

操作系统实验报报告



学 院（系）： 电子信息与电气工程学部

专 业： 计算机科学与技术

班 级： 电计1704

学 生 姓 名： 谢玉宁

学 号： 201792260

完 成 日 期： 2020年6月13日

Ps.本报告含索引导航，方便批阅

# 实验一、进程管理

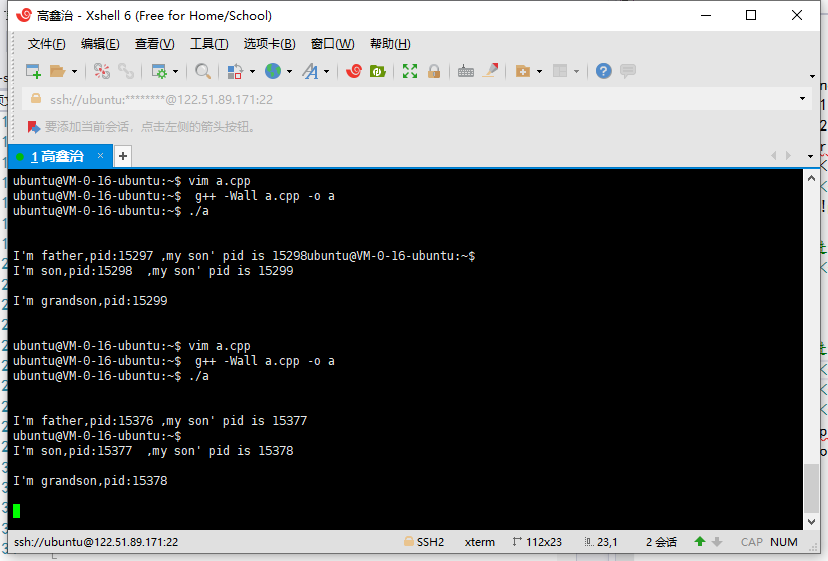
## 实验目的

加深对于进程并发执行概念的理解。实践并发进程的创建和控制方法。观察和体验进程的动态特性。进一步理解进程生命期期间创建、变换、撤销状态变换的过程。掌握进程控制的方法，了解父子进程间的控制和协作关系。练习Linux系统中进程创建与控制有关的系统调用的编程和调试技术。

## 运行结果展示

1. 父进程-子进程-子子进程

每个进程都执行自己独立的程序，打印自己的pid，每个父进程打印其子进程的pid;



1. 代码

#include <iostream>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

int main()

{

cout << endl;

pid\_t pid = -1;

pid\_t spid = -1;

pid = fork();

if (pid < 0)

cout << "create son failed" << endl;

else if (!pid)

{

pid\_t spid = -1;

spid = fork();

if(spid<0)

cout << "create grandson failed" << endl;

else if (!spid)

{

//孙子进程，打印pid

cout << "I'm grandson,pid:" << getpid() <<endl<<endl;

}

else

{

//儿子进程,打印pid

cout << "I'm son,pid:" << getpid();

cout << " ,my son' pid is " << spid << endl;

}

}

else

{

//父进程打印pid

cout << "I'm father,pid:" << getpid();

cout << " ,my son' pid is " << pid << endl;

if (spid != -1)

cout << " ,my grandson' pid is " << spid << endl;

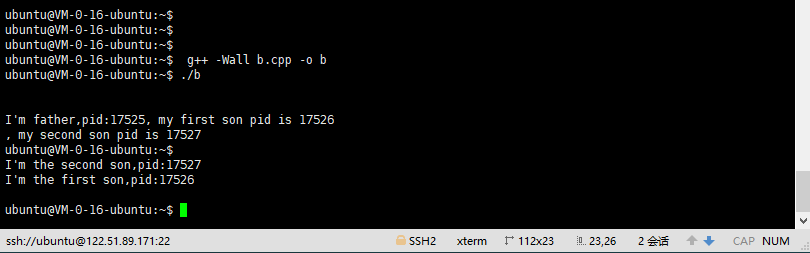
}

return 0;

}

1. 父进程-子进程1、子进程2

每个进程都执行自己独立的程序，打印自己的pid，父进程打印其子进程的pid;



### 2) 代码

#include <sys/types.h>

#include <sys/wait.h>

#include <iostream>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

int main()

{

cout << endl;

pid\_t pid1 = -1;

pid\_t pid2 = -1;

pid1 = fork();

if (pid1 < 0)

cout << "create son failed" << endl;

else if (!pid1)

{

//子进程

cout << "I'm the first son,pid:" << getpid() << endl;

}

else

{

pid2 = fork();

if (pid2 < 0)

cout << "create son failed" << endl;

else if (!pid2)

{

cout << endl;

cout<< "I'm the second son,pid:" << getpid() << endl;

}

else

{

//父进程打印pid

cout << endl;

cout << "I'm father,pid:" << getpid();

cout << ", my first son pid is " << pid1 << endl;

cout << ", my second son pid is " << pid2 << endl;

}

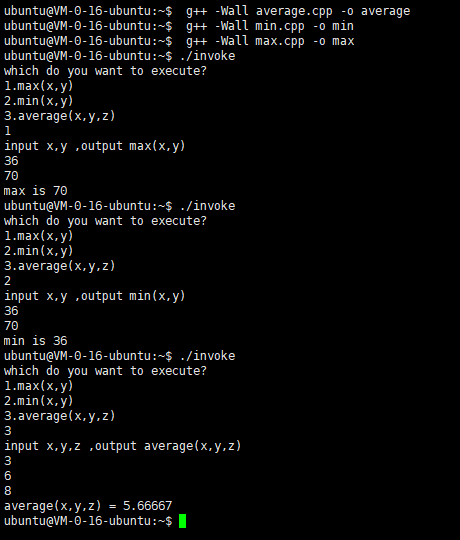
}

return 0;

}

1. exec函数族

编写一个命令处理程序，能处理max(m,n), min(m,n)和 average(m,n,l)这几个命令。（使用exec函数族）



### 3) 代码

**invoke.cpp**

#include <unistd.h>

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

using namespace std;

int main(int argc, char\* argv[])

{

int ans = 0;

char\* args[] = { NULL, NULL };

//char\* path[] = { NULL };

char max[10] = "max";

char min[10] = "min";

char average[10] = "average";

cout << "which do you want to execute?"<<endl<<

"1.max(x,y) \n2.min(x,y)\n3.average(x,y,z)\n";

int want = 0;

cin >> want;

switch (want)

{

case 1:

args[0] = max;

break;

case 2:

args[0] = min;

break;

case 3:

args[0] = average;

break;

default:

cout << "wrong choice!" << endl;

break;

}

while(1)

{

ans = execve(args[0], args, NULL);

cout << ans;

}

return 0;

}

**max.cpp**

#include <iostream>

#include <string>

using namespace std;

int main(int argc, char\* argv[]) {

int x = 0;

int y = 0;

char xstr[10] = {};

char ystr[10] = {};

cout << "input x,y ,output max(x,y)"<<endl;

cin.getline(xstr, 10);

cin.getline(ystr, 10);

sscanf(xstr, "%d", &x);

sscanf(ystr, "%d", &y);

cout << "max is " << (x > y ? x : y) << endl;

return 0;

}

**min.cpp**

#include <iostream>

#include <string>

using namespace std;

int main(int argc, char\* argv[]) {

int x = 0;

int y = 0;

char xstr[10] = {};

char ystr[10] = {};

cout << "input x,y ,output min(x,y)"<<endl;

cin.getline(xstr, 10);

cin.getline(ystr, 10);

sscanf(xstr, "%d", &x);

sscanf(ystr, "%d", &y);

cout << "min is " << (x > y ? y : x) << endl;

return 0;

}

**average.cpp**

#include <iostream>

#include <string>

using namespace std;

int main(int argc, char\* argv[]) {

int x = 0;

int y = 0;

int z = 0;

char xstr[10] = {};

char ystr[10] = {};

char zstr[10] = {};

cout << "input x,y,z ,output average(x,y,z)"<<endl;

cin.getline(xstr, 10);

cin.getline(ystr, 10);

cin.getline(zstr, 10);

sscanf(xstr, "%d", &x);

sscanf(ystr, "%d", &y);

sscanf(zstr, "%d", &z);

cout << "average(x,y,z) = " << float(x+y+z)/3 << endl;

return 0;

}

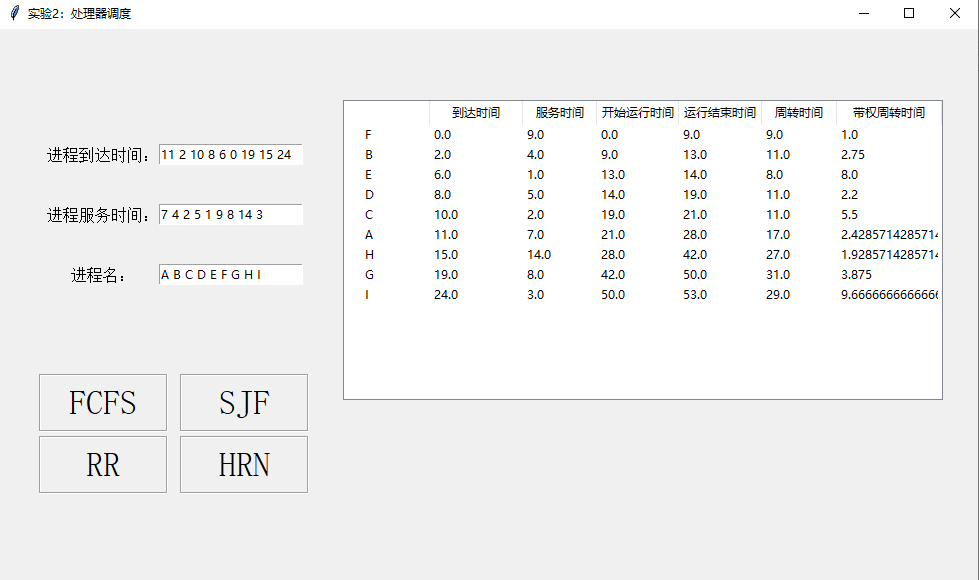
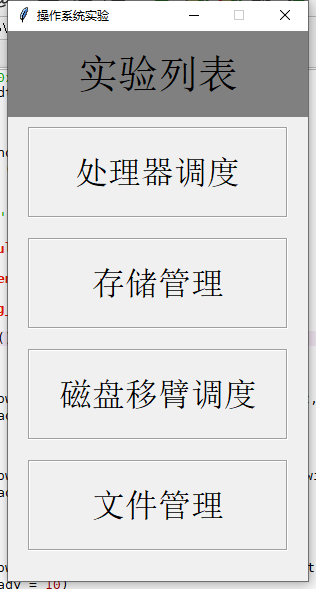
# 实验二、处理器调度

## 实验目的

熟悉使用各种单处理器调度的各种算法，加深对于处理机调度机制的理解。练习模拟算法的编程技巧，锻炼分析试验数据的能力。

## 运行结果展示

### 界面



输入进程到达时间序列：11 2 10 8 6 0 19 15 24

输入进程服务时间序列：7 4 2 5 1 9 8 14 3

输入进程名字：A B C D E F G H I

各个处理器调度算法运行结果如下图所示：

### FCFS



### SJF



### RR(q=1)



### HRN:



## 算法核心代码

*# -\*- coding: utf-8 -\*-  
"""  
Created on Thu Jun 11 09:34:49 2020  
  
@author: Lenovo  
"""  
  
# 处理机调度***import** pandas **as** pd  
*# 数据结构为DataFrame***def** DFrame\_process(arrival\_time, service\_time, process\_name=**None**):  
 process = pd.DataFrame({**'arrival\_time'**: pd.Series(arrival\_time),  
 **'service\_time'**: pd.Series(service\_time),  
 **'run\_start\_time'**: 0,  
 **'run\_finish\_time'**: 0,  
 **'turnaround\_time'**: 0,  
 **'turnaround\_time\_with\_weight'**: 0})  
 process.index = process\_name  
 **return** process  
  
  
**def** FCFS(process\_):  
 process = process\_.copy()  
 process\_num = len(process)  
 process.sort\_values(by=[**'arrival\_time'**, **'service\_time'**], inplace=**True**)  
  
 ready\_que = []  
 finish\_que = []  
 time\_dic = {}  
  
 time = 0  
 processor\_on = **False** p\_running = **None  
 while** (len(finish\_que) < process\_num):  
 *# 判断是否有进程进入就绪队列,并更新就绪队列* arrival\_p = process[process[**'arrival\_time'**] == time].index  
 **if** len(arrival\_p):  
 ready\_que.extend(arrival\_p)  
  
 time\_dic[time] = ready\_que.copy()  
  
 *# 判断是否有进程要结束* **if** processor\_on **and** time == process.loc[p\_running][**'run\_finish\_time'**]:  
 processor\_on = **False** finish\_que.extend(p\_running)  
 print(**"time:%d %s finish"** % (time, p\_running))  
  
 *# 判断是否有进程要运行，更新就绪队列* **if not** processor\_on **and** len(ready\_que):  
 p\_running = ready\_que[0]  
 print(**"time:%d %s start"** % (time, p\_running))  
 processor\_on = **True** process.loc[p\_running][**'run\_start\_time'**] = time  
 process.loc[p\_running][**'run\_finish\_time'**] = time + process.loc[p\_running][**'service\_time'**]  
 **del** ready\_que[0]  
  
 time\_dic[time] = ready\_que.copy()  
  
 time += 1  
 *# 计算周转时间* process[**'turnaround\_time'**] = process[**'run\_finish\_time'**] - process[**'arrival\_time'**]  
 process[**'turnaround\_time\_with\_weight'**] = process[**'turnaround\_time'**] / process[**'service\_time'**]  
  
 **return** process, time\_dic  
  
  
**def** SJF(process\_):  
 process = process\_.copy()  
  
 process\_num = len(process)  
 process.sort\_values(by=[**'arrival\_time'**, **'service\_time'**], inplace=**True**)  
  
 ready\_que = []  
 finish\_que = []  
 time\_dic = {}  
  
 time = 0  
 processor\_on = **False** p\_running = **None  
 while** (len(finish\_que) < process\_num):  
 *# 判断是否有进程进入就绪队列,并更新就绪队列* arrival\_p = process[process[**'arrival\_time'**] == time].index  
 **if** len(arrival\_p):  
 ready\_que.extend(arrival\_p)  
  
 time\_dic[time] = ready\_que.copy()  
  
 *# 判断是否有进程要结束* **if** processor\_on **and** time == process.loc[p\_running][**'run\_finish\_time'**]:  
 processor\_on = **False** finish\_que.extend(p\_running)  
 print(**"time:%d %s finish"** % (time, p\_running))  
  
 *# 判断是否有进程要运行，更新就绪队列* **if not** processor\_on **and** len(ready\_que):  
 p\_running = process[**'service\_time'**][ready\_que].idxmin()  
 print(**"time:%d %s start"** % (time, p\_running))  
 processor\_on = **True** process.loc[p\_running][**'run\_start\_time'**] = time  
 process.loc[p\_running][**'run\_finish\_time'**] = time + process.loc[p\_running][**'service\_time'**]  
 ready\_que.remove(p\_running)  
 time\_dic[time] = ready\_que.copy()  
  
 time += 1  
 *# 计算周转时间* process[**'turnaround\_time'**] = process[**'run\_finish\_time'**] - process[**'arrival\_time'**]  
 process[**'turnaround\_time\_with\_weight'**] = process[**'turnaround\_time'**] / process[**'service\_time'**]  
  
 **return** process, time\_dic  
  
  
**def** RR(process\_, q=1):  
 process = process\_.copy()  
 process\_num = len(process)  
 process.sort\_values(by=[**'arrival\_time'**, **'service\_time'**], inplace=**True**)  
  
 ready\_que = []  
 remain\_time = []  
  
 finish\_que = []  
 time\_dic = {}  
  
 time = 0  
 processor\_on = **False** p\_running = **None  
  
 while** (len(finish\_que) < process\_num):  
 *# 判断是否有进程进入就绪队列,并更新就绪队列* arrival\_p = process[process[**'arrival\_time'**] == time].index  
 need\_time = list(process[**'service\_time'**][arrival\_p])  
 **if** len(arrival\_p):  
 ready\_que.extend(arrival\_p)  
 remain\_time.extend(need\_time)  
  
 time\_dic[time] = ready\_que.copy()  
  
 *# 取就绪队列队首进程，运行q秒* **if not** processor\_on **and** len(ready\_que):  
 p\_running = ready\_que[0]  
 **if** process[**'service\_time'**][p\_running] == remain\_time[0]:  
 process[**'run\_start\_time'**][p\_running] = time  
  
 print(**"time:%d %s start"** % (time, p\_running))  
 processor\_on = **True** remain\_time[0] -= 1  
  
 time\_dic[time] = ready\_que.copy()  
  
 **del** ready\_que[0]  
 **if not** remain\_time[0]:  
  
 finish\_que.extend(p\_running)  
 print(**"time:%d %s finish"** % (time, p\_running))  
 process.loc[p\_running][**'run\_finish\_time'**]=time+1  
 **else**:  
 ready\_que.append(p\_running)  
 remain\_time.append(remain\_time[0])  
 time\_dic[time] = ready\_que.copy()  
  
 processor\_on = **False  
  
 del** remain\_time[0]  
  
 time += 1  
 *# process['run\_start\_time'] = process['arrival\_time']+1* process[**'turnaround\_time'**] = process[**'run\_finish\_time'**] - process[**'arrival\_time'**]  
 process[**'turnaround\_time\_with\_weight'**] = process[**'turnaround\_time'**] / process[**'service\_time'**]  
  
 **return** process, time\_dic  
  
  
**def** HRN(process\_):  
 process = process\_.copy()  
  
 process\_num = len(process)  
 process.sort\_values(by=[**'arrival\_time'**, **'service\_time'**], inplace=**True**)  
  
 ready\_que = []  
 finish\_que = []  
 time\_dic = {}  
  
 time = 0  
 processor\_on = **False** p\_running = **None  
 while** (len(finish\_que) < process\_num):  
 *# 判断是否有进程进入就绪队列,并更新就绪队列* arrival\_p = process[process[**'arrival\_time'**] == time].index  
 **if** len(arrival\_p):  
 ready\_que.extend(arrival\_p)  
 time\_dic[time] = ready\_que.copy()  
  
 *# 判断是否有进程要结束* **if** processor\_on **and** time == process.loc[p\_running][**'run\_finish\_time'**]:  
 processor\_on = **False** finish\_que.extend(p\_running)  
 print(**"time:%d %s finish"** % (time, p\_running))  
  
 *# 判断是否有进程要运行，更新就绪队列* **if not** processor\_on **and** len(ready\_que):  
 R = time - process[**'arrival\_time'**][ready\_que] / process[**'service\_time'**][ready\_que]  
 p\_running = R.idxmax()  
 print(**"time:%d %s start"** % (time, p\_running))  
 processor\_on = **True** process.loc[p\_running][**'run\_start\_time'**] = time  
 process.loc[p\_running][**'run\_finish\_time'**] = time + process.loc[p\_running][**'service\_time'**]  
 ready\_que.remove(p\_running)  
 time\_dic[time] = ready\_que.copy()  
  
 time += 1  
 *# 计算周转时间* process[**'turnaround\_time'**] = process[**'run\_finish\_time'**] - process[**'arrival\_time'**]  
 process[**'turnaround\_time\_with\_weight'**] = process[**'turnaround\_time'**] / process[**'service\_time'**]  
  
 **return** process, time\_dic  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 arrival\_time = [4, 2, 0, 6, 8]  
 service\_time = [4, 6, 3, 5, 2]  
 process\_name = [**'A'**, **'B'**, **'C'**, **'D'**, **'E'**]  
  
 process = DFrame\_process(arrival\_time, service\_time, process\_name)  
 process, time\_dic = HRN(process)

# 实验三、存储管理

## 实验目的

加深对于存储管理的了解，掌握虚拟存储器的实现原理；观察和了解重要的页面置换算法和置换过程。练习模拟算法的编程技巧，锻炼分析试验数据的能力。

## 界面

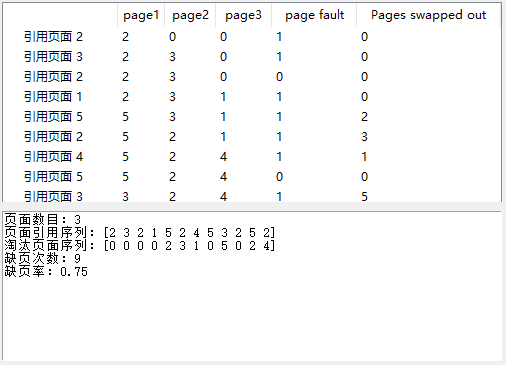


输入引用逻辑页面序列：2,3,2,1,5,2,4,5,3,2,5,2

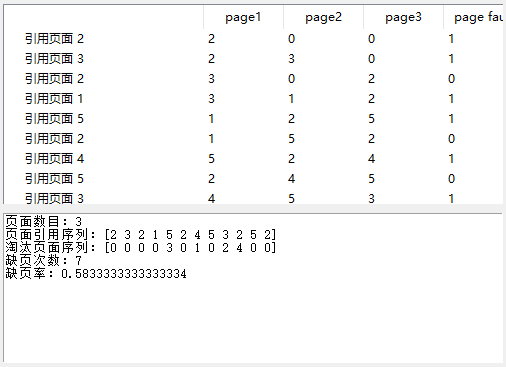
页面数目：3

## 运行结果展示

### FIFO



### LRU



## 算法核心代码

*# -\*- coding: utf-8 -\*-  
"""  
Created on Thu Jun 11 16:21:23 2020  
  
@author: Lenovo  
"""***import** numpy **as** np  
  
*#内存管理类***class** Storage\_manag():  
 **def** \_\_init\_\_(self, seq, frame\_num):  
 self.frame\_num = frame\_num  
 self.page\_seq = np.array(seq)  
 self.page\_faults = np.zeros\_like(self.page\_seq)  
 self.page\_out = np.zeros\_like(self.page\_seq)  
 self.page\_table = np.zeros((len(self.page\_seq), self.frame\_num), dtype=np.int32)  
 self.page\_faults[0] = 1  
 self.page\_table[0][0] = self.page\_seq[0]  
  
 self.page\_fault\_num = **None** self.page\_fault\_ratio = **None  
  
 def** FIFO(self):  
 count = 0  
 **for** i **in** range(1, len(self.page\_seq)):  
  
 page = self.page\_seq[i]  
 self.page\_table[i] = self.page\_table[i - 1].copy()  
 *#缺页，置换页面，设置缺页标记* **if** page **not in** self.page\_table[i]:  
 **if** (self.page\_table[i] == 0).any():  
 self.page\_table[i][self.page\_table[i].argmin()] = page  
 self.page\_faults[i] = 1  
 **continue** self.page\_faults[i] = 1  
 self.page\_out[i] = self.page\_table[i][count]  
 self.page\_table[i][count] = page  
 count = (count + 1) % self.frame\_num  
  
 *# statistics* self.page\_fault\_num = sum(self.page\_faults)  
 self.page\_fault\_ratio = self.page\_fault\_num / len(self.page\_seq)  
  
 **def** LRU(self):  
  
 **for** i **in** range(1, len(self.page\_seq)):  
 page = self.page\_seq[i]  
 self.page\_table[i] = self.page\_table[i - 1].copy()  
 *#缺页，置换页面，设置缺页标记* **if** page **not in** self.page\_table[i]:  
 **if** (self.page\_table[i] == 0).any():  
 self.page\_table[i][self.page\_table[i].argmin()] = page  
 self.page\_faults[i] = 1  
 **continue** self.page\_faults[i] = 1  
 self.page\_out[i] = self.page\_table[i][0]  
 self.page\_table[i][0] = page  
 lis = list(self.page\_table[i])  
 lis.append(lis[0])  
 **del** lis[0]  
 self.page\_table[i] = np.array(lis, dtype=np.int32)  
  
 **else**:  
 *# 没有缺页，更新页面最晚引用次序* idx = np.where(self.page\_table[i] == page)[0][0]  
 lis = list(self.page\_table[i])  
 lis.append(lis[idx])  
 **del** lis[idx]  
 self.page\_table[i] = np.array(lis, dtype=np.int32)  
  
 *# statistics* self.page\_fault\_num = sum(self.page\_faults)  
 self.page\_fault\_ratio = self.page\_fault\_num / len(self.page\_seq)  
  
 **def** show\_info(self):  
   
 print(**"页面数目："**, self.frame\_num)  
 print(**"页面引用序列："**, self.page\_seq)  
 print(**"淘汰页面序列："**, self.page\_out)  
 print(**"缺页次数："**, self.page\_fault\_num)  
 print(**"缺页率："**, self.page\_fault\_ratio)  
 *#方便输出text* **def** info\_text(self):  
 info = **"页面数目：{}\n页面引用序列：{}\n淘汰页面序列：{}\n缺页次数：{}\n缺页率：{}\n"**.format(  
 self.frame\_num, self.page\_seq, self.page\_out, self.page\_fault\_num, self.page\_fault\_ratio)  
  
 **return** info  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 seq = [4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5]  
 frame\_num = 3  
 *# a = Storage\_manag(seq,3)  
 # a.FIFO()  
 # a.show\_info()* a = Storage\_manag(seq, 3)  
 a.LRU()  
 a.show\_info()

# 实验四、磁盘移臂调度算法实验

## 实验目的

加深对于操作系统设备管理技术的了解，体验磁盘移臂调度算法的重要性；掌握几种重要的磁盘移臂调度算法，练习模拟算法的编程技巧，锻炼研究分析试验数据的能力。

## 实验说明

1. 示例实验程序中模拟两种磁盘移臂调度算法：SSTF算法和SCAN算法

2. 能对两种算法给定任意序列不同的磁盘请求序列，显示响应磁盘请求的过程。

3. 能统计和报告不同算法情况下响应请求的顺序、移臂的总量。

## 运行结果展示

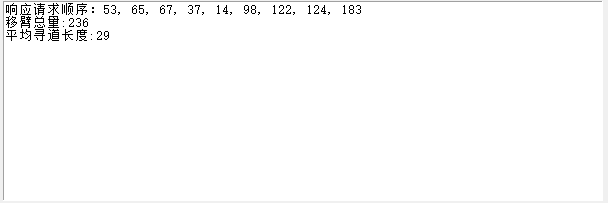
### 界面

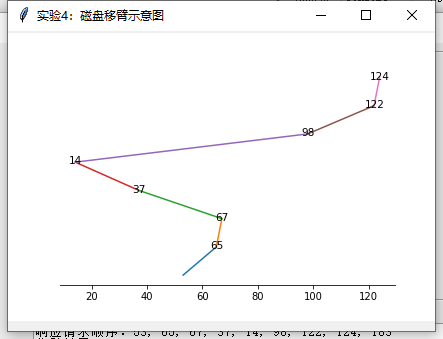


输入磁盘请求序列：98，183，37，122，14，124，65，67

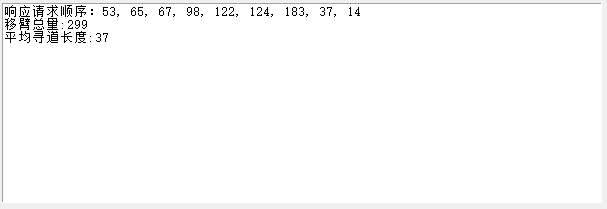
初始位置：53

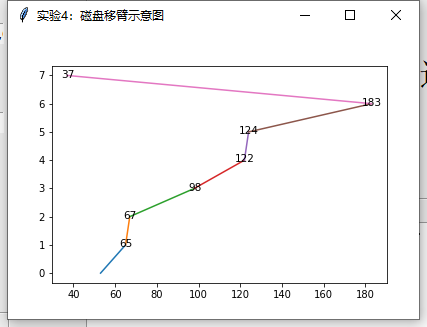
### SSTF





### SCAN





## 算法核心代码

*# -\*- coding: utf-8 -\*-  
"""  
Created on Thu Jun 11 21:47:31 2020  
  
@author: Lenovo  
"""***import** numpy **as** np  
  
seq = [90, 58, 55, 39, 38, 18, 150, 160, 184]  
init\_pos = 100  
  
  
**def** SSTF(init\_pos, seq):  
 move\_seq = [init\_pos]  
 pos = init\_pos  
 DR\_seq = sorted(seq)  
  
 **while** (len(DR\_seq)):  
 dis = [abs(ad - pos) **for** ad **in** DR\_seq]  
 next\_move = np.argmin(dis)  
 move\_seq.append(DR\_seq[next\_move])  
 pos = DR\_seq[next\_move]  
 **del** DR\_seq[next\_move]  
  
 MAL = sum([abs(move\_seq[i + 1] - move\_seq[i]) **for** i **in** range(len(seq))])  
 ARL = MAL / len(seq)  
  
 *# 顺序，移臂总量，平均寻道长度* **return** move\_seq, MAL, ARL  
  
  
**def** SCAN(init\_pos, seq, inc\_derection=**True**):  
 move\_seq = []  
  
 DR\_seq = seq.copy()  
 DR\_seq.append(init\_pos)  
 DR\_seq = sorted(DR\_seq)  
  
 b = [\_ == init\_pos **for** \_ **in** DR\_seq]  
 **if** inc\_derection:  
  
 idx = np.argmax(b)  
 move\_seq.extend(DR\_seq[idx:].copy())  
 move\_seq.extend(DR\_seq[idx - 1::-1].copy())  
  
 **else**:  
 idx = np.argmax(b) + sum(b)  
 move\_seq.extend(DR\_seq[idx - 1::-1])  
 move\_seq.extend(DR\_seq[idx:])  
 MAL = sum([abs(move\_seq[i + 1] - move\_seq[i]) **for** i **in** range(len(move\_seq) - 1)])  
 ARL = MAL / (len(move\_seq) - 1)  
  
 **return** move\_seq, MAL, ARL  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 move\_seq, MAL, ARL = SCAN(init\_pos, seq)

# 实验五、文件管理作业

## 实验目的

通过模拟文件的创建、删除操作，加深对于操作系统文件管理功能的了解，练习模拟算法的编程技巧，锻炼研究分析试验数据的能力。

## 实验说明

给出一个磁盘块序列：1、2、3、……、500，初始状态所有块为空的，每块的大小为2k。选择使用**空闲表**管理空闲块。对于基于块的索引分配执行以下步骤：

1. 随机生成2k-10k的文件50个，文件名为1.txt、2.txt、……、50.txt，按照上述算法存储到模拟磁盘中。
2. 删除奇数.txt（1.txt、3.txt、……、49.txt）文件
3. 新创建5个文件（A.txt、B.txt、C.txt、D.txt、E.txt），大小为：7k、5k、2k、9k、3.5k，按照与（1）相同的算法存储到模拟磁盘中。
4. 给出文件A.txt、B.txt、C.txt、D.txt、E.txt的盘块存储状态和所有空闲区块的状态。

空闲表法：

## 运行结果展示

### 界面



1. 随机生成50个文件

随机生成文件按钮，随机生成50个文件，存储到模拟磁盘



可以看到，现在模拟磁盘空闲表储存了了一项第一空闲块为155，大小为346块的空闲区域，磁盘信息如下：

totle block num: 500

block size: 2k

files num: 50

free blocks num: 346

1. 删除奇数.txt文件

删除奇数文件按钮，删除名为奇数.txt文件，更新空闲表：



可以看到，因为50个文件连续存储，现在模拟磁盘空闲表分裂成25块空闲区域，大小与位置在表中显示，磁盘信息如下：

totle block num: 500

block size: 2k

files num: 25

free blocks num: 425

文件数量变成了25个，空闲块变多。

1. 创建5个文件

新创建5个文件（A.txt、B.txt、C.txt、D.txt、E.txt），大小为：7k、5k、2k、9k、3.5k，按照与（1）相同的算法存储到模拟磁盘中。



通过创建按钮，创建5个文件，空闲块减少为409个，磁盘信息如下：

totle block num: 500

block size: 2k

files num: 30

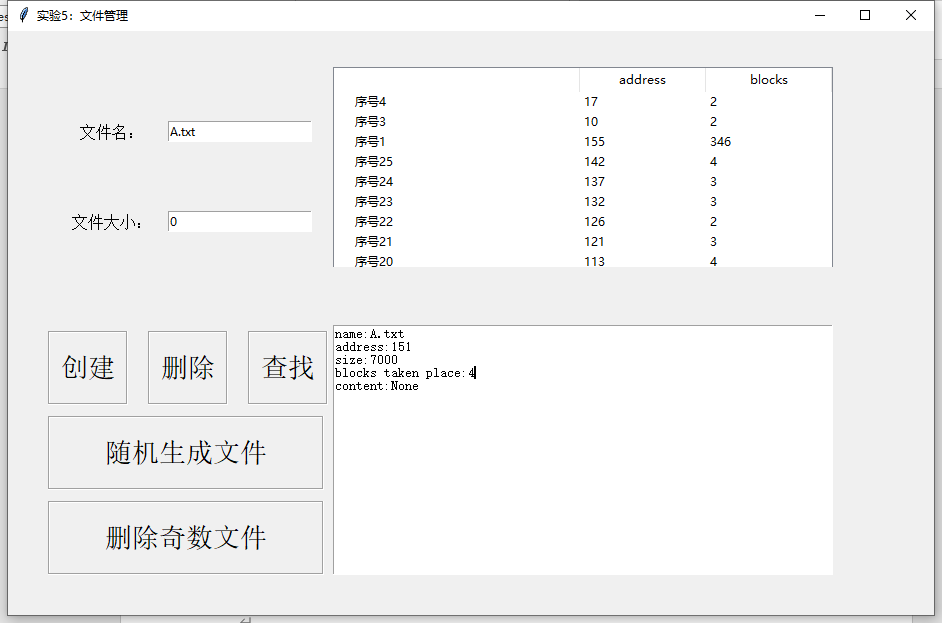
free blocks num: 409

文件变为30个

1. 查看文件盘块存储状态与空闲块状态

给出文件A.txt、B.txt、C.txt、D.txt、E.txt的盘块存储状态和所有空闲区块的状态。

通过查找按钮，找出A.txt、B.txt、C.txt、D.txt、E.txt的盘块：



name:A.txt

address:151

size:7000

blocks taken place:4

content:None



name:B.txt

address:7

size:5000

blocks taken place:3

content:None



name:C.txt

address:3

size:2000

blocks taken place:1

content:None



name:D.txt

address:21

size:9000

blocks taken place:5

content:None



name:E.txt

address:15

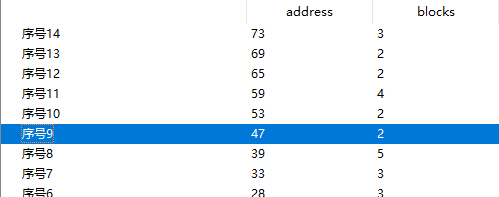
size:3500

blocks taken place:2

content:None

所有空闲区块的状态：

如空闲表所示，空闲区状态：



## 算法核心代码

*# -\*- coding: utf-8 -\*-  
"""  
Created on Thu Jun 11 23:06:00 2020  
  
@author: Lenovo  
"""***import** random  
  
  
**class** disk\_block():  
 **def** \_\_init\_\_(self, size=2000):  
 self.size = size  
 self.is\_empty = **True  
  
 def** taken(self):  
 self.is\_empty = **False  
  
 def** free\_up(self):  
 self.is\_empty = **True  
  
  
class** file():  
 **def** \_\_init\_\_(self, name, size, content=**None**):  
 self.name = name  
 self.size = size  
 self.address = **None** self.content = content  
  
 **def** save(self, address):  
 self.address = address  
  
 **def** write(self, content):  
 self.content = content  
  
 **def** show\_info(self):  
 print(**"name:"**, self.name)  
 print(**"address:"**, self.address)  
  
 print(**"size:"**, self.size)  
 print(**"content:"**, self.content)  
  
 **def** info\_text(self):  
 info = **"name:{}\naddress:{}\nsize:{}\ncontent:{}\n"**.format(  
 self.name, self.address, self.size, self.content)  
 **return** info  
  
  
**class** disk\_manager():  
 **def** \_\_init\_\_(self, block\_num=500, block\_size=2000):  
 self.block\_num = block\_num  
 self.block\_size = block\_size  
 self.vacancy\_table = {1: [1, 500]}  
 self.files\_num = 0  
  
 self.free\_blocks\_num = block\_num  
 self.directory = {}  
  
 **def** first\_fit(self, file):  
 **for** seq **in** self.vacancy\_table:  
 free\_chunk = self.vacancy\_table[seq]  
 block\_num = file.size // self.block\_size + 1  
 **if** free\_chunk[1] >= block\_num:  
 free\_chunk = self.vacancy\_table[seq].copy()  
 start\_block = free\_chunk[0] + block\_num  
 update\_free\_block = free\_chunk[1] - block\_num  
 **del** self.vacancy\_table[seq]  
  
 **if** update\_free\_block:  
 self.vacancy\_table[seq] = [start\_block, update\_free\_block]  
 **return** free\_chunk[0]  
  
 **else**:  
  
 **return False  
  
 def** Create\_file(self, name, size, content=**None**):  
 f = file(name, size, content)  
 **if** name **in** self.directory:  
 print(**"There has been a file with same name."**)  
 **return False  
  
 else**:  
 f = file(name, size)  
  
 blocks = size // self.block\_size + 1  
  
 address = self.first\_fit(f)  
  
 **if** address:  
 f.save(address)  
 self.directory[name] = [address, blocks, f]  
 print(**"Successfully created: address:%s blocks, size:%s blocks."** % (address, blocks))  
 self.files\_num += 1  
 self.free\_blocks\_num -= blocks  
 **return True  
 else**:  
 print(**"Creation failed: insufficient storage space!"**)  
 **return False  
  
 def** Delete\_file(self, name):  
  
 **if** name **not in** self.directory:  
 print(**"No such a file"**)  
 **return False** address, size, \_ = self.directory[name]  
  
 next\_seg\_add = address + size  
  
 pre\_free = **False** next\_free = **False** pre\_free\_seq = **None** next\_free\_seq = **None** *# 判断上下是否空闲* **for** seq **in** self.vacancy\_table:  
 free\_add, blocks = self.vacancy\_table[seq]  
  
 **if** free\_add + blocks == address:  
 pre\_free = **True** pre\_free\_seq = seq  
  
 **if** free\_add == next\_seg\_add:  
 next\_free = **True** next\_free\_seq = seq  
  
 *# 上下都是空闲表* **if** pre\_free **and** next\_free:  
 update\_add = self.vacancy\_table[pre\_free\_seq][0]  
 update\_blocks = self.vacancy\_table[pre\_free\_seq][1] + self.vacancy\_table[pre\_free\_seq][1] + size  
 self.vacancy\_table[pre\_free\_seq] = [update\_add, update\_blocks]  
  
 **del** self.vacancy\_table[next\_free\_seq]  
 *# 上下都不是空闲表* **elif not** (pre\_free **or** next\_free):  
 seq  
 min\_key = min(self.vacancy\_table.keys())  
 max\_key = max(self.vacancy\_table.keys())  
 **for** i **in** range(min\_key + 1, max\_key + 2):  
 **if** i **not in** self.vacancy\_table:  
 seq = i  
 self.vacancy\_table[seq] = [address, size]  
  
 *# 上是空闲，下不空闲* **elif** pre\_free **and not** next\_free:  
 update\_add = self.vacancy\_table[pre\_free\_seq][0]  
 update\_blocks = self.vacancy\_table[pre\_free\_seq][1] + size  
 self.vacancy\_table[pre\_free\_seq] = [update\_add, update\_blocks]  
 *# 上不空闲，下空闲* **else**:  
 update\_blocks = self.vacancy\_table[next\_free\_seq][1] + size  
 self.vacancy\_table[next\_free\_seq] = [address, update\_blocks]  
 **del** self.directory[name]  
 self.files\_num -= 1  
  
 self.free\_blocks\_num += size  
 print(**"Successfully deleted"**)  
 **return True  
  
 def** Search\_file(self, name):  
 file = self.directory.get(name, **None**)  
 **if** file **is None**:  
 print(**"No such a file"**)  
 **return None  
 return** file[2]  
  
 **def** show\_info(self):  
 print(**"totle block num:%d"** % self.block\_num)  
 print(**"block size:\t%dk"** % (self.block\_size // 1000))  
 print(**"files num:\t%d"** % self.files\_num)  
 print(**"free blocks num:%d"** % self.free\_blocks\_num)  
 print(**"vacancy table:\t"**, self.vacancy\_table)  
 print(**"directory:"**)  
 **for** key, value **in** self.directory.items():  
 print(key, value[:2])  
  
 **def** info\_text(self):  
  
 info = **"totle block num:\t%d\nblock size:\t\t%dk\nfiles num:\t\t%d\nfree blocks num:\t%d"** % (  
 self.block\_num, self.block\_size // 1000, self.files\_num, self.free\_blocks\_num)  
  
 **return** info  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
  
 dm = disk\_manager()  
  
 *# 随机生成50个文件* **for** i **in** range(50):  
 name = str(i) + **'.txt'** size = random.randint(2000, 10000)  
 dm.Create\_file(name, size)  
  
 **for** i **in** range(0, 50, 2):  
 name = str(i) + **'.txt'** dm.Delete\_file(name)  
  
 dm.Create\_file(**'A.txt'**, 7000)  
 dm.Create\_file(**'B.txt'**, 5000)  
 dm.Create\_file(**'C.txt'**, 2000)  
 dm.Create\_file(**'D.txt'**, 9000)  
 dm.Create\_file(**'E.txt'**, 3500)  
  
 dm.show\_info()  
  
 *# dm.Delete\_file(r'3.txt')  
 # dm.show\_info()*