

实验一: 实现单处理机下的进程调度程序

1120152035王奥博

语言:python2.7

OS:deepin-linux 15.5 amd64

网教上提交的代码为C++代码,我同时还写了一份python的代码,觉得python的代码更优美,因此这次的实验报告以python代码为例进行分析

数据结构及符号分析:

在本份代码中,对于整个程序,我定义了一个类**Threads**来保存信息和进行各种调度,对输入的进程,使用了python的内置**数据结构字典**来进行保存进程的各项信息,使用了列表**self.T**来保存所有的进程,如下:

```
12 def __init__(self):
13     self.method = int(stdin.readline())
14     self.T = []
15     while True:
16         line = stdin.readline()
17         if line == "" or line == "\n":
18             break
19         d = {}
20         d["id"], d["st"], d["runtime"], d["priority"], d["timeslice"] = [int(i.strip()) for i in line.split(r"/")]
21         d["used"] = False
22         # print d
23         self.T.append(d)
```

分别保存了进程号,到达时间,运行时间,优先级以及时间片等信息,同时添加了一项是否在使用中方便后续的调度(默认为False).

而对于输出的调度方案,我用了另一个字典:

```
28 def getAns(self, inThread):
29     ans = []
30     for i, j in enumerate(inThread):
31         d = {}
32         d["seq"], d["id"], d["priority"], d["runtime"] = (i + 1, j["id"], j["priority"], j["runtime"])
33         d["st"] = ans[i - 1]["ed"] if i else 0
34         d["ed"] = d["st"] + d["runtime"]
35         ans.append(d)
36
37     for i in ans:
38         print "%d/%d/%d/%d/%d" % (i["seq"], i["id"], i["st"], i["ed"], i["priority"])
```

保存了调度顺序,进程号,开始运行时间,运行结束时间,优先级等信息

可以看出,用python处理输入输出和数据保存是很方便的

调度算法的处理流程

对于五种调度算法,定义了5个函数,并在调用函数时使用了一种巧妙地方法选择对应函数:定义了一个函数元组,利用元祖的下标进行选择,这样就避免了用多个if或者switch进行选择,具体的实现如下:

```

224 if __name__ == "__main__":
225     lab1 = Threads()
226     method = (lab1.FCFS, lab1.SPF, lab1.SRF, lab1.RR, lab1.DP)
227     # pdb.set_trace()
228     method[lab1.method - 1]()
NORMAL lab1.py

```

接下来解释不同的调度算法:

先来先服务(FCFS)

先来先服务是这几种调度算法中最简单的,只需对输入进程的开始时间进行排序即可,因此实现先来先服务的核心代码只有一行(图中第41行):

```

40 def FCFS(self):
41     outThread = sorted(self.T, key = lambda x: x["st"])
42     # pprint(outThread)
43     self.getAns(outThread)
44

```

41行实现了按照程序开始时间进行排序,`self.getAns()`函数实现了对结果的格式化输出(代码的具体实现已经解释过)

对先来先服务进行测试如下图:

```

lab1 [master●] cat t1
1
1/0/24/1/1
2/0/3/1/1
3/0/3/1/1

lab1 [master●] cat t1 | python lab1.py
1/1/0/24/1
2/2/24/27/1
3/3/27/30/1
lab1 [master●]

```

最短进程优先(SPF)

对于最短进程优先,因为是不可抢占型的,所以最终的调度的记录数和输入线程的总数量是相同的,利用这一点等量关系寻找满足下列条件的进程即可:

1. 已经到达的进程
2. 该进程的运行时间在已到达的进程中最短
3. 该进程没有被调度过

具体实现如下:

```
45 def SPF(self):
46     inThread = sorted(self.T, key = lambda x: x["runtime"])
47     time = 0
48     outThread = []
49     for i in xrange(len(self.T)):
50         # pprint(inThread)
51         # print ""
52         for j in inThread:
53             if j["st"] <= time and not j["used"]:
54                 time += j["runtime"]
55                 j["used"] = True
56                 outThread.append(j)
57                 break
58         # pdb.set_trace()
59     self.getAns(outThread)
```

进行测试:

```
lab1 [master●] cat t2
2
1/0/7/1/1
2/2/4/1/1
3/4/1/1/1
4/5/4/1/1

lab1 [master●] cat t2 | python lab1.py
1/1/0/7/1
2/3/7/8/1
3/2/8/12/1
4/4/12/16/1
lab1 [master●]
```

最短剩余时间优先(SRF)

相比前两种调度方法,最短剩余时间优先的最大特点是可抢占,可抢占意味着最终的调度记录数 \geq 初始进程数.在实现上,我定义了一个列表timeTable,timeTable中存储了发生调度的时间点,包括:

1. 每个进程的到达时间
2. 某些进程的结束时间

对于每个时间点,只需找到满足下列条件的进程即可:

1. 该进程已经到达
2. 该进程的剩余时间最短且不为0

对于每个进程剩余时间的处理,分为两种:

1. 进程刚到达时,每个进程的剩余时间与运行时间相等
2. 进程运行时:
 1. 如果能运行到下一时间点,则该进程的剩余时间为当前剩余时间减去当前时间点与下一时间点间的时间
 2. 如果不能运行到下一时间点,则该进程的剩余时间为0,并且需要把该进程的结束时间添加进时timeTable

具体实现如下:

```

def SRF(self):
    inThread = sorted(self.T, key = lambda x: x["st"])
    for i in inThread:
        i["remain"] = i["runtime"]

    timeTable = [i["st"] for i in inThread]
    timeTable.append(sum([i["runtime"] for i in inThread]))
    timeTable.append(sum([i["runtime"] for i in inThread]))#多加一次总时间方便处理

    outThread = []
    idx = 0
    while True:
        t = timeTable[idx]
        if t == timeTable[-1]:
            break

        inThread = sorted(inThread, key = lambda x: x["remain"])
        for i in inThread:
            # pdb.set_trace()
            if i["remain"] and i["st"] <= t:
                #bug: 出现相同键值对完全相同的字典时,修改对所有字典起作用
                #python的引用: http://www.cnblogs.com/Xjng/p/3829368.html
                tmp = deepcopy(i)
                tmp["st"] = t
                if (timeTable[idx + 1] - t) <= i["remain"]:
                    tmp["runtime"] = (timeTable[idx + 1] - t)
                    i["remain"] -= (timeTable[idx + 1] - t)
                    outThread.append(tmp)
                    # pdb.set_trace()
                else:
                    tmp["runtime"] = tmp["remain"]
                    i["remain"] = 0
                    outThread.append(tmp)
                    timeTable = timeTable[: idx + 1] + [timeTable[idx] + tmp["runtime"]]
            + timeTable[idx + 1:]

            # pdb.set_trace()

        idx += 1
        break

    # pprint(outThread)
    self.getAns(outThread)

```

测试如下:

```
lab1 [master●] cat t3
3
1/0/7/1/1
2/2/4/1/1
3/4/1/1/1
4/5/4/1/1

lab1 [master●] cat t3 | python lab1.py
1/1/0/2/1
2/2/2/4/1
3/3/4/5/1
4/2/5/7/1
5/4/7/11/1
6/1/11/16/1
lab1 [master●]
```

在SRF的实现过程中,遇到了一个值得提一下的python特性

python在创建字典时,默认是以传址,即引用的方式传递的,也就是说,当我们创建两个相同的字典时,更改其中一个,另一个也会随着更改

为了避免字典引用带来的影响,可以使用深拷贝来声明字典`copy.deepcopy()`

在注释中放了一个讲解的清楚地链接 <http://www.cnblogs.com/Xjng/p/3829368.html>

```
lab1 [master●] cat lab1.py | grep -n cnblog
82:                #python的引用: http://www.cnblogs.com/Xjng/p/3829368.html
lab1 [master●]
```

时间片轮转法(RR):

RR与SRF的基本思想相同:维护timeTable,维护进程的剩余时间,与SRF不同的是:

- timeTable的选择不再以进程的到达时间初始化,而是按照时间片初始化,再进行后续的维护,时间片的初始化代码如下:

```
while True:
    timeTable.append(timeSlice * ii)
    ii += 1
    if timeTable[-1] >= T:
        timeTable = timeTable[:-1]
        break
```

选取满足条件的进程时需要体现轮转,循环的选取进程,轮转的代码具体实现如下:

```
138 于伪彩色          while True:
139 图像...df          if inThread[id]["remain"] and inThread[id]["st"] <= t:
140                    break
141                    id = (id + 1) % cnt
```

注意了这一点,只要稍微改一下SRF的代码即可,RR的代码实现如下:

```

def RR(self):
    inThread = sorted(self.T, key = lambda x: x["st"])
    for i in inThread:
        i["remain"] = i["runtime"]

    timeTable = []
    timeSlice = inThread[0]["timeslice"]
    ii = 0
    T = sum([i["runtime"] for i in inThread])
    while True:
        timeTable.append(timeSlice * ii)
        ii += 1
        if timeTable[-1] >= T:
            timeTable = timeTable[: -1]
            break

    # print "flag"
    timeTable.append(T)
    # pprint(timeTable)
    cnt = inThread[-1]["id"]

    idx = 0
    id = 0 #id + 1 为pid
    outThread = []
    while True:
        t = timeTable[idx]
        if t == timeTable[-1]:
            # pdb.set_trace()
            break
        # if len(outThread) >= 10:
        #     break

        # print "flag"
        # self.getAns(outThread)
        inThread = sorted(inThread, key = lambda x: x["st"])
        while True:
            if inThread[id]["remain"] and inThread[id]["st"] <= t:
                break
            id = (id + 1) % cnt

        if inThread[id]["remain"] and inThread[id]["st"] <= t:
            tmp = deepcopy(inThread[id])
            tmp["st"] = t
            if inThread[id]["remain"] >= timeSlice:
                tmp["runtime"] = timeSlice
                inThread[id]["remain"] -= timeSlice
                outThread.append(tmp)
            else:
                tmp["runtime"] = tmp["remain"]
                inThread[id]["remain"] = 0
                outThread.append(tmp)

        timeTable = timeTable[: idx + 1] + [timeTable[idx] + tmp["runtime"]] +

```



```

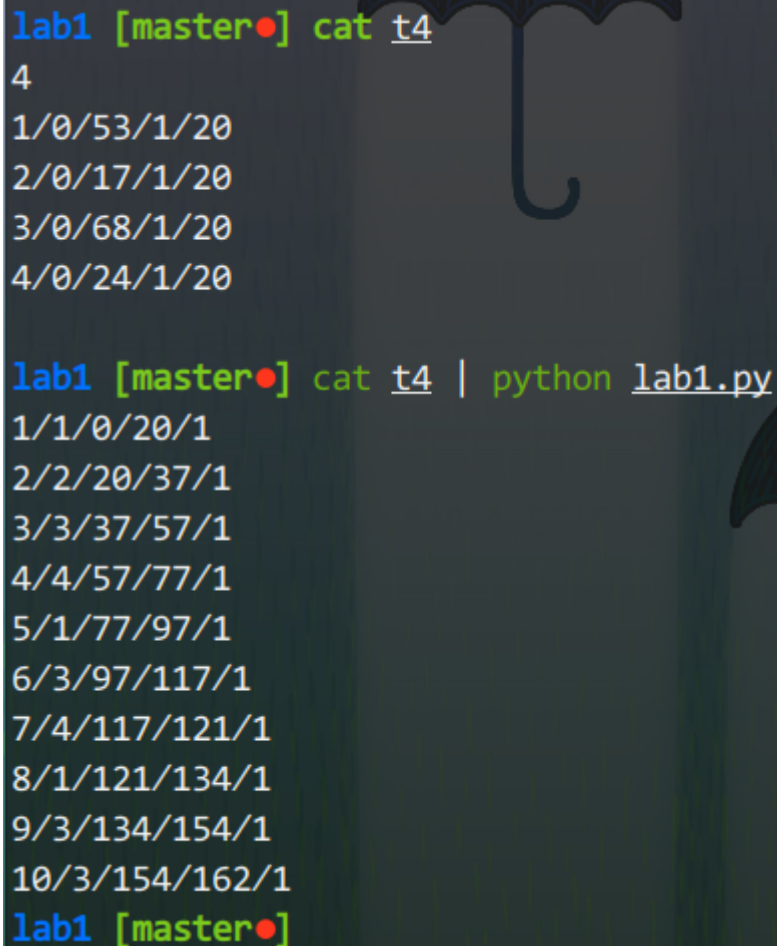
timeTable[idx + 1:]

        # if idx == 6:
        #     pdb.set_trace()
        id = (id + 1) % cnt
        idx += 1

    # pprint(outThread)
    self.getAns(outThread)

```

对RR进行测试:



```

lab1 [master●] cat t4
4
1/0/53/1/20
2/0/17/1/20
3/0/68/1/20
4/0/24/1/20

lab1 [master●] cat t4 | python lab1.py
1/1/0/20/1
2/2/20/37/1
3/3/37/57/1
4/4/57/77/1
5/1/77/97/1
6/3/97/117/1
7/4/117/121/1
8/1/121/134/1
9/3/134/154/1
10/3/154/162/1
lab1 [master●]

```

动态优先级(DP)

对于DP,可以看成是SRF与RR的综合,在具体实现上,也是通过维护timeTable和进程的剩余时间实现调度的,与SRF相比,timeTable的初始化数据为时间片的整数倍,具体实现如下:

```

185         ii = 0
186         while True:
187             timeTable.append(timeSlice * ii)
188             ii += 1
189             if timeTable[-1] >= T:
190                 timeTable = timeTable[: -1]
191                 break
192         timeTable.append(T)

```

与RR相比,进程每次的选取原则是:

1. 当前时间已到达
2. 该进程的优先级最高

选择最高优先级只需把RR的按剩余时间排序改为按优先级排序即可,注意需要在每次调度之后对该轮所有没调度的进程优先级减一,实现代码如下:

```

225         for i in inThread:
226             if i["used"] == True:
227                 i["used"] = False
228             else:
229                 #优先级最高为0
230                 i["priority"] = 0 if i["priority"] - 1 <= 0 else i["priority"] - 1
231

```

只要注意以上几点,在RR和SRF的基础上就可以很快写出DP的代码了,DP的完整代码如下:

```

def DP(self):
    inThread = sorted(self.T, key = lambda x: x["st"])
    for i in inThread:
        i["remain"] = i["runtime"]

    timeTable = []
    timeSlice = inThread[0]["timeslice"]
    T = sum([i["runtime"] for i in inThread])

    ii = 0
    while True:
        timeTable.append(timeSlice * ii)
        ii += 1
        if timeTable[-1] >= T:
            timeTable = timeTable[: -1]
            break
    timeTable.append(T)

    idx = 0
    outThread = []
    while True:
        t = timeTable[idx]
        if t == timeTable[-1]:
            break

        # if len(outThread) >= 8:
        #     break

        pdb.set_trace()
        inThread = sorted(inThread, key = lambda x: (x["priority"], x["st"]))
        for i in inThread:
            if i["remain"] and i["st"] <= t:
                i["used"] = True
                i["priority"] += 3
                tmp = deepcopy(i)
                tmp["st"] = t
                if tmp["remain"] >= timeSlice:
                    tmp["runtime"] = timeSlice
                    i["remain"] -= timeSlice
                    outThread.append(tmp)
                else:
                    tmp["runtime"] = tmp["remain"]
                    i["remain"] = 0
                    outThread.append(tmp)
                timeTable = timeTable[: idx + 1] + [timeTable[idx] + tmp["runtime"]]
            + timeTable[idx + 1:]

            idx += 1
            break

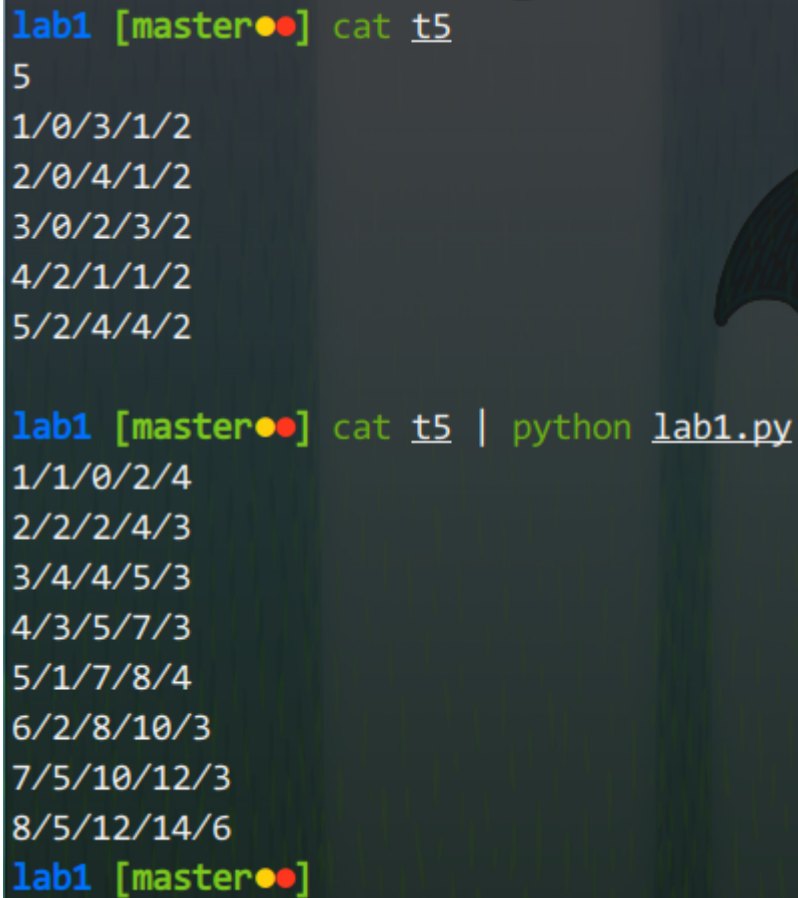
        for i in inThread:
            if i["used"] == True:
                i["used"] = False

```

```
        else:
            #优先级最高为0
            i["priority"] = 0 if i["priority"] - 1 <= 0 else i["priority"] - 1

    self.getAns(outThread)
```

测试如下:

A terminal window with a dark background and a faint umbrella illustration on the right. The prompt is 'lab1 [master●●]'. The first command is 'cat t5', which outputs five lines of numbers: '5', '1/0/3/1/2', '2/0/4/1/2', '3/0/2/3/2', and '4/2/1/1/2'. The second command is 'cat t5 | python lab1.py', which outputs eight lines of numbers: '1/1/0/2/4', '2/2/2/4/3', '3/4/4/5/3', '4/3/5/7/3', '5/1/7/8/4', '6/2/8/10/3', '7/5/10/12/3', and '8/5/12/14/6'. The prompt returns to 'lab1 [master●●]'.

```
lab1 [master●●] cat t5
5
1/0/3/1/2
2/0/4/1/2
3/0/2/3/2
4/2/1/1/2
5/2/4/4/2

lab1 [master●●] cat t5 | python lab1.py
1/1/0/2/4
2/2/2/4/3
3/4/4/5/3
4/3/5/7/3
5/1/7/8/4
6/2/8/10/3
7/5/10/12/3
8/5/12/14/6
lab1 [master●●]
```

源代码

本次试验的代码均已上传到

https://github.com/M4xW4n9/personal_repository/blob/master/OS/lab1/lab1.py

可通过

wget https://raw.githubusercontent.com/M4xW4n9/personal_repository/master/OS/lab1/lab1.py

进行下载

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Author__ = 'M4x'

from sys import stdin
from pprint import pprint
from time import sleep
from copy import deepcopy
import pdb

class Threads(object):
    def __init__(self):
        self.method = int(stdin.readline())
        self.T = []
        while True:
            line = stdin.readline()
            if line == "" or line == "\n":
                break
            d = {}
            d["id"], d["st"], d["runtime"], d["priority"], d["timeslice"] = [int(i.strip())
for i in line.split(r"/")]
            d["used"] = False
            # print d
            self.T.append(d)

            # print self.method
            # pprint(self.T)

    def getAns(self, inThread):
        ans = []
        for i,j in enumerate(inThread):
            d = {}
            d["seq"], d["id"], d["priority"], d["runtime"] = (i + 1, j["id"], j["priority"],
j["runtime"])
            d["st"] = ans[i - 1]["ed"] if i else 0
            d["ed"] = d["st"] + d["runtime"]
            ans.append(d)

        for i in ans:
            print "%d/%d/%d/%d/%d" % (i["seq"], i["id"], i["st"], i["ed"], i["priority"])

    def FCFS(self):
        outThread = sorted(self.T, key = lambda x: x["st"])
        # pprint(outThread)
        self.getAns(outThread)

    def SPF(self):
        inThread = sorted(self.T, key = lambda x: x["runtime"])
        time = 0
        outThread = []
        for i in xrange(len(self.T)):
            # pprint(inThread)
            # print ""

```

```

        for j in inThread:
            if j["st"] <= time and not j["used"]:
                time += j["runtime"]
                j["used"] = True
                outThread.append(j)
                break
# pdb.set_trace()
self.getAns(outThread)

def SRF(self):
    inThread = sorted(self.T, key = lambda x: x["st"])
    for i in inThread:
        i["remain"] = i["runtime"]

    timeTable = [i["st"] for i in inThread]
    timeTable.append(sum([i["runtime"] for i in inThread]))
    timeTable.append(sum([i["runtime"] for i in inThread]))#多加一次总时间方便处理

    outThread = []
    idx = 0
    while True:
        t = timeTable[idx]
        if t == timeTable[-1]:
            break

        inThread = sorted(inThread, key = lambda x: x["remain"])
        for i in inThread:
            # pdb.set_trace()
            if i["remain"] and i["st"] <= t:
                #bug: 出现相同键值对完全相同的字典时,修改对所有字典起作用
                #python的引用: http://www.cnblogs.com/Xjng/p/3829368.html
                tmp = deepcopy(i)
                tmp["st"] = t
                if (timeTable[idx + 1] - t) <= i["remain"]:
                    tmp["runtime"] = (timeTable[idx + 1] - t)
                    i["remain"] -= (timeTable[idx + 1] - t)
                    outThread.append(tmp)
                    # pdb.set_trace()
                else:
                    tmp["runtime"] = tmp["remain"]
                    i["remain"] = 0
                    outThread.append(tmp)
                    timeTable = timeTable[: idx + 1] + [timeTable[idx] + tmp["runtime"]]
+ timeTable[idx + 1:]
                    # pdb.set_trace()

            idx += 1
            break

        # pprint(outThread)
        self.getAns(outThread)

def RR(self):

```

```

inThread = sorted(self.T, key = lambda x: x["st"])
for i in inThread:
    i["remain"] = i["runtime"]

timeTable = []
timeSlice = inThread[0]["timeslice"]
ii = 0
T = sum([i["runtime"] for i in inThread])
while True:
    timeTable.append(timeSlice * ii)
    ii += 1
    if timeTable[-1] >= T:
        timeTable = timeTable[: -1]
        break

# print "flag"
timeTable.append(T)
# pprint(timeTable)
cnt = inThread[-1]["id"]

idx = 0
id = 0#id + 1为pid
outThread = []
while True:
    t = timeTable[idx]
    if t == timeTable[-1]:
        # pdb.set_trace()
        break
    # if len(outThread) >= 10:
    #     break

    # print "flag"
    # self.getAns(outThread)
    inThread = sorted(inThread, key = lambda x: x["st"])
    tmp = 0
    while True:
        if inThread[id]["remain"] and inThread[id]["st"] <= t:
            break
        id = (id + 1) % cnt

    if inThread[id]["remain"] and inThread[id]["st"] <= t:
        tmp = deepcopy(inThread[id])
        tmp["st"] = t
        if inThread[id]["remain"] >= timeSlice:
            tmp["runtime"] = timeSlice
            inThread[id]["remain"] -= timeSlice
            outThread.append(tmp)
        else:
            tmp["runtime"] = tmp["remain"]
            inThread[id]["remain"] = 0
            outThread.append(tmp)

    timeTable = timeTable[: idx + 1] + [timeTable[idx] + tmp["runtime"]] +

```

```

timeTable[idx + 1:]

        # if idx == 6:
        #     pdb.set_trace()
        id = (id + 1) % cnt
        idx += 1

# pprint(outThread)
self.getAns(outThread)

def DP(self):
    inThread = sorted(self.T, key = lambda x: x["st"])
    for i in inThread:
        i["remain"] = i["runtime"]

    timeTable = []
    timeSlice = inThread[0]["timeslice"]
    T = sum([i["runtime"] for i in inThread])

    ii = 0
    while True:
        timeTable.append(timeSlice * ii)
        ii += 1
        if timeTable[-1] >= T:
            timeTable = timeTable[: -1]
            break
    timeTable.append(T)

    idx = 0
    outThread = []
    while True:
        t = timeTable[idx]
        if t == timeTable[-1]:
            break

        # if len(outThread) >= 8:
        #     break

        pdb.set_trace()
        inThread = sorted(inThread, key = lambda x: (x["priority"], x["st"]))
        for i in inThread:
            if i["remain"] and i["st"] <= t:
                i["used"] = True
                i["priority"] += 3
                tmp = deepcopy(i)
                tmp["st"] = t
                if tmp["remain"] >= timeSlice:
                    tmp["runtime"] = timeSlice
                    i["remain"] -= timeSlice
                    outThread.append(tmp)
                else:
                    tmp["runtime"] = tmp["remain"]

                i["remain"] = 0

```



```

        outThread.append(tmp)
        timeTable = timeTable[: idx + 1] + [timeTable[idx] + tmp["runtime"]]
+ timeTable[idx + 1:]

        idx += 1
        break

    for i in inThread:
        if i["used"] == True:
            i["used"] = False
        else:
            #优先级最高为0
            i["priority"] = 0 if i["priority"] - 1 <= 0 else i["priority"] - 1

    self.getAns(outThread)

if __name__ == "__main__":
    lab1 = Threads()
    method = (lab1.FCFS, lab1.SPF, lab1.SRF, lab1.RR, lab1.DP)
    # pdb.set_trace()
    method[lab1.method - 1]()

```

测试方法:

因为在完成python版之前,我已在网教上通过了C++版的代码,因此python版的完成没有遇到太多问题.

在C++版的完成过程中,采用的测试方法是单元测试,即每完成一个功能进行一次测试,事实证明,这种测试方法很适合类似lab1这种多功能的程序.

另外值得一提的是,在python版的完成过程中学习了pdb的使用方法,觉得比较方便,这里记录一下

pdb类似于gdb,可实现对python脚本的debug,已知的使用方法有两种

1. 在脚本中

```

import pdb

pdb.set_trace()#在合适的地方下断点,运行python脚本时即可调试

```

2. 类似于gdb的使用,在shell中

```

pdb ./pythonScript

```

本次试验经验及体会

本次实验完成的是5种进程调度方法的实现,原理和具体的代码都很简单,只需要完成5种不同的排序即可.进程调度这一块我在大二学习多线程编程时就以初步了解,因此这次的实验没有遇到多少问题就完成了.

对我而言,我在本次试验中获得的最大收获是学到了利用不同语言的特性进行编程,比如在C++中存储进程我定义了一个结构体,而python没有结构体这种数据结构,经过查阅资料,通用的方法是定义一个python类,利用对类变量的调用模拟结构体,但我在实现时想到,python虽然没有结构体,但比起C++有一种特别的数据结构**字典**,经过思考利用字典完全可以代替结构体,于是我选择了利用字典来存储进程信息,事实证明这样写也很方便.

除此之外,我对语言的细节还有了更深的理解,比如python的拷贝与引用,上边已经分析过,这里不再赘述.