NSD Devops DAY01

1 案例1：forking基础应用

1.1 问题

编写一个myfork.py脚本，实现以下功能：

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，输出当前系统时间，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”

1.2 方案

子进程运行时是从 pid = os.fork() 下面语句执行，实际上，该语句是两条语句， os.frok() 是创建子进程语句，而 pid = 是赋值语句，所以在创建完子进程后，下一句为运行赋值语句。

进程调用fork函数时，操作系统会新建一个子进程，它本质上与父进程完全相同。操作系统是将当前的进程（父进程）复制了一份（子进程），然后分别在父进程和子进程内返回。子进程接收返回值为0，此时pid=0，而父进程接收子进程的pid作为返回值。调用fork函数后，两个进程并发执行同一个程序，首先执行的是调用了fork之后的下一行代码。

此时，pid两个值，同时满足判断语句if和else，按照顺序执行如下：

父进程先执行：程序先输出“In parent!”，然后父进程睡眠10s，即进程挂起10s

父进程挂起时，子进程开始执行：循环5次，每循环一次打印当前时间后睡眠1s，5s后结束五次循环，打印“child exit”，此时子进程已经结束

子进程接收后，父进程挂起尚未结束，当父进程睡眠时间结束后，打印“parent exit”，父进程也结束了。

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
01.  [root@localhost day09]# vim myfork.py
02.  #! /usr/bin/env python3
03.
04.  import os
05.  import time
06.  from datetime import datetime
07.
```

Top

```
pid = os.fork( )
if pid:
    print( "In parent! ")
    time.sleep( 10)
    print( "parent exit")
else:
    for i in range( 5) :
        print( datetime.now( ) )
        time.sleep( 1)
    print( "child exit")
```

步骤二：测试脚本执行

```
[ root@localhost day09] # python3 myfork.py
In parent!
2018- 09- 03 10: 48: 46.552528
2018- 09- 03 10: 48: 47.553714
2018- 09- 03 10: 48: 48.554800
2018- 09- 03 10: 48: 49.555901
2018- 09- 03 10: 48: 50.557035
child exit
parent exit
```

2 案例2：扫描存活主机

2.1 问题

创建forkping.py脚本，实现以下功能：

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

2.2 方案

定义函数ping()，该函数可实现允许ping通任何主机功能：

1.引用subprocess模块执行shell命令ping所有主机，将执行结果返回给rc变量，此时，如果ping不通返回结果为1，如果能ping通返回结果为0

2.如果rc变量值不为0，表示ping不通，输出down

3.否则，表示可以ping通，输出up

利用列表推导式生成整个网段的IP地址列表[172.40.58.1,172.40.58.2....]

循环遍历整个网段列表，每循环出一个ip，os.fork()生成1个子进程和1个父进程，

Top

此时，如果pid返回值为0，子进程以ip作为实际参数调用ping函数，调用后一定要exit()，确保子进程ping完一个地址后结束，不要再循环生成父子进程。

```
subprocess.call(
    'ping -c2 %s &> /dev/null' %host,
    shell=True
)
```

注意：shell命令ping所有主机时，ping发送一个ICMP请求，并且将输出重定向到/dev/null。这条语句返回其实就是ping值，就是python程序先创建shell进程，shell创建ping进程，ping进程运行返回值被shell等待，shell返回值给python程序wait，如果成功则为0.

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day09]#vim forkping.py

#!/usr/bin/env python3

import subprocess       #加载支持Linux系统内部命令模块
import os
#定义函数，允许ping任何主机，ping函数需要给IP作为参数
def ping(host):
    rc=subprocess.call(
        'ping -c2 %s &> /dev/null' %host,
        shell=True
    )           #定义ping命令的变量，返回值0:正常，返回值1：ping不通
    if rc:
        print('%s: down' %host)    #无法ping通打印down
    else:
        print('%s: up' %host)       #当re=0，表示可以ping通，打印up

if __name__=='__main__':
    #生成整个网段的IP列表[172.40.58.1,172.40.58.2....]
    ips=['172.40.58.%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:    #子进程负责调用ping函数
            ping(ip)
```

Top

```
25.                 exit( )         # 子进程ping完一个地址后结束，不要再循环
```

步骤二：测试脚本执行

```
01.  [ root@localhost day 09] # py thon3 forkping. py
02.  [ root@localhost day 09] # 172. 40. 58. 69: up
03.  172. 40. 58. 1: up
04.  172. 40. 58. 87: up
05.  172. 40. 58. 90: up
06.  172. 40. 58. 102: up
07.  172. 40. 58. 111: up
08.  172. 40. 58. 106: up
09.  172. 40. 58. 101: up
10.  172. 40. 58. 110: up
11.  172. 40. 58. 109: up
12.  172. 40. 58. 105: up
13.  172. 40. 58. 119: up
14.  …
15.  …
16.  …
17.  172. 40. 58. 14: down
18.  172. 40. 58. 15: down
19.  172. 40. 58. 6: down
20.  172. 40. 58. 5: down
21.  172. 40. 58. 10: down
22.  …
23.  …
24.  #未执行完毕。。。
```

3 案例3：利用fork创建TCP服务器

3.1 问题

创建tcp_time_server.py文件，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

Top

3.2 方案

面向对象编程方法编写TCP服务器：

1）TcpTimeServer():创建TcpTimeServer类

2）__init__():创建对象后自动调用init方法，初始化以下属性：

建立socket对象。

设置socket选项，当socket关闭后，本地端用于该socket的端口号立刻就可以被重用。

绑定socket对象IP和端口。

将套接字设为监听模式，准备接收客户端请求。利用listen()函数进行侦听连接。

3）将创建的TcpTimeServer()对象返回给s，让s实例来保存该对象

4）调用s.mainloop()方法：

利用while循环，accept()会等待并返回一个客户端的连接

os.fork生成子进程

然后进行if判断，如果是服务器的话（即父进程），关闭客户端套接字，并利用os.waitpid返回值循环处理客户机僵尸进程，僵尸进程处理完毕，结束循环，父进程重新开始循环，进入accept()连接去等待

如果是客户端的话（即子进程），关闭服务器套接字，并调用chat()方法，去与客户机聊天去。子进程结束后exit()退出。

5)调用chat()方法，用返回的客户端作为参数

循环将recv接收到的数据加上时间戳后send发送给客户端

关闭套接字，释放资源。

6)此时服务器能够同时处理多个客户端的请求，以多进程方式

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day09] # vim tcp_time_client.py
#! /usr/bin/env python3
import socket
import os
from time import strftime

class TcpTimeServer:
    def __init__(self, host='', port=21567):
        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 chat(self, c_sock):
```

Top

```
            while True:
                data = c_sock.recv(1024)
                if data.strip() == b'quit':
                    break
                data = '[%s] %s' %(strftime('%H:%M:%S'), data.decode('utf8'))
                c_sock.send(data.encode('utf8'))
            c_sock.close()

        def mainloop(self):
            while True:
                cli_sock, cli_addr = self.serv.accept()
                pid = os.fork()
                if pid:
                    cli_sock.close()
    #取出waitpid元组中第一个数，优先处理僵尸进程
    #waitpid()的返回值：如果子进程正在运行、尚未结束则返回0，否则返回子进程的PID
                    while True:
                        result = os.waitpid(-1, 1)[0]
                        if result == 0:
                            break
                    else:
                        self.serv.close()
                        self.chat(cli_sock)
                        exit()

            self.serv.close()

    if __name__ == '__main__':
        s = TcpTimeServer()
        s.mainloop()
```

步骤二：测试脚本执行

```
01. 执行脚本，启动服务
02. [root@localhost day09]# python3 tcp_time_server.py
```

以下两个客户端同时telnet与服务器端连接，可实现多用户通信：

Top

```
01. [root@localhost day09]# telnet 172.40.58.189 21567
```

```
Trying 172.40.58.189...
Connected to 172.40.58.189.
Escape character is '^]'.
nihao
[19:37:36] nihao
nizainali
[19:37:42] nizainali
hello world
[19:37:52] hello world
quit
Connection closed by foreign host.
[root@localhost day09]# telnet 172.40.58.189 21567
Trying 172.40.58.189...
Connected to 172.40.58.189.
Escape character is '^]'.
hello lilei
[19:38:33] hello lilei
I'm fine
[19:38:45] I'm fine
quit
Connection closed by foreign host.
```

4 案例4：扫描存活主机

4.1 问题

创建mtping.py脚本，实现以下功能：

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

4.2 方案

subprocess.call ()方法可以调用系统命令，其返回值是系统命令退出码，也就是如果系统命令成功执行，返回0，如果没有成功执行，返回非零值。

调用Ping对象，可以调用系统的ping命令，通过退出码来判断是否ping通了该主机。如果顺序执行，每个ping操作需要消耗数秒钟，全部的254个地址需要10分钟以上。而采用多线程，可以实现对这254个地址同时执行ping操作，并发的结果就是将执行时间缩短到了10秒钟左右。

4.3 步骤

实现此案例需要按照如下步骤进行。

Top

步骤一：编写脚本

```
[root@localhost day09]# vim mtping.py
#! /usr/bin/env python3

import subprocess
import threading

def ping(host):
    rc = subprocess.call(
        'ping -c2 %s &> /dev/null' % host,
        shell=True
    )
    if rc:
        print('%s: down' % host)
    else:
        print('%s: up' % host)

if __name__ == '__main__':
    ips = ['172.40.58.%s' % i for i in range(1, 255)]
    for ip in ips:
        # 创建线程，ping是上面定义的函数，args是传给ping函数的参数
        t = threading.Thread(target=ping, args=(ip,))
        t.start()  # 执行ping(ip)
```

面向对象代码编写方式如下：

定义Ping类，该类可实现允许ping通任何主机功能：

1.利用__init__方法初始化参数，当调用Ping类实例时，该方法自动调用

2. 利用__call__()方法让Ping类实例变成一个可调用对象调用，调用t.start()时， 引用subprocess模块执行shell命令ping所有主机，将执行结果返回给rc变量，此时，如果ping不通返回结果为1，如果能ping通返回结果为0

3.如果rc变量值不为0，表示ping不通，输出down

4.否则，表示可以ping通，输出up

利用列表推导式生成整个网段的IP地址列表[172.40.58.1,172.40.58.2....]

循环遍历整个网段列表，直接利用 Thread 类来创建线程对象，执行Ping(ip)。

```
[root@localhost day09]# vim mtping2.py
#! /usr/bin/env python3

import threading
import subprocess
```

Top

```
06.
07.     class Ping:
08.         def __init__(self, host):
09.             self.host = host
10.
11.         def __call__(self):
12.             rc = subprocess.call(
13.                 'ping -c2 %s &> /dev/null' % self.host,
14.                 shell=True
15.             )
16.             if rc:
17.                 print('%s: down' % self.host)
18.             else:
19.                 print('%s: up' % self.host)
20.
21.     if __name__ == '__main__':
22.         ips = ('172.40.58.%s' %i for i in range(1, 255))  # 创建生成器
23.         for ip in ips:
24.             # 创建线程，Ping是上面定义的函数
25.             t = threading.Thread(target=Ping(ip))  # 创建Ping的实例
26.             t.start()   #执行Ping(ip)
```

步骤二：测试脚本执行

```
01.     [root@localhost day09]# python3 udp_time_serv.py
02.     172.40.58.1: up
03.     172.40.58.69: up
04.     172.40.58.87: up
05.     172.40.58.90: up
06.     172.40.58.102: up
07.     172.40.58.101: up
08.     172.40.58.105: up
09.     172.40.58.106: up
10.     172.40.58.108: up
11.     172.40.58.110: up
12.     172.40.58.109: up
13.     ...
14.     ...
15.     ...
16.     ...
```

Top

```
17.  172.40.58.241: down
18.  172.40.58.242: down
19.  172.40.58.243: down
20.  172.40.58.245: down
21.  172.40.58.246: down
22.  172.40.58.248: down
23.  172.40.58.247: down
24.  172.40.58.250: down
25.  172.40.58.249: down
26.  172.40.58.251: down
27.  172.40.58.252: down
28.  172.40.58.253: down
29.  172.40.58.254: down
```

5 案例5：创建多线程时间戳服务器

5.1 问题

创建mttcp_server.py脚本，编写一个TCP服务器：

1. 服务器监听在0.0.0.0的12345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. 要求能够同时处理多个客户端的请求
5. 要求使用多线程的方式进行编写

5.2 方案

面向对象编程方法编写TCP服务器：

1）TcpTimeServer():创建TcpTimeServer类

2）__init__():创建对象后自动调用init方法，初始化以下属性：

建立socket对象。

设置socket选项，当socket关闭后，本地端用于该socket的端口号立刻就可以被重用。

绑定socket对象IP和端口。

将套接字设为监听模式，准备接收客户端请求。利用listen()函数进行侦听连接。

3）将创建的TcpTimeServer()对象返回给s，让s实例来保存该对象

4）调用s.mainloop()方法：

利用while循环，accept()会等待并返回一个客户端的连接

当有客户机连接上来之后，直接利用 Thread 类来创建工作线程，并让工作线程直接与客户机聊天通信（即调用chat()方法）

此时主线程在创建子线程工作线程后，主线程重新开始循环，进入accept()连接去等待

5)调用chat()方法，用返回的客户端作为参数

循环将recv接收到的数据加上时间戳后send发送给客户端

Top

关闭套接字，释放资源。

6)此时服务器能够同时处理多个客户端的请求，以多线程方式

5.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day09]# vim mttcp_server.py
#! /usr/bin/env python3

import socket
import threading
from time import strftime

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 chat(self, c_sock):
        while True:
            data = c_sock.recv(1024)
            if data.strip() == b'quit':
                break
            data = '[%s] %s' %(strftime('%H:%M:%S'), data.decode('utf8'))
            c_sock.send(data.encode('utf8'))
        c_sock.close()

    def mainloop(self):
        while True:
            cli_sock, cli_addr = self.serv.accept()
            t = threading.Thread(target=self.chat, args=(cli_sock,))
            t.start()

        self.serv.close()

if __name__ == '__main__':
    s = TcpTimeServer()
```

Top

```
35.         s.mainloop()
```

步骤二：测试脚本执行

```
01.     [root@localhost day09]# python3 mttcp_server.py
```

以下两个客户端同时telnet与服务器端连接，可实现多用户通信：

```
01.     [root@localhost day09]# telnet 172.40.58.189 12345
02.     Trying 172.40.58.189...
03.     Connected to 172.40.58.189.
04.     Escape character is '^]'.
05.     nihao
06.     [19:42:58] nihao
07.     I'm fine
08.     [19:43:08] I'm fine
09.     quit
10.     Connection closed by foreign host.
11.     [root@localhost day09]# telnet 172.40.58.189 12345
12.     Trying 172.40.58.189...
13.     Connected to 172.40.58.189.
14.     Escape character is '^]'.
15.     hello world!
16.     [19:43:22] hello world!
17.     hello lilei
18.     [19:43:32] hello lilei
19.     quit
20.     Connection closed by foreign host.
```

Top