编号：_____39061416____
版本：_____1.0_____

北京航空航天大學
BEIHANG UNIVERSITY

Project： 作业调度

编写	黄建宇	2012.05.01
复查	黄建宇	2012.05.01
批准	黄建宇	2012.05.01

目录

1 小组成员

每个人的贡献大小以此排名为依据，靠前的为贡献较大者
39061416 黄建宇

2 实验目的与总体结构

了解实验的目的，对于作业的整体设计说明，要求思路清晰，表达明确

2.1 实验目的

- 理解操作系统中调度的概念与调度策略。
- 学习 Linux 系统中进程控制以及进程间通信的概念与方法。
- 理解并掌握几种常用的调度算法，能分析各算法的特征和优劣。

2.2 需求说明

本实验要求实现一个作业调度程序，通过该程序可以完成作业的入队、出队、查看和调度。具体要求如下：
1.基本要求：
（1）实现作业调度程序scheduler，负责整个系统的运行。
这是一个无限循环运行的进程，其任务是响应作业的入队、出队以及状态查看请求，采用适当的算法调度各作业运行。
（2）实现作业入队命令。
格式
enq [-p num] e_file [args]
参数说明
-p num：设定作业的初始优先级，默认值为0。num 为优先级的值，范围为0～3。
e_file：启动作业执行的可执行文件（以□/□开始的绝对路径名）。
args：e_file 的运行参数。
用户通过该命令给scheduler 发送入队请求，将作业提交给系统运行。每一个作业提交以后，若创建成功，scheduler 都将为其分配一个唯一标识jid。scheduler 调度程序为每个作业创建一个进程，并将其状态置为READY，然后放入就绪队列中，打印作业信息。
（3）实现作业出队命令。
格式
deq jid
参数说明
jid：由scheduler 分配的作业号。
用户通过该命令给scheduler 发送出队请求，scheduler 将使该作业出队，然后清除相关的数据结构。若该作业正在运行，则需先终止其运行。每个用户都只能杀掉（kill）自己提交的作业。
（4）实现作业状态查看命令。

格式

stat

在标准输出上打印出就绪队列中各作业的信息。状态信息应该包括：

作业的jid；

- 作业提交者用户名；
- 作业执行的时间；
- 在就绪队列中的等待时间；
- 作业创建的时刻；
- 此时作业的状态（READY、RUNNING）。

（5）实现多级反馈的轮转调度算法。

每个作业有其动态的优先级，在用完分配的时间片后，可以被优先级更高的作业抢占运行。就绪队列中的进程等待时间越长，其优先级越高。每个作业都具有以下两种优先级：

- 初始优先级（initial priority）：在作业提交时指定，将保持不变，直至作业结束。
- 当前优先级（current priority）：由scheduler 调度更新，用以调度作业运行。scheduler 总是选择当前优先级最高的那个作业来运行。

作业当前优先级的更新主要取决于以下两种情况：

- 一个作业在就绪队列中等待了若干个时间片（如5 个），则将它的当前优先级加1（最高为3）。
- 若当前运行的作业时间片到，则中止该作业停运行（抢占式多任务），将其放入就绪队列中，它的当前优先级也恢复为初始优先级。

通过这样的反馈处理，使得每个作业都有执行的机会，避免了使低优先级的作业拖延而不能执行的情况发生。

出于简单的目的，假设只考虑作业的两种状态：

- READY：就绪状态，该作业在就绪队列（ready queue）中等待调度。
- RUNNING：运行状态，该作业正在运行。

2.扩展要求：

本实验程序给出了一种多级反馈的轮转算法的实现，要求学生对其性能进行分析，改进优先级的更新方式，从而实现更合理、高效的调度算法。

此外，实验中的显示作业状态命令（stat 命令）的实现是将信息直接输出在调度程序 scheduler 终端，这样的话当时间片较短时，显示出来的作业状态信息易被其它调度信息覆盖掉，不利于实验观察。因此，建议学生实现作业状态信息的反馈（一种实现方式是使用 FIFO 将作业状态信息传输给作业控制命令程序）。

2.3 设计说明

2.3.1 结构设计

➢ 程序逻辑设计

作业调度程序

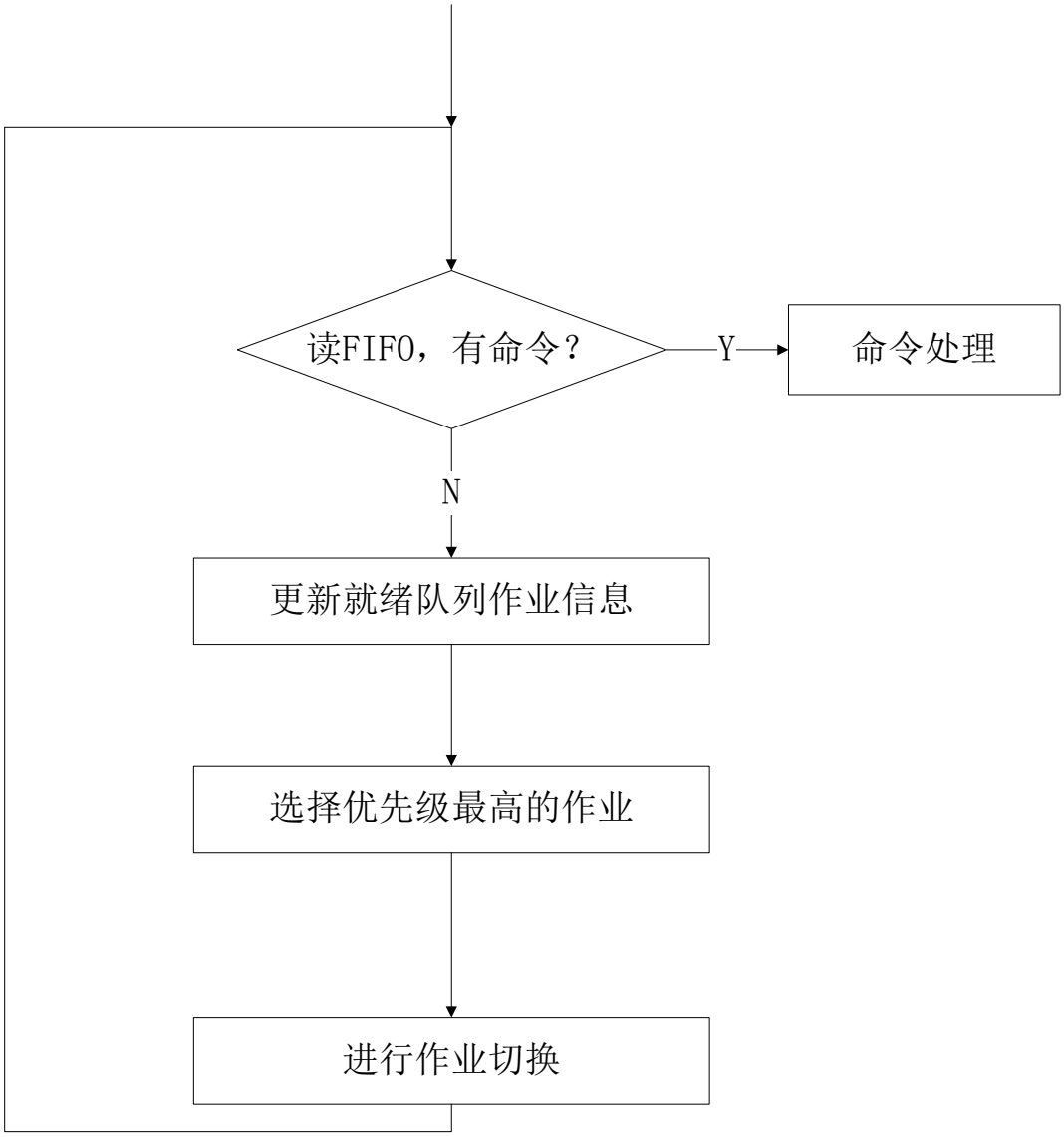

图2. 1作业调度程序程序逻辑流程图

作业控制命令

图2. 2作业控制命令程序逻辑图

2.3.2 功能设计

1.数据结构：

作业信息

```
struct jobnode
{
    int jid;//作业 ID
    int pid;//进程 ID
    int uid;//命令参数
    int defpri;//默认优先级
    int curpri;//当前优先级
    int runtime;//运行时间
    int waittime;//作业在等待队列中等待时间
    time_t create_time;//作业创建时间
    enum jobstate state;//作业状态
    enum cmdtype type;//命令类型（enq，deq，stat）
    char data[BUFLEN];//记录作业对应的命令名
};
```

就绪队列

```
struct queue
{
```

```
    struct jobnode job;
    struct queue *next;
};
```

作业状态

```
enum jobstate
{
    READY,RUNNING,DONE
};
```

命令类型

```
enum cmdtype
{
    ENQ=-1,DEQ=-2,STAT=-3
};
```

2.主要函数与接口说明

作业调度(scheduler.c)

```
void welcome();                          //欢迎界面
void getEnvPath(int len,char *buf);      //获取环境变量
void init();                             //初始化函数，获取环境参数，创建 FIFO 命名管道；
设置信号；初始化消息队列
void start();                            //时间设置函数，设置间隔定时器的时间值
void exit_scheduler();                   //退出处理函数，关闭 FIFO 命名管道
void sig_handler(int ,siginfo_t *,void *);//信号处理函数，对 sig 分类（SIGVTALRM,SIGCHLD）
处理
void childsig();                         //子函数变化（退出）时的信号处理函数
void scheduler();                        //关键的作业调度函数
void update();                           //优先级更新函数
void newjob();                           //读取 FIFO 命名管道，看有无进队列的命令读入
char * itoa(int);                        //数字转为字符
void do_enq(struct jobnode);             //ENQ 添加作业处理函数
void do_deq(struct jobnode);             //DEQ 删除作业处理函数
void do_stat(struct jobnode);            //STAT 显示当前作业处理函数
void selectjob();                        //作业切换时对之前作业状态的处理函数（分两种：
RUNNING,DONE）
void executejob();                       //作业切换时更改当前工作状态为 RUNNING,发出
SIGCONT 信号的执行函数
```

作业控制（enq.c deq.c stat.c）

```
void welcome();                          //欢迎界面
void help();                             //Usage 用法提示帮助
```

```
void transform(int argc);          //转换参数，将输入参数存入到 jobnode 节点中
char * itoa(int);                  //数字转为字符
void init();                       //初始化函数，创建 FIFO 命名管道
void writetofifo();                //将设置好参数的 jobnode 写入到 FIFO 中
void printtoscreen();              //将从作业调度函数输出结果的 FIFO 管道中读入的内
容打印到当前终端
void exitfree();                   //退出处理函数，释放内存
```

3.接口与调用关系设计

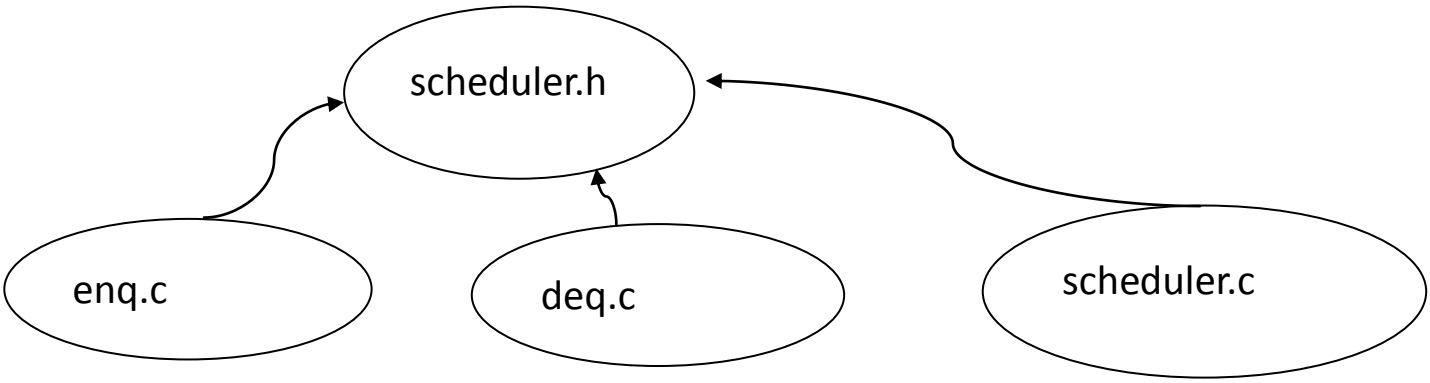

图2. 3 接口与调用关系图

4. 核心逻辑或功能代码

请见 3 实验内容中的 3.1 实现方法以及详细注释。

3 实验内容

详细说明实验的过程，实现方法及遇到的问题

3. 1实现方法

- 进程间通信

调度者会先建立一个接受命令的管道，每个用户均向这个管道写自己的命令。每个用户接受反馈信息的命令时另建立单独的管道来读取，调度者向每个用户建立的管道中写程序运行结果。每个用户都会创建三个管道，分别接收 enq、deq 和 stat 的反馈，这样才能避免信息混杂在一起。

- 互斥问题

由于进程通信的设计，需要互斥的地方只用调度者接受命令的管道。对于此，经过实验，发现每一次 write（fd，buff，size）是 buff 是一次写完，不同的 write 之间的 buff 不会交错写入管道。所以，每次向调度者发出命令时用一次 write 把命令写完，在命令中保存发出命

令者的信息，就可以保证不同用户不会同时向管道写信息导致的问题。不必采用文件锁的方法，简化了程序编写。

- scheduler

初始化：建立接收命令的管道，设置 10ms 定时信号和子进程信号的响应函数。然后每 10ms 产生定时的信号。

接受到子函数状态改变信号后，查看发信号的进程的状态，有正常终止、非正常终止和挂起三种，并在控制台输出。

接收到 10ms 的定时信号后，首先更新所有作业的状态，包括优先级、运行时间、等待时间。然后读入新的命令并执行（若没有，则跳过），命令包括三种：enq、deq、stat。然后挂起当前运行的命令，查找当前优先级最高的命令并执行。

enq 命令：首先获取新命令信息，包括用户标识符、优先级、要执行程序及参数，为作业标识符赋值。向作业链表中添加新的作业。建立子进程，子进程中 open 作业输出的 FIFO，并将标准输出重定向，然后让子进程执行新的作业。

deq 命令：首先获取作业标识符和用户标识符。open 向用户输出结果的 FIFO。查找作业链表，寻找作业。有三种情况：（1）找不到作业；（2）不是用户创建的作业，不能删除；（3）删除作业。根据情况向用户输出相应的信息。

stat 命令：open 向用户输出结果的 FIFO，遍历作业链表，向用户输出信息。

- enq

解析命令，获取优先级和要执行的程序，创建用户的 enqfifo，打开 scheduler 的 FIFO，向 scheduler 发送命令，用用户 FIFO 接收运行结果，并打印。

- deq

解析命令，获取作业号。创建用户的 deqfifo，打开 scheduler 的 fifo，向 scheduler 发送命令，用用户 FIFO 接收运行结果，并打印。

- stat

解析命令，获取优先级和要执行的程序，创建用户的 statfifo，打开 scheduler 的 FIFO，向 scheduler 发送命令，用用户 FIFO 接收运行结果，并打印。

3.2测试和使用说明

3.2.1 程序开发环境：

gcc、gdb

3.2.2 运行环境

unbuntu 9.04，linux 内核 2.6 以上版本

3.2.3 安装说明

1. 打开源码所在文件夹，make 编译。

2. 打开多个终端，分别运行 sudo ./scheduler、sudo ./enq *、sudo ./deq *、sudo ./stat 命令。具体实例请见 3.2.4 测试用例和运行结果分析:

3.2.4 测试用例和运行结果分析:

- **输入描述**

开启 6 个终端，输入命令：首先运行左上的./scheduler 等待 FIFO 输入。再运行右侧的 3 个终端分别输入 sudo ./enq ./a、sudo ./enq ./b、sudo ./enq ./c。最后在左下的终端输入 sudo ./stat 查看作业状态。

- **输出描述**

在下面的截图中有显示

- **运行结果**

基本测试

开启 6 个终端，输入命令如下（左边中间一个终端窗口留作后面的./deq 测试用），首先运行左上的./scheduler 等待 FIFO 输入。再运行右侧的 3 个终端分别输入 sudo ./enq ./a、sudo ./enq ./b、sudo ./enq ./c。最后在左下的终端输入 sudo ./stat 查看作业状态。

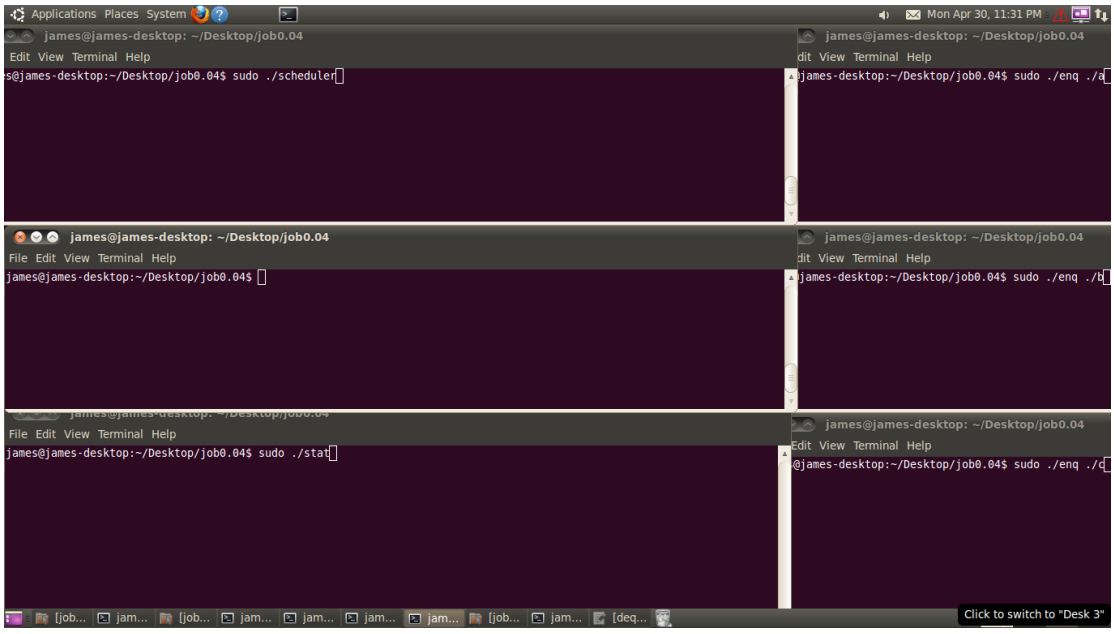

图3. 1 输入命令

运行左上的./scheduler 等待 FIFO 输入。再运行右侧的 3 个终端分别输入 sudo ./enq ./a、sudo ./enq ./b、sudo ./enq ./c 后的截图：

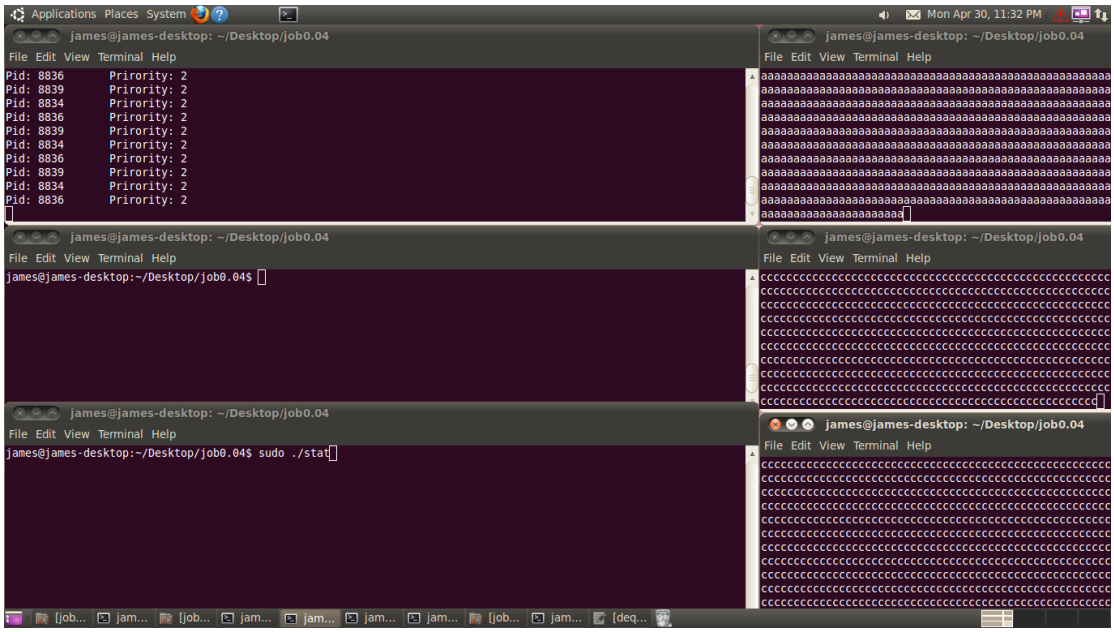

图3. 2 输入./scheduler 和./enq {./a|./b|./c}运行截图

在左下的终端输入 sudo ./stat 查看作业状态的截图：

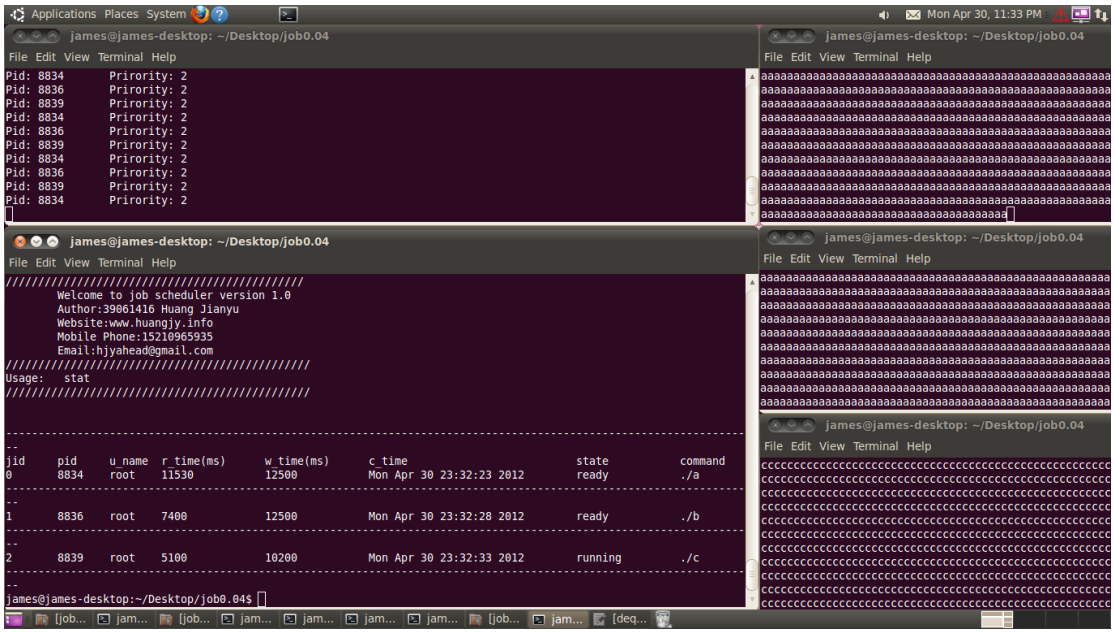

图3. 3 输入./stat运行截图

在左侧中间窗口输入 sudo ./deq 0（删除 0 号作业节点：./a）后的截图

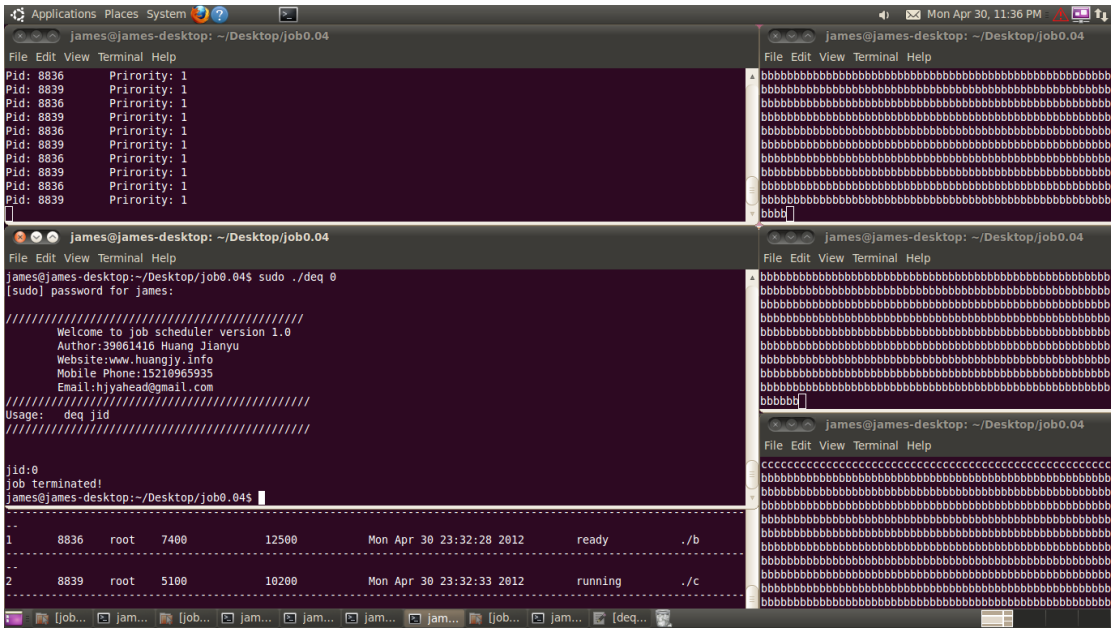

图3. 4 输入./deq 0运行截图

再在左侧中间窗口输入 sudo ./deq 1（删除 1 号作业节点：./b）后的截图

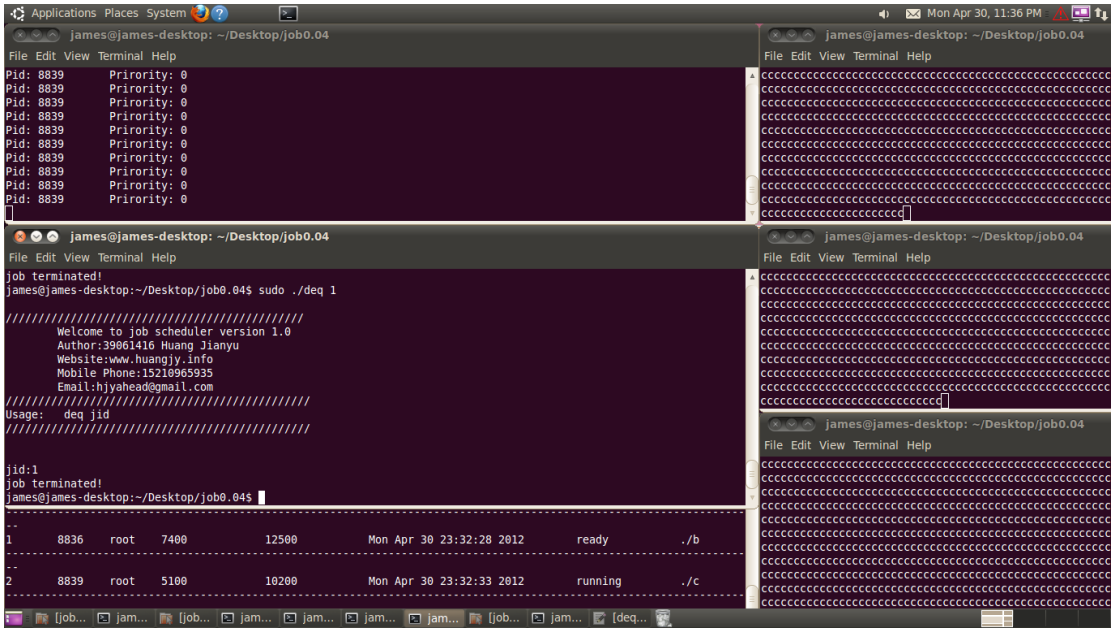

图3. 5输入./deq 1运行截图

最后在左侧中间窗口输入 sudo ./deq 2（删除 2 号作业节点：./c）后的截图

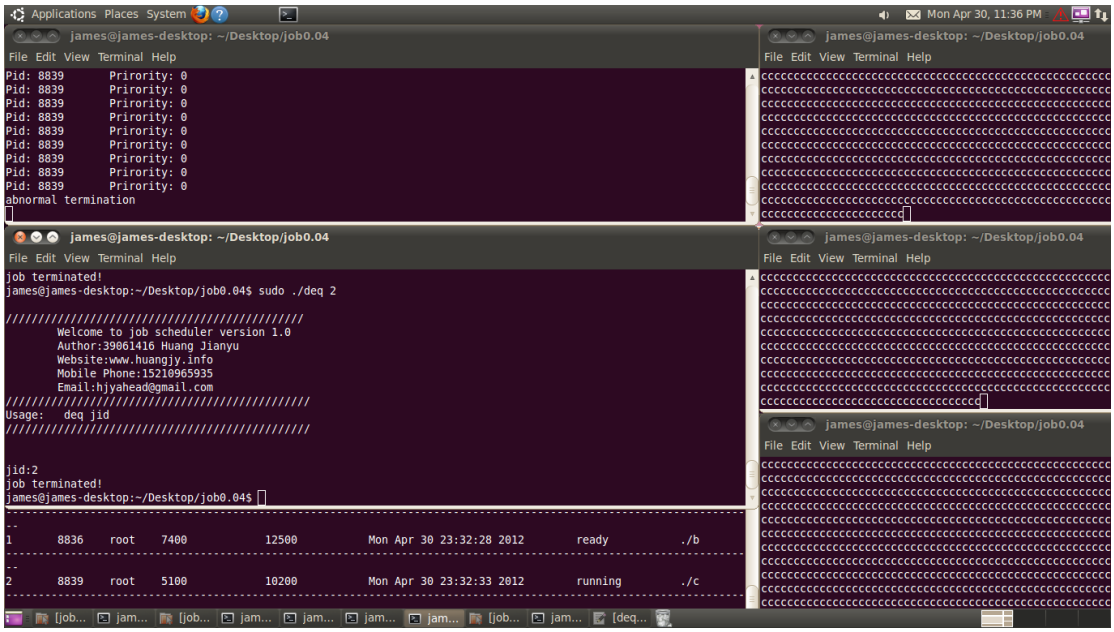

图3. 6输入./deq 2运行截图

最终输入 sudo ./stat 查看状态：

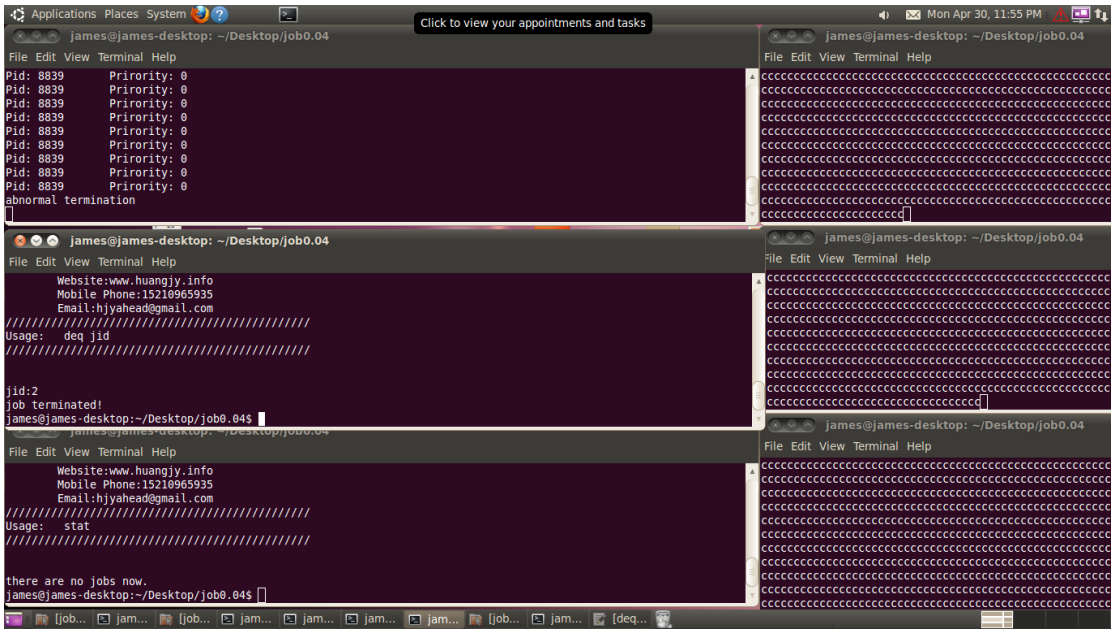

图3. 7 最终截图

调度算法测试

开启 3 个终端，输入命令如下：

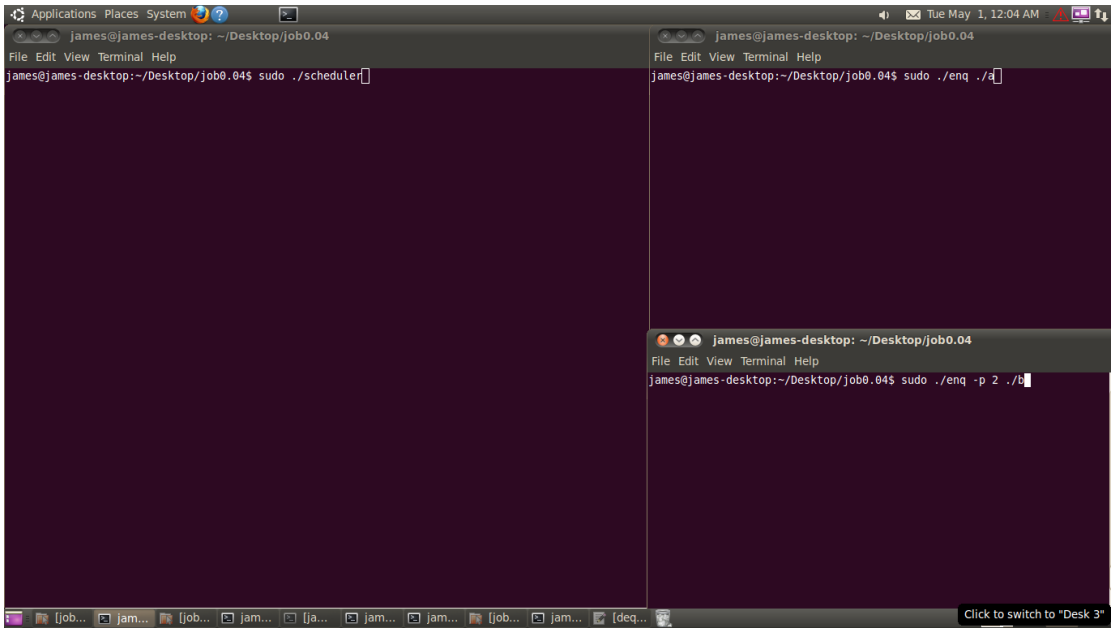

图3. 8 调度算法命令截图

运行结果：

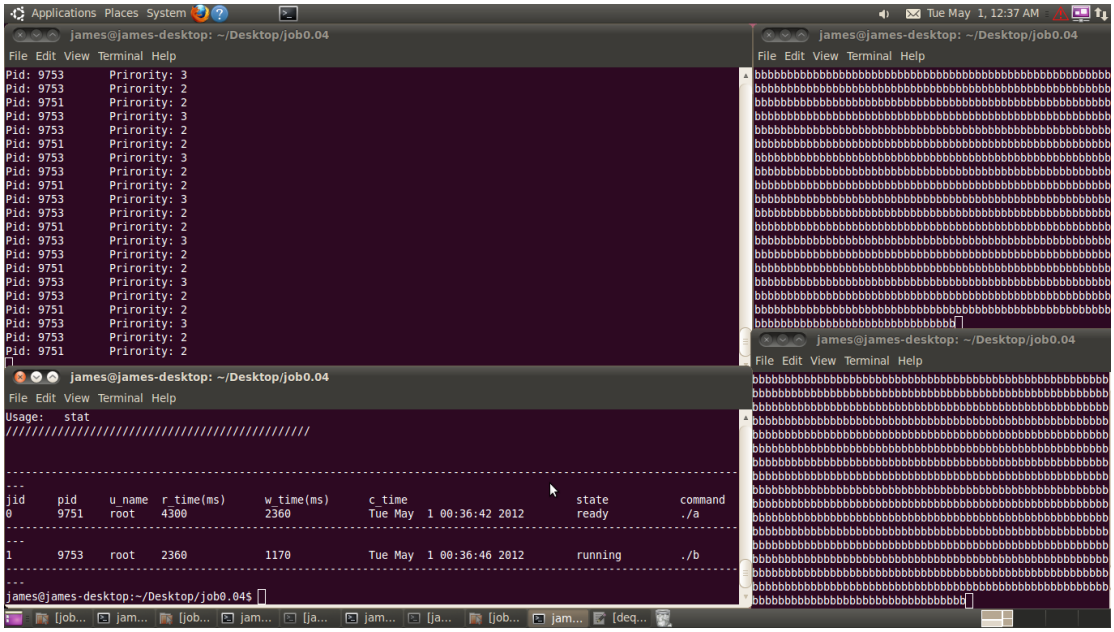

图3. 9 调度算法运行结果

➢ 结果分析

1. 从图 3. 1 到图 3. 7 可以看出，本程序实现了基本要求：实现作业调度程序 scheduler，负责整个系统的运行，实现作业入队命令，实现作业出队命令，实现作业状态查看命令，实现多级反馈的轮转调度算法。
2. 从图 3. 8 到图 3. 9 可以看出，本实验程序给出了一种多级反馈的轮转算法的实现，性能高效，实现扩展要求 1。（图 3. 9 中，每当 pid 为 9753 的进程执行 2 次时，pid 为 9751 的进程执行 1 次，因为 pid 为 9753 的./b 初始优先级高）
3. 从图 3. 3 和图 3. 9 可以看出，显示作业状态命令（stat 命令）实现作业状态信息的反馈（程序里实现是使用 FIFO 将作业状态信息传输给作业控制命令程序），完成了扩展要

求 2。

4 每个人的工作与会议记录

每次实验至少要开 4 次小组会议,每次会议都要记录以下内容:
a 说明每一位组员前一阶段完成的具体工作,是否按时按量完成任务。
b 说明每一位组员下一阶段需要完成的工作。
c 其他会议内容(如讨论的问题以及解决方案等)

4.1 会议时间表与会议记录

2012 年 4 月 16 日	作业调度设计	黄建宇
2012 年 4 月 20 日	作业调度文档书写	黄建宇
2012 年 4 月 22 日	作业调度文档审查，修改	黄建宇
2012 年 4 月 28-30 日	作业调度程序编写	黄建宇
2012 年 5 月 1-3 日	调试，debug，完善文档	黄建宇

每次会议都基本完成上次会议所计划内容，并规划下次完成进度。

5 其他说明

5.1 组内成员任务分工说明

(注明各个组员的工作量比例)
由于本组只有 1 人，工作量 100%。

5.2 实验完成部分说明

基本都已完成。
实现所有扩展要求：

- 本实验程序给出了一种多级反馈的轮转算法的实现，要求学生对其性能进行分析，改进优先级的更新方式，从而实现更合理、高效的调度算法。
- 此外，实验中的显示作业状态命令（stat 命令）的实现是将信息直接输出在调度程序 scheduler 终端，这样的话当时间片较短时，显示出来的作业状态信息易被其它调度信息覆盖掉，不利于实验观察。因此，建议学生实现作业状态信息的反馈（一种实现方式是使用 FIFO 将作业状态信息传输给作业控制命令程序）。

6 程序清单

makefile	makefile 编译文件
scheduler.h	全局数据结构保存
scheduler.c	作业调度程序
enq.c	作业控制命令：作业入队
deq.c	作业控制命令：作业出队
stat.c	作业控制命令：在标准输出上打印就绪队列中各作业的信息
cmdenv.conf	环境变量路径设置
a.c	测试程序 1，循环打印 a
b.c	测试程序 2，循环打印 b
c.c	测试程序 3，循环打印 c

7 实验心得

7.1 心得：

我感受到理论与实践之间的差距，在课堂上我感觉作业调度很简单，但到了 OS 课程设计实际模拟时，却遇到一个个困难。通过解决这一个个困难，我感觉我对课堂上的理论部分有了更深刻的理解。

特别感谢王雷老师精彩的授课，感谢助教辛苦的讲解。最终我高效率高质量的完成的实验的要求。

7.2 建议：

建议老师多给一些相关的资料，比如初步介绍一下本实验要用到的相关函数。我看了《Unix 高级环境编程》才真正理解 FIFO（命名管道）的相关知识。