forking工作原理

1.什么是forking

fork(分岔)在Linux系统中使用非常广泛

当某一命令执行时,父进程(当前进程)fork出一个子进程

父进程将自身资源拷贝一份,命令在子进程中运行时,就具有和父进程完全一样的运行环境

1. 僵尸进程

僵尸进程没有任何可执行代码,也不能被调度

如果系统中存在过多的僵尸进程,将因为没有可用的进程号而导致系统不能产生新的进程

对于系统管理员来说,可以试图杀死其父进程或重启系统来消除僵尸进程

forking编程

# import os

#

# print('startion..')

# os.fork() #生成子进程，后续代码将在父子进程中同时运行

# print('Hello world!')

# print('starting...')

# pid = os.fork() #父进程返回值是子进程的pid,子进程的返回值是0

# if pid:

# print('in parent')

# else:

# print('in child')

# print('done')

扫描存活主机

1. 通过ping测试主机是否可达

2. 如果ping不通,不管什么原因都认为主机不可用

3. 通过fork方式实现并发扫描

# import subprocess

# import os

#

#

# def ping(host):

# result = subprocess.call(

# 'ping -c2 %s &> /dev/null ' % host,

# shell = True

# )

# if result == 0:

# print('%s up' % host)

# else:

# print('%s down' % host)

#

#

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

# ips = ['176.121.207.%s' % i for i in range(1,255) ]

# for ip in ips:

# pid = os.fork() #父进程负责生成子进程

# if not pid: #子进程负责ping

# ping(ip)

# exit() #子进程ping完一个地址结束，不在循环

forking基础应用

编写一个forking脚本

1. 在父进程中打印“In parent”然后睡眠10秒

2. 在子进程中编写循环,循环5次,输出当系统时间,每次循环结束后睡眠1秒

3. 父子进程结束后,分别打印“parent exit”和“child exit”

# import os

# import time

#

# pid = os.fork()

#

# if pid:

# print('in parent.sleeping...')

# time.sleep(60)

# print(os.waitpid(-1,1)) #收拾子进程，如果子进程是僵尸进程，释放他

# #如果不是，父进程什么也不做，继续向下执行

# time.sleep(20)

#

# else:

# print('in child, slwwping...')

# time.sleep(15)

#

利用fork创建TCP服务器

编写TCP服务器

1. 服务器监听在0.0.0.0的21567端口上

2. 收到客户端数据后,将其加上时间戳后回送给客户端

3. 如果客户端发过来的字符全是空白字符,则终止与客

户端的连接

4. 服务器能够同时处理多个客户端的请求

5. 程序通过forking来实现

# import socket

#

# from time import strftime

# import os

# class TcpTimeServer:

# def \_\_init\_\_(self,host='',port=12345):

# self.addr = (host,port)

# self.serv = socket.socket()

# self.serv.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1)

# self.serv.bind(self.addr)

# self.serv.listen(1)

#

# def handle\_child(self,cli\_sock):

# while True:

# data = cli\_sock.recv(1024)

# if data.strip() == b'quit':

# break

# data = data.decode('utf8')

# sdata = '[%s] %s' % (strftime('%H:%M:%S'),data)

# cli\_sock.send(sdata.encode('utf8'))

# cli\_sock.close()

#

# def mainloop(self):

# while True:

# try:

# cli\_sock,cli\_addr = self.serv.accept()

# except KeyboardInterrupt:

# break

# pid = os.fork

# if pid:

# while True:

# result = os.waitpid(-1,1)[0]

# if result == 0:

# break

# cli\_sock.close()

# else:

# self.handle\_child(cli\_sock)

# self.serv.close()

# exit()

# self.serv.close()

#

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

# s = TcpTimeServer()

# s.mainloop()

1.多线程

2.多线程相关模块

thread和threading模块允许程序员创建和管理线程

hread模块提供了基本的线程和锁的支持,而threading提供了更高级别、功能更强的线程管理功能

推荐使用更高级别的threading模块

3.传递函数给Thread类

多线程编程有多种方法,传递函数给threading模块的Thread类是介绍的第一种方法

Thread对象使用start()方法开始线程的执行,使用join()方法挂起程序,直到线程结束

# 多进程，父进程将自身资源拷贝一份，生成子进程，程序在子进程中运行。所以，子进程和父进程都有各自独立的运行空间

# 多线程，包含在一个进程内，多个工作线程共享同一进程的资源。多线程没有僵尸问题

# GIL: 全局解释器锁。GIL的存在，导致某一时刻，只能把一个线程交给CPU处理

# 计算密集型程序：数据的提供不是问题，主要是CPU的效率，适全多进程

# IO密集型程序：CPU再快没用，因为它获得不到想要的数据，适合py多线程

# import subprocess

# import os

# import threading

#

# def ping(host):

# result = subprocess.call(

# 'ping -c2 %s &> /dev/null ' % host,

# shell = True

# )

# if result == 0:

# print('%s up' % host)

# else:

# print('%s down' % host)

#

#

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

# ips = ['176.121.207.%s' % i for i in range(1,255) ]

# for ip in ips: #主线程用户产生工作线程

# t = threading.Thread(target=ping,args=(ip,))

# t.start() #将会执行ping(ip),执行完就停止了

# #主线程不会等待工作线程结束，直接进入下次循环

扫描存活主机

1. 通过ping测试主机是否可达

2. 如果ping不通,不管什么原因都认为主机不可用

3. 通过多线程方式实现并发扫描

# import subprocess

# import threading

#

# class Ping:

# def \_\_init\_\_(self,host):

# self.host = host

#

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

# result = subprocess.call(

# 'ping -c2 %s &> /dev/null ' % self.host,

# shell = True

# )

# if result == 0:

# print('%s up' % self.host)

# else:

# print('%s down' % self.host)

#

#

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

# ips = ['176.121.207.%s' % i for i in range(1,255) ]

# for ip in ips: #主线程用户产生工作线程

# t = threading.Thread(target=Ping(ip))

# t.start() #Ping(ip)() <==> target()

# import time

# import os

# import threading

# def calc():

# result = 0

# for i in range(50000000):

# result += i

# print(result)

#

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

# start = time.time()

# calc()

# calc()

# end = time.time()

# print(end - start)

#####################

# start = time.time()

# pid = os.fork()

# if not pid:

# calc()

# exit()

# pid = os.fork()

# if not pid:

# calc()

# exit()

# os.waitpid(-1,0) #处理子进程，挂起父进程

# os.waitpid(-1,0)

# end = time.time()

# print(end - start)

####################################

# start = time.time()

# t1 = threading.Thread(target=calc)

# t1.start()

# t2 = threading.Thread(target=calc)

# t2.start()

# t1.join() #挂起主线程，等到t1线程结束后，再向下执行

# t2.join()

# end = time.time()

# print(end - start)

课堂练习：

百鸡百钱

for i in range(100//5): # //表示只留商，不要小数，舍弃余数

for j in range(100//3):

for k in range(100):

if i + j + k == 100 and i \* 5 + j \* 3 + k // 3 == 100:

print('公鸡:%s, 母鸡:%s,小鸡:%s' % (i, j, k))