

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](#)

```
#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);

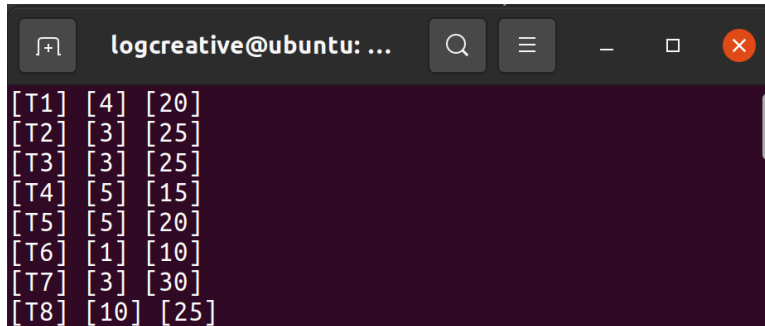
    struct node *temp = taskList;
    while(temp){
        Task* thisTask = temp->task;
```

```

        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：

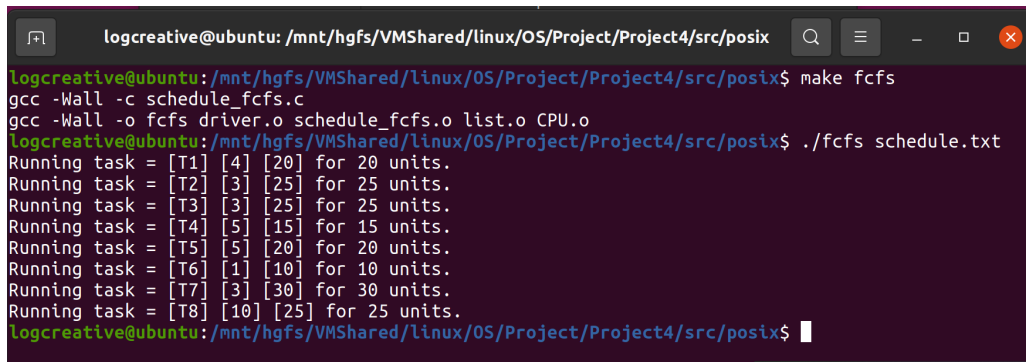


```

[T1] [4] [20]
[T2] [3] [25]
[T3] [3] [25]
[T4] [5] [15]
[T5] [5] [20]
[T6] [1] [10]
[T7] [3] [30]
[T8] [10] [25]

```

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。



```

logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$ make fcfs
gcc -Wall -c schedule_fcfs.c
gcc -Wall -o fcfs driver.o schedule_fcfs.o list.o CPU.o
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/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.
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$

```

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);
    }
}

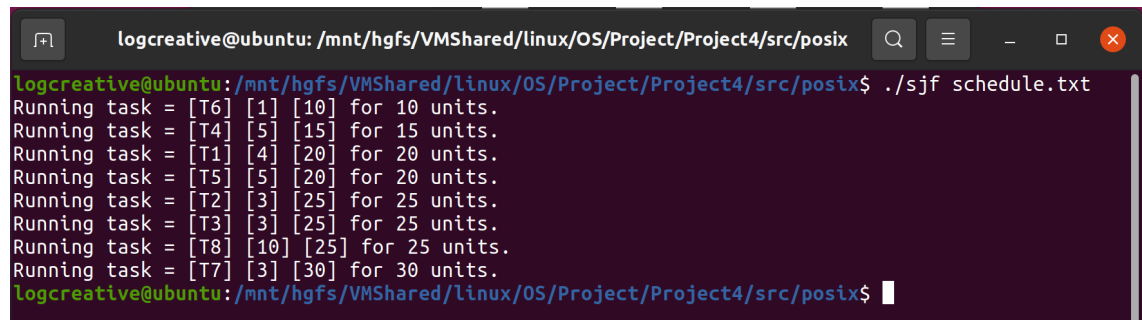
void schedule(){
    while(taskList){
        struct node *minNode = taskList;

        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。



```

logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$ ./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.
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$

```

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);
}
}

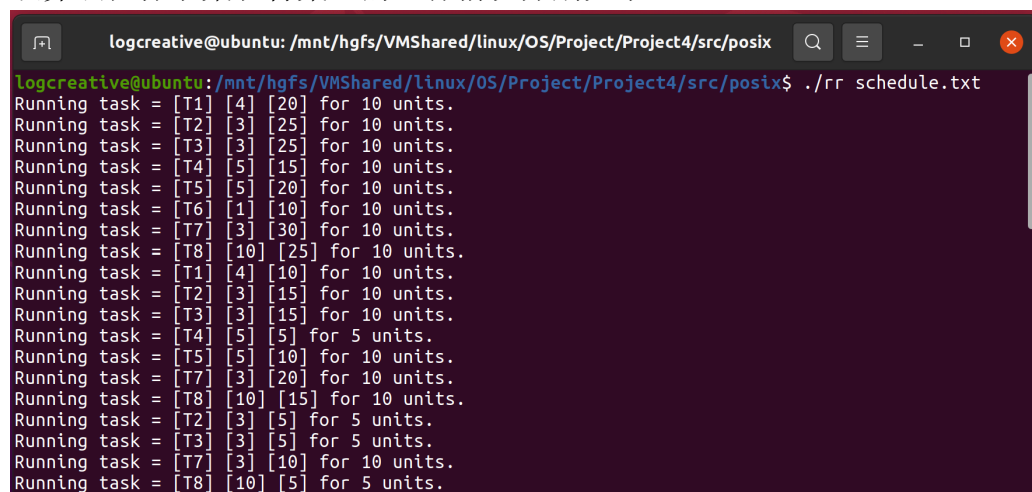
void schedule(){
    struct node *pivot = taskList;
    while(pivot){
        Task* thisTask = pivot->task;
        int slice = thisTask->burst>QUANTUM? QUANTUM : thisTask->burst;
        run(thisTask, slice);
        thisTask->burst -= slice;

        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：



```

logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/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 = [T7] [3] [10] for 10 units.
Running task = [T8] [10] [5] for 5 units.

```

```
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$ ./rr rr-schedule.txt
Running task = [T1] [40] [50] for 10 units.
Running task = [T2] [40] [50] for 10 units.
Running task = [T3] [40] [50] for 10 units.
Running task = [T4] [40] [50] for 10 units.
Running task = [T5] [40] [50] for 10 units.
Running task = [T6] [40] [50] for 10 units.
Running task = [T1] [40] [40] for 10 units.
Running task = [T2] [40] [40] for 10 units.
Running task = [T3] [40] [40] for 10 units.
Running task = [T4] [40] [40] for 10 units.
Running task = [T5] [40] [40] for 10 units.
Running task = [T6] [40] [40] for 10 units.
Running task = [T1] [40] [30] for 10 units.
Running task = [T2] [40] [30] for 10 units.
Running task = [T3] [40] [30] for 10 units.
Running task = [T4] [40] [30] for 10 units.
Running task = [T5] [40] [30] for 10 units.
Running task = [T6] [40] [30] for 10 units.
Running task = [T1] [40] [20] for 10 units.
Running task = [T2] [40] [20] for 10 units.
Running task = [T3] [40] [20] for 10 units.
Running task = [T4] [40] [20] for 10 units.
Running task = [T5] [40] [20] for 10 units.
Running task = [T6] [40] [20] for 10 units.
Running task = [T1] [40] [10] for 10 units.
Running task = [T2] [40] [10] for 10 units.
Running task = [T3] [40] [10] for 10 units.
Running task = [T4] [40] [10] for 10 units.
Running task = [T5] [40] [10] for 10 units.
Running task = [T6] [40] [10] for 10 units.
```

4. 优先级调度

Listing 4: `src/posix/schedule_priority.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：

```

logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$ ./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.

logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$ ./priority pri-schedule.txt
Running task = [T1] [1] [50] for 50 units.
Running task = [T2] [1] [50] for 50 units.
Running task = [T3] [1] [50] for 50 units.
Running task = [T4] [1] [50] for 50 units.
Running task = [T5] [1] [50] for 50 units.
Running task = [T6] [1] [50] for 50 units.
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$

```

5. 优先级-轮转调度

Listing 5: [src/posix/schedule_priority_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->next->task->priority >= newTask->priority)
    temp = temp->next;

if(temp == taskList && temp->task->priority < newTask->priority){
    taskList = malloc(sizeof(struct node));
    taskList->task = newTask;
    taskList->next = temp;
}
else if(!temp->next){
    struct node* newNode = malloc(sizeof(struct node));
    newNode->task = newTask;
    newNode->next = NULL;
    temp->next = newNode;
}
else 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)
            add(thisTask->name,thisTask->priority,thisTask->burst);
        pivot = pivot->next;
    }
}
}

```

优先级-轮转调度需要同时考虑两者，则在开始的加入 add 的时候就会对优先级进行排序，然后不断在本优先级轮转，运行完毕后会同样调用 add 函数插入，轮转本优先级后轮转下一优先级，直到所有的任务都被执行完毕。这里插入需要考虑更多的情况，仅仅基于给定的 list 实现方法。该算法在标准集和特集上的运行结果分别如下：

```

logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Projec...
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/OS/Project/Project4/src/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 = [T4] [5] [5] for 5 units.
Running task = [T5] [5] [10] 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 = [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.

```

```
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$ ./priority_rr pri-schedule.txt
Running task = [T1] [1] [50] for 10 units.
Running task = [T2] [1] [50] for 10 units.
Running task = [T3] [1] [50] for 10 units.
Running task = [T4] [1] [50] for 10 units.
Running task = [T5] [1] [50] for 10 units.
Running task = [T6] [1] [50] for 10 units.
Running task = [T1] [1] [40] for 10 units.
Running task = [T2] [1] [40] for 10 units.
Running task = [T3] [1] [40] for 10 units.
Running task = [T4] [1] [40] for 10 units.
Running task = [T5] [1] [40] for 10 units.
Running task = [T6] [1] [40] for 10 units.
Running task = [T1] [1] [30] for 10 units.
Running task = [T2] [1] [30] for 10 units.
Running task = [T3] [1] [30] for 10 units.
Running task = [T4] [1] [30] for 10 units.
Running task = [T5] [1] [30] for 10 units.
Running task = [T6] [1] [30] for 10 units.
Running task = [T1] [1] [20] for 10 units.
Running task = [T2] [1] [20] for 10 units.
Running task = [T3] [1] [20] for 10 units.
Running task = [T4] [1] [20] for 10 units.
Running task = [T5] [1] [20] for 10 units.
Running task = [T6] [1] [20] for 10 units.
Running task = [T1] [1] [10] for 10 units.
Running task = [T2] [1] [10] for 10 units.
Running task = [T3] [1] [10] for 10 units.
Running task = [T4] [1] [10] for 10 units.
Running task = [T5] [1] [10] for 10 units.
Running task = [T6] [1] [10] for 10 units.
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$
```