linux命令：

ldd: 查看一个应用程序需要哪些依赖的动态库(list dynamic dependencies): ldd /bin/ls

mount umount: mount 设备名 挂载点 mkdir /mnt/cdrom 一般CDRoom的设备文件是/dev/hdc,软驱的设备名是/dev/fd0

挂载光驱: mount /dev/hdc /mnt/cdroom

挂载软驱:mkdir /mnt/floppy mount /dev/fd0 /mnt/floppy

umount: umount -v /mnt/mysrc (通过挂载点卸载)

umount： umount -v /dev/sda1 (通过设备名卸载)

dd: 用于读取，转换并输出数据。dd可以从标准输入或文件中读取数据，根据指定的格式来转换数据，再输出到文件，设备或标准输出。(if=输入文件名，默认为标准输入，即指定源文件。of=输出文件名，默认为标准输出，即指定目的文件。 conv=<关键字>，关键字可以有多种：ascii:转换ebcdic为ascii, lcase:把大写字符转换为小写字符，ucase:把小写字符转换为大写字符)

显示目录和文件命令

ls -l;

stat filename(显示文件的详细信息：文件大小，类型，存取权限)   
dir(显示指定文件夹和目录的命令) dir directoryname ;

du：查看文件或目录占用空间情况

df(filesystem)：查看磁盘分区情况和大小

修改目录，文件权限

chmod(改变目录或文件权限) chmod 777(每个人(拥有者，群组，其他人)都有读写执行权限) filename

chmod a+x filename所有人都可以执行chmod u+x filename文件拥有者可以执行

(a(all):所有用户u(user):文件或目录所有者o(others):其他用户，g(group)群组)

+：添加某个权限 -：取消某个权限 r:可读(4) w:可写(2) x:可执行(1)

查看权限： ls –l filename: -rw-rw-r—

chown:将指定文件的拥有者改为指定的用户或组，用户可以是用户名或用户ID；root(系统管理员)用户使用这个命令：在将文件拷贝到另一个用户名的目录下之后，让这个用户拥有使用该文件的权限。

chown runnob:runnobgroup filename 拥有者设为runnob,群体的使用者runnobgroup

chown -r runnob:runnobgroup \* 所有文件的拥有者设为runnob，群体的使用者设为runnobgroup

创建和删除目录

mkdir; rmdir; rm -f

在当前不存在的目录下创建文件夹或文件：mkdir -p aaa/bbb.txt mkdir -p aaa/bbb/ccc.txt

touch filename 创建文件

cp filename copyname 复制文件

mv filename ./folder/ 移动文件 或重命名

scp(secure copy,是Linux系统下基于ssh登陆进行安全的远程文件拷贝命令):linux之间复制文件和目录

从本地复制到远程：  
scp localfile remote\_username@remote\_ip:remote\_folder 拷文件

scp localfile remote\_username@remote\_ip:remote\_file 拷文件

scp -r local\_folder remote\_username@remote\_ip:remote\_folder 拷目录

从远程复制到本地：从远程复制到本地，只需将从本地复制到远程的命令的后2个参数调换顺序即可。

scp [root@www.runoob.com:/home/root/others/music](mailto:root@www.runoob.com:/home/root/others/music) /home/space/music/1.mp3

scp -r [www.runoob.com:/home/root/others/](http://www.runoob.com:/home/root/others/) /home/space/music/

wget 下载文件到本地

显示文件内容的命令：

cat filename 显示filename的所有内容(可以用vim替换)

cat -n textfile1 > textfile2 把textfile1的内容加上行号后输入textfile2

head: 用于显示文件的前m行内容 head -n m filename

tail: 用于显示文件的后m行内容 tail –n m filename

动态输出最新m行(最末尾)的日志： tail -f m filename -f表示刷新flush

查找命令:

find: 用于在指定目录下查找文件和子目录；若不设置任何参数，则find命令将在当前目录下查找子目录与文件。并且将查找到的子目录和文件全部显示。

find . –name “.cpp” 将当前目录及其子目录下所有后缀是cpp的文件列出来

-i 忽略大小写 -ctime find . –ctime -20 最近20天内更新的文件列出来

which: 用于查找文件。 在环境变量$PATH设置的目录里查找符合条件的文件

which bash

whereis: 用于查找文件。在特定目录中查找符合条件的文件。这些文件属于原始代码，二进制文件，帮助文件 whereis -b查找二进制文件

file： 查看文件类型 file /var/mail

locate: 从保存文档和目录名称的数据库内查找文档或目录 locate filename

locate与find不同：find是去硬盘找，locate只是在/var/lib/slocate资料库中找。Locate的速度比find快，它并不是真的查找，而是查找数据库，文件数据库在/var/lib/slocate/slocate.db中。更新数据库命令:updatedb

sort:将文本内容以行单位排序: sort filename 默认的方式将文本文件的第一行以ASCII码次序排列 -b 忽略每行前面开始的空格字符 sort -b shelltest -o将排序后的结果存入指定文件。

cat testfile

test 30

Hello 95

Linux 85

sort testfile

Hello 95

Linux 30

test 30

linux查看环境变量:set,env,export命令都可以查看环境变量(简单一点的方式:echo $PATH)

env命令显示当前用户的变量。

linux设置环境变量：

export可以新增，修改，删除环境变量，供执行的程序使用。export的效力仅限于该次登入操作。

动态库路径的设置：export LD\_LIBRARY\_PATH = /home/….(动态库的目录) 只在当前的session有效，可以修改文件实现任何session都有效

环境变量设置：

export PATH=$PATH:/opt/bin/….. 当前session有效

修改profile文件 在/etc/profile文件中，添加export PATH=$PATH:/opt/bin/…. 永久生效

grep:用于查找文件里符合条件的字符串。如果发现文件的内容符合指定的查找内容，grep指令会把那一行显示出来

grep test \*file 查找后缀有file字样的文件中包含test字符串的文件，并打印出该字符串的行

grep -r update /etc/acpi 以递归的方式查找 /etc/acpi及其子目录下所有文件中包含字符串”update”的文件，并打印出该字符串所在的行的内容

grep -v test \*test\*(查找文件名包含test的文件中不包含test的行) 反向查找，通过-v，查找不符合条件行的内容

shutdown: 关机

reboot：重启

用户操作命令

su: 切换用户命令,变更为其他用户者

su root su jjz

sudo: 系统管理员的身份执行命令

passwd: 用来设置指定用户的密码 passwd jjz Enter new UNIX password: //输入新密码 Retype new UNIX password: //确认密码

ln: 为某一个文件在另外一个位置建立一个同步的链接；当我们需要在不同的目录，用到相同的文件时，我们不需要在每一个需要的目录下都放一个必须相同的文件。只需要在其他目录下用ln命令链接(link)它就可以，不必重复占用磁盘空间。

硬链接(hard link): 一个文件有多个别名(名称)，不占用实际空间。不能给目录创建硬链接；只能在同一个文件系统中才能创建。目的：让多个不在同一目录下的文件名，能同时修改同一个文件，其中一个修改后，所有与其有硬链接的文件都一起修改了。删除一个引用不会影响别的引用，除非所有引用都删除，才真正删除文件(而对软链接原文件删除，直接导致软链接不可用)。

软链接(symbolic link)：文件中存放的是被链接文件的路径地址；可以跨越文件系统；可以对一个不存在的文件名进行链接；可以对目录进行链接

ln -s log2013.log link2013 给log2013.log创建软连接link2013;若log2013.log丢失,link2013将丢失

ln log2013.log link2013 给文件log2013.log创建硬链接link2013

ssh:安全模式下远程登录

ssh root@192.168.1.123

telnet:远程登录 tcp协议，23端口

telnet 主机名 telnet 192.168.0.5

tcpdump –nn tcp port 23 host 210.27.48.1 截取主机210.27.48.1接收或发出的telnet包

netstat: 显示linux系统网络状态

-a 显示连接中的socket -t 显示tcp连接 –u 显示udp连接 –p显示pid和进程名称

查看端口使用情况：netstat –anpt

查看指定端口使用情况： netstat –anpt | grep 2181

查看占用指定端口的进程： netstat –anpt | grep 2181 或者lsof –i tcp:2181

ping: ping ip 使用ICMP报文，封装在ip包内，IP协议。原理：给目标网络发一个数据包，再要求对方返回一个同样大小的数据包来确定两台及其是否连接以及延时多少。

tcpdump(类型：ip/tcp/udp port host/net)： tcpdump tcp port 23 host 210.27.48.1

ftp(文件传输协议): 使用TCP。包括两个部分：FTP服务器(用来存储文件)，FTP客户端。用户使用FTP客户端通过FTP协议访问位于FTP服务器上的资源。开发网站时，通常利用FTP协议把网页或程序传到WEZB服务器上。FTP使用20(传输数据),21(传输控制信息)两个端口

lsof(list open files):显示所有进程打开的所有文件(每行信息包括pid,user,fd,device,size(这个文件的大小),name(打开文件的确切名称))。被打开的文件可以是普通的文件，目录，socket，字符设备文件，共享库，符号链接

lsof abc.txt 显示打开文件abc.txt的进程。lsof -c abc 列出abc进程现在打开的文件。

lsof -c –p 1234列出进程号为1234的进程所打开的文件

lsof：列出所有打开的文件。

lsof –u username 列出某个用户打开的文件信息

查看网络连接情况： lsof –i:port 显示与指定端口相关的网络信息 lsof -i tcp:80

列出所有打开的网络连接(tcp和udp)： lsof –i 使用-i显示所有网络连接

列出所有打开的tcp连接： lsof -i tcp

列出所有打开的udp连接：lsof -i udp

列出到指定主机的连接：lsof [–i@172.16.12.5](mailto:–i@172.16.12.5)

列出到指定主机指定端口的连接：lsof [–i@172.16.12.5:22](mailto:–i@172.16.12.5:22)

列出正在监听/建立的端口：lsof –i –sTCP:LISTEN/ESTABLISHED 或者netstat –anpt | grep LISTEN/ESTABLISHED 或者 lsof –i | grep LISTEN/ESTABLISHED

列出指定用户打开了什么(包括网络，文件)：lsof –u daniel u(user)

进程名和进程号互查：ps aux | grep pid/processname

进程的4中状态： 1.运行 2.中断(休眠，受阻，在等待某个条件的形成或接收到信号) 3.僵死(进程已终止，但进程描述符存在，直到父进程调用wait()后释放) 4.停止

lsof/netstat的一个重要作用：根据进程查端口，根据端口查进程

根据进程pid查端口：lsof -i | grep pid 或者 netstat –anpt | grep pid

根据端口port查进程：lsof –i:port 或者 netstat –anpt | grep port

linux命令都带有参数，有些命令可以接受标准输入作为参数。cat /etc/passwd | grep root

管道命令的作用是将cat /etc/passwd的标准输出转换为标准输入，提供给grep root作为参数，grep命令可以接受标准输入作为参数(等价于grep root /etc/passwd)。然而，大多数命令不接受标准输入作为参数，只能直接在命令行输入参数， echo “hello world” | echo无效。xargs命令的作用，是将标准输入作为命令行参数。

echo “hello world” | xargs echo

echo “one two three” | xargs mkdir 默认情况下，xargs将换行符和空格作为分隔符，把标准输入分解成一个个命令行参数。 mkdir依次创建三个目录。

xargs:能够捕获一个命令的输出，然后传递给另外一个命令。之所以能用到这个命令，关键是由于很多命令不支持|管道来传递参数，因此就有了xargs命令。一般与管道一起使用。

find / -type f –name ‘\*.log’ | xargs grep ‘error’ 从根目录开始查找所有扩展名为.log的文本文件，并找出文件中包含’error’的行。 (find /usr/sbin | ls -l 这个命令错误)

xargs结合find使用：

find . –name ‘\*.in’ | xargs grep ‘thermcontact’ 从当前目录开始查找所有扩展名为.in的文本文件，并找出文件中包含”thermcontact”的行

find /path -type f | xargs rm

wc(word count): 统计指定文件中的字节数，字数，行数，并将统计结果显示输出 –c 统计字节数 –l统计行数 –w统计字数

wc testfile wc testfile testfile\_1 testfile\_2 #统计三个文件的信息 wc –l test.txt 统计行数

通过管道统计prog.c函数中for语句通过的次数 grep “for” proc.c | wc –l

less,more: 可以随意浏览文件，支持翻页和搜索，支持向上翻页和向下翻页。

less logfile

nohup和&的区别：

nohup: no hangup，不挂断的运行命令，即可以忽略所有挂断(SIGUP： ctrl+c)信号。如果不将nohup命令的输出重定向，输出将附加到当前目录的nohup -s.out文件中。

nohup：不挂断的运行，注意仅仅nohup没有后台运行功能。&:表示后台运行。使用nohup运行命令可以使命令永久的执行下去，和用户终端没有关系，如果断开ssh，不会影响命令的执行。注意仅仅&指在后台运行，当用户ssh退出，挂起时候，命令自动跟着推出。

因此，将nohup和&结合起来： nohup cmd &。这样命令就永久在后台执行。

sh test.sh & : 将sh test.sh任务放到后台，即使关闭shell退出当前session依然继续运行，但标准输出和标准错误信息会丢失。

nohup sh test.sh: 将sh test.sh任务放到后台，关闭标准输入，前台不再能够接受任何输入(标准输入)，重定向标准输出和标准错误到当前目录下的nohup.out文件，即使关闭shell退出当前session依然继续运行。

nohup sh test.sh &: 将sh test.sh任务放到后台，但是依然可以使用标准输入，前台能够接收任何输入，重定向标准输出和标准错误到当前目录下的nohup.out文件，即使关闭shell退出当前session依然继续运行。

ps aux / -ef 用来显示所有的进程 (ps -ef查看全部格式的全部进程)

ps –T –p pid 查看某个进程的线程

top –H –p pid 使用top查看某个特定进程内线程的运行情况

vmstat -5(每5秒刷新1次):查看CPU使用率，内存，虚拟内存交换情况。IO读写情况

虚拟内存就是为了满足物理内存的不足而提出的概念，利用磁盘空间虚拟出的一块逻辑内存，用作虚拟内存的磁盘空间被称为交换空间(swap space)。内核会在适当时候将物理内存中不经常使用的数据块自动交换到虚拟内存中(交换空间)，而将经常使用的信息保留到物理内存。

swap:以磁盘来伪装成内存。 内存与swap之间按照内存页为单位来交换数据的，一般Linux中页的大小设置为4kb。而内存与磁盘则是按照块来交换数据的。交换的操作是磁盘IO操作而不是内存的load与store操作。swap设置过大，磁盘IO操作对于服务器负载压力很大。

linux查看资源使用情况：

查看内存使用情况： free –m //以MB为单位,此时同样可以看到swap的大小

查看磁盘空间使用情况： df –h

查看系统中各个进程的资源占用状况： top

磁盘配额管理：quota命令(为了合理的分配磁盘空间，需要限制用户对磁盘空间的使用)。

正则表达式: -i 忽略大小写

^: 行的开始 ^love 匹配所有以love开头的行

grep -n ‘^the’ regular\_express.txt -n 列出行号 ^the the开头

grep -n ‘^[a-z]’ regular.txt ^[a-z] 小写字母开头

grep –n ‘^[^a-zA-Z]’ regular.txt ^[^a-zA-Z] 非英文字母开头

[^a-z]: 方括号内表示非

$: 行的结尾 love$ 匹配所有以love结尾的行

grep -n ‘.$’ regular.txt 找以.结尾

grep -n ‘^$’ regular.txt 找空白行

正则表达式中.(小数点)代表一定有一个任意字符，\* 前面字符出现0或多次

(bash中通配符\*代表0或多个字符) ? 前面字符出现0或1次 + 前面字符出现至少1次

grep –n ‘g..d’ regular.txt 找出g??d的字符串

1:”Open Source” is a good mechanism to develop programs.

16.The world is the same with glad.

grep –n ‘ooo\*’ regular.txt 表示2个o后面再加上0或多个o(至少2个O以上的字符串)

grep –n ‘goo\*g’ regular.txt 表示开始和结尾都是g,中间至少有一个o

grep –n ‘g.\*g’ regular.txt 表示首尾是g,中间任意多个字符

限定连续正则表示式字符范围:

\{n,m\}：连续n到m个的前一个字符 \{n\}: 连续n个的前一个字符

因为{}的符号在shell是有特殊意义的。所以用\转义

grep -n ‘o\{2\}’ regular.txt 2个连续的o

grep -n ‘go\{2,5}\’ regular.txt g后面跟2到5个o

grep -n ‘go\{2,\}g’ regular.txt gooo\*g

linux下的两个特殊文件: /dev/null 和/dev/zero

/dev/null: 空设备，是一个特殊的设备文件，它等价于一个只写文件，并且所有写入它的内容都会永远丢失，而尝试从它那儿读取内容则什么也读不到。然而，/dev/null对命令行和脚本都非常有帮助。

cat filename > /dev/null, 则不会得到任何信息，将本该通过标准输出显示的文件信息重定向到了/dev/null. 等价于cat filename 1> /dev/null(默认重定向的1就是标准输出，2是标准错误输出)。

如果cat filename，若filename不存在，系统会报错：不存在那个文件或目录。如果不想看到错误输出，可以禁止标准错误： cat filename 2> /dev/null

有时候需要删除文件的内容而不删除文件本身(删除日志文件内容，文件本身不删除) : cat /dev/null > /var/log/messages

如果希望执行某个命令，但又不希望在屏幕上显示输出结果，那么可以将输出重定向到/dev/null cmd > /dev/null (起到静止输出的效果)

/dev/zero: 在类unix操作系统中，/dev/zero是一个特殊的文件,当你读它时，它会提供无限的空字符(NULL,ASCII NULL,0x00)。其中的一个典型用法是用它提供的字符流来覆盖信息，另一个常见用法是产生一个特定大小的空白文件。

/dev/random和/dev/urandom： /dev/random和/dev/urandom是linux系统中提供的随机伪设备。这两个设备的人物，是提供永不为空的随机字节数据流。很多解密程序与安全应用程序 (SSH Keys, SSL Keys等) 需要它们提供的随机数据流。

区别：

这两个设备的差异在于：/dev/random的random依赖于系统中断，因此在系统中断数不足时，/dev/random设备会一直封锁，尝试读取的进程就会进入等待状态，直到系统的中断数充分够用，/dev/random设备可以保证数据的随机性。

/dev/urandom不依赖系统的中断，也就不会造成进程忙等待，但是数据的随机性也不高。

git仓库ssh key的设置:

cd ~/.ssh，进入目录。文件夹下id\_rsa.pub这个文件内容就是key。复制到git仓库中(setting下的SSH and GPG keys)保存

如果本地没有这个密钥，用ssh-keygen -t rsa -C “邮箱地址”这个命令生成新的ssh密钥，回车提示输入密码，这个密码在提交项目时使用，若为空，则提交项目时不用输入密码。运行此命令，会生成id\_rsa.pub文件。

ssh-keygen: 用来生成ssh公钥认证所需要的公钥和私钥文件。

samba服务：作为windows和linux交互的媒介，让windows用户使用图形界面访问linux文件系统(在linux上配置samba服务并且为samba服务器添加用户) virtualbox必须使用桥接模式。

早期文件在不同主机之间传输使用FTP，存在的问题时无法直接修改主机上面的文件(想要修改主机上的某个文件时，需要将server端的文件下载到client端后才能修改，然后将文件重新回传到server端)。Linux上存在可以直接通过client端进行server端文件的修改(使用NFS协议) 。windows上也存在类似的协议(使用网上邻居：CIFS协议) ，但samba可以让windows和linux两个不同的平台相互共享文件。

开启samba服务： sudo /etc/init.d/smbd restart 在windows上访问sambda服务器：\\+10.84.70.178

awk, sed

awk:文本分析工具。awk把文件逐行读入，以空格为默认分隔符将每行切片，切开部分再进行各种分析处理。awk ‘{[pattern] action}’ {filenames} awk只能用’ ‘(单引号)

log.txt文本内容：

2 this is a test

3 Are you like awk

This’s a test

10 There are orange,apple,mango

将每行按空格或tab分隔，输出文本中的第1，第4项

awk ‘{print $1, $4}’ log.txt

2 a

3 like

That’s

10 orange,apple,mango

使用”,”分隔：需要指定分隔字符 awk –F -F相当于内置变量FS

awk –F, ‘{print $1, $2}’ log.txt 或者用awk ‘BEGIN{FS=’,’} {print $1, $2}’ log.txt

awk –v #设置变量

awk -va=1 ‘{print $1, $1+a}’ log.txt #设置变量a = 1

2 3

3 4

That’s 1

10 11

使用运算符：过滤第一列大于2的行：awk ‘$1>2’ log.txt

#输出

3 Are you like awk

This’s a test

10 There are orange,apple,mango

过滤第一列等于2的行：awk ‘$1==2 {print $1, $3}’ log.txt #输出: 2 is

过滤第一列大于2并且第二列等于’Are’的行：

awk ‘$1>2 && $2==’Are’’ {print $1, $2, $3}’ log.txt #输出 3 Are you

sed:利用sed编写脚本来处理文件。

系统管理命令: stat(显示指定文件的详细信息，比ls更详细) uname(显示系统信息) top(动态显示当前进程耗费资源) ps(显示此刻进程状态) du (查看目录大小：du –h aaa) df –h (f表示filesystem,查看磁盘大小) ifconfig 查看网络情况 ping测试网络联通 netstat(显示网络状态信息) kill free(对/proc/meminfo收集的信息的一个概述)

tar shutdown reboot halt

ctrl+c和ctrl+z的区别：都是中断命令，作用却不一样

ctrl+c是强制中断程序的执行，程序结束了。

ctrl+z是将任务中断(suspend)，但是此任务并没有结束，任务在进程中只是维持挂起的状态，用户可以使用fg/bg操作resume前台或后台的任务。fg命令resume被中断的任务，把任务放在前台执行。bg命令resume任务，并把被中断的任务放在后台执行(resume)

eg: vim一个文件时，想要执行其他操作，又不打算保存关闭vim 文件,可以ctrl+z，shell会将vim进程挂起(suspend)。当结束了其他操作之后，可以用fg命令继续vim文件(job:查找当前)

fg: linux把后台工作恢复到前台执行。 fg % jobid

bg: linux把暂停作业拿到后台处理。 bg jobid

linux中jobs的管理常用命令：

将命令或脚本放到后台去执行： & eg: ./run.sh &

ctrl+z: 把作业暂停到后台

jobs: 列出作业

用kill发送信号来操作jobs: kill -9 %job\_id(立刻强制删除一个作业)。Kill -15 %job\_id(以正常方式终止一项作业)

Linux查看某个进程中的线程：

ps,top默认查看进程，同样也可以查看进程中的线程

ps –L(查看线程) ； top –H(显示所有的线程s) –p pid(显示所有的进程)

在linux服务器大并发调优时，需要预先调优linux参数，修改linux最大文件句柄数,打开coredump等：

ulimit –a 显示当前的各种用户进程限制情况

ulimit –c unlimited 生成core文件size -c: core file size

ulimit –n 2048 可以打开多达2048个文件句柄 -n: open files

这种方法只对当前进程有效，重新打开一个shell或进程，参数就又还原了；永久保存的方法是修改系统参数

vim /etc/security/limits.conf

gdb调试：条件断点

b line-or-function if exp (b 46 if testsize == 100)

进入函数的单步执行：step

已进入某个函数，想退出该函数返回到它的调用函数中：finish

结束当前循环：until

调试coredump

gdb ./edgefe coredunp\_\*\*\*\*\* 如果发现没有调试信息,则 gcc –g

bt: 查看堆栈信息；f no: 查看某一栈帧；查看this指针：p \*this list: 显示代码

查看core进程的所有线程堆栈

查看运行的线程及其堆栈：pstack或者pstree(树形结构显示父子线程关系)

ps aux | grep a.out //查看当前运行的进程

info threads

thread n (切换到第n个线程)

bt (查看第n个线程的堆栈)

gdb调试子进程：

set follow-fork-mode [parent/child]

进入gdb后，直接设置，默认是parent. 使用show follow-fork-mode 来查询当前fork模式

gdb查看变量类型：

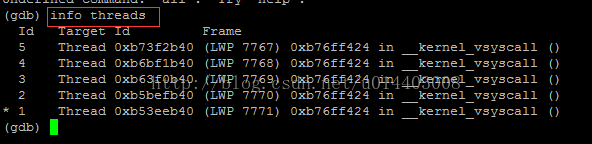
利用gdb查看线程信息

首先利用ps获得进程id, gdb attach processId;



查看正在运行的线程(有哪些线程，显示线程号)： info threads

切换线程: thread n(线程号)



打开所有线程的堆栈信息

thread apply all

bt

gdb调试多线程命令：

info threads 显示当前可以调用的所有线程

thread id 切换到某个要调试的线程

break thread\_test.c:123 thread all 在所有线程中相应的行上设置断点

thread apply id1 id2 command 让一个或多个线程执行gdb命令command

thread apply all command 让所有被调试线程执行gdb命令command

实际使用过多线程调试的人都可以发现，在使用step或者continue命令调试当前线程的时候，其它线程也是同时执行的，怎么只让被调试程序执行呢？

set scheduler-locking off 不锁定任何线程，也就是所有线程都执行(默认行为)

set scheduler-locking on 只有当前线程会执行

glibc:

glibc是GNU发布的libc库，即C运行库。Glibc是linux系统中最底层的api,几乎其他任何运行库都会依赖于glibc。是一种C函数库，是程序运行时使用到的一些API集合，一般是已预先编译好的，以二进制代码形式存在于linux系统中(功能包括：字符串处理，信号处理，pthread,socket,io,文件，动态内存分配malloc)

kill -1 pid: 让进程重启

kill -2 pid sigint对当前进程发送结束信号，信号被进程树接收到(不仅当前进程，子进程也会收到结束信号) = ctrl +c; ctrl+z将任务中断(执行睡眠)，但此任务并没有结束，它任然在进程中只是维持挂起状态使用fg命令恢复后台。

kill -9 pid: 数字9表示terminate(无条件终止，强制杀死进程(发送sigkill)，这个信号不能被捕获或忽略)

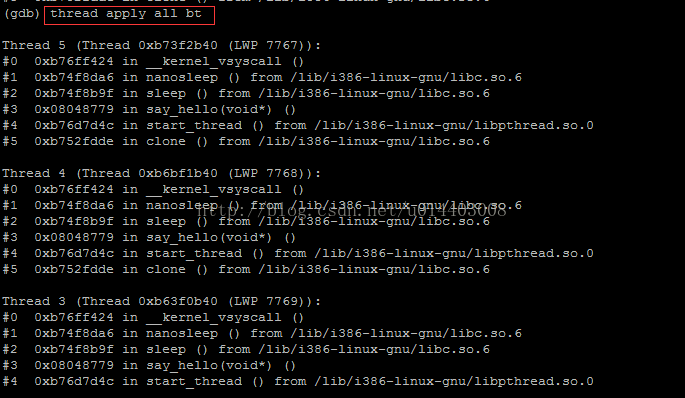
kill -6 pid 主动让进程abort/coredump(很多时候，我们需要知道变量的值，但加log再编译，又太慢了。单步调试太麻烦，主动core来看变量的值。开启core,ulimit –c,编译时-g)发送sigabrt信号

g++ test.cpp –g && ./a.out

ps –aux | grep a.out | awk ’{print $2}’ | xargs kill -6

分析core文件： gdb a.out core bt f n i locals

默认是 15：正常结束进程，是kill命令的默认信号 sigterm请求中断(让进程有好的退出，因为它可以被阻塞，所以有的进程不能被结束时，用kill -9强行终止)



查看指定线程堆栈信息

thread apply threadID bt

产生死锁的4个必要条件：

互斥条件；请求与保持条件；不剥夺条件；循环等待条件；

调试死锁(cpu占比很高，内存占比普通)：你项目中遇到过哪些问题，怎么解决的？多线程死锁，内存问题

情形：线程A占有了第一把锁，想要得到第二把锁。线程B得到了第二把锁，想要得到第一把锁。这样两个线程就发生了死锁现象。

gdb调试死锁方法：

ps aux | grep edgefe

gdb attach pid

thread apply all bt (查看所有线程堆栈)

查看每个线程的堆栈，寻找有pthread\_mutex\_lock(&lock1)的栈帧。打印锁的信息p lock1。可以看到\_owner = 7198。pthread\_mutex\_lock(&lock2), p lock2: 看到\_owner=7199(被哪个线程占有)

gdb attach processID ,ctrl +c中断程序，info threaeds查看所有线程，使用thread n切换线程，在线程中输入bt查看线程堆栈，定位程序停留位置，查看线程锁的占用或者是否有死循环

查看锁的owner是哪个线程。

Linux启动过程：

通电—>BIOS --->主引导记录--->操作系统。 一般认为加载内核是Linux启动流程的第一步。

加载内核： 操作系统接管硬件后，首先读入/boot目录下的内核文件

shell:

单个小括号(cmd1;cmd2;cmd3): 命令组。括号中的命令将会新开一个子shell顺序执行，括号中多个命令之间用分号隔开，最后一个命令可以没有分号。 $(cmd),命令替换：等同于`cmd`(反引号)， 将cmd执行一次。 初始化数组: array= (a b c d)

${var}: 用来做变量替换

$(()):双小括号，用作整数运算: + - \* / % & | …… ！ ((expr))：结构扩展并计算一个算术表达式的值，如果表达式的结果为0，那么为false，而一个非0值的表达式返回为true。可用作if, while,for等逻辑判断。

a=5 b=7 c=2

echo $((a+b\*c)) 19 echo $(( (a+b)/c ))

int=1

while(($int < 5)) #### ((expr))： 计算expr的值，并返回逻辑判断

do

echo “$int”

let “int++”

done

shell脚本传递参数： $n

$0: 表示执行的文件名 $1：表示传递给脚本的第一个参数……

$#:传递到脚本的参数个数 $\*：以一个单字符串显示所有向脚本传递的参数

$@: 与$\*相同，

shell中变量与等号之间不能有空隙

shell字符串：

单引号字符串(单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的)；

双引号字符串(双引号里可以有变量，双引号里可以出现转义字符)

str=”Hello, I known you are \”${name}\” ! \n” 输出Hello, I known you are “jjz” !

拼接字符串： 双引号拼接name=”jjz” greeting=”hello, ${name}” echo ${greeting}

获取字符串长度： str=”abcd” echo ${#str} #输出4

提取子字符串: string=”runoob is a great site” echo ${string:1:4} #输出 unoo

shell数组(仅支持一维数组)：用括号表示数组，数组元素用空格符号隔开

arr=(value0 value1 value2) arr[0]=val0 arr[1]=val1 arr[2]=val2

读取数组元素: val=${arr[n]} echo ${arr[@]}获取数组中的所有元素

获取数组的长度: len = ${#arr[@]}

获取单个数组元素的长度: len = ${#arr[0]}

shell基本运算符

原生bash不支持简单的数学运算，但是可以通过其他命令来实现，例如awk和expr,expr最常用。expr是表达式计算工具，使用它能完成表达式的求值操作。

val=` expr 2 + 2` echo “sum: ${val}” 表达式和运算符之间要有空格2+2不行。完整表达式要被` `包含(反引号)

`expr ${a} + ${b}`

或者$((a+b))

关系运算符：数字比较用 -eq -ge -gt -le

-eq:是否相等 –ne:是否不相等 –gt(greater than) -lt(less than)

-ge(greater equal than) -le(less equal than)

a=10 b=20

if [$a –eq $b] #### []执行基本的算数运算

then

echo “equal”

else

echo “not equal”

fi

逻辑运算符 && ||

&& [[$a –lt 100 && $b –gt 100]] 返回false

|| [[$a –lt 100 || $b –gt 100]]

字符串运算符： 字符串比较用 == !=

a=”abc” b=”def”

[$a = $b]返回false [$a != $b] 返回true [$a]返回true 检测字符串是否为空

for循环：

for var in item1 item2 item3 …itemN

do

cmd1 cmd2

done

###写成一行： for var in item1 item2…itemN; do cmd1 cmd2; done;

C++内存泄露几种情况：

在类的构造函数，析构函数中没有匹配的调用new和delete函数；

释放对象数组时在delete中没有使用方括号；

缺少拷贝构造函数，缺少重载赋值运算符(两次释放相同的内存double free,造成堆的崩溃)：若没有定义拷贝构造函数，编译器调用默认的拷贝构造函数，逐个成员拷贝的。如果逐个拷贝成员来复制指针变量，两个指针对象拥有同一个动态分配的内存地址空间，会造成两次释放同一地址空间(double free)。所以类中有指针成员变量，需要自己写拷贝构造函数和赋值函数，析构函数。

没有将基类的析构函数定义为虚函数

如何解决内存泄露：智能指针，RAII，代码检查，第三方工具

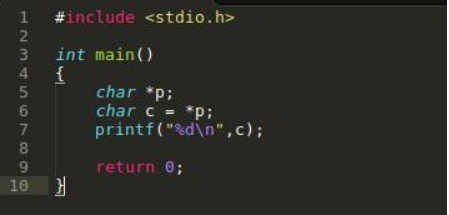
Linux 下内存问题检测神器:valgrind(开源工具) 所有代码要g++ -g

常见的内存问题一共7中：动态内存泄露；资源泄露，比如文件指针没close;动态内存越界；数组内存越界；动态内存double free；使用野指针(使用未初始化的指针)；释放野指针(即释放未初始化的指针)

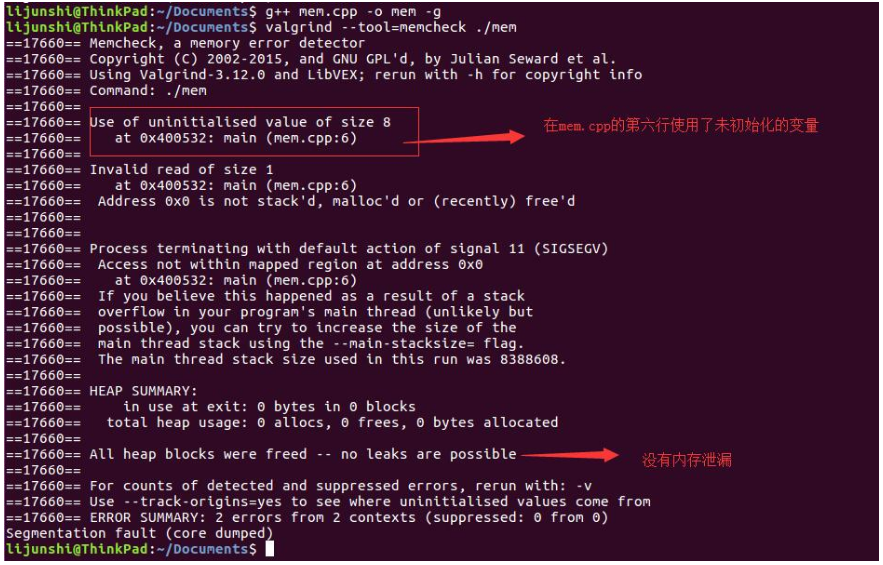
在linux系统下内存检测工具首推valgrind，valgrind是一个工具集，内存错误检测是它众多功能的一个，用的最多的功能是-----memcheck

该工具可以检测下列与内存相关的问题：未释放内存的使用；对释放后内存的读/写；对已分配内存块尾部的读/写；内存泄露；重复释放内存；不匹配的new/delete

定义了一个指针p,但未给它开辟空间，即是个野指针

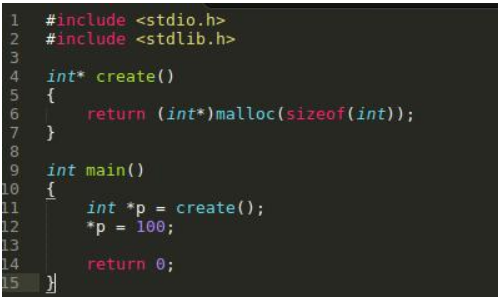


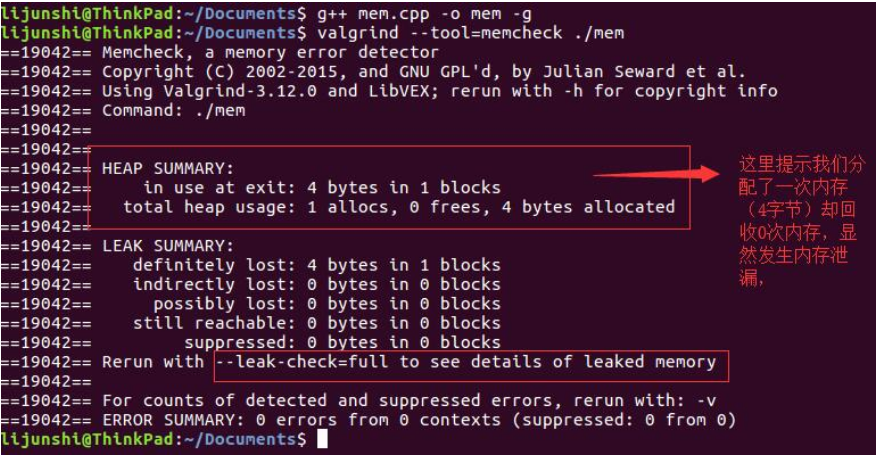
valgrind –tool=memcheck ./mem



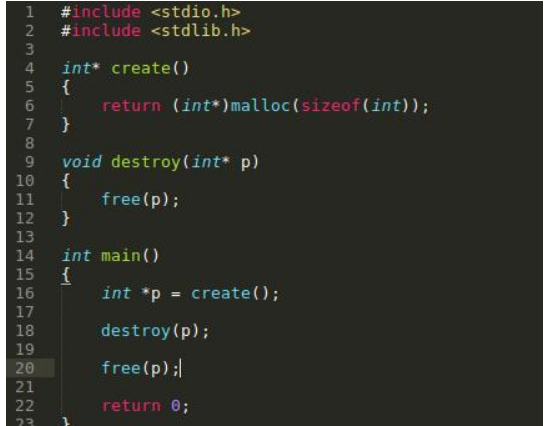
检测出使用了未初始化的变量，但未检测出内存泄露

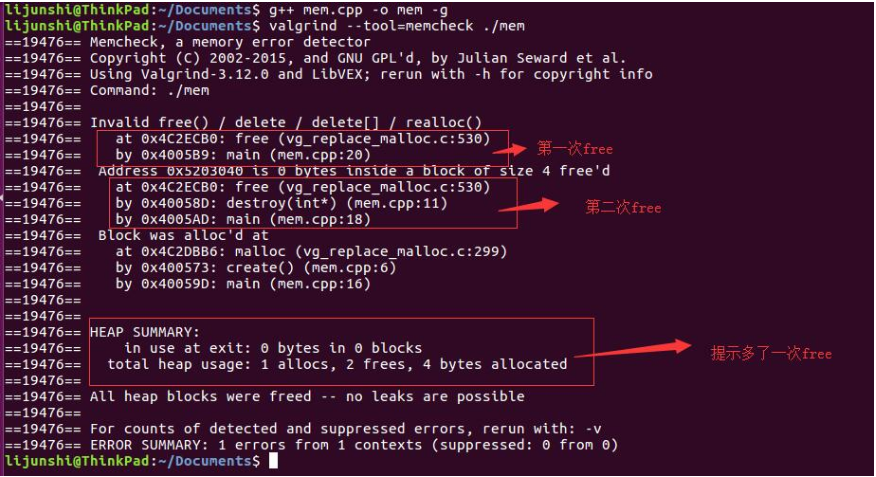
内存泄露：没有成对的使用malloc/free和new/delete





两次释放内存(double free)：





awk:相比于grep的查找，sed的编辑。awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。awk ‘{pattern + action}’ {filenames}

Shell 脚本问题：

向脚本传递参数：./script argument

在脚本中使用参数：$1:第一个传入的参数， $2:第二个参数…. ./copy.sh file1.txt /tmp/ cat copy.sh cp $1 $2 (把file1.txt复制到/tmp/目录下)

计算传递进脚本的参数个数：$#：传递到脚本的参数个数

脚本中获取脚本名称：$0:脚本文件名

打印传递给脚本的所有参数： echo $\*

输出当前shell的PID： echo $$

判断上一个命令是否运行成功：$? 显示前一个命令的退出状态。0表示没有错误，其他值表示有错误

if [ $? –ne 0]; then #或者 if [$? –eq 0]; then echo “succ”

echo “fail”

else

echo “sucess”

fi

tail: 查看文件的内容，有一个常用的参数-f，常用于查询正在改变的日志文件。tail -f filename会把filename文件里的最尾部的内容显示在屏幕上，并且不断刷新，只要filename更新就可以看到最新的文件内容。

获取文件的最后一行：tail -1 filename # tail filename输出文件最后几行内容

tail -f 循环读取 tail -n 100显示文件尾部的100行的内容 tail -n +5 filename显示文件第5行到末尾的内容

获取文件的第一行：head -1 filename # head filename 输出文件的前几行

获取一个文件每一行的第三个元素：awk ‘{print $3}’ filename

假设文件每行第一个元素是FIND，如何获取第二个元素：

awk ‘{if ($1 == “FIND”) print $2}’ filename

使用awk列出UID小于100的用户： awk –F: ‘$3<100’ /etc/passwd

调试bash脚本： #!/bin/bash –xv

写一个函数:

function func{ #python def func ….

echo “hello world!”

}

连接两个字符串：v1=”hello” v2=”world” v3=${v1}${v2} echo $v3 # helloworld

两个整数相加：v1=1 v2=2 let v3=$v1+$v2 echo $v3

其它方法：a=5 b=6 expr $a + $b echo $(($a + $b))

将整数a加给整数b并赋值给c

c=’expr $a + $b’

检查文件系统中是否存在某个文件:

if [ -f /var/log/messages ];then

echo “file exists”

fi

写出shell脚本中的循环语法：

for i in $(ls);do

echo item is $i

done

counter=0

while [ $counter –lt 10 ]; do

echo The counter is $counter

let counter=counter+1

done

输出数字0~100中3的倍数：

for (( i=0; i<=100; i=i+3)); do

echo “$i”

done

如何获取文本文件的第10行：head -10 filename | tail -1 #head -10表示获取前10行

获取文本文件的第10至第20行： head -20 filename | tail -10

如何在后台运行脚本，程序：在程序或者脚本之后添加&

chmod 500 script什么意思：r:4 w:2 x:1 使得脚本所有者可以读可以执行

单引号和双引号的区别：单引号不会将变量转换为值；双引号会计算变量的值并用值替代

如何用echo命令获取字符串变量的一部分：echo ${variable:x:y} x:起始位置 y:长度

var=”my name is peters, and I am developer.”

echo ${var:11:6} #显示peters

获取变量的最后5个字符：echo ${variable: -5}

获取变量的长度：${#variable}

计算本地用户数目：cat /etc/passwd | wc –l #统计行数

[ $a == $b ]和[ $a –eq $b ]有什么区别

[ $a == $b ] 用于字符串比较

[ $a –eq $b ] 用于数字比较

测试$a是否打印12的命令： if [ $a –gt 12 ];then … fi

测试$b是否小于12的命令： if [ $b -lt 12 ];then… fi

检查字符串是否以字母abc开头：if [[ &str == abc\* ]];then …. Fi

检查字符串是否完全等于abc： if [[ &str == abc ]]; then …. fi

在shell中定义数组：arr=(“hi” “my” “name” “is”)

打印数组的第一个元素：echo ${arr[0]}

打印数组所有元素：echo ${arr[@]} 或者 echo ${arr[\*]}

删除数组中索引为2的元素： unset arr[2]

在数组中添加id为333的元素： arr[333]=”new\_elem”

通过read命令，写入变量： read –p “dstport” destport

rpcinfo;

dtrace(性能之巅):

wireshark: .pcap

tcpdump:

iptable firewall