一、实验项目名称：进程与资源管理器设计

二、实验原理：

进程管理与调度、资源管理与分配

三、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。

四、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，完成对时钟中断的处理，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。

五、实验器材（设备、元器件）：设计平台：Windows，使用语言：python

六、实验步骤：

(一)系统功能需求分析

系统由三部分组成：驱动程序 test shell，测试文件或终端输入及进程与资源管理器。每个部分需要实现的功能如下：

1. 驱动程序 test shell
    a) 从终端或者测试文件读取命令
    b) 将用户需求转换成调度模拟的内核函数（即调度进程和资源管理器）
    c) 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
2. 测试文件或终端输入
    a) 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

3. 进程与资源管理器

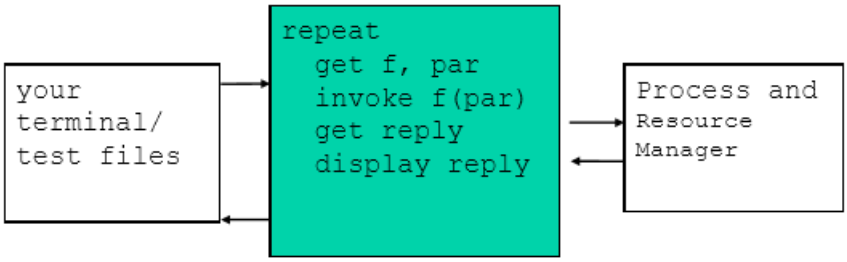
   a) 完成进程创建、撤销和进程调度

   b) 完成多单元 (multi_unit)资源的管理

   c) 完成资源的申请和释放

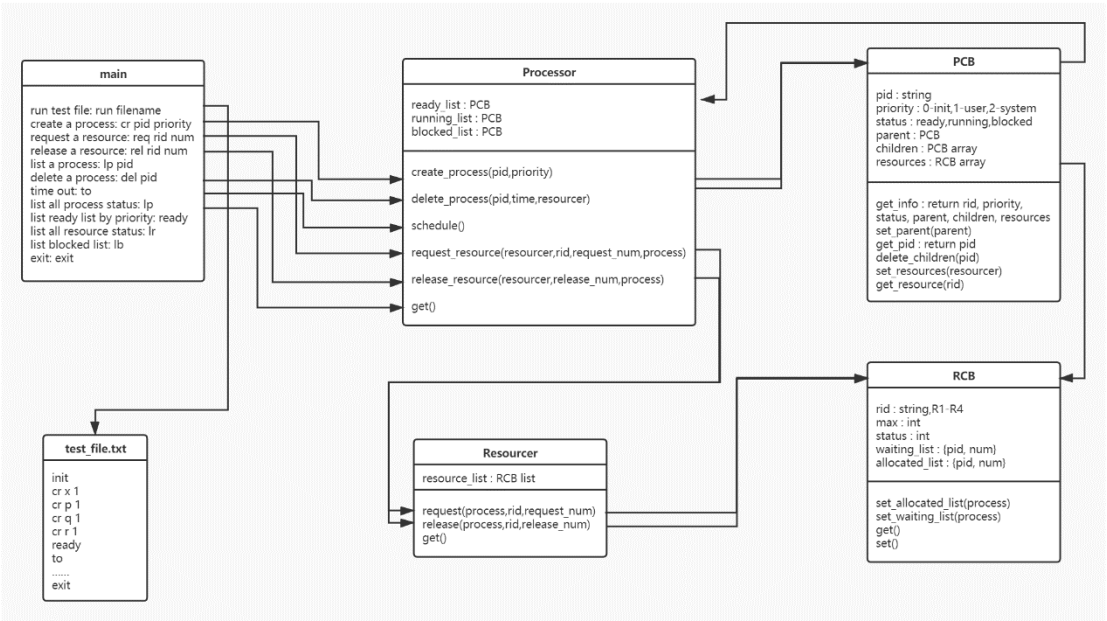
   d) 完成错误检测和定时器中断功能

系统总架构如下图所示：



(二)框架设计

系统首先从测试文件中读取命令，在 test shell 中解析成对应的命令，并调用并执行在进程资源管理器中的对应的函数。系统总体框架如下图所示：



接下来对执行各种命令的模块调用过程进行分析。

1. cr：该函数的输入参数为：Pid,Priority，用于创建一个 id 为 pid，优先级为 priorty 的新进程。当 test shell 接收到格式为 cr pid priority 的命令时，调用进程管理器 processor 的 create_process 函数实现进程创建。
2. init：该命令用于创建初始进程，默认 pid 为 init，priority 为 0，作为系统的初始进程，该进程的父进程设为空，init 进程不可删除且必须第一个创

建。

3. de：该函数的输入参数为：pid，用于删除 id 为 pid 的进程。当 test shell 接收到格式为 de pid 的命令时，首先判断 pid 是否为 init，若是 init 则打印“cannot delete init process”的报错信息，因为 init 进程不可被删除。若 pid 并非 init，则调用进程管理器的 delete_process 函数实现进程的删除。
4. req：该函数的输入参数为 rid，num，用于为当前正在运行的进程请求 num 个 id 为 rid 的资源。当 test shell 接收到格式为 req rid num 的命令时，首先判断 num 是否为正整数，若不满足该条件，则打印报错信息“request number should be a positive in teger”。否则调用进程管理器的 request_resource 为运行中的进程申请资源。
5. rel：该函数的输入参数为 rid，num，用于为当前运行的进程释放 num 个 id 为 rid 的资源。当 test shell 接收到格式为 rel rid num 的命令时，首先处理一些格式错误的情况，之后调用进程管理器的 release_resource 函数为该进程释放资源。
6. to：该函数不需要输入其他参数，用于模拟时钟中断。当 test shell 接收到 to 命令，直接调用进程管理器的 time_out 函数执行时钟中断的处理。
7. lr：该函数不需要输入其他参数，用于列出所有资源的信息，包括 rid，status 等。当 test shell 接收到 lr 命令时，调用资源管理器的 get_resource_list，遍历所有资源并返回其信息。
8. lp：该函数不需要输入其他参数，用于列出所有进程的信息，包括 rid，status 等。当 test shell 接收到 lp 命令时，分别调用进程管理器的 get_running_list，get_ready_list，get_blocked_list，分别遍历三种进程列表并返回其 pid。
9. lb：该函数不需要输入其他参数，用于列出在阻塞等待资源的进程列表 blocked_list。当 test shell 接收到 lb 命令时，调用资源管理器的 get_blocked 函数得到每个资源的等待队列。
10. run：该函数输入参数为 filename，用于读入名为 filename 的文件内容，将内容作为命令输入。

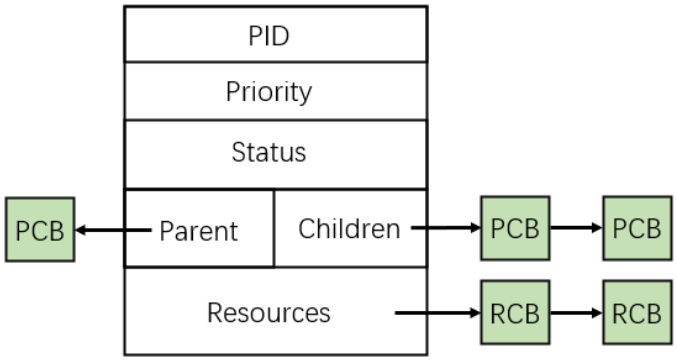
11. exit：该函数不需输入其他参数，用于结束进程。

(三)模块实现

下面将逐一分析具体模块的原理与实现。

1) PCB

进程控制块 PCB 记录了一个进程的运行情况和控制信息。PCB 结构如下图所示：



其中各组成部分如下：

a) Pid：程的 id 标识符，用于唯一的区分不同的进程

b) Priority：进程运行的优先级。分为三种：0，1，2

- 0—init
- 1—user
- 2—system

c) Status：该进程当前的状态。分为三种：ready，running，blocked

- ready：就绪状态，表示该进程已经分配到除 CPU 以外的资源，获得 CPU 的使用权就可以运行
- running：运行状态，表示该进程已经取得 CPU 的使用权。
- blocked：阻塞状态，表示该进程因为无法申请到足够的资源而无法执行的状态

d) Parent：父进程，进程在创建时正在运行的进程作为该进程的父进程

e) Children：子进程，当该进程在运行是创建的进程为子进程

f) Resources：该进程所占有的资源列表，元素为 RCB

代码实现如下：

PCB.py

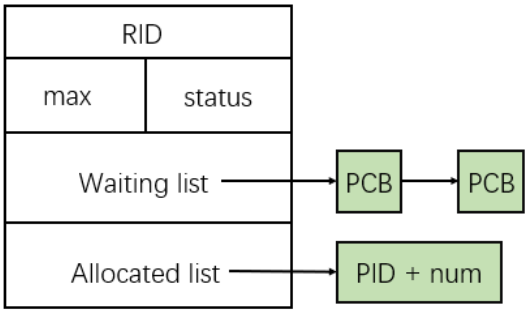
```
class PCB:
    def __init__(self,pid,priority):#__init__是特殊方法
        #声明_pid,_priority 等是 pcb 的变量
        self._pid=pid    #0=init,1=user,2=system
#用于确定放在 ready list 的第几位
        self._priority=priority
        self._status='ready'
        self._parent=None
        self._children=[]
        self._resources=[]
    def get_info(self):
        return {
            "pid":self._pid,
            "priority":self._priority,
            "status":self._status,
            "parent":self._parent.get_pid() if self._parent is not None else None,
            "children":[x.get_pid() for x in self._children],
            "resources":self._resources
        }

    def set_parent(self,parent):
        self._parent=parent

    def set_children(self,chilren):
        self._children.append(chilren)#把新孩子放到队尾

    def get_pid(self):
```

```
        return self._pid

    def get_parent(self):
        return self._parent

    def delete_children(self,pid):
        children=[x for x in self._children if x.get_pid()==pid]
        children_to_delete=children[0]
        self._children.pop(self._children.index(children_to_delete))

    def get_all_resources(self):
        return self._resources

    def get_status(self):
        return self._status

    def get_children(self):
        return self._children

    def set_status(self,status):
        self._status=status

    def get_priority(self):
        return self._priority

    def get_resource(self,rid):#返回 rid 对应的 rcb
        for x in self._resources:
            if x['rid']==rid:
                return x
```

```
            return []#没找到对应 rid 的资源则返回空


        def set_resources(self,resourcer):
            resource_exist=[x for x in self._resources if x['rid']==resourcer['rid']]
            if len(resource_exist)==0:
                self._resources.append(resourcer)
            else:
                for x in resource_exist:
                    if resourcer['status']==0:
                        self._resources.pop([y['rid'] for y in
self._resources].index(x['rid']))
                    else:
                    x['status']=resourcer['status']
```

2) RCB

资源控制块 RCB 记录了一个资源的所有信息，以用于分配和回收资源。RCB 结构如下所示：



其中各组成部分如下：

a) Rid：资源的唯一标识符，本实验中有 R1,R2,R3,R4 这 4 种。

b) Max：资源的初始分配量，也是最大持有量

c) Status：资源目前剩余的数量

d) Waiting list：在等待该资源的进程序列，元素为 PCB

e) Allocated list：已经为其分配该资源的进程序列

实现代码如下：

RCB.py

```
class RCB:
    def __init__(self,rid,status):
        self._rid=rid
        self._max=status
        self._status=status
        self._waiting_list=[]
        self._allocated_list=[]

    def set_allocated_list(self,process):
        #根据 pid 查 process 在不在分配队里，
        allocated_existed=[x for x in self._allocated_list if process['pid']==x['pid']]
        #若不在，则将该 process 放到分配队队尾
        if len(allocated_existed)==0:
            self._allocated_list.append(process)
        # 若在，对于每个进程，如果分配给他的是 0，就不要在分配队里了
        # 否则，把该 process 占有的记录到队里面
        # x 用于循环进程，status 表示分配出去了多少
        else:
            for x in allocated_existed:
                if process['status']==0:
                    self._allocated_list.pop([y['pid'] for y in self._allocated_list].index(x['pid']))
                else:
                    x['status']=process['status']

    def set_waiting_list(self,process):#某个进程在等待 Ri 资源？
```

```
            waiting_existed=[x for x in self._waiting_list if process['pid']==x['pid']]

            if len(waiting_existed)==0:#原来没在等，则加入等待序列
                self._waiting_list.append(process)
            else:#若已经在等
                for x in waiting_existed:
                    if process['status']==0:#该进程对于资源 Ri 需求量为 0，则把别的 pop 上来，把他移开
                        self._waiting_list.pop([y['pid'] for y in self._waiting_list].index(x['pid']))
                    else:#否则，按需要的量等待
                        x['status']=process['status']

        def get_status(self):
            return self._status

        def set_status(self,num):
            self._status=num

        def get_allocated_status(self,pid):#给某个进程分配了多少
            allocated=[x for x in self._allocated_list if x['pid']==pid]
            if len(allocated)!=0:
                return allocated[0]['status']
            else:
                return 0

        def get_rid(self):
            return self._rid

        def get_waiting_list(self):
```

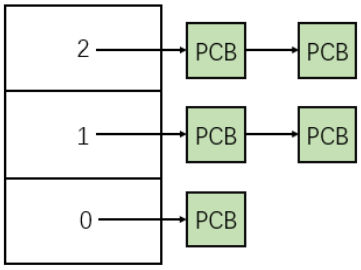
```
        return self._waiting_list


    def get_max(self):
        return self._max


    def get_allocated_list(self):
    return self._allocated_list
```

3) Processor

进程管理器 processor 用于管理、调度进程。该模块包含 3 个队列：

a) ready list：用于存放当前状态为 ready 的进程

ready list 结构如下：



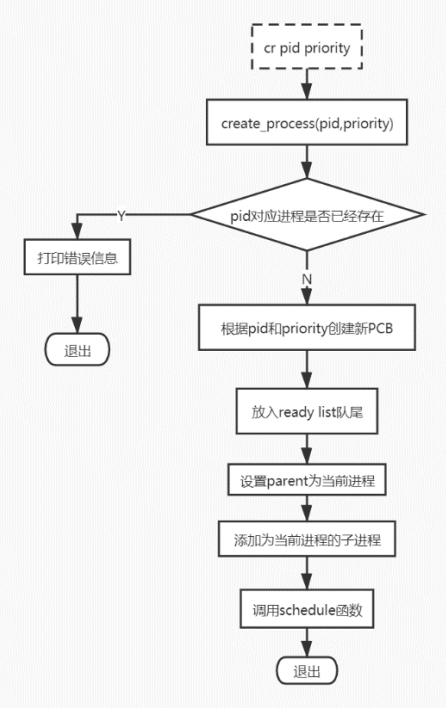
b) running list：用于存放当前状态为 running 的进程

c) blocked list：用于存放当前状态为 blocked 的进程

该模块还包含了供 test shell 模块调用的函数：

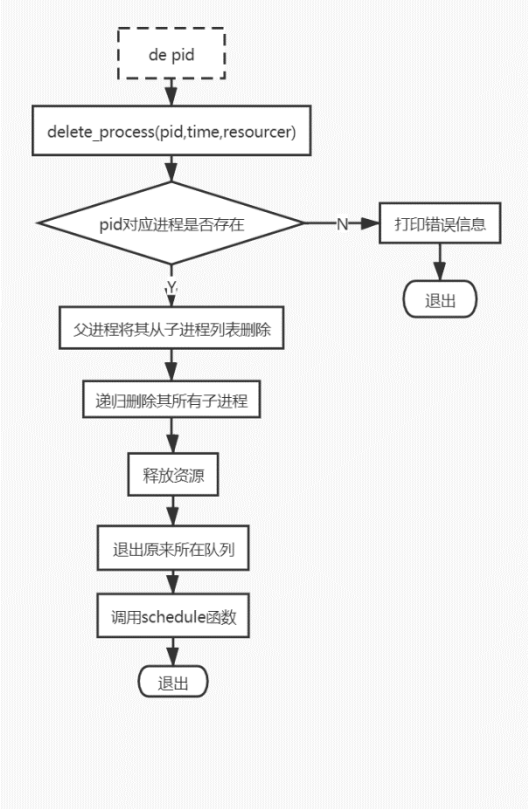
a) create_process：该函数首先根据 pid 判断该进程是否已经存在，若已经存在，则打印“process pid has existed”并终止该命令。若该进程之前并不存在，则根据输入的 pid 和 priority 创建一个新的 PCB，并将其加入进程管理器的 ready list，表明该进程已经准备好运行。由于是在进程运行过程中创建的新进程，故将当前 running list 的首位设为其父进程。对应的，将新进程设置为当前在运行的进程的子进程。最后再调用进程管理器的 schedule 函数进行进程的调度。

该函数流程图如下：



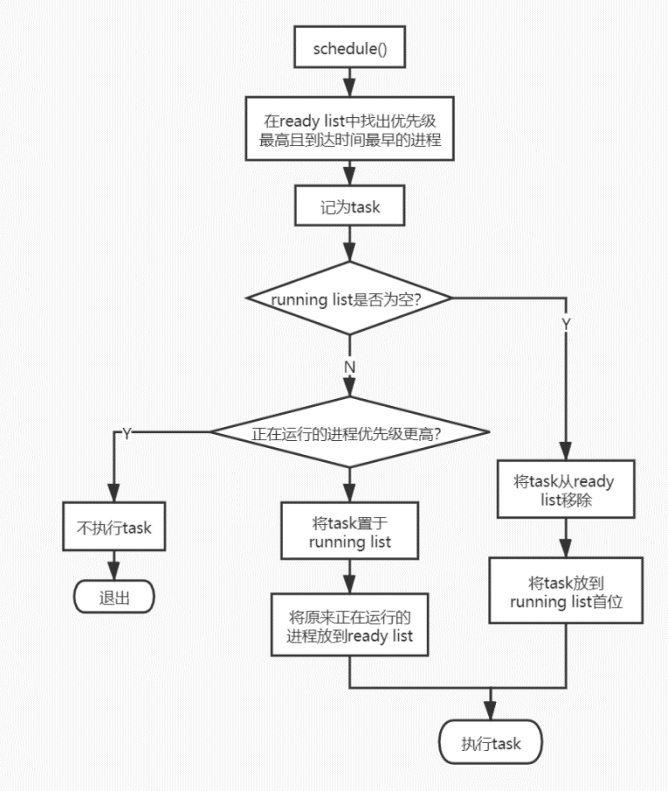
b) delete_process：该函数首先根据 pid 在进程管理器中调用 get_process_list 寻找对应进程，并判断进程不存在的错误情况。找到 pid 对应的进程 process 后，该进程的父进程将其移出子进程队列，该进程再将自己所有的子进程也递归删除。之后将该进程移出等待资源队列 waiting_list 和进程队列（根据状态判断是在 running/blocked/ready list 中的哪一个）。最后调用 schedule 函数重新调度进程。

该函数流程图如下：



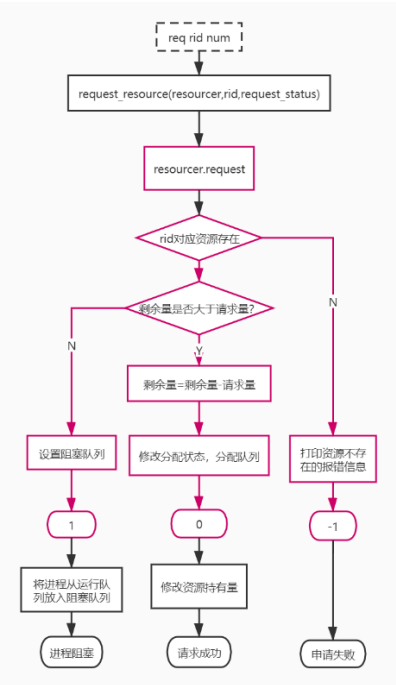
c) schedule：该函数首先在 ready list 中寻找出优先级最高，到达时间最早的进程 task，接着判断 running list 是否为空。若为空，则直接将 task 放到 running list 的第一个开始运行，若不为空，则比较当前正在运行的进程和 task 的优先级，若 task 的优先级更高，则抢占当前进程，将当前进程放到 ready list，将 task 放到 running list，若 task 优先级更低，则不操作，仍然让当前进程继续运行。

该函数流程图如下：



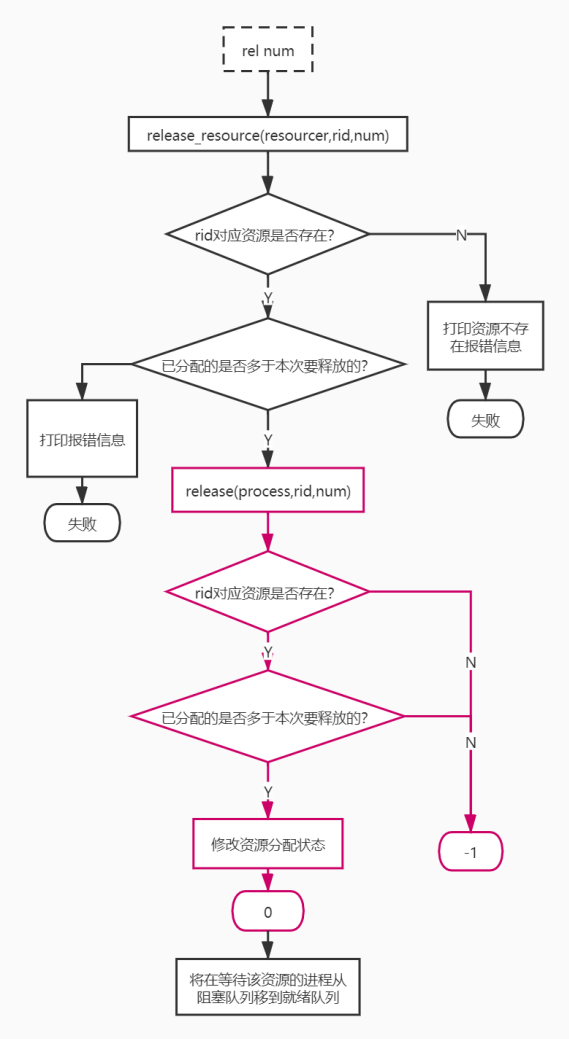
d) request_resource：该函数会再调用资源管理器的 request 函数来请求资源，根据 request 函数的返回值判断是否请求成功。若请求成功，则修改进程的资源状态，若请求的量过大，则将进程放入阻塞队列。最后调用 schedule 函数重新调度进程。

该函数流程图如下：



e) release_resource：该函数首先判断进程和资源的存在性，若不存在会打印报错信息并返回，若存在则判断原来为该进程分配的资源量与本次需要释放的资源量的大小，若本次需要释放的资源量小于等于原来为该进程分配的资源量，即本次操作是有效的，则调用资源管理器的 release 函数进行释放。若释放成功，则判断是否有进程在等待该资源，且如果该进程的需求量此时可以被满足，则该进程从阻塞队列中移出，并为其分配所需要的资源。最后调用 schedule 函数重新调度进程。

该函数流程图如下：

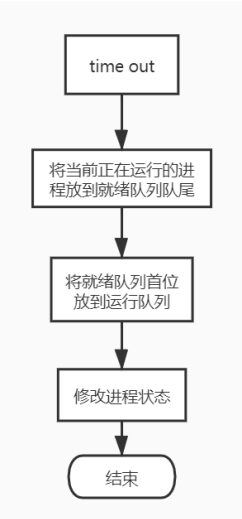


f) get_process_list：返回三个 list 的进程合集，即返回所有的进程信息。

g) time_out：该函数暂停当前的进程，并开始运行 ready list 的首位。首先将目前正在运行的进程从 running list 放置到 ready list 的尾部，并

将状态设为 ready，再从 running list 将其移除。最后调用 schedule 函数重新调度进程。

该函数流程图如下：



该模块代码实现如下：

Processor.py

```
from PCB import PCB


class Processor:
    def __init__(self):
        #创建内部的三个 list
        self._ready_list=[]
        self._running_list=[]
        self._blocked_list=[]


    def create_process(self,pid,priority):
        #用 pid 判断进程是否已经存在(pid  在所有进程的 pid 集合)


        if len(self._running_list)!=0 and pid in [x.get_pid() for x in self.get_process_list()]:
            print("process"+pid+"has existed")
            return
```

```
            #进程原来不存在,创建新的 pcb 并把它附加在 ready list 末尾
            new_pcb=PCB(pid,priority)
            self._ready_list.append(new_pcb)

            #双向设置 parent
            if len(self._running_list)!=0:
                new_pcb.set_parent(self._running_list[0])
                self._running_list[0].set_children(new_pcb)

            self.schedule()

        def delete_process(self,pid,time,resourcer):#资源管理器
            #根据 pid 找到对应的进程 process
            processes=[x for x in self.get_process_list() if x.get_pid()==pid]
            if len(processes)==0:
                print("process is not exist")
            else:
                process=processes[0]

            #父进程删除该子进程
            parent=process.get_parent()
            parent.delete_children(pid=pid)
            #删除孩子
            process_children=process.get_children()

[self.delete_process(pid=x.get_pid(),time=time+1,resourcer=resourcer) for x
in process_children]
```

```
            #释放资源
            resources=process.get_all_resources()
            if len(resources)!=0:
                for i in range(len(resources)):

self.release_resource(resourcer=resourcer,rid=resources[0]['rid'],release_statu
s=resources[0]['status'],process=process)

            #移出等待资源队列，对每个资源都 status=0 移除等待
            for x in resourcer._resource_list:
                x.set_waiting_list(process={
                    "pid":process.get_pid(),
                    "priority":process.get_priority(),
                    "status":0
                })

            #移除进程序列
            process_status=process.get_status()
            if process_status=='running':
                self._running_list.pop([x.get_pid() for x in
self._running_list].index(pid))
            elif process_status=='blocked':
                self._blocked_list.pop([x.get_pid() for x in
self._blocked_list].index(pid))
            elif process_status=='ready':
                self._ready_list.pop([x.get_pid() for x in
self._ready_list].index(pid))

            #调度
```

```
        if time==0:#time==0 说明是当前这个进程，调度一次，如果是子进程在删除的时候执行到这里，不执行调度
            self.schedule()


    def schedule(self):
        #在 ready list 中找出要运行的进程 tasks——优先级最高，到达最早
        system=[x for x in self._ready_list if x.get_priority()==2]
        user=[x for x in self._ready_list if x.get_priority()==1]
        if len(system)!=0:
            tasks=system
        elif len(user)!=0:
            tasks=user
        else:
            tasks=[x for x in self._ready_list if x.get_priority()==0]

        #若 running list 为空，直接把他放第一个开始运行
        if len(self._running_list)==0:
            self._running_list.append(tasks[0])
            tasks[0].set_status("running")
            self._ready_list.pop(self._ready_list.index(tasks[0]))
            print("process "+self._running_list[0].get_pid()+" is running")

        #若运行队列非空
        else:
            #tasks 优先级更大就抢占
            if tasks[0].get_priority()>self._running_list[0].get_priority():
                print("process "+tasks[0].get_pid()+" is running. "+"process " +self._running_list[0].get_pid()+" is ready")
```

```
                    #把原来在运行的放到 ready list
                    self._running_list[0].set_status("ready")
                    self._ready_list.append(self._running_list[0])
                    self._running_list.pop()
                    #把 tasks 放到 running list
                    tasks[0].set_status("running")
                    self._ready_list.pop(self._ready_list.index(tasks[0]))
                    self._running_list.append(tasks[0])
                #否则不动
                else:
                    print("process " + self._running_list[0].get_pid() + " is
running", end="\n")
                    return


    def  request_resource(self,resourcer,rid,request_status,process=None):#
资源管理器，rid，要多少，谁要
        #先处理特殊的 process 为 none 的情况，确定好 process
        if process is None:
            process=self._running_list[0]

        #调用 request 函数，得到返回值 code，根据 code 操作

code=resourcer.request(process=process,rid=rid,request_status=request_statu
s)
        print("process " + str(process.get_pid()) + " request " +
str(request_status) + " " + str(rid), end="\n")
```

```
            #若 code==0，请求成功
            if code==0:
                #判断
                process_resource=process.get_resource(rid)

                # 若本来没有，调用 set 进行设置
                if len(process_resource)==0:
                    process.set_resources(resourcer={
                        "rid":rid,
                        "status":request_status
                    })
                #若该 process 本来就有这个资源，叠加
                else:
                    process_resource['status']+=request_status

            #若 code==1，请求的太多，放入 blocked list
            elif code==1:
                process.set_status("blocked")
                self._blocked_list.append(process)
                self._running_list.pop(self._running_list.index(process))
                print("process "+process.get_pid()+" is blocked")
            self.schedule()


        def release_resource(self,resourcer,rid,release_status,process=None):#资源管理器，rid，放多少，哪个进程
            #process 没赋值，则默认为 running 的第一个
            if process is None:
                process=self._running_list[0]
```

```
            #如果 process 本来就没有 rid 的资源
            if len(process.get_resource(rid))==0:
                print("release resource that was not allocated")

            #正常情况。
            #先获得原来分配了多少
            status_allocated=int(process.get_resource(rid=rid)['status'])
            release_status=int(release_status)
            #若已分配的>要求释放的（释放部分），则调用 set，修改数值
            if status_allocated>=release_status:

code=resourcer.release(process=process,rid=rid,release_status=release_status
)
                #code=0,在资源方面释放成功
                if code==0:
                    process.set_resources(resourcer={
                        "rid":rid,
                        "status":status_allocated-release_status
                    })
                    print("release "+rid)
                    #把在等待 rid 的资源的进程释放出来
                    block_process=[x for x in self._blocked_list]#x is a pcb's
list，阻塞的进程们
                    flag=False
                    #对于每个阻塞的进程 x
                    for x in block_process:
                        rcb=resourcer.get_rcb(rid=rid)
                        waiting_list=rcb.get_waiting_list()#从阻塞队里面
找出等待该 rid 资源的进程们
                        for y in waiting_list:
```

```
                        if y['pid']==x.get_pid():
                        # 如果比我先来的都被阻塞了，我就退出
                            if flag is True: return
                        #否则
                            if rcb.get_status()>=y['status']:#剩余的资源大于正在阻塞的进程所需要的
                                #x 从 blocked list 去 ready list
                                x.set_status('ready')
                                x.set_resources(resourcer={
                                    "rid":rid,
                                    "status":y['status']
                                })
                                self._ready_list.append(x)

self._blocked_list.pop(self._blocked_list.index(x))
                                #y 从资源等待 list 释放

rcb.get_waiting_list().pop(rcb.get_waiting_list().index(y))

                                #索取资源

resourcer.request(process=x,rid=rid,request_status=y['status'])
                                #进行调度
                                self.schedule()
                                print("wake up process "+y['pid'],end="\n")
                            else:#被阻塞，阻塞判断位改变
                                flag=True
                #code=-1
                else:
```

```
                print("release error")
        #否则报错
        else:
            print("release too much")

def get_process_list(self):
        return self._running_list+self._ready_list+self._blocked_list

def time_out(self):
        length=len(self._ready_list)
        print("process " +self._running_list[0].get_pid()+" is ready")

        #running 的第一个放到 ready 尾部
        first_running_process=self._running_list[0]
        first_running_process.set_status('ready')
        self._ready_list.append(first_running_process)
        self._running_list.pop()

        #再次调度
        self.schedule()



def get_running_list(self):
        return [x.get_pid() for x in self._running_list]

def get_process_info(self,pid):
        process_list=self.get_process_list()
        info=[x.get_info() for x in process_list if x.get_pid()==pid]
        if len(info)==0:
            print("process"+pid+"is not existed")
```

```
            return
        print(info[0])

    def get_ready_list(self):
        return [x.get_pid() for x in self._ready_list]

    def get_blocked_list(self):
        return [x.get_pid() for x in self._blocked_list]

    def get_ready_list_zero(self):
        return [x.get_pid() for x in self._ready_list if x.get_priority()==0]

    def get_ready_list_one(self):
        return [x.get_pid() for x in self._ready_list if x.get_priority()==1]

    def get_ready_list_two(self):
        return [x.get_pid() for x in self._ready_list if x.get_priority()==2]
```

4) Resourcer

资源管理器 Resourcer 用于管理资源的申请，分配和回收。该模块包含一个 resource_list，由 4 个资源控制块 RCB 组成。该模块还包含了供进程管理器调度资源的函数：

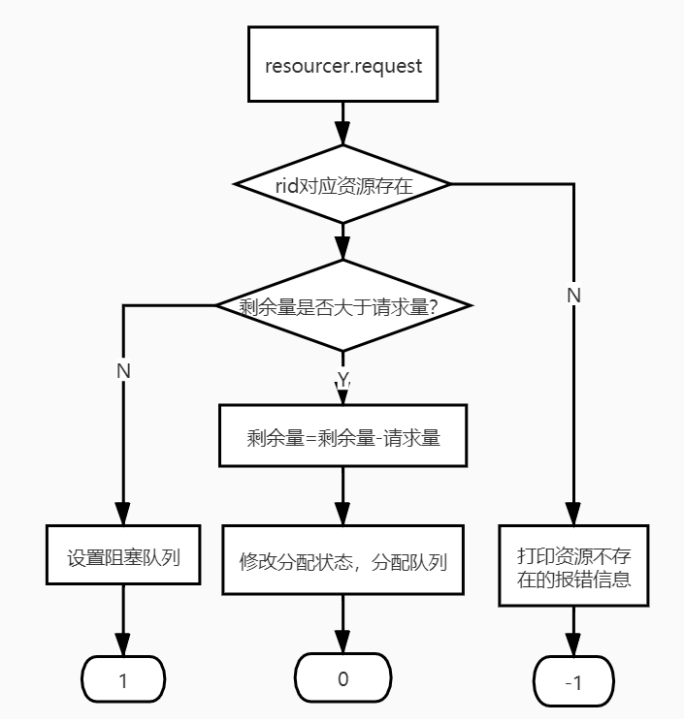
a) request：当进程管理器调用 request_resource 时，将在内部调用该函数进行分配过程并得到分配结果。该函数输入参数为：

- process：用于表明是哪个进程需要申请资源
- rid：用于表明需要申请哪个资源
- request_status：用于表明需要申请的资源数量

该函数首先在资源管理器的 resource_list 中找到 rid 对应的资源控制块 RCB，若未找到对应 RCB，则直接返回-1，告知进程管理器寻找资源失败。若寻找成功，则用 resource_requested 保存该 RCB，接下

来比较该资源的剩余数量与请求量，若剩余量大于或等于请求量，说明请求可以被满足，则将资源的剩余量减去请求量，并修改资源的分配列表，返回值设为 0 以告知进程管理器该分配操作成功。若剩余量小于申请量，则请求无法被满足，则返回值设为 1，用于告知进程管理器要将进程放置于阻塞队列，并修改该资源的等待队列。

该函数流程图如下：

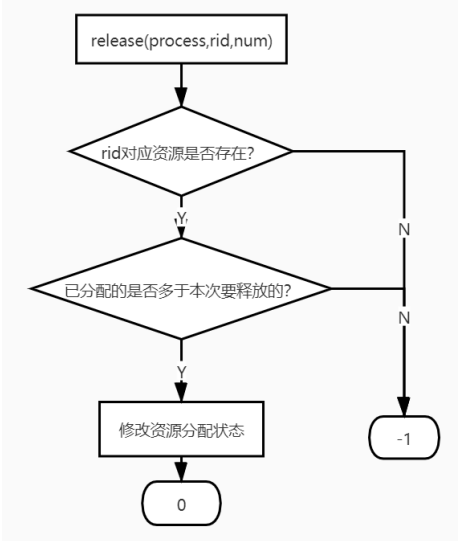


b) release：当进程管理器调用 request_status 时，将在函数内部调用函数以实现资源释放过程。该函数的输入参数有：

- process：用于表明是哪个进程需要申请资源
- rid：用于表明需要申请哪个资源
- release_status：用于表明需要申请的资源数量

该函数首先在资源管理器的 resource_list 中找到 rid 对应的资源控制块 RCB，若未找到对应 RCB，则直接返回-1，告知进程管理器寻找资源失败。若寻找成功，则用 release 保存该 RCB，接下来比较该资源原来分配除去的和这次要释放的量，若要释放的量大于原来分配除去的量，则打印报错信息并退出，否则修改该资源的分配状态，分配成功。

该函数流程图如下：



c) get_resource_list：该函数用于返回资源列表的 rid，status 等信息，用于被资源管理器和进程管理器在判断某个资源是否存在。

该模块的代码实现如下：

Resourcer.py

```
from RCB import RCB


class Resourcer:
    def __init__(self):
        r1 = RCB("R1", 1)
        r2 = RCB("R2", 2)
        r3 = RCB("R3", 3)
        r4 = RCB("R4", 4)

        self._resource_list=list()

        self._resource_list.append(r1)
        self._resource_list.append(r2)
        self._resource_list.append(r3)
        self._resource_list.append(r4)
```

```
        def request(self,process,rid,request_status):#哪个进程，要什么，要多少
            #若 rid 对应的资源存在于 resource list
            if len([x for x in self._resource_list if x.get_rid()==rid])!=0:
                resource_requested=[x for x in self._resource_list if x.get_rid()==rid][0]
            #若剩余的>=请求的，申请成功
                if resource_requested.get_status()>=request_status:
                    #剩余资源-=需求量

resource_requested.set_status(resource_requested.get_status()-request_status)
                    #维护分配资源

allocated_status=resource_requested.get_allocated_status(process.get_pid())
                    resource_requested.set_allocated_list(process={
                        "pid":process.get_pid(),
                        "priority":process.get_priority(),
                        "status":request_status+allocated_status
                    })
                    return 0
                else:
                    #若剩余的<请求的，阻塞进程
                    resource_requested.set_waiting_list(process={
                        "pid":process.get_pid(),
                        "priority":process.get_priority(),
                        "status":request_status
                    })
                    return 1
            else:
                print("Resource is not existed!")
```

```
                return -1


        def release(self,process,rid,release_status):
            #rid 对应的资源存在
            if len([x for x in self._resource_list if x.get_rid()==rid])!=0:
                release=[x for x in self._resource_list if x.get_rid()==rid][0]#release is a RCB class
                allocated_status=release.get_allocated_status(process.get_pid())
                #若要释放的大于本来给他分配的，失败
                if allocated_status<release_status:
                    print("release too much")
                    return -1
                else:
                #否则成功
                    #维护已分配状态
                    release.set_status(release_status+release.get_status())
                    #维护资源状态
                    release.set_allocated_list(process={
                        "pid": process.get_pid(),
                        "priority": process.get_priority(),
                        "status": allocated_status-release_status
                    })
                    return 0
            else:
                print("Resource is not existed!")
                return -1


        def get_rcb(self,rid):
            rcb=[x for x in self._resource_list if x.get_rid()==rid]
            if len(rcb)!=0:
```

```
            return rcb[0]
        else:
            return []


    def get_resource_list(self):
        return [{
            "rid:":x.get_rid(),
            "max:":x.get_max(),
            "status:":x.get_status(),
            "waiting list:":x.get_waiting_list(),
            "allocated list:": x.get_allocated_list()
            }
            for x in self._resource_list
        ]


    def get_blocked(self):
        return [{
            "rid":x.get_rid(),
            "pid":x.get_waiting_list()
        }
    for x in self._resource_list]
```

5) main

该函数作为程序的主程序部分，实现了程序的启动和 test shell 部分的功能。运行后，该函数首先调用 system_init 函数进行资源管理器和进程管理器的初始化工作，之后可接受用户直接输入命令，根据命令格式在 testshell 中选择合适的函数执行。

该模块代码实现如下：

main.py

```
import re
```

```
import sys
from Resourcer import Resourcer
from Processor import Processor

def system_init():
    processor_tmp=Processor()
    resourcer_tmp=Resourcer()

    #for x in processor_tmp.get_running_list():
     #    print(x + " ", end='')
    return processor_tmp,resourcer_tmp

def read_test_shell(filename):
    try:
        file=open(filename)
        text_lines=file.readlines()
        for line in text_lines:
            analysis(line)
        file.close()
    except IOError as e:
        print(e.strerror)
        return 0
    else:
        file.close()
        return 0

def analysis(inputs):
    #格式处理，删除两边空格，删除输入中多个空格
    inputs=inputs.strip()
    inputs=re.sub('[ ]+',' ',inputs)
```

```
    #解析输入，用空格分割，保存为 ip 数组
    ip=inputs.split(" ")
    #若 ip 长度为 3
    if len(ip)==3:
    #create process
        if ip[0]=='cr':#cr pid priority
            pid=ip[1]
            priority=int(ip[2])

            if priority!=1 and priority!=2:
                print("priority can only be 1 or 2")
                return 0

            processor.create_process(pid=pid,priority=priority)
    #request resource
        elif ip[0]=='req':#req rid num
            rid=ip[1]
            num=float(ip[2])
            if num.is_integer() and abs(num)==num:#num is positive integer
                processor.request_resource(resourcer=resourcer,rid=rid,
request_status=num)
            else:
                print("request number should be a positive integer")
                return 0

    #release resource
        elif ip[0]=='rel':
            rid=ip[1]
```

```
                # 特殊错误情况
                try:
                    num=float(ip[2])
                except ValueError as e:
                    print(e)
                    return 0
                else:
                    if num.is_integer() and abs(num)==num:
                        processor.release_resource(resourcer=resourcer, rid=rid,
release_status=num)
                    else:
                        print("release num should be a positive number")

            else:
                print(ip[0]+"is an invalid command")
                return 0
        #长度为 2
        elif len(ip)==2:
        #show a process
            if ip[0]=='lp':
                pid=ip[1]
                processor.get_process_info(pid=pid)
                return 0
        #delete process
            elif ip[0]=='de':
                pid=ip[1]
                if pid=='init':
                    print("cannot delete init process")
                    return 0
                processor.delete_process(pid=pid,time=0,resourcer=resourcer)
```

```
        elif ip[0]=='run':
            filename=ip[1]
            read_test_shell(filename)

        else:
            print(ip[0]+"is an invalid command")
#长度为 1
else:
#time out
        if inputs=='to':
            processor.time_out()
#list all process
        elif inputs=='lp':
            running=processor.get_running_list()
            ready=processor.get_ready_list()
            blocked=processor.get_blocked_list()

            print("running list:"+str(running))
            print("ready list:" + str(ready))
            print("blocked list:" + str(blocked))

            return 0
        elif inputs=='ready':
            zero=processor.get_ready_list_zero()
            one=processor.get_ready_list_one()
            two=processor.get_ready_list_two()
            running=processor.get_running_list()
            one=running+one
            print("priority=2:"+str(two))
            print("priority=1:" + str(one))
```

```
            print("priority=0:" + str(zero))
    #list all resources
        elif inputs=='lr':
            resource_list=resourcer.get_resource_list()
            [print(x) for x in resource_list]
            return 0
        elif inputs=='lb':
            block_list=resourcer.get_blocked()
            [print(x) for x in block_list]
            return 0
        elif inputs=='init':
            processor.create_process('init', 0)    #  创建 init 进程
    #exit
        elif inputs=='exit':
            return -1
    #help document
        elif inputs=='help':
            print("run test file: run filename")
            print("create a process: cr pid priority")
            print("request a resource: req rid num")
            print("release a resource: rel rid num")
            print("list a process: lp pid")
            print("delete a process: del pid")
            print("time out: to")
            print("list all process status: lp")
            print("list ready list by priority: ready")
            print("list all resource status: lr")
            print("list blocked list: lb")
            print("exit: exit")
            return 0
```

```
            else:
                print(inputs+"is an invalid command")


        #for x in processor.get_running_list():
        #      print("process "+x+" is running",end='\n')


if __name__=='__main__':
        processor, resourcer =system_init()


        if len(sys.argv)==2:
            filename=sys.argv[1]
            read_test_shell(filename)
        if len(sys.argv)==1:#读用户输入
            while(True):
                X=input(" ")
                code=analysis(X)
                if code==-1:
                    break
```

九、实验数据及结果分析：

(一)本次实验测试数据

Input.txt

```
init
cr x 1
cr p 1
cr q 1
cr r 1
ready
```

```
to
req R2 1
to
req R3 3
to
req R4 3
lr
to
to
req R3 1
req R4 2
req R2 2
lb
to
de q
to
to
lr
exit
```

(二)预期测试结果

1) init：

创建 init 进程，此时系统中仅有 init 进程，故输出：process init is running

2) cr x 1：

创建优先级为 1 的 x 进程，由于其优先级大于 init 进程，将会抢占进程 init，故输出：process x is running. process init is ready

3) cr p 1：

创建优先级为 1 的 p 进程，但由于其优先级与 x 相等，无法抢占 x 进程，故输出：process x is running.

4) cr q 1：

创建优先级为 1 的 q 进程，但由于其优先级与 x 相等，无法抢占 x 进程，故输出：process x is running.

5) cr r 1：

创建优先级为 1 的 r 进程，但由于其优先级与 x 相等，无法抢占 x 进程，故输出：process x is running.

6) ready：

按照 2，1，0 的优先级顺序输出此时的 ready list 信息，由于没有优先级为 2 的进程，故 priority =2 为空。优先级为 1 的进程的到达顺序为 x->p->q->r，x 正在运行，在 running list 中，故 priority=1 的有 p，q，r。优先级为 0 的仅有 init 进程。所以输出：

priority=2:[]

priority=1:['x', 'p', 'q', 'r']

priority=0:['init']

7) to：

由于 ready list 中 prority=1 的有 ['p', 'q', 'r']，且没有 priority=2 的进程，时钟中断发生后，进程 x 将被进程 p 抢占，此时 ready list 中 prority=1 的有 ['q', 'r', 'x']，running list 为 p 进程。故此时所以输出：

process x is ready

process p is running

8) req R2 1：

当前运行进程为 p，则为进程 p 申请 1 个 R2 资源。此时没有发生进程抢占，故输出：

process p request 1 R2

process p is running

9) to：

时钟中断发生，由于没有优先级为 2 的进程，故进程 p 被进程 q 抢占，此时 running list 为 q，ready list 中 priority=1 的有['r', 'x', 'p']，所以输出：

process p is ready

process q is running

10) req R3 3：

当前运行进程为 q，则为进程 q 申请 3 个 R3 资源，此时没有发生进程抢占，输出：

process q request 3 R3

process q is running

11) to：

时钟中断发生，由于没有优先级为 2 的进程，故进程 q 被进程 r 抢占，此时 running list 为 r，ready list 中 priority=1 的有['x', 'p', 'q']，所以输出：

process q is ready

process r is running

12) req R4 3：

当前运行进程为 r，则为进程 r 申请 3 个 R4 资源，此时没有发生进程抢占，输出：

process r request 3.0 R4

process r is running

13) lr：

由于 R2 被 p 占用 1 个，R3 被 q 占用 3 个，R4 被 r 占用 3 个，则输出：

{'rid:': 'R1', 'status:': 1}

{'rid:': 'R2', 'status:': 1}

{'rid:': 'R3', 'status:': 0}

{'rid:': 'R4', 'status:': 1}

14) to：

时钟中断发生，由于没有优先级为 2 的进程，故进程 r 被进程 x 抢占，此时 running list 为 x，ready list 中 priority=1 的有['p', 'q', 'r']，所以输出：

process r is ready

process x is running

15) to：

时钟中断发生，由于没有优先级为 2 的进程，故进程 x 被进程 p 抢占，此时 running list 为 p，ready list 中 priority=1 的有['q', 'r', 'x']，所以输出：

process x is ready

process p is running

16) req R3 1：

当前运行进程为 p，则为进程 p 申请 1 个 R3 资源，但 R3 的 status 已经为 0，所以进程 p 被阻塞，ready list 中的下一个进程 q 被调度，所以输出：

process p request 1.0 R3

process p is blocked

process q is running

17) req R4 2：

当前运行进程为 q，则为进程 q 申请 2 个 R4 资源，但 R4 的 status 只有 1，所以进程 q 被阻塞，ready list 中的下一个进程 r 被调度，所以输出：

process q request 2 R4

process q is blocked

process r is running

18) req R2 2：

当前运行进程为 r，则为进程 r 申请 2 个 R2 资源，但 R2 的 status 只有 1，所以进程 r 被阻塞，ready list 中的下一个进程 x 被调度，所以输出：

process r request 2 R2

process r is blocked

process x is running

19) lb：

进程 p 阻塞在 R3 上且需要 1 个 R3，进程 q 阻塞在 R4 上且需要 2 个 R4，进程 r 阻塞在 R2 上且需要 2 个 R2，所以输出：

{'rid': 'R1', 'pid': []}

{'rid': 'R2', 'pid': [{'pid': 'r', 'priority': 1, 'status': 2.0}]}

{'rid': 'R3', 'pid': [{'pid': 'p', 'priority': 1, 'status': 1.0}]}

{'rid': 'R4', 'pid': [{'pid': 'q', 'priority': 1, 'status': 2.0}]}

20) to：

时钟中断发生，ready list 中 priority=1 的只有['x']，x 开始运行。所以输出：

process x is ready

process x is running

21) de q：

进程 q 被删除，由于其没有子进程，所以只有进程 q 被删除，其占有的 3 个 R3 资源被释放，唤醒等待 1 个 R3 的进程 p。p 被放到 ready list 中，x 仍在运行，所以输出：

release R3

process x is running

wake up process p

process x is running

22) to：

时钟中断发生，原来 ready list 中 priority=1 的有['p']，p 进程将 x 进程抢占，所以输出：

process x is ready

process p is running

23) to：

时钟中断发生，原来 ready list 中 priority=1 的有['x']，，所以输出：

process p is ready

process x is running

24) exit：

退出程序，返回程序成功退出信息。

(三)实际运行结果

在 Pycharm 中运行结果截图如下：

```
C:\Users\cy\untitled\Scripts\python.exe D:/myos1/main.py
 run input.txt
process init is running
process x is running. process init is ready
process x is running
process x is running
process x is running
priority=2:[]
priority=1:['x', 'p', 'q', 'r']
priority=0:['init']
process x is ready
process p is running
process p request 1.0 R2
process p is running
process p is ready
process q is running
process q request 3.0 R3
process q is running
process q is ready
process r is running
process r request 3.0 R4
process r is running

{'rid:': 'R1', 'status:': 1}
{'rid:': 'R2', 'status:': 1.0}
{'rid:': 'R3', 'status:': 0.0}
{'rid:': 'R4', 'status:': 1.0}
process r is ready
process x is running
process x is ready
process p is running
process p request 1.0 R3
process p is blocked
process q is running
process q request 2.0 R4
process q is blocked
process r is running
process r request 2.0 R2
process r is blocked
process x is running
{'rid': 'R1', 'pid': []}
{'rid': 'R2', 'pid': [{'pid': 'r', 'priority': 1, 'status': 2.0}]}
{'rid': 'R3', 'pid': [{'pid': 'p', 'priority': 1, 'status': 1.0}]}
{'rid': 'R4', 'pid': [{'pid': 'q', 'priority': 1, 'status': 2.0}]}
process x is ready
process x is running
release R3
process x is running
wake up process p
process x is running
process x is ready
process p is running
process p is ready
process x is running
```

与(二)中对结果比较可见，程序成功的完成了应有的功能。

七、实验结论：

根据相关测试的实验结果可见，本次实验的代码完成了进程和资源管理器，test shell 的设计，成功实现了进程的创建与撤销，running, blocked, ready 三个状态的转换，基于优先级的进程调度。并且成功通过 to 命令模拟了时钟中断及对时钟中断的处理，以及对于资源的申请和释放。

八、对本实验过程及方法、手段的改进建议：

(一)打印 PCB 信息

PCB 作为进程控制块，保存了进程的所有信息，因而打印进程控制块的信息能帮助我们了解进程的状态。

(1) 实现原理

进程控制块 PCB 的信息都存储在 class PCB 中，因此只需要定义如下函数，并在 main 函数中调用即可：

```
    def get_info(self):
            return {
                "pid":self._pid,
                "priority":self._priority,
                "status":self._status,
                "parent":self._parent.get_pid() if self._parent is not None else None,
                "children":[x.get_pid() for x in self._children],
                "resources":self._resources
        }
```

在 main 函数中的 analysis()函数定义命令“lp pid”为打印进程 id 为 pid 的进程信息。

```
        if len(ip)==2:
        #show a process
            if ip[0]=='lp':
                pid=ip[1]
                processor.get_process_info(pid=pid)
            return 0
```

(2) 测试方法与结果预测

运行完之前的一系列测试命令后在命令行输入“lp x”命令。预期将输出：

```
{'pid': 'x', 'priority': 1, 'status': 'running', 'parent': 'init', 'children': ['p', 'r'], 'resources': []}
```

(3) 测试结果与结论

```
lp x
{'pid': 'x', 'priority': 1, 'status': 'running', 'parent': 'init', 'children': ['p', 'r'], 'resources': []}
```

由结果可见，成功实现了进程控制块信息的打印。