Tedu.cn
达内教育

运维开发实战

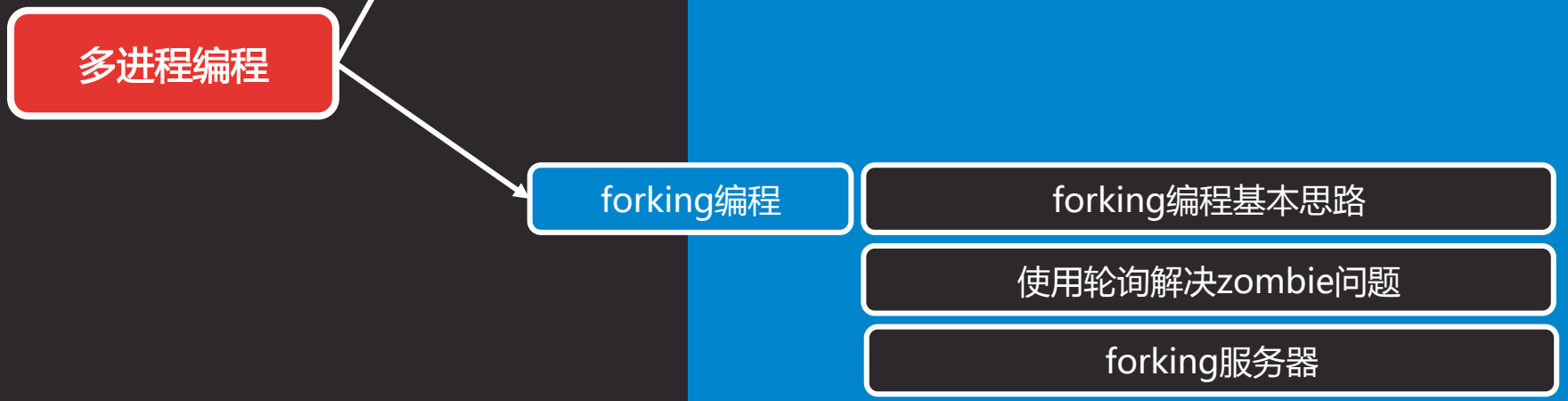
NSD DEVOPS

DAY01

内容

上午	09:00 ~ 09:30	多进程编程
	09:30 ~ 10:20	
	10:30 ~ 11:20	
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	多线程编程
	15:00 ~ 15:50	
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑

多进程编程


多进程编程
forking工作原理
什么是forking
进程的生命周期
僵尸进程
forking编程
forking编程基本思路
使用轮询解决zombie问题
forking服务器

forking工作原理

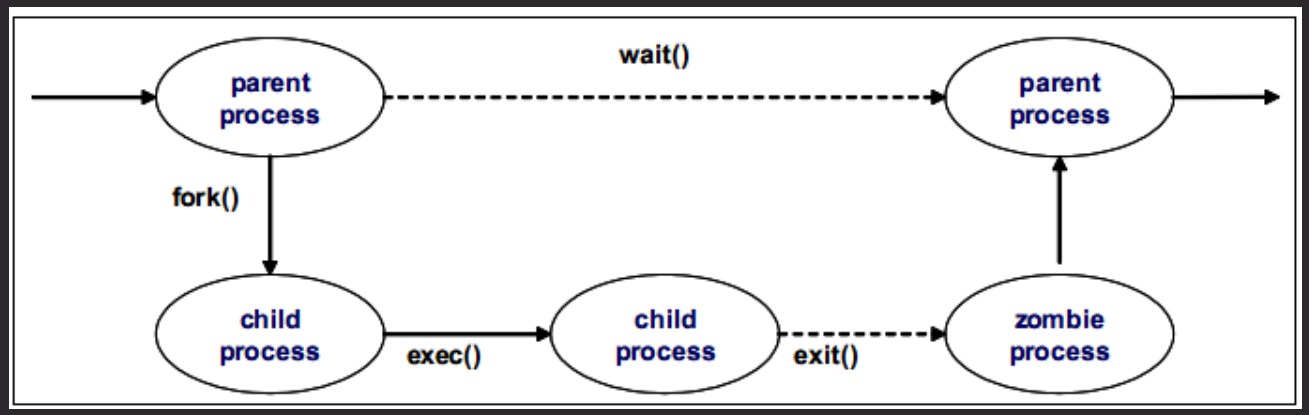
什么是forking

- 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

知识讲解

进程的生命周期

- 父进程fork出子进程并挂起
- 子进程运行完毕后，释放大部分资源并通知父进程，这个时候，子进程被称作僵尸进程
- 父进程获知子进程结束，子进程所有资源释放



知识讲解

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

知识讲解

forking编程

forking编程基本思路

- 需要使用os模块
- os.fork()函数实现forking功能
- python中，绝大多数的函数只返回一次，os.fork将返回两次
- 对fork()的调用，针对父进程返回子进程的PID；对于子进程，返回PID0

知识讲解

forking编程基本思路（续1）

- 因为所有的父子进程拥有相同的资源，所以在编写程序时要避免资源冲突

知识讲解

网络编程思路如下：

```
pid = os.fork()                 #实现forking
if pid:                         #在父进程中关闭子进程连接
    close_child_conn            #接着处理其他的连接请求
    handle_more_conn
else:                           #子进程关闭父进程连接，响应当
    close_parent_conn           #前的用户连接
    process_this_conn
```

案例1：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”

课堂练习

案例2：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

课堂练习

使用轮询解决zombie问题

- 父进程通过os.wait()来得到子进程是否终止的信息
- 在子进程终止和父进程调用wait()之间的这段时间，子进程被称为zombie（僵尸）进程
- 如果子进程还没有终止，父进程先退出了，那么子进程会持续工作。系统自动将子进程的父进程设置为init进程，init将来负责清理僵尸进程

知识讲解

使用轮询解决zombie问题（续1）

- python可以使用waitpid()来处理子进程
- waitpid()接受两个参数，第一个参数设置为-1，表示与wait()函数相同；第二参数如果设置为0表示挂起父进程，直到子程序退出，设置为1表示不挂起父进程
- waitpid()的返回值：如果子进程尚未结束则返回0，否则返回子进程的PID

知识讲解

使用轮询解决zombie问题（续2）

知识讲解

```
import os,  time
def reap():
    result = os.waitpid(-1, os.WNOHANG)     #WNOHANG即值为1
    print('Reaped child process %d' % result[0])
pid = os.fork()
if pid:
    print 'In parent. Sleeping 15s...'
    time.sleep(15)
    reap()
    time.sleep(5)
    print('parent done')
else:
    print 'In child. Sleeping 5s...'
    time.sleep(5)
    print('Child terminating.')
```

forking服务器

- 在网络服务器中，forking被广泛使用
- 如果服务器需要同时响应多个客户端，那么forking是解决问题最常用的方法之一
- 父进程负责接受客户端的连接请求
- 子进程负责处理客户端的请求

知识讲解

案例3：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

课堂练习

多线程编程


多线程编程
多线程工作原理
多线程的动机
多线程任务的工作特点
什么是进程
什么是线程
多线程编程
多线程相关模块
传递函数给Thread类
传递可调用类给Thread类
含有线程的服务器

Tedu.cn
达内教育

多线程工作原理

多线程的动机

知识讲解

- 在多线程（MT）编程出现之前，电脑程序的运行由一个执行序列组成，执行序列按顺序在主机的中央处理器（CPU）中运行
- 无论是任务本身要求顺序执行还是整个程序是由多个子任务组成，程序都是按这种方式执行的
- 即使子任务相互独立，互相无关（即，一个子任务的结果不影响其它子任务的结果）时也是这样
- 如果并行运行这些相互独立的子任务可以大幅度地提升整个任务的效率

多线程任务的工作特点

- 它们本质上就是异步的，需要有多个并发事务
- 各个事务的运行顺序可以是不确定的，随机的，不可预测的
- 这样的编程任务可以被分成多个执行流，每个流都有一个要完成的目标
- 根据应用的不同，这些子任务可能都要计算出一个中间结果，用于合并得到最后的结果

知识讲解

什么是进程

- 计算机程序只不过是磁盘中可执行的、二进制（或其它类型）的数据
- 进程（有时被称为重量级进程）是程序的一次执行
- 每个进程都有自己的地址空间、内存以及其它记录其运行轨迹的辅助数据
- 操作系统管理在其上运行的所有进程，并为这些进程公平地分配时间

知识讲解

什么是线程

知识讲解

- 线程（有时被称为轻量级进程）跟进程有些相似。不同的是，所有的线程运行在同一个进程中，共享相同的运行环境
- 线程有开始，顺序执行和结束三部分
- 线程的运行可能被抢占（中断），或暂时的被挂起（也叫睡眠），让其它的线程运行，这叫做让步
- 一个进程中的各个线程之间共享同一片数据空间，所以线程之间可以比进程之间更方便地共享数据以及相互通讯

什么是线程（续1）

- 线程一般都是并发执行的，正是由于这种并行和数据共享的机制使得多个任务的合作变为可能
- 需要注意的是，在单CPU 的系统中，真正的并发是不可能的，每个线程会被安排成每次只运行一小会，然后就把CPU 让出来，让其它的线程去运行

知识讲解

Tedu.cn
达内教育

多线程编程

多线程相关模块

- thread和threading模块允许程序员创建和管理线程
- thread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
- 只建议那些有经验的专家在想访问线程的底层结构的时候，才使用thread模块

知识讲解

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

```
import threading
import time
nums = [4, 2]

def loop(nloop, nsec):    #定义函数，打印运行的起止时间
    print('start loop %d, at %s' % (nloop, time.ctime()))
    time.sleep(nsec)
    print('loop %d done at %s' % (nloop, time.ctime()))
```

知识讲解

传递函数给Thread类（续1）

知识讲解

```
def main():
    print('starting at: %s' % time.ctime())
    threads = []
    for i in range(2):    #创建两个线程，放入列表
        t = threading.Thread(target = loop, args = (0, nums[i]))
        threads.append(t)

for i in range(2):
        threads[i].start()    #同时运行两个线程
    for i in range(2):
        threads[i].join()     #主程序挂起，直到所有线程结束
    print('all Done at %s' % time.ctime())

if __name__ == '__main__':
    main()
```

传递可调用类给Thread类

- 传递可调用类给Thread类是介绍的第二种方法
- 相对于一个或几个函数来说，由于类对象里可以使用类的强大的功能，可以保存更多的信息，这种方法更为灵活

知识讲解

传递可调用类给Thread类（续1）

知识讲解

```
#!/usr/bin/env python
import threading
import time
nums = [4, 2]
class ThreadFunc(object):       #定义可调用的类
    def __init__(self, func, args, name = ''):
        self.name = name
        self.func = func
        self.args = args
    def __call__(self):
        apply(self.func, self.args)

def loop(nloop, nsec):  #定义函数，打印运行的起止时间
    print('start loop %d, at %s' % (nloop, time.ctime()))
    time.sleep(nsec)
    print('loop %d done at %s' % (nloop, time.ctime()))
```

传递可调用类给Thread类（续2）

知识讲解

```
def main():
    print('starting at: %s' % time.ctime())
    threads = []
    for i in range(2):
        t = threading.Thread(target = ThreadFunc(loop, (i, nums[i]),
loop.__name__))
        threads.append(t)          #创建两个线程，放入列表
    for i in range(2):
        threads[i].start()
    for i in range(2):
        threads[i].join()
    print('all Done at %s' % time.ctime())

if __name__ == '__main__':
    main()
```

含有线程的服务器

- 多数的线程服务器有同样的结构
- 主线程是负责侦听请求的线程
- 主线程收到一个请求的时候，新的工作线程会被建立起来，处理客户端请求
- 客户端断开时，工作线程将终止
- 线程划分为用户线程和后台（daemon）进程，setDaemon将线程设置为后台进程

知识讲解

案例4：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

课堂练习

案例5：创建多线程时间戳服务器

Tedu.cn 达内教育

- 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

课堂练习

Tedu.cn
达内教育

总结和答疑