**Linux C++通讯架构实战\_卷1—学习笔记**

视频课程地址：

<https://study.163.com/course/introduction/1006470001.htm?share=1&shareId=1396930938>

目录

[1. 环境搭建 3](#_Toc4462)

[1.1. 安装Ubuntu虚拟机 3](#_Toc24835)

[1.2. 配置Ubuntu的固定IP地址 3](#_Toc20903)

[1.3. 配置远程连接 4](#_Toc13697)

[1.4. 安装gcc,g++等 5](#_Toc11481)

[2. 进入nginX之门 5](#_Toc20877)

[2.1. 为什么选择nginX 5](#_Toc10471)

[2.2. 安装nginX，搭建web服务器 5](#_Toc24934)

[2.2.1. 安装前提 5](#_Toc3959)

[2.2.2. nginX文件目录 6](#_Toc1289)

[2.2.3. nginx的编译和安装 7](#_Toc20920)

[2.3. nginx的启动和简单实用 7](#_Toc7530)

[3. nginx整体结构、进程模型 8](#_Toc14151)

[3.1. nginx的整体结构 8](#_Toc31812)

[3.1.1. master进程和worker进程概览（父子关系） 8](#_Toc6462)

[3.1.2. nginx进程模型 8](#_Toc21359)

[3.1.3. 调整worker进程数量 9](#_Toc12976)

[3.2. nginx进程模型细说 9](#_Toc26593)

[4. 终端和进程的关系 9](#_Toc6691)

[4.1. 终端与bash进程 9](#_Toc28103)

[4.2. 终端上的开启进程 10](#_Toc21453)

[4.3. 进程关系进一步分析 10](#_Toc24153)

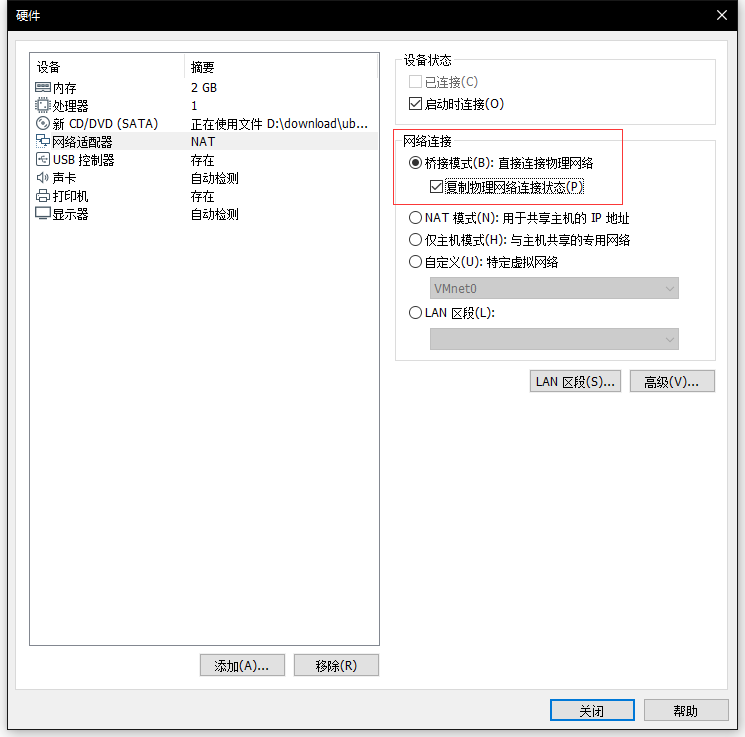
[4.4. strace工具的使用 11](#_Toc10893)

[4.5. 终端关闭时如何让进程不退出 12](#_Toc23893)

# 环境搭建

## 安装Ubuntu虚拟机

注意事项：网络适配器选择“桥接”，因为服务端需要固定的IP地址。



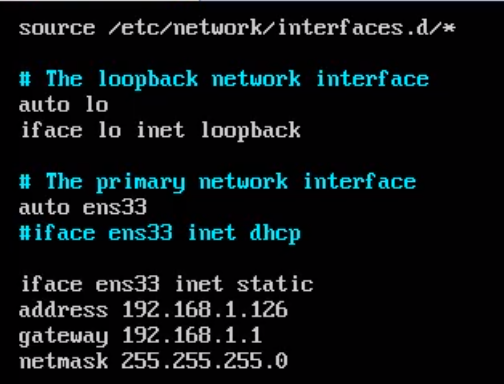
## 配置Ubuntu的固定IP地址

服务器端需要固定的IP地址方便客户端访问。

1. 要修改配置文件，需要vim编辑器，Ubuntu下安装vim的命令：

sudo apt-get install vim-gtk

1. Ubuntu下查看网络信息命令：ifconfig
2. 修改 /etc/network/interfaces 文件：sudo vim interfaces



1. 修改DNS

命令：sudo vim /etc/resolvconf/resolv.conf.d/base



1. 然后重启Ubuntu：sudo reboot

## 配置远程连接

1. 需要在linux上安装ssh服务:

检查有没有ssh服务：ps -e|grep ssh (没有输出代表没有安装)

安装ssh：sudo apt-get install openssh-server

1. 安装Xshell

## 安装gcc,g++等

sudo apt-get install build-essential

sudo apt-get install gcc

sudo apt-get install g++

# 进入nginX之门

## 为什么选择nginX

linux epoll技术； windows IOCP

单机10万并发；

epoll这种高并发技术好处就是：高并发只是占用更多内存就能做到。

## 安装nginX，搭建web服务器

### 安装前提

1. epoll, linux 内核版本为2.6或者以上；

命令：cat /proc/version

1. gcc，g++；
2. pcre库：函数库；支持解析正则表达式；

命令：sudo apt-get install libpcre3-dev

1. zlib库：压缩解压缩功能

命令：sudo apt-get install libz-dev

1. openssl库：ssl功能相关库。用于网站加密通讯

命令：sudo atp-get install libssl-dev

### nginX文件目录

解压nginx压缩包：tar -zxvf nginx-1.16.1.tar.gz

auto/:编译相关的脚本，可执行文件configure一会会用到这些脚本

cc/:检查编译器的脚本

lib/:检查依赖库的脚本

os/:检查操作系统类型的脚本

type/:检查平台类型的脚本

CHANGES:修复的bug，新增加的功能说明

CHANGES.ru:俄语版的CHANGES

conf/:默认的配置文件

configure:编译nginx之前必须先执行本脚本以生成一些必要的中间文件

contrib/:脚本和工具，典型的是vim高亮工具

vim/:vim高亮工具

html/:欢迎界面和错误界面相关的html文件

man/:nginx帮助文件目录

src/:nginx源码目录

core:核心代码

event:event(事件)模块相关代码

http:http(web服务)模块相关代码

mail:邮件模块相关代码

os:操作系统相关代码

stream:流处理相关代码

### nginx的编译和安装

1. 编译的第一步：用configure来进行编译之前的配置工作 ./configure

--prefix:指定最终安装到的目录：默认值 /usr/local/nginx

--sbin-path:用来指定可执行文件目录：默认值 sbin/ nginx

--conf-path:用来指定配置文件目录：默认值 conf/nginx.conf

执行configura文件后，会生成一些文件：

objs/:执行了configure生成的中间文件目录

ngx\_modules.c:内容决定了一会编译nginx的时候有哪些模块会被编译到nginx里边来

Makefile:执行了configure脚本产生的编译规则文件，执行make命令 时用到

1. 用make来编译，生成了可执行文件 make
2. 用make命令开始安装 sudo make install

## nginx的启动和简单实用

启动： sudo ./nginx

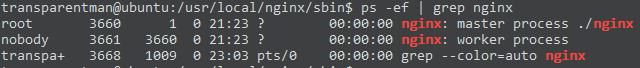
# nginx整体结构、进程模型

## nginx的整体结构

### master进程和worker进程概览（父子关系）

启动nginx，看到了一个master进程，一个worker进程

ps -ef命令：



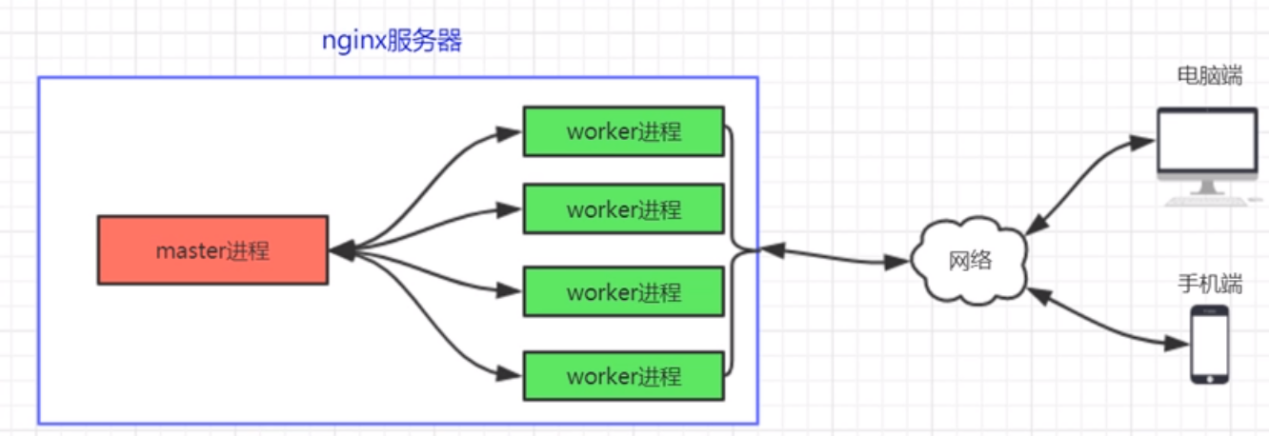
第一列：UID, 进程所属的用户id

第二列：进程ID（PID），用来唯一的标识一个进程

第三列：父进程ID（PPID）。fork(), worker进程是被master进程通过fork()创建出来的——worker进程是master进程的子进程

### nginx进程模型

1个master进程，1到多个worker进程，这种工作机制来对外服务的。这种工作机制保证了nginx能够稳定、灵活的运行。



1. master进程的责任：监控进程，不处理具体业务，专门用来管理和监控worker进程；master的角色是监工，比较清闲。
2. worker进程：用来干主要的活的，（和用户交互）。
3. master进程和worker进程之间要通讯，可以用 信号 ，也可以用 共享内存 。
4. 稳定性、灵活性的体现之一：worker进程一旦挂掉，那么master进程会立即fork()一个新的worker进程投入工作中去。

### 调整worker进程数量

worker进程几个合适呢？公认的做法：多核计算机，就让每个worker运行在一个单独的内核上，最大限度减少CPU进程切换成本，提高系统运行效率。

## nginx进程模型细说

nginx重载配置文件：sudo ./nginx -s reload

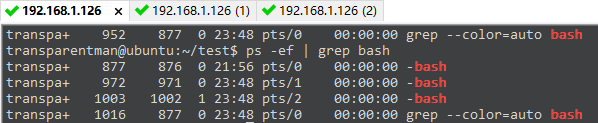
nginx能够热升级、能够热回滚

nginx的关闭：sudo ./nginx -s quit

# 终端和进程的关系

## 终端与bash进程

输入命令：ps -ef | grep bash



其中，pts是虚拟终端，每连接一个虚拟终端到linux操作系统，就会出现一个bash进程（shell[壳]），就是这个黑窗口，用来解释用户输入的命令。

bash = shell = 命令行解释器

## 终端上的开启进程

用命令查看：ps -la

随着终端的退出，这个终端上开启的正在运行的进程也退出了。

开启的可执行程序 是 bash的子进程。

## 进程关系进一步分析

查看命令：ps -eo pid,ppid,sid,tty,pgrp,comm,cmd | grep -E ‘bash|PID|test’

每个进程还属于一个进程组：一个或者多个进程的集合，每个进程组有一个唯一的进程组ID，可以调用系统函数来创建进程组、加入进程组。

“会话”（session）：是一个或者多个进程组的集合。

一般，只要不进行特殊的函数调用，一个bash（shell）上边运行的所有程序都属于一个会话，而这个会话有一个session leader；这个bash（shell）通常就是session leader；你也可以调用系统函数创建新的session。

1. 如果虚拟终端要断开的话，操作系统就会发送SIGHUP信号（终端断开信号），给session leader，一般来说就是这个bash进程。
2. bash进程受到SIGHUP信号后，bash会把这个信号发送给session里边的所有进程，收到这个SIGHUP信号的进程的缺省动作就是退出。

## strace工具的使用

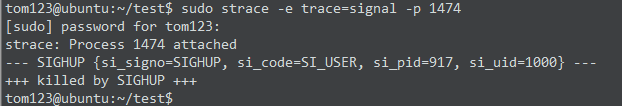
linux下调试分析诊断工具：可以跟踪程序执行时进程的系统调用以及所收到的信号；

跟踪test进程：sudo strace -e trace=signal -p 1474

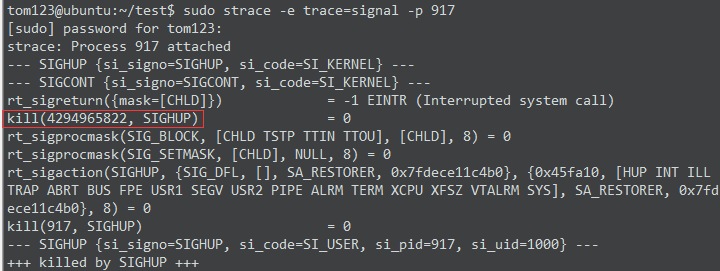
跟踪开启test的bash进程：sudo strace -e trace=signal -p 917

关闭终端后：

跟踪test输出：



跟踪bash输出：



kill(4294965822, SIGHUP) ：发送信号SIGHUP给这个-1474的绝对值所在的进程组；所以test进程就收到了SIGHUP信号。

综合来讲，这个bash先发送SIGHUP给同一个session里边的所有进程，然后再发送SIGHUP给自己。

## 终端关闭时如何让进程不退出

设想：

1. test进程拦截（test进程收到这个信号并告诉操作系统，我不想死，不要把我杀掉）SIGHUP信号，是不是可以？
2. test进程和bash进程不在同一个session里，是不是可以？

验证：

#include <stdio.h>

#include <unistd.h>

#include <signal.h>

int *main*()

{

*printf*("test进程开始执行了\n");

// 系统函数， 设置某个信号来的时候处理程序（用哪个函数处理）

*signal*(SIGHUP ,*SIG\_IGN*); // SIG\_IGN标志：我要求忽略这个信号，请操作系统不要用缺省的处理方式来对待我

while (1) {

*printf*("休息3秒\n");

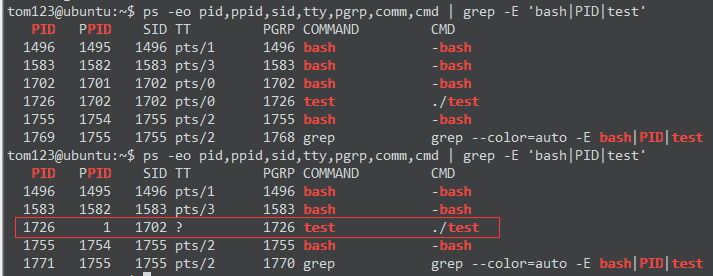
sleep(3);

}

return 0;

}

关闭终端后：



观察到开启test的bash进程1702被杀死了，但是test进程还在，而且父id变成了1，1就是Init这个老祖宗；因为本身的父进程被杀死了，test进程没被杀死，所以test进程就变成了孤儿进程，然后被Init收养了。

经过验证，设想A成立！

#include <stdio.h>

#include <unistd.h>

int main()

{

printf("test进程开始执行了\n");

pid\_t pid;

pid = fork(); // 系统函数，用来创建新进程。子进程会从fork()调用之后开始执行

if (pid < 0) {

printf("fork()进程出错！\n");

}

else if (pid == 0) {

// 子进程pid为0

printf("子进程开始执行！\n");

setsid(); // 设置新的sessionID

while (1) {

sleep(3);

printf("子进程休息3秒\n");

}

return 0;

}

else {

// 父进程会走到这里

while (1) {

sleep(3);

printf("父进程休息3秒\n");

}

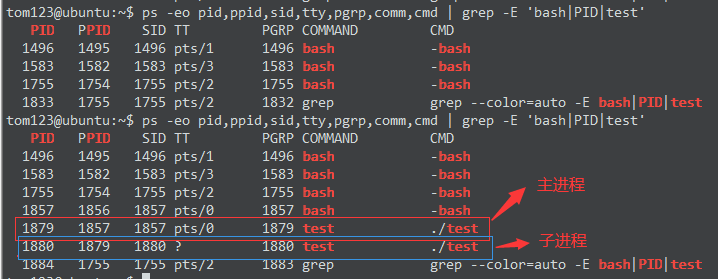
return 0;

}

return 0;

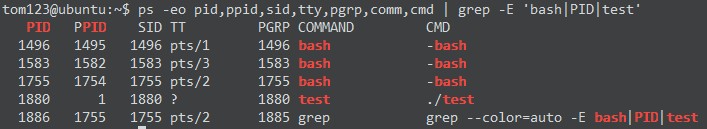
}

关闭终端前：



主进程和子进程的sessionID不一样！

关闭终端后：



主进程被杀死，子进程还在运行，因为sessionID不一样。

经过验证，设想B成立！

还有其他方法可以使终端退出时进程不退出：

1. setsid命令：启动一个进程，而且能够使启动的进程在一个新的session中，这样的话，终端关闭时该进程就不会退出。

命令：setsid ./test

1. nohup(no hang up不要挂断)，用该命令启动的进程跟上边忽略掉SIGHUP信号的道理相同，该命令会把输出内容存到当前目录下的nohup.out文件中。

命令：nohup ./test