[Top](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "page_top_case)

# NSD Devops DAY01

1. [案例1：forking基础应用](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "case1)
2. [案例2：扫描存活主机](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "case2)
3. [案例3：扫描存活主机](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "case3)
4. [案例4：爬取网页](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "case4)
5. [案例5：爬取图片](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "case5)
6. [案例6：处理下载错误](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "case6)
7. [案例7：利用多线程实现ssh并发访问](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/DEVOPS/DAY01/CASE/01/index.html" \l "case7)

## 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

1. [root@localhost day09]# vim myfork.py
2. #!/usr/bin/env python3
3. import os
4. import time
5. from datetime import datetime
6. pid = os.fork()
7. if pid:
8. print("In parent!")
9. time.sleep(10)
10. print("parent exit")
11. else:
12. for i in range(5):
13. print(datetime.now())
14. time.sleep(1)
15. print("child exit")

步骤二：测试脚本执行

1. [root@localhost day09]# python3 myfork.py
2. In parent!
3. 2018-09-03 10:48:46.552528
4. 2018-09-03 10:48:47.553714
5. 2018-09-03 10:48:48.554800
6. 2018-09-03 10:48:49.555901
7. 2018-09-03 10:48:50.557035
8. child exit
9. 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，

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。

1. subprocess.call(
2. 'ping -c2 %s &> /dev/null' % host,
3. shell=True
4. )

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

1. [root@localhost day09]# vim forkping.py
2. #!/usr/bin/env python3
3. import subprocess        #加载支持Linux系统内部命令模块
4. import os
5. #定义函数，允许ping任何主机，ping函数需要给IP作为参数
6. def ping(host):
7. rc = subprocess.call(
8. 'ping -c2 %s &> /dev/null' % host,
9. shell=True
10. )            #定义ping命令的变量，返回值0:正常，返回值1：ping不通
11. if rc:
12. print('%s: down' % host)    #无法ping通打印down
13. else:
14. print('%s: up' % host)        #当re=0，表示可以ping通，打印up
15. if \_\_name\_\_ == '\_\_main\_\_':
16. #生成整个网段的IP列表[172.40.58.1,172.40.58.2....]
17. ips = ['172.40.58.%s' % i for i in range(1, 255)]
18. for ip in ips:
19. pid = os.fork()    # 父进程负责生成子进程
20. if not pid:    # 子进程负责调用ping函数
21. ping(ip)
22. 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

1. [root@localhost day09]# python3 forkping.py
2. [root@localhost day09]# 172.40.58.69: up
3. 172.40.58.1: up
4. 172.40.58.87: up
5. 172.40.58.90: up
6. 172.40.58.102: up
7. 172.40.58.111: up
8. 172.40.58.106: up
9. 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：扫描存活主机

### 3.1 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

### 3.2 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。

### 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

1. [root@localhost day09]# vim mtping.py
2. #!/usr/bin/env python3
3. import subprocess
4. import threading
5. def ping(host):
6. rc = subprocess.call(
7. 'ping -c2 %s &> /dev/null' % host,
8. shell=True
9. )
10. if rc:
11. print('%s: down' % host)
12. else:
13. print('%s: up' % host)
14. if \_\_name\_\_ == '\_\_main\_\_':
15. ips = ['172.40.58.%s' % i for i in range(1, 255)]
16. for ip in ips:
17. # 创建线程，ping是上面定义的函数, args是传给ping函数的参数
18. t = threading.Thread(target=ping, args=(ip,))
19. 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)。

1. [root@localhost day09]# vim mtping2.py
2. #!/usr/bin/env python3
3. import threading
4. import subprocess
5. class Ping:
6. def \_\_init\_\_(self, host):
7. self.host = host
8. def \_\_call\_\_(self):
9. rc = subprocess.call(
10. 'ping -c2 %s &> /dev/null' % self.host,
11. shell=True
12. )
13. if rc:
14. print('%s: down' % self.host)
15. else:
16. print('%s: up' % self.host)
17. if \_\_name\_\_ == '\_\_main\_\_':
18. ips = ('172.40.58.%s' % i for i in range(1, 255)) # 创建生成器
19. for ip in ips:
20. # 创建线程，Ping是上面定义的函数
21. t = threading.Thread(target=Ping(ip)) # 创建Ping的实例
22. t.start() #执行Ping(ip)

步骤二：测试脚本执行

1. [root@localhost day09]# python3 udp\_time\_serv.py
2. 172.40.58.1: up
3. 172.40.58.69: up
4. 172.40.58.87: up
5. 172.40.58.90: up
6. 172.40.58.102: up
7. 172.40.58.101: up
8. 172.40.58.105: up
9. 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. ...
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

## 4 案例4：爬取网页

### 4.1 问题

编写一个get\_web.py脚本，实现以下功能：

1. 爬取的网页为http://www.tedu.cn
2. 保存的文件名为/tmp/tedu.html

### 4.2 方案

导入sys模块，用sys.argv方法获取get\_web函数实参，让用户在命令行上提供http://www.tedu.cn和/tmp/tedu.html两个参数，调用get\_web函数实现如下功能：

1)导入urllib模块，使用urllib模块的urlopen函数打开url（即网址），赋值给html

2)以写方式打开/tmp/tedu.html文件

3)以循环方式：

读html获取的数据，保存到data

将data写入/tmp/tedu.html

4)关闭html

### 4.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

1. [root@localhost day11]# vim get\_web.py
2. #!/usr/bin/env python3
3. import sys
4. from urllib.request import urlopen
5. def get\_web(url, fname):
6. html = urlopen(url)    #使用urllib模块的urlopen函数打开url，赋值给html
7. with open(fname, 'wb') as fobj:
8. while True:
9. data = html.read(4096)
10. if not data:
11. break
12. fobj.write(data)
13. html.close()
14. if \_\_name\_\_ == '\_\_main\_\_':
15. get\_web(sys.argv[1], sys.argv[2])        #让用户在命令行上提供网址和下载数据保存位置

步骤二：测试脚本执行

1. [root@localhost day11]# python3 get\_web.py http://www.tedu.cn /tmp/tedu.html
2. [root@localhost day11]# cat /tmp/tedu.html
3. 执行cat命令可以看到/tmp/tedu.html文件中爬取到的内容

## 5 案例5：爬取图片

### 5.1 问题

1. 将http://www.tedu.cn所有的图片下载到本地
2. 本地的目录为/tmp/images
3. 图片名与网站上图片名保持一致

### 5.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

1)爬取网页内容放入指定fname（即/tmp/tedu.html）文件中

创建get\_web.py文件，编写代码如下：

1. [root@localhost day11]# vim get­\_web.py
2. #!/usr/bin/env python3
3. import sys
4. from urllib.request import urlopen        #导入urllib
5. def get\_web(url, fname):    #url为爬取目标网址（www.tedu.cn）
6. #fname为爬取内容存储文件名
7. html = urlopen(url)    #使用urllib模块的urlopen函数打开url，赋值给html
8. with open(fname, 'wb') as fobj:    #以写方式打开文件
9. while True:
10. data = html.read(4096)         #读html获取的数据，保存到data
11. if not data:
12. break
13. fobj.write(data)        # 将data写入文件中
14. html.close()

2)利用正则匹配，将爬取到的fname文件内容中所有图片网址放入result列表

创建get\_url.py文件，编写代码如下：

1. [root@localhost day11]# vim get­\_url.py
2. #!/usr/bin/env python3
3. import sys
4. import re
5. def get\_url(patt, fname):        #patt可匹配图片网址正则表达式，
6. #fname为1）中爬取到内容的文件
7. cpatt = re.compile(patt)    #将正则表达式字符串形式编译为cpatt实例
8. result = []                #存放匹配正则表达式的图片网址
9. with open(fname) as fobj:        #打开爬取到网站（www.tedu.cn）内容的文件
10. for line in fobj:            #遍历fname文件
11. m = cpatt.search(line)        #使用cpatt实例查找匹配规则的网址
12. if m:
13. result.append(m.group())        #将匹配到的图片网址最加到result列表
14. return result                            #函数最终返回result列表
15. if \_\_name\_\_ == '\_\_main\_\_':
16. url = r'http://[.\w/-]+\.(jpg|png|jpeg|gif)'    #符合图片网址规则的正则表达式
17. print(get\_url(url, sys.argv[1]))

3)遍历图片列表result，将图片网址对应内容爬取下来存入指定文件

创建download.py文件，编写代码如下：

1. [root@localhost day11]# vim download.py
2. #!/usr/bin/env python3
3. import os
4. from get\_url import get\_url    #导入get\_url函数
5. from get\_web import get\_web    #导入get\_web函数
6. #调用get\_web函数爬取/http://www.tedu.cn网站内容，存入/tmp/tedu.html文件中
7. get\_web('http://www.tedu.cn/', '/tmp/tedu.html')
8. #符合图片网址正则表达式
9. img\_url = r'http://[.\w/-]+\.(jpg|png|jpeg|gif)'
10. #调用get\_url函数，从/tmp/tedu.html文件中获取符合匹配规则的图片网址，
11. #存入result列表中，将列表结果赋值给urls变量
12. urls = get\_url(img\_url, '/tmp/tedu.html')
13. #爬取到的图片存储目录
14. img\_dir = '/tmp/images'
15. #判断目录是否存在，如果不存在则创建该目录
16. if not os.path.exists(img\_dir):
17. os.mkdir(img\_dir)
18. #遍历图片网址列表，每次循环遍历出一个图片网址
19. for url in urls:
20. # url.split('/')[-1]：将网址切片，取最后一个字符命名图片
21. #将图片存储目录与图片名拼接，举例：fname=/tmp/images/XXX.png
22. fname = os.path.join(img\_dir, url.split('/')[-1])
23. #调用get\_web函数，爬取图片网址内容，存入fname文件中
24. get\_web(url, fname)

步骤二：测试脚本执行

1. [root@localhost day11]# python3 download.py
2. [root@localhost day11]# nautilus /tmp/images
3. 执行以上命令即可看到爬取的图片，且图片命名与网站上图片命名一致

## 6 案例6：处理下载错误

### 6.1 问题

1. 起动一个web服务
2. 在web服务器的文档目录下创建目录ban，权限设置为700
3. 编写python程序访问不存在的路径和ban目录，处理404和403错误
4. 404错误打印“无此页面”，403错误打印“无权访问”

### 6.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：启动一个web服务

1. [root@localhost ~]# systemctl restart httpd

步骤二：在web服务器的文档目录下创建目录ban，权限设置为700

1. [root@localhost ~]# mkdir -m 700 /var/www/html/ban

步骤三：如果访问的页面不存在或拒绝访问，程序将抛出异常

执行案例2中get\_web.py文件，访问不存在页面，抛出404异常如下：

1. [root@localhost day11]# python3 get\_web.py http://127.0.0.1/abc/ /tmp/abc.html
2. Traceback (most recent call last):
3. ...
4. ...
5. raise HTTPError(req.full\_url, code, msg, hdrs, fp)
6. urllib.error.HTTPError: HTTP Error 404: Not Found

执行案例2中get\_web.py文件，访问存在页面ban目录，抛出403权限异常如下：

1. [root@localhost day11]# python3 get\_web.py http://127.0.0.1/ban/ /tmp/abc.html
2. Traceback (most recent call last):
3. ...
4. ...
5. raise HTTPError(req.full\_url, code, msg, hdrs, fp)
6. urllib.error.HTTPError: HTTP Error 403: Forbidden

步骤三：编写python程序捕获异常

创建get\_web3.py文件，实现访问不存在的路径和ban目录时，捕获404和403错误

，同时404错误打印“无此页面”，403错误打印“无权访问”，代码如下：

1. import sys
2. from urllib.request import urlopen
3. from urllib.error import HTTPError    #导入urllib.error模块，用HTTPError捕获异常信息
4. def get\_web(url, fname):
5. try:
6. html = urlopen(url)    #打开网址时即可知道是否有异常，所以将本语句放入try语句
7. except HTTPError as e:     #捕获返回HTTPError类的实例e
8. print(e)
9. if e.code == 403:        #捕获异常状态码如果等于403
10. print('权限不足')    #输出'权限不足'
11. elif e.code == 404:    #捕获异常状态码如果等于404
12. print('没有那个地址')    #输出'没有那个地址'
13. return                        #return后面代码均不执行
15. with open(fname, 'wb') as fobj:
16. while True:
17. data = html.read(4096)
18. if not data:
19. break
20. fobj.write(data)
21. html.close()
22. if \_\_name\_\_ == '\_\_main\_\_':
23. get\_web(sys.argv[1], sys.argv[2])

测试脚本执行：

访问不存在页面：

1. [root@localhost day11]# python3 get\_web.py http://127.0.0.1/abc/ /tmp/abc.html
2. HTTP Error 404： Not Found
3. 没有那个地址

访问ban目录：

1. [root@localhost day11]# python3 get\_web.py http://127.0.0.1/ban/ /tmp/abc.html
2. HTTP Error 403： Forbidden
3. 权限不足

## 7 案例7：利用多线程实现ssh并发访问

### 7.1 问题

编写一个remote\_comm.py脚本，实现以下功能：

1. 在文件中取出所有远程主机IP地址
2. 在shell命令行中接受远程服务器IP地址文件、远程服务器密码以及在远程主机上执行的命令
3. 通过多线程实现在所有的远程服务器上并发执行命令

### 7.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：安装paramiko

paramiko 遵循SSH2协议，支持以加密和认证的方式，进行远程服务器的连接，可以实现远程文件的上传，下载或通过ssh远程执行命令。

1. [root@localhost ~]# pip3 install paramiko
2. ...
3. ...
4. Successfully installed bcrypt-3.1.4 paramiko-2.4.1 pyasn1-0.4.4 pynacl-1.2.1
5. You are using pip version 9.0.1, however version 18.0 is available.
6. You should consider upgrading via the 'pip install --upgrade pip' command.

测试是否安装成功

1. >>> import paramiko
2. >>>

步骤二：编写脚本

1. [root@localhost day11]# vim remote\_comm.py
2. #!/usr/bin/env python3
3. import sys
4. import getpass
5. import paramiko
6. import threading
7. import os
8. #创建函数实现远程连接主机、服务器密码以及在远程主机上执行的命令的功能
9. def remote\_comm(host, pwd, command):
10. #创建用于连接ssh服务器的实例
11. ssh = paramiko.SSHClient()
12. #设置自动添加主机密钥
13. ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())
14. #连接ssh服务器，添加连接的主机、用户名、密码填好
15. ssh.connect(hostname=host, username='root', password=pwd)
16. #在ssh服务器上执行指定命令，返回3项类文件对象，分别是，输入、输出、错误
17. stdin, stdout, stderr = ssh.exec\_command(command)
18. #读取输出
19. out = stdout.read()
20. #读取错误
21. error = stderr.read()
22. #如果有输出
23. if out:
24. #打印主机输出内容
25. print('[%s] OUT:\n%s' % (host, out.decode('utf8')))
26. #如果有错误
27. if error:
28. #打印主机错误信息
29. print('[%s] ERROR:\n%s' % (host, error.decode('utf8')))
30. #程序结束
31. ssh.close()
32. if \_\_name\_\_ == '\_\_main\_\_':
33. #设定sys.argv长度，确保remote\_comm函数中参数数量
34. if len(sys.argv) != 3:
35. print('Usage: %s ipaddr\_file "command"' % sys.argv[0])
36. exit(1)
37. #判断命令行上输入如果不是文件，确保输入的是文件
38. if not os.path.isfile(sys.argv[1]):
39. print('No such file:', sys.argv[1])
40. exit(2)
41. #fname为存储远程主机ip的文件，用sys.argv方法，可以在执行脚本时再输入文件名，更为灵活
42. fname = sys.argv[1]
43. #command为在远程主机上执行的命令，用sys.argv方法，可以在执行脚本时再输入相应命令，command为remote\_comm函数第三个参数
44. command = sys.argv[2]
45. #通过getpass输入远程服务器密码，pwd为remote\_comm函数第二个参数
46. pwd = getpass.getpass()
47. #打开存有远程主机ip的文件
48. with open(fname) as fobj:
49. #将遍历文件将ip以列表形式存入ips，line.strip()可以去掉每行ip后\n
50. ips = [line.strip() for line in fobj]
51. #循环遍历列表，获取ip地址，ip为remote\_comm函数第一个参数
52. for ip in ips:
53. #将读取到的ip地址作为remote\_comm函数实际参数传递给函数，ips中有几个ip地址循环几次
54. #创建多线程
55. t = threading.Thread(target=remote\_comm, args=(ip, pwd, command))
56. #启用多线程
57. t.start()

步骤三：测试脚本执行

1. #参数给少了效果如下：
2. [root@localhost day11]# python3 remote\_comm.py server\_addr.txt
3. Usage: remote\_comm.py ipaddr\_file “command”
4. #参数给多了效果如下：
5. [root@localhost day11]# python3 remote\_comm.py server\_addr.txt id zhangsan
6. Usage: remote\_comm.py ipaddr\_file “command”
7. #正常显示如下：
8. [root@localhost day11]# python3 remote\_comm.py server\_addr.txt “id zhangsan”
9. Password:
10. [192.168.4.2] OUT:
11. uid=1001(zhangsan) gid=1001(zhangsan) 组=1001(zhangsan)
12. [192.168.4.3] OUT:
13. uid=1001(zhangsan) gid=1001(zhangsan) 组=1001(zhangsan)
14. [root@localhost day11]# python3 remote\_comm.py server\_addr.txt “echo redhat | passwd –stdin root”
15. Password:
16. [192.168.4.3] OUT:
17. 更改用户root的密码：
18. passwd：所有的身份验证令牌已经成功更新。
19. [192.168.4.2] OUT:
20. 更改用户root的密码：
21. passwd：所有的身份验证令牌已经成功更新。
22. #此时密码已经变成redhat
23. [root@localhost day11]# python3 remote\_comm.py server\_addr.txt “id zhangsan”
24. Password:
25. [192.168.4.2] OUT:
26. uid=1001(zhangsan) gid=1001(zhangsan) 组=1001(zhangsan)
27. [192.168.4.3] OUT:
28. uid=1001(zhangsan) gid=1001(zhangsan) 组=1001(zhangsan)