# Threading模块

## 如何初始化一个线程示例：

### 第一种方式

创建一个threading.Thread()的实例，给它一个函数。

import threading

from time import sleep, ctime

loops = [4, 2]

def loop(nloop, nsec):

print '\nstart loop:', nloop, 'at:', ctime()

sleep(nsec)

print '\nloop', nloop, 'done at:', ctime()

def main():

print '\nstarting at:\n', ctime()

threads = []

nloops = range(len(loops))

#create the threads using threading

for i in nloops:

t = threading.Thread(target=loop, args=(i,loops[i]))

threads.append(t)

#start threads

for i in nloops:

threads[i].start()

#wait for all

for i in nloops:

threads[i].join() #block

print '\nall Done at:', ctime()

if \_\_name\_\_ == '\_\_main\_\_':

main()

/-----------------------------------------------------------------------------------------------------------------------------------------------------

### 第二种方式

创建一个threading.Thread的实例，传给它一个可调用类对象，类中使用\_\_call\_\_()函数调用函数

import threading

from time import sleep, ctime

loops = [4,2]

class ThreadFunc(object):

def \_\_init\_\_(self, func, args, name=''):

self.name = name

self.func = func

self.args = args

#when you create a new thread, Thread instance will invoke our ThreadFunc

# instance, and at that time, it will call the function \_\_call\_\_()

# Hence, we have a tuple arguments , so we use the self.res

def \_\_call\_\_(self):

self.res = self.func(\*self.args) #invoke the func

def loop(nloop, nsec):

print '\nstart loop:', nloop, 'at:', ctime()

sleep(nsec)

print '\nloop', nloop, 'done at:', ctime()

def main():

print 'start at:', ctime()

threads = [] #list

nloops = range(len(loops))

#crate all threads

for i in nloops:

t = threading.Thread(target=ThreadFunc(loop, (i, loops[i]),loop.\_\_name\_\_))

threads.append(t)

#start all threads

for i in nloops:

threads[i].start()

for i in nloops:

threads[i].join()

print 'all Done at:', ctime()

if \_\_name\_\_ == '\_\_main\_\_':

main()

/-----------------------------------------------------------------------------------

### 第三种方式

派生一个threading.Thread出一个子类，创建这个子类的实例，使用run调用函数

import threading

from time import sleep, ctime

loops = (4,2)

class MyThread(threading.Thread):

def \_\_init\_\_(self, func, args, name=''):

threading.Thread.\_\_init\_\_(self) #base class func

self.name = name

self.func = func

self.args = args

# in the other way, use \_\_call\_\_()

def run(self):

apply(self.func,self.args)

def loop(nloop, nsec):

print 'start loop', nloop, 'at:', ctime()

sleep(nsec)

print 'loop', nloop, 'done at:', ctime()

def main():

print 'starting at:',ctime()

threads = []

nloops = range(len(loops))

for i in nloops:

t = MyThread(loop, (i,loops[i]), loop.\_\_name\_\_)

threads.append(t)

for i in nloops:

threads[i].start()

for i in nloops:

threads[i].join()

print 'all Done at:', ctime()

if \_\_name\_\_ == '\_\_main\_\_':

main()

## 如何使用锁

<http://www.oschina.net/code/snippet_16840_1815>

## 如何使用condition，event对象

<http://www.cnblogs.com/twelfthing/articles/2095502.html>

# Python库管理

Python库管理主要包含如下几个库，他们的依赖关系按照从上到下的顺序依次而来。

* Easy\_install
* Setuptools
* Distribute
* Pypi

最后的库安装在libs目录下建立了有easy\_install的路径文件，里面明文记录了安装好的python库，如果直接安装distribute，会自动安装easy\_install和setuptools，之后再安装pypi。

安装好后在python的script目录下会存在两个exe文件，easy\_install和pip.exe，两者能在命令行下执行，安装和卸载python库，相比较而言，pip的反馈信息和命令行接口更为完善。

# Python的Logging模块

# 原始套接字编程

sock\_raw原始套接字编程可以接收到本机网卡上的数据帧或者数据包,对与监听网络的流量和分析是很有作用的.一共可以有3种方式创建这种 socket

1.socket(AF\_INET, SOCK\_RAW, IPPROTO\_TCP|IPPROTO\_UDP|IPPROTO\_ICMP)发送接收ip数据包

2.socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_IP|ETH\_P\_ARP|ETH\_P\_ALL))发送接收以太网数据帧

3.socket(AF\_INET, SOCK\_PACKET, htons(ETH\_P\_IP|ETH\_P\_ARP|ETH\_P\_ALL))过时了,不要用啊

理解一下SOCK\_RAW的原理, 比如网卡收到了一个 14+20+8+100+4 的udp的以太网数据帧.

首先,网卡对该数据帧进行硬过滤(根据网卡的模式不同会有不同的动作,如果设置了promisc混杂模式的话,则不做任何过滤直接交给下一层输 入例程,否则非本机mac或者广播mac会被直接丢弃).按照上面的例子,如果成功的话,会进入ip输入例程.但是在进入ip输入例程之前,系统会检查系 统中是否有通过socket(PF\_PACKET, SOCK\_RAW, ..)创建的套接字.如果有的话并且协议相符,在这个例子中就是需要ETH\_P\_IP或者ETH\_P\_ALL类型.系统就给每个这样的socket接收缓 冲区发送一个数据帧拷贝.然后进入下一步.

其次,进入了ip输入例程(ip层会对该数据包进行软过滤,就是检查校验或者丢弃非本机ip或者广播ip的数据包等,具体要参考源代码),例子 中就是如果成功的话会进入udp输入例程.但是在交给udp输入例程之前,系统会检查系统中是否有通过socket(AF\_INET, SOCK\_RAW, ..)创建的套接字.如果有的话并且协议相符,在这个例子中就是需要IPPROTO\_UDP类型.系统就给每个这样的socket接收缓冲区发送一个数据 帧拷贝.然后进入下一步.

最后,进入udp输入例程 ...

ps:如果校验和出错的话,内核会直接丢弃该数据包的.而不会拷贝给sock\_raw的套接字,因为校验和都出错了,数据肯定有问题的包括所有信息都没有意义了.

进一步分析他们的能力.

1. socket(AF\_INET, SOCK\_RAW, IPPROTO\_UDP);

能:该套接字可以接收协议类型为(tcp udp icmp等)发往本机的ip数据包,从上面看的就是20+8+100.

不能:不能收到非发往本地ip的数据包(ip软过滤会丢弃这些不是发往本机ip的数据包).

不能:不能收到从本机发送出去的数据包.

发送的话需要自己组织tcp udp icmp等头部.可以setsockopt来自己包装ip头部

这种套接字用来写个ping程序比较适合

2. socket(PF\_PACKET, SOCK\_RAW, htons(x));

这个套接字比较强大,创建这种套接字可以监听网卡上的所有数据帧.从上面看就是20+20+8+100.最后一个以太网crc从来都不算进来 的,因为内核已经判断过了,对程序来说没有任何意义了.

能: 接收发往本地mac的数据帧

能: 接收从本机发送出去的数据帧(第3个参数需要设置为ETH\_P\_ALL)

能: 接收非发往本地mac的数据帧(网卡需要设置为promisc混杂模式)

协议类型一共有四个

ETH\_P\_IP 0x800      只接收发往本机mac的ip类型的数据帧

ETH\_P\_ARP 0x806      只接受发往本机mac的arp类型的数据帧

ETH\_P\_RARP 0x8035     只接受发往本机mac的rarp类型的数据帧

ETH\_P\_ALL 0x3         接收发往本机mac的所有类型ip arp rarp的数据帧, 接收从本机发出的所有类型的数据帧.(混杂模式打开的情况下,会接收到非发往本地mac的数据帧)

发送的时候需要自己组织整个以太网数据帧.所有相关的地址使用struct sockaddr\_ll 而不是struct sockaddr\_in(因为协议簇是PF\_PACKET不是AF\_INET了),比如发送给某个机器,对方的地址需要使用struct sockaddr\_ll.

这种socket大小通吃,强悍

下面是一段相关的代码:

|  |
| --- |
| ...     int sockfd = socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ALL));     struct sockaddr\_ll sll;      memset( &sll, 0, sizeof(sll) );      sll.sll\_family = PF\_PACKET;     struct ifreq ifstruct;    strcpy(ifstruct.ifr\_name, "eth0");    ioctl(sockfd, SIOCGIFINDEX, &ifstruct);     sll.sll\_ifindex = ifstruct.ifr\_ifindex;      sll.sll\_protocol = htons(ETH\_P\_ALL);     if(bind(fd, (struct sockaddr \*) &sll, sizeof(sll)) == -1 ) {        perror("bind()");     ... |

|  |
| --- |
| int set\_promisc(char \*interface, int fd) {         struct ifreq ifr;          strcpy(ifr.ifr\_name, interface);         if(ioctl(fd, SIOCGIFFLAGS, &ifr) == -1) {                  perror("iotcl()");                  return -1;          }          ifr.ifr\_flags |= IFF\_PROMISC;         if(ioctl(fd, SIOCSIFFLAGS, &ifr) == -1) {                  perror("iotcl()");                  return -1;          }          return 0; }  int unset\_promisc(char \*interface, int fd) {         struct ifreq ifr;          strcpy(ifr.ifr\_name, interface);         if(ioctl(fd, SIOCGIFFLAGS, &ifr) == -1) {                  perror("iotcl()");                  return -1;          }          ifr.ifr\_flags &= ~IFF\_PROMISC;         if(ioctl(fd, SIOCSIFFLAGS, &ifr) == -1) {                  perror("iotcl()");                  return -1;          }          return 0; } |

3. socket(AF\_INET, SOCK\_PACKET, htons(ETH\_P\_ALL))这个最好不要用,反正我不用...

总结使用方法: 1.只想收到发往本机某种协议的ip数据包的话用第一种就足够了

            2. 更多的详细的内容请使用第二种.包括ETH\_P\_ALL参数和混杂模式都可以使它的能力不断的加强.

ps:很多自己的想法.虚拟机测试环境.有错欢迎指出交流

qq:110024218

# Python的SSL模块