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](#_Toc34492598)

[1.1. 安装Ubuntu虚拟机 3](#_Toc34492599)

[1.2. 配置Ubuntu的固定IP地址 3](#_Toc34492600)

[1.3. 配置远程连接 4](#_Toc34492601)

[1.4. 安装gcc,g++等 5](#_Toc34492602)

[2. 进入nginX之门 6](#_Toc34492603)

[2.1. 为什么选择nginX 6](#_Toc34492604)

[2.2. 安装nginX，搭建web服务器 6](#_Toc34492605)

[2.2.1. 安装前提 6](#_Toc34492606)

[2.2.2. nginX文件目录 6](#_Toc34492607)

[2.2.3. nginx的编译和安装 7](#_Toc34492608)

[2.3. nginx的启动和简单实用 8](#_Toc34492609)

[3. nginx整体结构、进程模型 9](#_Toc34492610)

[3.1. nginx的整体结构 9](#_Toc34492611)

[3.1.1. master进程和worker进程概览（父子关系） 9](#_Toc34492612)

[3.1.2. nginx进程模型 9](#_Toc34492613)

[3.1.3. 调整worker进程数量 10](#_Toc34492614)

[3.2. nginx进程模型细说 10](#_Toc34492615)

[4. 终端和进程的关系 11](#_Toc34492616)

[4.1. 终端与bash进程 11](#_Toc34492617)

[4.2. 终端上的开启进程 11](#_Toc34492618)

[4.3. 进程关系进一步分析 11](#_Toc34492619)

[4.4. strace工具的使用 12](#_Toc34492620)

[4.5. 终端关闭时如何让进程不退出 13](#_Toc34492621)

[4.6. 后台运行 & 16](#_Toc34492622)

[5. 信号的概念、认识、处理动作 17](#_Toc34492623)

[5.1. 信号的基本概念 17](#_Toc34492624)

[5.2. 通过kill命令认识一些信号 17](#_Toc34492625)

[5.3. 进程的状态 18](#_Toc34492626)

[5.4. 常用信号列举： 19](#_Toc34492627)

[5.5. 信号处理的相关动作 20](#_Toc34492628)

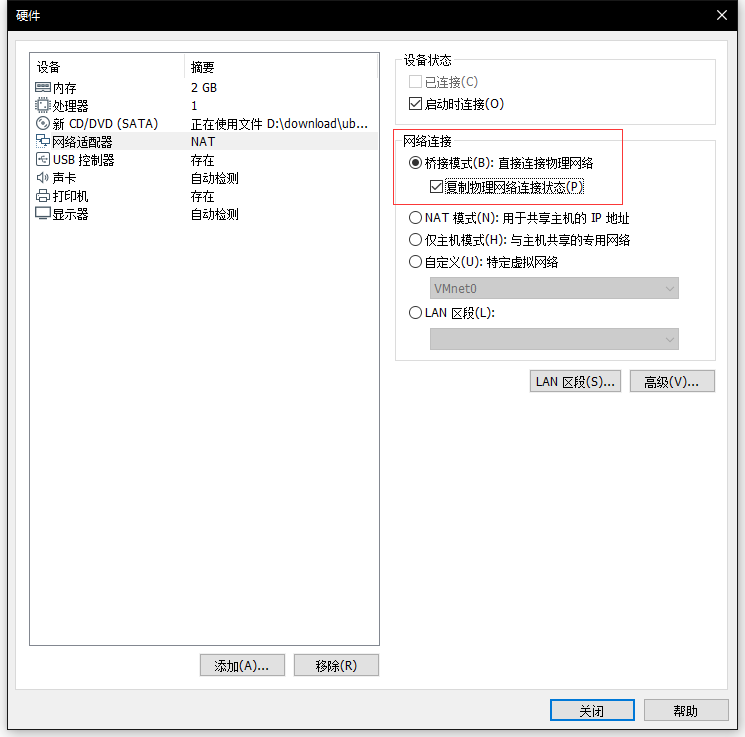
[6. Unix/Linux体系结构 、信号编程初步 21](#_Toc34492629)

[6.1. Unix/Linux操作系统体系结构 21](#_Toc34492630)

# 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。



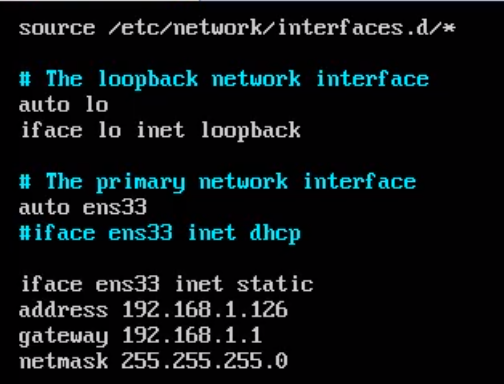
## 配置Ubuntu的固定IP地址

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

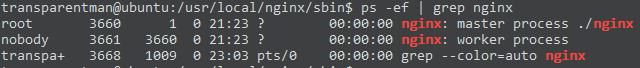
# nginx整体结构、进程模型

## 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：



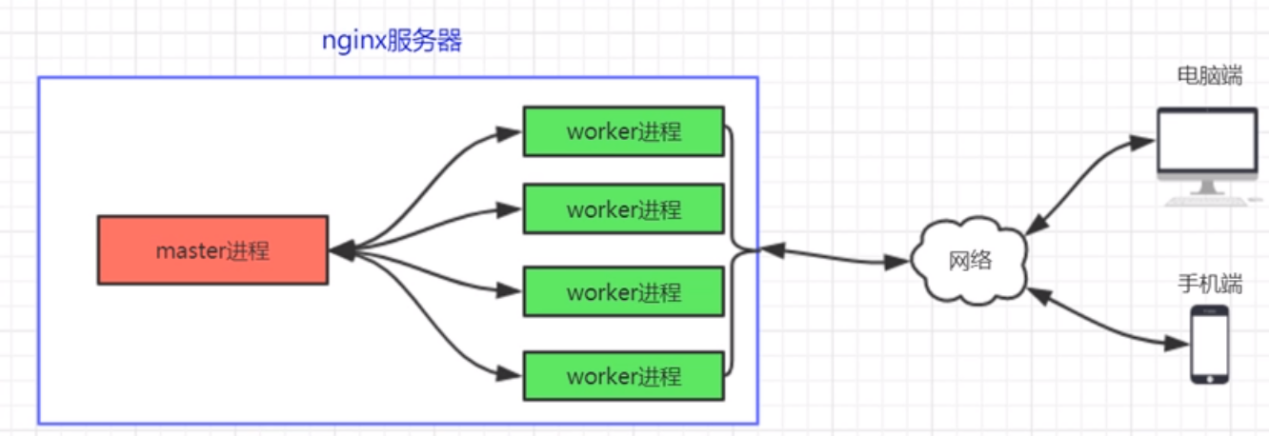
第一列：UID, 进程所属的用户id

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

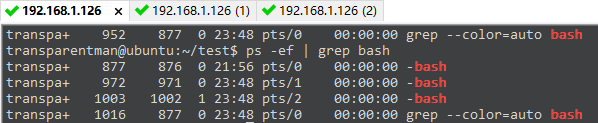
nginx能够热升级、能够热回滚

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

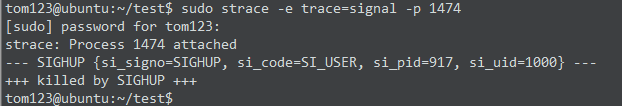
linux下调试分析诊断工具：可以跟踪程序执行时进程的系统调用以及所收到的信号；

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

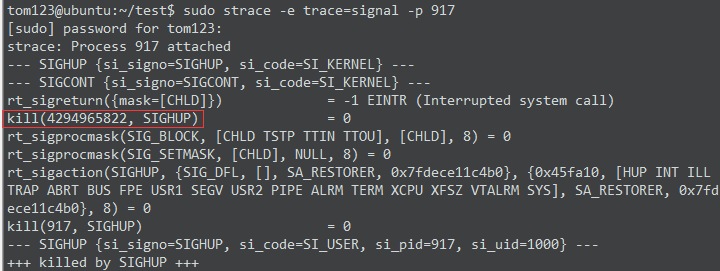
跟踪开启test的bash进程：sudo strace -e trace=signal -p 917

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");

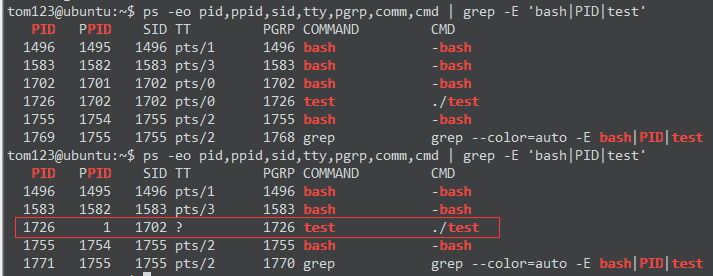
sleep(3);

}

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");

}

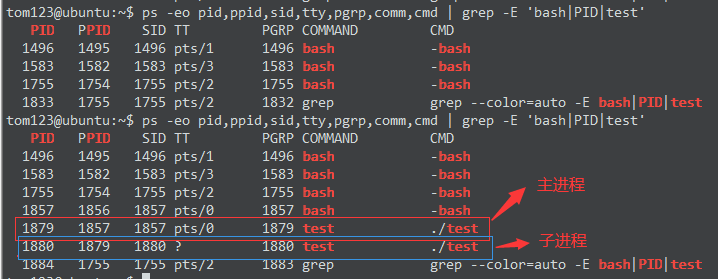
return 0;

}

return 0;

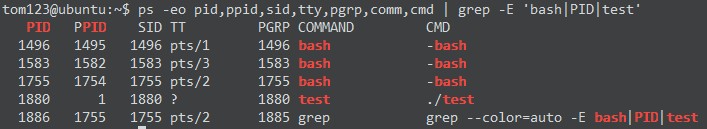
}

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

## 后台运行 &

后台执行就是执行这个程序的同时，你的终端可以干其他事情；如果不用后台执行，那么执行这个程序后，你的终端就只能等这个程序完成后才能继续执行其他的操作。

可以用命令fg切换到前台。

# 信号的概念、认识、处理动作

## 信号的基本概念

进程间常用的通信手段：发送信号。

信号：通知（事件通知），用来通知某个进程发生了某一个事情；事情、信号都是突发事件，也就是说信号是异步发生的；信号也被称呼为“软件中断”。

信号是如何产生的呢？

1. 某个进程发送给另外一个进程或者发送给自己（比如说热更新时，新的进程给老的进程发信号）；
2. 由内核（操作系统）发送个某个进程：
   1. 通过在键盘输入命令ctrl+c[中断信号]，kill命令。
   2. 内存访问异常，除数为0等等...硬件都会检测到并且通知内核；

信号的名字，都是以SIG开头；

UNIX以及类UNIX操作系统支持的信号数量各不相同（·10~60多个之间）；

信号是宏定义（数字，从1开始的正整数常量）；

在/目录下查找文件内容中关键词的命令：

sudo find / -name “signal.h” | xargs grep -in “SIGHUP”

## 通过kill命令认识一些信号

kill：kill 进程id，它的工作是给进程发个信号；

kill能给进程发送多种信号；

1. 如果单纯的用kill进程id，那么就是往这个进程发送SIGTERM信号（终止信号）
2. 如果用kill -数字 进程id，那么就能发出跟这个数字对应的信号。

kill 命令不同数字所能发出的不同信号

|  |  |  |
| --- | --- | --- |
| kill的参数 | 该参数发出的信号 | 操作系统缺省动作 |
| -1 | SIGHUP（连接断开） | 终止掉进程（进程没了） |
| -2 | SIGINT（终端中断符，比如ctrl+c） | 终止掉进程（进程没了） |
| -3 | SIGQUIT（终端退出符，比如ctrl+\） | 终止掉进程（进程没了） |
| -9 | **SIGKILL（终止）** | 终止掉进程（进程没了） |
| -18 | SIGCONT（使暂停的进程继续） | 忽略（进程依旧在运行不受影响） |
| -19 | **SIGSTOP（停止），可用SIGCONT继续，但任务被放到了后台** | 停止进程（**不是终止，进程还在**） |
| -20 | SIGTSTP（终端停止符，比如ctrl+z），但任务被放到了后台，可用SIGCONT继续 | 停止进程（**不是终止，进程还在**） |

## 进程的状态

ps -eo pid,ppid,sid,tty,pgrp,comm,stat | grep -E ‘bash|PID|test’

ps aux | grep -E ‘bash|PID|test’ // aux是BSD风格的显示格式

进程状态：

|  |  |
| --- | --- |
| 状态 | 含义 |
| D | 不可中断的休眠状态(通常是I/O的进程)，可以处理信号，有 延迟 |
| R | 可执行状态&运行状态(在运行队列里的状态) |
| S | 可中断的休眠状态之中（等待某事件完成），可以处理信号 |
| T | 停止或被追踪（被作业控制信号所停止） |
| Z | 僵尸进程 |
| X | 死掉的进程 |
| < | 高优先级的进程 |
| N | 低优先级的进程 |
| L | 有些页被锁进内存 |
| s | Session leader（进程的领导者），在它下面有子进程 |
| t | 追踪期间被调试器所停止 |
| + | 位于前台的进程组 |

kill只是发个信号，而不是单纯的杀死的意思。

## 常用信号列举：

|  |  |
| --- | --- |
| 信号名 | 信号含义 |
| SIGHUP（连接断开） | 是终端断开信号，如果终端接口检测到一个连接断开，发送此信号到该终端所在的会话首进程（前面讲过），缺省动作会导致所有相关的进程退出(上节课也重点讲了这个信号，xshell断开就有这个信号送过来)；  Kill -1 进程号也能发送此信号给进程； |
| SIGALRM（定时器超时） | 一般调用系统函数alarm创建定时器，定时器超时了就会这个信号； |
| SIGINT（中断） | 从键盘上输入ctrl+C（中断键）【比如你进程正跑着循环干一个事】，这一ctrl+C就能打断你干的事，终止进程；  **但shell会将后台进程对该信号的处理设置为忽略（也就是说该进程若在后台运行则不会收到该信号）**； |
| SIGSEGV（无效内存） | 内存访问异常，除数为0等，硬件会检测到并通知内核；其实这个SEGV代表段违例（segmentation violation），你有的时候运行一个你编译出来的可执行的c程序，如果内存有问题，执行的时候就会出现这个提示； |
| SIGIO（异步I/O） | 通用异步I/O信号，咱们以后学通讯的时候，如果通讯套接口上有数据到达，或发生一些异步错误，内核就会通知我们这个信号； |
| SIGCHLD（子进程改变） | 一个进程终止或者停止时，这个信号会被发送给父进程；（我们想象下nginx，worker进程终止时 master进程应该会收到内核发出的针对该信号的通知）； |
| SIGUSR1,SIGUSR2（都是用户定义信号） | 用户定义的信号，可用于应用程序，用到再说； |
| SIGTERM（终止） | 一般你通过在命令行上输入kill命令来杀一个进程的时候就会触发这个信号，收到这个信号后，你有机会退出前的处理，实现这种所谓优雅退出的效果； |
| SIGKILL（终止） | 不能被忽略，这是杀死任意进程的可靠方法，不能被进程本身捕捉 |
| SIGSTOP（停止） | 不能被忽略，使进程停止运行，可以用SIGCONT继续运行，但进程被放入到了后台 |
| SIGQUIT（终端退出符） | 从键盘上按ctrl+\  **但shell会将后台进程对该信号的处理设置为忽略（也就是说该进程若在后台运行则不会收到该信号）**； |
| SIGCONT（使暂停进程继续） | 使暂停的进程继续运行 |
| SIGTSTP（终端停止符） | 从键盘上按ctrl+z，进程被停止，并被放入后台，可以用SIGCONT继续运行 |

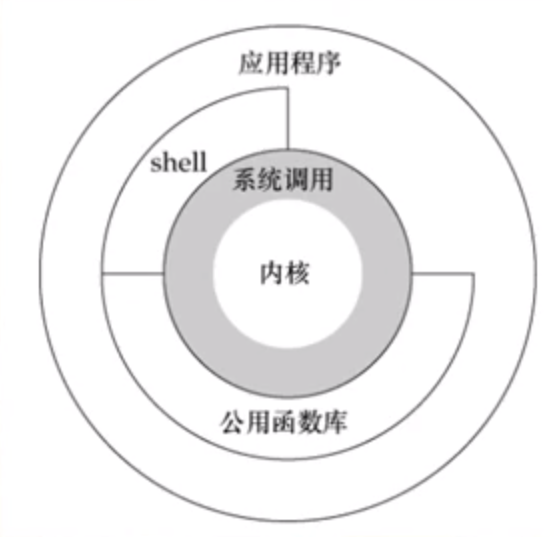
## 信号处理的相关动作

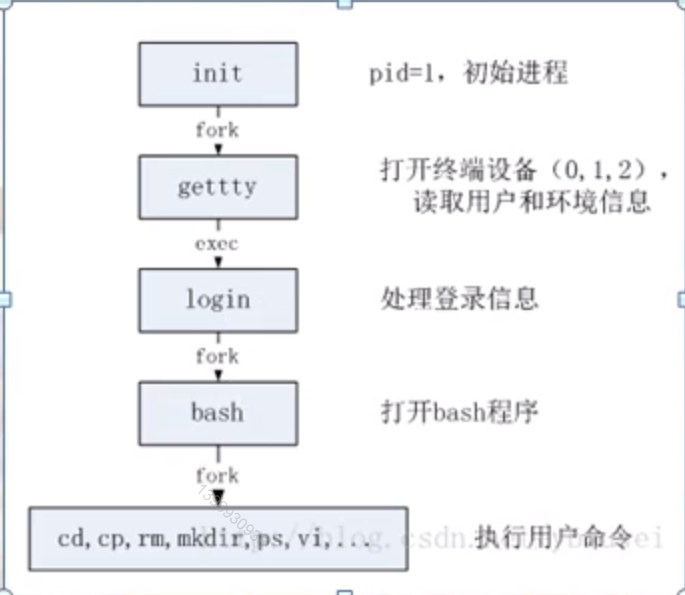
当某个信号出现时，我们可以按三种方式之一进行处理，我们称之为信号的处理或者与信号相关的动作。

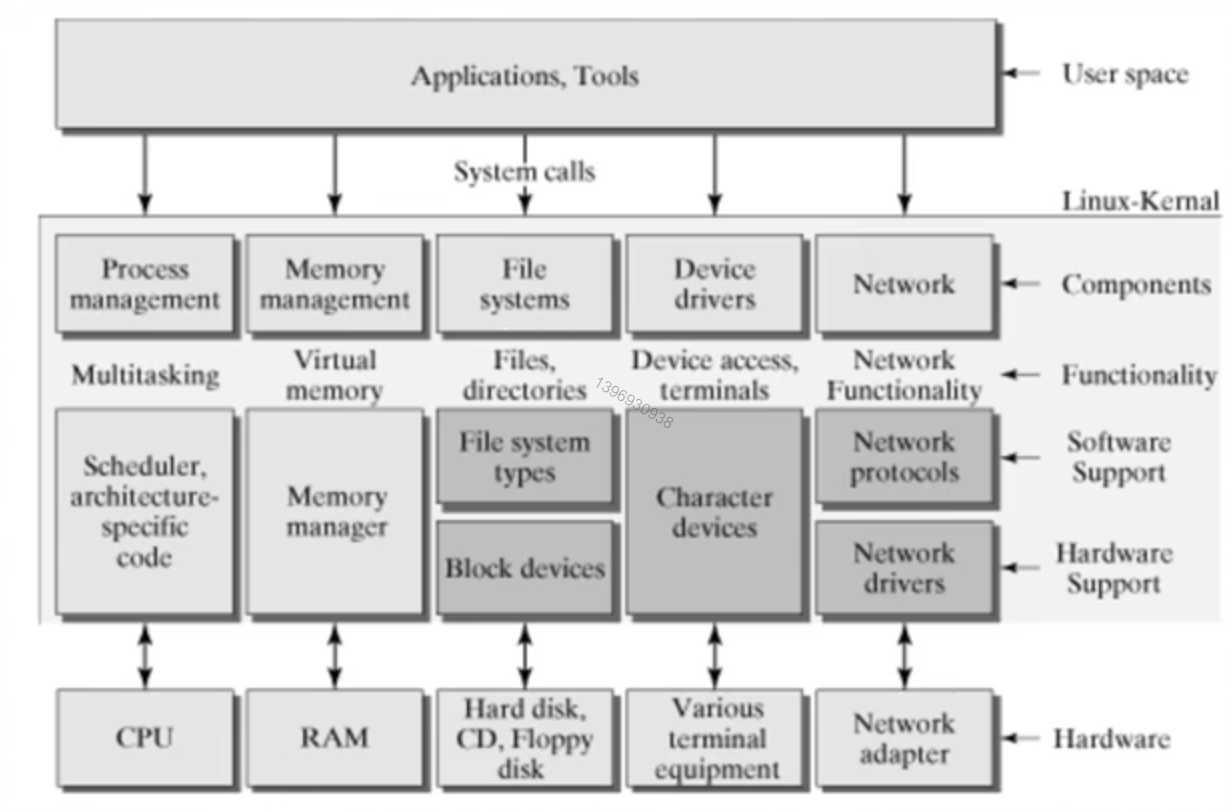
1. 执行系统默认动作，绝大多数信号的默认动作是杀死你这个进程。
2. 忽略此信号（但是不包括SIGKILL和SIGSTOP）
3. 捕捉该信号：我写个处理函数，信号来的时候，我就用处理函数去处理；（但是不包括SIGKILL和SIGSTOP）

# Unix/Linux体系结构 、信号编程初步

## Unix/Linux操作系统体系结构







类Unix操作系统体系结构分为两个状态：用户态 和 内核态

1. 操作系统/内核：用来控制计算机硬件资源，提供应用程序运行的环境；

我们写的程序，要么运行在用户态，要么运行在内核态。一般运行在用户态；

换个角度理解：用户态就是最外圈应用程序的活动空间；

1. 系统调用：就是一些函数（系统函数），只需要调用这些函数就可以了；
2. shell： bash(borne again shell[重新装配的shell])，bash是shell的一种，linux上默认采用的是bash这种shell；

通俗一点理解，bash是一个可执行程序，主要作用是 把用户输入的命令翻译给操作系统（命令解释器）;

shell分隔系统调用和引用程序，有胶水的感觉。

1. 用户态、内核态之间的切换

运行于用户态的进程可以执行的操作和访问的资源会受到极大限制（用户态权限小）；

而运行在内核态的进程可以执行任何操作并且在资源的使用上没有限制（内核态权限大）；

一个进程执行的时候，大部分时间是处于用户态下的，只有需要内核所提供的服务时才会切换到内核态，内核态做的事情完成后，又转回用户态；

比如说malloc() printf()：这种状态的转换时操作系统干的，不需要我们介入；

疑问：为什么要区分用户态、内核态？

大概有两个目的：

1. 一般情况下，程序都运行在用户态状态，权限小，不至于危害到系统的其他部分；当你要干一些危险的事情的时候，系统给你提供接口，让你去干。
2. 既然这些接口是系统提供给你的，那么这些接口也是操作系统统一管理的。

因为资源是有限的，如果大家都来访问这些资源，又不加以管理，问题一个是访问冲突，一个是被访问的资源如果耗尽，那么系统还可能崩溃；

因此系统提供这些接口，就是为了减少有限的资源的访问以及使用上的冲突。

那么什么时候从用户态切换到内核态去呢？

1. 系统调用，比如调用malloc();
2. 异常事件，比如来了个信号；
3. 外围设备中断；

总结起来，只需要知道用户态、内核态 两者根据需要自动切换即可。

## signal函数范例

信号来了之后，我们可以用signal函数来忽略或者捕捉。

### 可重入函数

可重入函数就是我们在信号处理函数中调用它是安全的。

可重入函数：在信号处理程序中保证调用安全的函数，这些函数是可重入的并被称为异步信号安全的。