操作系统上机报告

**专业班级： 计本0602**

**学 号： 200611182**

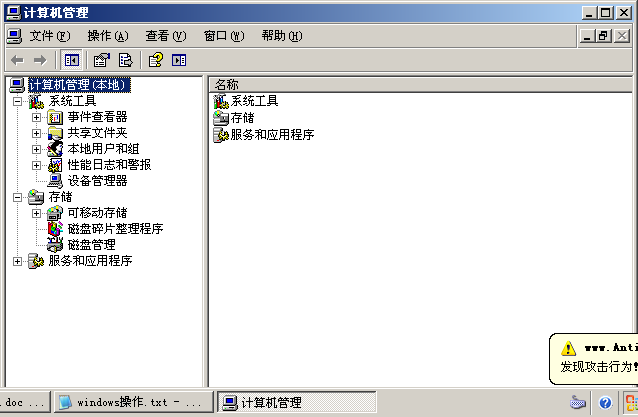
**班内序号： 25**

**指导老师： 贾 晓 艳**

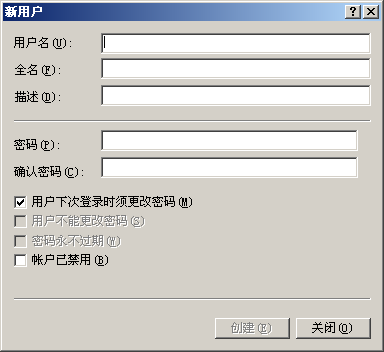
**学生姓名： 熊 建 市**

**第一部分 Windows基本操作**

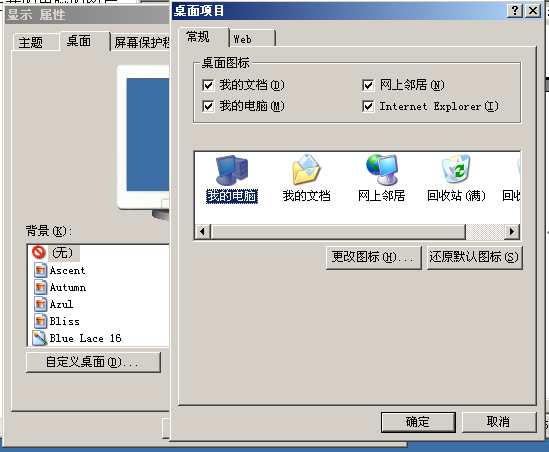
* 创建多用户
* 右击【我的电脑】>>选择【管理】



* 选中【本地用户和组】中的【用户】>>右击选择【新建用户】



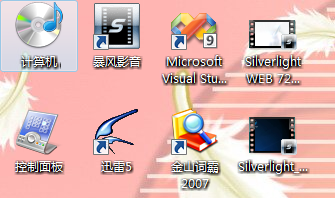
* 如上图填好用户名和密码单击【创建】即可.
* 更改我的电脑的图标
* 桌面上单击右键选择【属性】>>【桌面】>>【自定义桌面】



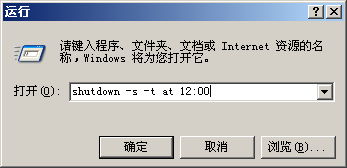
* 选中【我的电脑】>>单击【更改图标】



* 选中一个图标单击【确定】，效果如图计算机的图标改成了上图选中的图标



* 定时关机
* 单击【开始】>>【运行】



* 如图输入【shutdown –s –t at 12:00】单击【确定】就会在12:00准时关机.

**第二部分 编程实验**

1.问题的提出

* 设计一个有多个进程并行的进程调度程序.

2.功能要求

* C#编写一个简化的模拟进程调度程序，其调度算法采用优先数算法和时间片轮转法。
* 该程序能描述五个或五个以上的进程调度情况。

3.系统功能模块框图设计

Y

N

输入N

输入S

不具有抢断性的优先数算法

时间片轮转法

已就绪的进程按优先数的由高到低排序

运行完一个时间片就排到队列的后面

判断进程是否就绪

创建一个进程类

实例化五个进程，定义一个全局变量作为模拟时间，定义静态数组，未就绪数组，已就绪数组，定义一个全局变量作为你选则了何种进程调度方式

主函数

初始化所有进程，数组

输出所有进程当前状态

放入就绪队列

放入未就绪队列

*已就绪*

*未就绪*

*如果时间等于0*

时间增长1

判断时间片是否用完

Y

判断已经运行的时间是否等于总时间

从就绪队列中删除

判断已经运行的进程是否运行完

如果时间不等于0

如果时间不等于0

判断所有的进程是否运行完

Y

N

N

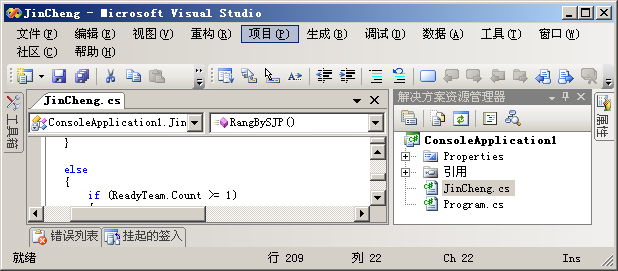
**如图程序放在JinCheng.cs类文件中，Program.cs中为主程序**  

图 1-1

**如图1-1:JinCheng.cs是程序类文件(所有的函数都是在JinCheng.cs类文件里写的)**

**代码如下:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Data; 命名空间

using System.Configuration;

using System.Collections;

namespace ConsoleApplication1

{

定义一个CourseInput类具有id,prc,cputime,alltime,next,state,InputTime

class CourseInput

{ public string id; **//标识符**

public int prc; **//优先数**

public int cputime; **//时间片**

public int alltime; **//总时间**

public int next; **//进程已经运行的时间**

public string state; **//当前状态**

public int InputTime; **//进程输入时间**

}

class JinCheng

{ public CourseInput Course1 = new CourseInput(); **//创建进程**

public CourseInput Course2 = new CourseInput(); **//创建进程**

public CourseInput Course3 = new CourseInput(); **//创建进程**

public CourseInput Course4 = new CourseInput(); **//创建进程**

public CourseInput Course5 = new CourseInput(); **//创建进程**

public ArrayList IdStaticTeam = new ArrayList(); **//创建进程id数组**

public ArrayList NoReadyTeam = new ArrayList(); **//创建未就绪进程id数组**

public ArrayList ReadyTeam = new ArrayList(); **//创建未就绪就绪队列**

public static int i = 0; **//虚拟时钟**

public static int k = 0;

public string Choose = ""; **//存储你选择了何种算法**

**//添加测试数据函数**

public void DataPrepareFunction()

{

Course1.id = "a"; Course1.prc = 4; Course1.cputime = 6; Course1.alltime = 10; Course1.InputTime = 0;

Course2.id = "b"; Course2.prc = 6; Course2.cputime = 4; Course2.alltime = 20; Course2.InputTime = 5;

Course3.id = "c"; Course3.prc = 5; Course3.cputime = 5; Course3.alltime = 15; Course3.InputTime = 4;

初始化进程和未就绪数组

Course4.id = "d"; Course4.prc = 8; Course4.cputime = 8; Course4.alltime = 24; Course4.InputTime = 2;

Course5.id = "e"; Course5.prc = 3; Course5.cputime = 7; Course5.alltime = 30; Course5.InputTime = 20;

**//初始化next和state**

if (i == 0)

{ Course1.next = 0;

Course2.next = 0;

Course3.next = 0;

Course4.next = 0;

Course5.next = 0;

}

NoReadyTeam.Add(Course1.id);

NoReadyTeam.Add(Course2.id);

NoReadyTeam.Add(Course3.id);

NoReadyTeam.Add(Course4.id);

NoReadyTeam.Add(Course5.id);

}

**//输出进程的当前状态**

public void CourseNowState()

{ if (i == 0)

{

Console.WriteLine("标识符 " + "优先数 " + "时间片" + "总时间 " + "进程进入时间 ");

for (int j = 0; j < IdStaticTeam.Count; j++)

{ Console.WriteLine(" " + Course(j, IdStaticTeam).id + " " + Course(j, IdStaticTeam).prc + " " + Course(j, IdStaticTeam).cputime + " " + Course(j, IdStaticTeam).alltime + " " + Course(j, IdStaticTeam).InputTime); ;

}

}else

{

Console.WriteLine("标识符 " + "优先数 " + "时间片" + "总时间 " + "进程进入时间 " + "进程当前状态 " + "以运行的时间");

for (int j = 0; j < IdStaticTeam.Count; j++)

{ Console.WriteLine(" " + Course(j, IdStaticTeam).id + " " + Course(j, IdStaticTeam).prc + " " + Course(j, IdStaticTeam).cputime + " " + Course(j, IdStaticTeam).alltime + " " + Course(j, IdStaticTeam).InputTime + " " + Course(j, IdStaticTeam).state + " " + Course(j, IdStaticTeam).next); ;

}

}

}

**//初始化IdStaticTeam和NoReadyTeam数组**

**初始化IdStaticTeam和NoReadyTeam数组**

public void UpdateArray()

{

IdStaticTeam.Add(Course1.id);

IdStaticTeam.Add(Course2.id);

IdStaticTeam.Add(Course3.id);

IdStaticTeam.Add(Course4.id);

IdStaticTeam.Add(Course5.id);

}

**//根据数组中的id值获取Course（进程）的名称**

public CourseInput Course(int j,ArrayList MiddleArray)

{ if (MiddleArray.Count>=1)

**根据MiddleArray数组中的进程id返回进程的名称**

{switch (MiddleArray[j].ToString())

{case "a":return Course1;

case "b":return Course2;

case "c":return Course3;

case "d":return Course4;

case "e":return Course5;

default :return null;

}

} else

{return null;}

}

//判断进程是否就绪（当i（虚拟时钟）=0时，选择进程调度的方式）

public void ConfirmReady()

{for (int a = 0; a < NoReadyTeam.Count; a++)

{if (Course(a,NoReadyTeam).InputTime<= i)

{Course(a, NoReadyTeam).state = "就绪";

**判断进程是否就绪（当i（虚拟时钟）=0时，选择进程调度的方式）**

ReadyTeam.Add(Course(a, NoReadyTeam).id.ToString());

NoReadyTeam.Remove(

Course(a, NoReadyTeam).id.ToString());

}else{ Course(a, NoReadyTeam).state = "未就绪";}

}if (i == 0)

{Console.WriteLine("请选择进程调度算法");

Console.WriteLine("输入【N】表示选择

不具有抢断性的优先数算法");

Console.WriteLine("输入【S】表示选择时间片轮转法");

Choose = Console.ReadLine();

if (Choose == "N")

{ Console.WriteLine("你选择了不具有抢断性的

优先数调度算法");RangeByYXS();

}else if (Choose == "S")

{ Console.WriteLine("你选择了时间片调度算法");

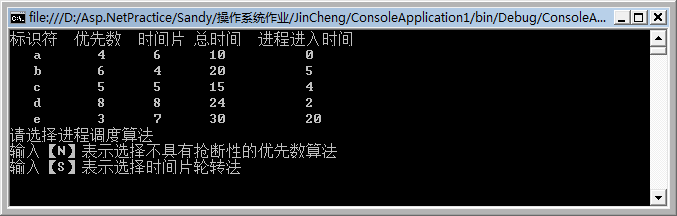
RangBySJP();

}

}

}

**如下图即为：选择进程调度的方式**



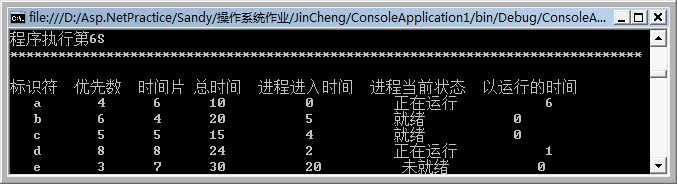
//为进程执行判断就绪函数

public void ReadsTeam()

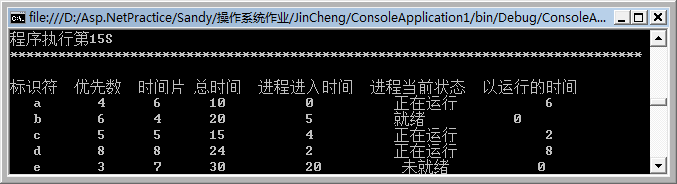
{ ConfirmReady(); //调用判断进程是否就绪的函数ConfirmReady（） Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); Console.Read();

}

**如下图：当a运行完后，d就放在了就绪进程数组的最上面，所以a运行完一个时间片后，就放到就绪进程数组的最后面，d就开始运行了。**



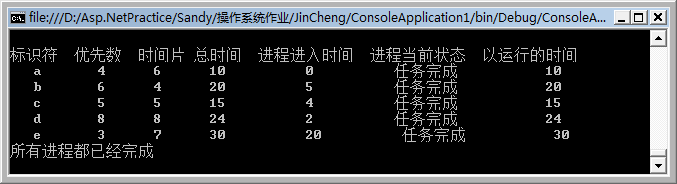
**如下图：d运行完后放到就绪队列的末尾，c开始运行。**



**如下图：当就绪队列中的进程轮流执行完一遍后，a又开始运行，然后依次循环。**



**如下图：所有进程都执行完成。**



//时间片轮转顺序调整

public void PanDuanSJP()

**已经运行完一个时间片的进程的id放到已就绪进程数组（ReadyTeam）的末尾**

{if(ReadyTeam.Count>=1)

{ string a = ReadyTeam[0].ToString();

for (int j = 0; j < ReadyTeam.Count-1; j++)

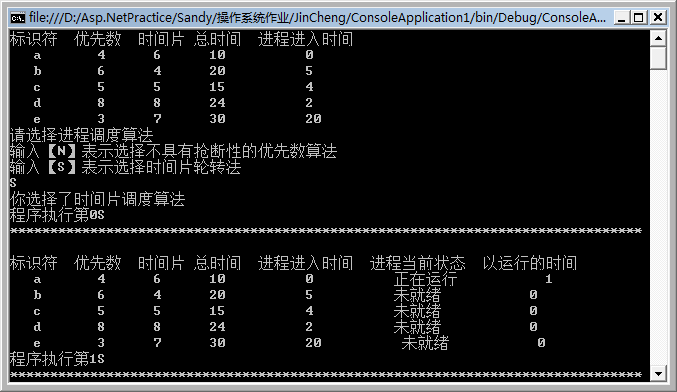
{ReadyTeam[j] = ReadyTeam[j + 1].ToString();}

ReadyTeam[ReadyTeam.Count - 1] = a;

}

}

**如下图：选择了时间片调度算法**



**//时间片轮转法（调用Couse（）函数取得进程，判断一个进程的时间片是否运行完，如果已经运行完就调用时间片排序函数（PanDuanSJP()），**

**如果未运行玩就调用判断就绪函数ReadyTeam（），输出进程状态CourseNowState();如果进程已经运行的时间已经到达总时间就把进程从就绪进程数组中删除）**

public void RangBySJP()

{if ((Course(0, IdStaticTeam).next == Course(0,IdStaticTeam).alltime) && (Course(1, IdStaticTeam).next == Course(1, IdStaticTeam).alltime) && (Course(2, IdStaticTeam).next == Course(2, IdStaticTeam).alltime) && (Course(3, IdStaticTeam).next == Course(3, IdStaticTeam).alltime) && (Course(4, IdStaticTeam).next == Course(4, IdStaticTeam).alltime))

{Console.WriteLine("所有进程都已经完成");}

else {if (ReadyTeam.Count >= 1)

{for(int j=0;j<Convert.ToInt32(Course(0,ReadyTeam).cputime);j++)

{Course(0, ReadyTeam).next++;

if (Course(0, ReadyTeam).next >= Course(0, ReadyTeam).alltime)

{ Course(0, ReadyTeam).state = "任务完成";

ReadyTeam.Remove(Course(0, ReadyTeam).id.ToString());

ReadsTeam();

CourseNowState();

RangBySJP();

}else{ Console.WriteLine("程序执行第" + i + "S");

Course(0, ReadyTeam).state = "正在运行";

i++;

ReadsTeam();

CourseNowState();

Console.Read();

}

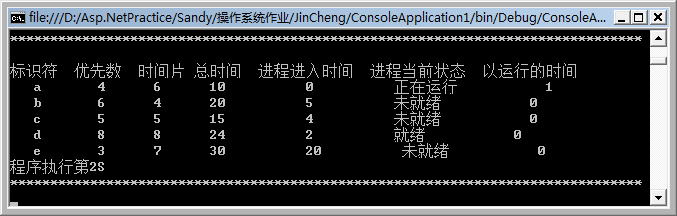
} PanDuanSJP();RangBySJP();

}

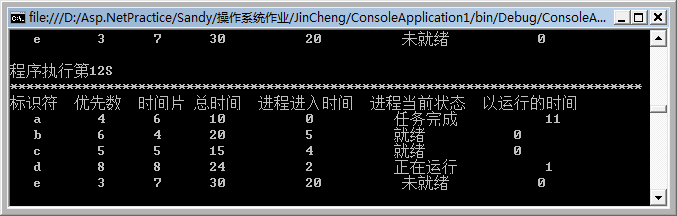
}

}

**如下图：先运行就绪队列中的第一个进程。**



**如下图：当最先就绪的进程a运行完后，对就绪队列进程优先级的高低进行排序。就绪队列中优先级最高的d进程。**



**//将进程id按进程的优先数由高到底排列**

public void PanDuanYXJ()

{ for (int a = 0; a < ReadyTeam.Count; a++)

{for (int b = 0;b<ReadyTeam.Count-1-a;b++)

{if (Convert.ToInt32(Course(b, ReadyTeam).prc) < Convert.ToInt32(Course(b + 1, ReadyTeam).prc))

{string c = ReadyTeam[b+1].ToString();

ReadyTeam[b] = ReadyTeam[b].ToString();

ReadyTeam[b+1] = c;

}

}

}for (int j = 0; j < ReadyTeam.Count; j++)

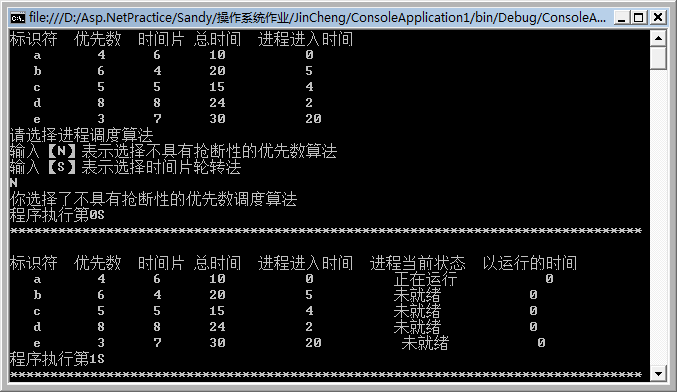
{Console.WriteLine(ReadyTeam[j].ToString());}

if (Choose == "N")

{ RangeByYXS();}

}

**如下图：选择了不具有抢断性的优先数算法。**



**//不具有抢断性的优先数算法（当就绪进程数组第一次不为空的时候，就调用优先数排序函数（PanDuanYXJ（）），然后就运行就绪数组中的第一个元素，运行期间时间总是自动加1，不断的调用判断进程是否就绪的函数（ReadsTeam（））和输出进程当前状态的函数（CouseNowState（）），因为是不具有抢断性的优先数算法，如果就绪队列中新就绪的进程比正在运行的进程优先级高的进程，仍然运行当前正在运行的进程，而不运行新就绪的进程，当正在运行的进程运行完后，就将其从就绪队列中移除，然后再将就绪队列按照优先级的高低排序，在运行就绪队列中的第一个元素。如此重复执行，知道所有进程全部运行完，就结束程序。**

public void RangeByYXS()

{if (ReadyTeam.Count >= 1)

{for (; Course(0, ReadyTeam).next <= Course(0, ReadyTeam).alltime; Course(0, ReadyTeam).next++)

{ Console.WriteLine("程序执行第" + i + "S");

Course(0, ReadyTeam).state = "正在运行";

i++;

ReadsTeam();

CourseNowState();

Console.Read();

}

Console.WriteLine("程序执行" + i + "S");

Course(0, ReadyTeam).state = "任务完成";

ReadsTeam();

CourseNowState();

ReadyTeam.Remove(Course(0, ReadyTeam).id);

RangeByYXS();

if (ReadyTeam.Count >= 2)

{PanDuanYXJ();}

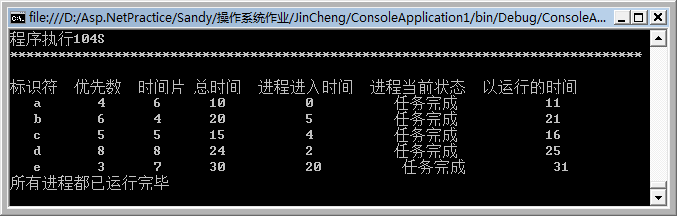
}else{Console.WriteLine("所有进程都已运行完毕");

Console.Read();

}

}

**如下图：所有的进程都已经执行完毕。**



public void Time()  **//JinCheng.cs类中的主函数**

{if (i == 0) **//i作为模拟时间**

{ DataPrepareFunction();**//将所有的进程初始化**

UpdateArray();**//更行所有的数组**

CourseNowState();**//输出进程的当前状态**

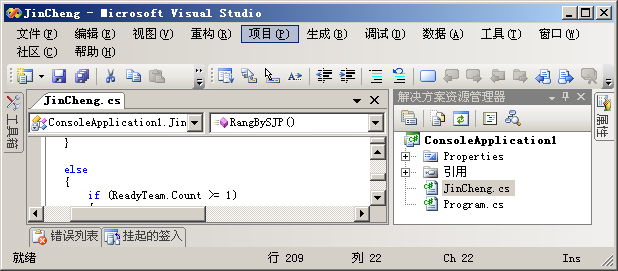
ConfirmReady();**//判断是否有进程已经就绪**

}

}

}

}



如图1-1:Program.cs是执行JinCheng.cs的类文件

代码如下:

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

JinCheng Course = new JinCheng();**//实例化进程调度类**

Course.Time();**//执行进程调度类中的主函数**

}

}

}

**第三部分 Linux指令的基本操作及例子**

文件目录操作命令：

ls 命令

功能：用来列出目录的内容。

参数: -l (表示以长格式列出)

-a (表示要列出所有的文件)

例1: $ ls -al 用长格式列出所有的文件包括隐藏文件

touch 命令

功能: 生成一个空文件或用来修改文件存取时间

参数: -d (yyyymmdd的形式给出要修改的时间)

例1: $ touch abc

若abc文件存在，则修改为系统当前的时间；若不存在，则生成一个为当前时间的空文件。

例2: $ touch -d 19771101 abc

将abc文件的日期改为19771101。

mkdir 命令

功能: 建立目录 。

参数: -p (建立所有不存在父目录的目录)

例1: $ mkdir -p xnd/wll

在当前目录下建立xnd和xnd下的wll的目录,也就是连续建立两个目录（xnd和xnd/wll)。

rmdir 命令

功能: 删除一个空的目录。

参数: -p (递归删除目录)

例1: $ rmdir -p xnd/wll

将删除xnd下的wll的目录，若xnd目录为空也删除它。

cp 命令

功能: 复制文件或目录 。

参数: -f (强行覆盖)

-r (拷贝目录)

例1: $ cp -r /usr/xu/ /usr/liu/

将/usr/xu目录中的所有文件及其子目录拷 贝到目录/usr/liu中。

mv 命令

功能: 搬移和更名文件或目录

参数: -f (强行覆盖)

例1: $ mv wll.txt wll.doc

将文件wll.txt重命名为wll.doc

rm 命令

功能: 删除文件或目录。

参数: -f (不需用户确认)

-r (删除目录)

例1: $ rm -rf test/

把test及其目录下的所有文件与目录全部删除，不用用户确认。

file 命令

功能: 检测文件类型

参数: -z (检测压缩文件 )

例1: $ file \*

检测当前目录下所有文件的类型

cat 命令

功能: 连接两个或多个文件

参数: -v (用显示二进制文件 )

-A (显示特殊字符 )

例1: $ cat file1 file2 > file3

filel和文件file2内容合并起来，放入文件file3中。

more 命令

功能: 显示文件内容

参数: 可与其他命令合用，如(ls、ps等)。

例1: $ more wll.txt

显示wll.txt文件的内容。

less 命令

功能: 显示文件内容

参数: 可与其他命令合用，如(ls、ps等)。

例1: $ less wll.txt

显示wll.txt文件的内容。

head 命令

功能: 显示指定文件的前若干行 。

参数:

例1: $ head -3 example.c

显示文件example.c的前3行。

tail 命令

功能: 显示指定文件的后若干行

参数:

例1: $ tail -4 example

显示文件example的最后4行

sort 命令

功能: 对文件内容进行排序

参数:

例1: $ sort text

对文件text进行排序

paste 命令

功能: 将文件的行连接起来，即进行横向合并

参数:

例1: $ paste file1 file2 >file3

将文件file1和file2合并输出到文件file3

pr 命令

功能: 处理文件以便打印，输出到标准输出

参数: -n (行号)

例1: $ pr file -n

将file文件的各行加上行号后输出

ln 命令

功能: 连接一个文件 ,相当一个快捷方式

参数: -s (建立软连接)

例1: $ ln –s wll.txt test.link

将为wll.txt建立一个连接test.link

find 命令

功能: 查找文件或目录

参数: -name(查找匹配所给字串的所有件,\* ?)

例1: $ find –name wll\*

将查找/home目录下的所有以wll开头的文件及其子目录

grep 命令

功能: 查找文件的内容，内容匹配则输出文件名

参数:

例1: $ grep data \*

在当目录的所有文件中查找含有data字符串的文件中。

pwd 命令

功能: 显示当前目录

参数:

例1: $ pwd

输出: /home

cd 命令

功能: 改变当前目录

参数:

例1: $ cd /etc 进入etc目录

例2: $ cd - 返回上一次进入的目录

参数: -f (强制)

-d (解压缩)

例1: $ gzip usr.tar

将usr.tar文件压缩为usr.tar.gz。

**第四部分 总结**

通过这次实验，我学会使用类，了解了计算机进程调度的集中方法及具体思想，学会了是用ArrayList数组。学会了怎样开启Linux系统和一些简单的Linux指令。但这次实验还是有一些不足，就是没有作出具有抢断性的优先数进程调度算法，当已经基本了解了具有抢断性的优先数调度算法的思想。通过这次实验我学到了很多，也懂得了很多。让我了解了自己的许多不足之处，我以后一定回更加努力的学习，掌握更多、更深层次的知识。