

Project4 实验报告

余北辰 519030910245

1 实验概述

1.1 实验名称

Scheduling Algorithms

1.2 实验内容

1. 模拟FCFS, SJF, Priority, RR, Priority_rr五种不同的调度方案
2. 计算每种方案的平均周转时间, 平均等待时间和平均响应时间。

2 实验环境

- Ubuntu 18.04.5 LTS
- Linux version 5.4.0-72-generic
- VirtualBox 6.1.18

3 实验过程与结果展示

在提供的源代码中, 其他的部分已经为我们写好, 我们所需要的做的就是补充好五个 `schedule.h` 文件, 完成其中的 `add()` 函数和 `schedule()` 函数, 分别实现添加进程到就绪队列以及实现对应的进程调度的功能。

为了进行平均周转时间, 平均等待时间和平均响应时间的计算, 在 `task.h` 中添加变量:

```
1 extern int task_tid;
2 extern int total_turnaround_time;
3 extern int total_waiting_time;
4 extern int total_response_time;
```

在driver.c中添加输出：

```
1     printf("Average turnaround time = %lf units.\n", 1.0 * total_turnaround_time  
    / task_tid);  
2     printf("Average waiting time = %lf units.\n", 1.0 * total_waiting_time /  
    task_tid);  
3     printf("Average response time = %lf units.\n", 1.0 * total_response_time /  
    task_tid);
```

3.1 FCFS

FCFS的实现最为简单。

对于add函数，只要简单地将新进程添加到就绪队列的末尾即可。这里根据课本的提示，使用 `__sync_fetch_and_add()` 函数，原子地实现tid的自加1。

```
1     void add(char *name, int priority, int burst)  
2     {  
3         Task *new_task = malloc(sizeof(Task));  
4         new_task->burst = burst;  
5         new_task->name = name;  
6         new_task->priority = priority;  
7         new_task->response_time = new_task->turnaround_time = new_task-  
    >waiting_time = 0;  
8         new_task->tid = __sync_fetch_and_add(&task_tid, 1);  
9         insert(&task_list, new_task);  
10    }
```

调度算法的实现也较为简单，只要将就绪队列链表从头结点开始遍历即可：

```
1     void schedule()  
2     {  
3         int time = 0;  
4         Node *p = task_list;  
5         while (p != NULL)  
6         {  
7             while (p->next != NULL)  
8             {  
9                 p = p->next;  
10            }  
11            int slice = p->task->burst;  
12            run(p->task, slice);  
13            p->task->waiting_time = p->task->response_time = time;  
14            time += slice;
```

```

15     p->task->turnaround_time = time;
16     __sync_fetch_and_add(&total_response_time, p->task->response_time);
17     __sync_fetch_and_add(&total_turnaround_time, p->task-
    >turnaround_time);
18     __sync_fetch_and_add(&total_waiting_time, p->task->waiting_time);
19     delete (&task_list, p->task);
20     free(p->task);
21     p = task_list;
22 }
23 }

```

测试结果:

```

polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project4
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ ./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 = 94.375000 units.
Average waiting time = 73.125000 units.
Average response time = 73.125000 units.
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ |

```

3.2 SJF

SJF的 `add()` 函数和FCFS相同，都只需要将新进程添加到就绪队列的末尾即可。

`schedule()` 如下，每次都寻找就绪队列中最短运行时间的进程：

```

1  void schedule()
2  {
3      int time = 0;
4      Node *p = task_list;
5      while (p != NULL)
6      {
7          int shortest_burst = p->task->burst;
8          Node *q = p;
9          while (p->next != NULL)
10         {
11             p = p->next;
12             if (p->task->burst <= shortest_burst)
13             {
14                 shortest_burst = p->task->burst;

```

```

15         q = p;
16     }
17 }
18 int slice = q->task->burst;
19 run(q->task, slice);
20 q->task->waiting_time = q->task->response_time = time;
21 time += slice;
22 q->task->turnaround_time = time;
23 __sync_fetch_and_add(&total_response_time, q->task->response_time);
24 __sync_fetch_and_add(&total_turnaround_time, q->task-
>turnaround_time);
25 __sync_fetch_and_add(&total_waiting_time, q->task->waiting_time);
26 delete (&task_list, q->task);
27 free(q->task);
28 p = task_list;
29 }
30 }

```

测试结果：

```

polaris@polaris-VirtualBox: ~/course/Operating-Systems/Proj
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ ./sjf schedule.txt
Running task = [T6] [1] [10] for 10 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T8] [10] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
Average turnaround time = 82.500000 units.
Average waiting time = 61.250000 units.
Average response time = 61.250000 units.
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$

```

3.3 RR

RR的 `add()` 函数和FCFS相同，都只需要将新进程添加到就绪队列的末尾即可。

`schedule()` 函数如下，类似于FCFS，而每次进程只允许最多slice的时间后就被中止并重新插入就绪队列的末尾：

```

1 void schedule()
2 {
3     int time = 0;
4     Node *p = task_list;
5     while (p != NULL)
6     {
7         while (p->next != NULL)
8         {
9             p = p->next;

```

```

10     }
11     int slice = p->task->burst;
12     run(p->task, slice);
13     p->task->waiting_time = p->task->response_time = time;
14     time += slice;
15     p->task->turnaround_time = time;
16     __sync_fetch_and_add(&total_response_time, p->task->response_time);
17     __sync_fetch_and_add(&total_turnaround_time, p->task-
18 >turnaround_time);
19     __sync_fetch_and_add(&total_waiting_time, p->task->waiting_time);
20     delete (&task_list, p->task);
21     free(p->task);
22     p = task_list;
23 }

```

测试结果如下：

```

polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project4
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ ./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 = [T7] [3] [10] for 10 units.
Running task = [T8] [10] [5] for 5 units.
Average turnaround time = 128.750000 units.
Average waiting time = 107.500000 units.
Average response time = 35.000000 units.
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$

```

3.4 priority

priority调度的 `add()` 函数和FCFS相同，都只需要将新进程添加到就绪队列的末尾即可。

`schedule()` 函数与SJF类似，每次都寻找就绪队列中最高优先级的进程：

```

1 void schedule()
2 {
3     int time = 0;
4     Node *p = task_list;
5     while (p != NULL)
6     {
7         int biggest_priority = p->task->priority;

```

```

8      Node *q = p;
9      while (p->next != NULL)
10     {
11         p = p->next;
12         if (p->task->priority >= biggest_priority)
13         {
14             biggest_priority = p->task->priority;
15             q = p;
16         }
17     }
18     int slice = q->task->burst;
19     run(q->task, slice);
20     q->task->waiting_time = q->task->response_time = time;
21     time += slice;
22     q->task->turnaround_time = time;
23     __sync_fetch_and_add(&total_response_time, q->task->response_time);
24     __sync_fetch_and_add(&total_turnaround_time, q->task-
>turnaround_time);
25     __sync_fetch_and_add(&total_waiting_time, q->task->waiting_time);
26     delete (&task_list, q->task);
27     free(q->task);
28     p = task_list;
29 }
30 }

```

测试结果如下：

```

polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project4
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ ./priority schedule.txt
Running task = [T8] [10] [25] for 25 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 = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T6] [1] [10] for 10 units.
Average turnaround time = 96.250000 units.
Average waiting time = 75.000000 units.
Average response time = 75.000000 units.
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ |

```

3.5 priority-RR

priority-RR调度的 `add()` 函数和FCFS相同，都只需要将新进程添加到就绪队列的末尾即可。

而 `schedule()` 函数则复杂一些，属于priority和RR的结合，每次都寻找就绪队列中最高优先级的进程，且只允许运行最多slice的时间后就被中止并重新插入就绪队列的末尾：

```

1 void schedule()
2 {
3     int slice = QUANTUM;
4     int time = 0;
5     Node *p = task_list;
6     while (p != NULL)
7     {
8         int biggest_priority = p->task->priority;
9         Node *q = p;
10        while (p->next != NULL)
11        {
12            p = p->next;
13            if (p->task->priority >= biggest_priority)
14            {
15                biggest_priority = p->task->priority;
16                q = p;
17            }
18        }
19        if (q->task->burst > slice)
20        {
21            run(q->task, slice);
22            if (q->task->response_time == -1)
23                q->task->response_time = time;
24            q->task->burst -= slice;
25            time += slice;
26            delete (&task_list, q->task);
27            insert(&task_list, q->task);
28        }
29        else if (q->task->burst <= slice)
30        {
31            run(q->task, q->task->burst);
32            if (q->task->response_time == -1)
33                q->task->response_time = time;
34            time += q->task->burst;
35            q->task->turnaround_time = time;
36            q->task->waiting_time = time - q->task->origin_burst;
37            __sync_fetch_and_add(&total_response_time, q->task-
38            >response_time);
39            __sync_fetch_and_add(&total_turnaround_time, q->task-
40            >turnaround_time);
41            __sync_fetch_and_add(&total_waiting_time, q->task->waiting_time);
42            delete (&task_list, q->task);
43            free(q->task);

```

```
42     }
43     p = task_list;
44 }
45 }
```

测试结果如下：

```
polaris@polaris-VirtualBox: ~/course/Operating-Systems/Project/Project4
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ ./priority_rr schedule.txt
Running task = [T8] [10] [25] for 10 units.
Running task = [T8] [10] [15] for 10 units.
Running task = [T8] [10] [5] for 5 units.
Running task = [T4] [5] [15] for 10 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T4] [5] [5] for 5 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T1] [4] [20] for 10 units.
Running task = [T1] [4] [10] 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 = [T2] [3] [15] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T7] [3] [20] 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.
Running task = [T6] [1] [10] for 10 units.
Average turnaround time = 105.000000 units.
Average waiting time = 83.750000 units.
Average response time = 68.750000 units.
polaris@polaris-VirtualBox:~/course/Operating-Systems/Project/Project4$ |
```

4 实验总结

1. 由于本实验的大体框架已经搭好，所要做的只是修改代码而已，因此整体难度不太大
2. RR和priority-RR调度算法在打印时，一开始没有减去已经运行过的时间，后来已改正

5 实验参考资料

- 实验参考书籍：Operating System Concept, 10th edition
- 实验源代码网址：<https://github.com/greggagne/osc10e>