计算机系统体系结构 Project4

517030910326 王孝诚

2019.11.17

实验环境

Windows 10 下使用 VMWare Workstation 15 Player 创建和运行虚拟机，虚拟机环境是 Linux 发行版 Ubuntu16.04.6 LTS。

1 调度算法

这次我们将完成 5 种 CPU 调度算法，分别是：1）First-Come, First-Served；2）Shortest-Job-First；3）Priority；4）Round-Robin；5）Priority+Round-Robin。在给出的代码中，已经实现了任务结构体的定义，驱动程序的入口和出口，list 数据结构及其相关操作，自己只需要完成具体算法的实现。

1.1 设计思路

1. **通用的 add() 函数**各个调度的 add 操作是一样的，新申请一块空间存储一个新的任务 task，然后将各个参数值赋给它，最后 insert 到当前的链表里即可。要注意的是这里实现的链表是一个先进后出的栈，后面有时根据算法需要对其做逆序的操作。

2. **FCFS** 这个比较简单，直接将链表里的 task 按添加顺序执行即可。由于存储的 head 指针指向最后一个添加的任务，所以要先将链表逆序，再依次取出任务执行。执行过的任务从链表中 delete。

3. **SJF** 由于每次执行最短的任务，所以在执行之前需要选出最小 burst 的一个，然后执行之并删除。这个也比较简单。

4. **Priority** 和 SJF 十分类似，代码中只要将 burst 换成 priority，把求最小换成求最大即可。

5. **Round-Robin** 也需要先将链表逆序，然后依次取出任务执行；每次运行 10 单位时间后，将其 burst 减去 10，若 burst 小于 10 则代表任务结束，将其删除；然后取出链表中下一个任务，如此循环直至不再有未完成任务。

6. **Priority-rr** 这里有几种方式来实现，一种是将为每个 priority 建立单独的链表，从而由高优先级的链表开始，若链表元素大于一个则运行 rr。另一种是先将原链表按优先级排序，然后顺序执行，若检测到相同优先级的任务则开始执行 rr。我选择了后面一种方式。检测到当前元素的优先级与下一个的优先级相等，则将当前元素设置为头部，从当前元素开始向后执行 rr，当发现后一个优先级不再相等时，将其设为尾部，然后在头部与尾部间循环 rr；完成后将指针指向尾部，则可以继续按 priority 顺序执行。

1.2 核心代码解释

1. **通用的 add() 函数**

```
1  void add(char *name, int priority, int burst)
2  {
3      struct task* a_task = malloc(sizeof(struct task));
```

```
    a_task->name = name;
    a_task->priority = priority;
    a_task->burst = burst;
    insert(&head, a_task);
    task_num++;
}
```

2. **FCFS**

```
void schedule()
{
    struct node *tmp;
    struct node *prev = NULL;
    while(head->next!=NULL){// 逆序
        tmp = head->next;
        head->next = prev;
        prev = head;
        head = tmp;
    }
    head->next = prev;
    while (task_num>0){// 顺序执行
        run(head->task, head->task->burst);
        delete(&head, head->task);
        task_num--;
    }
}
```

3. **SJF**

```
void schedule()
{
    /* 申明和定义一些变量 */

    int crt_max = 10000;
    while (task_num>0){
        tmp = head;
        while (tmp!=NULL){// 选出最小burst的任务
            if(tmp->task->burst < crt_max){
                crt_max = tmp->task->burst;
                tmp_task = tmp->task;
            }
            tmp = tmp->next;
        }

        /* 执行任务 */
    }
}
```

4. **Priority**

```
/* 与sjf类似 */
```

```
while (tmp!=NULL){ // 选出priority最大的任务
    if(tmp->task->priority > crt_min){
        crt_min = tmp->task->priority;
        tmp_task = tmp->task;
    }
    tmp = tmp->next;
}
/* 与sjf类似 */
```

5. **Round-Robin**

```
void schedule()
{
    /* 声明变量和逆序操作 */

    while (task_num>0){
        if(tmp->task->burst <= 10){// 若任务会结束
            run(tmp->task,tmp->task->burst);
            delete(&head, tmp->task);
            task_num--;
        }else{// 若任务不会结束
            run(tmp->task,10);
            tmp->task->burst -= 10;
        }
        tmp = tmp->next;
        if(tmp == NULL){// 到了队尾，回到队头
            tmp = head;
        }
    }
}
```

6. **Priority-rr**

```
void schedule()
{
    struct node *tmp;
    int crt_max = 10000;
    struct task *tmp_task;
    struct node *rr_tail;
    int tmp_num = task_num;
    int pr;

    while (task_num>0){// 按priority从大到小排序
        tmp = head;
        while (tmp!=NULL){
            if(tmp->task->priority < crt_max){
                crt_max = tmp->task->priority;
                tmp_task = tmp->task;
            }
            tmp = tmp->next;
        }
```

```
            insert(&new_head, tmp_task);
            delete(&head, tmp_task);
            task_num--;
            crt_max = 10000;
        }

        while(tmp_num > 0){
            if(new_head->next != NULL && new_head->task->priority ==
                new_head->next->task->priority){// 进入rr循环
                tmp = new_head;
                pr = new_head->task->priority;
                while(new_head != rr_tail){
                    if(tmp->task->burst<=10){
                        run(tmp->task, tmp->task->burst);
                        delete(&new_head, tmp->task);
                        tmp_num--;
                    }else{
                        run(tmp->task, 10);
                        tmp->task->burst -= 10;
                    }
                    if(tmp->next == NULL || tmp->next->task->priority != pr
                        ){// 设置队尾
                        rr_tail = tmp->next;
                    }
                    tmp = tmp->next;
                    if(tmp == rr_tail){// 回到队头
                        tmp = new_head;
                    }
                }
            }else{// 按照priority执行
                run(new_head->task, new_head->task->burst);
                delete(&new_head, new_head->task);
                tmp_num--;
            }
        }
    }
```

2 实验结果

```
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.
```

(a) FCFS

```
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.
```

(b) SJF

```
Running task = [T8] [10] [25] for 25 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 20 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.
```

(c) Priority and RR

```
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.
```

(d) RR

```
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.
```

(e) Priority