**黑龙江大学**

**实 验 报 告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程名称** | **计算机操作系统** | | | | | |
| **实验项目名称** | **进程调度** | | | | | |
| **实验时间**  **（日期及节次）** | **2016年11月4日 5-6节** | | | | | |
| **专业** | **计算机** | | **学生所在学院** | | **计算机科学技术学院** | |
| **年级** | **2014级** | | **学号** | | **20123349** | |
| **姓名** | **刘伟** | | **指导教师** | | **朱敬华** | |
| **实验室名称** |  | | | | | |
| **实验成绩** | **预习情况** | **操作技术** | **实验报告** | **附加：综合创新能力** | | **实验**  **综合成绩** |
|  |  |  |  | |  |
| **教师签字** |  | | | | | |

**黑龙江大学教务处**

**实验五 进程调度管理**

**一．实验名称：**进程调度管理

**二．实验目的：**

在实验一的基础上实现先来先服务FCFS、短作业优先SJF以及时间片轮转调度算法。

**三．实验内容：**

我们可以根据创建进程的系统时钟，取相对时钟作为进程的到达时间，利用随机数产生每个进程的估计运行时间。利用模拟系统中提供的算法分别计算其相应的周转时间和带权周转时间。

具体过程：

1. 利用绝对时间和相对时钟产生一组进程的到达时刻和运行时间。
2. 实现FCFS算法：根据进程的到达时间的先后次序来完成对若干进程的调度。
3. 实现SJF算法：根据当前时间已经到达进程的需要运行时间选取其中时间最小的进程最先运行。
4. 实现时间片轮转算法：首先要求确定时间片的大小，依据进程的到达时间依次加入队列，每次分配一个时间片大小的时间，如果没有完成参与下一次的竞争，当最后需要一个小于等于时间片的时间时本进程完成，同时退出队列。

计算每种算法调度后，系统的平均周转时间和平均带权周转时间。

**四．实验代码**

var processArr = [];

var x = [];

function change(){

processArr.forEach(function (item,index) {

item.flag = true;

});

}

processArr = [

{

name: 'A',

arrive: 0,

server: 4

},

{

name: 'B',

arrive: 1,

server: 3

},

{

name: 'C',

arrive: 2,

server: 5

},

{

name: 'D',

arrive: 3,

server: 2

},

{

name: 'E',

arrive: 4,

server: 4

}

];

function FCFS(arr){

change();

var flag = false;

var timer = 0;

var temp;

var len = arr.length;

while(len > 0){

flag = false;

for(var i = 0 ; i < arr.length; i++){

if (arr[i].arrive <= timer && arr[i].flag == true){

flag = true;

temp = i;

len--;

break;

}

}

if(flag){

arr[temp].begin = timer;

arr[temp].flag = false;

timer += arr[temp].server

arr[temp].end = timer;

arr[temp].turn = arr[temp].end - arr[temp].arrive;

arr[temp].weightTurn = arr[temp].turn / arr[temp].server;

console.log(" "+arr[temp].name +" | "+" "+arr[temp].arrive +" | "+" "+arr[temp].server +" | "+" "+arr[temp].end +" | "+" "+arr[temp].turn +" | "+" "+arr[temp].weightTurn);

}else{

timer ++;

}

}

}

function SJF(arr){

change();

var flag = false;

var timer = 0;

var temp;

var len = arr.length;

var joblen = 100;

while(len > 0){

joblen = 100;

flag = false;

for(var i = 0 ; i < arr.length; i++){

if (arr[i].arrive <= timer && arr[i].flag == true){

flag = true;

if(arr[i].server<joblen){

joblen = arr[i].server;

temp = i;

}

}

}

if(flag){

len--;

arr[temp].begin = timer;

arr[temp].flag = false;

timer += arr[temp].server

arr[temp].end = timer;

arr[temp].turn = arr[temp].end - arr[temp].arrive;

arr[temp].weightTurn = arr[temp].turn / arr[temp].server;

console.log(" "+arr[temp].name +" | "+" "+arr[temp].arrive +" | "+" "+arr[temp].server +" | "+" "+arr[temp].end +" | "+" "+arr[temp].turn +" | "+" "+arr[temp].weightTurn);

}else{

timer ++;

}

}

}

function RR(arr,num){

change();

var lenArr = [];

var len = 0;

var recoder = [];

arr.forEach(function (item,index){

lenArr.push(item.server);

len += item.server;

});

console.log(arr[0].beign);

var timer = 0;

while(len>0){

var flag1 = true;

arr.forEach(function (item,index){

if(item.flag == true){

flag1 = false;

}

});

flag = false;

var temp;

for(var i = 0 ; i < arr.length; i++){

if (arr[i].arrive <= timer && arr[i].flag == true){

flag = true;

temp = i;

break;

}

}

if(flag||flag1){

if(flag1){

for(var i = 0 ; i<lenArr.length; i++){

if(lenArr[i] > 0){

var t = lenArr[i]>num? num : lenArr[i];

lenArr[i]-=t;

// console.log(lenArr[i]);

timer+=t;

len-=t;

for(var j = 0; j<t;j++){

// console.log(arr[index].name);

recoder.push(arr[i].name);

}

}

}

}

arr[temp].flag = false;

var t = lenArr[temp]>num?num:lenArr[temp];

lenArr[temp]-=t;

timer+=t;

len-=t;

for(var j = 0; j<t;j++){

recoder.push(arr[temp].name);

}

}else{

timer++;

recoder.push(' ');

}

}

arr.forEach(function (item,index){

item.turn = recoder.lastIndexOf(item.name)+1-item.arrive;

item.weightTurn = item.turn/item.server;

console.log(" "+item.name +" | "+" "+item.arrive +" | "+" "+item.server +" | "+" "+item.end +" | "+" "+item.turn +" | "+" "+item.weightTurn);

});

console.log(recoder);

return recoder;

}

function RRR(){

var str = 'ABCDEABCDEABCEACE';

var arr = str.split('');

x = arr;

processArr.forEach(function(item,index,array){

item.end = arr.lastIndexOf(item.name)+1;

item.turn = arr.lastIndexOf(item.name)+1-item.arrive;

item.weightTurn = item.turn/item.server;

console.log(" "+item.name +" | "+" "+item.arrive +" | "+" "+item.server +" | "+" "+item.end +" | "+" "+item.turn +" | "+" "+item.weightTurn);

});

}

var readline = require('readline');

var rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

rl.setPrompt("OS Demo > ");

rl.prompt();

rl.on('line',function(input){

var shell = input.trim();

switch(shell){

case 'HELP':

console.log("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"+"\n-HELP ---- 帮助信息\n-FCFS ---- 先来先服务\n-SJF ---- 短作业优先\n-RR ---- 时间片轮转\n-EXIT ---- 退出\n"+"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 'FCFS':

console.log("\n 进程名称 | 达到时间 | 服务时间 | 完成时间 | 周转时间 | 带权周转时间 | \n");

FCFS(processArr);

break;

case 'SJF':

console.log("\n 进程名称 | 达到时间 | 服务时间 | 完成时间 | 周转时间 | 带权周转时间 | \n");

SJF(processArr);

break;

case 'RR':

console.log("\n 进程名称 | 达到时间 | 服务时间 | 完成时间 | 周转时间 | 带权周转时间 | \n");

RRR();

console.info();

console.log(" "+x.join(' '));

break;

case 'EXIT':

console.log("\n 程序已经退出！ \n");

rl.close();

process.exit(0);

break;

default:

console.log("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"+

"\* 操作系统进程调度实验演示！输入'HELP'查看更多帮助信息！ \*"+

"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

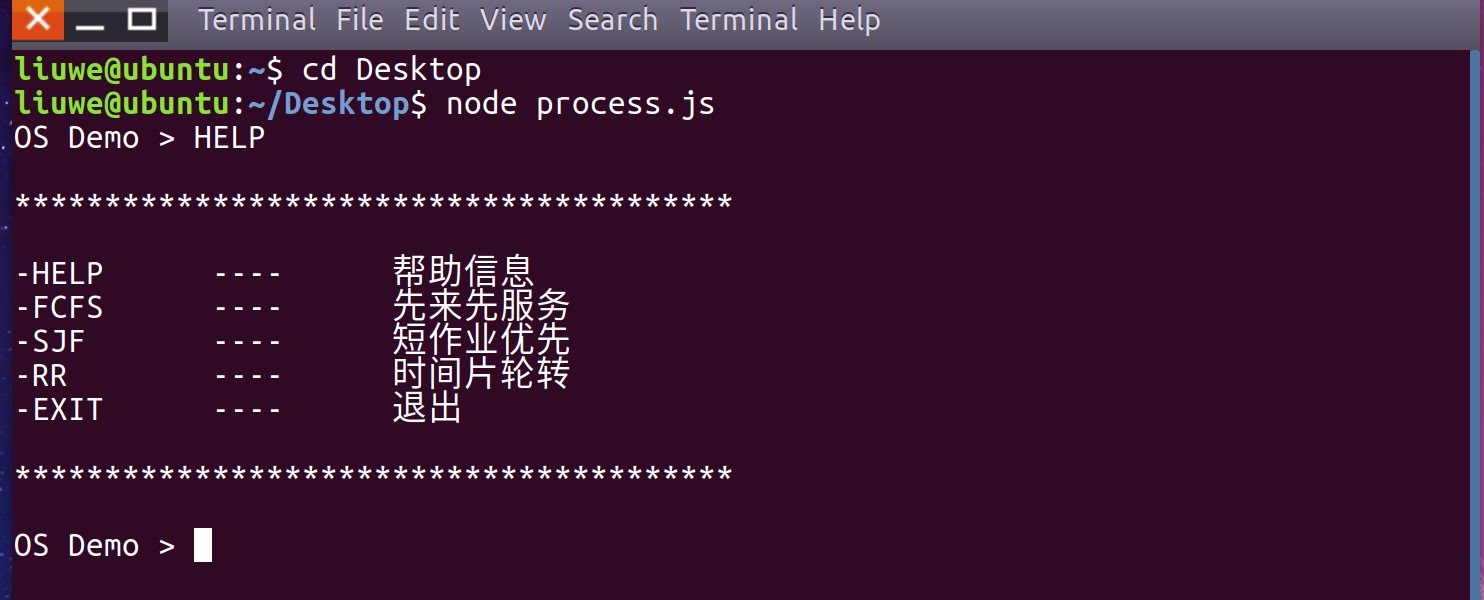
);

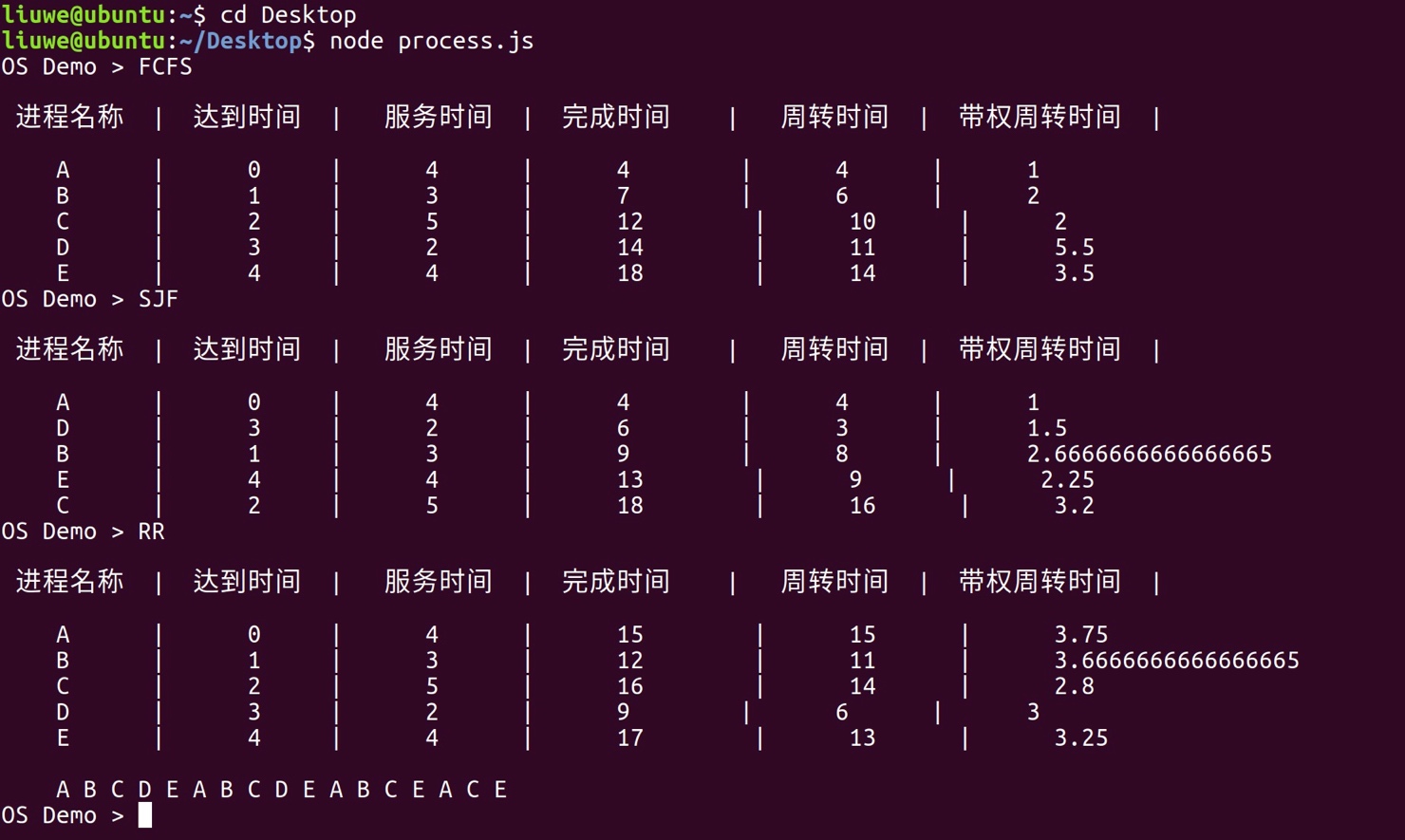
}

rl.prompt();

});

**五．实验截图：**

****



**六．实验心得：**

通过本次实验，了解了进程调度的基本方式和三种进程调度算法和实现方式。了解进程调度三种算法的区别和实现方式。