

## 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;
int value = 0;

void add(char *name, int priority, int burst){
    Task* newTask = (Task*) malloc(sizeof(Task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;
    newTask->tid = __sync_fetch_and_add(&value,1);
    newTask->exec = 0;

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

```

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：

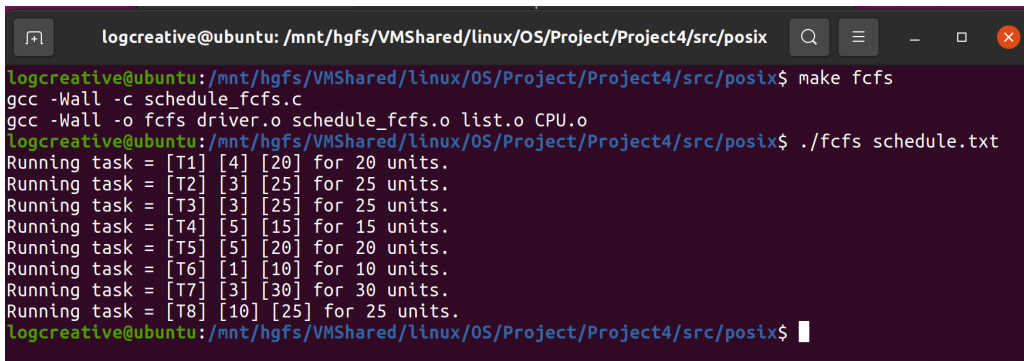


```

[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>

int value = 0;
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;
}

```

```

newTask->tid = __sync_fetch_and_add(&value,1);
newTask->exec = 0;

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);
}
}

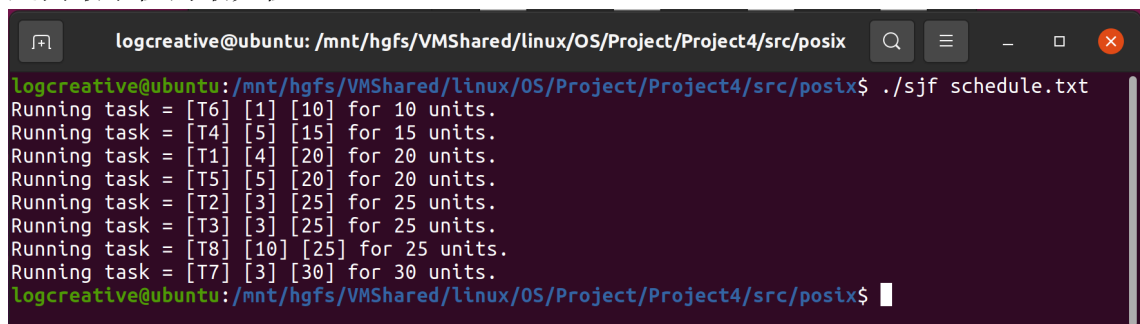
void schedule(){
    while(taskList){
        struct node *minNode = taskList;

        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>

```

```

int value = 0;
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;
    newTask->tid = __sync_fetch_and_add(&value,1);
    newTask->exec = 0;

    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);

        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$ ./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>

int value = 0;
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;
    newTask->tid = __sync_fetch_and_add(&value,1);
```

```

newTask->exec = 0;

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-sched
ule.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>

int value = 0;
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;
    newTask->tid = __sync_fetch_and_add(&value,1);
    newTask->exec = 0;

    addTask(newTask);
}

void addTask(Task* newTask){
    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;

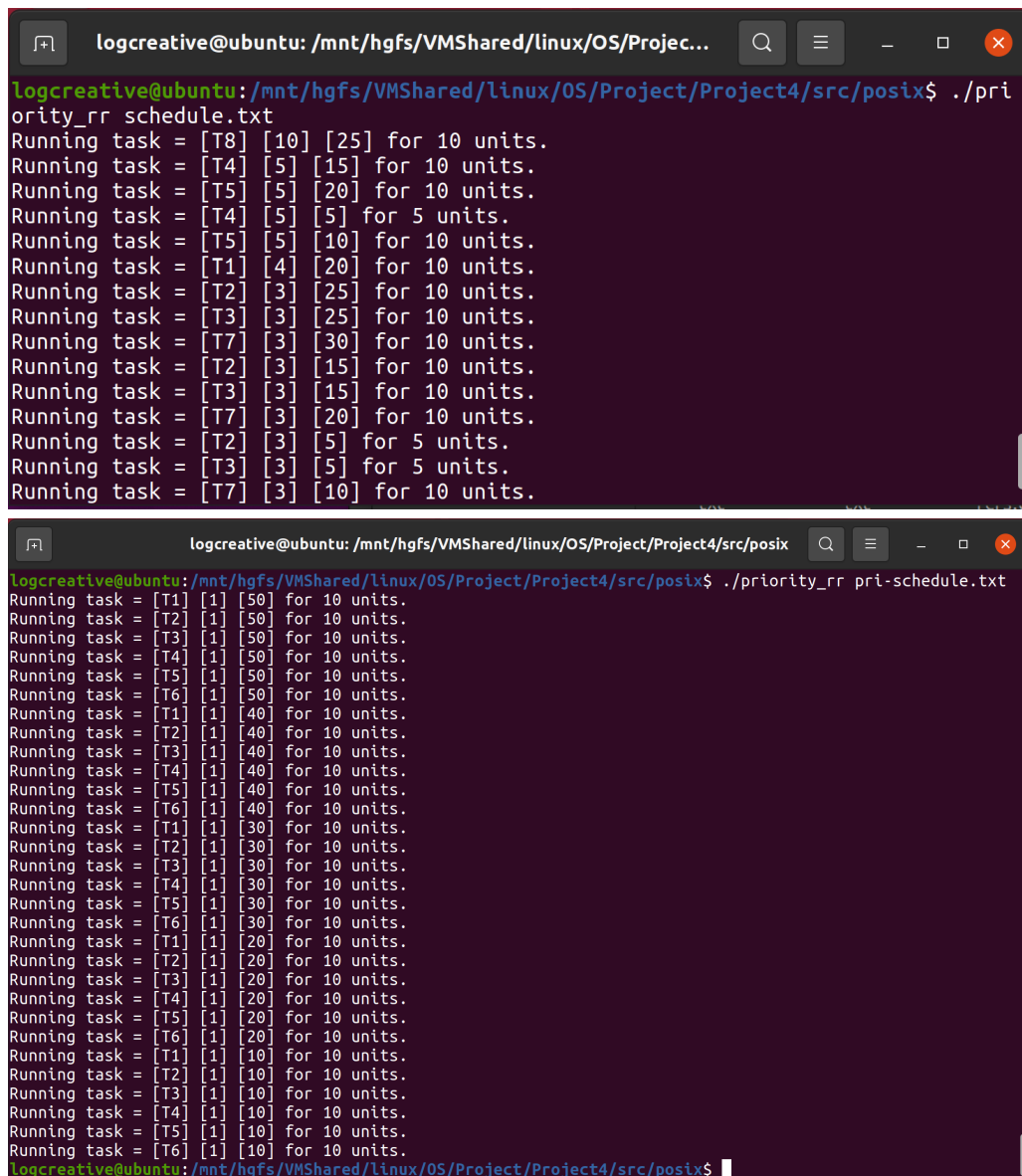
        if(temp == taskList && temp->task->priority < newTask->priority){
            taskList = malloc(sizeof(struct node));
            taskList->task = newTask;
            taskList->next = temp;
        }
        else if(!temp->next){
            struct node* newNode = malloc(sizeof(struct node));
            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);

        delete(&taskList, thisTask);
        if(thisTask->burst > 0)
            addTask(thisTask);
        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$ ./pri
ority_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.
```

## 二 Java 语言实现版本

“may be completed in either C or Java”，所以只需要使用一种语言完成即可。

### 三 1. 增加原子整数的 tid。在 schedule 外添加

```
int value = 0;
```

在新建任务时，添加

```
newTask->tid = __sync_fetch_and_add(&value,1);
```



即可，以上的程序皆按照这种方式修复。运行正常。

2. 为了计算周转时间、等待时间和相应时间，需要修改一些文件的定义。  
添加 `task` 类型的一个属性 `exec` 用于计算上一次运行结束的时刻，初始会在相应的初始化阶段被赋值为 0。

Listing 6: [src/posix/task.h](#)

```
/**
 * Representation of a task in the system.
 */

#ifndef TASK_H
#define TASK_H

// representation of a task
typedef struct task {
    char *name;
    int tid;
    int priority;
    int burst;
    int exec;
} Task;

#endif
```

重点修改 CPU 部分的定义。添加 CPU 时钟 `clock`，当一个任务即将运行的时候，如果从未运行过，也即 `exec` 是 0，就会添加任务数，并且计入相应时间。等待时间是当前时刻与上一时间片执行完毕时刻的差值。执行完毕后，增加时钟时间，修改剩余时间，赋值该事件片执行结束时间，如果没有剩余时间，就将当前的时钟时间计入周转时间。

Listing 7: [src/posix/cpu.c](#)

```
/**
 * "Virtual" CPU that also maintains track of system time.
 */

#include <stdio.h>

#include "task.h"

int clock = 0;

int tasknum = 0;

int turnaround = 0;
int waiting = 0;
int response = 0;

// run this task for the specified time slice
void run(Task *task, int slice) {

    if(task->exec==0){
```

```

        tasknum++;
        response += clock;
    }
    waiting += clock - task->exec;

    printf("Running task = [%s] [%d] [%d] for %d units.\n", task->name, task->priority, task->burst,
           slice);

    clock += slice;
    task->burst -= slice;
    task->exec = clock;
    if(task->burst==0) turnaround += clock;
}

void printAvg(){
    printf("Task Num:\t%d\n", tasknum);
    printf("Turnaround Avg:\t%2f\n", (double) turnaround / tasknum);
    printf("Waiting Avg:\t%2f\n", (double) waiting / tasknum);
    printf("Response Avg:\t%2f\n", (double) response / tasknum);
}

```

修改 driver.c 在 schedule() 结束后调用 printAvg() 函数，打印对应的时间。  
运行示例如下：

```

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.
Running task = [T6] [1] [10] for 10 units.
Task Num: 8
Turnaround Avg: 75.625000
Waiting Avg: 65.625000
Response Avg: 50.625000
logcreative@ubuntu: /mnt/hgfs/VMShared/linux/OS/Project/Project4/src/posix$

```

所有的算法在标准集上的结果如下：

算法	周转时间	等待时间	响应时间
FCFS	94.375	73.125	73.125
SJF	82.500	61.250	61.250
优先级	96.250	75.000	75.000
轮转	128.750	107.500	35.000
优先级-轮转	75.625	65.625	50.625

