# 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

```
int schedule_single_task()
2
3
       struct node* temp=(*head_first_in_head);
       struct node* prev;
4
       while (temp->next!=NULL)
5
6
       {
           prev=temp;
7
           temp=temp->next;
8
       }
9
10
       //stimulation of running the task which is in the tail of list
11
       //counting associated information (waiting time, turn around time, response
12
13
       total_waiting_time+=current_time;
14
       total_response_time+=current_time;
15
16
       run (temp->task, temp->task->burst);
17
18
       current time+=temp->task->burst;
19
       total_trunaround_time+=current_time;
20
       number_of_tasks++;
21
22
       //after simulation, delete the node from the list
23
       prev->next=NULL;
24
       if (temp==(*head_first_in_head))
25
           return ALL_TASK_SCHEDULED;
26
27
       else
           return !ALL_TASK_SCHEDULED;
28
```

```
29
  }
30
  there are two differences of adding/deleting elements in the list
31
   in fcfs:
32
       There is no conditional branch in the while loop in line 46, so prev may
33
       As a reuslt, the line 65 may be an segmentation fault.
34
35
       This is a special case because the deleted element is always the last ele
   in \ sjf, priority:
36
       The conditional branch ensures that pre_of_minTime and minTime are not No
37
       so the deleting operation makes sense.
38
39
   */
   void schedule()
40
   {
41
       while (schedule single task()!=ALL TASK SCHEDULED);
42
       printf ("Average turnaround time is \%.31f \ n", total_trunaround_time/number.
43
       printf("Average response time is %.31f\n", total_response_time/number_of_
44
       printf("Average waiting time is %.31f\n", total_waiting_time/number_of_tal
45
46
47
```

## 4.2 Shortest Job First scheduling

```
int schedule_single_task()
2
       struct node* temp=(*head_first_in_head);
3
       struct node* prev_of_temp=temp;
4
5
       struct node* prev of minTime=NULL;
6
7
       struct node* minTime=temp;
       while (temp!=NULL)
8
9
       {
           if (temp->task->burst<minTime->task->burst)
10
11
12
                minTime=temp;
                prev_of_minTime=prev_of_temp;
13
14
           prev of temp=temp;
15
           temp=temp->next;
16
       }
17
18
19
       //counting associated information (waiting time, turn around time, response
20
21
22
       total_waiting_time+=current_time;
       total_response_time+=current_time;
23
24
       run (minTime->task, minTime->task->burst);
25
26
       current_time+=minTime->task->burst;
27
       total_trunaround_time+=current_time;
28
```

```
number_of_tasks++;
29
30
       //after simulation, delete the node from the list
31
32
       there is an special case in the schedule sif
33
           when the shortest job is the head of list.
34
35
       In such case, the shortest job doesn't have a pre-node.
36
       */
       if (minTime==(*head_first_in_head))
37
           (*head_first_in_head)=(*head_first_in_head)->next;
38
       else
39
           prev_of_minTime->next=minTime->next;
40
41
       if ((*head first in head)==NULL)
42
           return ALL_TASK_SCHEDULED;
43
44
       else
           return !ALL TASK SCHEDULED;
45
46
   void schedule()
47
48
       while (schedule_single_task()!=ALL_TASK_SCHEDULED);
49
       printf ("Average turnaround time is \%.31f \ n", total_trunaround_time/number.
50
       printf("Average response time is %.31f\n", total_response_time/number_of_
51
       printf("Average waiting time is %.31f\n", total_waiting_time/number_of_tal
52
53
```

## 4.3 Round Robin scheduling

```
int single_round_robin()
1
2
       struct node* temp=(*head_first_in_head);
3
       struct node* prev=temp;
4
       while (temp!=NULL)
5
6
            if (temp->task->burst>QUANTUM)
7
            {
8
                 if (temp->last end execution==0)
9
                     total_response_time+=current_time;
10
                 total_waiting_time+=(current_time - temp->last_end_execution);
11
12
                 run (temp->task ,QUANTUM);
13
                 temp \rightarrow task \rightarrow burst = QUANTUM;
14
15
                 current time+=QUANTUM;
16
                temp->last_end_execution=current_time;
17
            }
18
            else
19
20
                 if (temp=>last_end_execution==0)
21
                     total_response_time+=current_time;
22
```

```
total_waiting_time+=(current_time-temp->last_end_execution);
23
24
25
                run (temp->task, temp->task->burst);
26
                // the task is over and should be deleted
27
                if (temp==(*head_first_in_head)) //mean temp is the head
28
29
                     (*head_first_in_head)=temp->next;
30
31
                else
32
33
                    prev->next=temp->next;
34
35
36
                current_time+=(temp->task->burst);
37
                total_trunaround_time+=current_time;
38
                number of tasks++;
39
           }
40
           prev=temp;
41
           temp=temp->next;
42
43
       if ((*head_first_in_head)==NULL)
44
            return ALL_TASK_SCHEDULED;
45
       else
46
           return !ALL_TASK_SCHEDULED;
47
48
   void schedule()
49
50
       if (head_first_in_head==NULL)
51
       {
52
           head_first_in_head=malloc(sizeof(struct node*));
53
            (*head_first_in_head)=NULL;
54
55
       //turn around the list to make the first-comming list appear at the head
56
       struct node* temp=(*head_first_in_tail);
57
       while (temp!=NULL)
58
       {
59
            insert (head_first_in_head , temp->task );
60
           temp=temp->next;
61
       }
62
63
       while (single_round_robin()!=ALL_TASK_SCHEDULED);
64
       printf ("Average turnaround time is \%.31f \ n", total_trunaround_time/number.
65
       printf("Average response time is %.31f\n", total_response_time/number_of_
66
       printf("Average waiting time is %.31f\n", total_waiting_time/number_of_ta
67
68
69
```

### 4.4 Priority Based scheduling

```
int schedule_single_task()
2
   {
       struct node* temp=(*head_first_in_head);
3
       struct node* prev_of_temp=temp;
4
5
       struct node* prev of maxPriority=NULL;
6
       struct node* maxPriority=temp;
7
       while (temp!=NULL)
8
9
           if (temp->task->priority>maxPriority->task->priority)
10
           {
11
                maxPriority=temp;
12
                prev_of_maxPriority=prev_of_temp;
13
14
           prev_of_temp=temp;
15
           temp=temp->next;
16
       }
17
18
19
20
       //counting associated information(waitingtime, turnaround time, response
21
       total_waiting_time+=current_time;
22
       total_response_time+=current_time;
23
24
       run (maxPriority->task, maxPriority->task->burst);
25
26
27
       current_time+=maxPriority->task->burst;
       total trunaround time+=current time;
28
29
       number_of_tasks++;
30
       //after simulation, delete the node from the list
31
32
       there is an special case in the schedule_priority
33
34
           when the job with the biggest proirity is the head of list.
35
       In such case, the job with the biggest proirity doesn't have a pre-node.
36
       */
       if ( maxPriority == (*head_first_in_head ) )
37
           (*head_first_in_head)=(*head_first_in_head)->next;
38
       else
39
           prev_of_maxPriority->next=maxPriority->next;
40
41
       if ((*head_first_in_head)==NULL)
42
           return ALL_TASK_SCHEDULED;
43
44
       else
           return !ALL_TASK_SCHEDULED;
45
46
   void schedule()
47
48
       while (schedule_single_task()!=ALL_TASK_SCHEDULED);
49
       printf("Average turnaround time is %.31f\n", total_trunaround_time/number
50
```

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

## 4.5 Priority Based Round Robin Scheduling

```
int single_priority_round_robin()
2
       struct node* temp=(*head_sort_by_priority);
3
       struct node* prev=temp;
4
       while (temp!=NULL)
5
6
            if (temp->task->burst>QUANTUM)
7
            {
8
                 if (temp->last end execution==0)
9
                     total_response_time+=current_time;
10
                 total_waiting_time+=(current_time - temp->last_end_execution);
11
12
                run (temp->task ,QUANTUM);
13
                temp \rightarrow task \rightarrow burst = QUANTUM;
14
15
16
                 current_time+=QUANTUM;
17
                temp->last_end_execution=current_time;
18
            }
19
            else
20
21
            {
                 if (temp->last end execution==0)
22
                     total_response_time+=current_time;
23
                 total_waiting_time+=(current_time-temp->last_end_execution);
24
25
                run (temp->task, temp->task->burst);
26
                 // the task is over and should be deleted
27
                 if (temp==(*head_sort_by_priority)) //mean temp is the head
28
29
                     (*head_sort_by_priority)=temp->next;
30
31
                 else
32
                 {
33
                     prev->next=temp->next;
34
35
36
                 current time+=(temp->task->burst);
37
                 total_trunaround_time+=current_time;
38
39
                number_of_tasks++;
            }
40
            prev=temp;
41
42
            temp=temp->next;
43
       if ((*head_sort_by_priority)==NULL)
44
```

```
return ALL_TASK_SCHEDULED;
45
       else
46
           return !ALL_TASK_SCHEDULED;
47
48
   void schedule()
49
50
       if ( head_sort_by_priority==NULL)
51
       {
52
           head_sort_by_priority=malloc(sizeof(struct node*));
53
           (*head_sort_by_priority)=NULL;
54
55
       //turn around the list to make the first-comming list appear at the head
56
       while ((*head_first_in_tail)!=NULL)
57
58
           struct node* temp=(*head_first_in_tail);
59
           struct node* prev=temp;
60
           struct node* min priority=temp;
61
           struct node* prev_min_priority=min_priority;
62
            while (temp!=NULL)
63
64
                if (temp->task->priority <min_priority->task->priority)
65
66
                    min_priority=temp;
67
                    prev_min_priority=prev;
68
69
                prev=temp;
70
71
                temp=temp->next;
72
           insert (head_sort_by_priority, min_priority->task); //add the task with
73
           //delete the node with max prority
74
           if ( min_priority == (*head_first_in_tail ))
75
76
                (*head_first_in_tail)=min_priority->next;
77
78
           }
79
           else
80
                prev_min_priority->next=min_priority->next;
81
82
83
       while (single priority round robin ()!=ALL TASK SCHEDULED);
84
85
       printf("Average turnaround time is %.31f\n", total_trunaround_time/number
86
       printf("Average response time is %.31f\n", total_response_time/number_of_
87
       printf("Average waiting time is %.31f\n", total_waiting_time/number_of_tal
88
89
```

# 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 = [T1]
Running task = [T2]
                                   [25]
[15]
                             [3]
[5]
Running task =
                      [T3]
                                         for 25 units.
                                                15 units.
Running task =
                      [T4]
                                          for
Running task =
                      [T5]
                                   [20]
                                          for 20 units.
Running task =
                      [T6]
                                   [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 = [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

```
File Edit View Search Terminal Help
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

```
File Edit View Search Terminal Help
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]
Running task = [T2] [3]
                                    for 10 units.
                               [20]
                               [25]
                                     for 10 units.
                   [T3]
[T7]
                         [3]
[3]
                               [25] for 10 units.
[30] for 10 units.
Running task =
Running task =
                   [T6]
                         [1]
                               [10] for 10 units.
Running task =
Running task
                   [T8]
                         [10] [15] for 10 units.
Running task =
                   [T4]
                         [5] [5] for 5 units.
                         [5]
[4]
[3]
Running task =
                    [T5]
                               [10] for 10 units.
                   [T1]
                               [10] for 10 units.
[15] for 10 units.
Running task =
                   [T2]
[T3]
                               [15] for 10 units.
[15] for 10 units.
Running task =
Running task =
                         [3]
Running task =
                   [T7]
                               [20] for 10 units.
                   [T8] [10] [5] for 5 units.
Running task =
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