Linux操作系统基本命令的使用

这里列举了部分常用的Linux命令，用于Linux命令行中运行的涉及系统配置、文件（目录）等操作：

**（1）系统信息命令：**

**arch 显示机器的处理器架构(1)**

**uname -m 显示机器的处理器架构(2)**

**uname -r 显示正在使用的内核版本**

cat /proc/cpuinfo 显示CPU info的信息

cat /proc/version 显示内核的版本

lspci -tv 罗列 PCI 设备

lsusb -tv 显示 USB 设备

date 显示系统日期

cal 2007 显示2007年的日历表

date 041217002007.00 设置日期和时间 - 月日时分年.秒

clock –w 将时间修改保存到 BIOS

**（2）关机 (系统的关机、重启以及登出 )**

**shutdown -h now 关闭系统(1)**

**init 0 关闭系统(2)**

**telinit 0 关闭系统(3)**

shutdown -h hours:minutes & 按预定时间关闭系统

shutdown -c 取消按预定时间关闭系统

shutdown -r now 重启(1)

reboot 重启(2)

logout 注销

**（3）文件和目录**

cd /home 进入 '/ home' 目录'

cd .. 返回上一级目录

cd ../.. 返回上两级目录

cd 进入个人的主目录

cd ~user1 进入个人的主目录

cd - 返回上次所在的目录

pwd 显示工作路径

ls 查看目录中的文件

ls -F 查看目录中的文件

ls -l 显示文件和目录的详细资料

ls -a 显示隐藏文件

ls \*[0-9]\* 显示包含数字的文件名和目录名

mkdir dir1 创建一个叫做 'dir1' 的目录'

mkdir dir1 dir2 同时创建两个目录

mkdir -p /tmp/dir1/dir2 创建一个目录树

rm -f file1 删除一个叫做 'file1' 的文件'

rmdir dir1 删除一个叫做 'dir1' 的目录'

rm -rf dir1 删除一个叫做 'dir1' 的目录并同时删除其内容

rm -rf dir1 dir2 同时删除两个目录及它们的内容

mv dir1 new\_dir 重命名/移动 一个目录

cp file1 file2 复制一个文件

cp dir/\* . 复制一个目录下的所有文件到当前工作目录

cp -a /tmp/dir1 . 复制一个目录到当前工作目录

cp -a dir1 dir2 复制一个目录

ln -s file1 lnk1 创建一个指向文件或目录的软链接

ln file1 lnk1 创建一个指向文件或目录的物理链接

**(4) 文件搜索**

find / -name file1 从 '/' 开始进入根文件系统搜索文件和目录

find / -user user1 搜索属于用户 'user1' 的文件和目录

find /home/user1 -name \\*.bin 在目录 '/ home/user1' 中搜索带有'.bin'结尾的文件

find /usr/bin -type f -atime +100 搜索在过去100天内未被使用过的执行文件

find /usr/bin -type f -mtime -10 搜索在10天内被创建或者修改过的文件

find / -name \\*.rpm -exec chmod 755 '{}' \; 搜索以'.rpm' 结尾的文件并定义其权限

find / -xdev -name \\*.rpm 搜索以 '.rpm' 结尾文件，忽略光驱、捷盘等可移动设备

locate \\*.ps 寻找以 '.ps' 结尾的文件 - 先运行 'updatedb' 命令

whereis halt 显示一个二进制文件、源码或man的位置

which halt 显示一个二进制文件或可执行文件的完整路径

**(5) 挂载一个文件系统**

mount /dev/hda2 /mnt/hda2 挂载一个叫做hda2的盘-确定目录'/mnt/hda2'已经存在

umount /dev/hda2 卸载一个叫做hda2的盘-先从挂载点'/ mnt/hda2'退出

fuser -km /mnt/hda2 当设备繁忙时强制卸载

umount -n /mnt/hda2 运行卸载操作而不写入 /etc/mtab 文件- 当文件为只读或当磁盘写满时非常有用

mount /dev/fd0 /mnt/floppy 挂载一个软盘

mount /dev/cdrom /mnt/cdrom 挂载一个cdrom或dvdrom

mount /dev/hdc /mnt/cdrecorder 挂载一个cdrw或dvdrom

mount /dev/hdb /mnt/cdrecorder 挂载一个cdrw或dvdrom

mount -o loop file.iso /mnt/cdrom 挂载一个文件或ISO镜像文件

mount -t vfat /dev/hda5 /mnt/hda5 挂载一个Windows FAT32文件系统

mount /dev/sda1 /mnt/usbdisk 挂载一个usb 捷盘或闪存设备

**(6) 磁盘空间**

df -h 显示已经挂载的分区列表

ls -lSr |more 以尺寸大小排列文件和目录

du -sh dir1 估算目录 'dir1' 已经使用的磁盘空间'

du -sk \* | sort -rn 以容量大小为依据依次显示文件和目录的大小

**(7) 用户和群组**

groupadd group\_name 创建一个新用户组

groupdel group\_name 删除一个用户组

groupmod -n new\_group\_name old\_group\_name 重命名一个用户组

useradd -c "Name Surname " -g admin -d /home/user1 -s /bin/bash user1 创建一个属于 "admin" 用户组的用户

useradd user1 创建一个新用户

userdel -r user1 删除一个用户 ( '-r' 排除主目录)

usermod -c "User FTP" -g system -d /ftp/user1 -s /bin/nologin user1 修改用户属性

passwd 修改口令

passwd user1 修改一个用户的口令 (只允许root执行)

chage -E 2005-12-31 user1 设置用户口令的失效期限

pwck 检查 '/etc/passwd' 的文件格式和语法修正以及存在的用户

grpck 检查 '/etc/passwd' 的文件格式和语法修正以及存在的群组

newgrp group\_name 登陆进一个新的群组以改变新创建文件的预设群组

**(8) 文件的权限**(使用 "+" 设置权限，使用 "-" 用于取消)

ls -lh 显示权限

ls /tmp | pr -T5 -W$COLUMNS 将终端划分成5栏显示

chmod ugo+rwx directory1 设置目录的所有人(u)、群组(g)以及其他人(o)以读（r ）、写(w)和执行(x)的权限

chmod go-rwx directory1 删除群组(g)与其他人(o)对目录的读写执行权限

chown user1 file1 改变一个文件的所有人属性

chown -R user1 directory1 改变一个目录的所有人属性并同时改变改目录下所有文件的属性

chgrp group1 file1 改变文件的群组

chown user1:group1 file1 改变一个文件的所有人和群组属性

chmod o-t /home/public 禁用一个目录的 STIKY 位

**(9) 文件的特殊属性**(使用 "+" 设置权限，使用 "-" 用于取消)

chattr +a file1 只允许以追加方式读写文件

chattr +c file1 允许这个文件能被内核自动压缩/解压

chattr +d file1 在进行文件系统备份时，dump程序将忽略这个文件

chattr +i file1 设置成不可变的文件，不能被删除、修改、重命名或者链接

chattr +s file1 允许一个文件被安全地删除

chattr +S file1 一旦应用程序对这个文件执行了写操作，使系统立刻把修改的结果写到磁盘

chattr +u file1 若文件被删除，系统会允许你在以后恢复这个被删除的文件

lsattr 显示特殊的属性

**(10) 打包和压缩文件**

bunzip2 file1.bz2 解压一个叫做 'file1.bz2'的文件

bzip2 file1 压缩一个叫做 'file1' 的文件

gunzip file1.gz 解压一个叫做 'file1.gz'的文件

gzip file1 压缩一个叫做 'file1'的文件

gzip -9 file1 最大程度压缩

rar a file1.rar test\_file 创建一个叫做 'file1.rar' 的包

rar a file1.rar file1 file2 dir1 同时压缩 'file1', 'file2' 以及目录 'dir1'

rar x file1.rar 解压rar包

unrar x file1.rar 解压rar包

tar -cvf archive.tar file1 创建一个非压缩的 tarball

tar -cvf archive.tar file1 file2 dir1 创建包含了'file1', 'file2'以及'dir1'的档案文件

tar -tf archive.tar 显示一个包中的内容

tar -xvf archive.tar 释放一个包

tar -xvf archive.tar -C /tmp 将压缩包释放到 /tmp目录下

tar -cvfj archive.tar.bz2 dir1 创建一个bzip2格式的压缩包

tar -xvfj archive.tar.bz2 解压一个bzip2格式的压缩包

tar -cvfz archive.tar.gz dir1 创建一个gzip格式的压缩包

tar -xvfz archive.tar.gz 解压一个gzip格式的压缩包

zip file1.zip file1 创建一个zip格式的压缩包

zip -r file1.zip file1 file2 dir1 将几个文件和目录同时压缩成一个zip格式的压缩包

unzip file1.zip 解压一个zip格式压缩包

**(11) RPM包**（Fedora, Redhat及类似系统）

rpm -ivh package.rpm 安装一个rpm包

rpm -ivh --nodeeps package.rpm 安装一个rpm包而忽略依赖关系警告

rpm -U package.rpm 更新一个rpm包但不改变其配置文件

rpm -F package.rpm 更新一个确定已经安装的rpm包

rpm -e package\_name.rpm 删除一个rpm包

rpm -qa 显示系统中所有已经安装的rpm包

rpm -qa | grep httpd 显示所有名称中包含 "httpd" 字样的rpm包

rpm -qi package\_name 获取一个已安装包的特殊信息

rpm -qg "System Environment/Daemons" 显示一个组件的rpm包

rpm -ql package\_name 显示一个已经安装的rpm包提供的文件列表

rpm -qc package\_name 显示一个已经安装的rpm包提供的配置文件列表

rpm -q package\_name --whatrequires 显示与一个rpm包存在依赖关系的列表

rpm -q package\_name --whatprovides 显示一个rpm包所占的体积

rpm -q package\_name --scripts 显示在安装/删除期间所执行的脚本l

rpm -q package\_name --changelog 显示一个rpm包的修改历史

rpm -qf /etc/httpd/conf/httpd.conf 确认所给的文件由哪个rpm包所提供

rpm -qp package.rpm -l 显示由一个尚未安装的rpm包提供的文件列表

rpm --import /media/cdrom/RPM-GPG-KEY 导入公钥数字证书

rpm --checksig package.rpm 确认一个rpm包的完整性

rpm -qa gpg-pubkey 确认已安装的所有rpm包的完整性

rpm -V package\_name 检查文件尺寸、 许可、类型、所有者、群组、MD5检查以及最后修改时间

rpm -Va 检查系统中所有已安装的rpm包- 小心使用

rpm -Vp package.rpm 确认一个rpm包还未安装

rpm2cpio package.rpm | cpio --extract --make-directories \*bin\* 从一个rpm包运行可执行文件

rpm -ivh /usr/src/redhat/RPMS/`arch`/package.rpm 从一个rpm源码安装一个构建好的包

rpmbuild --rebuild package\_name.src.rpm 从一个rpm源码构建一个 rpm 包

**(12) 查看文件内容**

cat file1 从第一个字节开始正向查看文件的内容

tac file1 从最后一行开始反向查看一个文件的内容

more file1 查看一个长文件的内容

less file1 类似于 'more' 命令，但是它允许在文件中和正向操作一样的反向操作

head -2 file1 查看一个文件的前两行

tail -2 file1 查看一个文件的最后两行

tail -f /var/log/messages 实时查看被添加到一个文件中的内容

**(13)文本处理**

grep Aug /var/log/messages 在文件 '/var/log/messages'中查找关键词"Aug"

grep ^Aug /var/log/messages 在文件'/var/log/messages'中查找以"Aug"开始的词汇

grep [0-9] /var/log/messages 选择 '/var/log/messages' 文件中所有包含数字的行

grep Aug -R /var/log/\* 在目录 '/var/log' 及随后的目录中搜索字符串"Aug"

sort file1 file2 排序两个文件的内容

sort file1 file2 | uniq 取出两个文件的并集(重复的行只保留一份)

sort file1 file2 | uniq -u 删除交集，留下其他的行

sort file1 file2 | uniq -d 取出两个文件的交集(只留下同时存在于两个文件中的文件)

**(14) 字符设置和文件格式转换**

dos2unix filedos.txt fileunix.txt 将一个文本文件的格式从MSDOS转换成UNIX

unix2dos fileunix.txt filedos.txt 将一个文本文件的格式从UNIX转换成MSDOS

**(15) 文件系统分析**

badblocks -v /dev/hda1 检查磁盘hda1上的坏磁块

fsck /dev/hda1 修复/检查hda1磁盘上linux文件系统的完整性

fsck.ext2 /dev/hda1 修复/检查hda1磁盘上ext2文件系统的完整性

e2fsck /dev/hda1 修复/检查hda1磁盘上ext2文件系统的完整性

e2fsck -j /dev/hda1 修复/检查hda1磁盘上ext3文件系统的完整性

fsck.ext3 /dev/hda1 修复/检查hda1磁盘上ext3文件系统的完整性

fsck.vfat /dev/hda1 修复/检查hda1磁盘上fat文件系统的完整性

fsck.msdos /dev/hda1 修复/检查hda1磁盘上dos文件系统的完整性

dosfsck /dev/hda1 修复/检查hda1磁盘上dos文件系统的完整性

**(16) 初始化一个文件系统**

mkfs /dev/hda1 在hda1分区创建一个文件系统

mke2fs /dev/hda1 在hda1分区创建一个linux ext2的文件系统

mke2fs -j /dev/hda1 在hda1分区创建一个linux ext3(日志型)的文件系统

mkfs -t vfat 32 -F /dev/hda1 创建一个 FAT32 文件系统

fdformat -n /dev/fd0 格式化一个软盘

mkswap /dev/hda3 创建一个swap文件系统

**(17) SWAP文件系统**

mkswap /dev/hda3 创建一个swap文件系统

swapon /dev/hda3 启用一个新的swap文件系统

swapon /dev/hda2 /dev/hdb3 启用两个swap分区

**(18) 备份**

dump -0aj -f /tmp/home0.bak /home 制作一个 '/home' 目录的完整备份

dump -1aj -f /tmp/home0.bak /home 制作一个 '/home' 目录的交互式备份

restore -if /tmp/home0.bak 还原一个交互式备份

dd if=/dev/sda of=/tmp/file1 备份磁盘内容到一个文件

dd if=/dev/hda of=/dev/fd0 bs=512 count=1 做一个将 MBR (Master Boot Record)内容复制到软盘的动作

dd if=/dev/fd0 of=/dev/hda bs=512 count=1 从已经保存到软盘的备份中恢复MBR内容

**(19) 光盘**

cdrecord -v gracetime=2 dev=/dev/cdrom -eject blank=fast -force 清空一个可复写的光盘内容

mkisofs /dev/cdrom > cd.iso 在磁盘上创建一个光盘的iso镜像文件

mount -o loop cd.iso /mnt/iso 挂载一个ISO镜像文件

**(20) 网络**（以太网和WIFI无线）

ifconfig eth0 显示一个以太网卡的配置

ifup eth0 启用一个 'eth0' 网络设备

ifdown eth0 禁用一个 'eth0' 网络设备

ifconfig eth0 192.168.1.1 netmask 255.255.255.0 控制IP地址

ifconfig eth0 promisc 设置 'eth0' 成混杂模式以嗅探数据包 (sniffing)

dhclient eth0 以dhcp模式启用 'eth0'

route -n show routing table

route add -net 0/0 gw IP\_Gateway configura default gateway

route add -net 192.168.0.0 netmask 255.255.0.0 gw 192.168.1.1 configure static route to reach network '192.168.0.0/16'

route del 0/0 gw IP\_gateway remove static route

echo "1" > /proc/sys/net/ipv4/ip\_forward activate ip routing

hostname show hostname of system

**4. Linux程序设计及调试**

**（1） C程序的编译与运行**

在windows下编辑一个名为HelloWorld.c程序如下（注意Linux下的文件名的字符大小写是不相同的）：

#include <stdio.h>

main()

{

printf(“Hello World!\n”;

}

编译该程序： gcc –o HelloWorld HelloWord.c

运行编译等后的执行文件（HelloWorld）: **./HelloWorld**

屏幕显示：　　　Ｈello World!

**(2) C程序的GDB调试**

I 编辑源程序（注意下面源代码中的空行也保留）：tst.c

#include <stdio.h>

int func(int n)

{

int sum=0,i;

for(i=0; i<n; i++)

{

sum+=i;

}

return sum;

}

main()

{

int i;

long result = 0;

for(i=1; i<=100; i++)

{

result += i;

}

printf("result[1-100] = %d \n", result );

printf("result[1-250] = %d \n", func(250) );

}

II 编译生成执行文件：[root@localhost ~]# **gcc -g tst.c -o tst**

（注意必须有 –g 参数，否则执行代码无调试信息）

III 使用GDB调试：

[root@localhost ~]# **gdb tst** <---------- 启动GDB

GNU gdb Red Hat Linux (6.1post-1.20040607.41rh)

Copyright 2004 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are

welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread\_db library "/lib/tls/libthread\_db.so.1".

(gdb)

这时已处于GDB的调试环境，（gdb）提示符后可以用输入调试命令，关于gd b调试命令，可在网上搜索，自行测试，本实验只对部分命令进行验证：

**l**  **1** <-------------------- 字母**l**是命令（list的缩写），后面数字1从第一行开始例出原码。

1 #include <stdio.h>

2

3 int func(int n)

4 {

5 int sum=0,i;

6 for(i=1; i<=n; i++)

7 {

8 sum+=i;

9 }

10 return sum;

(gdb) **l** <-------------------- 这里命令l表示继续显示下面源代码

11

12

13

14 main()

15 {

16 int i;

17 long result = 0;

18 for(i=1; i<=100; i++)

19 {

20 result += i;

(gdb) <--------------------这里回车表示重复上面命令，继续显示后面代码

21 }

22

23 printf("result[1-100] = %d \n", result );

24 printf("result[1-250] = %d \n", func(250) );

25 }

26

(gdb) **break 16** <-------------------- 设置断点，在源程序第16行处（break 也可简写为b）。

Breakpoint 1 at 0x8048496: file tst.c, line 16.

(gdb) **break func** <-------------------- 设置断点，在函数func()入口处。

Breakpoint 2 at 0x8048456: file tst.c, line 5.

(gdb) **info break** <-------------------- 查看断点信息。

Num Type Disp Enb Address What

1 breakpoint keep y 0x08048496 in main at tst.c:16

2 breakpoint keep y 0x08048456 in func at tst.c:5

(gdb) **r**  <--------------------- 运行程序，run命令简写

Starting program: /root/tst

Breakpoint 1, main () at tst.c:17 <---------- 在断点处停住。

17 long result = 0;

(gdb) **n** <--------------------- 单条语句执行，next命令简写。

18 for(i=1; i<=100; i++)

(gdb) **n**

20 result += i;

(gdb) **n**

18 for(i=1; i<=100; i++)

(gdb) **n**

20 result += i;

(gdb) **c**  <--------------------- 继续运行程序，continue命令简写。

Continuing.

result[1-100] = 5050 <----------程序输出。

<----------程序输出回车换行(程序中的“**\n**”)

Breakpoint 2, func (n=250) at tst.c:5

5 int sum=0,i;

(gdb) **n**

6 for(i=1; i<=n; i++)

(gdb) **p i** <--------------------- 打印变量i的值，print命令简写。

$1 = 12398580 <--------------变量i的当前值是随机数

(gdb) **n**

8 sum+=i;

(gdb) **n**

6 for(i=1; i<=n; i++)

(gdb) **p sum**

$2 = 1

(gdb) **n**

8 sum+=i;

(gdb) **p i**

$3 = 2

(gdb) **n**

6 for(i=1; i<=n; i++)

(gdb) **p sum**

$4 = 3

(gdb)**b t** <--------------------- 查看函数堆栈。

#0 func (n=250) at tst.c:5

#1 0x080484e4 in main () at tst.c:24

#2 0x400409ed in \_\_libc\_start\_main () from /lib/libc.so.6

(gdb) **finish** <--------------------- 退出函数。

Run till exit from #0 func (n=250) at tst.c:5

0x080484e4 in main () at tst.c:24

24 printf("result[1-250] = %d n", func(250) );

Value returned is $6 = 31375

(gdb) **c** <--------------------- 继续运行。

Continuing.

result[1-250] = 31375 <----------程序输出。

<----------程序输出回车换行(程序中的“**\n**”)

Program exited with code 026. <--------程序退出，调试结束。

(gdb) **q**  <--------------------- 退出gdb。

[root@localhost ~]#

**(3) C程序的汇编级调试**

[root@localhost ~]# **gcc -o tst tst.c**  <--------产生的执行文件没有调试信息

[root@localhost ~]# **gdb tst**

GNU gdb Red Hat Linux (6.1post-1.20040607.41rh)

Copyright 2004 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are

welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "i386-redhat-linux-gnu"...(no debugging symbols found)...Using host libthread\_db library "/lib/tls/libthread\_db.so.1".

(gdb) **l**  <---------执行l命令，将看见下面的信息无法列出文件内容

No symbol table is loaded. Use the "file" command.

(gdb)**b main** <---------设置断点在main函数入口

Breakpoint 1 at 0x804839e

(gdb) **b func**  <---------设置断点在func函数入口

Breakpoint 2 at 0x804836e

(gdb) **disas main** <---------从main函数开始反汇编

Dump of assembler code for function main:

0x08048398 <main+0>: push %ebp

0x08048399 <main+1>: mov %esp,%ebp

0x0804839b <main+3>: sub $0x8,%esp

0x0804839e <main+6>: and $0xfffffff0,%esp

0x080483a1 <main+9>: mov $0x0,%eax

0x080483a6 <main+14>: add $0xf,%eax

0x080483a9 <main+17>: add $0xf,%eax

0x080483ac <main+20>: shr $0x4,%eax

0x080483af <main+23>: shl $0x4,%eax

0x080483b2 <main+26>: sub %eax,%esp

0x080483b4 <main+28>: movl $0x0,0xfffffff8(%ebp)

0x080483bb <main+35>: movl $0x1,0xfffffffc(%ebp)

0x080483c2 <main+42>: cmpl $0x64,0xfffffffc(%ebp)

0x080483c6 <main+46>: jg 0x80483d7 <main+63>

0x080483c8 <main+48>: mov 0xfffffffc(%ebp),%eax

0x080483cb <main+51>: lea 0xfffffff8(%ebp),%edx

0x080483ce <main+54>: add %eax,(%edx)

0x080483d0 <main+56>: lea 0xfffffffc(%ebp),%eax

0x080483d3 <main+59>: incl (%eax)

0x080483d5 <main+61>: jmp 0x80483c2 <main+42>

0x080483d7 <main+63>: sub $0x8,%esp

0x080483da <main+66>: pushl 0xfffffff8(%ebp)

0x080483dd <main+69>: push $0x80484ec

0x080483e2 <main+74>: call 0x80482b0 <\_init+56>

0x080483e7 <main+79>: add $0x10,%esp

0x080483ea <main+82>: sub $0x8,%esp

0x080483ed <main+85>: push $0xfa

0x080483f2 <main+90>: call 0x8048368 <func>

0x080483f7 <main+95>: add $0x4,%esp

0x080483fa <main+98>: push %eax

0x080483fb <main+99>: push $0x8048500

0x08048400 <main+104>: call 0x80482b0 <\_init+56>

0x08048405 <main+109>: add $0x10,%esp

0x08048408 <main+112>: leave

0x08048409 <main+113>: ret

0x0804840a <main+114>: nop

0x0804840b <main+115>: nop

---Type <return> to continue, or q <return> to quit---

(gdb) **disas func** <---------从func函数开始反汇编

Dump of assembler code for function func:

0x08048368 <func+0>: push %ebp

0x08048369 <func+1>: mov %esp,%ebp

0x0804836b <func+3>: sub $0x8,%esp

0x0804836e <func+6>: movl $0x0,0xfffffffc(%ebp)

0x08048375 <func+13>: movl $0x1,0xfffffff8(%ebp)

0x0804837c <func+20>: mov 0xfffffff8(%ebp),%eax

0x0804837f <func+23>: cmp 0x8(%ebp),%eax

0x08048382 <func+26>: jg 0x8048393 <func+43>

0x08048384 <func+28>: mov 0xfffffff8(%ebp),%eax

0x08048387 <func+31>: lea 0xfffffffc(%ebp),%edx

0x0804838a <func+34>: add %eax,(%edx)

0x0804838c <func+36>: lea 0xfffffff8(%ebp),%eax

0x0804838f <func+39>: incl (%eax)

0x08048391 <func+41>: jmp 0x804837c <func+20>

0x08048393 <func+43>: mov 0xfffffffc(%ebp),%eax

0x08048396 <func+46>: leave

0x08048397 <func+47>: ret

End of assembler dump.

End of assembler dump.

(gdb) **r** <---------运行程序将停在第一个断点处

Starting program: /root/tst

(no debugging symbols found)...(no debugging symbols found)...

Breakpoint 1, 0x0804839e in main ()

(gdb) **n** <---------无法单步执行，如下可见直接运行到第二个断点位置

Single stepping until exit from function main,

which has no line number information.

result[1-100] = 5050

Breakpoint 2, 0x0804836e in func ()

(gdb) **ni**  <---------执行下一条指令（ni: next instruction）

0x08048375 in func ()

(gdb) **ni**

0x0804837c in func ()

(gdb) **si**  <---------单步执行下一条指令（si: step instruction）

0x0804837f in func ()

(gdb) **display /x $eax** <---------每条指令执行后自动以十六进制显示寄存器eax的内容

1: /x $eax = 0x1

(gdb) **display /x $ebx** <---------每条指令执行后自动以十六进制显示寄存器eax的内容

2: /x $ebx = 0xbd2ff4

(gdb) **ni** <--------- 执行一条指令后自动显示寄存器eax和ebx的内容

0x08048382 in func ()

2: /x $ebx = 0xbd2ff4

1: /x $eax = 0x1

(gdb) **disp /x \*0x8048375** <------自动显示地址0x8048375内存单元的内容

4: /x \*134513525 = 0x1f845c7

(gdb) **ni** <------一条指令执行后自动显示内存、寄存器的内容

0x08048384 in func ()

4: /x \*134513525 = 0x1f845c7

2: /x $ebx = 0xbd2ff4

1: /x $eax = 0x1

**(4) C嵌入汇编代码**

输入源代码：

int main()

{

int input, output,temp;

input = 1;

\_\_asm\_\_ \_\_volatile\_\_ ("movl $0, %%eax;\n\t"

"movl %%eax, %1;\n\t"

"movl %2, %%eax;\n\t"

"movl %%eax, %0;\n\t"

:"=m"(output),"=m"(temp) /\* output \*/

:"r"(input) /\* input \*/

);

return 0;

}

编译成含调试信息的执行文件：

[root@localhost ~]# **gcc -g -o ts ts.c**

[root@localhost ~]# **gdb ts**

GNU gdb Red Hat Linux (6.1post-1.20040607.41rh)

Copyright 2004 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are

welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread\_db library "/lib/tls/libthread\_db.so.1".

(gdb) **list 1**

1 int main()

2 {

3 int input, output,temp;

4 input = 1;

5 \_\_asm\_\_ \_\_volatile\_\_ ("movl $0, %%eax;\n\t"

6 "movl %%eax, %1;\n\t"

7 "movl %2, %%eax;\n\t"

8 "movl %%eax, %0;\n\t"

9 :"=m"(output),"=m"(temp) /\* output \*/

10 :"r"(input) /\* input \*/

(gdb)

11 );

12 return 0;

13 }

(gdb)

Line number 14 out of range; ts.c has 13 lines.

(gdb) **disas main**

Dump of assembler code for function main:

0x08048334 <main+0>: push %ebp

0x08048335 <main+1>: mov %esp,%ebp

0x08048337 <main+3>: sub $0x18,%esp

0x0804833a <main+6>: and $0xfffffff0,%esp

0x0804833d <main+9>: mov $0x0,%eax

0x08048342 <main+14>: add $0xf,%eax

0x08048345 <main+17>: add $0xf,%eax

0x08048348 <main+20>: shr $0x4,%eax

0x0804834b <main+23>: shl $0x4,%eax

0x0804834e <main+26>: sub %eax,%esp

0x08048350 <main+28>: movl $0x1,0xfffffffc(%ebp)

0x08048357 <main+35>: mov 0xfffffffc(%ebp),%eax

0x0804835a <main+38>: mov $0x0,%eax

0x0804835f <main+43>: mov %eax,0xfffffff4(%ebp)

0x08048362 <main+46>: mov %eax,%eax

0x08048364 <main+48>: mov %eax,0xfffffff8(%ebp)

0x08048367 <main+51>: mov $0x0,%eax

0x0804836c <main+56>: leave

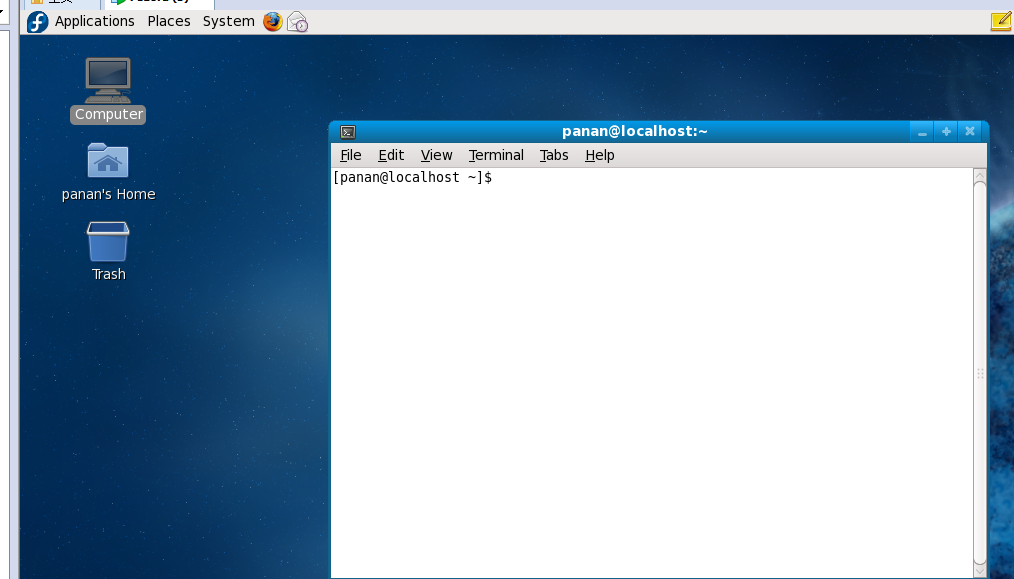
0x0804836d <main+57>: ret

End of assembler dump.

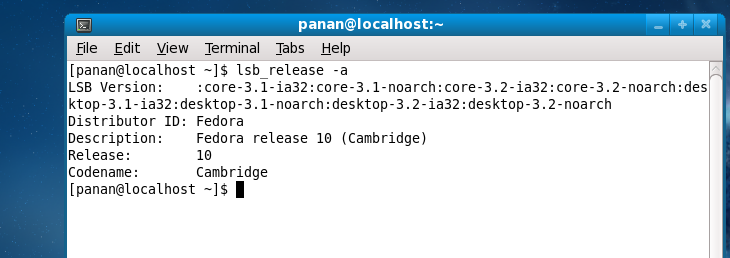
(gdb)

**五.实验截图**

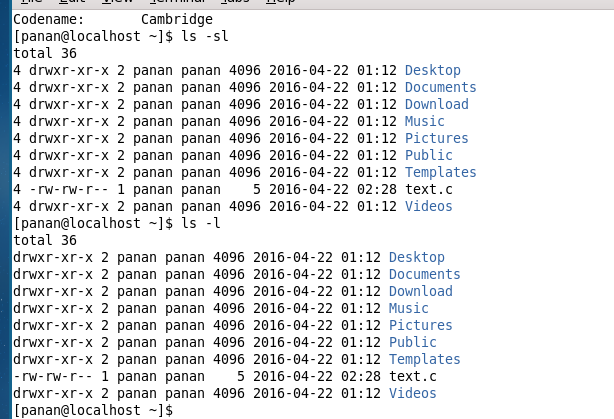
**系统信息： 运行图：**



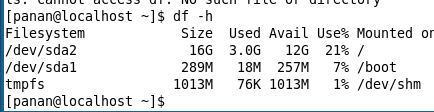
**系统版本：**



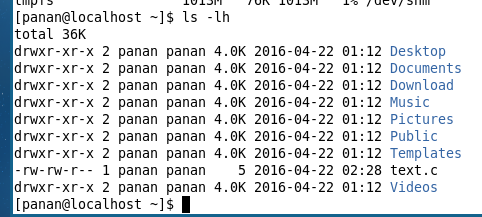
**文件目录操作：**



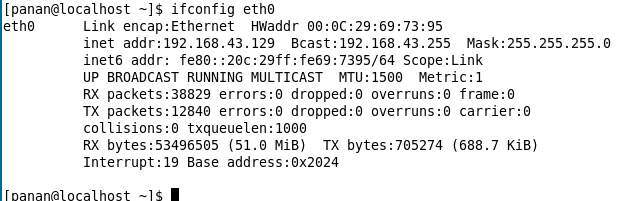
**磁盘空间:**



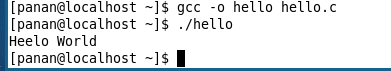
**文件权限：**



**以太网配置：**



**程序调试：运行Hello World**



**实验总结：**

这个实验总的来说还是比较简单的，基础的虚拟机的安装以及一些验证型的命令。但在做实验的过程中还是有一些错误。一开始安装的时候镜像文件出错导致安装中途死机。在用VI编辑C语言程序的时候，因为编辑器不会自动检测错误，然后很多不好的习惯就会暴露出来，包括简单的符号问题以及少冒号之类的低级错误。但最终还是完成了实验。第一个实验有很多入门的知识，学到很多。