# OS_Lab_06_Report

## PB21111686_赵卓

### 实验题目

- 实现除 FCFS 之外的另外两种调度。

### 实验目的

- 在第五次实验的基础上，添加另外两种调度方式（包含抢占式）。

### 实验内容

- 本次实验在 Lab5 的基础上完成，实现了文档中要求的抢占式 RR 轮转调度和非抢占式 SJF 调度（即 PRIO 调度）。

### 实验模块实现

- hook 机制的实现
  在前面的实验实现时钟时已经使用到了 hook 机制。在本次实验中，我们遵循机制与策略相分离的原则，使用hook机制来提供抽象接口，适用于所有的调度算法。为此我们定义 scheduler 结构体，同时定义全局变量 scheduler sysSch ，它向外提供一个初始化函数接口，其功能是将具体的调度算法接口连接到抽象接口上，并在vga屏幕底端显示当前调度算法类型。此函数根据 sysSch.type 判断所要初始化的调度算法，而 sysSch.type 将在系统启动时，由用户输入数字选择。scheduler 定义如下：

```
typedef struct scheduler {
unsigned int type;
myTCB * (*nextTsk)(void);
void (*enqueueTsk)(myTCB *tsk);
myTCB * (*dequeueTsk)(void);
void (*schedulerInit)(void);
void (*schedule)(void);
void (*tick_hook)(void);
} scheduler;
```

- 抢占式 RR 轮转调度
  - RR 调度给每个进程分配相同大小的时间片，轮流执行，当时间片结束时，抢占调度下一个进程。调度实现函数如下：

```
void scheduleRR(void) {
        while (1) {

                myTCB *nextTsk;
                int idleTid;

                if (taskQueueFIFOEmpty(rdyQueueRR)) {
                        if (!idleTsk)
                                idleTid = createTsk(idleTskBody);

                        nextTsk = idleTsk = tcbPool[idleTid];
                }
                else
                        nextTsk = dequeueTskRR();

                if (nextTsk == idleTsk && currentTsk == idleTsk) {
                        continue;
                }

                if (currentTsk) {
                        if (currentTsk->para->exec_time * TICK_FREQ
            <= currentTsk->runTime || currentTsk == idleTsk)
                                destroyTsk(currentTsk->tid);
                        if (currentTsk == idleTsk)
                                idleTsk = NULL;
                }

                nextTsk->state = TSK_RUNNING;
                currentTsk = nextTsk;

                currentTsk->lastScheduledTime = get_current_time();
                context_switch(&BspContext, currentTsk->stkTop);
        }
}
```

  - 为了实现抢占，在时间片结束时，每次tick发生时，检查当前任务的运行时间，如果可以整除时间片大小，且当前时间大于此任务最近一次被调度的时间，则进行抢占调度。这样就可以在时间片未用完时发生抢占，实现如下：

```
void preemptCurrentTaskRR_hook(void) {
if (currentTsk == idleTsk)
        return;

if (currentTsk->runTime % (TIME_SLICE * TICK_FREQ) == 0
&& get_current_time() > currentTsk->lastScheduledTime) {
        currentTsk->state = TSK_READY;
        enqueueTskRR(currentTsk);
        context_switch(&currentTsk->stkTop, BspContext);
}
}
```

- 非抢占式 SJF 调度
  - 优先队列。SJF 调度需要维护一个优先队列，根据各进程的优先级从队列中调度进程执行。我们定义该队列如下，其中 tcb 是队列中 TCB 的指针的顺序表，动态分配；tail 是队尾；capacity 是队列容量；cmp 是比较函数。

    ```
    typedef struct taskQueuePrio {
    myTCB **tcb;
    int tail;
    int capacity;
    int (*cmp)(const myTCB *a, const myTCB *b);
    } taskQueuePrio;
    ```

    同时还定义了一系列操作队列的 API 如下，可以进行出队、入队、判断队空等操作。

    ```
    void taskQueuePrioInit(taskQueuePrio **queue, int capacity,
    int (*cmp)(const myTCB *a, const myTCB *b));
    int taskQueuePrioEmpty(taskQueuePrio *queue);
    myTCB * taskQueuePrioNext(taskQueuePrio *queue);
    void taskQueuePrioEnqueue(taskQueuePrio *queue, myTCB *tsk);
    myTCB * taskQueuePrioDequeue(taskQueuePrio *queue);
    ```

  - SJF 调度实现。有了优先队列之后，我们只需要根据各进程的优先级比较进行调度即可。实现如下：

```
int compare_exec_time(const myTCB *a, const myTCB *b) {
    if (getTskPara(EXEC_TIME, a->para) ==
    getTskPara(EXEC_TIME, b->para))
    return getTskPara(ARRV_TIME, a->para) -
    getTskPara(ARRV_TIME, b->para);
    else
    return getTskPara(EXEC_TIME, a->para) -
    getTskPara(EXEC_TIME, b->para);
}
```

- 进程实现：
  - 动态进程实现。Lab5 中的进程提前插入就绪队列，而本次实验为了实现动态增加进程，模拟实际情况，使用新增的优先队列，作为到达队列，利用时钟中断，定期检查到达队列，当一个任务被创建，它首先应被插入到达队列。即定义到达队列如下：

    ```
    taskQueuePrio *arrvQueue;
    ```

  - 进程动态添加。有了到达队列之后，我们只需比较每个进程的到达时间来将其插入即可。为此，我们定义了一个比较函数，比较任务到达时间，每次 tick 发生，都会检测该队列，检查到达队列中到达时间最小的任务，如果当前时间不小于其到达时间，则将其取出，插入就绪队列。实现如下：

    ```
    int compare_arrv_time(const myTCB *a, const myTCB *b) {
        return getTskPara(ARRV_TIME, a->para) - getTskPara(ARRV_TIME, b-
        >para);
    }
    ```

  - 队列周期性检查。实现上述功能后，还需要一个 hook 函数，周期性检查就绪队列，实现如下：

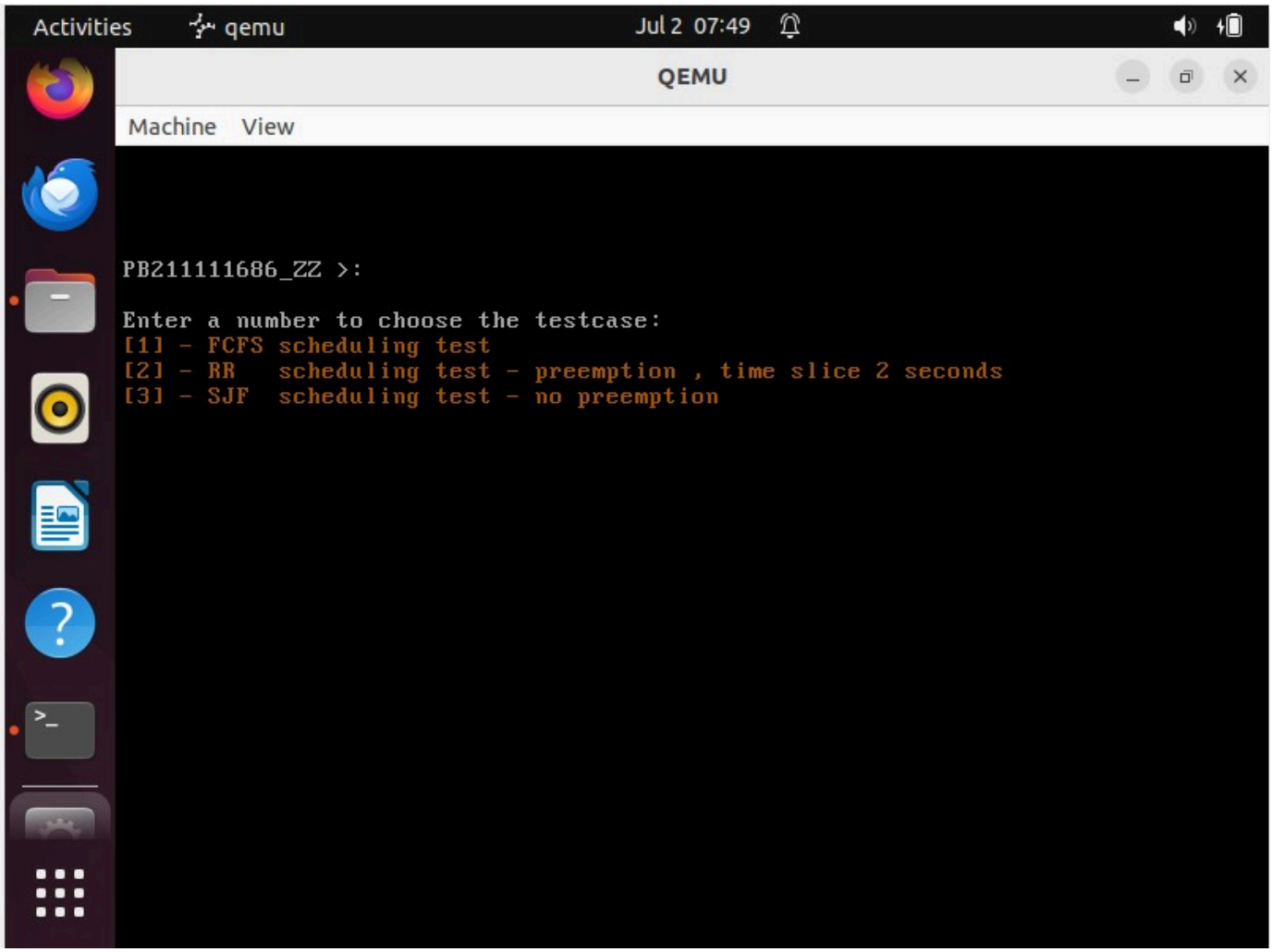
    ```
    void startArrivedTask_hook(void) {
        if (taskQueuePrioEmpty(arrvQueue))
        return;
        myTCB *nextTask = taskQueuePrioNext(arrvQueue);
        if (get_current_time() >= getTskPara(ARRV_TIME, nextTask->para)) {
        tskStart(nextTask->tid);
        taskQueuePrioDequeue(arrvQueue);
        }
    }
    ```

    在初始化时，我们对 hook 函数也进行初始化：

```
void taskArrvQueueInit(void) {
    taskQueuePrioInit(&arrvQueue, taskNum, compare_arrv_time);
    append2HookList(startArrivedTask_hook);
}
```

# 实验结果

- 运行脚本，显示个人学号和姓名缩写，以及选择调度方式的提示信息，根据信息输入指令测试：

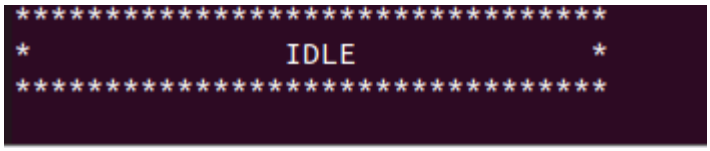



- FCFS 测试
  - 我们设置了 FCFS 测试样例如下，每个样例都有到达时间和执行时间：

```
void initFCFSCases(void) {
   int newTskTid0 = createTsk(myTskFCFS0);
   setTskPara(ARRV_TIME, 2, tcbPool[newTskTid0]->para);
   setTskPara(EXEC_TIME, 5, tcbPool[newTskTid0]->para);

   int newTskTid1 = createTsk(myTskFCFS1);
   setTskPara(ARRV_TIME, 0, tcbPool[newTskTid1]->para);
   setTskPara(EXEC_TIME, 3, tcbPool[newTskTid1]->para);

   int newTskTid2 = createTsk(myTskFCFS2);
   setTskPara(ARRV_TIME, 10, tcbPool[newTskTid2]->para);
   setTskPara(EXEC_TIME, 3, tcbPool[newTskTid2]->para);


   enableTask(newTskTid0);
   enableTask(newTskTid1);
   enableTask(newTskTid2);
 }
```

- 选择 FCFS 测试结果如下，符合测试样例的应有执行情况：

```
**************************************
   TID : 2
   Arrive Time    : 0
   Execute Time   : 1  /  3
**************************************
**************************************
   TID : 2
   Arrive Time    : 0
   Execute Time   : 2  /  3
**************************************
**************************************
   TID : 2
   Arrive Time    : 0
   Execute Time   : 3  /  3
**************************************
**************************************
   TID : 1
   Arrive Time    : 2
   Execute Time   : 1  /  5
**************************************
**************************************
   TID : 1
   Arrive Time    : 2
   Execute Time   : 2  /  5
**************************************
**************************************
   TID : 1
   Arrive Time    : 2
   Execute Time   : 3  /  5
**************************************
**************************************
   TID : 1
   Arrive Time    : 2
   Execute Time   : 4  /  5
**************************************
**************************************
   TID : 1
   Arrive Time    : 2
   Execute Time   : 5  /  5
**************************************
********************************
*              IDLE              *
********************************
**************************************
   TID : 3
   Arrive Time    : 10
   Execute Time   : 1  /  3
**************************************
**************************************
   TID : 3
   Arrive Time    : 10
   Execute Time   : 2  /  3
**************************************
**************************************
   TID : 3
   Arrive Time    : 10
   Execute Time   : 3  /  3
**************************************
```

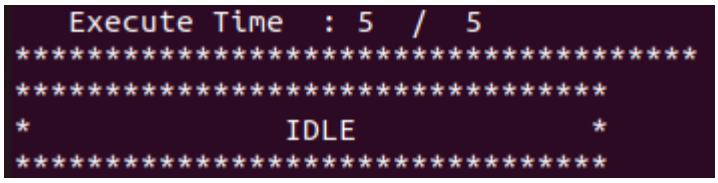



- RR 测试
  - 我们设置了 RR 测试样例如下：

```
void initRRCases(void) {

    int newTskTid0 = createTsk(myTskRR0);
    setTskPara(ARRV_TIME, 0, tcbPool[newTskTid0]->para);
    setTskPara(EXEC_TIME, 3, tcbPool[newTskTid0]->para);

    int newTskTid1 = createTsk(myTskRR1);
    setTskPara(ARRV_TIME, 1, tcbPool[newTskTid1]->para);
    setTskPara(EXEC_TIME, 3, tcbPool[newTskTid1]->para);

    int newTskTid2 = createTsk(myTskRR2);
    setTskPara(ARRV_TIME, 3, tcbPool[newTskTid2]->para);
    setTskPara(EXEC_TIME, 1, tcbPool[newTskTid2]->para);

    int newTskTid3 = createTsk(myTskRR3);
    setTskPara(ARRV_TIME, 4, tcbPool[newTskTid3]->para);
    setTskPara(EXEC_TIME, 5, tcbPool[newTskTid3]->para);


    enableTask(newTskTid1);
    enableTask(newTskTid0);
    enableTask(newTskTid3);
    enableTask(newTskTid2);
}
```

  - 选择 RR 测试结果如下，符合测试样例的应有执行情况：

```
**************************************
   TID : 1
   Arrive Time   : 0
   Execute Time  : 1  /  3
**************************************
**************************************
   TID : 1
   Arrive Time   : 0
   Execute Time  : 2  /  3
**************************************
**************************************
   TID : 2
   Arrive Time   : 1
   Execute Time  : 1  /  3
**************************************
**************************************
   TID : 2
   Arrive Time   : 1
   Execute Time  : 2  /  3
**************************************
**************************************
   TID : 1
   Arrive Time   : 0
   Execute Time  : 3  /  3
**************************************
**************************************
   TID : 3
   Arrive Time   : 3
   Execute Time  : 1  /  1
**************************************
**************************************
   TID : 4
   Arrive Time   : 4
   Execute Time  : 1  /  5
**************************************
**************************************
   TID : 4
   Arrive Time   : 4
   Execute Time  : 2  /  5
**************************************
**************************************
   TID : 2
   Arrive Time   : 1
   Execute Time  : 3  /  3
**************************************
**************************************
   TID : 4
   Arrive Time   : 4
   Execute Time  : 3  /  5
**************************************
**************************************
   TID : 4
   Arrive Time   : 4
   Execute Time  : 4  /  5
**************************************
**************************************
   TID : 4
   Arrive Time   : 4
```

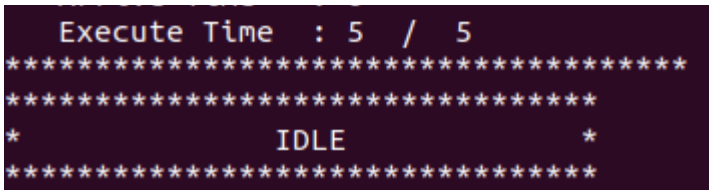



- SJF 测试：
  - 我们设置了 SJF 测试样例如下：

```
void initSJFCases(void) {
    int newTskTid0 = createTsk(myTskSJF0);
    setTskPara(ARRV_TIME, 0, tcbPool[newTskTid0]->para);
    setTskPara(EXEC_TIME, 1, tcbPool[newTskTid0]->para);

    int newTskTid1 = createTsk(myTskSJF1);
    setTskPara(ARRV_TIME, 0, tcbPool[newTskTid1]->para);
    setTskPara(EXEC_TIME, 5, tcbPool[newTskTid1]->para);

    int newTskTid2 = createTsk(myTskSJF2);
    setTskPara(ARRV_TIME, 1, tcbPool[newTskTid2]->para);
    setTskPara(EXEC_TIME, 4, tcbPool[newTskTid2]->para);

    int newTskTid3 = createTsk(myTskSJF3);
    setTskPara(ARRV_TIME, 3, tcbPool[newTskTid3]->para);
    setTskPara(EXEC_TIME, 2, tcbPool[newTskTid3]->para);


    enableTask(newTskTid1);
    enableTask(newTskTid2);
    enableTask(newTskTid3);
    enableTask(newTskTid0);
}
```

  - 选择 SJF 测试结果如下，符合测试样例的应有执行情况：

```
**************************************
   TID : 1
   Arrive Time   : 0
   Execute Time  : 1  /  1
**************************************
**************************************
   TID : 3
   Arrive Time   : 1
   Execute Time  : 1  /  4
**************************************
**************************************
   TID : 3
   Arrive Time   : 1
   Execute Time  : 2  /  4
**************************************
**************************************
   TID : 3
   Arrive Time   : 1
   Execute Time  : 3  /  4
**************************************
**************************************
   TID : 3
   Arrive Time   : 1
   Execute Time  : 4  /  4
**************************************
**************************************
   TID : 4
   Arrive Time   : 3
   Execute Time  : 1  /  2
**************************************
**************************************
   TID : 4
   Arrive Time   : 3
   Execute Time  : 2  /  2
**************************************
**************************************
   TID : 2
   Arrive Time   : 0
   Execute Time  : 1  /  5
**************************************
**************************************
   TID : 2
   Arrive Time   : 0
   Execute Time  : 2  /  5
**************************************
**************************************
   TID : 2
   Arrive Time   : 0
   Execute Time  : 3  /  5
**************************************
**************************************
   TID : 2
   Arrive Time   : 0
   Execute Time  : 4  /  5
**************************************
**************************************
   TID : 2
   Arrive Time   : 0
```




## 遇见的问题和解决方法

- 遇到的主要问题是 Hook 机制的使用，由于使用较少，不够熟练，在网络资料和人工智能的帮助下解决了相应问题。

## 本学期实验回顾和总结

- 本学期操作系统实验从简单的启动头开始，一步步构建了一个简单的操作系统，最后实现了几种进程的调度算法，涉及了许多课程讲解的知识，了解了操作系统基本原理，得到了不少收获。