操作系统(D)

项目 4

李子龙 518070910095

2021 年 3 月 21 日

调度算法

一 C 语言实现版本

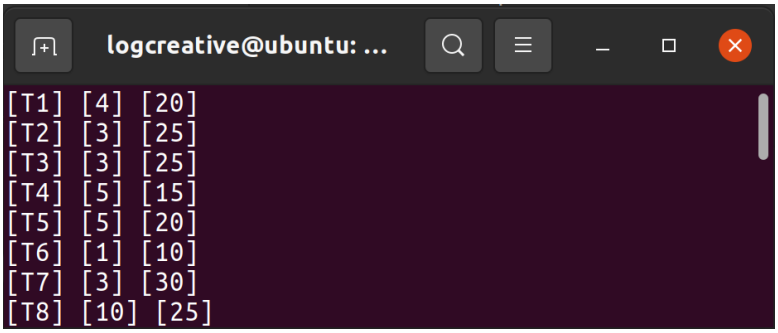
1. FCFS

Listing 1: src/posix/schedule_fcfs.c

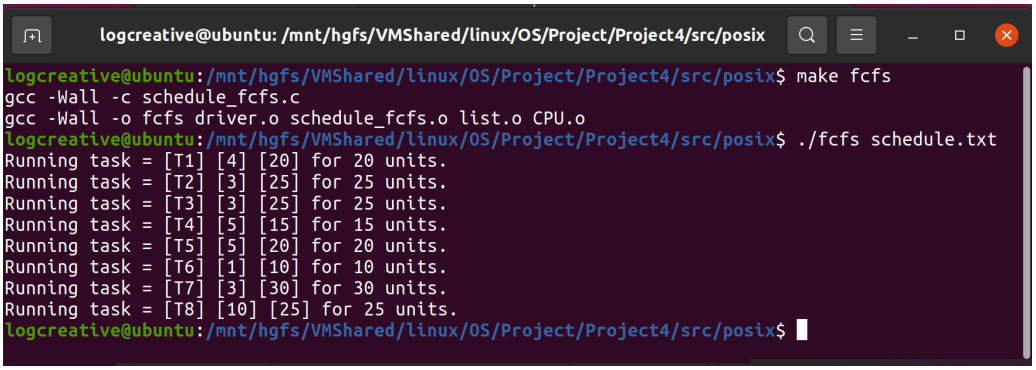
```c
#include "schedulers.h"
#include "list.h"
#include "cpu.h"
#include <stdio.h>
#include <stdlib.h>

struct node* taskList = NULL;

void add(char *name, int priority, int burst){
    Task* newTask = (Task*) malloc(sizeof(Task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;

    if(!taskList){
        taskList = malloc(sizeof(struct node));
        taskList->task = newTask;
        taskList->next = NULL;
    } else {
        struct node* temp = taskList;
        while(temp->next) temp = temp->next;
        insert(&temp->next,newTask);
    }

}

void schedule(){

    // traverse(taskList);

    struct node *temp = taskList;
    while(temp){
        Task* thisTask = temp->task;
```

```
        run(thisTask,thisTask->burst);
        temp = temp->next;
        delete(&taskList, thisTask);
    }
}
```

将输入的任务按照先后次序形成链表，注意第一个任务插入的时候特别进行了处理，之后按照给定的链表方法使用地址输入需要被替代位置的元素，之后顺移。遍历结果如下图所示：



之后按照 FCFS 的规则遍历该链表运行即可，运行完成后就会把链表上对应的元素删除。



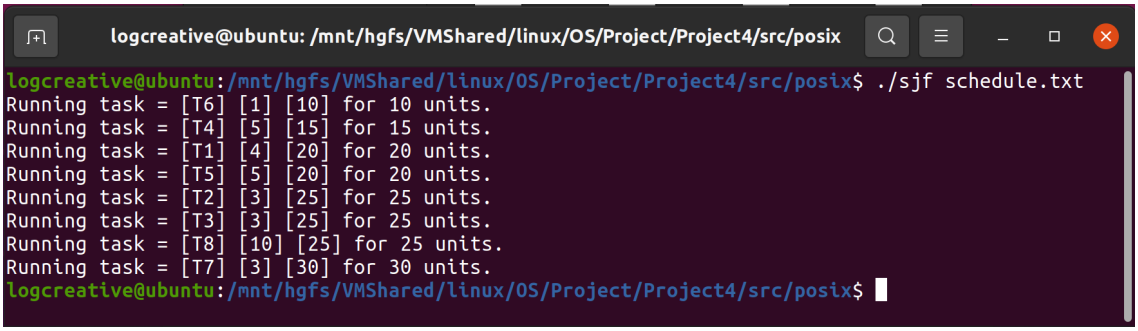
2. SJF

Listing 2: src/posix/schedule_sjf.c

```
#include "schedulers.h"
#include "list.h"
#include "cpu.h"
#include <stdio.h>
#include <stdlib.h>

struct node* taskList = NULL;

void add(char *name, int priority, int burst){
    Task* newTask = (Task*) malloc(sizeof(Task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;

    if(!taskList){
        taskList = malloc(sizeof(struct node));
        taskList->task = newTask;
```

```
        taskList->next = NULL;
    } else {
        struct node* temp = taskList;
        while(temp->next) temp = temp->next;
        insert(&temp->next,newTask);
    }

}

void schedule(){
    while(taskList){
        struct node *minNode = taskList;

        struct node *pivot = taskList;
        while(pivot){
            if(pivot->task->burst<minNode->task->burst)
                minNode = pivot;
            pivot = pivot->next;
        }

        Task* thisTask = minNode->task;
        run(thisTask,thisTask->burst);
        delete(&taskList, thisTask);
    }
}
```

SJF 算法要求首先处理运行时间短的任务。这里采用了调度时寻找时间最短任务方法。每次对链表进行最小值搜索，存入 `minNode` 中，使用小于号即可保证取的是同最小值的最先值。



3. RR

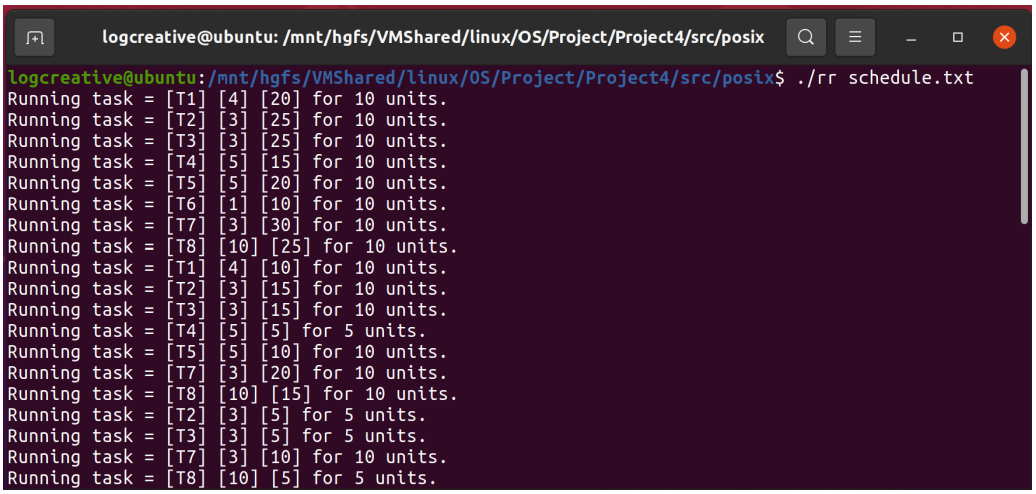
Listing 3: src/posix/schedule_rr.c

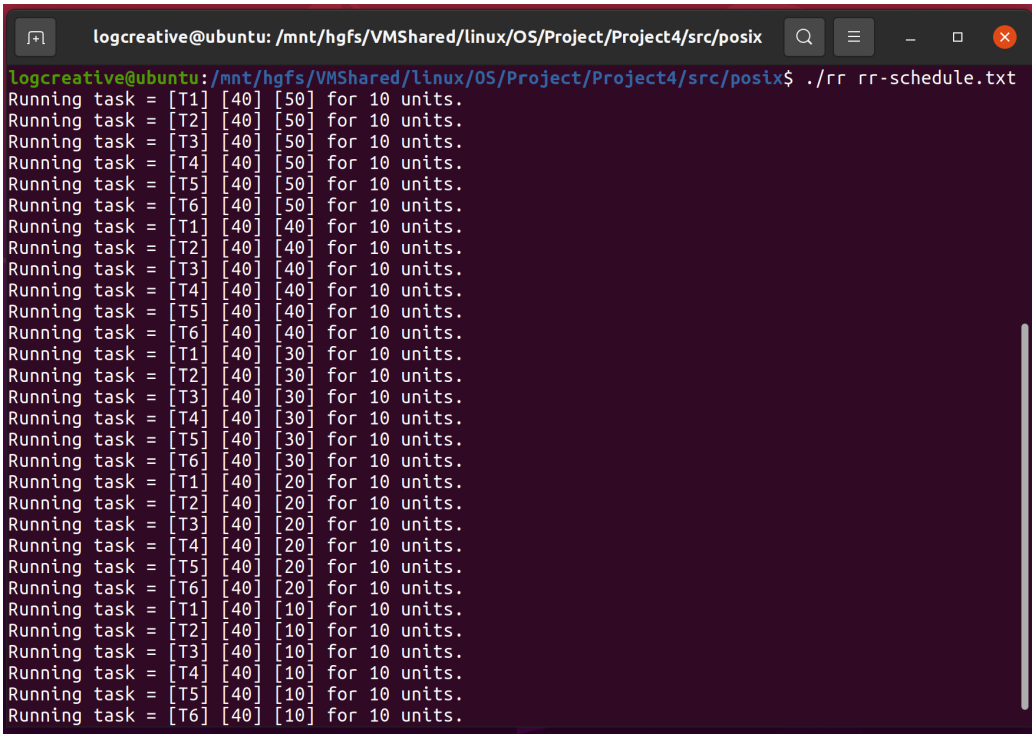
```
#include "schedulers.h"
#include "list.h"
#include "cpu.h"
#include <stdio.h>
#include <stdlib.h>

struct node* taskList = NULL;

void add(char *name, int priority, int burst){
    Task* newTask = (Task*) malloc(sizeof(Task));
    newTask->name = name;
    newTask->priority = priority;
```

```
    newTask->burst = burst;

    if(!taskList){
        taskList = malloc(sizeof(struct node));
        taskList->task = newTask;
        taskList->next = NULL;
    } else {
        struct node* temp = taskList;
        while(temp->next) temp = temp->next;
        insert(&temp->next,newTask);
    }

}

void schedule(){
    struct node *pivot = taskList;
    while(pivot){
        Task* thisTask = pivot->task;
        int slice = thisTask->burst>QUANTUM? QUANTUM : thisTask->burst;
        run(thisTask, slice);
        thisTask->burst -= slice;

        delete(&taskList, thisTask);
        if(thisTask->burst > 0){
            struct node* temp = taskList;
            while(temp->next) temp = temp->next;
            insert(&temp->next,thisTask);
        }
        pivot = pivot->next;
    }
}
```

轮转调度使用 FCFS 按照顺序让每一个任务运行一个时间量 QUANTUM 的时间，一个未运行结束的程序会被插入到列表的最后。使用一个 pivot 的指针追踪正在进行的任务。

该算法在标准集和特集上的运行情况分别如下：





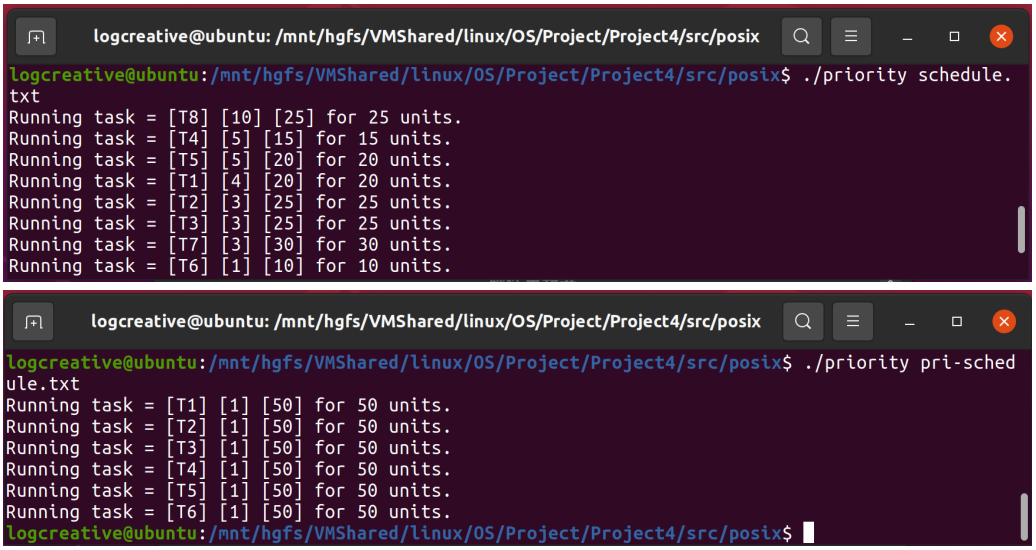
4. 优先级调度

Listing 4: src/posix/schedule_priority.c

```c
#include "schedulers.h"
#include "list.h"
#include "cpu.h"
#include <stdio.h>
#include <stdlib.h>

struct node* taskList = NULL;

void add(char *name, int priority, int burst){
    Task* newTask = (Task*) malloc(sizeof(Task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;

    if(!taskList){
        taskList = malloc(sizeof(struct node));
        taskList->task = newTask;
        taskList->next = NULL;
    } else {
        struct node* temp = taskList;
        while(temp->next) temp = temp->next;
        insert(&temp->next,newTask);
    }

}

void schedule(){
    while(taskList){
        struct node *maxpriNode = taskList;
```

```
        struct node *pivot = taskList;
        while(pivot){
            if(pivot->task->priority>maxpriNode->task->priority)
                maxpriNode = pivot;
            pivot = pivot->next;
        }

        Task* thisTask = maxpriNode->task;
        run(thisTask,thisTask->burst);
        delete(&taskList, thisTask);
    }
}
```

类似于 SJF，只不过是比较的任务中优先级的最大值，同优先级取最先进行的。该算法在标准集与特集上的运行结果分别如下：



5. 优先级–轮转调度

二