  avg_turnaround_time /= n;
  avg_waiting_time /= n;

  printf("\nAverage Turnaround Time: %.2f\n", avg_turnaround_time);
  printf("Average Waiting Time: %.2f\n", avg_waiting_time);

  return 0;
     }
```



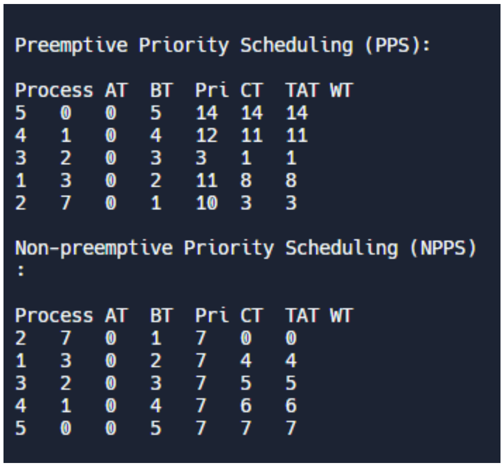
```
Process AT  BT  CT  TAT WT
1   3   5   14  11  6
2   7   3   10  3   0
3   2   1   3   1   0
4   1   2   4   3   1
5   0   3   6   6   3

Average Turnaround Time: 4.80
Average Waiting Time: 2.00
```

4. Priorty Preemtive And Non-Preemptive:-

```
#include <limits.h>
#include <stdio.h>

#define MAX 5

struct Process {
  int name;
  int arrival_time;
  int burst_time;
  int priority;
  int completion_time;
  int turnaround_time;
  int waiting_time;
};

void swap(struct Process *a, struct Process *b) {
  struct Process temp = *a;
  *a = *b;
  *b = temp;
}

void sort_by_arrival_time(struct Process processes[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (processes[j].arrival_time > processes[j + 1].arrival_time) {
        swap(&processes[j], &processes[j + 1]);
      }
    }
  }
}

void sort_by_priority(struct Process processes[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (processes[j].priority > processes[j + 1].priority) {
        swap(&processes[j], &processes[j + 1]);
      }
    }
  }
}

void calculate_completion_time(struct Process processes[], int n) {
  int current_time = 0;
  for (int i = 0; i < n; i++) {
    if (current_time < processes[i].arrival_time) {
      current_time = processes[i].arrival_time;
    }
    current_time += processes[i].burst_time;
    processes[i].completion_time = current_time;
  }
}

void calculate_turnaround_time(struct Process processes[], int n) {
  for (int i = 0; i < n; i++) {
    processes[i].turnaround_time =
        processes[i].completion_time - processes[i].arrival_time;
  }
}

void calculate_waiting_time(struct Process processes[], int n) {
  for (int i = 0; i < n; i++) {
    processes[i].waiting_time =
        processes[i].turnaround_time - processes[i].burst_time;
  }
}

void display_process_info(struct Process processes[], int n) {
  printf("\nProcess\tAT\tBT\tPri\tCT\tTAT\tWT\n");
  for (int i = 0; i < n; i++) {
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i].name,
           processes[i].arrival_time, processes[i].burst_time,
           processes[i].priority, processes[i].completion_time,
           processes[i].turnaround_time, processes[i].waiting_time);
  }
```

```
}

int main() {
  struct Process processes[MAX];
  int n = MAX;
  int at[] = {3, 7, 2, 1, 0};
  int bt[] = {5, 3, 1, 2, 3};
  int priority[] = {2, 1, 3, 4, 5};

  for (int i = 0; i < n; i++) {
    processes[i].name = i + 1;
    processes[i].arrival_time = at[i];
    processes[i].burst_time = bt[i];
    processes[i].priority = priority[i];
    processes[i].completion_time = -1;
    processes[i].turnaround_time = -1;
    processes[i].waiting_time = -1;
  }

  // Sort processes by arrival time
  sort_by_arrival_time(processes, n);

  // Implement Priority Scheduling (Preemptive)
  int current_time = 0;
  int completed_processes = 0;
  while (completed_processes < n) {
    int highest_priority_index = -1;
    int highest_priority = INT_MAX;
    for (int i = 0; i < n; i++) {
      if (processes[i].arrival_time <= current_time &&
          processes[i].burst_time > 0) {
        if (processes[i].priority < highest_priority) {
          highest_priority = processes[i].priority;
          highest_priority_index = i;
        }
      }
    }
    if (highest_priority_index != -1) {
      processes[highest_priority_index].burst_time--;
      current_time++;
      if (processes[highest_priority_index].burst_time == 0) {
        processes[highest_priority_index].completion_time = current_time;
        completed_processes++;
      }
    } else {
      current_time++;
    }
  }

  // Calculate turnaround time and waiting time for preemptive priority
  // scheduling
```

```
 calculate_turnaround_time(processes, n);
 calculate_waiting_time(processes, n);

 // Display process information for preemptive priority scheduling
 printf("\nPreemptive Priority Scheduling (PPS):\n");
 display_process_info(processes, n);

 // Sort processes by priority for non-preemptive priority scheduling
 sort_by_priority(processes, n);

 // Calculate completion time for non-preemptive priority scheduling
 calculate_completion_time(processes, n);

 // Calculate turnaround time and waiting time for non-preemptive priority
 // scheduling
 calculate_turnaround_time(processes, n);
 calculate_waiting_time(processes, n);

 // Display process information for non-preemptive priority scheduling
 printf("\nNon-preemptive Priority Scheduling (NPPS):\n");
 display_process_info(processes, n);

 return 0;
    }
```



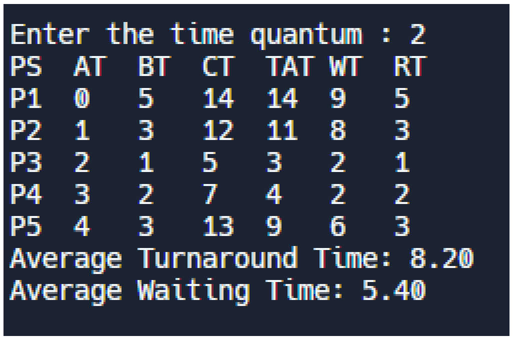
```
Preemptive Priority Scheduling (PPS):

Process AT  BT  Pri CT  TAT WT
5   0   0   5   14  14  14
4   1   0   4   12  11  11
3   2   0   3   3   1   1
1   3   0   2   11  8   8
2   7   0   1   10  3   3

Non-preemptive Priority Scheduling (NPPS)
:

Process AT  BT  Pri CT  TAT WT
2   7   0   1   7   0   0
1   3   0   2   7   4   4
3   2   0   3   7   5   5
4   1   0   4   7   6   6
5   0   0   5   7   7   7
```

5. Round Robin :-

```
#include <stdio.h>

struct ps {
  char p_name[20];
  int at, bt, ct, tat, wt, flag, rt;
};

struct queue {
  int front, rear;
  struct ps a[100];
```

```
};

int at[] = {0, 1, 2, 3, 4};
int bt[] = {5, 3, 1, 2, 3};

int n = 5;

int main() {
  float avg_tat, avg_wt;
  int TQ;
  struct ps p[n];

  for (int i = 0; i < n; i++) {
    sprintf(p[i].p_name, "P%d", i + 1);
    p[i].at = at[i];
    p[i].bt = bt[i];
    p[i].flag = 0;
    p[i].rt = bt[i];
  }

  printf("Enter the time quantum : ");
  scanf("%d", &TQ);

  int time = 0;
  int ps = 0;
  struct queue q;
  q.front = q.rear = -1;

  while (1) {
    int done = 1; // Flag to check if all processes are done

    for (int i = 0; i < n; i++) {
      if (p[i].flag == 0) {
        done = 0; // At least one process is not done
        if (p[i].rt > 0) {
          if (p[i].rt <= TQ) {
            time += p[i].rt;
            p[i].rt = 0;
            p[i].ct = time;
            p[i].tat = p[i].ct - p[i].at;
            p[i].wt = p[i].tat - p[i].bt;
            p[i].flag = 1;
          } else {
            time += TQ;
            p[i].rt -= TQ;
          }
        }
        if (p[i].rt == 0 && q.rear != -1) {
          struct ps temp = q.a[q.front];
          q.front++;
          if (q.front > q.rear) {
```

```
          q.front = q.rear = -1;
        }
        q.a[q.rear + 1] = temp;
        q.rear++;
      }
    }
  }

  if (done)
    break; // All processes are done

  for (int i = ps; i < n; i++) {
    if (p[i].flag == 0 && p[i].at <= time) {
      if (q.rear == -1) {
        q.front = q.rear = 0;
        q.a[q.rear] = p[i];
      } else {
        q.rear++;
        q.a[q.rear] = p[i];
      }
      ps++;
    }
  }
}

// Calculate average turnaround time and average waiting time
float total_tat = 0, total_wt = 0;
for (int i = 0; i < n; i++) {
  total_tat += p[i].tat;
  total_wt += p[i].wt;
}
avg_tat = total_tat / n;
avg_wt = total_wt / n;

printf("PS\tAT\tBT\tCT\tTAT\tWT\tRT\n");
for (int i = 0; i < n; i++) {
  printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].p_name, p[i].at, p[i].bt,
         p[i].ct, p[i].tat, p[i].wt, p[i].bt);
}

printf("Average Turnaround Time: %.2f\n", avg_tat);
printf("Average Waiting Time: %.2f\n", avg_wt);

return 0;
    }
```



```
Enter the time quantum : 2
PS  AT  BT  CT  TAT WT  RT
P1  0   5   14  14  9   5
P2  1   3   12  11  8   3
P3  2   1   5   3   2   1
P4  3   2   7   4   2   2
P5  4   3   13  9   6   3
Average Turnaround Time: 8.20
Average Waiting Time: 5.40
```