**海南大学网络空间学院**

**实 验 报 告**

**实验课程： 《操作系统原理及安全》实验\_\_\_\_\_\_\_\_\_\_\_**

**实验名称： 实验二 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**学号：** 20213006839  **姓名：** 甄五四\_\_\_\_

**专业班级：** 信息安全（密码学方向）理科实验班**指导教师：** 秦小立

**完成日期：** 2023 年 5 月 19 日

**成绩：**

**评阅：**

**教师签名：**

1. **实验目的**

1．加深对进程概念的理解，明确进程和程序的区别；

2．进一步认识并发执行的实质。

1. **实验任务**

1) 进程的创建

编写一段程序，使用系统调用fork()创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符；父进程显示字符“a”，子进程分别显示字符“b”和“c”。试观察记录屏幕上的显示结果，并分析原因。

2）进程的控制

修改已编写的程序，将每个进程的输出由单个字符改为一句话，再观察程序执行时屏幕上出现的现象，并分析其原因。如果在程序中使用系统调用lockf()来给每个进程加锁，可以实现进程之间的互斥，观察并分析出现的现象。

1. **实验环境**

VMware15.5 PRO + Ubuntu 20.04

1. **实验内容及步骤(题目、相应程序代码、运行结果)**

**1) 进程的创建**

#include <stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main( )

{

int p1,p2;

while((p1=fork( )) == -1); /\*创建子进程p1\*/

if (p1= =0) putchar('b');

else

{

while((p2=fork( )) = = -1); /\*创建子进程p2\*/

if(p2 ==0) putchar('c');

else putchar('a');

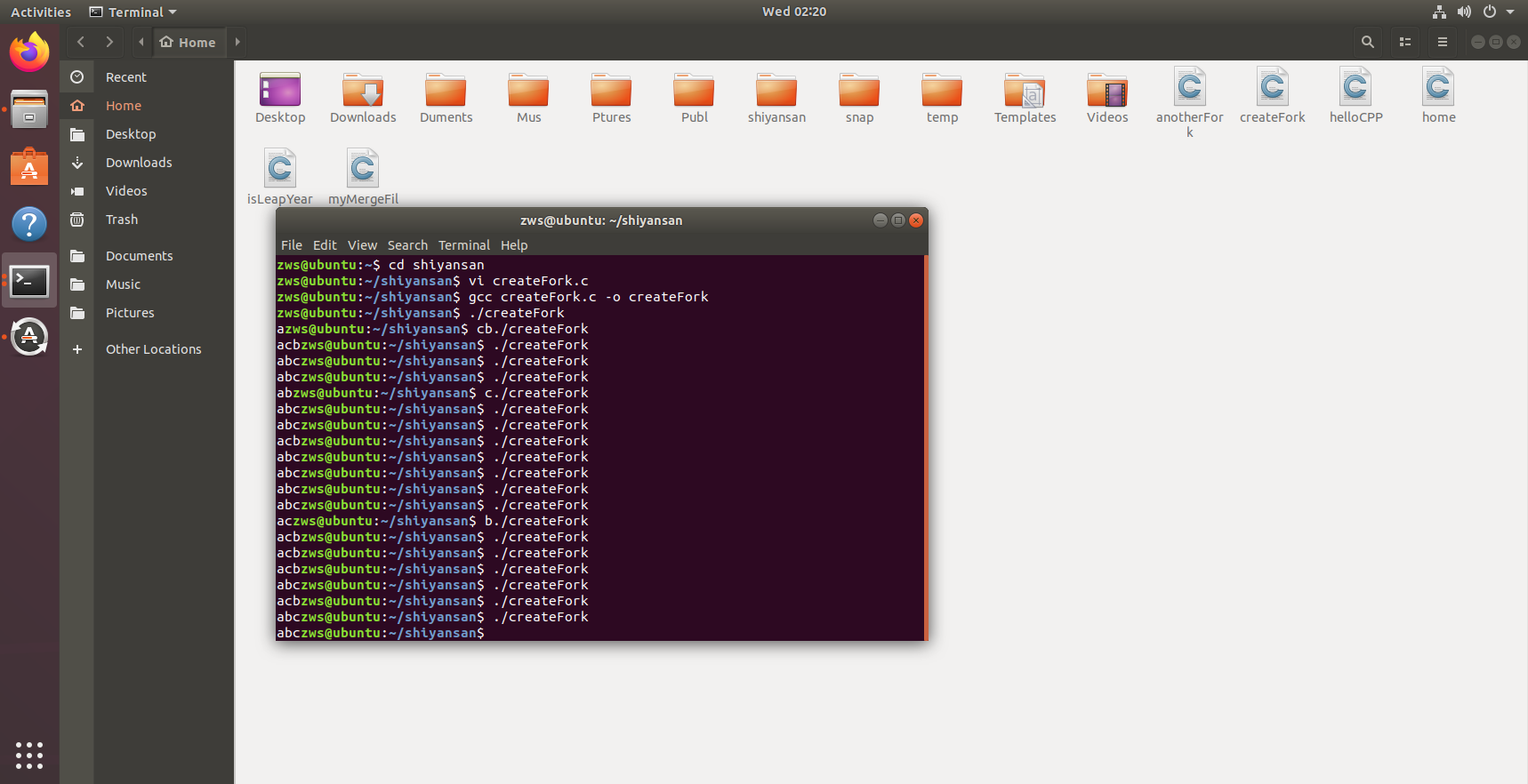
}

return 0;

}

运行结果：

acb



分析：从进程并发执行来看，输出bc，cb，bac，acb,abc等情况都有可能。

原因：fork()创建进程所需的时间要多于输出一个字符的时间，因此在主进程创建进程2的同时，进程1就输出了“b”，而进程2和主程序的输出次序是有随机性的，所以会出现上述结果。在输出结果acb中父进程先被调度，在标准输出中写入字符 ‘a’。然后 p1 进程得到调度，在标准输出中写入字符 ‘b’。最后 p2 进程得到调度，在标准输出中写入字符 ‘c’。

**2）进程的控制**

程序1：

#include <stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main( )

{

int p1,p2,i;

while((p1=fork( )) == -1); /\*创建子进程p1\*/

if (p1 ==0)

for(i=0;i<500;i++)

printf("child %d\n",i);

else

{

while((p2=fork( )) == -1); /\*创建子进程p2\*/

if(p2= =0)

for(i=0;i<500;i++)

printf("son %d\n",i);

else

for(i=0;i<500;i++)

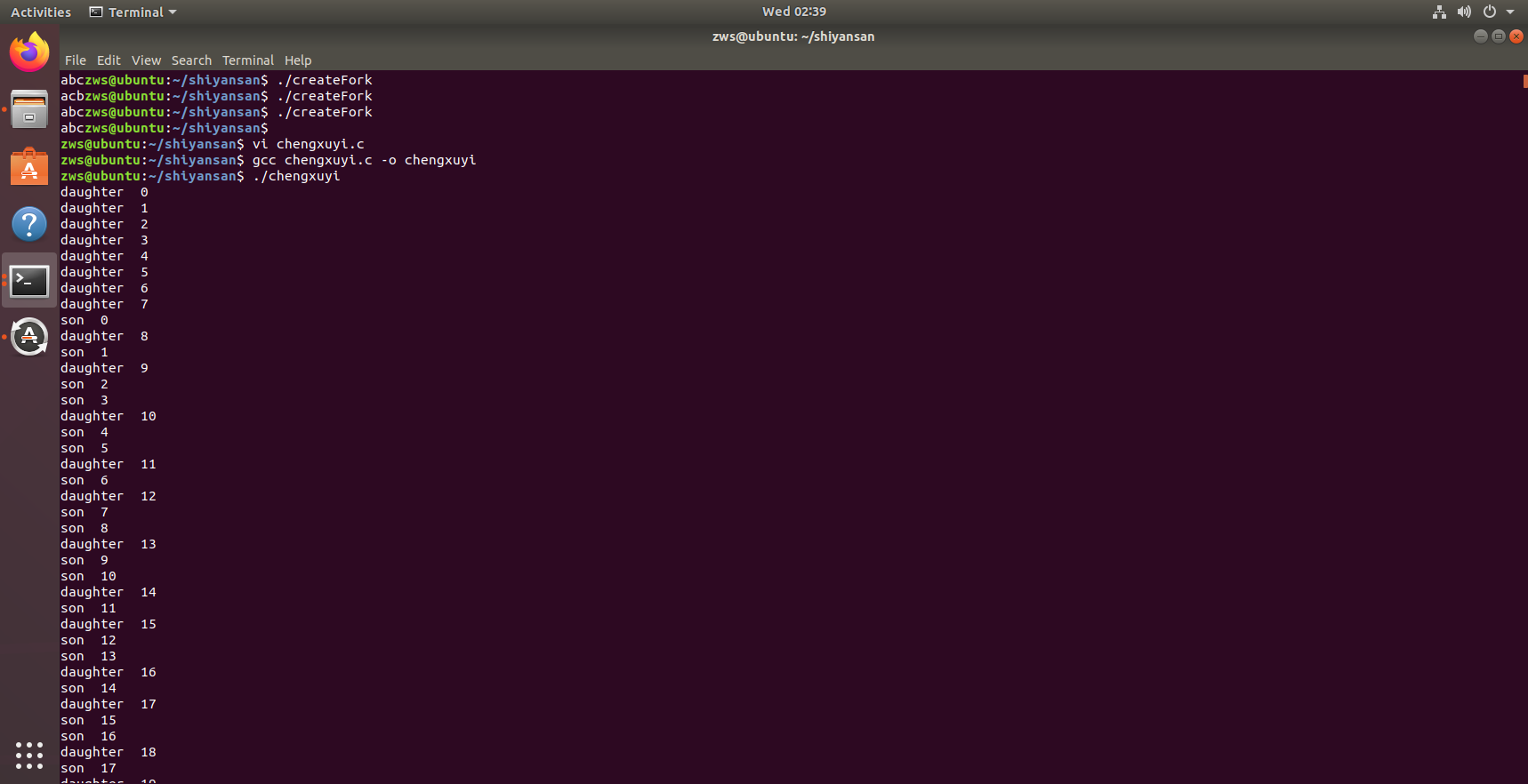
printf("daughter %d\n",i);

}

return 0;

}

运行结果：



分析：由于函数printf()输出的字符串之间不会被中断，因此，字符串内部的字符顺序输出时不变。但是，由于进程并发执行时的调度顺序和父子进程的抢占处理机问题，输出字符串的顺序和先后随着执行的不同而发生变化。这与打印单字符的结果相同。在程序启动时，父进程会通过调用 fork() 系统调用创建子进程 p1 和 p2。在父进程中执行第一个 for 循环语句输出 “daughter i” 语句时，子进程还没有开始执行，父进程就会多次执行输出语句,直到子进程 p1 和 p2 被创建成功之后，才会分别执行它们的代码段输出 “child i” 和 “son i”。

程序2：

#include <stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main( )

{

int p1，p2，i；

while ((p1=fork())==-1)；

if(p1==0)

{

lockf(1，1，0)；

for(i=0；i<500；i++) printf(“Child％d\n”，i)；

lockf(1，0，0)；

}

else

{

while((p2=fork())==-1)；

if(p2==0)

{

lockf(1，1，0);

for(i=0；i<500；i++) printf("son％d\n”, i);

lockf(1，0，0)；

}

else

{

lockf(1，1，0)；

for(i=0；i<500；i++) printf("daughter％d\n”，i)；

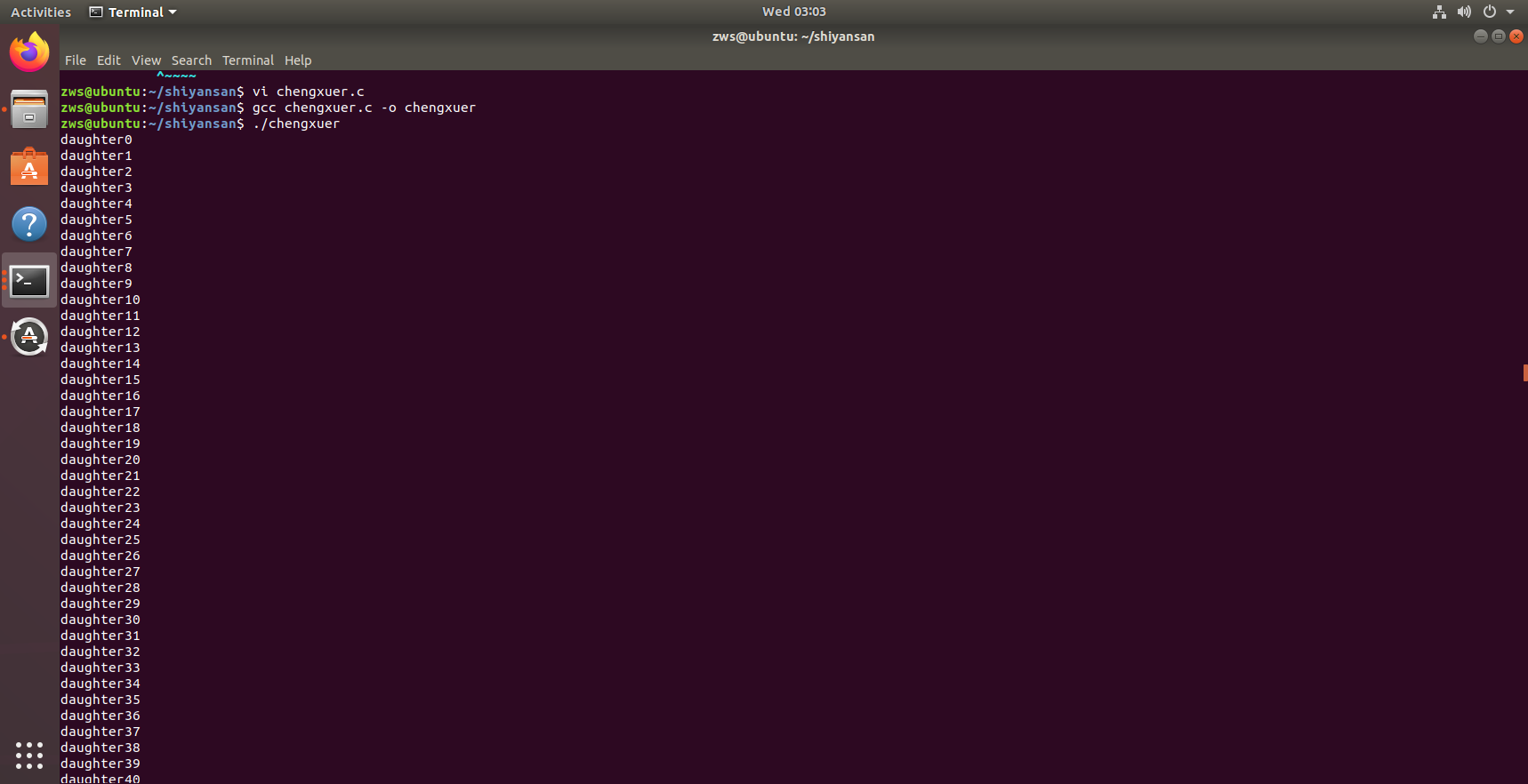
lockf(1，0，0)；

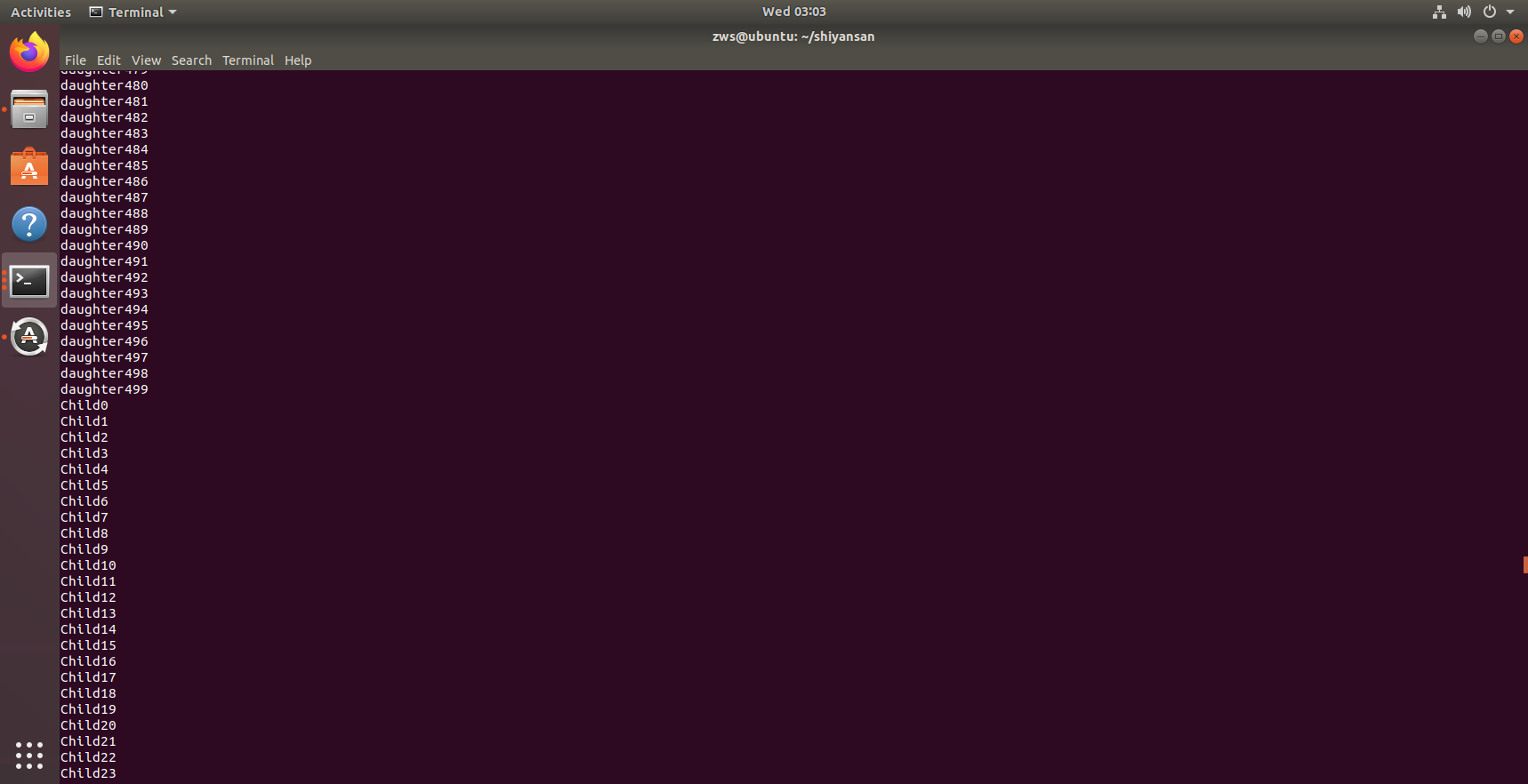
}

}

}

运行结果：





大致与未上锁的输出结果相同，也是随着执行时间不同，输出结果的顺序有所不同。

分析：因为上述程序执行时，不同进程之间不存在共享临界资源(其中打印机的互斥性已由操作系统保证)问题，所以，加锁与不加锁效果相同。因为三个进程输出有互斥性，父进程和子进程中的 lockf() 函数用来对输出语句进行互斥保护，确保每次输出时只有一个进程在输出，而其他进程必须等待到当前进程的输出完成后，才能进行输出。在输出完一条语句后，对应进程调用 lockf() 函数释放资源，使其他进程可以输出。当一个进程释放了资源后，随即又被操作系统分配到了资源，此时它便可以直接进行下一条输出语句，这个过程是由操作系统控制的，输出语句的顺序可能仍然是不确定的。

**3.一个模拟进程调度程序**

题目：设计一个有 N个进程并发执行的进程调度模拟程序。

进程调度算法：采用最高优先级优先的调度算法（即把处理机分配给优先级最高的进程）和先来先服务（若优先级相同）算法。

(1).  每个进程有一个进程控制块（PCB）表示。进程控制块包含如下信息：进程名、优先级、到达时间、需要运行时间、已用CPU时间、进程状态等等。

(2).  进程的优先级及需要的运行时间可以事先人为地指定，进程的运行时间以时间片为单位进行计算。

(3).  每个进程的状态可以是就绪 r（ready）、运行R（Running）、或完成F（Finished）三种状态之一。

(4).  就绪进程获得 CPU后都只能运行一个时间片。用已占用CPU时间加1来表示。

(5).  如果运行一个时间片后，进程的已占用 CPU时间已达到所需要的运行时间，则撤消该进程，如果运行一个时间片后进程的已占用CPU时间还未达所需要的运行时间，也就是进程还需要继续运行，此时应将进程的优先数减1（即降低一级），然后把它插入就绪队列等待调度。

(6).  每进行一次调度程序都打印一次运行进程、就绪队列中各个进程的 PCB，以便进行检查。

(7).  重复以上过程，直到所要进程都完成为止。

代码：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<time.h>

#define N 3

typedef struct jcb{

char name[10];//进程名

int reqtime;//要求服务时间

int prio;//优先级

int runtime;//运行时间

char status;

}PCB;

int intarr=0,intfin=0;//到达进程个数，完成进程个数

PCB pcbarr[24],pcbfin[24];

void Mune()

{

printf("\n\n");

printf("\t\t|------------------------------------------------|\n");

printf("\t\t| 进程调度模拟程序 |\n");

printf("\t\t|------------------------------------------------|\n");

printf("\t\t| 0:退出 |\n");

printf("\t\t| 1:运行（最高优先级优先算法） |\n");

printf("\t\t| 2:插入 |\n");

printf("\t\t| 3:删除 |\n");

printf("\t\t|------------------------------------------------|\n");

printf("请选择<0~3>：");

}

void Input()//手动输入

{

int i;

int n=0;

printf("请输入进程个数:");

scanf("%d",&n);

for(i=0;i<n;i++,intarr++)

{

printf("\n第%d个进程：\n请输入进程名:",i+1);

scanf("%s",pcbarr[intarr].name);

printf("请输入要求服务时间:");

scanf("%d",&pcbarr[intarr].reqtime);

pcbarr[intarr].prio=N;//优先级为3

if(i==0)

pcbarr[intarr].status='r';

else

pcbarr[intarr].status='w';

}

}

void Output()//输出排序后队列

{

int i=0;

printf("\n\n运行队列的是\n");

printf("\tname\treqtime\truntime\tprio\tstatus\n");

if(intarr!=0){

printf("N%d\t%s\t%d\t%d\t%d\t%c\n",i,pcbarr[i].name,pcbarr[i].reqtime,pcbarr[i].runtime,pcbarr[i].prio,pcbarr[i].status);

}

printf("\n就绪队列的是\n");

printf("\tname\treqtime\truntime\tprio\tstatus\n");

for(i=1;i<intarr;i++){

printf("N%d\t%s\t%d\t%d\t%d\t%c\n",i,pcbarr[i].name,pcbarr[i].reqtime,pcbarr[i].runtime,pcbarr[i].prio,pcbarr[i].status);

}

printf("\n已完成队列的是\n");

printf("\tname\treqtime\truntime\tprio\tstatus\n");

for(i=0;i<intfin;i++){

printf("N%d\t%s\t%d\t%d\t%d\t%c\n",i,pcbfin[i].name,pcbfin[i].reqtime,pcbfin[i].runtime,pcbfin[i].prio,pcbfin[i].status);

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//按优先级排序

void Sort()

{

int i,j;

PCB temp;

for(i=0;i<intarr-1;i++)

{

for(j=i+1;j<intarr;j++)

{

if(pcbarr[i].prio<pcbarr[j].prio)

{

temp=pcbarr[i];

pcbarr[i]=pcbarr[j];

pcbarr[j]=temp;

}

}

}

pcbarr[0].status='r';

for(i=1;i<intarr;i++)

{

pcbarr[i].status='w';

}

}

void running()

{

int slice,i,k;

slice=1;

for(i=1;i<((N+1)-pcbarr[0].prio);i++)

slice=slice\*2;

for(i=1;i<=slice;i++)

{

pcbarr[0].runtime++;

if (pcbarr[0].runtime==pcbarr[0].reqtime)

break;

}

if(pcbarr[0].runtime==pcbarr[0].reqtime)

{

printf("\n 进程 [%s] 已完成.\n",pcbarr[0].name);

pcbfin[intfin]=pcbarr[0];

for(k=0;k<=intarr;k++)

{

pcbarr[k]=pcbarr[k+1];

}

intarr--;

intfin++;

}

else

{

if(pcbarr[0].prio>1) pcbarr[0].prio--;

}

}

void Delete()

{

char b[20];

int i,j,key=0;

printf("请输入要删除的进程名：");

scanf("%s",b);

for(i=0;i<intarr;i++)

{

if(strcmp(b,pcbarr[i].name)==0)

{

key=1;

for(j=i;j<=intarr;j++)

{

pcbarr[j]=pcbarr[j+1];

}

intarr--;

}

}

for(i=0;i<intfin;i++)

{

if(strcmp(b,pcbfin[i].name)==0)

{

key=1;

for(j=i;j<=intfin;j++)

{

pcbfin[j]=pcbfin[j+1];

}

intfin--;

}

}

if(key==0)

{

printf("查找不到该进程！");

}

}

Int main()

{

int chose;

Input();

Output();

while(1)

{

Mune();

scanf("%d",&chose);

switch(chose)

{

case 0:exit(0);break;

case 1:running();break;

case 2:Input();break;

case 3:Delete();break;

default:printf("输入错误!");

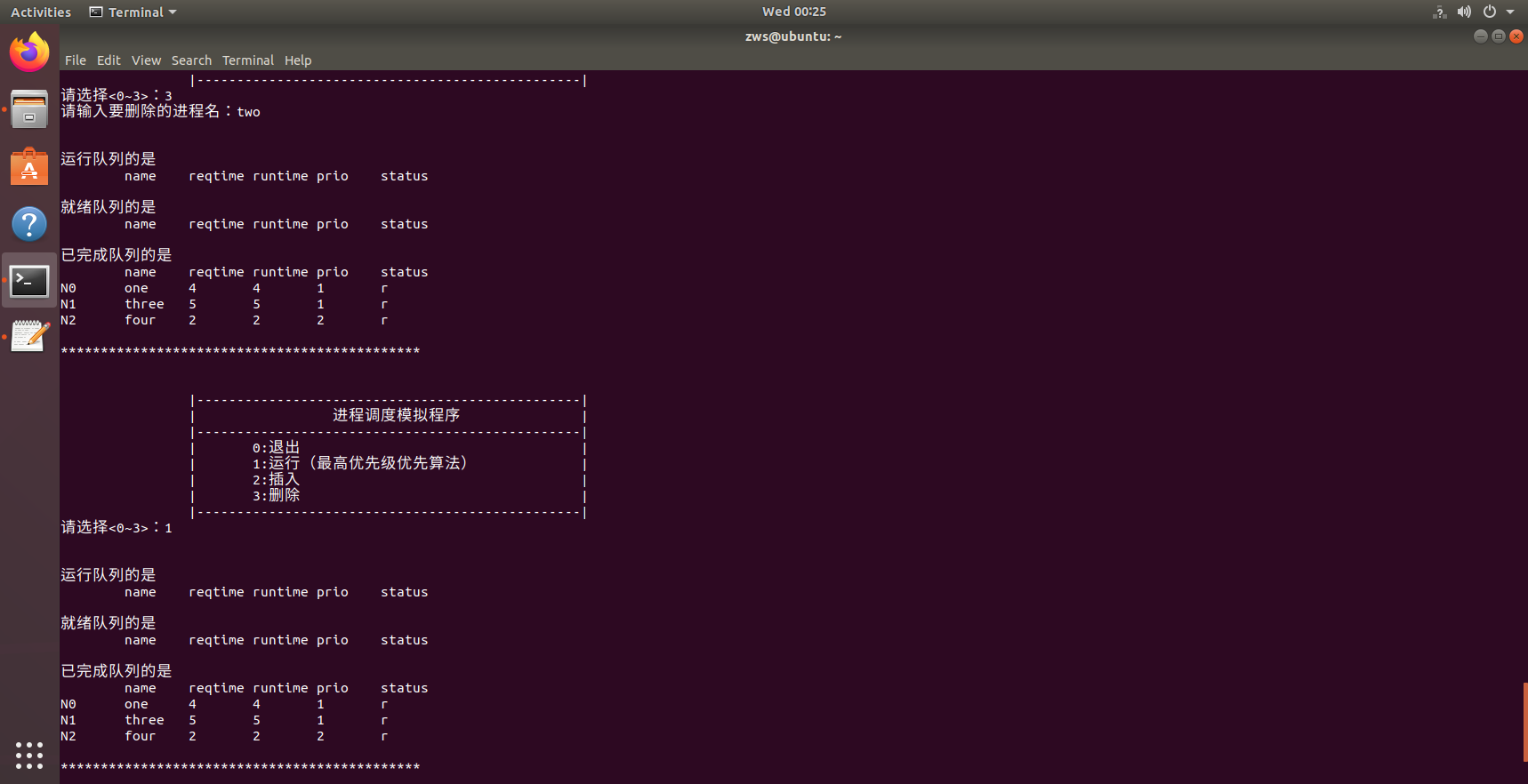
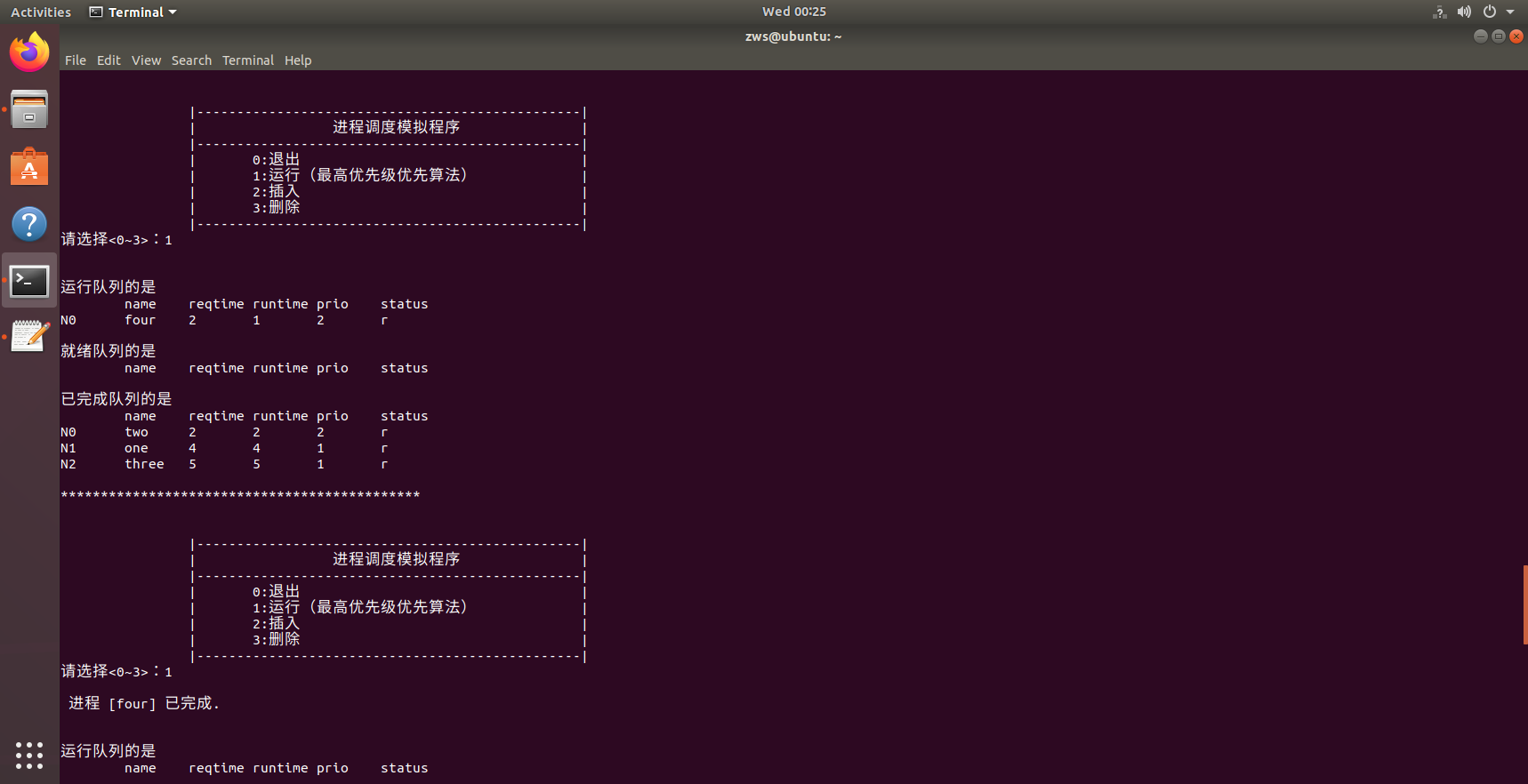
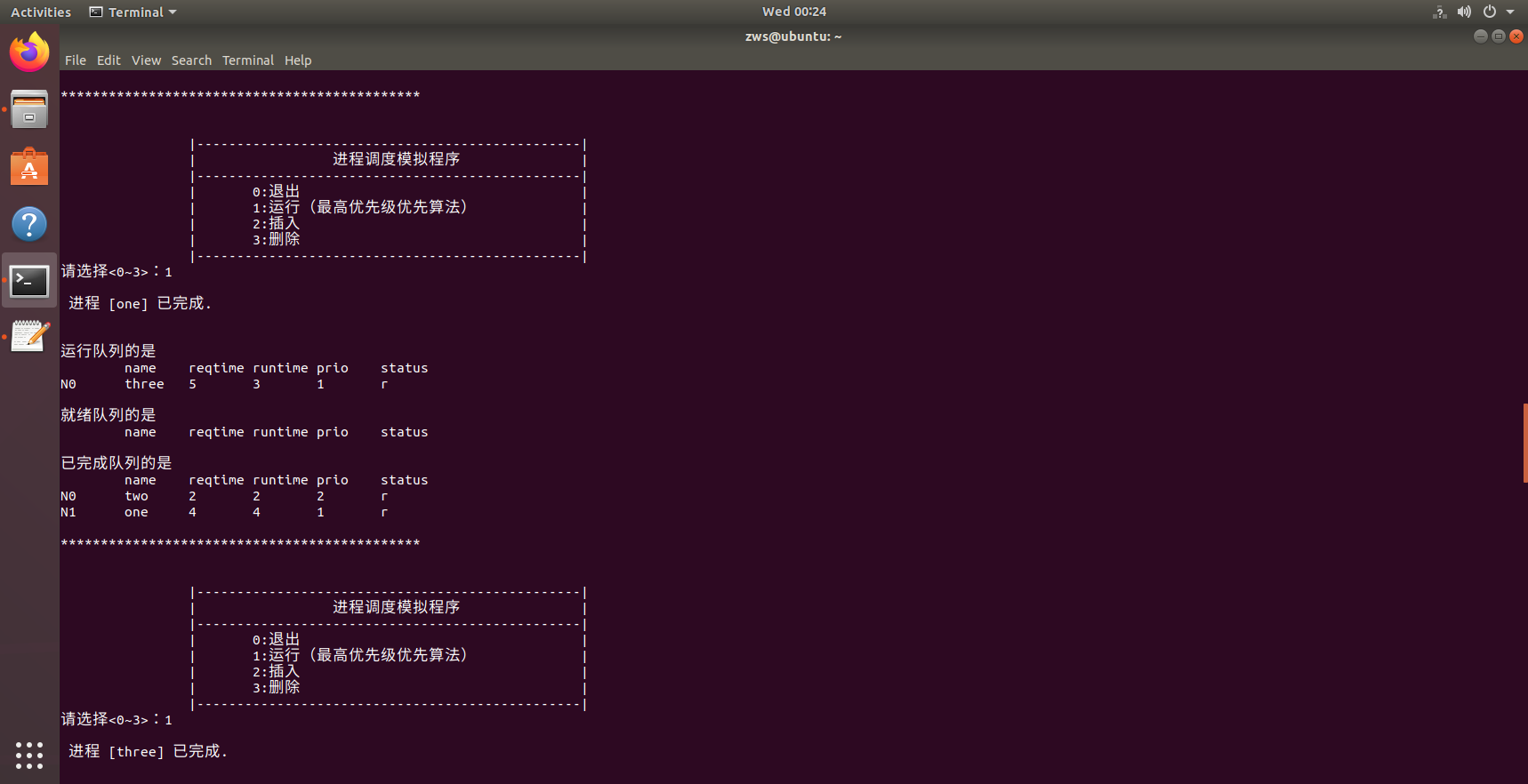
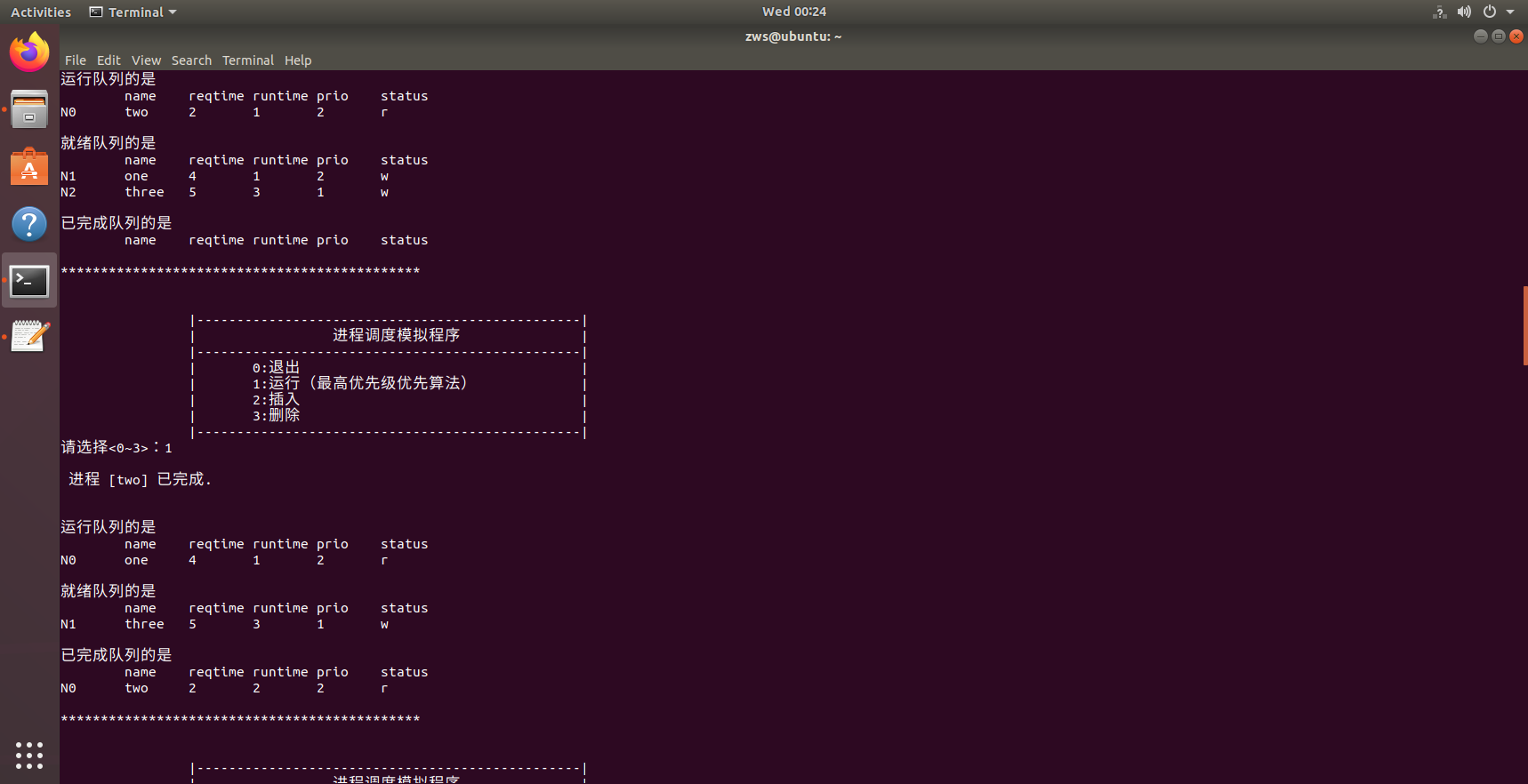
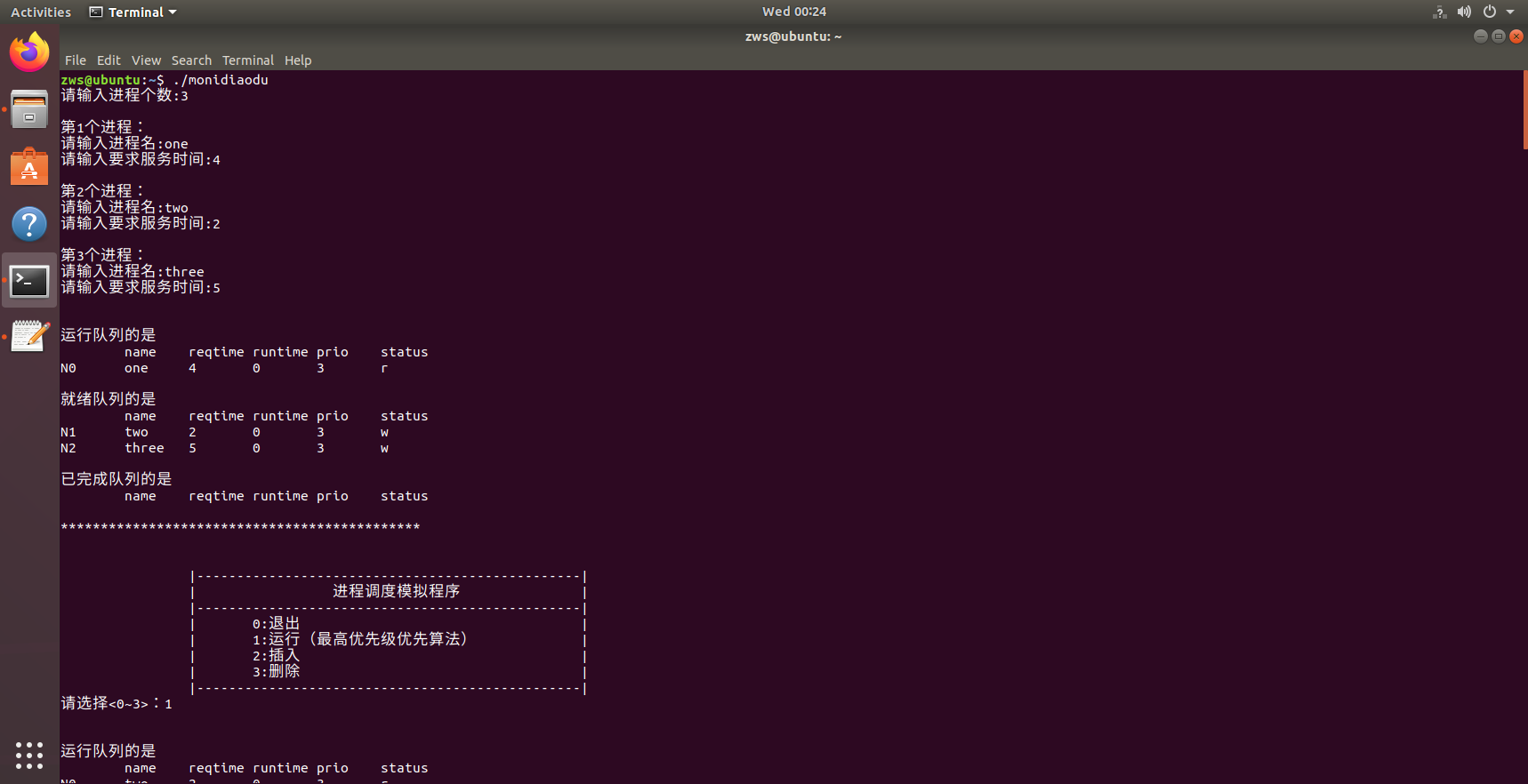
}

Sort();

Output();

}

}



## 五．实验报告

心得体会：加深了对进程概念的理解，进一步明确进程和程序的区别，进一步认识并发执行的实质。由于进程并发执行时的调度顺序和父子进程的抢占处理机问题，输出字符串的顺序和先后随着执行的不同而发生变化。通过进程调度模拟程序，我进一步理解并发调度算法的原理和实现过程，深入了解了进程控制块(PCB)的作用和内容。对于优先级高的进程先执行，以及如何实现时间片轮转等功能也有了更深的认识与思考，通过时间片，进程的已占用 CPU时间已达到所需要的运行时间，则撤消该进程，如果运行一个时间片后进程的已占用CPU时间还未达所需要的运行时间，也就是进程还需要继续运行，此时应将进程的优先数减1（即降低一级），然后把它插入就绪队列等待调度。根据PCB中优先级信息优先级高的进程最先先被调度。