**引言**

通过模拟操作系统原理的实现，加深对操作系统工作原理和操作系统实现方法的理解；

通过模拟操作系统原理的实现练习编程

**1 模拟系统的可执行程序**

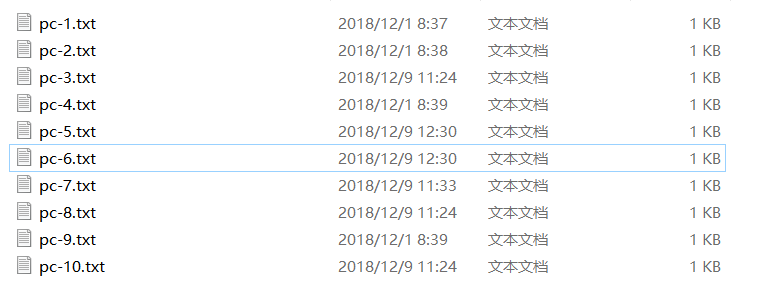
x=?; 给x赋值一位数

x++; x加1

x--; x减1

!??； 第一个？表示阻塞原因A,B(I/O申请），第二个？为一位数，表示阻塞时间（cpu循环次数）；

end. 表示程序结束，其中包括文件路径名和x的值（软中断方式处理）。



**2.硬件模拟**

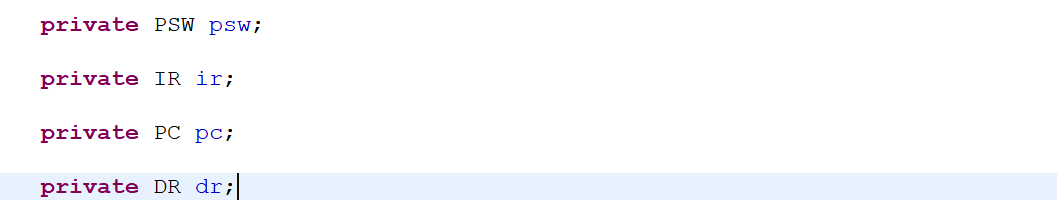
(1)寄存器的模拟：

程序状态寄存器PSW//可以用1表示时钟中断，2表示输入输出中断，4表示软中断，可以组合1+2,1+4,,2+4,1+2+4

指令寄存器IR //存放4个字符即可

程序计数器 //可分为两部分，PCB编号和指令位置

数据缓冲寄存器DR//存放x的值



1. cpu的模拟：

用函数CPU( )（该函数不能有参数）模拟中央处理器。

该函数主要负责解释“可执行文件”中的命令。

x=?; x++; x--; !??； end.

代码如下：

package os;

import java.util.List;

import javax.swing.DefaultListModel;

import javax.swing.ListModel;

import entity.PCB;

import entity.PSW;

import util.CatchUtil;

import util.FileOp;

import util.InteruptUtil;

import util.TimeThread;

/\*\*\* @author hp

\* @date : 2018年12月8日 下午5:07:26\*/

public class CPU {

//cpu

public static void Run() throws Exception

{

synchronized (OS.interuptflag) {

if (OS.interuptflag.size()>0) {

List<PCB> list=OS.interuptflag;

for(int i=0;i<list.size();i++)

{

PCB interuptedpcb = OS.interuptflag.get(i);

OS.waitlist.remove(interuptedpcb);

OS.preparelist.add(interuptedpcb);

MainForm.waitcontent.removeElement(interuptedpcb.getOuttitle() + ":" + interuptedpcb.getBlock() + "-0");

MainForm.readycontent.addElement(interuptedpcb.getOuttitle());

}

OS.interuptflag.clear();

}

}

if(OS.preparelist.size()<=0)

{

//转向闲逛进程

new TimeThread(1).run();

MainForm.nowproc.setText("当前无进程");

MainForm.execing.setText("");

MainForm.timesplite.setText("");

MainForm.betwres.setText("");

//System.out.println("所有进程执行结束！！！！！！！");

return;

}

//时间片

int time=5;

MainForm.timesplite.setText(String.valueOf(time));

boolean isintrupt=false;

//PCB pcb=OS.pcbs.get(OS.preparelist.get(0));

PCB pcb=OS.preparelist.get(0);

OS.preparelist.remove(0);

//ListModel<String> model=MainForm.readylist.getModel();

try {

MainForm.readycontent.removeElement(pcb.getOuttitle());

MainForm.nowproc.setText(pcb.getOuttitle());

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

String noworder=pcb.getIr().getNowpoint();

Integer x=pcb.getDr().getX();

while(!noworder.equals("end.")&&time!=0)

{

synchronized (OS.interuptflag) {

if (OS.interuptflag.size()>0) {

List<PCB> list=OS.interuptflag;

for(int i=0;i<list.size();i++)

{

PCB interuptedpcb = OS.interuptflag.get(i);

OS.waitlist.remove(interuptedpcb);

OS.preparelist.add(interuptedpcb);

MainForm.waitcontent.removeElement(interuptedpcb.getOuttitle() + ":" + interuptedpcb.getBlock() + "-0");

MainForm.readycontent.addElement(interuptedpcb.getOuttitle());

}

OS.interuptflag.clear();

}

}

MainForm.execing.setText(noworder);

//System.out.println(noworder);

if(noworder.contains("="))

{

x=Integer.valueOf(noworder.substring(2, 3));

}

else if(noworder.contains("++"))

{

x++;

}

else if(noworder.contains("--"))

{

x--;

}

else if(noworder.contains("!"))

{

//中断

pcb.setBlock(noworder.substring(1, 2));

int blocktime=Integer.valueOf(noworder.substring(2,3));

pcb.getPsw().setState(1);

pcb.getIr().setNowpoint(OS.catches[pcb.getPc().getNextpoint()]);

pcb.getPc().setNextpoint(pcb.getPc().getNextpoint()+1);

//进程加入等待队列

//OS.waitlist.add(pcb.getPc().getPcbpost());

OS.waitlist.add(pcb);

MainForm.waitcontent.addElement(pcb.getOuttitle()+":"+pcb.getBlock()+"-"+blocktime);

//制定时间之后把等待队列里面的线程移入就绪队列

//System.out.println(blocktime);

//new InteruptUtil(blocktime, pcb.getPc().getPcbpost()).run();

new InteruptUtil(blocktime, pcb).run();

isintrupt=true;

break;

}

MainForm.betwres.setText(String.valueOf(time));

pcb.getDr().setX(x);

time--;

int nextpoint=pcb.getPc().getNextpoint();

noworder=OS.catches[nextpoint];

pcb.getIr().setNowpoint(noworder);

pcb.getPc().setNextpoint(nextpoint+1);

new TimeThread(1).run();

MainForm.timesplite.setText(String.valueOf(time));

}

if(!isintrupt)

{

if(time==0)

{

//OS.preparelist.add(pcb.getPc().getPcbpost());

OS.preparelist.add(pcb);

MainForm.readycontent.addElement(pcb.getOuttitle());

System.out.println(pcb.getOuttitle()+"时间片超时");

}

else

{

//结果写入

FileOp.resWrite(pcb.getOuttitle()+".txt", x);

MainForm.rescontent.addElement(pcb.getOuttitle()+":"+x);

//内存恢复

CatchUtil.catchRecycle(pcb.getCatchadd(), pcb.getSize());

CatchUtil.catchArrange();

System.out.println(pcb.getOuttitle()+"执行结束");

//PCB恢复

OS.pcbs.remove(pcb);

}

}

else

{

System.out.println(pcb.getOuttitle()+"Io中断");

}

//TODO 时间片清零

System.out.println("CPU Has Done!");

}

}

1. 中断的模拟：

中断的发现应该是硬件的工作，这里在函数CPU中加检测PSW的方式来模拟

在CPU（）函数中，每执行一条指令之前，先检查PSW，判断有无中断，若有进行中断处理，然后再运行解释指令。

中断检测：

synchronized (OS.interuptflag) {

if (OS.interuptflag.size()>0) {

List<PCB> list=OS.interuptflag;

for(int i=0;i<list.size();i++)

{

PCB interuptedpcb = OS.interuptflag.get(i);

OS.waitlist.remove(interuptedpcb);

OS.preparelist.add(interuptedpcb);

MainForm.waitcontent.removeElement(interuptedpcb.getOuttitle() + ":" + interuptedpcb.getBlock() + "-0");

MainForm.readycontent.addElement(interuptedpcb.getOuttitle());

}

OS.interuptflag.clear();

}

}

中断处理：

//中断

pcb.setBlock(noworder.substring(1, 2));

int blocktime=Integer.valueOf(noworder.substring(2,3));

pcb.getPsw().setState(1);

pcb.getIr().setNowpoint(OS.catches[pcb.getPc().getNextpoint()]);

pcb.getPc().setNextpoint(pcb.getPc().getNextpoint()+1);

//进程加入等待队列

//OS.waitlist.add(pcb.getPc().getPcbpost());

OS.waitlist.add(pcb);

MainForm.waitcontent.addElement(pcb.getOuttitle()+":"+pcb.getBlock()+"-"+blocktime);

//制定时间之后把等待队列里面的线程移入就绪队列

//System.out.println(blocktime);

//new InteruptUtil(blocktime, pcb.getPc().getPcbpost()).run();

new InteruptUtil(blocktime, pcb).run();

中断处理程序：

//System.out.println("111111111111");

MainForm.waitcontent.removeElement(pcb.getOuttitle()+":"+pcb.getBlock()+"-"+this.time);

this.time-=1;

MainForm.waitcontent.addElement(pcb.getOuttitle()+":"+pcb.getBlock()+"-"+this.time);

if(this.time==0)

{

synchronized (OS.interuptflag) {

//int index=Util.getPcbIndex(OS.waitlist, this.order);

OS.interuptflag.add(pcb);

this.pcb.getPsw().setState(0);

//OS.waitlist.remove(this.pcb);

//OS.preparelist.add(this.pcb);

//MainForm.waitcontent.removeElement(pcb.getOuttitle()+":"+pcb.getBlock()+"-"+this.time);

//MainForm.readycontent.addElement(pcb.getOuttitle());

this.cancel();

}

}

**３ 进程控制**

1. Pcb结构和组织方式

private String outtitle;

private int intitle;

private PSW psw;

private IR ir;

private PC pc;

private DR dr;

//阻塞原因

private String block;

private int catchadd;

private int size;

1. 进程创建

//开机装载PCB

OS.emptycatch.put(0, 128);

String []progressnames=FileOp.getFileName();

DefaultListModel<String> model=new DefaultListModel<String>();

for (int i = 0; i < progressnames.length; i++) {

String orders[]=FileOp.getFileContent(progressnames[i]);

int progresssize=orders.length;

//System.out.println(progressnames[i]+progresssize);

if(OS.pcbs.size()<10)

{

//申请内存块

int bginpost=-1;

try {

bginpost = CatchUtil.catchAllocate(orders);

} catch (Exception e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

if(bginpost==-1)

{

//申请内存失败

}

PCB pcb=new PCB();

pcb.setOuttitle(progressnames[i].replace(".txt", ""));

pcb.setIntitle((int)Math.random()\*100);

pcb.setSize(orders.length);

pcb.setCatchadd(bginpost);

pcb.setDr(new DR(null));

pcb.setIr(new IR(orders[0]));

pcb.setPc(new PC(OS.pcbs.size(), bginpost+1)); //默认所有的进程的语句数都超过二

//TODO

pcb.setPsw(new PSW(0));

pcb.setBlock("");

//装载pcb

OS.pcbs.add(pcb);

//进入就绪队列

//OS.preparelist.add(OS.pcbs.size()-1);

OS.preparelist.add(pcb);

//model.addElement(pcb.getOuttitle());

readycontent.addElement(pcb.getOuttitle());

}

}

readylist.setModel(readycontent);

waitlist.setModel(waitcontent);

1. 进程撤销

//结果写入

FileOp.resWrite(pcb.getOuttitle()+".txt", x);

MainForm.rescontent.addElement(pcb.getOuttitle()+":"+x);

//内存恢复

CatchUtil.catchRecycle(pcb.getCatchadd(), pcb.getSize());

CatchUtil.catchArrange();

System.out.println(pcb.getOuttitle()+"执行结束");

//PCB恢复

OS.pcbs.remove(pcb);

1. 进程阻塞

I/O中断：

//中断

pcb.setBlock(noworder.substring(1, 2));

int blocktime=Integer.valueOf(noworder.substring(2,3));

pcb.getPsw().setState(1);

pcb.getIr().setNowpoint(OS.catches[pcb.getPc().getNextpoint()]);

pcb.getPc().setNextpoint(pcb.getPc().getNextpoint()+1);

//进程加入等待队列

//OS.waitlist.add(pcb.getPc().getPcbpost());

OS.waitlist.add(pcb);

MainForm.waitcontent.addElement(pcb.getOuttitle()+":"+pcb.getBlock()+"-"+blocktime);

//制定时间之后把等待队列里面的线程移入就绪队列

//System.out.println(blocktime);

//new InteruptUtil(blocktime, pcb.getPc().getPcbpost()).run();

new InteruptUtil(blocktime, pcb).run();

时间片中断：

if(time==0)

{

//OS.preparelist.add(pcb.getPc().getPcbpost());

OS.preparelist.add(pcb);

MainForm.readycontent.addElement(pcb.getOuttitle());

System.out.println(pcb.getOuttitle()+"时间片超时");

}

进程唤醒

synchronized (OS.interuptflag) {

if (OS.interuptflag.size()>0) {

List<PCB> list=OS.interuptflag;

for(int i=0;i<list.size();i++)

{

PCB interuptedpcb = OS.interuptflag.get(i);

OS.waitlist.remove(interuptedpcb);

OS.preparelist.add(interuptedpcb);

MainForm.waitcontent.removeElement(interuptedpcb.getOuttitle() + ":" + interuptedpcb.getBlock() + "-0");

MainForm.readycontent.addElement(interuptedpcb.getOuttitle());

}

OS.interuptflag.clear();

}

}

**４进程调度**

先来先服务:

//时间片

int time=5;

MainForm.timesplite.setText(String.valueOf(time));

boolean isintrupt=false;

//PCB pcb=OS.pcbs.get(OS.preparelist.get(0));

PCB pcb=OS.preparelist.get(0);

OS.preparelist.remove(0);

//ListModel<String> model=MainForm.readylist.getModel();

try {

MainForm.readycontent.removeElement(pcb.getOuttitle());

MainForm.nowproc.setText(pcb.getOuttitle());

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

String noworder=pcb.getIr().getNowpoint();

Integer x=pcb.getDr().getX();

**5 其他功能（其他代码）**

内存分配及整理：

package util;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;

import os.MainForm;

import os.OS;

/\*\*\* @author hp

\* @date : 2018年12月8日 下午2:43:05\*/

public class CatchUtil {

//内存分配，返回内存表中起始位置

public static int catchAllocate(String []orders)throws Exception

{

String strs[]=OS.catches;

int length=orders.length;

Map<Integer, Integer> map=OS.emptycatch;

Set<Integer> keys= map.keySet();

int mink=-1,minv=129;

Boolean isinsert=false;

for(Integer key:keys)

{

int value=map.get(key);

if(value>=length&&value<minv)

{

mink=key;

minv=value;

isinsert=true;

}

}

for (int i = 0; i < length; i++) {

strs[mink+i]=orders[i];

}

int value=map.get(mink);

map.remove(mink);

map.put(mink+length, value-length);

OS.emptycatch=map;

OS.fullcatch.put(mink, length);

if(!isinsert)

{

return -1;

}

catchMapping();

return mink;

}

/\*\*

\* @param start 开始位置

\* @param length 长度

\* \*/

public static void catchRecycle(int start,int length) throws Exception

{

//重置内存

String strs[]=OS.catches;

for(int i=start;i<start+length;i++)

{

strs[i]="";

}

OS.catches=strs;

strs=OS.catches;

//删除已占内存表

OS.fullcatch.remove(start);

Map<Integer, Integer> map=OS.emptycatch;

//向后扫描

int nextkey=start+length;

if(map.containsKey(nextkey))

{

int finvalue=length+map.get(nextkey);

map.remove(nextkey);

map.put(start, finvalue);

}else

{

map.put(start, length);

}

//向前扫描

/\*Set<Integer> keys = map.keySet();

//使用迭代器、foreach时，不能再循环内部删除集合内容!!!!!!!!

int frontkey=-1;

for(Integer key : keys)

{

if(key<start&&key+map.get(key)==start)

{

int finlength=length+map.get(key);

map.remove(key);

map.put(key, finlength);

frontkey=key;

break;

}

}

if(frontkey>-1)

{

int finlength=length+map.get(frontkey);

map.remove(frontkey);

map.put(frontkey, finlength);

}\*/

//catchMapping();

OS.emptycatch=map;

}

//磁盘整理

public static void catchArrange() throws Exception

{

Map<Integer, Integer> map=OS.emptycatch;

Set<Integer> keys = map.keySet();

int arr[]=new int[129];

for(Integer integer:keys)

{

int key=integer;

int value=map.get(integer);

arr[key]=value;

}

for(int i=0;i<128;i++)

{

int k=i+arr[i];

if(arr[k]>0)

{

arr[i]+=arr[k];

arr[k]=0;

i=0;

}

}

Map<Integer, Integer> newmap=new HashMap();

for(int i=0;i<128;i++)

{

if(arr[i]>0)

{

newmap.put(i, arr[i]);

//System.out.println("arr"+i+"="+arr[i]);

}

}

OS.emptycatch=newmap;

catchMapping();

}

public static void showCatch() throws Exception

{

catchArrange();

catchArrange();

Map<Integer,Integer> map=OS.fullcatch;

Set<Integer> keys = map.keySet();

System.out.println("Full:");

for(Integer key : keys)

{

System.out.println(key+" "+map.get(key));

}

map=OS.emptycatch;

keys = map.keySet();

System.out.println("Empty:");

for(Integer key : keys)

{

System.out.println(key+" "+map.get(key));

}

}

public static void catchMapping() throws Exception

{

String strs[]=OS.catches;

int i;

MainForm.catchcontent1.clear();

MainForm.catchcontent2.clear();

MainForm.catchcontent3.clear();

MainForm.catchcontent4.clear();

for(i=0;i<32;i++)

{

if(strs[i]==null)

{

MainForm.catchcontent1.addElement(" ");

}

else {

if(strs[i].equals(""))

{

MainForm.catchcontent1.addElement(" ");

}

else {

MainForm.catchcontent1.addElement("█");

}

}

}

for(i=32;i<64;i++)

{

if(strs[i]==null)

{

MainForm.catchcontent2.addElement(" ");

}

else {

if(strs[i].equals(""))

{

MainForm.catchcontent2.addElement(" ");

}

else {

MainForm.catchcontent2.addElement("█");

}

}

}

for(i=64;i<96;i++)

{

if(strs[i]==null)

{

MainForm.catchcontent3.addElement(" ");

}

else {

if(strs[i].equals(""))

{

MainForm.catchcontent3.addElement(" ");

}

else {

MainForm.catchcontent3.addElement("█");

}

}

}

for(i=96;i<128;i++)

{

if(strs[i]==null)

{

MainForm.catchcontent4.addElement(" ");

}

else {

if(strs[i].equals(""))

{

MainForm.catchcontent4.addElement(" ");

}

else {

MainForm.catchcontent4.addElement("█");

}

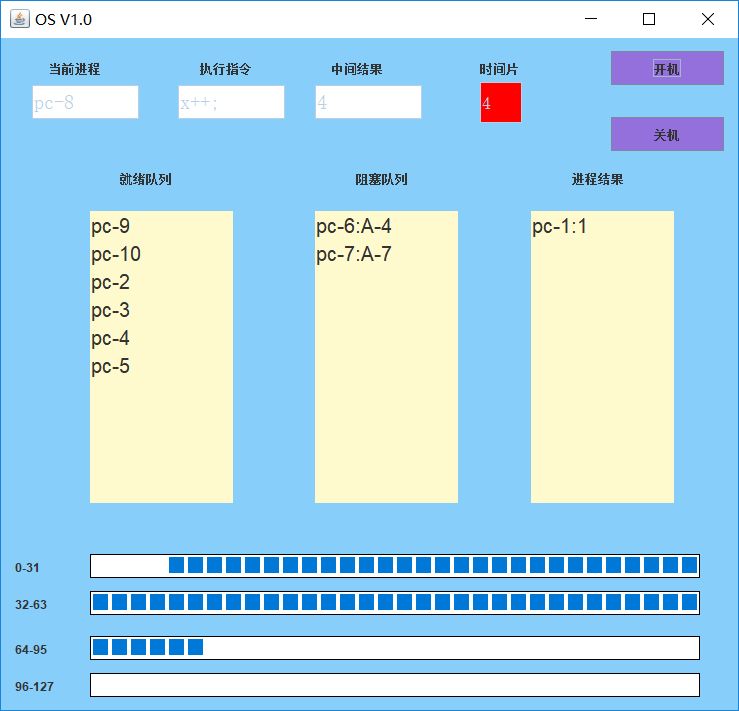
}

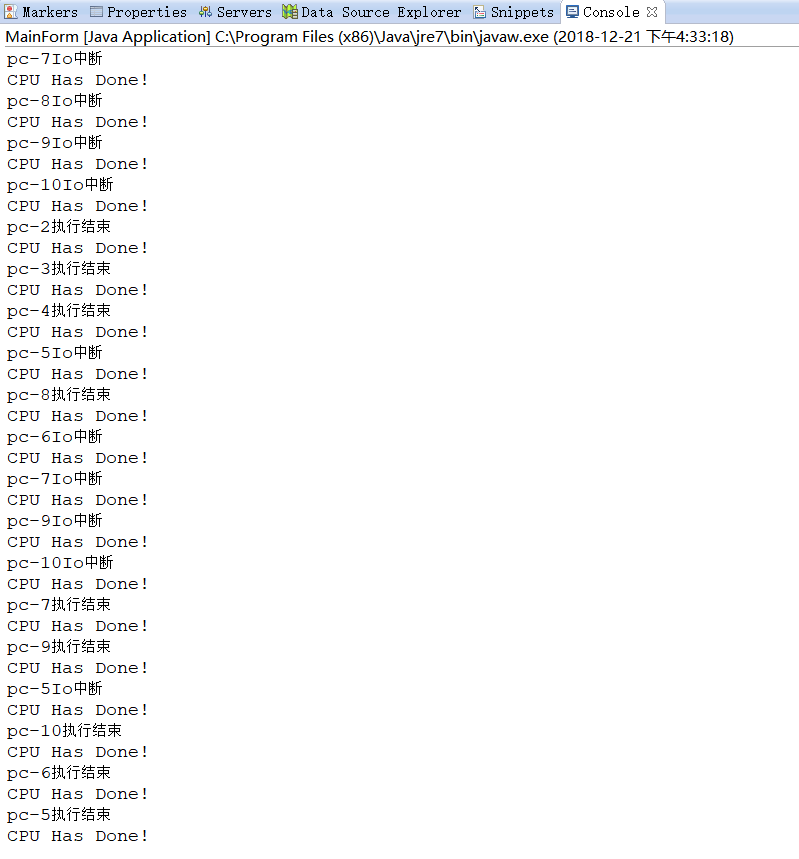
}

}

}

1. **系统测试与运行**





**7 总结**

在这次课程设计的过程中，我复习了操作系统中关于进程的知识，加深了我对操作系统内部结构的理解，而且还使我掌握JAVA语言多线程的写法，极大地锻炼了我编程的能力，收获颇丰。