操作系统 project-4 实验报告

∗ 姓名: 管仁阳　学号:519021911058　邮箱: guanrenyang@sjtu.edu.cn

1 实验名称

Scheduling Algorithms.

2 实验目的

1. 编写 C 程序模拟以下 CPU 调度方式:
   (a) First Come First Serve
   (b) Shortest Job First
   (c) Round Robin
   (d) Priority Based Scheduling
   (e) Priority Based Round Robin Scheduling
2. 统计每个调度方式的 Average Turnaround Time, Average Waiting Time 和 Averaged Response Time。

3 预备知识

3.1 调度算法:

3.1.1 First Come First Serve Scheduling

将提交到 CPU 的任务被组织成一个等待队列，**优先提交到 CPU 的任务优先处理**。

3.1.2 Shortest Job First Scheduling

将提交到 CPU 的任务组成成一个以执行时间为优先级的队列，**执行时间最短的任务优先处理**。在默认情况下，*SJF* 调度算法是**非抢占式**的，即当进程被提交时有进程正在被处理，那么当前进程。

3.1.3 Round Robin Scheduling

首先定义一个时间片 q。任务处理使用轮盘循环的方式：依次执行每个任务长度为 q 的一段时间，以此循环。若一个任务在某个时间片内完成，就提前结束任务的执行。

3.1.4 Priority Based Scheduling

给每一个任务分配一个优先级。将提交到 CPU 的任务被组织成一个等待队列，**优先级较高的任务优先处理**。

3.1.5 Priority Based Round Robin Scheduling

给每一个任务分配一个优先级，**先执行优先级较高的任务**。**同一优先级的任务集合使用 Round Robin 调度方式调度**。

3.2 调度性能指标

3.2.1 Turnaroud Time

任务从**被提交**到**执行完成**所需要的时间。

3.2.2 (Total) Waiting Time

任务从**上次结束执行**到**此次开始执行**所需时间。(特别地：对于首次开始执行任务来说，*Waiting Time* 为从提交到开始执行的时间)。

Total Waiting Time 为每一个任务的所有 Waiting Time 之和。

3.2.3 (Total) Response Time

任务从**被提交**到**第一次开始执行**所需时间。

Total Waiting Time 为每一个任务的 Response Time 之和。

4 实验内容

4.1 First Come First Serve scheduling

```c
int schedule_single_task()
{
    struct node* temp=(*head_first_in_head);
    struct node* prev;
    while(temp->next!=NULL)
    {
        prev=temp;
        temp=temp->next;
    }

    //stimulation of running the task which is in the tail of list
    //counting associated information(waitingtime, turnaround time, response

    total_waiting_time+=current_time;
    total_response_time+=current_time;

    run(temp->task,temp->task->burst);

    current_time+=temp->task->burst;
    total_trunaround_time+=current_time;
    number_of_tasks++;

    //after simulation, delete the node from the list
    prev->next=NULL;
    if(temp==(*head_first_in_head))
        return ALL_TASK_SCHEDULED;
    else
        return !ALL_TASK_SCHEDULED;
```

```
}
/*
there are two differences of adding/deleting elements in the list
in fcfs:
    There is no conditional branch in the while loop in line 46, so prev may
    As a reuslt, the line 65 may be an segmentation fault.
    This is a special case because the deleted element is always the last ele
in sjf, priority:
    The conditional branch ensures that pre_of_minTime and minTime are not NU
    so the deleting operation makes sense.
*/
void schedule()
{
    while(schedule_single_task()!=ALL_TASK_SCHEDULED);
    printf("Average turnaround time is %.3lf\n",total_trunaround_time/number_
    printf("Average response time is %.3lf\n",total_response_time/number_of_
    printf("Average waiting time is %.3lf\n",total_waiting_time/number_of_tas

}
```

4.2 Shortest Job First scheduling

```
int schedule_single_task()
{
    struct node* temp=(*head_first_in_head);
    struct node* prev_of_temp=temp;

    struct node* prev_of_minTime=NULL;
    struct node* minTime=temp;
    while(temp!=NULL)
    {
        if(temp->task->burst<minTime->task->burst)
        {
            minTime=temp;
            prev_of_minTime=prev_of_temp;
        }
        prev_of_temp=temp;
        temp=temp->next;
    }


    //counting associated information(waitingtime, turnaround time, response

    total_waiting_time+=current_time;
    total_response_time+=current_time;

    run(minTime->task,minTime->task->burst);

    current_time+=minTime->task->burst;
    total_trunaround_time+=current_time;
    number_of_tasks++;

    //after simulation, delete the node from the list
    /*
    there is an special case in the schedule_sjf
        when the shortest job is the head of list.
    In such case, the shortest job doesn't have a pre-node.
    */
    if(minTime==(*head_first_in_head))
        (*head_first_in_head)=(*head_first_in_head)->next;
    else
        prev_of_minTime->next=minTime->next;

    if((*head_first_in_head)==NULL)
        return ALL_TASK_SCHEDULED;
    else
        return !ALL_TASK_SCHEDULED;
}
void schedule()
{
    while(schedule_single_task()!=ALL_TASK_SCHEDULED);
    printf(”Average turnaround time is %.3lf\n”,total_trunaround_time/number_
    printf(”Average response time is %.3lf\n”,total_response_time/number_of_
    printf(”Average waiting time is %.3lf\n”,total_waiting_time/number_of_tas
}
```

4.3 Round Robin scheduling

```
int single_round_robin()
{
    struct node* temp=(*head_first_in_head);
    struct node* prev=temp;
    while(temp!=NULL)
    {
        if(temp->task->burst>QUANTUM)
        {
            if(temp->last_end_execution==0)
                total_response_time+=current_time;
            total_waiting_time+=(current_time - temp->last_end_execution);

            run(temp->task,QUANTUM);
            temp->task->burst-=QUANTUM;

            current_time+=QUANTUM;
            temp->last_end_execution=current_time;
        }
        else
        {
            if(temp->last_end_execution==0)
                total_response_time+=current_time;
            total_waiting_time+=(current_time-temp->last_end_execution);


            run(temp->task,temp->task->burst);
            // the task is over and should be deleted
            if(temp==(*head_first_in_head))//mean temp is the head
            {
                (*head_first_in_head)=temp->next;
            }
            else
            {
                prev->next=temp->next;
            }

            current_time+=(temp->task->burst);
            total_trunaround_time+=current_time;
            number_of_tasks++;
        }
        prev=temp;
        temp=temp->next;
    }
    if((*head_first_in_head)==NULL)
        return ALL_TASK_SCHEDULED;
    else
        return !ALL_TASK_SCHEDULED;
}
void schedule()
{
    if(head_first_in_head==NULL)
    {
        head_first_in_head=malloc(sizeof(struct node* ));
        (*head_first_in_head)=NULL;
    }
    //turn around the list to make the first-comming list appear at the head
    struct node* temp=(*head_first_in_tail);
    while(temp!=NULL)
    {
        insert(head_first_in_head,temp->task);
        temp=temp->next;
    }

    while(single_round_robin()!=ALL_TASK_SCHEDULED);
    printf("Average turnaround time is %.3lf\n",total_trunaround_time/number_
    printf("Average response time is %.3lf\n",total_response_time/number_of_
    printf("Average waiting time is %.3lf\n",total_waiting_time/number_of_ta

}
```

4.4 Priority Based scheduling

```c
int schedule_single_task()
{
    struct node* temp=(*head_first_in_head);
    struct node* prev_of_temp=temp;

    struct node* prev_of_maxPriority=NULL;
    struct node* maxPriority=temp;
    while(temp!=NULL)
    {
        if(temp->task->priority>maxPriority->task->priority)
        {
            maxPriority=temp;
            prev_of_maxPriority=prev_of_temp;
        }
        prev_of_temp=temp;
        temp=temp->next;
    }


    //counting associated information(waitingtime, turnaround time, response

    total_waiting_time+=current_time;
    total_response_time+=current_time;

    run(maxPriority->task,maxPriority->task->burst);

    current_time+=maxPriority->task->burst;
    total_trunaround_time+=current_time;
    number_of_tasks++;

    //after simulation, delete the node from the list
    /*
    there is an special case in the schedule_priority
        when the job with the biggest proirity is the head of list.
    In such case, the job with the biggest proirity doesn't have a pre-node.
    */
    if(maxPriority==(*head_first_in_head))
        (*head_first_in_head)=(*head_first_in_head)->next;
    else
        prev_of_maxPriority->next=maxPriority->next;

    if((*head_first_in_head)==NULL)
        return ALL_TASK_SCHEDULED;
    else
        return !ALL_TASK_SCHEDULED;
}
void schedule()
{
    while(schedule_single_task()!=ALL_TASK_SCHEDULED);
    printf("Average turnaround time is %.3lf\n",total_trunaround_time/number_
    printf("Average response time is %.3lf\n",total_response_time/number_of_
    printf("Average waiting time is %.3lf\n",total_waiting_time/number_of_tas
}
```

4.5 Priority Based Round Robin Scheduling

```
int single_priority_round_robin()
{
    struct node* temp=(*head_sort_by_priority);
    struct node* prev=temp;
    while(temp!=NULL)
    {
        if(temp->task->burst>QUANTUM)
        {
            if(temp->last_end_execution==0)
                total_response_time+=current_time;
            total_waiting_time+=(current_time - temp->last_end_execution);

            run(temp->task,QUANTUM);
            temp->task->burst-=QUANTUM;


            current_time+=QUANTUM;
            temp->last_end_execution=current_time;
        }
        else
        {
            if(temp->last_end_execution==0)
                total_response_time+=current_time;
            total_waiting_time+=(current_time-temp->last_end_execution);

            run(temp->task,temp->task->burst);
            // the task is over and should be deleted
            if(temp==(*head_sort_by_priority))//mean temp is the head
            {
                (*head_sort_by_priority)=temp->next;
            }
            else
            {
                prev->next=temp->next;
            }

            current_time+=(temp->task->burst);
            total_trunaround_time+=current_time;
            number_of_tasks++;
        }
        prev=temp;
        temp=temp->next;
    }
    if((*head_sort_by_priority)==NULL)
```

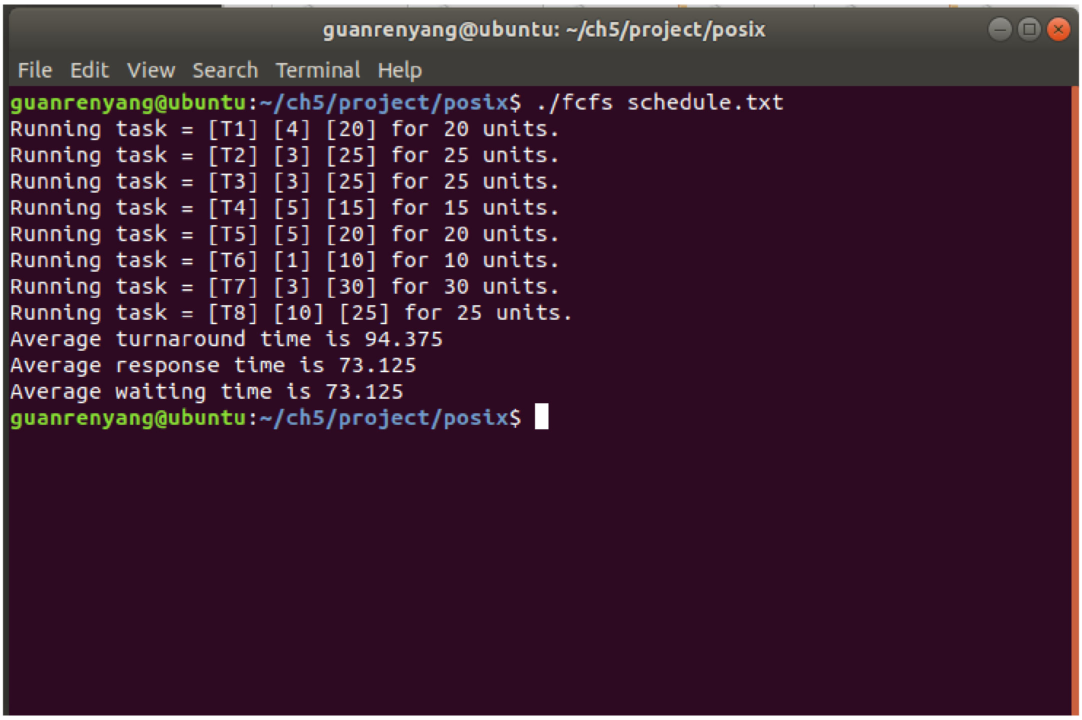
```
        return ALL_TASK_SCHEDULED;
    else
        return !ALL_TASK_SCHEDULED;
}
void schedule()
{
    if(head_sort_by_priority==NULL)
    {
        head_sort_by_priority=malloc(sizeof(struct node*));
        (*head_sort_by_priority)=NULL;
    }
    //turn around the list to make the first-comming list appear at the head
    while((*head_first_in_tail)!=NULL)
    {
        struct node* temp=(*head_first_in_tail);
        struct node* prev=temp;
        struct node* min_priority=temp;
        struct node* prev_min_priority=min_priority;
        while(temp!=NULL)
        {
            if(temp->task->priority<min_priority->task->priority)
            {
                min_priority=temp;
                prev_min_priority=prev;
            }
            prev=temp;
            temp=temp->next;
        }
        insert(head_sort_by_priority,min_priority->task);//add the task with
        //delete the node with max prority
        if(min_priority==(*head_first_in_tail))
        {
            (*head_first_in_tail)=min_priority->next;
        }
        else
        {
            prev_min_priority->next=min_priority->next;
        }
    }
    while(single_priority_round_robin()!=ALL_TASK_SCHEDULED);

    printf("Average turnaround time is %.3lf\n",total_trunaround_time/number_
    printf("Average response time is %.3lf\n",total_response_time/number_of_
    printf("Average waiting time is %.3lf\n",total_waiting_time/number_of_tas
}
```

5 实验结果

5.1 First Come First Serve Scheduling

模拟结果与性能指标见图 1.



guanrenyang@ubuntu: ~/ch5/project/posix
File Edit View Search Terminal Help

```
guanrenyang@ubuntu:~/ch5/project/posix$ ./fcfs schedule.txt
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T8] [10] [25] for 25 units.
Average turnaround time is 94.375
Average response time is 73.125
Average waiting time is 73.125
guanrenyang@ubuntu:~/ch5/project/posix$ 
```

图 1: Result

5.2 Shortest Job First Scheduling

模拟结果与性能指标见图 2.

5.3 Round Robin Scheduling

模拟结果与性能指标见图 3.

5.4 Priority Based Scheduling

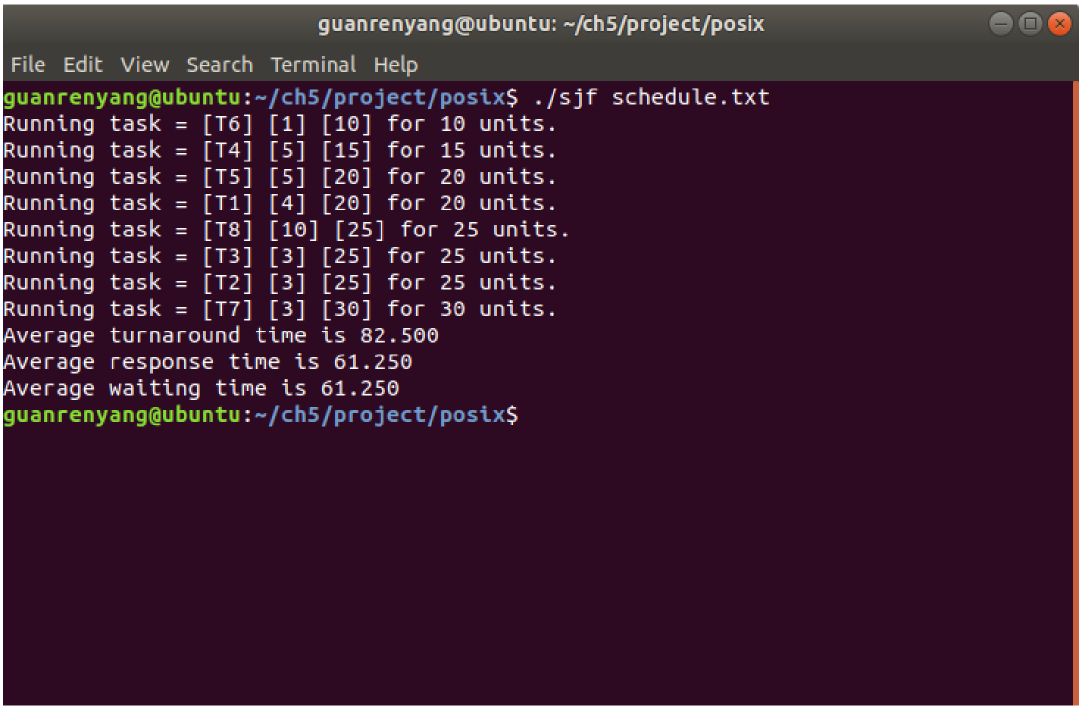
模拟结果与性能指标见图 4.

5.5 Priority Based Round Robin Scheduling

模拟结果与性能指标见图 5.

6 总结与思考

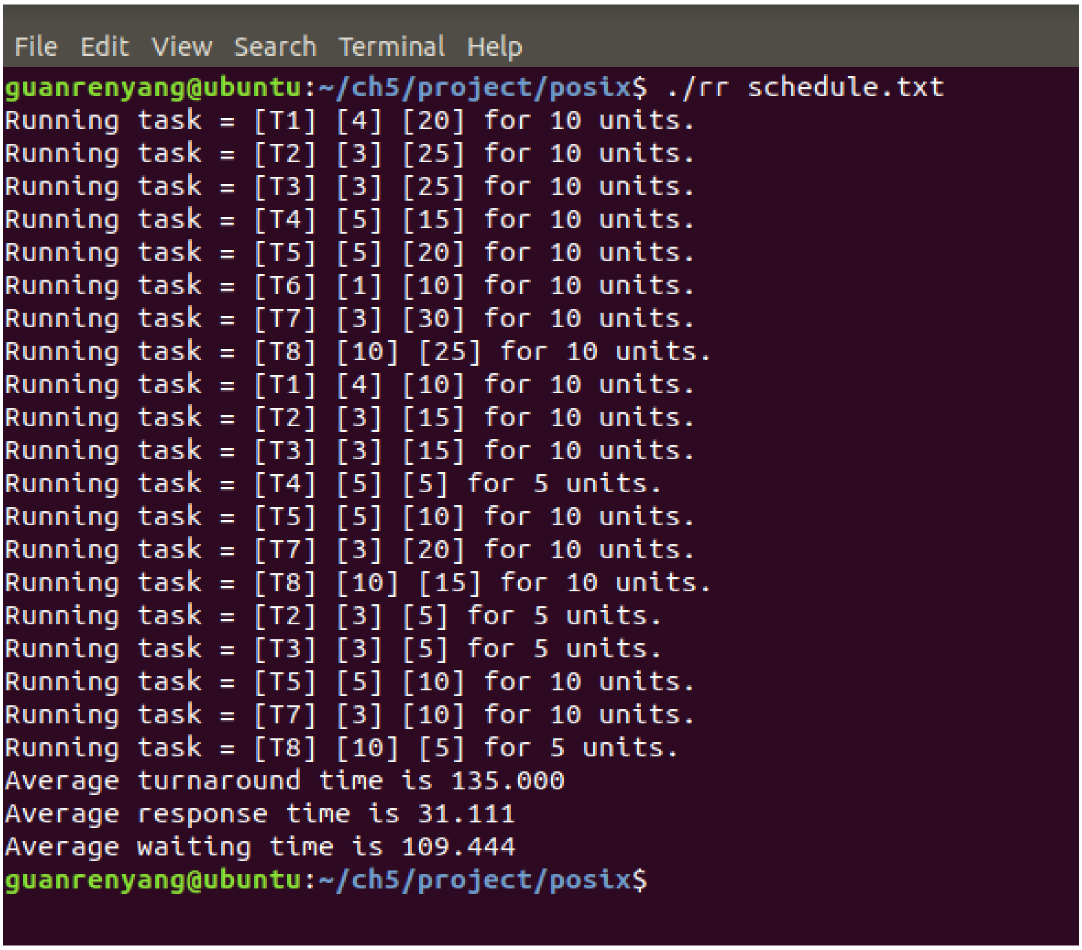
通过此次实验我更加深入地了解了 CPU 实现 FCFS、SJF、Round Robin、Priority Based Scheduling、Priority Based Round Robin Scheduling 这五种调度方式的具体方法。此外，我还使用对这五种调度方式的性能进行了模拟并且对模拟的性能指标进行了计算——通过实际操作来认识不同调度方式的优劣所在。



guanrenyang@ubuntu: ~/ch5/project/posix
File Edit View Search Terminal Help

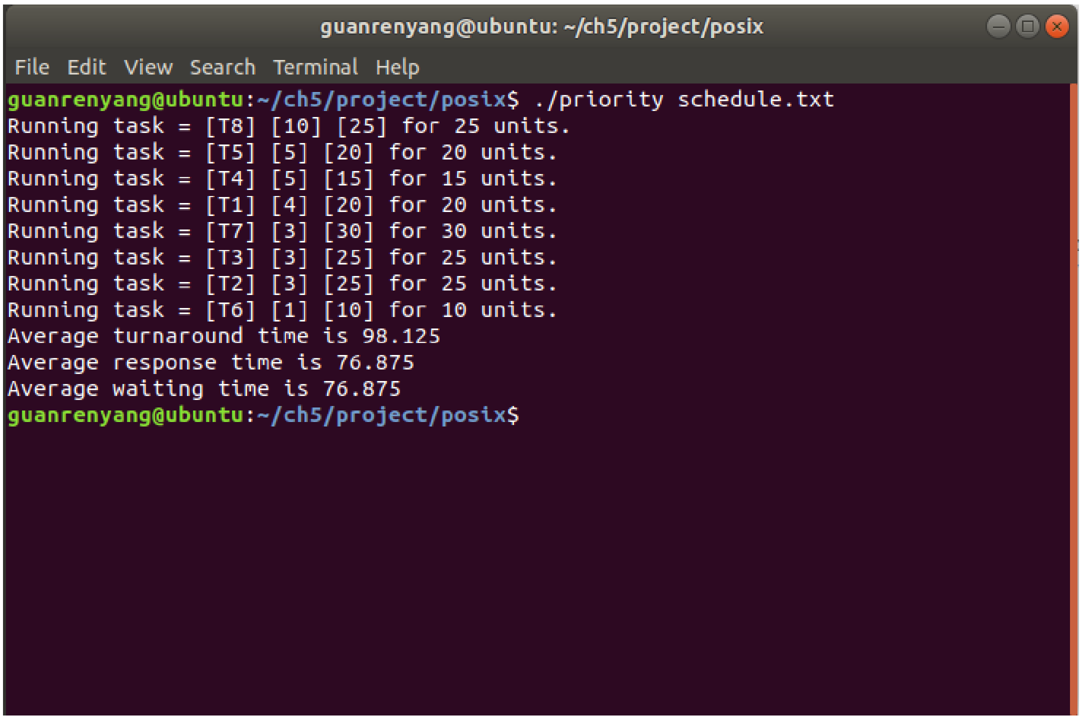
```
guanrenyang@ubuntu:~/ch5/project/posix$ ./sjf schedule.txt
Running task = [T6] [1] [10] for 10 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T8] [10] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
Average turnaround time is 82.500
Average response time is 61.250
Average waiting time is 61.250
guanrenyang@ubuntu:~/ch5/project/posix$
```

图 2: Result



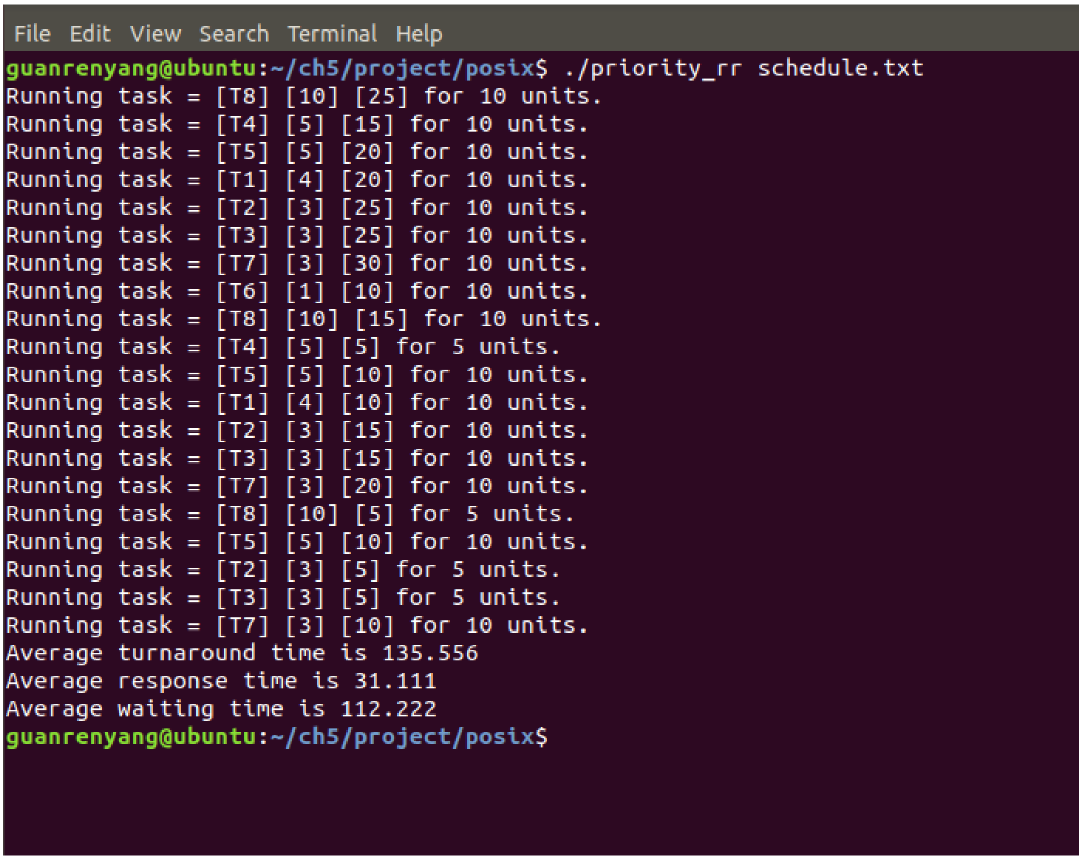
```
guanrenyang@ubuntu:~/ch5/project/posix$ ./rr schedule.txt
Running task = [T1] [4] [20] for 10 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T4] [5] [15] for 10 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T8] [10] [25] for 10 units.
Running task = [T1] [4] [10] for 10 units.
Running task = [T2] [3] [15] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T4] [5] [5] for 5 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T7] [3] [20] for 10 units.
Running task = [T8] [10] [15] for 10 units.
Running task = [T2] [3] [5] for 5 units.
Running task = [T3] [3] [5] for 5 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T7] [3] [10] for 10 units.
Running task = [T8] [10] [5] for 5 units.
Average turnaround time is 135.000
Average response time is 31.111
Average waiting time is 109.444
guanrenyang@ubuntu:~/ch5/project/posix$
```

图 3: Result


guanrenyang@ubuntu: ~/ch5/project/posix
File Edit View Search Terminal Help

```
guanrenyang@ubuntu:~/ch5/project/posix$ ./priority schedule.txt
Running task = [T8] [10] [25] for 25 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T6] [1] [10] for 10 units.
Average turnaround time is 98.125
Average response time is 76.875
Average waiting time is 76.875
guanrenyang@ubuntu:~/ch5/project/posix$
```

图 4: Result



```
guanrenyang@ubuntu:~/ch5/project/posix$ ./priority_rr schedule.txt
Running task = [T8] [10] [25] for 10 units.
Running task = [T4] [5] [15] for 10 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T1] [4] [20] for 10 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T8] [10] [15] for 10 units.
Running task = [T4] [5] [5] for 5 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T1] [4] [10] for 10 units.
Running task = [T2] [3] [15] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T7] [3] [20] for 10 units.
Running task = [T8] [10] [5] for 5 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T2] [3] [5] for 5 units.
Running task = [T3] [3] [5] for 5 units.
Running task = [T7] [3] [10] for 10 units.
Average turnaround time is 135.556
Average response time is 31.111
Average waiting time is 112.222
guanrenyang@ubuntu:~/ch5/project/posix$
```

图 5: Result