**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：郭宇航 学号：2016100104014 指导教师：薛瑞尼**

**实验地点：主楼A2-412 实验时间：2019.5.25**

1. **实验室名称：计算机实验室**
2. **实验项目名称：**

**系统化思维模式下计算机操作系统进程与资源管理设计**

1. **实验学时：4**
2. **实验原理：**
3. **进程的三种基本状态：**
4. **就绪状态：当前进程已经分配得到除CPU外的所有必要的资源，只要再获得CPU即可运行，此时进程的状态被称为就绪状态。**
5. **执行状态：进程已经获得了CPU，其程序正在执行，在单处理机的系统中只有一个进程处于执行状态。**
6. **阻塞状态：处于执行状态的进程由于请求资源失败或其他事情而暂停无法继续执行下去时，放弃处理机而暂时处于暂停状态，此时进程的状态被称为阻塞状态或者等待状态或者封锁状态。**
7. **总体的设计：**

**图1 系统总体结构**

**系统总体架构如图1所示，最右边部分为进程与资源管理器，属于操作系统内核的功能。该管理器具有如下功能：完成进程创建、撤销和进程调度；完成多单元资源的管理；完成资源的申请和释放；完成错误检测和定时器中断功能。**

**图1中间绿色部分为驱动程序test shell, 设计与实现test shell，该test shell将调度所设计的进程与资源管理器来完成测试。Test shell的应具有的功能：**

**1)从终端或者测试文件读取命令；**

**2）将用户需求转换成调度内核函数（即调度进程和资源管理器）；**

**3)在终端或输出文件中显示结果：如当前运行的进程、错误信息等。**

**图1最左端部分为：通过终端（如键盘输入）或者测试文件来给出相应的用户命令，以及模拟硬件引起的中断**

**Test\_shell的功能如4.1所述，代码示例如图1中绿色部分。**

Test shell要求完成的命令（Mandatory Commands）

-init

-cr <name> <priority>(=1 or 2) // create process

-de <name> // delete process

-req <resource name> <# of units> // request resource

-rel <resource name> <# of units> // release resource

-to // time out

**3.进程和资源管理设计**

进程状态： ready/running/blocked

进程操作：

* 创建(create)： (none) -> ready
* 撤销(destroy)： running/ready/blocked -> (none)
* 请求资源(Request): running -> blocked (当资源没有时，进程阻塞)
* 释放资源(Release): blocked -> ready (因申请资源而阻塞的进程被唤醒)
* 时钟中断(Time\_out): running -> ready
* 调度：ready -> running / running ->ready

#### 进程控制块结构（PCB）

* **PID（name）**
* **resources //: resource which is occupied**
* **Status: Type & List// type: ready, block, running…., //List: RL(Ready list) or BL(block list)**
* **Creation\_tree: Parent/Children**
* **Priority: 0, 1, 2 (Init, User, System)**

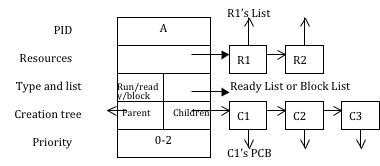


图2 PCB结构示意

就绪进程队列：Ready list (RL)



图3 Ready list 数据结构

3个级别的优先级，且优先级固定无变化

2 =“system”

1 = “user”

0 = “init”

每个PCB要么在RL中，要么在block list中 。当前正在运行的进程，根据优先级，可以将其放在RL中相应优先级队列的首部。

资源数据结构：

**资源的表示：**设置固定的资源数量，4类资源，R1，R2，R3，R4，每类资源Ri有i个

资源控制块Resource control block (RCB) 如图5所示

* + RID: 资源的ID
  + Status: 空闲单元的数量
  + Waiting\_List: list of blocked process



图5 资源数据结构RCB

**4.主要函数**

* **创建进程：**

Create(initialization parameters)// initialization parameters可以为进程的ID和优先级，优先级：初始进程0、用户进程1和系统进程2。

{

create PCB data structure

initialize PCB using parameters //包括进程的ID，优先级、状态等

link PCB to creation tree /\*连接父亲节点和兄弟节点，当前进程为 父亲节点，父亲节点中的子节点为兄弟节点\*/

insert(RL, PCB)//插入就绪相应优先级队列的尾部

Scheduler()

}

Init进程在启动时创建，可以用来创建第一个系统进程或者用户进程。新创建的进程或者被唤醒的进程被插入到就绪队列（RL）的末尾。

示例：

图4中，虚线表示进程A为运行进程，在进程A运行过程中，创建用户进程B：cr B 1，数据结构间关系图4所示

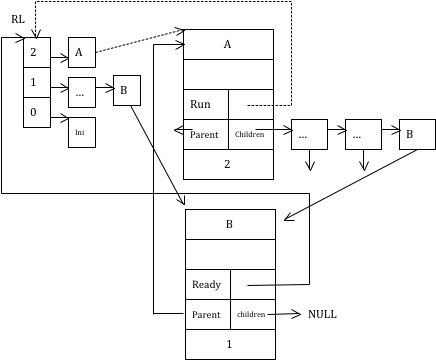


图4 进程数据结构间关系

(为了简单起见，A和B分别指向RL的链接可以不要)

* 撤销进程

Destroy (pid)

{

get pointer p to PCB using pid

Kill\_Tree(p)

Scheduler() //调度其他进程执行

}

Kill\_Tree(p)

{

for all child processes q Kill\_Tree(q) //嵌套调用，撤销所有子孙进程

free resources //和release调用类似的功能

delete PCB and update all pointers

}

Process can be destroyed by any of its ancestors or by itself (exit)

* **进程请求资源**

所有的资源申请请求按照FIFO的顺序进行

* **情况一**：当一类资源数量本身只有一个的情况

Request(rid)

{

r = Get\_RCB(rid);

if (r->Status == 'free') //只有一个资源时可以用free和allocated来表示资源状态

{

r->Status = 'allocated‘;

insert(self->Resources, r);//self 为当前请求资源的进程PCB，insert以后r为self进程占有的资源, 参PCB结构图

}

else

{

self->Status.Type = 'blocked';

self->Status.List = r;//point to block list, self is blocked by r

remove(RL, self);// remove self from the ready list(self can be put at the head of RL when it is running).

insert(r->Waiting\_List, self);// 将进程self插入到资源r的等待队列尾部

Scheduler();

}

｝

* 情况二：一类资源有多个的情况（multi\_unit）

Request(rid, n) // n为请求资源数量

{

r = Get\_RCB(rid);

if (u ≥ n) // u为r->Status.u, 即可用资源数量，参资源数据结构图

{

u-n;

insert(self->Resources, r, n);//self 为当前请求资源的进程PCB，insert以后n 个r为self进程占有的资源, 参PCB结构图

}

else

{

if (n>k) exit;//k为资源r的总数，申请量超过总数时，将打印错误信息并退出

self->Status.Type = 'blocked'; //只要u<n, 就不分配，进程阻塞

self->Status.List = r;//point to block list 注意：此时block list 是n个 r

remove(RL, self);// remove self from the ready list，因为运行进程位于绪队列首部，所以此时将它从就绪队列移除

insert(r->Waiting\_List, self);// 将进程self插入到资源r的等待队列尾部

Scheduler();

}

｝

* **进程释放资源**
* 情况一：一类资源只有1个的情况

Release(rid)

{

r = Get\_RCB(rid);

remove(self->Resources, r);//将r从进程self占用的资源中移走

if (r->Waiting\_List == NIL) //没有进程在等待资源r

{

r->Status = 'free';

}

else

{

remove(r->Waiting\_List, q);//q为waiting\_list中第一个阻塞进程

q->Status.Type = 'ready';

q->Status.List = RL;//就绪队列

insert(q->Resources, r);

insert(RL, q); //q插入就绪队列中相应优先级队列的末尾

Scheduler();

}

}

* 情况二：一类资源有多个的情况

Release(rid,n) //rid为资源ID，n为释放的资源数量

{

r = Get\_RCB(rid);

/\*remove r from self->resources, and u= u + n，将资源r从当前进程占用的资源列表里移除，并且资源r的可用数量从定义结构体：PCB和RCB。PCB：pid + type + resource + resource number + 优先级；RCB：四类资源：R1，R2，R3，R4每类资源的结构：u变为u+n\*/

remove(self->Resources, r, n);

/\*如果阻塞队列不为空, 且阻塞队列首部进程需求的资源数req小于等于可用资源数量u，则唤醒这个阻塞进程，放入就绪队列\*/

while (r->Waiting\_List != NIL && u>=req\_num)

{

u=u- req\_num; //可用资源数量减少

remove(r->Waiting\_List, q); // 从资源r的阻塞队列中移除

q->Status.Type = 'ready';

q->Status.List = RL;

insert(q->Resources, r); //插入r到q所占用的资源中

insert(RL, q); // 插入q到就绪队列

｝

Scheduler(); //基于优先级的抢占式调度策略，因此当有进程获得资源时，需要查看当前的优先级情况并进行调度

}

**5.进程调度与时钟中断的设计**

调度策略

* + 基于3个优先级别的调度：2，1，0
  + 使用基于优先级的抢占式调度策略，在同一优先级内使用时间片轮转（RR）
  + 基于函数调用来模拟时间共享
  + 初始进程(Init process)具有双重作用：

虚设的进程：具有最低的优先级，永远不会被阻塞

进程树的根

* Scheduler:

Called at the end of every kernel call

(1) Scheduler() {

(2) find highest priority process p

(3) if (self->priority < p->priority ||

(4) self->Status.Type != 'running' ||

(5) self == NIL)

(6) preempt(p, self)//在条件(3)(4)(5)下抢占当前进程

}

Condition (3): called from create or release, 即新创建进程的优先级或

资源释放后唤醒进程的优先级高于当前进程优先级

Condition (4): called from request or time-out， 即请求资源使得当前运行进程阻塞或者时钟中断使得当前运行进程变成就绪

Condition (5): called from destroy，进程销毁

Preemption: //抢占，将P变为执行，输出当前运行进程的名称

* Change status of p to running (status of self already changed to ready/blocked)
* Context switch—output name of running process
* 时钟中断（Time out）：模拟时间片到或者外部硬件中断

Time\_out()

{

find running process q; //当前运行进程q

remove(RL, q);// remove from head? yes

q->Status.Type = 'ready';

insert(RL, q);// insert into tail? yes

Scheduler();

}

**6.系统初始化设计**

启动时初始化管理器：

具有3个优先级的就绪队列RL初始化；

Init进程；

4类资源，R1，R2，R3，R4，每类资源Ri有i个

1. **实验目的：**

**设计和实现进程与资源管理，并完成Test shell的编写，以建立系统的进程管理、调度、资源管理和分配的知识体系，从而加深对操作系统进程调度和资源管理功能的宏观理解和微观实现技术的掌握。**

1. **实验内容：**

在实验室提供的软硬件环境中，设计并实现一个基本的进程与资源管理器。该管理器能够完成进程的控制，如进程创建与撤销、进程的状态转换；能够基于优先级调度算法完成进程的调度，模拟时钟中断，在同优先级进程中采用时间片轮转调度算法进行调度；能够完成资源的分配与释放，并完成进程之间的同步。该管理器同时也能完成从用户终端或者指定文件读取用户命令，通过Test shell模块完成对用户命令的解释，将用户命令转化为对进程与资源控制的具体操作，并将执行结果输出到终端或指定文件中。

1. **实验器材（设备、元器件）：**

**硬件条件：华硕飞行堡垒fx50j i5-4200HQ 4G**

**软件平台：Windows10专业版python3.6.0 Pycharm 2017.2.3**

1. **实验步骤：**

**首先定义进程控制块PCB与资源控制块RCB以及整个系统的进程处理模块：process\_handle\_sector**

1. **class** RCB:
2. **def** \_\_init\_\_(self):
3. self.Rid = None
4. self.Initial = 0
5. self.Remain = 0
6. self.Waiting\_list = []
8. **class** process\_handle\_sector:
9. **def** \_\_init\_\_(self):
10. #定义资源1的阻塞队列
11. self.R1 = RCB()
12. self.R1.Rid = 'R1'
13. self.R1.Initial = 1
14. self.R1.Remain = 1
16. self.R2 = RCB()
17. self.R2.Rid = 'R2'
18. self.R2.Initial = 2
19. self.R2.Remain = 2
21. self.R3 = RCB()
22. self.R3.Rid = 'R1'
23. self.R3.Initial = 3
24. self.R3.Remain = 3
26. self.R4 = RCB()
27. self.R4.Rid = 'R1'
28. self.R4.Initial = 4
29. self.R4.Remain = 4
30. #初始化三个不同优先级的就绪队列
31. self.RL = [[],[],[]]
32. self.current\_process = None
33. self.current\_process\_pid = -1
34. self.resource\_list = [self.R1.Waiting\_list,
35. self.R2.Waiting\_list,
36. self.R3.Waiting\_list,
37. self.R4.Waiting\_list]
38. self.destroyed\_process = []
40. **class** PCB:
41. **def** \_\_init\_\_(self):
42. self.Pid = None
43. self.type = 'Ready' #对于一个进程来说：ready/running/blocked
44. self.resources = []
45. self.resources\_num = []
46. self.priority = None
47. self.request\_resources = None
48. self.request\_resources\_num = -1
49. self.parent = None
50. self.children = None

**下面给出一些主要函数的代码部分：**

**首先是创建进程函数：**

1. **def** create\_process(command,process\_sector):
2. #create PCB data structure
3. process = PCB()
4. #initialize PCB using parameters 包括进程的ID，优先级、状态等
5. process.Pid = command[1]
6. process.priority = int(command[2])
7. #寻找当前执行的进程，并将原本在就绪队列中的进程pop出来然后改为正在运行的程序
8. **if** process.priority == 2:
9. process\_sector.RL[0].append(process)
10. #优先级为2的进程，直接放入就绪队列1，并且判断是否是就绪队列1中的唯一元素，如果是直接执行？是否有问题？
11. **if** process\_sector.RL[0][0] == process:
12. process\_sector.current\_process = process
13. process\_sector.current\_process\_pid = process.Pid
14. process\_sector.RL[0].pop(0)
15. process.type = 'running'
16. **elif** process.priority == 1:
17. process\_sector.RL[1].append(process)
18. #如果system级别的进程就绪队列中没有进程则考虑第二优先级的进程就绪队列
19. **if** len(process\_sector.RL[0]) == 0 **and** process\_sector.RL[1][0] == process:
20. **if** process\_sector.current\_process != None:
21. **pass**
22. **else**:
23. process\_sector.current\_process = process
24. process\_sector.current\_process\_pid = process.Pid
25. process\_sector.RL[1].pop(0)
26. process.type = 'running'
27. **else**:
28. process\_sector.RL[2].append(process)

**销毁进程函数：**

1. **def** block2ready\_function(waiting\_list,RCB\_use,ready\_list):
2. count = 0
3. **if** len(waiting\_list) != 0:
4. #计算在阻塞队列中有多少进程在释放资源后可以进入就绪队列
5. **for** i **in** waiting\_list:
6. **if** i.request\_resources\_num <= RCB\_use.Remain:
7. count +=1
8. RCB\_use.Remain -= i.request\_resources\_num
9. #print(count)
10. #count计算了所有可以进入就绪队列的阻塞进程数量，遍历这些进程，拿走资源且进入就绪状态
11. **if** count <= 0:
12. #表示释放的资源连阻塞队列的第一个进程都无法满足
13. **pass**
14. **else**:
15. #这种情况下释放的资源可以满足一些阻塞的进程使用
16. **for** j **in** range(count):
17. waiting\_list[j].type = 'ready'
18. waiting\_list[j].resources.append(RCB\_use.Rid)
19. waiting\_list[j].resources\_num.append(waiting\_list[j].request\_resources\_num)
20. waiting\_list[j].request\_resources = None
21. waiting\_list[j].request\_resources\_num = -1
22. **if** waiting\_list[j].priority == 2:
23. ready\_list[0].append(waiting\_list[j])
24. **elif** waiting\_list[j].priority == 1:
25. ready\_list[1].append(waiting\_list[j])
26. **else**:
27. ready\_list[2].append(waiting\_list[j])
28. **else**:
29. #此时阻塞队列为空，不进行任何操作
30. **pass**
31. #销毁一个进程
32. **def** destroy\_process(command,process\_sector):
33. destroy\_pid = command[1]
34. #print('the process are going to be destroyed!',destroy\_pid)
35. '''''
36. 这种写法有一点点小bug，主要存在于如果某个进程非连续申请同一个资源的时候，存在资源和数量列表出现不一致的情况，需要进一步改进
37. '''
38. #首先看销毁的进程是否是当前进程
39. **if** process\_sector.current\_process.Pid == destroy\_pid:
40. #如果是当前进程需要被销毁，则直接由running变为None且放入销毁队列中
41. process\_sector.current\_process.type = None
42. process\_sector.destroyed\_process.append(process\_sector.current\_process)
43. **if** process\_sector.current\_process.resources == []:
44. **pass**
45. **if** 'R1' **in** process\_sector.current\_process.resources:
46. #获取r1资源在该进程资源队列中的位置
47. r1\_index = process\_sector.current\_process.resources.index('R1')
48. process\_sector.R1.Remain += process\_sector.current\_process.resources\_num[r1\_index]
49. process\_sector.current\_process.resources\_num[r1\_index] = -1
50. #当销毁了进程后将资源恢复同时考虑是否存在资源等待队列中的进程可以进入就绪队列
51. block2ready\_function(process\_sector.R1.Waiting\_list,process\_sector.R1,process\_sector.RL)
52. **if** 'R2' **in** process\_sector.current\_process.resources:
53. r2\_index = process\_sector.current\_process.resources.index('R2')
54. process\_sector.R2.Remain += process\_sector.current\_process.resources\_num[r2\_index]
55. process\_sector.current\_process.resources\_num[r2\_index] = -1
56. block2ready\_function(process\_sector.R2.Waiting\_list,process\_sector.R2,process\_sector.RL)
57. **if** 'R3' **in** process\_sector.current\_process.resources:
58. r3\_index = process\_sector.current\_process.resources.index('R3')
59. process\_sector.R3.Remain += process\_sector.current\_process.resources\_num[r3\_index]
60. process\_sector.current\_process.resources\_num[r3\_index] = -1
61. block2ready\_function(process\_sector.R3.Waiting\_list,process\_sector.R3,process\_sector.RL)
62. **if** 'R4' **in** process\_sector.current\_process.resources:
63. r4\_index = process\_sector.current\_process.resources.index('R4')
64. process\_sector.R4.Remain += process\_sector.current\_process.resources\_num[r4\_index]
65. process\_sector.current\_process.resources\_num[r4\_index] = -1
66. # 当销毁了进程后将资源恢复同时考虑是否存在资源等待队列中的进程可以进入就绪队列
67. block2ready\_function(process\_sector.R4.Waiting\_list,process\_sector.R4,process\_sector.RL)
68. **else**:
69. **pass**
70. # 在释放完所有的资源后改变当前的进程
71. **if** len(process\_sector.RL[0]) != 0:
72. process\_sector.current\_process = process\_sector.RL[0][0]
73. process\_sector.current\_process\_pid = process\_sector.RL[0][0].Pid
74. process\_sector.RL[0].pop(0)
75. **elif** len(process\_sector.RL[0]) == 0 **and** len(process\_sector.RL[1]) != 0:
76. process\_sector.current\_process = process\_sector.RL[1][0]
77. process\_sector.current\_process\_pid = process\_sector.RL[1][0].Pid
78. process\_sector.RL[1].pop(0)
79. #下面考虑需要销毁的进程在就绪队列中
80. **if** len(process\_sector.RL[0])!=0:
81. #遍历就绪队列1查看要销毁的进程是否在其中
82. **for** i **in** process\_sector.RL[0]:
83. **if** i.Pid == destroy\_pid:
84. i.type = None
85. process\_sector.destroyed\_process.append(i)
86. **if** i.resources == []:
87. **pass**
88. **else**:
89. **if** 'R1' **in** i.resources:
90. r1\_index\_1 = i.resources.index('R1')
91. process\_sector.R1.Remain += i.resources\_num[r1\_index\_1]
92. i.resources\_num[r1\_index\_1] = -1
93. block2ready\_function(process\_sector.R1.Waiting\_list,process\_sector.R1,process\_sector.RL)
94. **if** 'R2' **in** i.resources:
95. r2\_index\_1 = i.resources.index('R2')
96. process\_sector.R2.Remain += i.resources\_num[r2\_index\_1]
97. i.resources\_num[r2\_index\_1] = -1
98. block2ready\_function(process\_sector.R2.Waiting\_list,process\_sector.R2,process\_sector.RL)
99. **if** 'R3' **in** i.resources:
100. r3\_index\_1 = i.resources.index('R3')
101. process\_sector.R3.Remain += i.resources\_num[r3\_index\_1]
102. i.resources\_num[r3\_index\_1] = -1
103. block2ready\_function(process\_sector.R3.Waiting\_list,process\_sector.R3,process\_sector.RL)
104. **if** 'R4' **in** i.resources:
105. r4\_index\_1 = i.resources.index('R4')
106. process\_sector.R4.Remain += i.resources\_num[r4\_index\_1]
107. i.resources\_num[r4\_index\_1] = -1
108. block2ready\_function(process\_sector.R4.Waiting\_list,process\_sector.R4,process\_sector.RL)
109. process\_sector.RL[0].remove(i)
110. #遍历就绪队列2检查需要销毁的进程是否在其中
111. **if** len(process\_sector.RL[1])!=0:
112. **for** i **in** process\_sector.RL[1]:
113. **if** i.Pid == destroy\_pid:
114. i.type = None
115. process\_sector.destroyed\_process.append(i)
116. **if** i.resources == []:
117. **pass**
118. **else**:
119. **if** 'R1' **in** i.resources:
120. r1\_index\_2 = i.resources.index('R1')
121. process\_sector.R1.Remain += i.resources\_num[r1\_index\_2]
122. i.resources\_num[r1\_index\_2] = -1
123. block2ready\_function(process\_sector.R1.Waiting\_list,process\_sector.R1,process\_sector.RL)
124. **if** 'R2' **in** i.resources:
125. r2\_index\_2 = i.resources.index('R2')
126. process\_sector.R2.Remain += i.resources\_num[r2\_index\_2]
127. i.resources\_num[r2\_index\_2] = -1
128. block2ready\_function(process\_sector.R2.Waiting\_list,process\_sector.R2,process\_sector.RL)
129. **if** 'R3' **in** i.resources:
130. r3\_index\_2 = i.resources.index('R3')
131. process\_sector.R3.Remain += i.resources\_num[r3\_index\_2]
132. i.resources\_num[r3\_index\_2] = -1
133. block2ready\_function(process\_sector.R3.Waiting\_list,process\_sector.R3,process\_sector.RL)
134. **if** 'R4' **in** i.resources:
135. r4\_index\_2 = i.resources.index('R4')
136. process\_sector.R4.Remain += i.resources\_num[r4\_index\_2]
137. i.resources\_num[r4\_index\_2] = -1
138. block2ready\_function(process\_sector.R4.Waiting\_list,process\_sector.R4,process\_sector.RL)
139. process\_sector.RL[1].remove(i)
140. #查看需要销毁的进程是否在阻塞队列R1中
141. **if** len(process\_sector.R1.Waiting\_list) != 0:
142. **for** i **in** process\_sector.R1.Waiting\_list:
143. #如果需要销毁的进程在R1的阻塞队列中，将此进程从阻塞进程中取出然后加入destroy队列中
144. **if** i.Pid == destroy\_pid:
145. i.type = None
146. process\_sector.destroyed\_process.append(i)
147. process\_sector.R1.Waiting\_list.remove(i)
148. #这里需要判断无论在哪个阻塞队列中，阻塞的进程可能也占用了一些其他的资源
149. **if** i.resources == []:
150. **pass**
151. **else**:
152. **if** 'R1' **in** i.resources:
153. r1\_index\_3 = i.resources.index('R1')
154. process\_sector.R1.Remain += i.resources\_num[r1\_index\_3]
155. i.resources\_num[r1\_index\_3] = -1
156. block2ready\_function(process\_sector.R1.Waiting\_list,process\_sector.R1,process\_sector.RL)
157. **if** 'R2' **in** i.resources:
158. r2\_index\_3 = i.resources.index('R2')
159. process\_sector.R2.Remain += i.resources\_num[r2\_index\_3]
160. i.resources\_num[r2\_index\_3] = -1
161. block2ready\_function(process\_sector.R2.Waiting\_list,process\_sector.R2,process\_sector.RL)
162. **if** 'R3' **in** i.resources:
163. r3\_index\_3 = i.resources.index('R3')
164. process\_sector.R3.Remain += i.resources\_num[r3\_index\_3]
165. i.resources\_num[r3\_index\_3] = -1
166. block2ready\_function(process\_sector.R3.Waiting\_list,process\_sector.R3,process\_sector.RL)
167. **if** 'R4' **in** i.resources:
168. r4\_index\_3 = i.resources.index('R4')
169. process\_sector.R4.Remain += i.resources\_num[r4\_index\_3]
170. i.resources\_num[r4\_index\_3] = -1
171. block2ready\_function(process\_sector.R4.Waiting\_list,process\_sector.R4,process\_sector.RL)
173. **if** len(process\_sector.R2.Waiting\_list) != 0:
174. **for** i **in** process\_sector.R2.Waiting\_list:
175. #如果需要销毁的进程在R1的阻塞队列中，将此进程从阻塞进程中取出然后加入destroy队列中
176. **if** i.Pid == destroy\_pid:
177. i.type = None
178. process\_sector.destroyed\_process.append(i)
179. process\_sector.R2.Waiting\_list.remove(i)
180. #这里需要判断无论在哪个阻塞队列中，阻塞的进程可能也占用了一些其他的资源
181. **if** i.resources == []:
182. **pass**
183. **else**:
184. **if** 'R1' **in** i.resources:
185. r1\_index\_4 = i.resources.index('R1')
186. process\_sector.R1.Remain += i.resources\_num[r1\_index\_4]
187. i.resources\_num[r1\_index\_4] = -1
188. block2ready\_function(process\_sector.R1.Waiting\_list,process\_sector.R1,process\_sector.RL)
189. **if** 'R2' **in** i.resources:
190. r2\_index\_4 = i.resources.index('R2')
191. process\_sector.R2.Remain += i.resources\_num[r2\_index\_4]
192. i.resources\_num[r2\_index\_4] = -1
193. block2ready\_function(process\_sector.R2.Waiting\_list,process\_sector.R2,process\_sector.RL)
194. **if** 'R3' **in** i.resources:
195. r3\_index\_4 = i.resources.index('R3')
196. process\_sector.R3.Remain += i.resources\_num[r3\_index\_4]
197. i.resources\_num[r3\_index\_4] = -1
198. block2ready\_function(process\_sector.R3.Waiting\_list,process\_sector.R3,process\_sector.RL)
199. **if** 'R4' **in** i.resources:
200. r4\_index\_4 = i.resources.index('R4')
201. process\_sector.R4.Remain += i.resources\_num[r4\_index\_4]
202. i.resources\_num[r4\_index\_4] = -1
203. block2ready\_function(process\_sector.R4.Waiting\_list,process\_sector.R4,process\_sector.RL)
205. #判断需要销毁的进程是否在R3的阻塞队列中
206. **if** len(process\_sector.R3.Waiting\_list) != 0:
207. **for** i **in** process\_sector.R3.Waiting\_list:
208. # 如果需要销毁的进程在R1的阻塞队列中，将此进程从阻塞进程中取出然后加入destroy队列中
209. **if** i.Pid == destroy\_pid:
210. i.type = None
211. process\_sector.destroyed\_process.append(i)
212. process\_sector.R3.Waiting\_list.remove(i)
213. # 这里需要判断无论在哪个阻塞队列中，阻塞的进程可能也占用了一些其他的资源
214. **if** i.resources == []:
215. **pass**
216. **else**:
217. **if** 'R1' **in** i.resources:
218. r1\_index\_5 = i.resources.index('R1')
219. process\_sector.R1.Remain += i.resources\_num[r1\_index\_5]
220. i.resources\_num[r1\_index\_5] = -1
221. block2ready\_function(process\_sector.R1.Waiting\_list,process\_sector.R1,process\_sector.RL)
222. **if** 'R2' **in** i.resources:
223. r2\_index\_5 = i.resources.index('R2')
224. process\_sector.R2.Remain += i.resources\_num[r2\_index\_5]
225. i.resources\_num[r2\_index\_5] = -1
226. block2ready\_function(process\_sector.R2.Waiting\_list,process\_sector.R2,process\_sector.RL)
227. **if** 'R3' **in** i.resources:
228. r3\_index\_5 = i.resources.index('R3')
229. process\_sector.R3.Remain += i.resources\_num[r3\_index\_5]
230. i.resources\_num[r3\_index\_5] = -1
231. block2ready\_function(process\_sector.R3.Waiting\_list,process\_sector.R3,process\_sector.RL)
232. **if** 'R4' **in** i.resources:
233. r4\_index\_5 = i.resources.index('R4')
234. process\_sector.R4.Remain += i.resources\_num[r4\_index\_5]
235. i.resources\_num[r4\_index\_5] = -1
236. block2ready\_function(process\_sector.R4.Waiting\_list,process\_sector.R4,process\_sector.RL)
237. #判断是否在R4的阻塞队列中
238. **if** len(process\_sector.R4.Waiting\_list) != 0:
239. **for** i **in** process\_sector.R4.Waiting\_list:
240. # 如果需要销毁的进程在R1的阻塞队列中，将此进程从阻塞进程中取出然后加入destroy队列中
241. #print('enter this process!')
242. **if** i.Pid == destroy\_pid:
243. i.type = None
244. # print('test result',i.resources)
245. process\_sector.destroyed\_process.append(i)
246. process\_sector.R4.Waiting\_list.remove(i)
247. # 这里需要判断无论在哪个阻塞队列中，阻塞的进程可能也占用了一些其他的资源
248. **if** i.resources == []:
249. **pass**
250. **else**:
251. **if** 'R1' **in** i.resources:
252. r1\_index\_6 = i.resources.index('R1')
253. process\_sector.R1.Remain += i.resources\_num[r1\_index\_6]
254. i.resources\_num[r1\_index\_6] = -1
255. block2ready\_function(process\_sector.R1.Waiting\_list,process\_sector.R1,process\_sector.RL)
256. **if** 'R2' **in** i.resources:
257. r2\_index\_6 = i.resources.index('R2')
258. process\_sector.R2.Remain += i.resources\_num[r2\_index\_6]
259. i.resources\_num[r2\_index\_6] = -1
260. block2ready\_function(process\_sector.R2.Waiting\_list,process\_sector.R2,process\_sector.RL)
261. **if** 'R3' **in** i.resources:
262. r3\_index\_6 = i.resources.index('R3')
263. process\_sector.R3.Remain += i.resources\_num[r3\_index\_6]
264. i.resources\_num[r3\_index\_6] = -1
265. block2ready\_function(process\_sector.R3.Waiting\_list,process\_sector.R3,process\_sector.RL)
266. **if** 'R4' **in** i.resources:
267. r4\_index\_6 = i.resources.index('R4')
268. process\_sector.R4.Remain += i.resources\_num[r4\_index\_6]
269. i.resources\_num[r4\_index\_6] = -1
270. block2ready\_function(process\_sector.R4.Waiting\_list,process\_sector.R4,process\_sector.RL)
271. **else**:
272. **print**('ERROR!')

**时间片周期轮转函数：**

1. #时间周期转完更换当前进程
2. **def** change\_current\_process(process\_sector):
3. **if** len(process\_sector.RL[0]) != 0:
4. **if** process\_sector.current\_process.priority == 2:
5. process\_sector.current\_process.type = 'ready'
6. process\_sector.RL[0].append(process\_sector.current\_process)
7. process\_sector.current\_process = process\_sector.RL[0][0]
8. process\_sector.current\_process\_pid = process\_sector.RL[0][0].Pid
9. process\_sector.RL[0].pop(0)
10. **elif** process\_sector.current\_process.priority == 1:
11. process\_sector.current\_process.type = 'ready'
12. process\_sector.RL[1].append(process\_sector.current\_process)
13. process\_sector.current\_process = process\_sector.RL[0][0]
14. process\_sector.current\_process\_pid = process\_sector.RL[0][0].Pid
15. process\_sector.RL[1].pop(0)
16. **elif** len(process\_sector.RL[0]) == 0 **and** len(process\_sector.RL[1])!=0:
17. #这里表示就绪队列1为空，没有优先级为2的进程在等待，直接考虑就绪队列2
18. **if** process\_sector.current\_process.priority == 1:
19. process\_sector.current\_process.type = 'ready'
20. process\_sector.RL[1].append(process\_sector.current\_process)
21. process\_sector.current\_process = process\_sector.RL[1][0]
22. process\_sector.current\_process\_pid = process\_sector.RL[1][0].Pid
23. process\_sector.RL[1].pop(0)
24. **else**:
25. **pass**
26. **else**:
27. #这里表示两个就绪队列都为空，则当前没有其他的进程，这里不考虑第三优先级的进程
28. **pass**

**请求资源函数：**

1. #请求资源操作
2. **def** request\_resources(command,process\_sector):
3. request\_r = command[1]
4. request\_num = int(command[2])
5. process\_sector.current\_process.request\_resources = request\_r
6. process\_sector.current\_process.request\_resources\_num = request\_num
7. **if** process\_sector.current\_process.request\_resources == 'R1':
8. **if** process\_sector.current\_process.request\_resources\_num <= process\_sector.R1.Remain:
9. #此时可以给当前进程分配资源
10. process\_sector.R1.Remain -= process\_sector.current\_process.request\_resources\_num
11. process\_sector.current\_process.resources.append('R1')
12. process\_sector.current\_process.resources\_num.append(process\_sector.current\_process.request\_resources\_num)
13. process\_sector.current\_process.request\_resources = None
14. process\_sector.current\_process.request\_resources\_num = -1
15. **else**:
16. #此时不能分配资源
17. process\_sector.current\_process.request\_resources = request\_r
18. process\_sector.current\_process.request\_resources\_num = request\_num
19. process\_sector.current\_process.type = 'blocked'
20. process\_sector.resource\_list[0].append(process\_sector.current\_process)
21. **if** len(process\_sector.RL[0])!=0:
22. process\_sector.current\_process = process\_sector.RL[0][0]
23. process\_sector.current\_process\_pid = process\_sector.RL[0][0].Pid
24. process\_sector.RL[1].pop(0)
25. **elif** len(process\_sector.RL[0])==0 **and** len(process\_sector.RL[1])!=0:
26. process\_sector.current\_process = process\_sector.RL[1][0]
27. process\_sector.current\_process\_pid = process\_sector.RL[1][0].Pid
28. process\_sector.RL[1].pop(0)
29. **else**:
30. **pass**
31. **if** process\_sector.current\_process.request\_resources == 'R2':
32. **if** process\_sector.current\_process.request\_resources\_num <= process\_sector.R2.Remain:
33. #此时可以给当前进程分配资源
34. process\_sector.R2.Remain -= process\_sector.current\_process.request\_resources\_num
35. process\_sector.current\_process.resources.append('R2')
36. process\_sector.current\_process.resources\_num.append(process\_sector.current\_process.request\_resources\_num)
37. process\_sector.current\_process.request\_resources = None
38. process\_sector.current\_process.request\_resources\_num = -1
39. **else**:
40. #此时不能分配资源
41. process\_sector.current\_process.request\_resources = request\_r
42. process\_sector.current\_process.request\_resources\_num = request\_num
43. process\_sector.current\_process.type = 'blocked'
44. process\_sector.resource\_list[1].append(process\_sector.current\_process)
45. **if** len(process\_sector.RL[0])!=0:
46. process\_sector.current\_process = process\_sector.RL[0][0]
47. process\_sector.current\_process\_pid = process\_sector.RL[0][0].Pid
48. process\_sector.RL[1].pop(0)
49. **elif** len(process\_sector.RL[0])==0 **and** len(process\_sector.RL[1])!=0:
50. process\_sector.current\_process = process\_sector.RL[1][0]
51. process\_sector.current\_process\_pid = process\_sector.RL[1][0].Pid
52. process\_sector.RL[1].pop(0)
53. **else**:
54. **pass**
55. **if** process\_sector.current\_process.request\_resources == 'R3':
56. **if** process\_sector.current\_process.request\_resources\_num <= process\_sector.R3.Remain:
57. #此时可以给当前进程分配资源
58. process\_sector.R3.Remain -= process\_sector.current\_process.request\_resources\_num
59. process\_sector.current\_process.resources.append('R3')
60. process\_sector.current\_process.resources\_num.append(process\_sector.current\_process.request\_resources\_num)
61. process\_sector.current\_process.request\_resources = None
62. process\_sector.current\_process.request\_resources\_num = -1
63. **else**:
64. #此时不能分配资源
65. process\_sector.current\_process.request\_resources = request\_r
66. process\_sector.current\_process.request\_resources\_num = request\_num
67. process\_sector.current\_process.type = 'blocked'
68. process\_sector.resource\_list[2].append(process\_sector.current\_process)
69. **if** len(process\_sector.RL[0])!=0:
70. process\_sector.current\_process = process\_sector.RL[0][0]
71. process\_sector.current\_process\_pid = process\_sector.RL[0][0].Pid
72. process\_sector.RL[1].pop(0)
73. **elif** len(process\_sector.RL[0])==0 **and** len(process\_sector.RL[1])!=0:
74. process\_sector.current\_process = process\_sector.RL[1][0]
75. process\_sector.current\_process\_pid = process\_sector.RL[1][0].Pid
76. process\_sector.RL[1].pop(0)
77. **else**:
78. **pass**
79. **if** process\_sector.current\_process.request\_resources == 'R4':
80. **if** process\_sector.current\_process.request\_resources\_num <= process\_sector.R4.Remain:
81. #此时可以给当前进程分配资源
82. process\_sector.R4.Remain -= process\_sector.current\_process.request\_resources\_num
83. process\_sector.current\_process.resources.append('R4')
84. process\_sector.current\_process.resources\_num.append(process\_sector.current\_process.request\_resources\_num)
85. process\_sector.current\_process.request\_resources = None
86. process\_sector.current\_process.request\_resources\_num = -1
87. **else**:
88. #此时不能分配资源
89. process\_sector.current\_process.request\_resources = request\_r
90. process\_sector.current\_process.request\_resources\_num = request\_num
91. process\_sector.current\_process.type = 'blocked'
92. process\_sector.resource\_list[3].append(process\_sector.current\_process)
93. **if** len(process\_sector.RL[0])!=0:
94. process\_sector.current\_process = process\_sector.RL[0][0]
95. process\_sector.current\_process\_pid = process\_sector.RL[0][0].Pid
96. process\_sector.RL[1].pop(0)
97. **elif** len(process\_sector.RL[0])==0 **and** len(process\_sector.RL[1])!=0:
98. process\_sector.current\_process = process\_sector.RL[1][0]
99. process\_sector.current\_process\_pid = process\_sector.RL[1][0].Pid
100. process\_sector.RL[1].pop(0)
101. **else**:
102. **pass**

**读取txt文件的代码：**

1. **def** load\_test\_shell(filepath):
2. shell\_txt = []
3. with open(filepath,'r') as file\_load:
4. **while** True:
5. lines = file\_load.readline()
6. **if** **not** lines:
7. **break**
8. **pass**
9. shell\_txt.append(lines.split())
10. **return** shell\_txt

**汇总上述的所有函数的代码以及主函数：**

1. **def** main\_function(shell\_text,process\_sector):
2. init\_process = init\_function('init')
3. result = []
4. result.append(init\_process)
5. **for** i **in** shell\_text:
6. **if** i[0] == 'cr':
7. create\_process(i,process\_sector)
8. result.append(scheduler(process\_sector))
9. **elif** i[0] == 'to':
10. change\_current\_process(process\_sector)
11. result.append(scheduler(process\_sector))
12. **elif** i[0] == 'req':
13. request\_resources(i,process\_sector)
14. result.append(scheduler(process\_sector))
15. **elif** i[0] == 'de':
16. destroy\_process(i,process\_sector)
17. result.append(scheduler(process\_sector))
18. **else**:
19. **print**('Error!检查输入文件！')
20. **return** result

23. **if** \_\_name\_\_ == '\_\_main\_\_':
24. shell\_txt = load\_test\_shell('1.txt')
25. #print(shell\_txt)
26. process\_sector = process\_handle\_sector()
27. process\_result = main\_function(shell\_txt,process\_sector)
28. **for** i **in** process\_result:
29. **print**(i,end=' ')
30. '''''
31. print('\*\*\*\*\*\*\*\*\*\*this is for debug\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
32. print('the length of ready\_List1',len(process\_sector.RL[1]))
33. print('the length of R3 Waiting list',len(process\_sector.R3.Waiting\_list))
34. print('R3 remains',process\_sector.R3.Remain)
35. print('q process resources',process\_sector.R4.Waiting\_list[0].Pid)
36. '''

**测试样例如下：**

输入测试命令或将测试命令放在测试文件input.txt中，内容为：

cr x 1

cr p 1

cr q 1

cr r 1

to

req R2 1

to

req R3 3

to

req R4 3

to

to

req R3 1

req R4 2

req R2 2

to

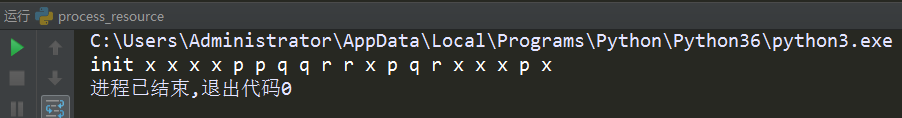
de q

to

to

输出结果应为：init x x x x p p q q r r x p q r x x x p x

**测试结果如下：**



**输出结果与实际结果一致，实验成功。**

1. **实验结论：**
2. **总结及心得体会：**
3. **对本实验过程及方法、手段的改进建议：**

**报告评分：**

**指导教师签字：**