操作系统实验报告：实验二

组长：姬站华 学号：13061084

组员：衣德良 学号：13061070

迟禄 学号：13061084

陈世涉猎 学号：13061089

# 1需求说明

## 1.1基本要求和提高要求

基本要求：完成10个调试任务

提高要求：实现多级反馈轮转调度算法。

## 1.2完成情况

基本全部实现。

# 2设计说明

## 2.1流程示意图

## 2.2所使用的系统调用的列表

仅以几个较为重要的系统调用为例进行说明：

1. setitimer():

可以为调度程序进行计时，每当时间片到期时就向进程发出信号，提醒调度程序进行调度。

1. sigaction():

查询或设置信号处理方式。该程序中，此函数的用法为：

sigaction(SIGCHLD,&newact,&oldact1);

sigaction(SIGVTALRM,&newact,&oldact2);

SIGCHLD:在子进程状态发生变化时产生并发送给父进程。

SIGVTALRM:实际时间报警时钟信号

这两行代码产生的效果是：只要子进程状态发生变化或定时器时间到了，都会调用一次newact.sa\_sigaction,即本程序中的sig\_handler().

1. kill():

发送信号。本程序中出现的几种类型信号有：

SIGKILL:杀死进程

SIGCONT:继续进程

SIGSTOP:挂起进程

1. mkfifo():

创建FIFO文件，用于进程间的通信。一个终断通过运行deq/enq/stat，创建FIFO文件并将相应指令写入，另一个终端的job通过读取FIFO文件调用相应的函数。

## 2.3调试任务实现说明：

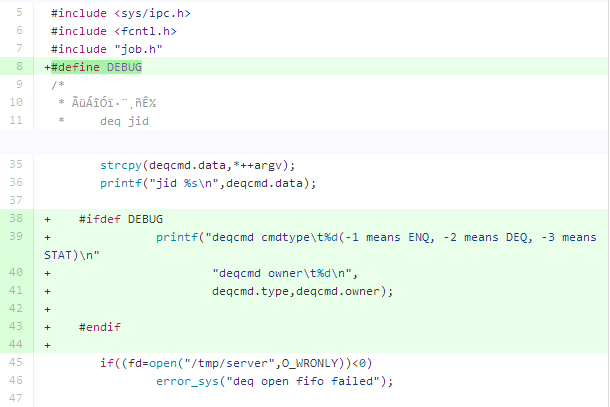
由于任务一至四PPT中已详细给出，在此就不赘述了。另外，由于DEBUG信息过多，输出不方便，调试任务的运行结果就不截图了。

任务五：DEQ和STAT命令调试：

在stat.c中加入：



在deq.c中加入：



任务六：updateall函数调试：

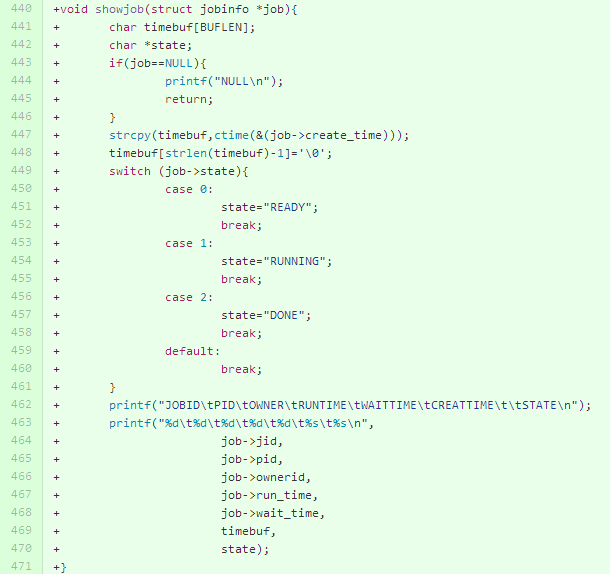
在scheduer函数的updateall()前后各加一个do\_stat函数（即利用stat的显示功能来显示当前的作业情况）

任务七：命令调试：

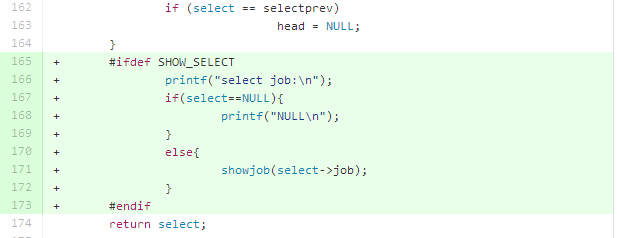
在scheduler函数的switch…case语句中的每个指令前后，各加一个do\_stat函数。

任务八：jobselect函数调试：

添加新的函数，以显示作业信息：



然后，在jobselect函数中添加相应的代码：

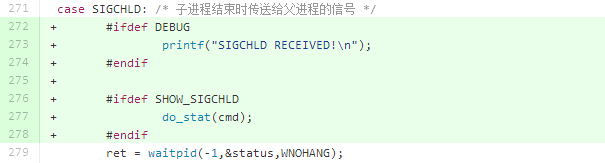


任务九：jobswitch函数调试：

在scheduler函数的jobswitch前后调用do\_stat函数。

任务十：进程运行完成调试：

在sighandler函数中，加入：



同时将Demo.c进行适当修改，为其设置一个停止条件，本小组改为：循环20次结束。

## 2.4提高要求实现说明

首先说明一下本小组对多级反馈轮转调度算法的理解：

多级：共1至3三个等级的队列，默认为1级。

反馈：每个等待时间到达10s的作业优先级都会升一级，但最高级仍为3级。升级后的作业获得时间片运行结束后，会回到初始优先级。

轮转：同优先级的作业轮转调度，不同优先级运行的时间片不同，最高优先级队列的轮转时间为1s，次高优先级为2s，最低优先级为5s。轮转时间仅仅以当前优先级为参考，与最初优先级无关。

抢占式：当新加入的作业优先级比正在运行的作业优先级大时，立刻获得时间片，否则放入等待作业的队尾。

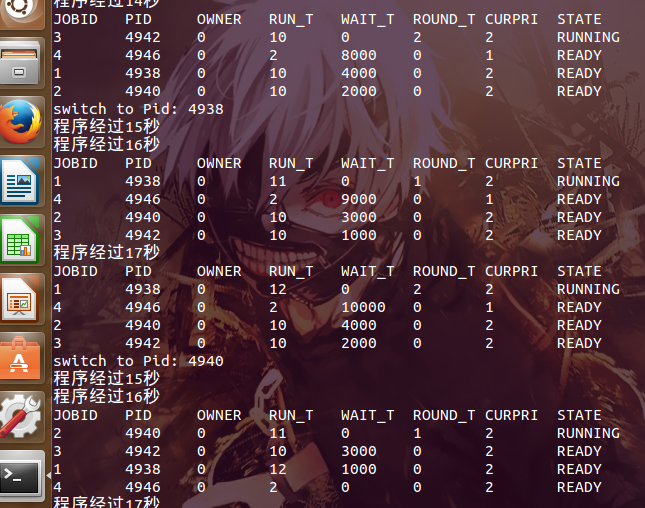
本小组对该算法的实现：

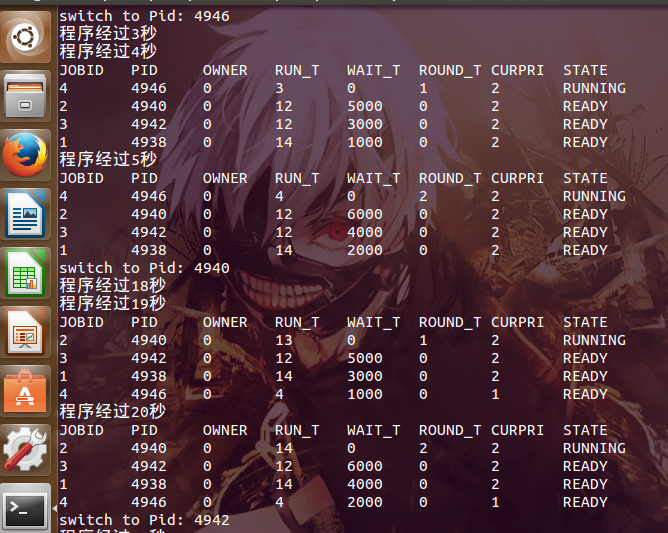
虽然有三个级别的队列，但本小组并没有真正创建三个队列，而是放在同一个队列中，利用jobinfo结构中的优先级进行区分。每次jobselect的时候，从中选择最先出现的最高优先级作业进行运行（效果上与选择高优先级队列的队首作业运行相同）。这样做就会面临一个问题：即无法通过head指针直接判断该优先级队列是否为空，所以，每次切换作业的时候，需要遍历一遍队列，查看该优先级作业是否全部运行结束，以决定是否开始运行低优先级的作业。同时，在updateall的时候，若有作业升级，需要将其重新加至队尾（在本程序中体现为touchjob函数）。

对轮转时间的实现：在jobinfo中添加新的变量：int round\_time，以此来记录此次运行的时间（run\_time是总共运行时间），与优先级的时间片进行比较，以判断是否进行作业切换。

运行结果截图（为方便演示，每次更新作业队列的时候均打印一次作业信息）：

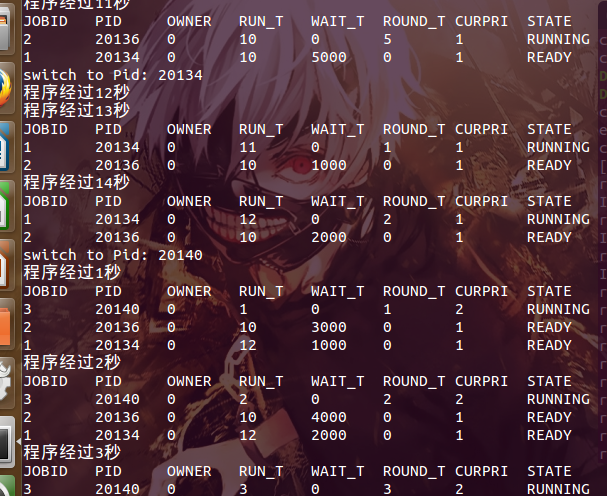
反馈：



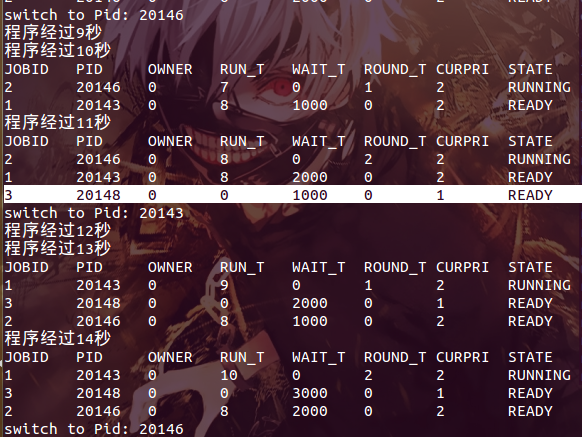


PID:4946作业在等待了10s后，从1级升为2级，并放在队尾。运行结束后回到1级，放在队尾。同时也体现了：高优先级作业未运行结束时，低优先级作业一直处于等待状态。

抢占：



两个低优先级的作业正在交替运行着，当20134刚轮转2s时，被新加入的较高级作业20140抢占。



但当两个2级作业轮转时，新加入的1级作业并不抢占，而是放入队尾。

## 2.5 关于Bug

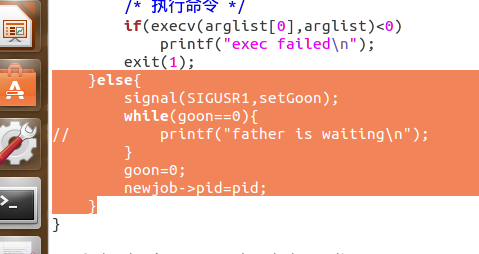
关于助教课上提到的bug，本小组在调试过程中也发现了，但解决方法不同。本小组采用了信号来解决此问题。

原因已经很清楚了，在此就不解释了，直接上解决方案：

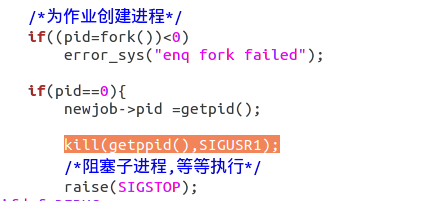
第一步：设置全局变量goon，初始值为0.

第二步：创建新的函数setGoon，作用是将goon置1.

第三步：修改do\_enq函数。在fork后的父进程中，注册自定义信号SIGUSR1，并令父进程等待子进程发出的信号：



第四步：令父进程等待后，只需要在子进程的恰当位置向父进程发送信号通知其可以运行即可：



当然，为了下一次重新使用goon变量，需要在父进程中令goon=0。

与助教的方法对比之后发现，助教大人的方法更加简洁。顿时感慨Linux系统调用之强大与自身之渺小啊、、、

# 3收获与感想

通过本次作业，不仅加深了本小组对作业调度的理解，比较了各种调度算法的不同及各自的优缺点，而且还提高了我们实践的能力，进一步了解了Linux是怎样具体实现作业调度的。同时也激发了我们的想象力，研究更合理、更高效的调度算法。

# 4 联系方式

写的有些匆忙，难免有不详细的地方。如有什么问题，[请与1820366459@qq.com](mailto:请与1820366459@qq.com)联系，望您谅解！