VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

B.Tech. Winter Semester 2023-24
School Of Computer Science and Engineering (SCOPE)

Digital Assignment - III

Operating System Lab

Apurva Mishra, 22BCE2791

9 September 2024

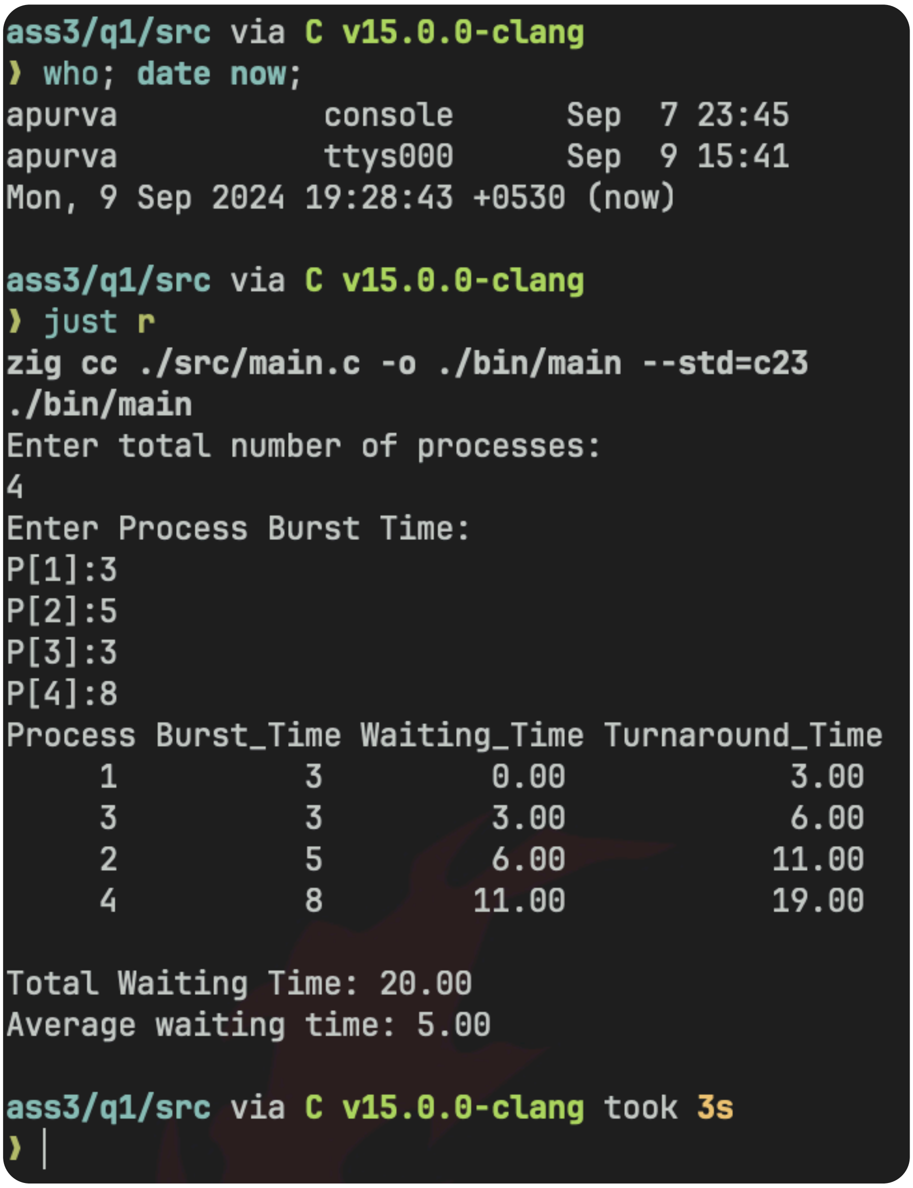
1. Questions

Problem 1.1.
Write a LINUX C Program for the Implementation of shortest remaining time first (SRTF) Scheduling Algorithm.

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

// Job structure
typedef struct {
  int uuid;
  int time; // burst time
} Job;

Job job_new(int uuid, int time) {
  Job job;
  job.uuid = uuid;
  job.time = time;
  return job;
}

// Queue structure
typedef struct {
  int capacity;
  int length;
  Job *jobs;
} Queue;

Queue *queue_new() {
  Queue *queue = malloc(sizeof(Queue));
  Job *jobs = malloc(sizeof(Job) * 10);
  queue->capacity = 5;
  queue->length = 0;
  queue->jobs = jobs;
  return queue;
}

bool queue_is_empty(Queue *queue) {
  if (queue->length <= 0) {
    return true;
  }
  return false;
}

void increase_capacity(Queue *queue) {
  Job *new_jobs = malloc(sizeof(Job) * (queue->capacity + 5));
  for (int i = 0; i < queue->length; i++) {
    new_jobs[i] = queue->jobs[i];
  }
  queue->capacity += 5;
  free(queue->jobs);
  queue->jobs = new_jobs;
}
```

```c
void queue_add_job(Queue *queue, Job job) {
  if (queue->length == queue->capacity) {
    increase_capacity(queue);
  }
  for (int i = 0; i < queue->length; i++) {
    if (queue->jobs[i].time > job.time) {
      for (int j = queue->length; j > i; j--) {
        queue->jobs[j] = queue->jobs[j - 1];
      }
      queue->jobs[i] = job;
      queue->length += 1;
      return;
    }
  }
  queue->jobs[queue->length] = job;
  queue->length += 1;
}

void sort_queue_by_burst_time(Queue *queue) {
  for (int i = 0; i < queue->length - 1; i++) {
    for (int j = 0; j < queue->length - i - 1; j++) {
      if (queue->jobs[j].time > queue->jobs[j + 1].time) {
        // Swap jobs
        Job temp = queue->jobs[j];
        queue->jobs[j] = queue->jobs[j + 1];
        queue->jobs[j + 1] = temp;
      }
    }
  }
}

void input_jobs(Queue *queue) {
  int n_job;
  printf("Enter total number of processes:\n");
  scanf("%d", &n_job);
  printf("Enter Process Burst Time:\n");
  for (int i = 0; i < n_job; i++) {
    int burst_time;
    printf("P[%d]:", i + 1);
    scanf("%d", &burst_time);
    queue_add_job(queue, job_new(i + 1, burst_time));
  }
  sort_queue_by_burst_time(queue);
}

int queue_process(Queue *queue) {
  float total = 0;
  float twaiting = 0;
  printf("Process Burst_Time Waiting_Time Turnaround_Time\n");
  fflush(stdout);
  for (int i = 0; i < queue->length; i++) {
    printf("%6d %10d %12.2f %15.2f\n", queue->jobs[i].uuid, queue->jobs[i].time,
           total, total + queue->jobs[i].time);
    twaiting += total;
    total += queue->jobs[i].time;
  }
  printf("\nTotal Waiting Time: %.2f\n", twaiting);
  printf("Average waiting time: %.2f\n", twaiting / queue->length);
  return total;
}

int main() {
  Queue *queue = queue_new();
  input_jobs(queue);
  queue_process(queue);
  free(queue->jobs);
  free(queue);
  return 0;
}
```

Output



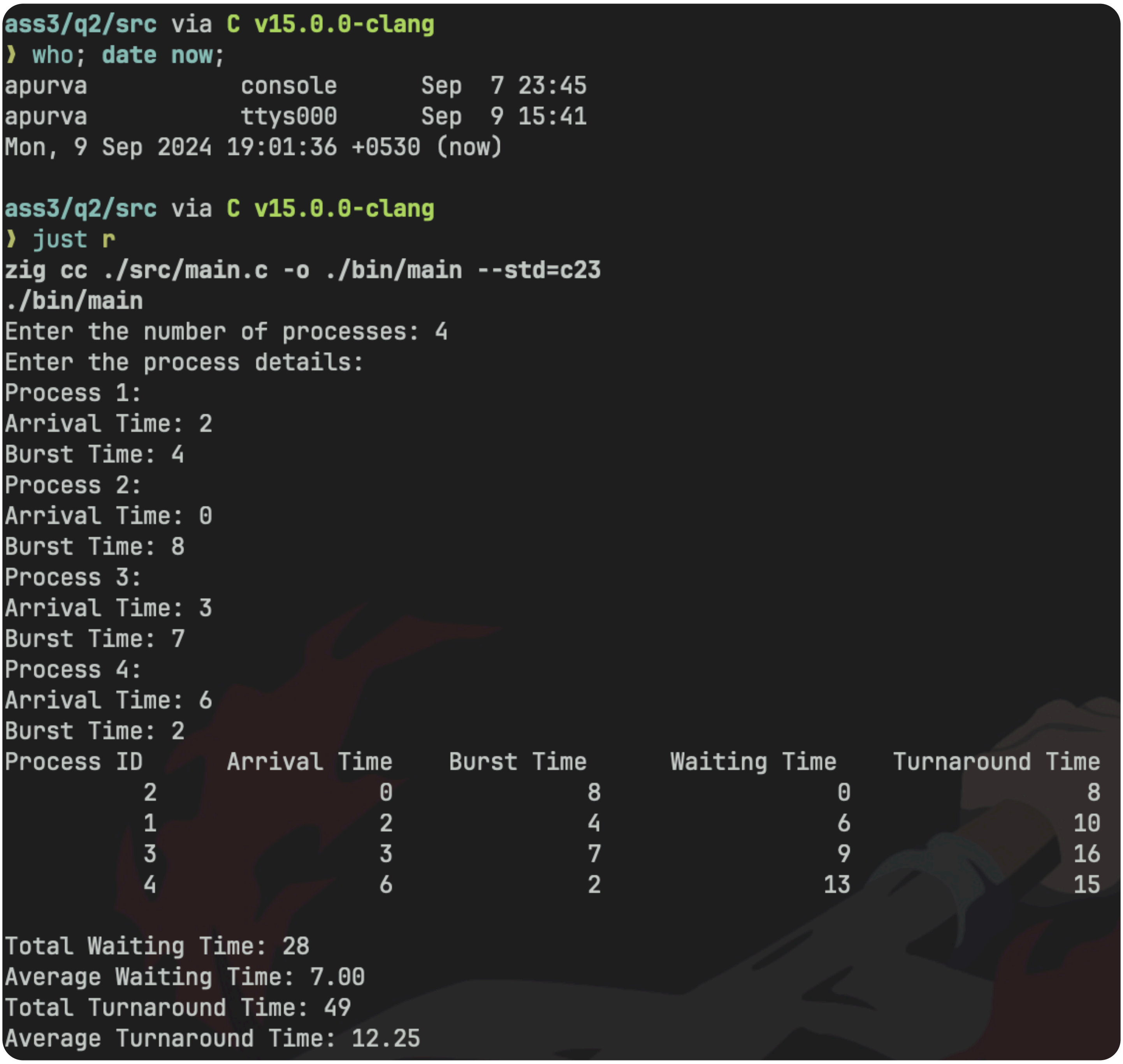
```
ass3/q1/src via C v15.0.0-clang
❯ who; date now;
apurva           console      Sep  7 23:45
apurva           ttys000      Sep  9 15:41
Mon, 9 Sep 2024 19:28:43 +0530 (now)

ass3/q1/src via C v15.0.0-clang
❯ just r
zig cc ./src/main.c -o ./bin/main --std=c23
./bin/main
Enter total number of processes:
4
Enter Process Burst Time:
P[1]:3
P[2]:5
P[3]:3
P[4]:8
Process Burst_Time Waiting_Time Turnaround_Time
     1          3          0.00            3.00
     3          3          3.00            6.00
     2          5          6.00           11.00
     4          8         11.00           19.00

Total Waiting Time: 20.00
Average waiting time: 5.00

ass3/q1/src via C v15.0.0-clang took 3s
❯ |
```

Problem 1.2.
Create a LINUX C program to implement Priority CPU Scheduling with varying arrival times. Processes will be scheduled based on their arrival time and priority.

CPU Scheduling based on Arrival Time

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int id;
  int arrival_time;
  int burst_time;
  int waiting_time;
  int turnaround_time;
} Process;

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

void sortProcesses(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 processQueue(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;
    }

    processes[i].waiting_time = current_time - processes[i].arrival_time;
    processes[i].turnaround_time =
        processes[i].waiting_time + processes[i].burst_time;

    current_time += processes[i].burst_time;
  }
}

int main() {
  int n_processes;
  printf("Enter the number of processes: ");
  scanf("%d", &n_processes);

  Process *processes = (Process *)malloc(n_processes * sizeof(Process));

  printf("Enter the process details:\n");
  for (int i = 0; i < n_processes; i++) {
    processes[i].id = i + 1;
    printf("Process %d:\n", processes[i].id);
    printf("Arrival Time: ");
    scanf("%d", &processes[i].arrival_time);
    printf("Burst Time: ");
    scanf("%d", &processes[i].burst_time);
  }

  sortProcesses(processes, n_processes);

  processQueue(processes, n_processes);

  printf(
      "Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
  int total_waiting_time = 0, total_turnaround_time = 0;
  for (int i = 0; i < n_processes; i++) {
    printf("%11d\t%12d\t%11d\t%13d\t%15d\n", processes[i].id,
           processes[i].arrival_time, processes[i].burst_time,
           processes[i].waiting_time, processes[i].turnaround_time);
    total_waiting_time += processes[i].waiting_time;
    total_turnaround_time += processes[i].turnaround_time;
  }

  printf("\nTotal Waiting Time: %d\n", total_waiting_time);
  printf("Average Waiting Time: %.2f\n",
         (float)total_waiting_time / n_processes);
  printf("Total Turnaround Time: %d\n", total_turnaround_time);
  printf("Average Turnaround Time: %.2f\n",
         (float)total_turnaround_time / n_processes);

  free(processes);
  return 0;
}
```

Output



```
ass3/q2/src via C v15.0.0-clang
❯ who; date now;
apurva           console      Sep  7 23:45
apurva           ttys000      Sep  9 15:41
Mon, 9 Sep 2024 19:01:36 +0530 (now)

ass3/q2/src via C v15.0.0-clang
❯ just r
zig cc ./src/main.c -o ./bin/main --std=c23
./bin/main
Enter the number of processes: 4
Enter the process details:
Process 1:
Arrival Time: 2
Burst Time: 4
Process 2:
Arrival Time: 0
Burst Time: 8
Process 3:
Arrival Time: 3
Burst Time: 7
Process 4:
Arrival Time: 6
Burst Time: 2
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
         2               0               8               0               8
         1               2               4               6              10
         3               3               7               9              16
         4               6               2              13              15

Total Waiting Time: 28
Average Waiting Time: 7.00
Total Turnaround Time: 49
Average Turnaround Time: 12.25
```

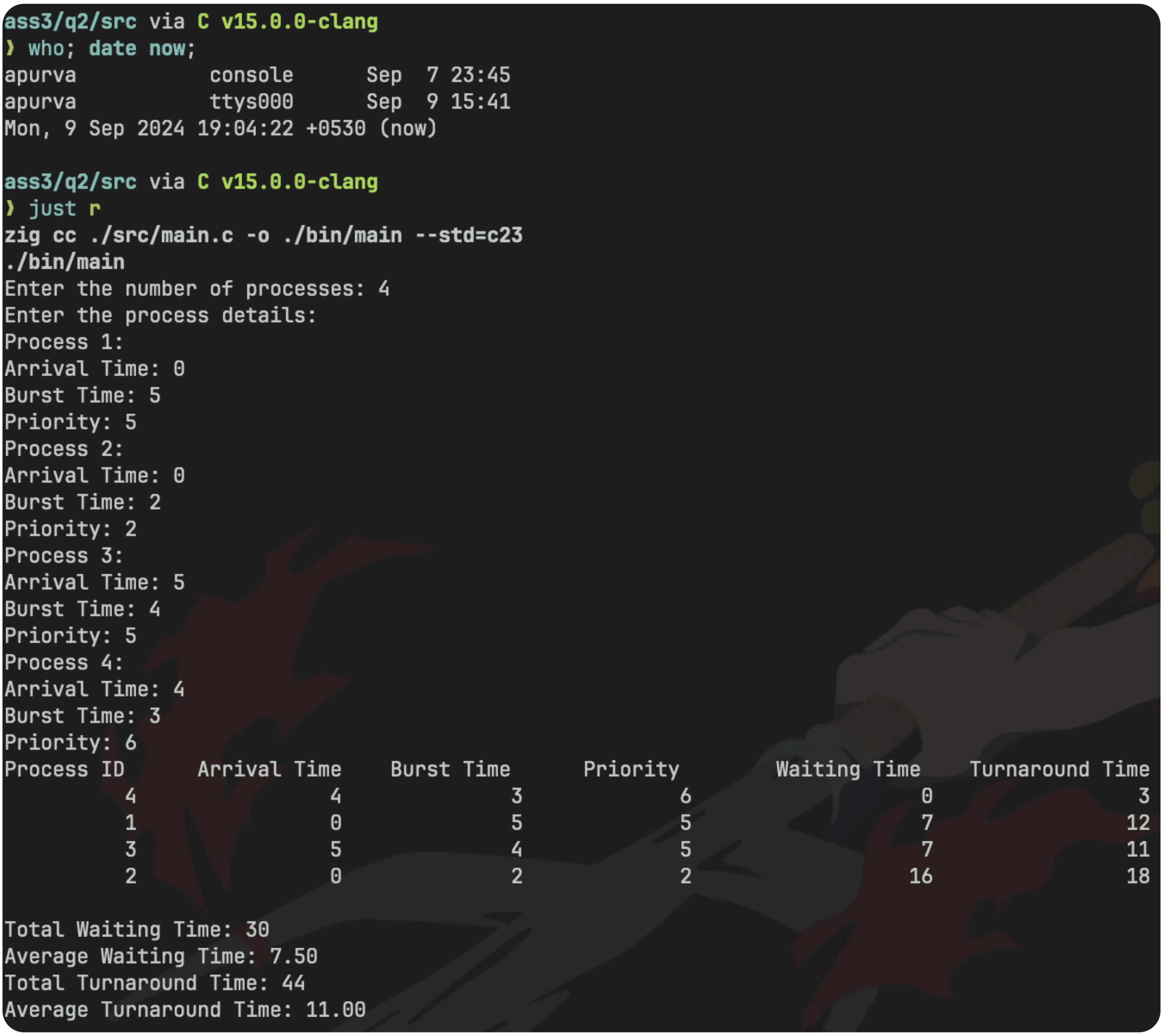
)

Problem 1.3.
Create a LINUX C program to implement Priority CPU Scheduling with varying arrival times. Processes will be scheduled based on their arrival time and priority.

CPU Scheduling based on Arrival Time

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

typedef struct {
  int id;
  int arrival_time;
  int burst_time;
  int priority;
  int waiting_time;
  int turnaround_time;
} Process;

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

void sortProcesses(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 ||
          (processes[j].priority == processes[j + 1].priority &&
           processes[j].id > processes[j + 1].id)) {
        swap(&processes[j], &processes[j + 1]);
      }
    }
  }
}

void processQueue(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;
    }

    processes[i].waiting_time = current_time - processes[i].arrival_time;
    processes[i].turnaround_time =
        processes[i].waiting_time + processes[i].burst_time;

    current_time += processes[i].burst_time;
  }
}

int main() {
  int n_processes;
  printf("Enter the number of processes: ");
  scanf("%d", &n_processes);

  Process *processes = (Process *)malloc(n_processes * sizeof(Process));

  printf("Enter the process details:\n");
  for (int i = 0; i < n_processes; i++) {
    processes[i].id = i + 1;
    printf("Process %d:\n", processes[i].id);
    printf("Arrival Time: ");
    scanf("%d", &processes[i].arrival_time);
    printf("Burst Time: ");
    scanf("%d", &processes[i].burst_time);
    printf("Priority: ");
    scanf("%d", &processes[i].priority);
  }

  sortProcesses(processes, n_processes);

  processQueue(processes, n_processes);

  printf("Process ID\tArrival Time\tBurst Time\tPriority\tWaiting "
         "Time\tTurnaround Time\n");
  int total_waiting_time = 0, total_turnaround_time = 0;
  for (int i = 0; i < n_processes; i++) {
    printf("%11d\t%12d\t%11d\t%9d\t%13d\t%15d\n", processes[i].id,
           processes[i].arrival_time, processes[i].burst_time,
           processes[i].priority, processes[i].waiting_time,
           processes[i].turnaround_time);
    total_waiting_time += processes[i].waiting_time;
    total_turnaround_time += processes[i].turnaround_time;
  }

  printf("\nTotal Waiting Time: %d\n", total_waiting_time);
  printf("Average Waiting Time: %.2f\n",
         (float)total_waiting_time / n_processes);
  printf("Total Turnaround Time: %d\n", total_turnaround_time);
  printf("Average Turnaround Time: %.2f\n",
         (float)total_turnaround_time / n_processes);

  free(processes);
  return 0;
}
```

Output



```
ass3/q2/src via C v15.0.0-clang
❯ who; date now;
apurva           console      Sep  7 23:45
apurva           ttys000      Sep  9 15:41
Mon, 9 Sep 2024 19:04:22 +0530 (now)

ass3/q2/src via C v15.0.0-clang
❯ just r
zig cc ./src/main.c -o ./bin/main --std=c23
./bin/main
Enter the number of processes: 4
Enter the process details:
Process 1:
Arrival Time: 0
Burst Time: 5
Priority: 5
Process 2:
Arrival Time: 0
Burst Time: 2
Priority: 2
Process 3:
Arrival Time: 5
Burst Time: 4
Priority: 5
Process 4:
Arrival Time: 4
Burst Time: 3
Priority: 6
Process ID      Arrival Time    Burst Time      Priority        Waiting Time    Turnaround Time
          4                4             3             6               0                 3
          1                0             5             5               7                12
          3                5             4             5               7                11
          2                0             2             2              16                18

Total Waiting Time: 30
Average Waiting Time: 7.50
Total Turnaround Time: 44
Average Turnaround Time: 11.00
```