

Python程序语言入门与应用



python

Life is short, use Python
人生苦短，我用Python



新乡医学院

Python程序语言入门与应用

第十二章 Python 多线程编程

王海蛟

新乡医学院





下周课程&课后作业



第11章 Python 图形化界面开发基础

课后练习

 编写一个具有一个按钮控件的GUI。

代码文件命名: **11-1BTnGui**

 在GUI窗体上面显示鼠标当前的位置

代码文件命名: **11-2Mospos**

.py代码文件打包(11.学号
+姓名)发送到
python_xxmu@163.com



下周课程&课后作业



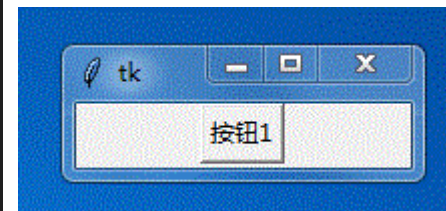
第11章 Python 图形化界面开发基础

课后练习

代码文件命名: 11-1BTnGui

编写一个具有一个按钮控件的GUI。

```
E: > 课程 > python > 第12次课 > 上周作业 > 1.py > ...
1  import tkinter as tk
2  def myaction():
3      S1.set('按钮被点击过了')
4  top=tk.Tk()
5  top.geometry('100x100')
6  S1=tk.StringVar()
7  top.title='添加按钮'
8  bn=tk.Button(top,textvariable=S1,command=myaction)
9  S1.set('按钮1')
10 bn.pack()
11 top.mainloop()
```





下周课程&课后作业



第11章 Python 图形化界面开发基础

课后练习

代码文件命名: 11-2Mospos

在GUI窗体上面显示鼠标当前的位置

```
1  import tkinter as tk
2  def motder(event):
3      s='【鼠标事件】\n\n产生事件的控件: {0}\n鼠标位置: (x={1},y={2})'\n
4      s=s+'\n鼠标事件相对屏幕左上角位置: (x={3},y={4})\n'.format(event.widget,
5      event.x,event.y,event.x_root,event.y_root,)
6      la['text']=s
7      top=tk.Tk()
8      top.title='显示鼠标位置'
9      top.geometry('500x500')
10     top.resizable(width=False,height=False)
11     la=tk.Label(top,bg='blue',width=100,height=20,fg='yellow',
12     font=('times',14,'bold'))
13     la.pack()
14     btn=tk.Button(top,bg='red',text='按钮',width=50,height=20).pack()
15     top.bind('<Button-1>',motder)
16     top.mainloop()
```

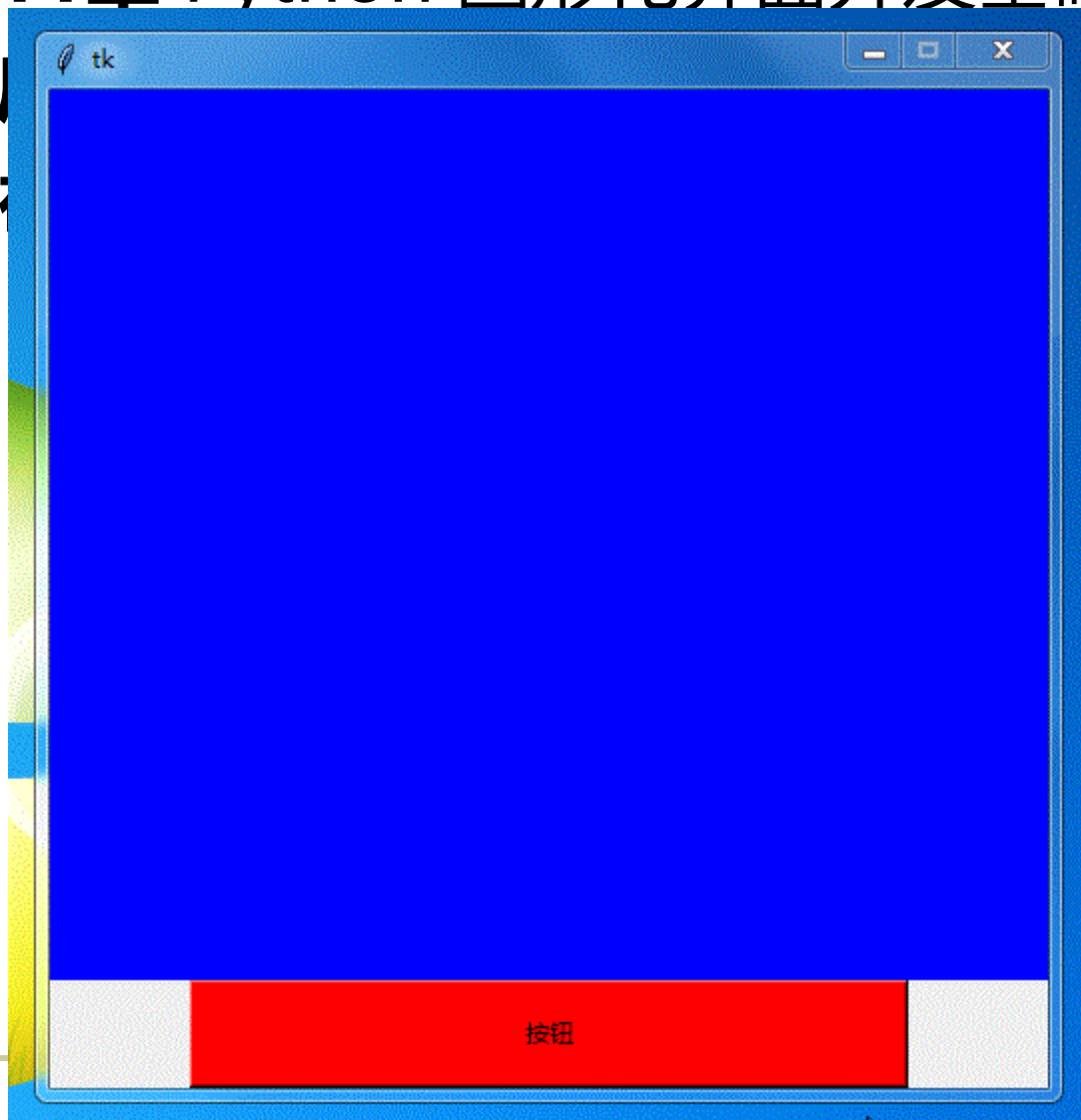


下周课程&课后作业



第11章 Python 图形化界面开发基础

课



2Mospos



新乡医学院

Python程序语言入门与应用

第十二章 Python 多线程编程

王海蛟

新乡医学院





学习目标



人物简介

Martin Fowler为ThoughtWorks的首席科学家。《敏捷软件开发宣言》(Manifesto for Agile Software Development)的作者之一。从上个世纪80年代开始，他就一直从事软件开发的工作。著书众多，在开发领域做出过卓越贡献，被称为“软件开发教父”。

Any fool can write code that a computer can understand. Good programmers write code that humans can understand
任何一个傻瓜都会写能够让机器理解的代码，只有好的程序员才能写出人类可以理解的代码。——Martin Fowler



本课概要



- ❏ 第12章 Python 多线程编程
 - ❏ 12.1线程和进程基础
 - ❏ 12.2python线程处理
 - ❏ 13.3线程优先级列队模块queue
 - ❏ 12.4使用模块subprocess创建进程



学习目标



基本要求

☞ 掌握

- ☞ 线程和进程创建方法

☞ 理解&了解

- ☞ 线程和进程基础知识

- ☞ 线程优先级列队模块queue



第12章 Python 多线程编程



12.1 线程和进程基础

定义

- 进程是程序的一次执行，该进程可与其它进程并发执行；它是一个动态的实体，在传统的操作系统设计中，进程既是资源的基本分配单元，也是基本的执行单元。

进程与程序的区别和联系

- 程序是静态的，进程是动态的。程序是有序代码的集合；进程是程序的一次执行。
- 进程是暂时的，程序的永久的。进程是一个变化的过程，有生命周期，暂时存在，程序没有生命周期，可长久保存。



第12章 Python 多线程编程



12.1 线程和进程基础

Windows 任务管理器

文件(F) 选项(O) 查看(V) 帮助(H)

应用程序 进程 服务 性能 联网 用户

映像名称	用户名	CPU	内存 (...	线程数	映像路径
conhost.exe	Administrator	00	1,360 K	1	C:\Wind
pythonw.exe	Administrator	00	20,092 K	3	C:\User
360se.exe *32	Administrator	02	1,080,...	47	C:\User
dwm.exe	Administrator	00	16,592 K	5	C:\Wind
splwow64.exe	Administrator	00	1,712 K	6	C:\Wind
YoudaoDictHel...	Administrator	00	11,404 K	12	C:\User
conhost.exe	Administrator	00	4,028 K	2	C:\Wind
2345PinyinPic...	Administrator	00	4,640 K	6	C:\Prog
pythonw.exe	Administrator	00	11,876 K	3	C:\User
cmd.exe	Administrator	00	940 K	1	C:\Wind
Code.exe	Administrator	00	45,276 K	36	C:\User
Code.exe	Administrator	00	74,636 K	17	C:\User
dllhost.exe	Administrator	00	1,920 K	4	C:\Wind
python.exe	Administrator	00	27,380 K	12	C:\User
sesvc.exe *32	Administrator	00	3,636 K	5	C:\User
RAVCpl64.exe	Administrator	00	4,632 K	11	C:\Prog

☐ 显示所有用户的进程 (S)

进程数: 117 CPU 使用率: 12% 物理内存: 72%



第12章 Python 多线程编程



12.1 线程和进程基础

Refresh Options Find handles or DLLs System information						
Processes Services Network Disk						
Name	PID	CPU	I/O total...	Private ...	User name	Description
System Idle Process	0	78.34		0	NT AUTHO... \SYSTEM	
System	4	0.13		124 kB	NT AUTHO... \SYSTEM	NT Kernel & System
smss.exe	356			576 kB	NT AUTHO... \SYSTEM	Windows 会话管理器
Interrupts		1.01		0		Interrupts and DPCs
csrss.exe	572			2.34 MB	NT AUTHO... \SYSTEM	Client Server Runtime Proc...
wininit.exe	620			1.78 MB	NT AUTHO... \SYSTEM	Windows 启动应用程序
services.exe	676			5.68 MB	NT AUTHO... \SYSTEM	服务和控制器应用程序
svchost.exe	852			4.81 MB	NT AUTHO... \SYSTEM	Windows 服务主进程
dllhost.exe	2624			2.21 MB	MS-2... \Administrator	COM Surrogate
kzip_main.exe	4768			6.38 MB	MS-2... \Administrator	52好压
TXPlatform.exe	8292			2.55 MB	MS-2... \Administrator	腾讯QQ多客户端管理服务
svchost.exe	936			5.7 MB	... \NETWORK SERVICE	Windows 服务主进程
svchost.exe	160			19.19 MB	NT ... \LOCAL SERVICE	Windows 服务主进程
audiodg.exe	2776	0.27		16.45 MB	NT ... \LOCAL SERVICE	Windows 音频设备图形隔离
svchost.exe	492	0.02	688 B/s	234.31 ...	NT AUTHO... \SYSTEM	Windows 服务主进程
dwm.exe	880	0.28		31.53 MB	MS-2... \Administrator	桌面窗口管理器
WUDFHost.exe	6736			2.08 MB	NT ... \LOCAL SERVICE	Windows 驱动程序基础 - 用...
svchost.exe	748	0.07		26.46 MB	NT AUTHO... \SYSTEM	Windows 服务主进程
taskeng.exe	5472			1.98 MB	NT AUTHO... \SYSTEM	任务计划程序引擎
GoogleUpdat...	512			2.07 MB	NT AUTHO... \SYSTEM	Google 安装程序
svchost.exe	1080			3.43 MB	NT AUTHO... \SYSTEM	Windows 服务主进程
svchost.exe	1112			7.57 MB	NT ... \LOCAL SERVICE	Windows 服务主进程
igfxCUIService.exe	1164			2.46 MB	NT AUTHO... \SYSTEM	igfxCUIService Module
ZhuDongFangYu.exe	1420			15.05 MB	NT AUTHO... \SYSTEM	360主动防御服务模块
svchost.exe	1444	0.03	464 B/s	17.47 MB	... \NETWORK SERVICE	Windows 服务主进程
spoolsv.exe	1728			9.55 MB	NT AUTHO... \SYSTEM	后台处理程序子系统应用程序
svchost.exe	1768			11.22 MB	NT ... \LOCAL SERVICE	Windows 服务主进程
armsvc.exe	1852			1.22 MB	NT AUTHO... \SYSTEM	Adobe Acrobat Update Ser...
AlibabaProtect.exe	1888	0.04		19.61 MB	NT AUTHO... \SYSTEM	Alibaba PC Safe Service
CAJSHost.exe	1912			1.72 MB	NT AUTHO... \SYSTEM	TTKN@CAJHost
FlashHelperService...	1960			8.63 MB	NT AUTHO... \SYSTEM	Flash Helper Service



第12章 Python 多线程编程



12.1 线程和进程基础

The screenshot displays the Windows Task Manager interface. On the left, a list of running processes is shown, including python.exe, Code.exe, winpty-agent.exe, cmd.exe, CodeHelper.exe, and YoudaoDict.exe. The YoudaoDict.exe process is selected, and its Properties window is open, showing the Threads tab. The Threads tab lists various threads with their TID, CPU usage, Cycles delta, Start address, and Priority. The thread list includes threads from YoudaoDict.exe, Acrobat2Dict.dll, TextExtractorImpl32.dll, libcef.dll, and libzplay.dll. The Start module is C:\Users\Administrator\AppData\Local\youdao\dict\Application\8.5.1.0\libzplay.dll. The thread list is as follows:

TID	CPU	Cycles delta	Start address	Priority
5976	1.05	134,756,203	YoudaoDict.exe!updatePatch+0...	Normal
6184	0.13	16,328,306	Acrobat2Dict.dll!DllStopMonitor...	Normal
5724	0.13	16,143,995	TextExtractorImpl32.dll!GetText...	Normal
4188	0.02	3,086,645	YoudaoDict.exe!updatePatch+0...	Normal
6156	0.02	2,440,248	YoudaoDict.exe!updatePatch+0...	Normal
5248		291,057	libcef.dll!cef_time_delta+0x621e0	Normal
4780		177,128	libcef.dll!cef_time_delta+0x621e0	Normal
4816		98,764	ntdll.dll!TpCallbackIndependent...	Normal
10148			ntdll.dll!TpCallbackIndependent...	Normal
10020			ntdll.dll!TpCallbackIndependent...	Normal
9708			ntdll.dll!TpCallbackIndependent...	Normal
9656			ntdll.dll!TpCallbackIndependent...	Normal
9192			libzplay.dll!zplay_DestroyZPlay+...	Normal
8976			wdmaud.driv!modMessage+0x1df	-3
8512			ntdll.dll!TpCallbackIndependent...	Normal

Start module: C:\Users\Administrator\AppData\Local\youdao\dict\Application\8.5.1.0\libzplay.dll
Started: 下午 5:29:54 2019-11-25
State: Wait:UserRequest Priority: 8
Base priority: 8



第12章 Python 多线程编程



12.1 线程和进程基础

Task Manager screenshot showing running processes and the Properties window for YoudaoDict.exe (6408).

Running Processes:

Process Name	PID	Private Bytes	Working Set	Session ID	Architecture	Description
le.exe	2312	0.03	54.38 MB	MS-2...	Administrator	Visual Studio Code
Code.exe	2604		89.69 MB	MS-2...	Administrator	Visual Studio Code
Code.exe	5456		161.66 ...	MS-2...	Administrator	Visual Studio Code
Code.exe	5540	0.01	77.38 MB	MS-2...	Administrator	Visual Studio Code
python.exe	5332		73.29 MB	MS-2...	Administrator	Python
python.exe	4696		17.07 MB	MS-2...	Administrator	Python
python.exe	5092		16.31 MB	MS-2...	Administrator	Python
python.exe	6964			MS-2...	Administrator	Python
Code.exe	6292			MS-2...	Administrator	Visual Studio Code
Code.exe	7840			MS-2...	Administrator	Visual Studio Code
winpty-agent.exe	4196			MS-2...	Administrator	Winpty-agent.exe
cmd.exe	2232			MS-2...	Administrator	Command Prompt
CodeHelper.exe	8088			MS-2...	Administrator	CodeHelper.exe
winpty-agent.exe	4784			MS-2...	Administrator	Winpty-agent.exe
cmd.exe	4460			MS-2...	Administrator	Command Prompt
cmd.exe	8576			MS-2...	Administrator	Command Prompt
python....	2700			MS-2...	Administrator	Python
Code.exe	3684			MS-2...	Administrator	Visual Studio Code
Code.exe	5744			MS-2...	Administrator	Visual Studio Code
daoDict.exe	6408			MS-2...	Administrator	YoudaoDict.exe
oudaoDictHelper....	3416			MS-2...	Administrator	YoudaoDictHelper.exe
oudaoEH.exe	3984			MS-2...	Administrator	YoudaoEH.exe
oudaoWSH.exe	8164			MS-2...	Administrator	YoudaoWSH.exe
oudaoDictHelper....	1308			MS-2...	Administrator	YoudaoDictHelper.exe
eenToGif.exe	9028			MS-2...	Administrator	eenToGif.exe
cessHacker.exe	1176			MS-2...	Administrator	cessHacker.exe
eCrashHandler.exe	3796			MS-2...	Administrator	eCrashHandler.exe
eCrashHandler64....	3804			MS-2...	Administrator	eCrashHandler64.exe

YoudaoDict.exe (6408) 属性

Threads:

TID	CPU	Cycles delta	Start address	Priority
5976	0.42	53,410,028	YoudaoDict.exe!updatePatch+0...	Normal
5724	0.15	19,253,680	TextExtractorImpl32.dll!GetText...	Normal
6184	0.14	17,606,931	Acrobat2Dict.dll!DllStopMonitor...	Normal
4188	0.03	3,278,617	YoudaoDict.exe!updatePatch+0...	Normal
6156	0.02	2,400,564	YoudaoDict.exe!updatePatch+0...	Normal
10148			ntdll.dll!TpCallbackIndependent...	Normal
10020			ntdll.dll!TpCallbackIndependent...	Normal
9708			ntdll.dll!TpCallbackIndependent...	Normal
9192			libzplay.dll!zplay_DestroyZPlay+...	Normal
8976			wdmaud.driv!modMessage+0x1df	-3
8512			ntdll.dll!TpCallbackIndependent...	Normal
7788			ntdll.dll!TpCallbackIndependent...	Normal
7736			libcef.dll!cef_time_delta+0x621e0	Normal
7564			libcef.dll!cef_time_delta+0x621e0	Normal
7188			libcef.dll!cef_time_delta+0x621e0	Normal

Start module: C:\Users\Administrator\AppData\Local\youdao\dict\Application\8.5.1.0\libzplay.dll

Started: 下午 5:29:54 2019-11-25

State: Wait:UserRequest **Priority:** 8

Kernel time: 00:00:00.015 **Base priority:** 8

User time: 00:00:00.000 **I/O priority:** Normal

Context switches: 92 **Page priority:** Normal

Cycles: 47,818,689 **Ideal processor:** 0:2



第12章 Python 多线程编程



12.1 线程和进程基础

进程与程序的区别和联系

- 进程还是操作系统资源分配和保护的基本单位，程序没有此功能。
- 进程与程序的对应关系。通过多次执行，一个程序可对应多个进程；通过调用关系，一个进程可包括多个程序。
- 进程与程序的结构不同。

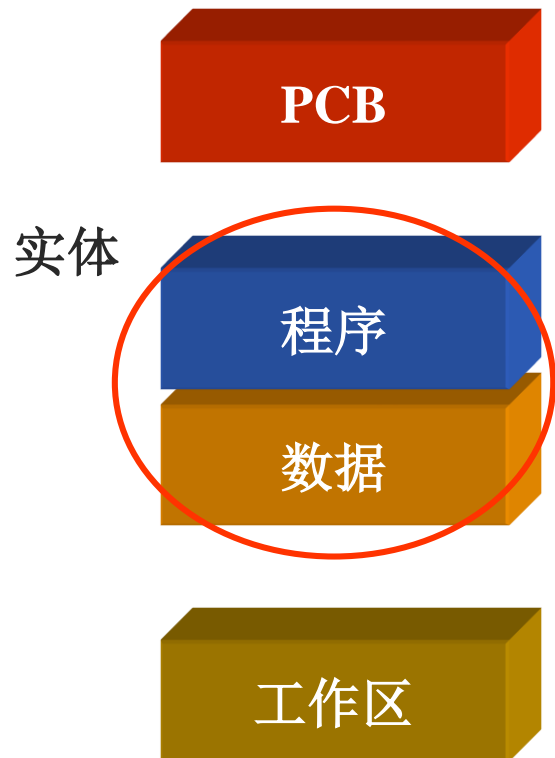


第12章 Python 多线程编程



12.1 线程和进程基础

进程的组成



- **Process Control Block**
灵魂，进程存在的唯一标志。

- 程序：描述了进程要完成的功能，是进程执行时不可修改的部分。
- 数据：进程执行时用到的数据（用户输入的数据、常量、静态变量）。

- 工作区：参数传递、系统调用时使用的动态区域（堆栈区）。



第12章 Python 多线程编程



12.1 线程和进程基础



进程的组成

PCB

- **Process Control Block**
灵魂，进程存在的唯一标志。

是为了管理进程设置的一个数据结构。是系统感知进程存在的唯一标志。

通常包含如以下的信息：

- (1)进程标识符(唯一)
- (2)进程当前状态，通常同一状态的进程会被放到同一个队列；
- (3)进程的程序和数据地址
- (4)进程资源清单。列出所拥有的除CPU以外的资源记录。
- (5)进程优先级。反应进程的紧迫程度
- (6)CPU现场保护区。记录中断时的CPU状态
- (7)进程队列的PCB的链接字。
- (9)进程相关的其他信息。记账用的，如占用CPU多长时间等。

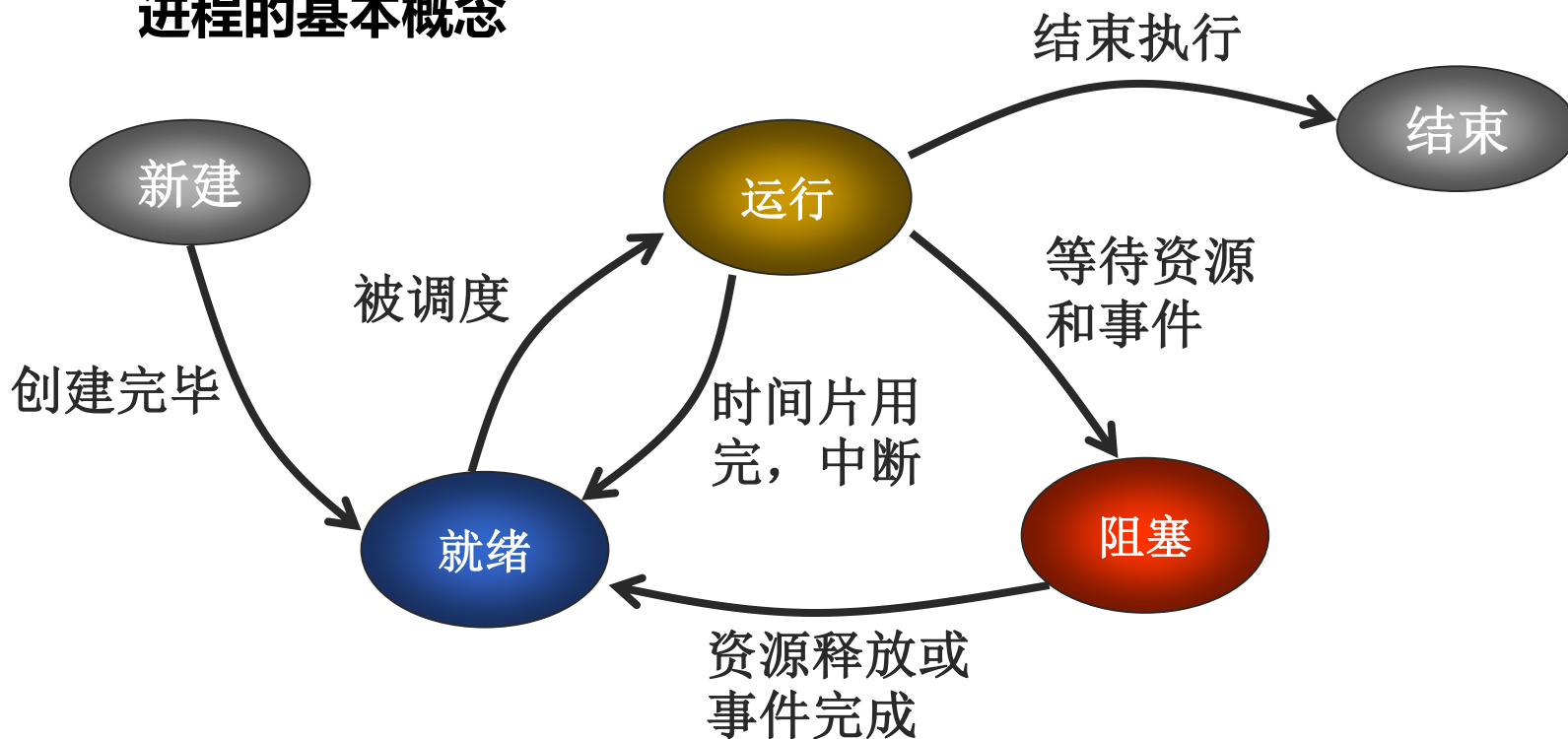


第12章 Python 多线程编程



12.1 线程和进程基础

进程的基本概念



五种进程状态转换



第12章 Python 多线程编程

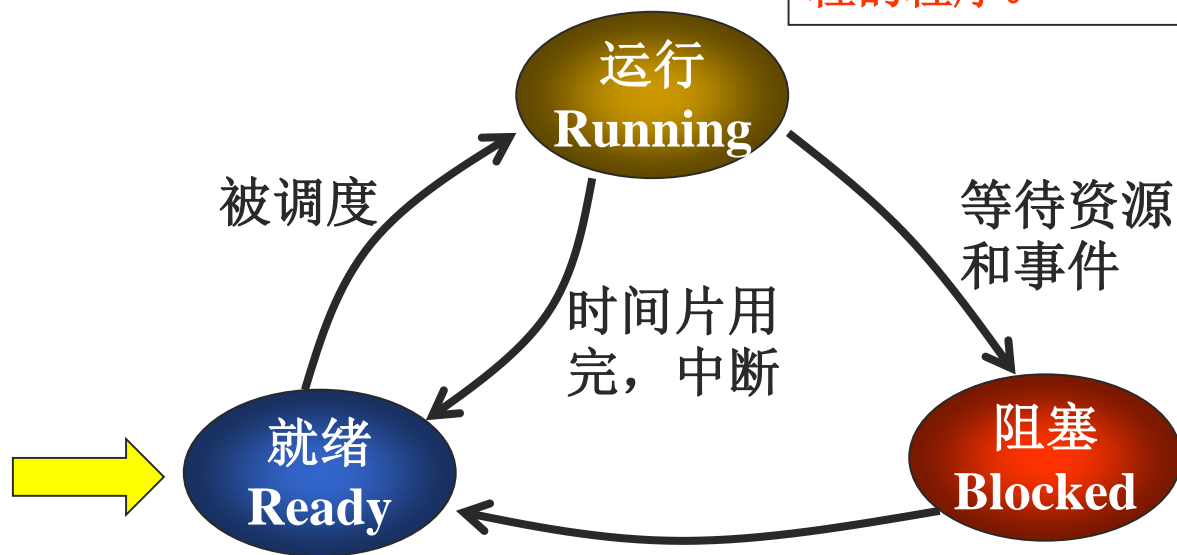


12.1 线程和进程基础



进程/线程的执行

进程占有处理机，处理机正在执行该进程的程序。



进程已获得除处理机外的所需资源，等待分配处理机执行。

也叫等待、挂起、睡眠态，此时进程因等待某种条件（如I/O操作或进程同步）无法运行。引起进程阻塞的原因很多，系统将根据不同的阻塞原因将进程插入某个相应的阻塞队列中。



第12章 Python 多线程编程



12.1 线程和进程基础

进程的特征

- ☞ 并发性：执行时间可以重叠；
- ☞ 动态性：有生命周期，存在不同的状态；
- ☞ 独立性：独立执行，是资源分配和调度的独立单位；
- ☞ 制约性：虽然独立执行，但可能存在相互制约关系；
- ☞ 异步性：各进程执行时间相对独立，不确定；
- ☞ 结构性：拥有固定结构。



第12章 Python 多线程编程



12.1 线程和进程基础

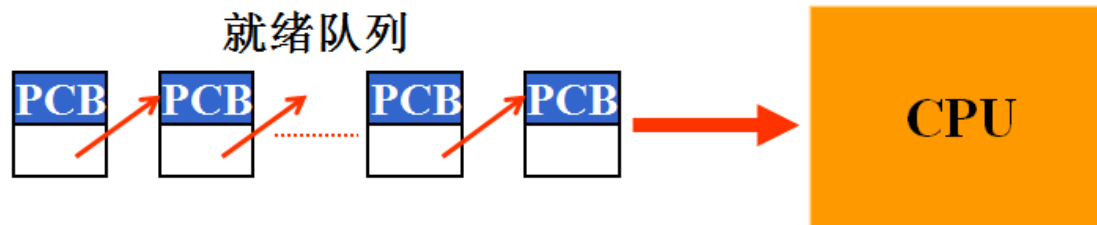
进程的调度

定义

- ❖ 就是按照一定的算法，从就绪队列中选择某个进程占用**CPU**的方法——对**CPU**资源进行合理的分配使用，以提高处理机利用率，并使各进程公平得到处理机资源。

进程调度算法

- ❖ 先来先服务调度算法 (FCFS, First Come First Served)





第12章 Python 多线程编程



12.1 线程和进程基础



进程的调度

❖ 基于优先数的调度算法(Priority Scheduling Algorithm)

- 思想：给每一个进程设置一个优先数(优先级)，系统在调度时优先选择具有高优先级的进程占用**CPU**。具有相同优先数的进程按照**FCFS**算法执行。
- 优先数的确定：
 - ① 运行前：可根据外设的使用情况，运行时间的长短，紧急程度，重要程度等因素确定。
 - ② 运行中：
 - **静态优先数法**：进程创建时就规定好它的优先数，这个数值在进程运行时不变。
 - **动态优先数法**：进程的优先数在执行过程中可以根据情况变化而改变。



第12章 Python 多线程编程



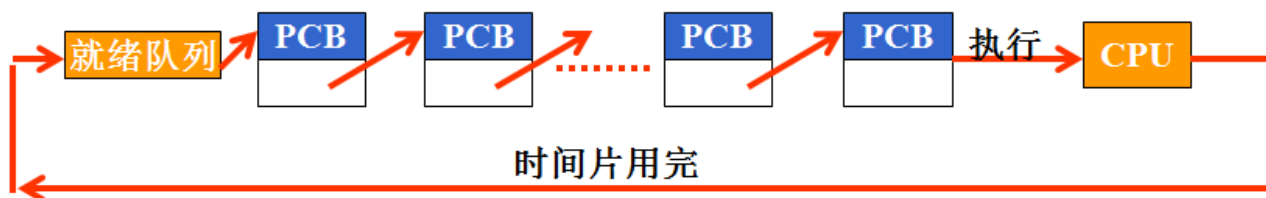
12.1 线程和进程基础



进程的调度

❖ 时间片轮转法 (RR, Round Robin)

- 特点：专门为分时系统设计。类似于**FCFS**算法但是增加了抢占及进程间的切换功能。
- 思想：系统规定一个时间长度(时间片/时间量)作为允许一个进程运行的时间，如果在这段时间该进程没有执行完，则必须让出**CPU**等待下一次分配的时间片。





第12章 Python 多线程编程



12.1 线程和进程基础

进程间的相互作用

同步

- ❖ 进程之间相互合作、协同工作的关系称为进程的同步。简单说来就是：多个相关进程在执行次序上的协调。进程间的直接制约。

临界资源

- ❖ 也称独占资源，是指在一段时间内只允许一个进程访问的资源。例如打印机，磁带机，也可以是进程共享的数据、变量等。

互斥

- ❖ 定义：当多个进程因为争夺临界资源而互斥执行称为进程的互斥。进程间的间接制约。



第12章 Python 多线程编程



12.1 线程和进程基础



线程的基本概念

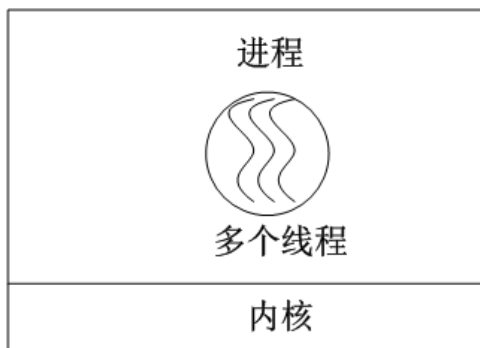


定义

- ❖ 线程(**thread**)也叫轻型进程，是一个可执行的实体单元。它代替以往的进程，成为现代操作系统中处理机调度的基本单位。



线程和进程的关系



多线程模型

- 1、线程是进程的一个组成部分，线程由进程创建，因此一个进程中至少存在一个线程，线程还可以创建其它线程。
- 2、进程是资源分配和保护的基本单位，线程只能在进程的地址空间活动，线程只能使用其所在进程的资源。



第12章 Python 多线程编程



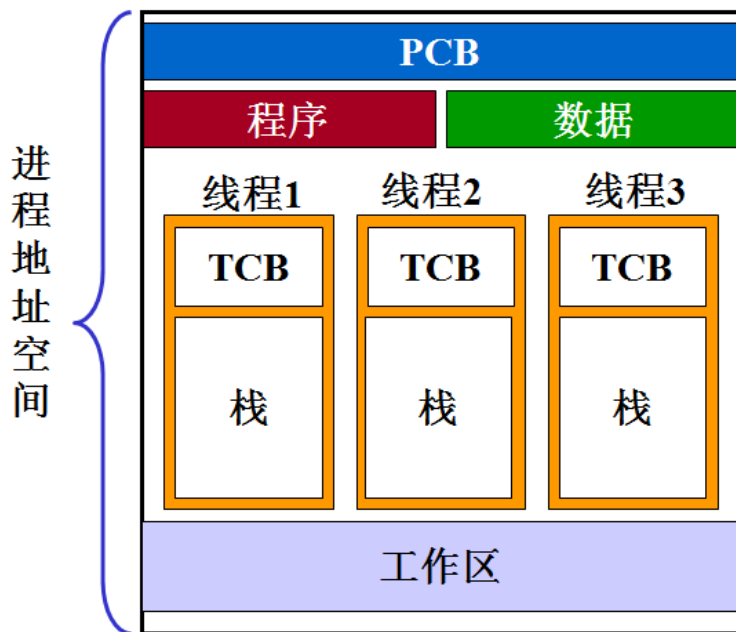
12.1 线程和进程基础



线程的基本概念



线程的结构



线程的特点：

- 1、线程作为基本的调度单位，其状态有：就绪、运行、阻塞等；
- 2、进程中都所有线程共享进程的存储空间和分配资源。



第12章 Python 多线程编程



12.1 线程和进程基础

线程的基本概念

线程优势

- ❖ 创建和撤消线程的开销非常小。

- 不需要向系统请求独立的地址空间及进行相关的地址空间复制(例如父子进程), 因此创建和撤销线程系统的开销要远小于进程。

- ❖ 切换迅速。

- 线程的上下文环境要比进程简单的多, 因此线程间的切换远比进程快的多。

- ❖ 通信效率高。

- 同一进程中的线程由于共享同一地址空间, 通信时不需要借助内核功能。

- ❖ 并发度高。

- 在多处理机系统中, 对进程的个数是有所限制的, 但对线程的个数理论上不存在限制, 更发挥了多处理机系统的优势。



第12章 Python 多线程编程



12.2python线程处理

 在python中, Threading模块可以进行多线程的创建和管理。

1.Threading模块进行线程的创建:

1.通过类继承的方式:

Class mythread (threading.Thread) :

2.通过直接在线程中运行函数

threading.Thread(target = func1, args)



第12章 Python 多线程编程



12.2python线程处理



通过继承类threading.Thread创建线程：

```
1 import threading
2 class mythread(threading.Thread):
3     ... def __init__(self, num):
4     ...     threading.Thread.__init__(self)
5     ...     self.num = num
6     ... def run(self):
7     ...     print('I am {0}\n'.format(self.num))
8 t1 = mythread(1)
9 t2 = mythread(2)
10 t3 = mythread(3)
11 t1.start()
12 t2.start()
13 t3.start()
```

I am 1

I am 2

I am 3



第12章 Python 多线程编程



12.2python线程处理



通过直接在线程中运行函数创建线程：

```
1  import threading
2  def func1(x, y):
3      for i in range(x, y):
4          print(i)
5  t1=threading.Thread(target = func1, args = (15, 20))
6  t1.start()
7
8
```

```
9  t1.join()
10 t1 = threading.Thread(target = func1, args = (15, 20))
11 t1.start()
12 t1.join()
13 t1 = threading.Thread(target = func1, args = (15, 20))
14 t1.start()
15
16
17
18
19
```



第12章 Python 多线程编程



12.2python线程处理

Threading模块具有的常用方法：

- `threading.active_count()`：返回当前处于alive状态的Thread对象数量
- `threading.current_thread()`：返回当前Thread对象
- `threading.get_ident()`：返回当前线程的线程标识符，是一个非负整数，并没特殊含义，该整数可能会被循环利用。
- `threading.enumerate()`：返回当前处于alive状态的所有Thread对象列表
- `threading.main_thread()`：返回主线程对象，即启动Python解释器的线程对象。
- `threading.stack_size([size])`：返回创建线程时使用的栈的大小，如果指定size参数，则用来指定后续创建的线程使用的栈大小，size必须是0（表示使用系统默认值）或大于32K的正整数



第12章 Python 多线程编程



✚ 12.2python线程处理

- ✚ Threading模块具有的常用方法：
 - ✚ **Run()**:表示线程活动的方法。
 - ✚ **Start()**:启动现场活动。
- ✚ 可以通过为**Thread**类的构造函数传递一个可调用对象来创建线程。
- ✚ 可以继承**threading.Thread**类创建派生类，并重写**__init__**和**run**方法，实现自定义线程对象类。
- ✚ 创建了线程对象以后，可以调用其**start()**方法来启动，该方法自动调用该类对象的**run**方法，此时该线程处于**alive**状态，直至**run**方法结束。



第12章 Python 多线程编程



12.2python线程处理

Thread类对象的成员:

成员	说明
start()	自动调用run
run()	线程代码，用来实现线程的功能、活动，可以在子类中重写该方法
__init__(self, group=None, target=None, name=None,args=(), kwargs=None, verbose=None)	构造函数
name	用来读取或设置线程的名字
ident	线程标识，非0
is_alive()	测试线程是否处于alive
daemon	布尔值，表示线程是否为守护线程



第12章 Python 多线程编程



12.2python线程处理

join([timeout]): 等待被调线程结束后再继续执行后续代码，timeout为最长等待时间，单位为秒。

```
import threading
```

```
import time
```

```
def func1(x, y):
```

```
    for i in range(x, y):
```

```
        print(i)
```

```
    time.sleep(10)
```

```
t1=threading.Thread(target = func1, args = (15, 20))
```

```
t1.start()
```

```
t1.join(5)
```

```
t2=threading.Thread(target = func1, args = (5, 10))
```

```
t2.start()
```

```
线程3.py - E:\课程\python\第12次课\shili\线程3.py (3.7.3)
File Edit Format Run Options Window Help
import threading
import time
def func1(x, y):
    for i in range(x, y):
        print(i)
        time.sleep(10)

t1=threading.Thread(target = func1, args = (15, 20))
t1.start()
t1.join(5)
t2=threading.Thread(target = func1, args = (5, 10))
t2.start()
```



第12章 Python 多线程编程



12.2python线程处理



`isAlive()`：测试线程是否处于运行状态

主 > 课程 > python > 第12次课 > shili > 4-1.py > ...

```
1 import threading
2 import time
3 def func1(x, y):
4     for i in range(x, y):
5         print(i)
6         #time.sleep(10)
7
8 t1=threading.Thread(target = func1, args = (15, 20))
9 t1.start()
10 t1.join(5) #注释掉这里试试
11 t2=threading.Thread(target = func1, args = (5, 10))
12 t2.start()
13 t2.join() #注释掉这里试试
14
15 print(t1.isAlive())
16 print(t2.isAlive())
17
```

```
15
16
17
18
19
5
6
7
8
9
False
False
```




第12章 Python 多线程编程



12.2python线程处理



`isAlive()`：测试线程是否处于运行状态

```
1  import threading
2  import time
3  def func1(x, y):
4      for i in range(x, y):
5          print(i)
6          #time.sleep(10)
7
8  t1=threading.Thread(target = func1, args = (15, 20))
9  t1.start()
10 #t1.join(5)-注释掉这里试试
11 t2=threading.Thread(target = func1, args = (5, 10))
12 t2.start()
13 #t2.join()-注释掉这里试试
14
15 print(t1.isAlive())
16 print(t2.isAlive())
17
```

```
15
16
17
18True
True5
6
7
8
9
19
```

```
15
16
17
18
195
6
7
8True
9
True
```

```
15
16
17
18
19
True
True
5
6
7
8
9
```



第12章 Python 多线程编程



12.2python线程处理

Thread对象中的daemon属性

- ✚ 在脚本运行过程中有一个主线程，若在主线程中创建了子线程，则：
 - 1) 当子线程的daemon属性为False时，主线程结束时会检测子线程是否结束，如果子线程尚未完成，则主线程会等待子线程完成后再退出；
 - 2) 当子线程的daemon属性为True时，主线程运行结束时不对子线程进行检查而直接退出，同时子线程将随主线程一起结束，而不论是否运行完成。
- ✚ 以上论述不适用于IDLE中的交互模式或脚本运行模式，因为在交互模式下的主线程只有在退出Python时才终止。



第12章 Python 多线程编程



12.2python线程处理



Thread对象中的daemon属性

```
1  import threading
2  import time
3  class mythread(threading.Thread):
4      ...def __init__(self, num, threadname):
5          ...    threading.Thread.__init__(self, name=threadname)
6          ...    self.num = num
7          ...    self.name=threadname
8          ...    #self.daemon=True
9      ...def run(self):
10         ...    time.sleep(self.num)
11         ...    print(self.num,self.name)
12
13     t1 = mythread(1, 't1')
14     t2 = mythread(5, 't2')
15     t2.daemon = True
16     #t2.setDaemon(False)
17     print(t1.daemon)
18     print(t2.daemon)
19
20     t1.start()
21     t2.start()
22
```

```
False
True
1 t1
```



第12章 Python 多线程编程



12.2python线程处理



Thread对象中的daemon属性

```
1 import threading
2 import time
3 class mythread(threading.Thread):
4     def __init__(self, num, threadname):
5         threading.Thread.__init__(self, name = threadname)
6         self.num = num
7         self.name = threadname
8         #self.daemon = True
9     def run(self):
10         time.sleep(self.num)
11         print(self.num, self.name)
12
13 t1 = mythread(1, 't1')
14 t2 = mythread(5, 't2')
15 t2.daemon = True
16 t2.setDaemon(False)
17 print(t1.daemon)
18 print(t2.daemon)
19
20 t1.start()
21 t2.start()
```



```
False
False
1 t1
5 t2
```



第12章 Python 多线程编程



12.2python线程处理

-  线程同步技术之Lock/RLock对象
-  如果多个线程对某个数据进行修改，为了保证数据的正确性，需要对线程进行同步操作。使用Lock可以锁定共享资源。

◆ Lock是比较低级的同步原语，当被锁定以后不属于特定的线程。

一个锁有两种状态：locked和unlocked




- ◆ 如果锁处于unlocked状态，acquire()方法将其修改为locked并立即返回；如果锁已处于locked状态，则阻塞当前线程并等待其他线程释放锁，然后将其修改为locked并立即返回。
- ◆ release()方法将锁状态由locked修改为unlocked并立即返回，如果锁状态本来已经是unlocked，调用该方法将会抛出异常。



第12章 Python 多线程编程



12.2python线程处理

-  可重入锁RLock对象也是一种常用的线程同步原语，可被同一个线程acquire多次。
-  当处于locked状态时，某线程拥有该锁；当处于unlocked状态时，该锁不属于任何线程。
-  RLock对象的acquire()/release()调用对可以嵌套，仅当最后一个或者最外层的release()执行结束，锁被设置为unlocked状态。



第12章 Python 多线程编程



12.2python线程处理

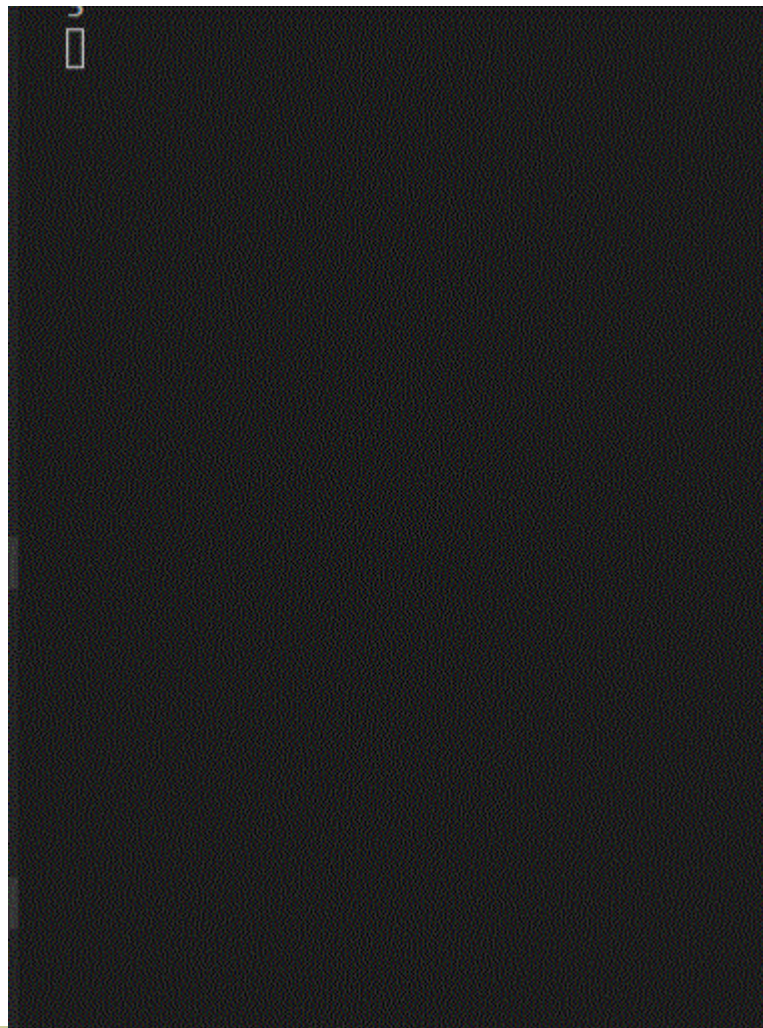
```
1  import threading
2  import time
3  class mythread(threading.Thread):
4      ...def __init__(self):
5          ...threading.Thread.__init__(self)
6      ...def run(self):
7          ...global x
8          ...lock.acquire()...#下面代码的共享数据被锁定
9          ...for i in range(3):
10             ...x = x + i
11             ...time.sleep(2)
12             ...print(x)
13             ...lock.release()...#上面代码的共享数据被解锁释放，此时允许其它程序调用。
14 lock = threading.RLock()...#lock = threading.Lock()
15 tl = []
16 for i in range(10):...#创建10个线程对象，并放入列表中
17     ...t = mythread()
18     ...tl.append(t)
19 x = 0
20 for i in tl:
21     ...i.start()
```




第12章 Python 多线程编程



12.2python线程处理





第12章 Python 多线程编程



❖ 12.3线程优先级模块queue

- ❖ Python的queue模块中提供了同步的、线程安全的队列类，包括FIFO（先入先出）队列Queue，LIFO（后入先出）队列LifoQueue，和优先级队列PriorityQueue。这些队列都实现了锁原语，能够在多线程中直接使用。可以使用队列来实现线程间的同步。
- ❖ Queue对象中的常用方法：
 - ✓ Queue.qsize() 返回队列的大小
 - ✓ Queue.empty() 如果队列为空，返回True,反之False
 - ✓ Queue.full() 如果队列满了，返回True,反之False
 - ✓ Queue.task_done() 在完成一项工作之后，Queue.task_done()函数向任务已经完成的队列发送一个信号



第12章 Python 多线程编程



❖ 12.3线程优先级模块queue

❖ Queue对象中的常用方法：

- ✓ `Queue.get([block[, timeout]])` 获取队列, `timeout`等待时间
- ✓ `Queue.get_nowait()` 相当`Queue.get(False)`,
- ✓ `Queue.put(item)` 写入队列, `timeout`等待时间
- ✓ `Queue.put_nowait(item)` 相当`Queue.put(item, False)`
- ✓ `Queue.join()` 实际上意味着等到队列为空, 再执行别的操作



第12章 Python 多线程编程



❖ 12.3线程优先级模块queue

❖ Queue对象中的常用方法：

- ✓ `Queue.get([block[, timeout]])` 获取队列, `timeout`等待时间
- ✓ `Queue.get_nowait()` 相当`Queue.get(False)`,
- ✓ `Queue.put(item)` 写入队列, `timeout`等待时间
- ✓ `Queue.put_nowait(item)` 相当`Queue.put(item, False)`
- ✓ `Queue.join()` 实际上意味着等到队列为空, 再执行别的操作



第12章 Python 多线程编程



12.3 线程优先级模块queue

Queue实例:

```
E: > 课程 > python > 第12次课 > shili > 7.py > ...
```

```
1 import threading
2 import queue
3 q=queue.Queue()
4 ✓ for i in range(5):
5     |... q.put(i)
6 ✓ while not q.empty():
7     |... print(q.get())
```

```
0
1
2
3
4
```



第12章 Python 多线程编程



12.3 线程优先级模块queue

Queue实例:

```
1  import threading
2  import queue
3  import time
4  class myThread(threading.Thread):
5      def __init__(self, threadname):
6          threading.Thread.__init__(self)
7          self.threadname=threadname
8      def run(self):
9          print('线程{}启动!\n'.format(self.threadname))
10         while not thrend:
11             queueLock.acquire()
12             if not q.empty():
13                 print(self.threadname, '正在打印queue队列里面的', q.get(), '数据。 \n')
14                 queueLock.release()
15             else:
16                 print('queue队列数据取出完毕，线程{}退出!\n'.format(self.threadname))
17                 queueLock.release()
18                 break
19             time.sleep(5)
20 q=queue.Queue()
21 thrend=0
22 threadnamelist=['t1','t2','t3','t4','t5']
23 ths=[]
24 for i in range(1,20):
25     q.put(i)
26 queueLock=threading.RLock()
27 for name in threadnamelist:
28     t=myThread(name)
29     t.start()
30     ths.append(t)
31 for t in ths:
32     t.join()
33 print('主线程退出')
```




第12章 Python 多线程编程



12.3 线程优先级模块queue

Queue实例:

线程t1启动!
线程t5启动!
线程t2启动!
线程t3启动!
线程t4启动!

t1 正在打印queue队列里面的 1 数据。
t5 正在打印queue队列里面的 2 数据。
t2 正在打印queue队列里面的 3 数据。
t3 正在打印queue队列里面的 4 数据。
t4 正在打印queue队列里面的 5 数据。
t1 正在打印queue队列里面的 6 数据。
t5 正在打印queue队列里面的 7 数据。
t2 正在打印queue队列里面的 8 数据。
t3 正在打印queue队列里面的 9 数据。
t4 正在打印queue队列里面的 10 数据。
t1 正在打印queue队列里面的 11 数据。
t5 正在打印queue队列里面的 12 数据。
t2 正在打印queue队列里面的 13 数据。
t3 正在打印queue队列里面的 14 数据。
t4 正在打印queue队列里面的 15 数据。
t1 正在打印queue队列里面的 16 数据。
t5 正在打印queue队列里面的 17 数据。
t2 正在打印queue队列里面的 18 数据。
t3 正在打印queue队列里面的 19 数据。
queue队列数据取出完毕, 线程t4退出!
queue队列数据取出完毕, 线程t1退出!
queue队列数据取出完毕, 线程t5退出!
queue队列数据取出完毕, 线程t2退出!
queue队列数据取出完毕, 线程t3退出!
主线程退出

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
```




第12章 Python 多线程编程



12.3 线程优先级模块queue

LIFO (后入先出) 队列LifoQueue:

```
1 import threading
2 import queue
3 import time
4 class myThread (threading.Thread):
5     def __init__(self, threadname):
6         threading.Thread.__init__(self)
7         self.threadname=threadname
8     def run(self):
9         print('线程{}启动!\n'.format(self.threadname))
10        while not thrend:
11            queueLock.acquire()
12            if not q.empty():
13                print(self.threadname,'正在打印queue队列里面的',q.get(),'数据。 \n')
14                queueLock.release()
15            else:
16                print('queue列队数据取出完毕，线程{}退出!\n'.format(self.threadname))
17                queueLock.release()
18                break
19            time.sleep(1)
20 q=queue.LifoQueue()
21 thrend=0
22 threadnamelist=['t1','t2','t3','t4','t5']
23 ths=[]
24 for i in range(1,20):
25     q.put(i)
26 queueLock=threading.RLock()
27 for name in threadnamelist:
28     t=myThread(name)
29     t.start()
30     ths.append(t)
31 for t in ths:
32     t.join()
33 print('主线程退出')
```

t1 正在打印queue队列里面的 19 数据。
t3 正在打印queue队列里面的 18 数据。
t2 正在打印queue队列里面的 17 数据。
t4 正在打印queue队列里面的 16 数据。
t5 正在打印queue队列里面的 15 数据。
t1 正在打印queue队列里面的 14 数据。
t3 正在打印queue队列里面的 13 数据。
t2 正在打印queue队列里面的 12 数据。
t4 正在打印queue队列里面的 11 数据。
t5 正在打印queue队列里面的 10 数据。
t1 正在打印queue队列里面的 9 数据。
t3 正在打印queue队列里面的 8 数据。
t2 正在打印queue队列里面的 7 数据。
t4 正在打印queue队列里面的 6 数据。
t5 正在打印queue队列里面的 5 数据。
t1 正在打印queue队列里面的 4 数据。
t3 正在打印queue队列里面的 3 数据。
t2 正在打印queue队列里面的 2 数据。
t4 正在打印queue队列里面的 1 数据。
queue列队数据取出完毕，线程t5退出!
queue列队数据取出完毕，线程t1退出!
queue列队数据取出完毕，线程t3退出!
queue列队数据取出完毕，线程t2退出!
queue列队数据取出完毕，线程t4退出!
主线程退出



第12章 Python 多线程编程



12.4 使用模块subprocess创建进程

- subprocess模块允许我们生成一个新进程，连接到新进程的输出、输入、错误信息，并获取它们的返回码。
- subprocess模块用以替换一些老版本python的老旧模块

```
1 | os.system  
2 | os.spawn*  
3 | os.popen*  
4 | popen2.*  
5 | commands.*
```



第12章 Python 多线程编程



12.4使用模块subprocess创建进程

- 在subprocess模块中提供了3个方法和一个类来实现运行管理子进程，官方建议普通的运用中我们可以直接调用提供的函数来快速完成子进程的运行，并获取运行结果。

✓ 第一种创建进程方式：

`subprocess.call(args, *, stdin=None, stdout=None, stderr=None, shell=False)`

args: 可以是一个string，数组或者列表。用来存放需要执行的命令和参数，例如 ‘echo hello’ 或者('echo', 'hello')

stdin, stdout, stderr分别就是标准输出，标准输入，标准错误输出。默认值则继承自父进程的相关设定，也可以指向PIPE或一个文件对象。

shell:布尔值，是否调用一个shell来执行。当为true时会调用系统默认的shell环境来执行命令。对于windows的环境来说，只有当需要运行cmd内置命令的时候才需要设为true。



第12章 Python 多线程编程



12.4 使用模块subprocess创建进程

✓ 第一种创建进程方式:

`subprocess.call(args, *, stdin=None, stdout=None, stderr=None, shell=False)`

```
1 import subprocess
2 return_code = subprocess.call('C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe')
3 print(return_code)
4
```

```
1 import subprocess
2 return_code = subprocess.call(r'python E:\课程\python\第12次课\shili\7.py', shell=True)
3 print(return_code)
4
```

```
1 import subprocess
2 return_code = subprocess.call(['python', 'E:\\课程\\python\\第12次课\\shili\\7.py'])
3 print(return_code)
4
```



第12章 Python 多线程编程



12.4使用模块subprocess创建进程

✓ 第二种创建进程方式：使用类Popen创建进程

```
class subprocess.Popen( args,  
    bufsize=0,  
    executable=None,  
    stdin=None,  
    stdout=None,  
    stderr=None,  
    preexec_fn=None,  
    close_fds=False,  
    shell=False,  
    cwd=None,  
    env=None,  
    universal_newlines=False,  
    startupinfo=None,  
    creationflags=0)
```

args:

args参数。可以是一个字符串，可以是一个包含程序参数的列表。要执行的程序一般就是这个列表的第一项，或者是字符串本身。



第12章 Python 多线程编程



12.4 使用模块subprocess创建进程

✓ 第二种创建进程方式：使用类Popen创建进程

```
1 import subprocess
2 return_code = subprocess.Popen(r'C:\Program Files\Mozilla Firefox\firefox.exe')
3 print(return_code)
4
```

```
1 import subprocess
2 return_code = subprocess.Popen(r'python E:\课程\python\第12次课\shili\7.py', shell=True)
3 print(return_code)
4
```

```
1 import subprocess
2 return_code = subprocess.Popen(['python', 'E:\\课程\\python\\第12次课\\shili\\7.py'])
3 print(return_code)
4
```



第12章 Python 多线程编程



12.4使用模块subprocess创建进程

✓ 第二种创建进程方式：使用类Popen创建进程

- .poll() 返回子进程运行状态，主要是两种结果，None代表尚未运行完，而一个返回码则代表已经运行完成并且是成功或失败了
- .kill() 强行终止子进程
- .send_signal(...) 向子进程发送一个信号
- .terminate() 终止子进程
- .pid 子进程的pid
- .returncode 子进程的返回码
- .stdin/stdout/stderr 子进程的标准输入流，标准输出和标准错误输出，都是类文件对象



下周课程&课后作业



第11章 Python 图形化界面开发基础

课后练习

- 编写含有三个线程的程序（不包括主线程），并使用 LOCK 锁锁定共享资源。

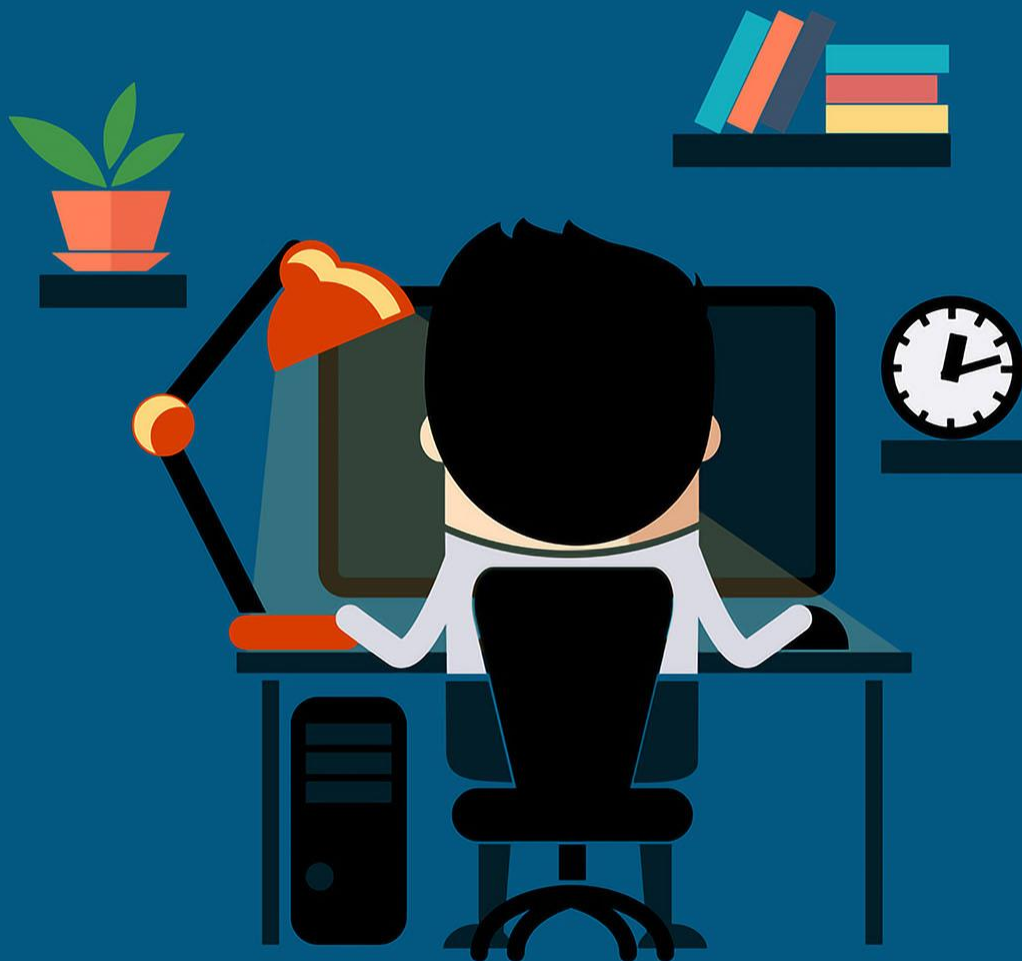
代码文件命名：12-1thrLoc

- 打开一个子进程并获取其进程ID

代码文件命名：11-2ProID

.py 代码文件打包(12.学号
+ 姓名)发送到
python_xxmu@163.com

编程辣么好，还等什么？开始学习吧！



Programing is an Art