

计算机操作系统原理

实验报告

班 级： 1504201

姓 名： 殷悦

学 号： 150120526

实验编号： 实验一

序言

为了培养学生的实践动手能力，帮助学生体会与验证所学的知识，设计该实验体系。课内实践安排 8 学时，其余需求学生可以课下完成。

实验报告基本要求：

1. 学生提交各个实验程序的流程图，并填入实验报告中；
2. 实验报告附上运行结果的截图；
3. 对实验的结果进行分析，写出自己的分析结果；
4. 实验的源代码、可执行代码和实验报告需要打包交到班级统一刻盘提交保存；个人文件的命名以个人的姓名+学号命名；班级的以班级编号命名。
5. 程序可视性要好，要有试验程序证明作品的正确性。
6. 所有的实验报告采用 A4 打印，字体字号遵循模版规定。
7. 源程序要加注释，要有测试数据及结果。
8. 每个实验的报告要另起一页。

基本要求：每个学生必须完成前 4 个实验，实验结果将逐个检查，通过者，方可提交实验报告，并获得实验成绩。

有精力的同学可以额外选作其他实验，额外多做的实验根据完成质量可以酌情加分 1-5 分。

报告的封面格式以及内容的格式不要自己调整。

实验一 进程及其资源管理

一、实验目的

1. 理解资源共享与互斥特性以及操作系统管理资源的基本方法。
2. 学会使用高级语言进行多线程编程的方法。
3. 掌握利用 VC++或 Java 线程库实现一个管理器，用来实现操作系统对进程及其资源的管理功能。
4. 通过该实验，学生可在源代码级完成进程及其资源管理方案的分析、功能设计、编程实现，控制进程间的同步、互斥关系。

二、实验要求

1. 知识基础：学生应在完成对进程和线程、调度、死锁等章节的学习。
2. 开发环境与工具：

硬件平台——个人计算机。

软件平台——Windows 操作系统，根据需要，任选安装 VC++语言、java 语言或 C 语言开发环境。

三、实验内容

1. 开发一个函数，**建立进程控制块和资源控制块结构**，并实现相关数据结构的初始化。
2. 开发一系列操作，由进程调用这些操作，达到控制进程申请或释放各种资源的目的。

四、实验方案指导

该实验方案由以下几个关键设计项目组成：

1. 进程数据结构表示。
2. 资源数据结构表示。
3. 进程对资源的操作。
4. 调度程序。
5. 用户功能 shell 界面。

五、实验方案实现范例

以下是对该项目中包含的设计功能的实现方法、实现过程、技术手段的描述，供参考。

1. 进程数据结构表示。

使用结构类型设计实现进程 PCB 表，它包含以下成员：

- ①进程 ID——进程的唯一标识，供其他进程引用该进程；
- ②内存——是一个指针链表，它在创建进程时已申请完毕，可用链表实现；
- ③其他资源——指除去内存之外的所有资源；
- ④进程状态——包括两个数据类型，一个是状态码，另一个是状态队列链表指针；
- ⑤生成树——包括两个数据类型，本进程的父进程和本进程的子进程；
- ⑥优先级——供进程调度程序使用，用来确定下一个运行进程，可以设定为静态整数。

2. 资源数据结构。

每个资源都用一个称为资源控制块的数据结构表示。使用结构类型设计实现资源控制块 RCB。

资源控制块包括以下字段成员：

- ①RID-资源的唯一标识，由进程引用；
- ②资源状态——空闲 / 已分配；
- ③等待队列——是被本资源阻塞的进程链表，本资源正被其他所有资源都设定为静态数据，系统启动时初始化。

3. 进程管理及进程对资源的操作。

进程操作及进程状态转换归纳如下：

- ①进程创建——（无）→就绪；
- ②申请资源——运行→阻塞；
- ③资源释放——阻塞→就绪；
- ④删除进程——（任何状态）→（无）；

5.调度程序——就绪→运行或运行→就绪。

具体实现步骤如下：

(1)根据上述数据结构，用高级语言设计相应函数，分别实现创建进程、删除进程、挂起进程、唤醒进程等功能。

(2)设计一个函数，实现调度程序，在每个进程操作执行完毕后，自动调用执行该调度程序。

(3)实现两个资源操作：申请资源和释放资源。

相关参考算法如下：

=====

```

request(RID)                                /*申请资源算法*/
{
    r=get_RCB(RID);                          /*获取资源控制块首地址*/
    if (r->status=='free') {                  /*资源可用*/
        r->status='allocated';                /*分配给调用进程，*/
        insert(self->other_resources,r); } /*插入一个 RCB 指针指向进程资源链
表； */
    else {                                    /*资源不可用*/
        self->status.type='blocked';          /*记录阻塞*/
        self->status.list=r;                  /*指向所请求资源的 RCB*/
        remove(RL,self);                      /*将进程从就绪队列中删除*/
        insert(r->waiting_list,self); }       /*插入资源等待队列*/
    scheduler( );                            /*调度程序运行选择下一个运行进
程*/
}

release(RID)                                /*释放资源算法*/
{
    r=get_RCB(RID);                          /*获取资源控制块首地址*/
    remove(self->other_resource,r);            /*从进程资源链表中删除该资源*/
    if(waiting_list==NULL) r->status='free'; /*等待队列为空，置资源状态为空闲
*/
    else {                                    /*等待队列不为空*/
        remove(r->waiting_list,q);           /*从等待队列中移出一个进程 q*/
        q->status.type='ready';               /*将进程 q 的状态设为就绪*/
        q->status.list=RL;                    /*进程 q 的状态指针指向就绪队列*/
        insert(RL,q); }                     /*进程 q 插入就绪队列*/
    scheduler( );                            /*调度程序运行选择下一个运行进程*/
}
=====

```

4. 调度程序。

调度策略采用**固定优先级和可剥夺优先级调度算法**。

即调度程序必须维护 n 个不同优先级的就绪队列，各就绪队列可为空，也可包含多个进程。0 级进程优先级最低， $n-1$ 级进程优先级最高。创建进程时就赋予了固定的优先级，并在进程的生存期内保持不变。当新进程创建或阻塞进程被唤醒时，它就被插入同级的就绪队列中。

调度程序按“先来先服务”和优先级“从高到低”的方式处理就绪队列。即从最

高优先级的非空就绪队列的队首选择一个进程进入运行态。这样的调度策略很容易导致“饥饿”，进程出现。因为对进程 q 来说，只有当优先级高于自己的所有进程都运行完毕，或都进入阻塞状态时，它才能得到运行权。

可以假定系统中至少有一个进程处于就绪态。为确保这一点，设计一个特殊进程 `init`，该进程在系统初始化时自动创建，并赋予最低优先级 0 级。`init` 进程有两个作用：充当闲逛进程，该进程运行时不申请任何资源，以免被阻塞；作为第一个被创建的进程，它没有父进程，可创建比自己优先级高的其他进程。所以 `init` 进程是进程生成树的根进程。

采用优先级策略的调度程序的常见结构知下所示：

```
=====
scheduler()
{  找出最高优先级进程 p;

    If(self->priority < p->priority) || self->status.type != 'running' || self = nil)

        preempt(p, self);    /*调度进程 p，替换当前进程 self*/

}
```

```
=====
```

每当任一进程的操作执行完毕，必须执行进程调度程序，它是当前运行进程的一部分。进程调用该函数，后者决定该进程是继续执行还是被其他进程剥夺运行权。作出判断的依据是：**是否存在高优先级进程 p** ，如果存在， p 将剥夺 `self` 的运行权。

当前进程的运行权被剥夺的情况有以下两种：

①当前进程刚刚完成 `release()` 操作，由此被唤醒进程的优先级可能高于当前进程。

②当前进程刚刚完成 `create()` 操作，新创建进程的优先级可能高于当前进程。

在以下两种情况下，新挑选的进程必须剥夺当前进程的运行权：

①当前进程刚刚完成 `request()` 操作，并且申请的资源 `timeout` 则当前进程的状态就改为“阻塞”；或者由于分时运行进程的需要，调度程序被一个 `timeout` 操作调用运行。在 `timeout` 操作中当前进程的状态改为“就绪”。在上述情况中，当前进程的状态都不是“运行”。所以当前进程必须停止运行，此时就绪队列中最高优先级的进程 p 得到执行。

②当进程刚刚完成 `destroy` 操作，进程自己删除自身，它的 `PCB` 表不再存在。此时调度程序被执行，从就绪队列中选出最高优先级的进程 p ，并令其执行。

剥夺操作包括以下工作：将选中的最高优先级进程 p 的状态改为“运行”。如果当前进程依然存在且没有阻塞，则将其状态改为“就绪”，以便随后能得到执行。最后，进行上下文切换，保留当前 `CPU` 的各个寄存器值，放入 `PCB` 表。装入中选进程 p 的寄存器值。

本实现方案没有对实际的 CPU 进行访问来保存或恢复寄存器的值，因此上下文切换的任务只是将正在运行进程的名字显示在终端屏幕上。从这一点可以认为，用户终端屏幕开始扮演当前运行进程功能的角色。

5. 用户 shell 界面。

为 RCB 试和演示管理器的 CPU 能，本方案设计开发一个 shell 界面，它可以重复接受终端输入的命令，唤醒管理器执行相应的功能，并在屏幕上显示结果。

使用上述系统，终端就能展示当前进程。只要输入一个命令，就中断当前程序的执行，shell 界面调用进程资源管理器中的函数 F，并传递终端输入的参数。该函数执行后将改变 PCB、RCB 及其他数据结构中的信息。当调度程序执行时，决定下一个要运行的进程，并改变其状态值。保存当前进程的 CPU 各寄存器值（虚拟 CPU），然后恢复中选进程的值。调度程序将系统状态信息显示在屏幕上，提示下一步操作。特别地，它始终提示正在运行的进程是什么，即终端和键盘正在为哪个进程服务。另外，函数 F 也可能返回一个错误码，shell 也将它显示在屏幕上。

shell 命令的语法格式规定如下：

命令名 参数

例如，执行命令行“cr A 1”时将调用对应的管理器函数 create(A, 1)，即创建一个名为 A、优先级为 1 的进程。同理，命令“rq R”将调用函数 request(R)执行。

以下显示说明 shell 界面的交互内容（假定进程 A 的优先级为 1，并且正在运行）。由“*”开始的行视为 shell 的输出结果。提示符“>”后面是提示用户输入的命令。

```
.....
* process A is running
> cr B 2
* process B is running
> cr C 1
* process B is running
> re q R1
* process B is blocked; process A is running
.....
```

6. 进程及资源管理器的升级版。

可对上述基本型进程功能资源管理器进行功能扩展，使管理器能够处理时钟到时中断和 I/O 处理完成中断。

(1)相对时钟到时中断。假设系统提供一个硬件时钟。周期性产生一个时钟到时中断。引发调用函数 `timeout()` 的执行。

(2) 110 处理完成中断。使用名为 IO 的资源表示所有的 I/O 设备。该资源的 RCB 由以下两部分组成：

IO

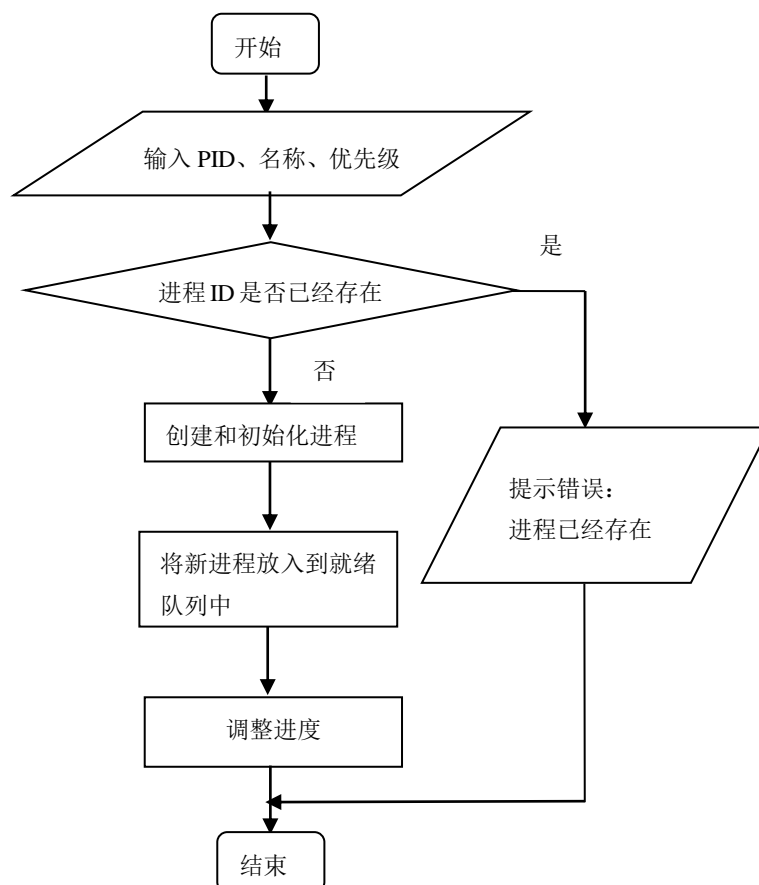
Waiting_list

(3)扩展 shell。显示当前运行进程，并添加一个系统调用 `request_100`。终端也能表示硬件，用户能够模拟两种类型的中断：时钟到时、I/O 完成处理。为了实现以上功能，必须添加新的 shell 命令，调用以下三个系统调用：`request_IO()`，`IO_competion`，`timeout()`。

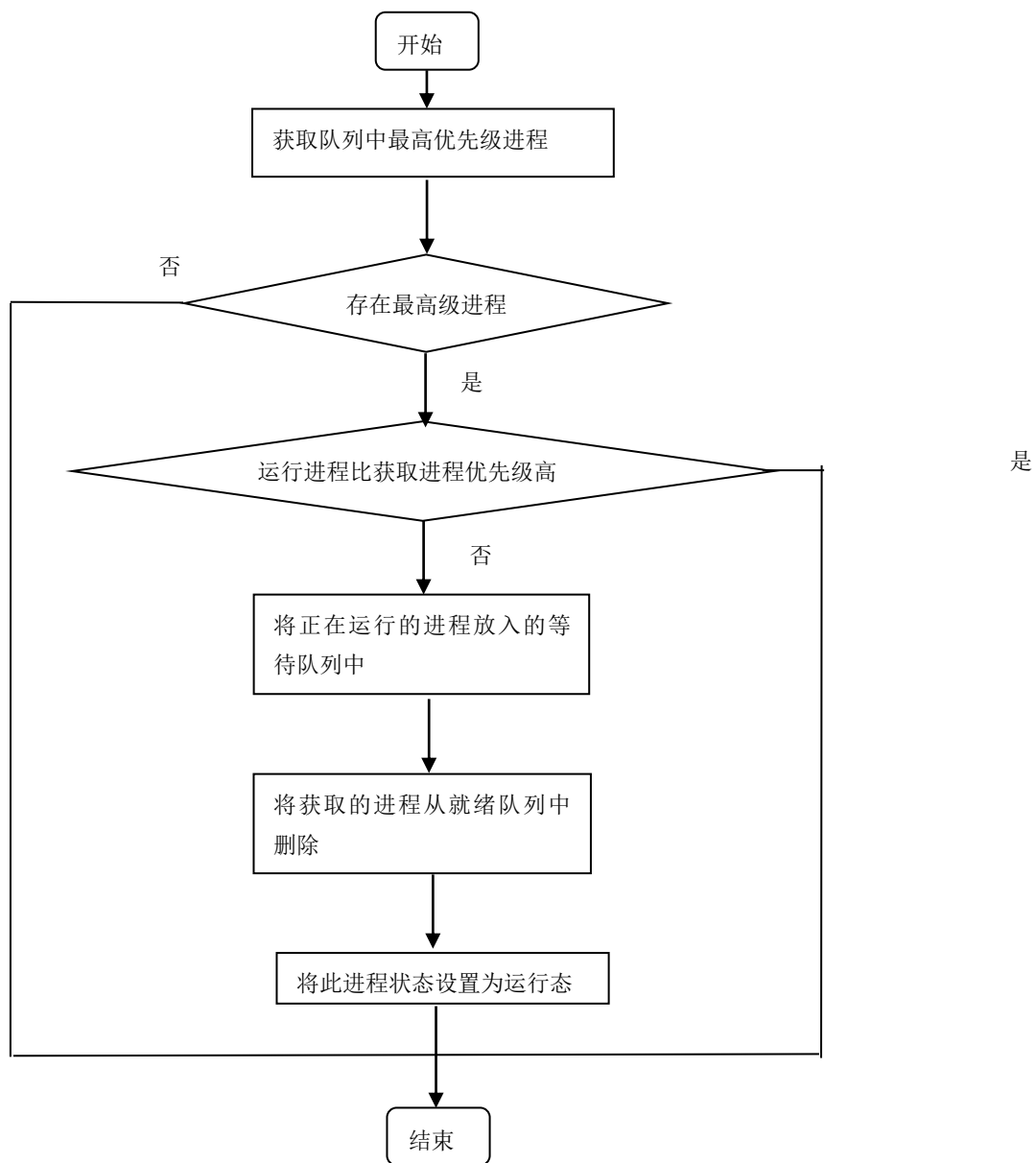
以下部分由学生填写：

1. 程序流程图

(1) 初始化进程流程图

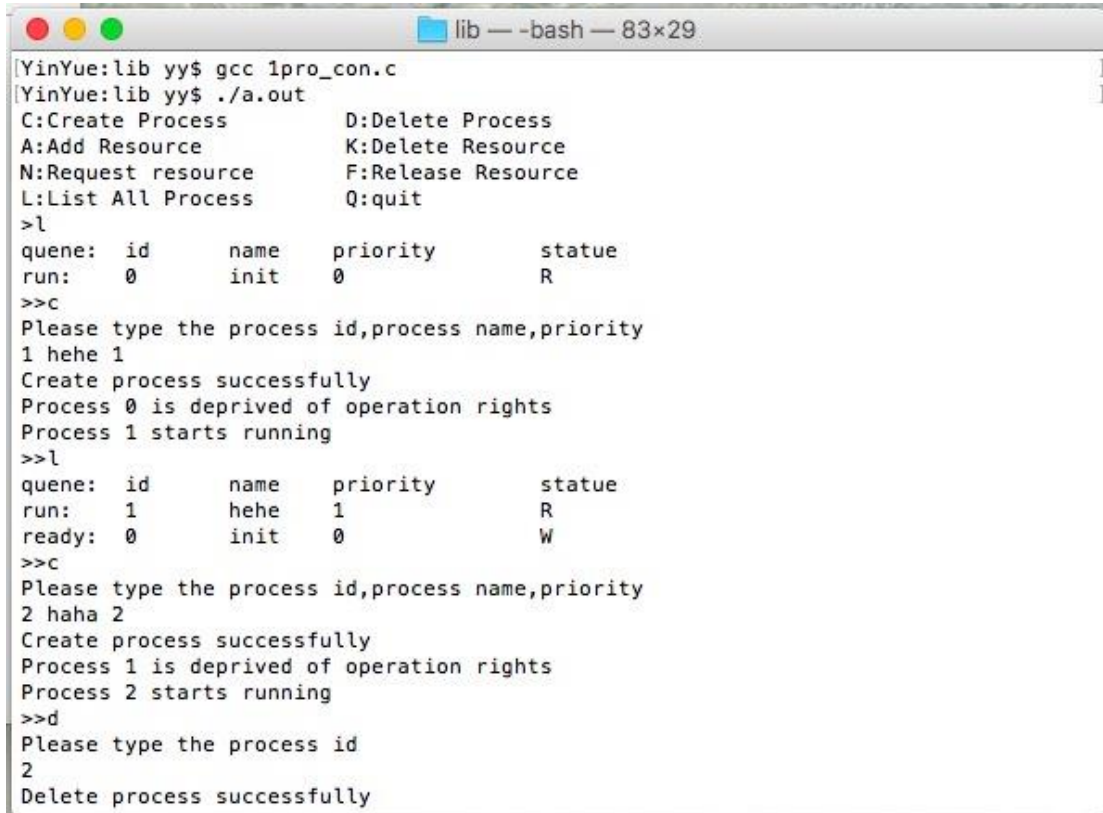


(2) 采用优先级策略的进程调度流程图。



2. 实验结果

(1) 手动调度进程运行结果图。



```
[YinYue:lib yy$ gcc 1pro_con.c
[YinYue:lib yy$ ./a.out
C:Create Process          D:Delete Process
A:Add Resource            K:Delete Resource
N:Request resource        F:Release Resource
L:List All Process        Q:quit
>l
queue: id      name  priority    statue
run:    0      init   0           R
>>c
Please type the process id,process name,priority
1 hehe 1
Create process successfully
Process 0 is deprived of operation rights
Process 1 starts running
>>l
queue: id      name  priority    statue
run:    1      hehe   1           R
ready:  0      init   0           W
>>c
Please type the process id,process name,priority
2 haha 2
Create process successfully
Process 1 is deprived of operation rights
Process 2 starts running
>>d
Please type the process id
2
Delete process successfully
```

```
lib — -bash — 83x36
C:Create Process      D:Delete Process
A:Add Resource        K:Delete Resource
N:Request resource    F:Release Resource
L>List All Process    Q:quit
>c
Please type the process id,process name,priority
1 hehe 1
Create process successfully
Process 0 is deprived of operation rights
Process 1 starts running
>>a
Please type the rid
1
Add resource success successfully
>>r
Please type the request pid with process rid
1 1
Process 1 request resource 1 successfully
>>c
Please type the process id,process name,priority
2 haha 2
Create process successfully
Process 1 is deprived of operation rights
Process 2 starts running
>>r
Please type the request pid with process rid
2 1
Process 2 is blocked
>>f
Please type the release rid with process pid
1 1
Process 1 free resource 1 successfully
Blocked process 2 get resource
Blocked process 2 enter ready quene
Process 1 is deprived of operation rights
Process 2 start running
```

(2) 最高优先级+先来先服务算法调度进程运行结果图。

```
lib - bash - 112x49
C:Create Process      D:Delete Process
A:Add Resource        K:Delete Resource
N:Request resource    F:Release Resource
L>List All Process    Q:quit
>c
Please type the process id,process name,priority
1 hehe 1
Create process successfully
Process 0 is deprived of operation rights
Process 1 starts running
>>c
Please type the process id,process name,priority
2 haha 2
Create process successfully
Process 1 is deprived of operation rights
Process 2 starts running
>>c
Please type the process id,process name,priority
3 xixi 3
Create process successfully
Process 2 is deprived of operation rights
Process 3 starts running
>>l
queue: id      name  priority  statue
run:    3      xixi   3          R
ready:  2      haha   2          W
ready:  1      hehe   1          W
ready:  0      init   0          W
>>c
Please type the process id,process name,priority
4 heihei 3
Create process successfully
>>l
queue: id      name  priority  statue
run:    3      xixi   3          R
ready:  4      heihei 3          W
ready:  2      haha   2          W
ready:  1      hehe   1          W
ready:  0      init   0          W
>>d
Please type the process id
3
Delete process successfully
>>l
queue: id      name  priority  statue
run:    4      heihei 3          R
ready:  2      haha   2          W
ready:  1      hehe   1          W
ready:  0      init   0          W
```

3. 结果分析

(1) 手动调度进程运行分析。

1. 创建进程 ID 为 1，名称为 hehe，优先级为 1，此进程优先级大于正在运行的进程 ID 为 0，名称为 init，优先级为 0，因此新进程为 1，名称为 hehe，优先级为 1 剥夺执行权开始运行
2. 创建进程 ID 为 1，名称为 hehe，优先级为 1，创建 PID 为 1 的资源，进程 ID 为 1 的进程申请 PID 为 1 的资源，PID 为 1 的资源为空闲状态，因此进程 1 获得 PID 为 1 的资源。
创建进程 ID 为 2，名称为 haha，优先级为 2，进程 ID 为 2 的进程申请 PID 为 1 的资源，PID 为 1 的资源为占用状态，因此 ID 为 2 的进程进入 PID 为 1 的资源的阻塞队列，ID 为 2 的进程被阻塞，ID 为 1 的进程仍在执行。
ID 为 1 的进程释放资源 1，被阻塞的 ID 为 2 的进程获取 PID 为 1 的资源，因此 ID 为 2 的进程进入就绪状态，ID 为 2 的进程开始执行。

(2) 最高优先级+先来先服务算法调度进程分析。

3. 创建进程 ID 为 1，名称为 hehe，优先级为 1，此进程优先级大于正在运行的进程 ID 为 0，名称为 init，优先级为 0，因此新进程 ID 为 1，名称为 hehe，优先级为 1 剥夺执行权开始运行。

创建进程 ID 为 2，名称为 haha，优先级为 2，此进程优先级大于正在运行的进程 ID 为 1，名称为 hehe，优先级为 1，因此新进程 ID 为 2，名称为 haha，优先级为 2 剥夺执行权开始运行。

创建进程 ID 为 3，名称为 xixi，优先级为 3，此进程优先级大于正在运行的进程 ID 为 2，名称为 haha，优先级为 2，因此新进程 ID 为 3，名称为 xixi，优先级为 3 剥夺执行权开始运行。

创建进程 ID 为 4，名称为 heihei，优先级为 3，此进程和正在运行的进程优先级相等，因此先来先运行，新创建的进程在就绪队列里等待被执行，正在运行的进程继续运行。

删除 ID 为 3 的进程，此进程正在运行，删除后正在运行的进程为空，因此就绪队列中最高优先级的进程开始运行，ID 为 4 的进程开始运行，其余就绪队列中其余进程继续等待。

