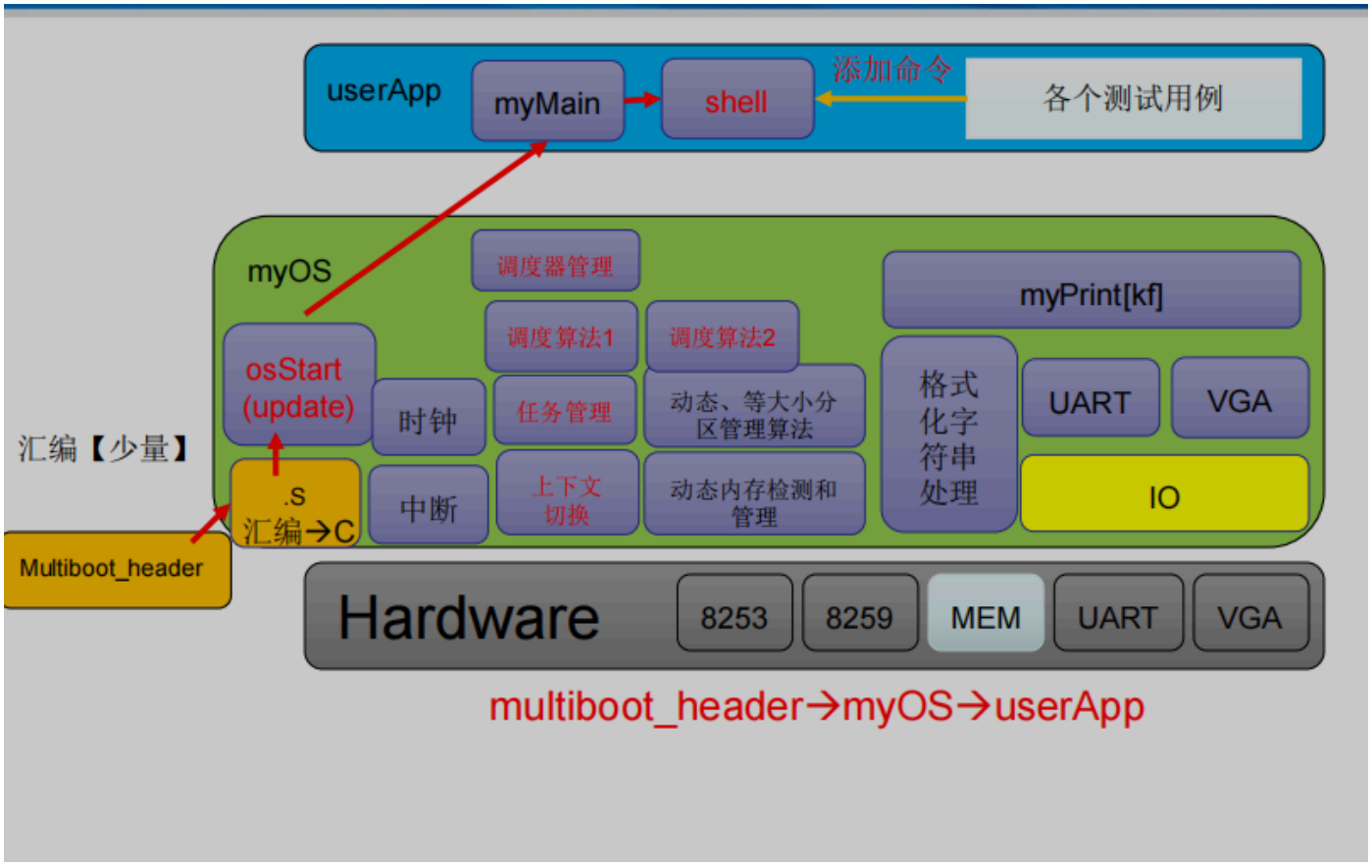
实验6 调度算法

原理说明



上面的图片来自于老师给出的实验PPT，较上一个实验相比，增加了各调度算法模块以及调度器的接口，以实现各调度算法。

本次实验的主流程为：由Multiboot_header引导程序启动->osStart->初始化时钟并开中断，初始化任务管理->进入用户程序运行，实现各测试用调度算法或启动Shell。

主要功能模块实现

我的实验6基于助教给出的框架代码完成，各全局变量均未做修改，在打印任务信息的函数中又增加了打印运行时间的部分以便于调试。

1.FCFS调度相关

```
void taskDequeue_FCFS() {
    // 实现出队操作，调度方式为FCFS
    if(rqFCFSIsEmpty()){
        return;
    }
    if(rqFCFS.head == rqFCFS.tail){
        rqFCFS.head->TSK_State = TSK_WAIT;
        rqFCFS.head = rqFCFS.tail = NULL;
    }
    else {
        myTCB* tmp = rqFCFS.head;
        rqFCFS.head->TSK_State = TSK_WAIT;
        rqFCFS.head = tmp->nextTCB;
        kfree((unsigned long)tmp);
    }
}
struct myTCB* nextTask_FCFS() {
    // 获取下一个 FCFS 调度的任务
    if(rqFCFSIsEmpty()){
        return NULL;
    }
    else {
        return rqFCFS.head;
    }
}
```

这两个函数的实现于上一个实验中已经说明过，这里不再赘述。

2.RR调度相关

```
struct myTCB* nextTask_RR() {
    // 以 RR 调度算法，获取下一个任务
    if(rqFCFSIsEmpty()){
        return NULL;
    }
    else {
        return rqFCFS.head;
    }
}
void RR_hook() {
    // RR 调度的 HOOK 函数
    if((currentTask!=NULL)&&(getTickTime()%100 == 0)){
        currentTask->thisRunTime++;
    }
    if((currentTask->thisRunTime >=2)&&(getTickTime()%100 == 0)){
        currentTask->thisRunTime = 0;
        taskEnqueue_FCFS(currentTask);
        taskEnd();
    }
}
```

```
void schedule_RR(void){
    // RR调度算法
    while(1) {
        currentTask=NULL;
        struct myTCB* nextTask;
        nextTask=nextTask_FCFS();
        if(nextTask) {
            taskDequeue_FCFS();
            currentTask=nextTask;
            context_switch(&BspContext,currentTask->stackTop);
        }
    }
}
```

RR调度的nextTask()和schedule()和FCFS在逻辑上是一致的，最为核心的部分是RR_hook()函数。RR_hook()函数会维护currentTask的thisRuntime值并适时的进行入队出队的维护，同时提供抢占调度的算法。

3.优先级调度相关

```
void taskEnqueue_PRIO(myTCB *task) {
    // 实现按照优先级调度算法，加入就绪队列
    unsigned int priority = task->taskPara->priority;
    if(rqFCFSIsEmpty()){
        rqFCFS.head = task;
        rqFCFS.tail = task;
        rqFCFS.tail->nextTCB = NULL;
    }
    else if(rqFCFS.head == rqFCFS.tail){
        if(priority < rqFCFS.head->taskPara->priority){
            task->nextTCB = rqFCFS.head;
            rqFCFS.head = task;
        }
        else {
            rqFCFS.tail->nextTCB = task;
            rqFCFS.tail = task;
            rqFCFS.tail->nextTCB = NULL;
        }
    }
    else {
        myTCB* prev = rqFCFS.head;
        myTCB* curr = rqFCFS.head;
        if(priority < rqFCFS.head->taskPara->priority){
            task->nextTCB = rqFCFS.head;
            rqFCFS.head = task;
            return;
        }
        while (curr->nextTCB)
        {
            if(priority > curr->taskPara->priority){
                prev = curr;
                curr = curr->nextTCB;
            }
```

```
            else {
                task->nextTCB = curr;
                prev->nextTCB = task;
            }
        }
        rqFCFS.tail->nextTCB = task;
        rqFCFS.tail = task;
        rqFCFS.tail->nextTCB = NULL;
    }
}
```

实际上这个过程是向有序链表中插入有特定值的结点的过程，即链表的按序插入。需要考虑队列为空，只有一个元素，有多个元素的情况；对于有多个元素的情况，只需要按照最常见的双指针实现链表的插入排序即可实现。

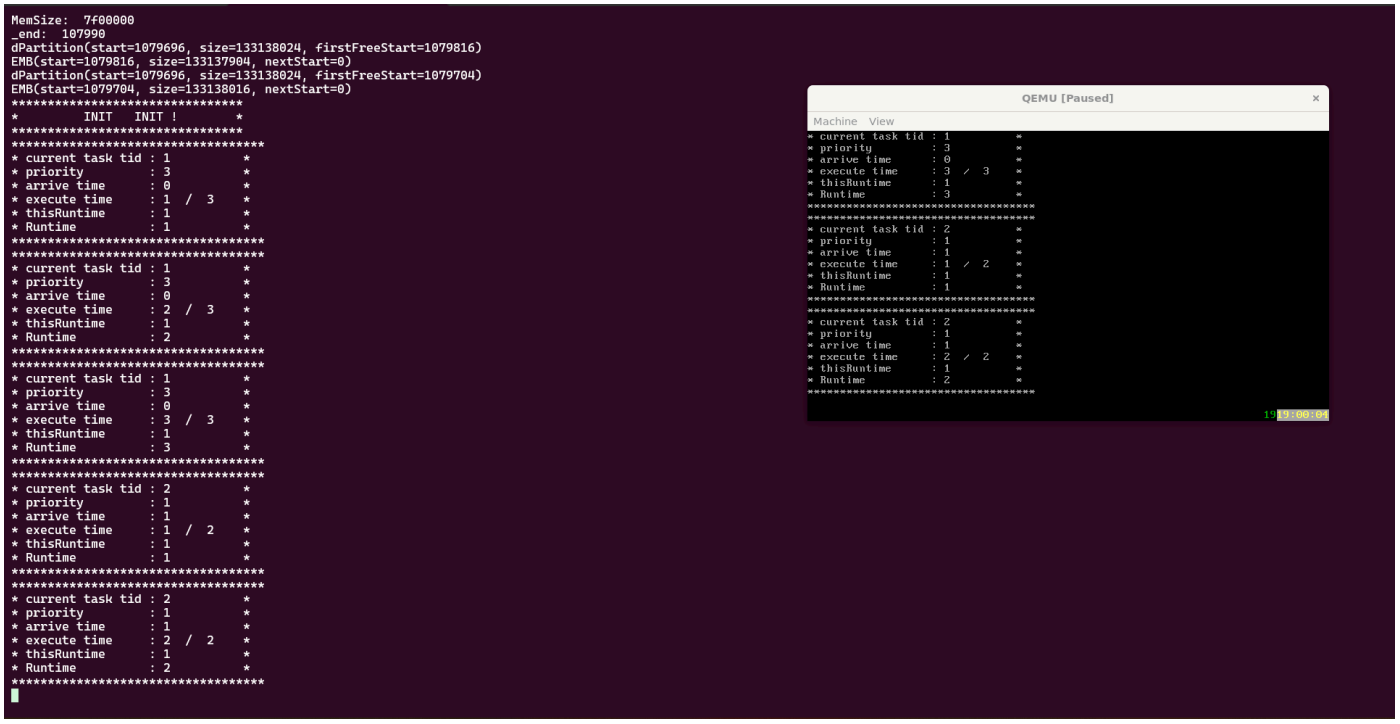
4.任务参数相关

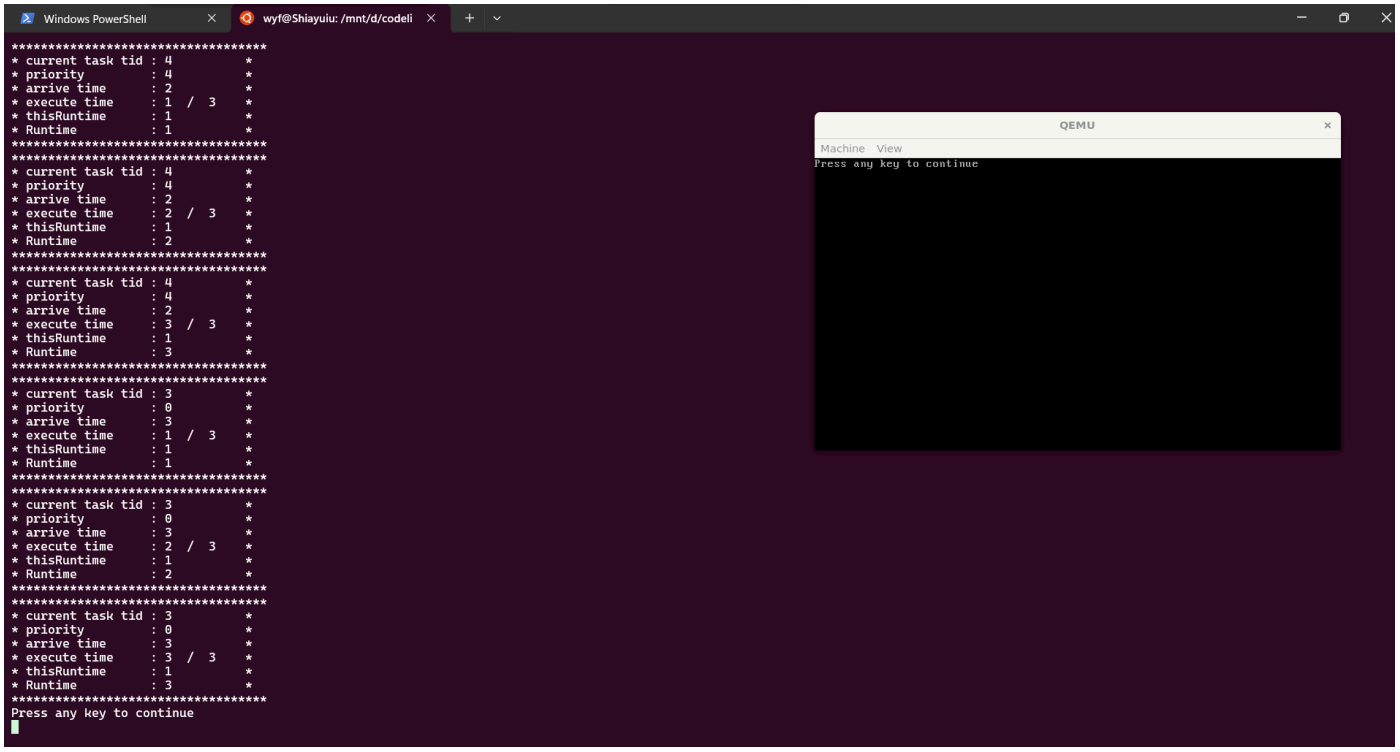
```
void setTaskPara(unsigned int tag,unsigned int value,struct taskPara* buffer) {
    // 实现设置任务参数操作
    switch (tag) {
        case PRIO:
            buffer->priority = value;
            break;
        case ARRTIME:
            buffer->arrTime = value;
            break;
        case EXETIME:
            buffer->exetime = value;
            break;
    }
}
unsigned int getTaskPara(int tag,struct taskPara *para) {
    switch (tag) {
        case PRIO:
            return para->priority;
            break;
        case ARRTIME:
            return para->arrTime;
            break;
        case EXETIME:
            return para->exetime;
            break;
        default:
            return 0;
    }
}
void addRunTime() {
    // 增加当前任务的运行时间
    if((currentTask!=NULL)&&(getTickTime()%100==0))
        currentTask->runTime++;
}
```

运行结果及说明

在wsl的Ubuntu环境下运行，得到的结果如下：

1.FCFS调度（按1->2->4->3的执行顺序）

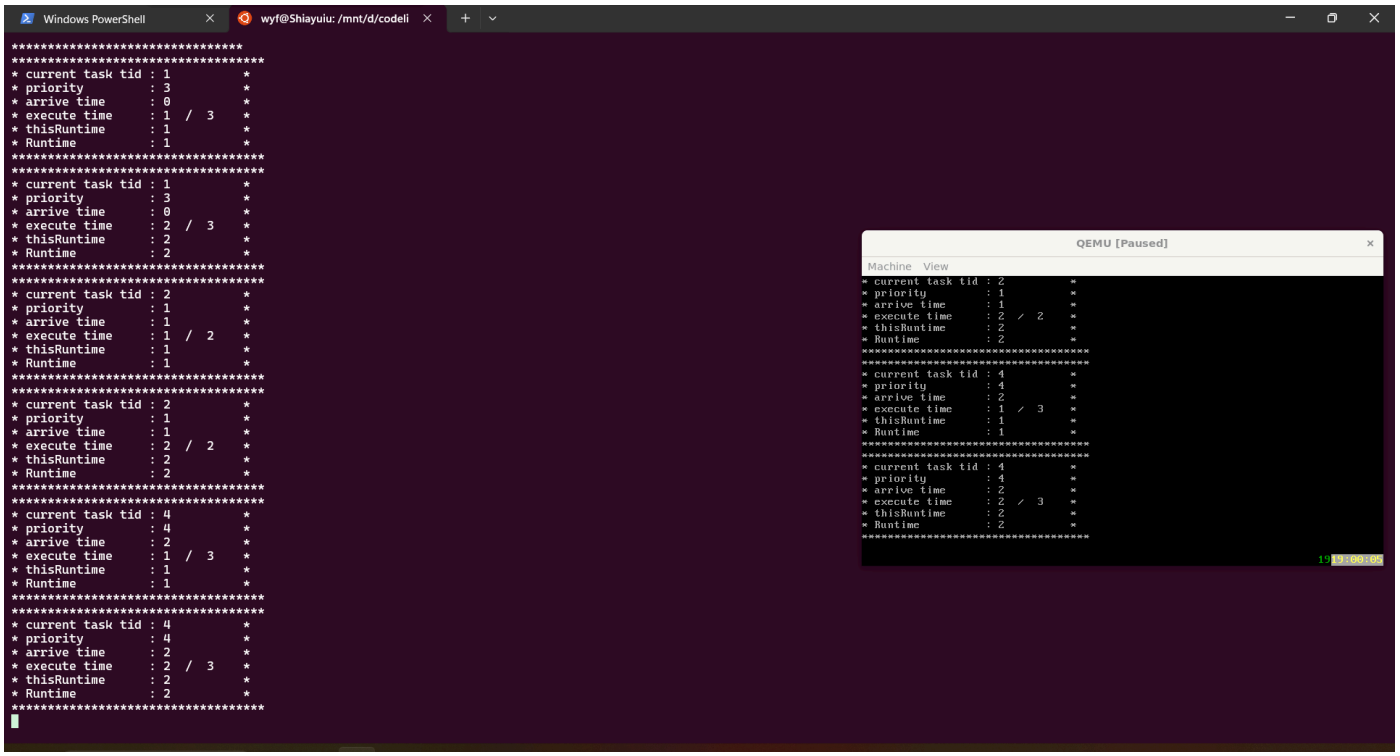


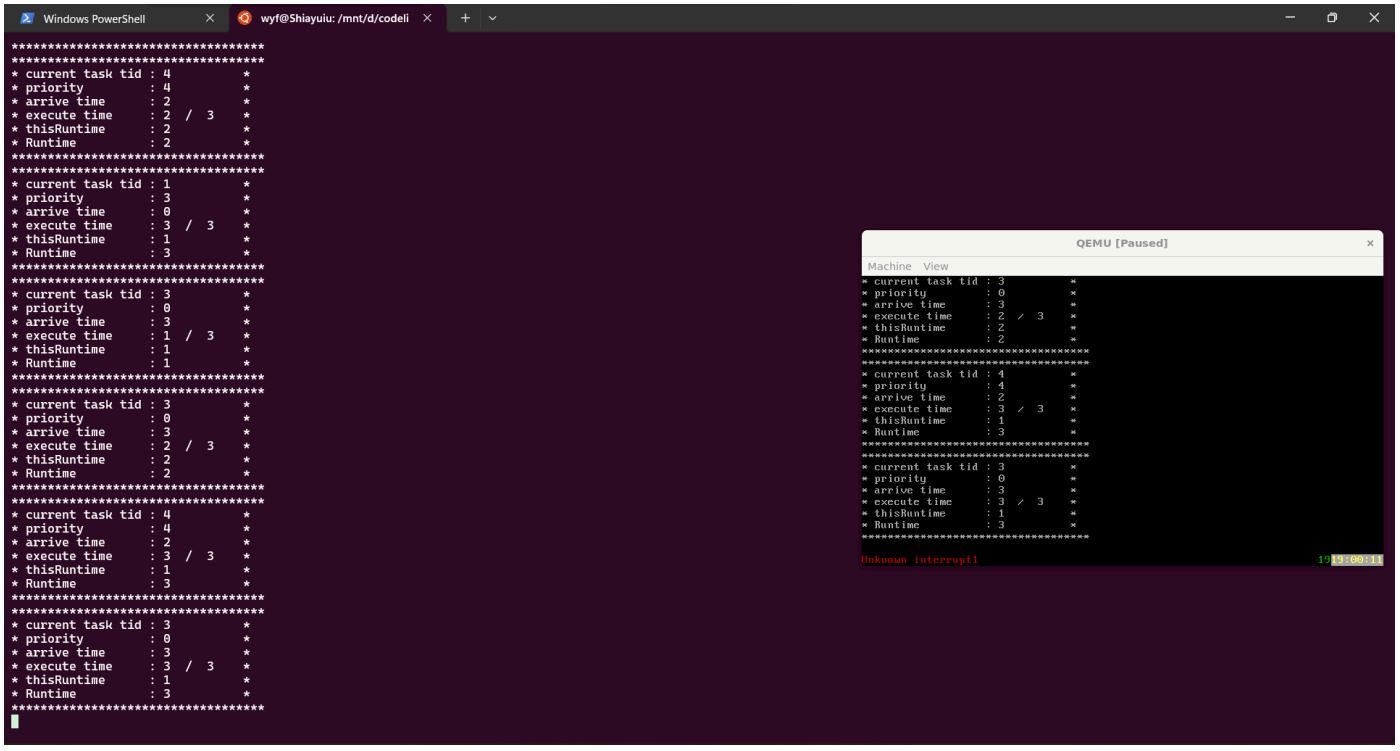


2.RR调度

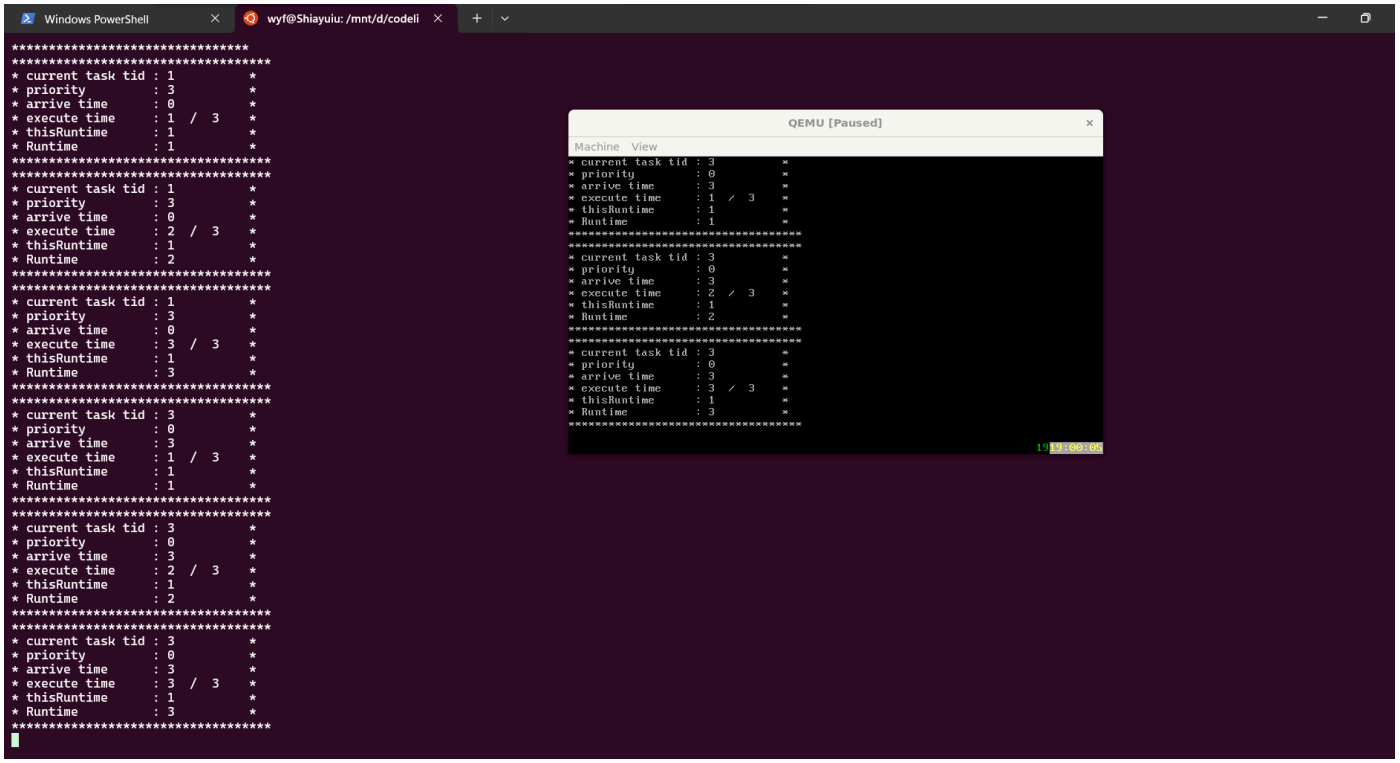
根据时间片轮转调度，任务在执行2个时间后到队尾，此时队列顺序241。 执行任务2，任务2执行2个时间结束，此时队列顺序413。 执行任务4，任务4执行2个时间后到队

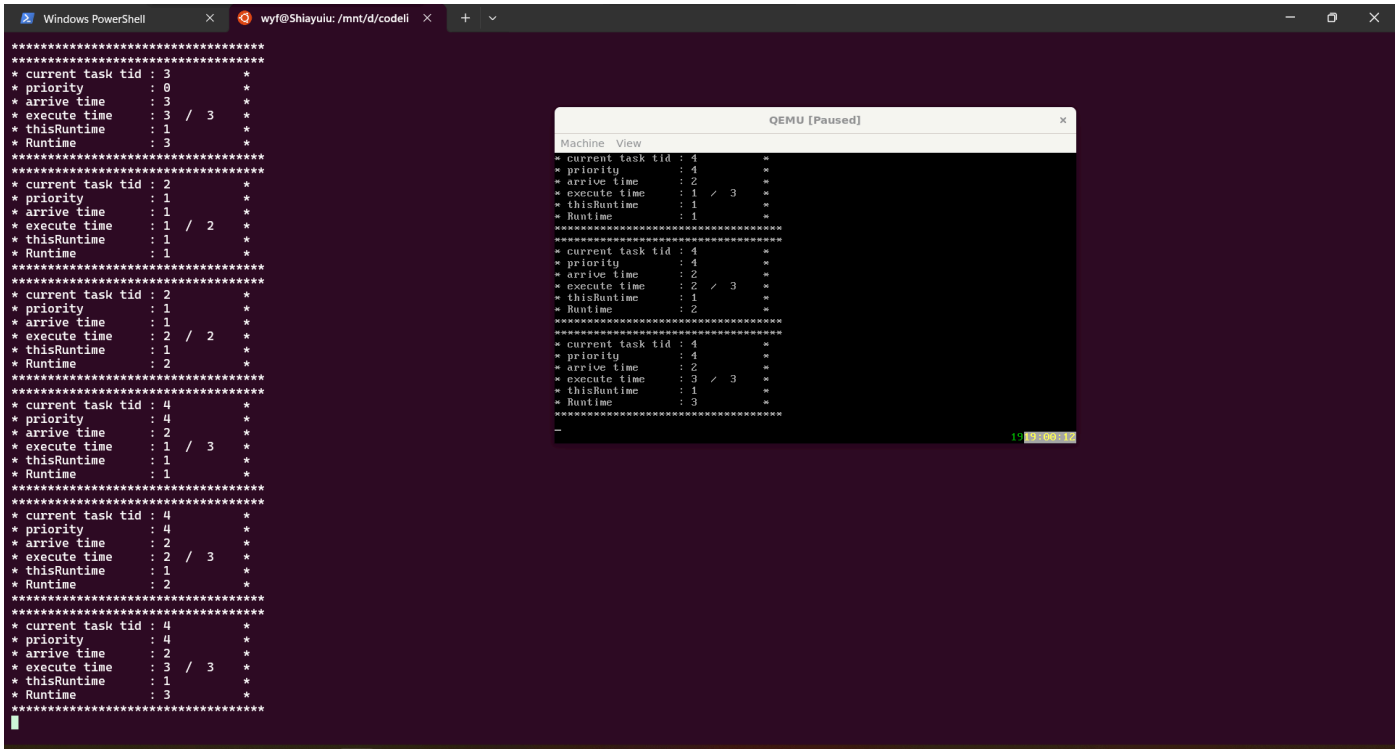
尾，此时队列顺序134。 执行任务1，1个时间后结束。再执行任务3，执行2个时间后到队尾，此时队列顺序43。 执行任务4，1个时间后结束，再执行任务3，一个时间后结束



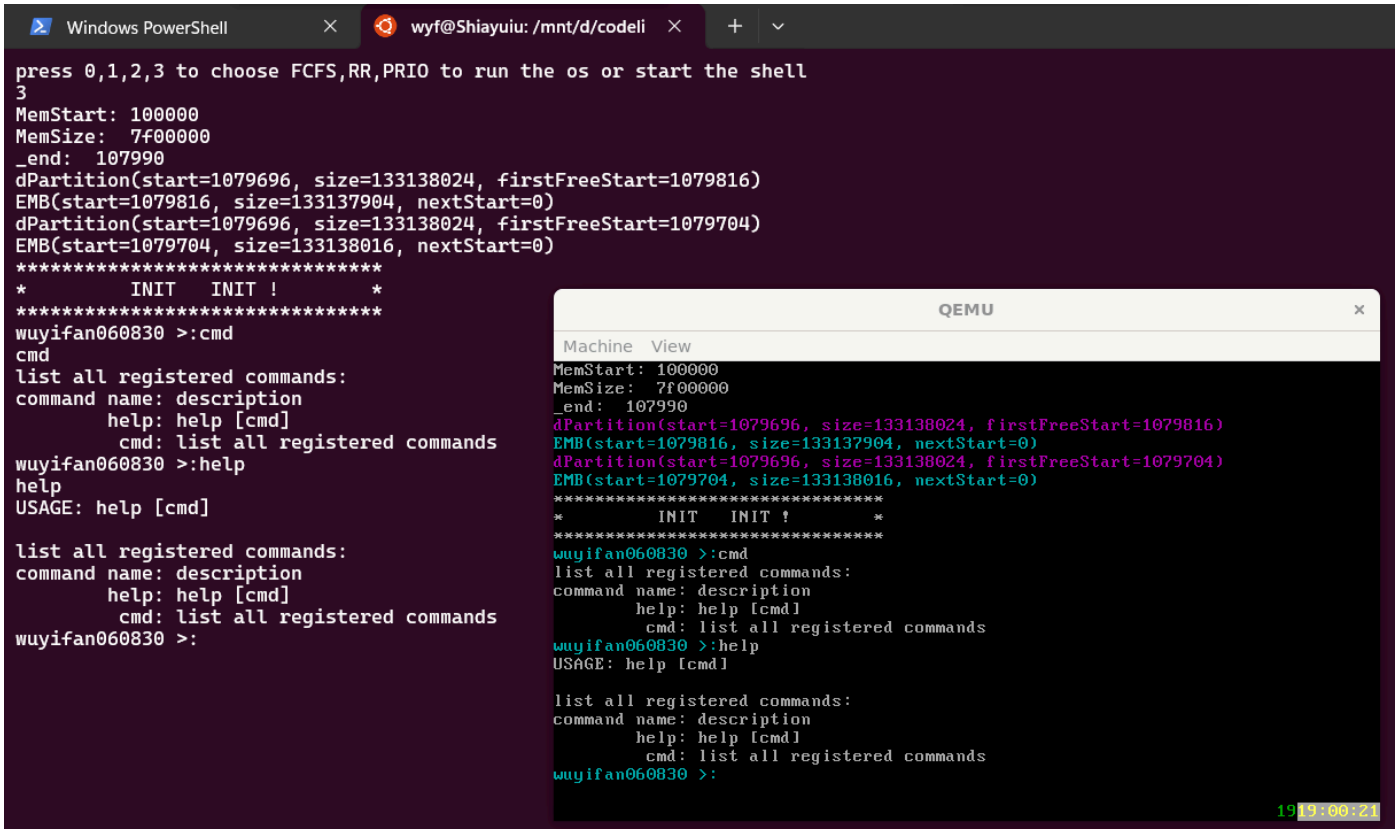


3.优先级调度（按1->3->2->4的执行顺序）


Windows PowerShell
wyf@Shiayuiu: /mnt/d/codeli
QEMU [Paused]
Machine View


Windows PowerShell
wyf@Shiayuiu: /mnt/d/codeli
QEMU [Paused]
Machine View

4.shell



以上四个部分的测试样例运行结果均符合预期。