

---

# 操作系统

# 实验指导手册

赵霞

北京工商大学

计算机与信息工程学院

2012-9

## 目录

前 言.....	3
实验 1: Linux 的安装与基本操作 .....	4
目的与要求.....	4
1. 用 Ubuntu 11.04 的光盘安装系统.....	4
2. 操作系统启动模式.....	4
3. 用户管理.....	5
4. 进程管理.....	7
5. 文件系统命令.....	7
6. vim 编辑器 .....	9
7. 使用 U 盘.....	11
参考命令: .....	11
要求: .....	11
8. 基本网络管理命令.....	12
9. 其他网络管理命令.....	12
10. SSH 服务与客户端.....	13
11. 网络服务器配置 .....	13
实验 2: shell 编程和 GCC 编程环境.....	15
目的.....	15
1. shell 编程 .....	15
2. GCC 编程环境 .....	16
3. GDB 调试 .....	18
实验 3: 编程与调试: 进程管理.....	22
目的.....	22
1. 练习 1.....	22
2. 练习 2.....	23
3. 练习 3.....	24
4. 练习 4.....	25
5. 练习 5.....	26
实验 4: 编程与调试: 内存管理.....	28
目的.....	28
1. 练习 1.....	28
2. 练习 2.....	29
3. 练习 3.....	30
实验 5: 编程与调试: 文件操作.....	31
目的.....	31
1. 练习 1 .....	31

2. 练习 2.....	33
3. 练习 3.....	36
实验 6：编程与调试：网络通信.....	41
目的.....	41
1. 练习 1.....	41
2. 练习 2.....	42
3. 练习 3.....	44
4. 练习 4.....	46
实验 7：编程与调试：线程编程.....	48
目的.....	48
1. 练习 1.....	48
2. 练习 2.....	49
综合实验.....	56
1. 题目： .....	56
2. 目的.....	56
3. 实验要求与评价.....	56
4. 实验内容及学时安排.....	57

# 前 言

本实验指导手册供学习《操作系统》课程的学生和教师使用。

实验内容包括：独立实验共 7 个，综合实验 1 个。实验 1 偏重使用和操作，实验 2-7 偏重编程和开发，由教师根据课时安排和学生的实际情况选择。学生也可以根据自己的情况和兴趣在课外学习和研究使用参考。

给读者的建议：

- 1、 **对于 Linux 操作系统的初学者**，配合各类 Linux 操作系统使用类的书籍使用，建议到图书馆里借阅适合自己的相关参考书。
- 2、 **对于有 Linux 基础的读者**，可以参考综合实验，从实验 2 做起，围绕开发目的进行实验课的学习和实践，最终提交有一定功能的软件，锻炼综合的开发能力。
- 3、 **Internet 是很好的老师**。如果有上网条件，建议读者遇到问题后，直接利用 Google 上网搜索问题的答案，经过自己的消化理解和思考尝试，解决问题。在本课程的学习过程中提高发现问题、解决问题的能力。
- 4、 **做每个操作之前，确定自己的目标和理由**。每次实验过程中，以“我想做....，我为什么要这样做？”开头，明确自己每个操作的目的，思考系统给出的显示结果的含义，从而发现问题，不断学习新知识。
- 5、 **通过总结巩固和提高能力，与他人共享**。每次实验后，用简洁清晰的文字，记下来目标、你的解决方法（使用的命令）及结果、遇到的问题、解决的方法、从网上收集的相关资料，汇总整理为自己的《工作笔记》或者放到自己的 blog 上，选用合适的内容撰写实验报告。**希望每个人能够在实验中有收获，在实验报告中展示自己的特色。**

编者：赵 霞

# 实验 1: Linux 的安装与基本操作

## 目的与要求

1. 了解 linux 操作系统的安装和启动过程;
2. 了解 Linux 文件的组织结构;
3. 熟悉 X Window 环境。
4. 设计每个命令和功能的命令序列及参数, 记录实验结果, 分析遇到的问题和解决方法。

## 1. 用 Ubuntu 11.04 的光盘安装系统

- 在 VmWare 软件中, 创建 Linux 虚拟机, 设置:
  - 1) 光驱来源为 Ubuntu 11.04 映像文件
  - 2) 网卡类型为 NAT 类型
  - 3) 硬盘大小为 10G, 内存为 512M (如果机器的内存大于 2G, 则设置为 800M)。
    - 启动虚拟机, 从光驱启动操作系统安装过程
    - 按照提示, 逐步安装操作系统
    - 虚拟机和 Windows 之间的切换用 Alt+Ctrl
    - 查看 Linux 操作系统目录结构, 特别是系统配置目录和源代码所在的目录及文件;
    - 启动 X Window, 熟悉窗口界面的环境。

## 2. 操作系统启动模式

Linux 使用 grub 启动工具引导操作系统。grub 启动工具的基本功能是引导多种操作系统, 用户可以在操作系统启动之前选择引导不同的操作系统内核, 或者为内核指定不同的引导参数。

早期流行的 Red Hat Linux 操作系统有 6 种启动模式, 也叫运行级别, 给用户提供使用系统的不同方式。运行级定义在 Red Hat Linux 里面的 /etc/inittab 文件里。

- 0 - 停机, 机器关闭 (一般不把 initdefault 设置为 0 )
- 1 - 单用户模式, 类似 Win9x 下的安全模式
- 2 - 多用户, 但是没有 NFS
- 3 - 字符模式, 完全多用户模式, 标准的运行级
- 4 - 没有用到, 可以按需定制

5 - X11 , X Window 图形界面, 完全多用户模式

6 - 重新启动 (不要把 `initdefault` 设置为 6 )

Ubuntu 从 Debian 发行版精挑细选的软件包, 同时保留了 Debian 强大的软件包管理系统, 以便简易的安装或彻底的删除程序。因此对启动模式的管理方法与 Debian 相同。其中 0, 1, 6 运行级别没有变, 从 2 到 5 都变成了和 Red Hat Linux 的 5 运行级别相同的图形界面模式。

其中, 最特殊的是运行级别 1, 类似于 Windows 里面的安全模式, 主要用于修复系统。除了只启动必须的操作系统服务之外, 这个模式最大的特点在于默认帐号是超级用户, 并且不需要输入密码。因为超级用户具有对全部系统的访问权限, 如果被非法用户利用, 则系统无任何安全保证。对于需要保密和安全的系统, 撤掉作为控制终端的键盘和显示器, 使得未授权用户不可交互式启动系统, 即可避免这一安全隐患。

如果用户的密码丢失, 可以通过在 grub 启动过程中, 手动进入运行级别 1, 重置用户密码, 并进入系统。

具体步骤如下:

- 重启系统, 在 BIOS 自检之后, 马上按 shift 键不放, 就进入 grub 菜单, 显示诸如下面的内容  
Ubuntu, Linux 2.6.38-8-generic  
Ubuntu, Linux 2.6.38-8-generic (recovery mode)  
Memory test (memtest86+)  
Memory test (memtest86+, serial console 115200)
- 选择有(recovery mode)标识这行, 按”e”进入编辑模式  
Linux /vmlinuz-2.6.38-8-generic root=UUID=f0077127\*3e53 ro single
- 把 ro single 改成 rw single  
linux /vmlinuz-2.6.31-14-generic root=UUID=f0077127\*3e53 rw single
- 按 ctrl+x 或者 F10 引导, 选择进入具有 root 权限的单用户模式  
root#

在单用户模式下的 root 用户, 具有超级用户的所有权限, 例如修改密码等。此时除了必要的操作, 尽量不要修改系统文件。

- 用 passwd 命令修改 root 口令  
在命令提示符后面输入 **passwd**, 根据提示输入密码即可。
- 执行 pwd, ls 等命令, 理解操作用户的访问权限

### 3. 用户管理

- 普通用户和超级用户

- 以超级用户身份执行特权命令：sudo 命令名，需要输入密码
- Passwd, shadow, group 文件的内容

### 要求：

- 1) 建立名为自己的班号的组（例如，如果你是 091 班的，用户名为“cs091”），在该组下为 3 个同学建立新用户名，设置密码可用时间为半年，登录 shell 为你的系统中支持的各种不同的 shell，如 bsh, csh, bash 等。
- 2) 移动到在你自己姓名的用户目录中，建立一个 lab1 目录。把你第一次实验中建立的 hello.c 文件及其相关文件。
- 3) 在 lab1 目录下，把可执行的 hello 文件访问权限改为不可执行。
- 4) 将每一步执行的命令和结果都写入实验报告中，分析遇到的问题和解决方法。

### 参考命令：

- 查看用户管理文件：/etc/passwd, /etc/shadow, /etc/etc/group
- 用命令行方式添加、删除用户  
adduser [选项] <newusername>  
userdel <用户名>  
groupadd <新组名>  
groupdel <组名>
- 访问权限管理  
修改访问权限: chmod  
字符方式: chmod <ugoa>{+-}<rw> 文件名  
数字方式: chmod xxx 文件名  
举例: chmod a-rwx,u+rw,g+r file  
举例: chmod 700 file  
选项: -R  
chmod -R 777 filename  
改变所属用户  
chgrp [-R]  
chown [-R] [newuser] [newgroup] filename  
例: chown -R .bc /usr/bc\_project
- 用户帐号的查看  
whoami  
who  
w

## 4. 进程管理

要求:

- 1) 用后台方式启动 `gedit` 进程, 并在前台和后台之间切换。
- 2) 用 `kill` 命令, 杀死该进程。
- 3) 将执行的命令和结果都写入实验报告中。

参考命令:

- `ps` 命令
- `top` 命令
- 在后台执行进程  
命令 `&`
- 前台和后台切换  
`^z, bg, fg`
- 进程控制 `kill`  
`kill [-9] PID`  
`kill -l` 列出所有信号
- 设置进程优先级  
`nice [option] command`
- 自动执行的任务:

`at`: 在指定的时间执行指定的命令, 执行结果以 email 方式返回用户信箱

交互式: `at <时间> Enter`

由文件读入: `at 1730 -f job Enter`

`batch`: `at` 的另一个版本

`cron`: 每分钟检查 `crontab`, 定期执行

`crontab -e` 或者 `crontab filename`

`-l, -r, -u username`

## 5. 文件系统命令

- `shell` 命令的基本格式是:  
命令名 [选项] <参数 1> <参数 2> .....
- `man <command>`
- `info <command>`
- `help [command]`



- 查看几个文件: passwd, fstab, grub.conf。利用 man 、 info 、 help 命令查看本次实验所要使用的命令
- 文件查看和连接命令 cat  
cat [选项] <file1> ...
- 分屏显示命令 more  
more [选项] <file>...
- 按页显示命令 less  
less [选项] <filename>
- 复制命令 cp  
cp [选项] <source> <dest>  
或者 cp [选项] <source>... <directory>
- 删除命令 rm  
rm [选项] <name>...
- 移动或重命名命令 mv  
mv [选项] <source> <dest>  
或者 mv [选项] <source>... <directory>
- 创建目录命令 mkdir  
mkdir [-p] <dirName>...
- 删除删除空目录命令 rmdir  
rmdir [-p] <dirName>
- 切换工作目录命令 cd  
cd <dirName>
- 显示当前路径命令 pwd  
pwd
- 查看目录命令 ls  
ls [选项] [<name>...]
- 查找文件或者目录命令 find  
find [path...] [expression]
- 文件定位命令 locate/slocate  
locate [选项] <search string>
- 链接 ln  
ln [选项] <source> <dest>
- 改变文件或目录时间的命令 touch  
touch [选项] <file1> [file2 ...]
- 命令格式为:  
tar <主选项> [辅选项] <文件或者目录>
- 压缩和解压命令 gzip  
gzip [选项] <文件名>

- 解压命令 `unzip`  
`unzip [选项] <压缩文件名>`
- 显示文字命令 `echo`  
`echo [-n] <字符串>`
- 显示日历命令 `cal`  
`cal [选项] [[月] 年]`
- 日期时间命令 `date`  
 显示日期和时间的命令格式为:  
`date [选项] [+FormatString]`  
 设置日期和时间的命令格式为:  
`date <SetString>`
- 清除屏幕命令 `clear`

## 6. vim 编辑器

`vim` 是一个具有很多命令的功能非常强大的编辑。

**目标:** 1、能够熟练地用 `vim` 编辑器编辑各种文本文件

2、自己通过看 `VIM` 的帮助文件和网上的相关资料, 学习其他用法

**操作:** 用 `vim` 创建一个 `c` 程序, 显示 “hello world”

- 用 `vi hello.c` 创建一个文件, 输入标准的 `C` 语言程序
- 用 `gcc -o hello hell.c` 命令, 编译链接该程序, 并修改出现的错误
- 用 `./hello` 命令执行该程序, 观察结果

<code>x</code>	删除当前字符
<code>dd</code>	删除一整行
<code>J</code>	删除一个换行符, 也就是连接两行
<code>u</code>	撤销
<code>U</code>	撤销一整行的修改
<code>CTRL-R</code>	重做
<code>a</code>	当前字符后添加
<code>A</code>	行尾添加文本
<code>ZZ</code>	保存并退出
<code>zz</code>	当前行与光标一起移到窗口中间
<code>:q</code>	退出, 加!表示强制退出
<code>:e!</code>	重新装载原来的文件
<code>w</code>	光标移至下一个单词的词首

e	光标移至下一个单词的词尾
b	光标移至前一个单词的词首
ge	光标移至前一个单词的词尾
\$	光标至行尾
0	行首
^	行首第一个非空字符
%	向前找到第一个括号字符，或者匹配对应的一对括号
gg	移至文件第一行
G	移至文件最后一行
50G	移至第 50 行
50%	移至文件的 50% 处，也就是中间的位置
H	当前窗口文本的最顶端
M	当前窗口文本的中间
L	当前窗口文本的最下方
CTRL-U	向上滚动半屏
CTRL-D	向下滚动半屏
CTRL-E	上滚一行
CTRL-Y	下滚一行
CTRL-F	向下翻页
CTRL-B	向上翻页
zz	光标所在行移至窗口中间
zt	光标所在行移至窗口顶端
zb	光标所在行移至窗口底端
/str	查找单词"str"
?str	反向查找单词"str"
使用 n 或者 N 正向或反向查找下一个匹配的单词	
/>只匹配单词末尾，/<只匹配单词开头	
/查找可以使用正则表达式，正则表达式的语法这里略去	
:set ignorecase	忽略大小写
*	取得当前光标上的单词，并向前查找匹配的字符串
#	取得当前光标上的单词，并反向查找匹配的字符串
c	修改文本，删除并切换至插入模式
cw	删除一个词，切换至插入模式
cc	修改一整行
x	dl，删除当前光标下的字符
X	dh，删除当前光标前的字符
D	d\$，删除至行尾
C	c\$，修改至行尾

s	cl, 修改一个字符
S	cc, 修改一整行
r	替换单个字符
.	重复最后一次的修改操作
v	启动可视模式, 选择文本
V	按整行选择文本
CTRL-V	区块选择文本
o	可视模式下 o 命令表示另一端
p	粘贴文本, 复制的和删除的文本可以重新粘贴出来 如果文本是一整行, 则会插入到下方; 否则插入至光标后面。
P	类似, 只是插入的方向相反, p 和 P 均可以使用多次
y	拷贝文本, 常见组合命令: yw, ye, y\$等
yy	拷贝一整行
Y	拷贝一整行, 同上
"*yy	拷贝至剪贴板
"*p	从剪贴板粘贴
daw	光标位于单词中间位置, 使用本命令可以删除整个单词
cis	aw, is, as 是 VIM 里的文本对象, 表示一个单词, is, as 表示句子
R	进入替换模式
~	改变光标下字符的大小写
I	称到当前行的第一个非空字符并启动插入模式
A	移到行尾启动插入模式

## 7. 使用 U 盘

### 参考命令:

- fdisk -l 查看当前文件系统分区信息
- mkdir /mnt/usb
- mount /dev/sdb /mnt/usb
- umount [挂载点或设备名]

### 要求:

- 1) 挂载 USB 移动硬盘; 在移动硬盘上建立目录, 把上次实验创建的 c 文件拷贝到移动硬盘上该目录下; 进行其他操作; 最后卸载该移动硬盘。
- 2) 将程序和每一步执行的命令和结果都写入实验报告中, 分析遇到的问题和解决方法。

## 8. 基本网络管理命令

要求：

- 1) 使用网络配置命令，按照实验室的 IP 分配规则，为你所在的机器配置 IP 地址、网关、DNS 服务器。
- 2) 用下面查看网络情况的命令，查看网络。
- 3) 将每一步执行的命令和结果都写入实验报告中，分析遇到的问题和解决方法。

参考命令：

- 使用图形配置工具
- 使用终端命令 `ifconfig`  
`ifconfig <设备名> <IP 地址> netmask <掩码>`  
例如：  
`ifconfig eth0 192.168.15.11 netmask 255.255.255.0`  
`ifconfig eth1 21.156.299.13 netmask 255.255.255.0`  
`ifconfig eth0:0 192.168.17.21 netmask 255.255.255.0`  
`ifconfig <设备名> [up|down]`  
查看系统目前所有活跃的网络接口的详细信息

## 9. 其他网络管理命令

- `ping [选项] <目的主机名或 IP 地址>`
- 显示数据包经过路由的命令 `traceroute`
- `traceroute <目的主机 IP 或域名>`
- Netstat：显示路由表、网络端口及每个网络连接的情况
- 添加/删除路由记录  
`route add|del -net <网络号> netmask <网络掩码> dev <设备名>`  
`route add -net 200.1.1.0 netmask 255.255.255.0 dev eth0`
- 添加或者删除默认网关：  
`route add|del default gw <网关名或网关 IP>`  
例如：  
`route add default gw 200.1.1.254`  
`route del default gw 200.1.1.254`
- 查看网络配置文件  
`/etc/sysconfig/network`

```
/etc/sysconfig/network-scripts/*  
/etc/host.conf  
/etc/hosts  
/etc/resolv.conf  
/etc/protocols  
/etc/services  
/etc/xinetd.conf
```

## 10. SSH 服务与客户端

用 SSH 客户端实现从 Windows 操作系统访问 Linux 虚拟机（有 VMware 虚拟机的环境下做该试验）

- 1) 在 Windows 安装 F-Secure SSH 软件
- 2) 确认 VMware 虚拟机网络模式为 NAT 模式，如果不是的话，修改并重启虚拟机
- 3) 在 Linux 中用 ifconfig 命令查看 Linux 虚拟机的 IP 地址，启动 sshd 服务
- 4) 在 SSH 里面启动快速连接，连接地址为 Linux 的 IP 地址，用 Linux 上的用户名登录
- 5) 找到对应的文件，从 Linux 操作系统中下载到 Window 操作系统的对应目录下
- 6) 如果 SSH 无法连接 Linux，通常是由于防火墙的原因。在 Linux 中用 setup 命令修改防火墙设置，关闭或者允许 SSH 通过
- 7) 如果无法激活虚拟网卡，原因是 VMware 提供的虚拟网卡驱动有一点问题：
  - 以 root 权限，编辑 /etc/sysconfig/network-scripts/ifcfg-eth<n> 其中<n>是数字，比如 eth0。
  - 在文件中添加：

```
check_link_down()  
{  
    return 1;  
}
```
  - 然后 ifdown eth0 / ifup eth0

## 11. 网络服务器配置

- 1) 要求：使用三种不同的管理工具，写出启动、重启、关闭服务等操作等命令和执行效果。
  - 图形界面工具：主菜单/系统配置/服务器设置/服务

- 字符界面工具：ntsysv
  - 命令行界面工具：chkconfig
- 2) 练习配置系统中的服务，查看主要的配置选项，根据自己设定的目标，修改相关选项。
- 用 `service` 命令控制系统服务的当前状态。`service` 服务名 `[start|stop|restart]`
  - 配置超级服务器 `xinetd`，系统缺省是配置好的，阅读分析相关的配置文件。
  - 配置 `Apache` 服务器和相关网页文件，使得可以用浏览器浏览你自己做的网页。
  - 配置 `vsftpd` 服务器，使得可以通过 `ftp` 客户端从你的主机上下载文件。
- 3) 将每一步执行的命令和结果都写入实验报告中。

## 实验 2: shell 编程和 GCC 编程环境

### 目的

1. 学习和掌握 shell 编程的基本和常用的方法;
2. 理解用户接口的命令形式;
3. 掌握常用的系统命令。

### 1. shell 编程

#### 要求:

- 1) 查看/etc/passwd 文件看用户使用的 shell 类型
- 2) 通过 cat /etc/shells 命令查看安装的 shell
- 3) 编辑、执行 shell 文件
  - vi, 编辑、保存文件
  - ls -l 查看文件权限
  - chmod 改变程序执行权限
  - 直接键入文件名运行文件
- 4) 将程序和每一步执行的命令和结果都写入实验报告中, 分析遇到的问题和解决方法。

#### 编程题目:

- 1) 请用户输入自己的名字, 然后在屏幕上向用户问好, 并显示其输入的名字。

```
#!/bin/bash
#a simple shell script example
#a function
function say_hello()
{
    echo "Please enter your name:"
    read name
    echo "hello  $name"
}

echo "Programme starts here...."
say_hello
```



```
echo "Programme ends."
```

- 2) 让用户输入一个目录，进入该目录，并显示执行结果。

```
#!/bin/bash
```

```
#another example script of if
```

```
echo "input a directory, please!"
```

```
read dir_name
```

```
if cd $dir_name > /dev/null 2>&1
```

```
then
```

```
    echo "enter $dir_name succeed"
```

```
else
```

```
    echo "enter $dir_name failed"
```

```
fi
```

**挑战：**

- 1) 修改这个程序，使用 if 语句判断并显示输入的文件属性，并显示出结果。
- 2) 输入并修改书上 166 页的脚本程序，帮助用户对文件进行备份。

## 2. GCC 编程环境

**要求：**

- 1) 用 GCC 编译工具，生成 hello.c 文件相应的汇编文件、目标文件、可执行文件。

**示例程序：**

```
#include <stdio.h>
```

```
int func()
```

```
{
```

```
    int i, sum=0;
```

```
    for (i = 0; i < 10; i++)
```

```
        sum += i;
```

```
    return sum;
```

```
}
```

```
int main (void)
```

```
{
```

```
    printf("helle word! sum = %d\n",func());
```

```
    return 0;
}
```

- 2) 用 `objdump` 命令查看可执行文件中的符号信息，先用 `objdump -help` 察看相关的参数。
- 3) 用 `vim` 编辑三个文件，分别是主文件，两个子文件。在子文件中分别定义一个函数分别显示一条信息，表明该函数所在的文件名，在主文件中调用。编辑创建一个 `makefile` 文件，来编译链接这三个文件，最后生成可执行文件，并执行。
- 4) 将程序和 `makefile` 文件内容、每一步执行的命令和结果都写入实验报告中。

#### 示例程序：

- `myapp.c`

```
#include <stdio.h>
#include "greeting.h"

#define N 20
int main (void)
{
    char name[N];
    printf("Your name, please: ");
    scanf("%s", name);
    greeting(name);
    return 0;
}
```
- `/functions/greeting.c`

```
#include <stdio.h>
#include "greeting.h"

void greeting(char *name)
{
    printf("Hello %s !\n", name);
}
```
- `/functions/greeting.h`

```
#ifndef _GREETING_H
#define _GREETING_H

void greeting (char * name);

#endif
```

- makefile

```
my_app:greeting.o my_app.o
gcc my_app.o greeting.o -o my_app
greeting.o:functions\greeting.c functions\greeting.h
gcc -c functions\greeting.c
my_app.o:my_app.c functions\greeting.h
gcc -c my_app.c -Ifunctions
```

### 3. GDB 调试

要求：

- 1) 熟悉常用的 gdb 命令，能够用 gdb 跟踪调试简单的程序
- 2) 输入书上的 gdb 调试示例程序或者自己编写程序，用 gdb 调试，记录主要的错误及其调试的方法和使用的命令，写入实验报告。

常用命令：

- List 列文件清单

```
(gdb) list line1,line2
```

- run 执行程序

在它后面可以跟随发给该程序的任何参数，包括标准输入和标准输出说明符(<和>)和外壳通配符(\*、?、[、])在内。利用 set args 命令可以修改发送给程序的参数，而使用 show args 命令就可以查看其缺省参数的列表。

```
(gdb) set args -b -x
```

- backtrace 命令为堆栈提供向后跟踪功能。

Backtrace 命令产生一张列表，包含着从最近的过程开始的所以有效过程和调用这些过程的参数。

- print 检查各个变量的值

```
(gdb) print p (p 为变量名)
```

- whatis 显示某个变量的类型

```
(gdb) whatis p
```

```
type = int *
```

print 可以显示被调试的语言中任何有效的表达式。表达式除了包含程序中的变量外，还可以包含以下内容：

- ◆ 对程序中函数的调用

```
(gdb) print find_entry(1,0)
```

- ◆ 数据结构和和其他复杂对象

```
(gdb) print *table_start
```

```
$8={e=reference=' \000' ,location=0x0,next=0x0}
```

## ◆ 值的历史成分

(gdb) print \$1 (\$1 为历史记录变量, 在以后可以直接引用 \$1 的值)

## ◆ 人为数组

人为数组提供了一种显示存储器块（数组节或动态分配的存储区）内容的方法。早期的调试程序没有很好的方法将任意的指针换成一个数组。就像对待参数一样，查看内存中在变量 h 后面的 10 个整数，一个动态数组的语法如下所示：

base@length

因此，要想显示在 h 后面的 10 个元素，可以使用 h@10：

(gdb) print h@10

\$13=(-1, 345, 23, -234, 0, 0, 0, 98, 345, 10)

## ● 断点 (breakpoint)

break 命令（可以简写为 b）可以用来在调试的程序中设置断点，有如下四种形式：

- ◆ break line-number 使程序恰好在执行给定行之前停止。
- ◆ break function-name 使程序恰好在进入指定的函数之前停止。
- ◆ break line-or-function if condition 如果 condition（条件）是真，程序到达指定行或函数时停止。
- ◆ break routine-name 在指定例程的入口处设置断点

可以在各个原文件中设置断点，而不是在当前的原文件中设置断点，其方法如下：

(gdb) break filename:line-number

(gdb) break filename:function-name

要想设置一个条件断点，可以利用 break if 命令，如下所示：

(gdb) break line-or-function if expr

例：(gdb) break 46 if testsize==100

从断点继续运行：countinue 命令

## ● 断点的管理

- ◆ 显示当前 gdb 的断点信息：

(gdb) info break 以如下的形式显示所有的断点信息：

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x000028bc	in init_random at qsort2.c:155
2	breakpoint	keep	y	0x0000291c	in init_organ at qsort2.c:168

删除指定的某个断点：

(gdb) delete breakpoint 1

该命令将会删除编号为 1 的断点，如果不带编号参数，将删除所有的断点

(gdb) delete breakpoint

禁止使用某个断点

(gdb) disable breakpoint 1

该命令将禁止断点 1，同时断点信息的 (Enb) 域将变为 n

允许使用某个断点

(gdb) enable breakpoint 1

该命令将允许断点 1, 同时断点信息的 (Enb)域将变为 y

清除原文件中某一代码行上的所有断点

(gdb)clean number

注: number 为原文件的某个代码行的行号

- 变量的检查和赋值
- ◆ whatis:识别数组或变量的类型
- ◆ ptype:比 whatis 的功能更强, 他可以提供结构的定义
- ◆ set variable:将值赋予变量
- ◆ print 除了显示一个变量的值外, 还可以用来赋值
  
- 单步执行
- ◆ next 不进入的单步执行
- ◆ step 进入的单步执行

如果已经进入了某函数, 而想退出该函数返回到它的调用函数中, 可使用命令 finish

- 函数的调用
  - ◆ call name 调用和执行一个函数
- (gdb) call gen\_and\_sork( 1234, 1, 0 )
- (gdb) call printf( "abcd" )
- \$1=4
- ◆ finish 结束执行当前函数, 显示其返回值 (如果有的话)

- 机器语言工具

有一组专用的 gdb 变量可以用来检查和修改计算机的通用寄存器, gdb 提供了目前每一台计算机中实际使用的 4 个寄存器的标准名字:

- ◆ \$pc : 程序计数器
- ◆ \$fp : 帧指针 (当前堆栈帧)
- ◆ \$sp : 栈指针
- ◆ \$ps : 处理器状态
- 原文件的搜索

search text:该命令可显示在当前文件中包含 text 串的下一行。

Reverse-search text:该命令可以显示包含 text 的前一行。

- 命令的历史

为了允许使用历史命令, 可使用 set history expansion on 命令

(gdb) set history expansion on

- 小结: 常用的 gdb 命令

backtrace 显示程序中的当前位置和表示如何到达当前位置的栈跟踪 (同义词: where)

breakpoint 在程序中设置一个断点

cd 改变当前工作目录

clear 删除刚才停止处的断点

commands 命中断点时，列出将要执行的命令

continue 从断点开始继续执行

delete 删除一个断点或监测点；也可与其他命令一起使用

display 程序停止时显示变量和表达式

down 下移栈帧，使得另一个函数成为当前函数

frame 选择下一条 continue 命令的帧

info 显示与该程序有关的各种信息

jump 在源程序中的另一点开始运行

kill 异常终止在 gdb 控制下运行的程序

list 列出相应于正在执行的程序的原文件内容

next 执行下一个源程序行，从而执行其整体中的一个函数

print 显示变量或表达式的值

pwd 显示当前工作目录

pype 显示一个数据结构（如一个结构或 C++ 类）的内容

quit 退出 gdb

reverse-search 在源文件中反向搜索正规表达式

run 执行该程序

search 在源文件中搜索正规表达式

set variable 给变量赋值

signal 将一个信号发送到正在运行的进程

step 执行下一个源程序行，必要时进入下一个函数

undisplay display 命令的反命令，不要显示表达式

until 结束当前循环

up 上移栈帧，使另一函数成为当前函数

watch 在程序中设置一个监测点（即数据断点）

**whatis** 显示变量或函数类型

## 实验 3：编程与调试：进程管理

### 目的

学习和掌握进程创建、执行、等待、退出等进程管理的基本原理和系统调用的使用方法。


学习和掌握进程通信中信号量、消息队列、共享内存的基本原理和常用的系统调用。

### 1. 练习 1

- 1) **要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例程序 1：**最简单的父进程创建子进程的例子。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main (void)
{
    pid_t pid; 
    pid = fork();
    if (pid < 0) {
        printf(“fork error\n”);
        return(0);
    }
    else if (pid == 0) {
        printf(“This is child process!”); /*子进程执行的指令*/
    }
    else {
        printf(“This is parent process!”); /*父进程执行的指令*/
    }
    return(0); /*子进程和父进程都执行的指令*/
}
```

- 2) 修改该程序，在父进程中定义一个变量，初值为 5，并打印该变量的值；在子进程改变在父进程创建子进程之前定义的变量的值，减 1，打印该变量的值；在父进程中加 1，打印该变量的值。观察该变量在子进程和父进程中的值的变化，并记录。

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    pid_t  pid;
    int data = 5;

    if ((pid=fork())<0)
    {
        printf("fork error\n");
        return(0);
    }
    else if (pid==0)
    {
        data--;
        printf("child's data is :%d\n",data);
        return(0);
    }
    else
    {
        printf("parent's data is :%d\n",data);
    }
    return(0);
}
```

- 3) 用 vfork 代替 fork，重新步骤 3。观察结果，并分析与 fork 的区别。

## 2. 练习 2

- 1) **要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果。

**示例程序 2：**在程序中直接调用 execv 启动 shell 命令 ps 查看系统当前的进程信息。

```
#include <unistd.h>
#include <stdio.h>
#include <syslog.h>
main()
```



```

{
    int ret;
    char *path = "/bin/ps";
    char *argv[5] = {"ps", "-a", "-x", NULL};
    if (execv(path, argv))
    {
        syslog(LOG_INFO, "Error executing a program");
    }
    return(0);
}

```

- 2) 修改该程序，在这个程序中加载练习 1 生成的可执行文件。
- 3) 用 fork 创建一个子进程，由其调用 execve 启动 shell 命令 ps 查看系统当前的进程信息。

### 3. 练习 3

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果。

**示例程序：**父进程创建子进程后等待子进程结束，并显示子进程结束时的信息。

```

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    pid_t pid;
    char *message, buf;
    int n, exit_code;

    printf("forking a child process\n");
    pid = fork();
    printf("PID of child process: %d\n", pid);
    switch(pid)
    {
        case -1: perror("fork failed");
                return(1);
        case 0: printf("This is the child\n");
                printf("Please enter a character = ");
                buf = getchar();
    }
}

```

```

        exit_code = 37;
        break;
    default: printf("This is the parent\n");
        exit_code = 0;
        break;
}
if (pid != 0)
{
    int stat;
    pid_t child_pid;
    child_pid = wait(&stat);
    printf("Child has finished PID = %d\n", child_pid);
    if (WIFEXITED(stat))
    {
        printf("Child returned with code %d\n", WEXITSTATUS(stat));
        printf("Child entered character = %c\n", buf);
    }
    else
        printf("Child terminated abnormally\n");
}
return(return_code);
}

```

思考：编译运行这个程序，你会发现程序运行的结果并不像你期望的那样，为什么呢？你能否找出原因并且修改程序？如果有困难的话，和你的同学或者老师讨论：)

## 4. 练习 4

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例程序：**简单的信号处理：捕获用户从控制终端键入的 Ctrl+C 键，并显示信号的值。

```

#include <signal.h>
#include <stdio.h>
#include <unistd.h>

void int_func(int sig)
{
    printf(" in int_func, receive signal=%d\n", sig);
    (void) signal (SIGINT, SIG_DFL);
    return;
}

```

```
int main()
{
    (void) signal(SIGINT, int_func);
    while(1)
    {
        printf("hello world!\n");
        sleep(1);
    }
}
```

## 5. 练习 5

### 1) 要求:

用 **vim** 编辑创建下列文件，用 **GCC** 编译工具，生成可调试的可执行文件，记录并分析执行结果。

**示例程序：**用 **C** 语言编一个程序，父进程创建一个管道和一个子进程，子进程向管道写一字符串，父进程读出该字符串。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>

#define MAXCHARS 20
int main()
{
    int fd[2], size;
    pid_t pid;
    char str[]="hello world!\n";
    char buf[MAXCHARS];
    if (pipe(fd)<0)
    {
        printf("pipe error\n");
        return(1);
    }
    if ((pid=fork()) <0)
    {
        printf("fork error\n");
        return(1);
    }
    if (pid==0)
    {
```

```
        close(fd[0]);
        printf("child write the string : %s",str);
        write(fd[1], str, strlen(str));
        return(0);
    }else
    {
        close(fd[1]);
        size=read(fd[0], buf, sizeof (str));
        printf("parent read the string : %s",buf);
        return(0);
    }
}
```

- 2) 仔细观察输出结果，看看有什么意外？找到原因并修改程序？
- 3) 改写程序，让父进程向管道里写，子进程从管道里读。
- 4) 管道的读写端 fd[0]和 fd[1]是否可以互换呢？设计一个方案来解答这个问题。

## 实验 4：编程与调试：内存管理

### 目的

操作系统的发展使得系统完成了大部分的内存管理工作，对于程序员而言，这些内存管理的过程是完全透明不可见的。因此，程序员开发时从不关心系统如何为自己分配内存，而且永远认为系统可以分配给程序所需的内存。在程序开发时，程序员真正需要做的就是：申请内存、使用内存、释放内存。其它一概无需过问。本章的 3 个实验程序帮助同学们更好地理解从程序员的角度应如何使用内存。

### 1. 练习 1

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例程序：**申请内存、使用内存以及释放一块内存。

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
int main(void)
{
    char *str;
    /* 为字符串申请分配一块内存 */
    if ((str = (char *) malloc(10)) == NULL)
    {
        printf("Not enough memory to allocate buffer\n");
        return(1); /* 若失败则结束程序 */
    }
    /* 拷贝字符串 "Hello" 到已分配的内存空间 */
    strcpy(str, "Hello");
    /* 显示该字符串 */
    printf("String is %s\n", str);
    /* 内存使用完毕，释放它 */
    free(str);
}
```

```
    return 0;
}
```

## 2. 练习 2

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果。

**示例程序：**在刚才实验的基础上重分配内存

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>

int main(void)
{
    char *str;

    /* 为字符串申请分配一块内存 */
    if ((str = (char *) malloc(10)) == NULL)
    {
        printf("Not enough memory to allocate buffer\n");
        return(1); /* 若失败则结束程序 */
    }

    /* 复制 "Hello" 字符串到分配到的内存 */
    strcpy(str, "Hello");

    /* 打印出字符串和其所在的地址 */
    printf("String is %s\n Address is %p\n", str, str);
    /* 重分配刚才申请到的内存空间，申请增大一倍 */
    if ((str = (char *) realloc(str, 20)) == NULL)
    {
        printf("Not enough memory to allocate buffer\n");
        return(1); /* 监测申请结果，若失败则结束程序，养成这个好习惯 */
    }

    /* 打印出重分配后的地址 */
    printf("String is %s\n New address is %p\n", str, str);
}
```

```
/* 释放内存空间 */  
free(str);  
return 0;  
}
```

### 3. 练习 3

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果。

**示例程序：**自动分配内存函数的使用

```
include <stdio.h>  
#include <alloca.h>  
void test(int a)  
{  
    char *newstack;  
    /* 申请一块内存空间 */  
    newstack = (char *) alloca(len);  
    if (newstack)  
        /* 若成功，则打印出空间大小和起始地址 */  
        printf("Alloca(0x%X) returned %p\n", len, newstack);  
    else  
        /* 失败则报告错误，我们是做实验，目前无需退出 */  
        printf("Alloca(0x%X) failed\n", len);  
} /* 函数退出，内存自动释放，无需干预 */  
void main()  
{  
    /* 申请一块 256 字节大小的内存空间，观察输出情况 */  
    test(256);  
    /* 再申请一块更大内存空间，观察输出情况 */  
    test(16384);  
}
```

## 实验 5：编程与调试：文件操作

### 目的

学习和掌握文件控制的基本原理和常用的系统调用。

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

### 1. 练习 1

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例：**利用用户自定义的缓冲区，使对文件的单字节读/写操作在该缓冲区中进行，只有当用户缓冲区空或满时，才调用 read/write 从(或“向”)文件读、写数据。

```
// comprehensive example 1 - read/write operations
```

```
#include <unistd.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
#define SIZ      5
```

```
void rd();
```

```
void wr();
```

```
char buffer[SIZ] = "\0";
```

```
int nread = 0;
```

```
int fd;
```

```
int main(int argc, char **argv)
```

```
{
```

```
    int rw;
```



```
if (argc < 2) {
    fprintf(stderr, "illegal operation");
    return(RETURN_FAILURE);
}

if (!strcmp("read", argv[1])) {
    rw = 1; /* read mode */
    printf("[read mode]\n");
}
else {
    if (!strcmp("write", argv[1])) {
        rw = 0; /* write mode */
        printf("[write mode]\n");
    }
}

switch (rw) {
    case 0:
        wr();
        break;
    case 1:
        rd();
        break;
    default:
        break;
}

return(0);
}

void rd()
{
    if ((fd = open("text", O_RDONLY)) == -1) {
        fprintf(stderr, "file open error\n");
        return(RETURN_FAILURE);
    }

    while ((nread = read(fd, buffer, SIZ)) != 0) {
```

```
        write(1, buffer, nread);
    }

    close(fd);
}

void wr()
{
    while (nread < SIZ) {
        buffer[nread++] = getc(stdin);
        if (nread > SIZ)
            printf("Buffer overrun");
    }

    if ((write(1, buffer, nread)) != nread)
        write(2, "A write error has occurred\n", 27);

    if ((fd = open("text", O_WRONLY)) == -1) {
        fprintf(stderr, "file open error\n");
        return(RETURN_FAILURE);
    }
    else
        write(fd, buffer, SIZ);

    close(fd);
}
```

## 2. 练习 2

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例：**设计一交互实用工具，实用该工具可以查看文件的状态信息；改变文件的模式、访问时间；显示目录文件内容并在其中执行任意的 shell 命令。

```
// Comprehensive example 2 - interactive tools
#include <stdio.h>
#include <stdlib.h>
#include <syslog.h>
```

```
#include <string.h>
#include <sys/stat.h>
#include <time.h>

#define MAXSIZE 64

main(int argc, char *argv[])
{
    char option;
    char command[MAXSIZE], attribute[MAXSIZE];
    struct stat buf;
    struct tm time;

    if (argc > 1)
    {
        printf("1) List file attribute\n");
        printf("2) Modify file attribute\n");
        printf("3) Modified last access time\n");
        printf("4) Enter shell command\n");
        printf("Enter choice: ");
        option = getc(stdin);

        switch(option)
        {
            case '1': if ((stat(argv[1], &buf)) == -1)
                {
                    printf("Error getting file status ...");
                }
                printf("\nFile Status of file %s\n", argv[1]);
                printf("device: %0x\n", (dev_t) buf.st_dev);
                printf("inode: %d\n", (ino_t) buf.st_ino);
                printf("protection: %d\n", (mode_t) buf.st_mode);
                printf("number of hard links: %d\n", (nlink_t)
buf.st_nlink);

                printf("user ID of owner: %d\n", (uid_t) buf.st_uid);
                printf("group ID of owner: %d\n", (gid_t) buf.st_gid);
                printf("device type: %0x\n", (dev_t) buf.st_rdev);
                printf("total size, in bytes: %d\n", (off_t) buf.st_size);
```

```

        printf("blocksize for filesystem: %ld\n", (unsigned long)
buf.st_blksize);
        printf("number of blocks allocated: %ld\n", (unsigned long)
buf.st_blocks);
        printf("time of last access: %ld\n", (time_t) buf.st_atime);
        printf("time of last modification: %ld\n", (time_t)
buf.st_mtime);
        printf("time of last change: %ld\n\n", (time_t) buf.st_ctime);
        break;
    case '2': printf("Enter new attribute eg: 755 (for rwxr-xr-x): ");
        scanf("%s", attribute);
        strcpy(command, "chmod ");
        strcat(command, attribute);
        strcat(command, " ");
        strcat(command, argv[1]);
        system(command);
        break;
    case '3': utime(argv[1], NULL);    // force access & modified time to
                                     // current time
        if ((stat(argv[1], &buf)) == -1)
        {
            printf("Error getting file status ...");
        }
        printf("New access time is: %d", (time_t) buf.st_atime);
        break;
    case '4': printf("Enter any shell command to apply to file eg: ls/cat:
");
        scanf("%s", attribute);
        strcat(command, attribute);
        strcat(command, " ");
        strcat(command, argv[1]);
        system(command);
        break;
    default: printf("Invalid Option: Usage : tool <filename>\n");
        break;
}
}
else

```

```
{  
    printf("Usage : tool <filename>\n");  
}  
return(0);  
}
```

### 3. 练习 3

**要求：**要求：用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例：**设计一个电子通信录，完成 show, append, delete, search, save and return, quit 功能。

// Comprehensive example 4: electronic address book

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define MAXSIZE      1000
```

```
struct Entry {  
    char name[20];  
    char emailaddr[50];  
};
```

```
FILE *fp;
```

```
char filename[] = "addrbook";
```

```
struct Entry addr_book[MAXSIZE];
```

```
int len;
```

```
void show_all() {  
    int i;  
    for (i = 0; i < len; i++)  
        printf("%s    %s\n", addr_book[i].name, addr_book[i].emailaddr);  
}
```

```
void append_record() {  
    printf("name: ");  
    scanf("%s", addr_book[len].name);
```

```
printf("email address: ");
scanf("%s", addr_book[len].emailaddr);

len++;
}

int search_by_name(char *name) {
    int i;

    for (i = 0; i < len; i++)
        if (!strcmp(name, addr_book[i].name))
            return i;

    return -1;
}

void delete_record() {
    char name[20];
    int index;
    int i;

    printf("name: ");
    scanf("%s", name);

    index = search_by_name(name);

    if (index == -1) {
        printf("record not found\n");
        return;
    }

    for (i = index; i < len - 1; i++) {
        strcpy(addr_book[i].name, addr_book[i+1].name);
        strcpy(addr_book[i].emailaddr, addr_book[i+1].emailaddr);
    }

    len--;
```

```
}

void search_record() {
    char name[20];
    int index;

    printf("name: ");
    scanf("%s", name);

    index = search_by_name(name);

    if (index == -1) {
        printf("record not found\n");
        return;
    }

    printf("email address: %s\n", addr_book[index].emailaddr);
}

void save_file() {
    int i;

    if ((fp = fopen(filename, "w")) == NULL) {
        perror("Open address book failed");
        return(1);
    }

    for (i = 0; i < len; i++)
        fprintf(fp, "%s\n%s\n", addr_book[i].name, addr_book[i].emailaddr);

    fclose(fp);
}

int main() {
    char command[10];
    char commandset[][10] = {"show", "append", "delete", "search", "save", "return"};
    int i;
```

```
if ((fp = fopen(filename, "w+")) == NULL) {
    perror("Open address book failed");
    return(1);
}

len = 0;

while (1) {
    if (fscanf(fp, "%s\n%s\n", &addr_book[len].name,
               &addr_book[len].emailaddr) == EOF)
        break;
    len++;
}

fclose(fp);

printf("Input your command: show, append, delete, search, save and return\n");
while (1) {
    scanf("%s", command);

    for (i = 0; i < 6; i++)
        if (!strcmp(command, commandset[i]))
            break;

    if (i == 6) {
        printf("Invalid command\n");
        continue;
    }

    switch (i) {
        case 0:
            show_all();
            break;

        case 1:
            append_record();
            break;
```



```
        case 2:
            delete_record();
        break;

        case 3:
            search_record();
        break;

        case 4:
            save_file();
        break;

        case 5:
            return(0);
    }
}
```

## 实验 6：编程与调试：网络通信

### 目的

Linux 的网络编程主要基于网络 Socket 编程，在本实验中，针对网络编程部分的讲解内容，分别以 TCP 服务端实验程序、客户端实验程序；UDP 服务器端实验程序、客户端实验程序为例，让同学们领会网络通信的过程。

### 1. 练习 1

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例程序：**TCP 客户端实验程序

说明：本程序通过命令行参数指定远程服务器，利用 `gethostbyname()` 函数获得主机地址，并与其建立 TCP 连接，读取服务器端的相应字符串。

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3000          /* 定义我们的通信端口 */
#define MAXDATASIZE 100   /* 定义我们一次接收的最大长度 */

int main(int argc, char * argv[]) {
    int sockfd, recv_bytes;
    char buf[1024];
    struct hostent *he;
    struct sockaddr_in srvaddr;
    /* 如果执行参数错误，退出 */
    if(argc != 2) {
        perror("Usage: tcp-client <hostname>\n");
```

```

    return(1);
}
/* 取主机地址，若出错退出 */
if((he = gethostbyname(argv[1])) == NULL) {
    perror("Error when gethostbyname");
    return(1);
}
/* 建立一个 socket 以便通信，若失败则退出 */
if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Create socket error");
    return(1);
}
/* 将地址结构清 0 后，添入我们的数据 */
bzero(&srvaddr, sizeof(srvaddr));
srvaddr.sin_family = AF_INET;
srvaddr.sin_port = htons(PORT);
srvaddr.sin_addr = *((struct in_addr *)he->h_addr);
/* 开始连接！若出错则退出 */
if(connect(sockfd, (struct sockaddr *)&srvaddr, sizeof(struct sockaddr)) == -1) {
    perror("Error when connect");
    return(1);
}
printf("process %d connect to %s\n", getpid(), (struct in_addr *)he->h_addr);
/* 从服务器端获取应答的字符串 */
if((recv_bytes = read(sockfd, buf, MAXDATASIZE)) == -1) {
    perror("read error");
    return(1);
}
/* 格式化字符串，然后打印 */
buf[recv_bytes] = '\0';
printf("Form Server: %s", buf);
close(sockfd);
}

```

## 2. 练习 2

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，

记录并分析执行结果。

**示例程序：TCP 服务器端实验程序**

说明：与上一个 TCP 客户端程序配合，可以向客户端发送应答字符。

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3000          /* 定义我们的通信端口 */
#define BACKLOG 10        /* 定义处理队列长度 */

main() {
    int sockfd, new_fd;
    struct sockaddr_in srvaddr;
    struct sockaddr_in cliaddr;
    int sin_size;
    /* 建立 socket 以便通信 */
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Creat socket error");
        return(1);
    }
    bzero(&srvaddr, sizeof(srvaddr));
    srvaddr.sin_family = AF_INET;
    srvaddr.sin_port = htons(PORT);
    srvaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    /* 绑定 socket */
    if(bind(sockfd, (struct sockaddr *)&srvaddr, sizeof(struct sockaddr)) == -1) {
        perror("Error when bind");
        return(1);
    }
    /* 开始在 socket 上侦听 */
    if(listen(sockfd, BACKLOG) == -1) {
        perror("listen error");
```

```

        return(1);
    }
    /* 循环等待，处理连接请求和发送应答信息 */
    for(;;) {
        sin_size = sizeof(struct sockaddr_in);
        if((new_fd = accept(sockfd, (struct sockaddr *)&cliaddr, &sin_size)) == -1) {
            perror("Error when accept");
            return(1);
        }
        printf("Server: got connection from %s \n", inet_ntoa(cliaddr.sin_addr));
        if(write(new_fd, "Hello, This is SERVER\n", 22) == -1)
            perror("Error when send string");
        close(new_fd);
    }
    close(sockfd);
}

```

### 3. 练习 3

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果。

**示例程序：**UDP 客户端实验程序

说明：客户端从标准输入读入一行数据，发送该数据，并接收服务器端的返回结果。

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

#define PORT 3300
#define BUF 1024

void udpc_requ(int sockfd, const struct sockaddr_in * addr, int len) {
    char buf[BUF];
    int n;

```

```
printf("\nPlease input:");
for(;;fgets(buf, BUF, stdin)!= NULL;) {
    /* 发送数据 */
    sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *) addr, len);
    /* 接收服务器端的回应 */
    n = recvfrom(sockfd, buf, BUF, 0, NULL, NULL);
    buf[n] = 0;
    printf("\nString from Server:");
    fputs(buf, stdout);
    printf("\nPlease input:");
}
}

int main(int argc, char * argv[]) {
    int sockfd;
    struct sockaddr_in addr;
    /* 检查参数，如不对则提示并退出 */
    if(argc != 2) {
        fprintf(stderr, "Usage: udp-client <IP address>\n");
        return(1);
    }
    /* 创建 socket 以便通信 */
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd < 0) {
        fprintf(stderr, "Create socket error.\n");
        return(1);
    }
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    if(inet_aton(argv[1], &addr.sin_addr) < 0) {
        fprintf(stderr, "IP address to number error\n");
        return(1);
    }
    /* 调用用户函数，发送并接收数据 */
    udpc_requ(sockfd, &addr, sizeof(addr));
    close(sockfd);
    return 0;
}
```

## 4. 练习 4

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果。

**示例程序：**UDP 服务器端实验程序

**说明：**服务器监听指定的端口，等待请求，并回应。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

#define PORT 3300
#define MSG 1024

void udps_respon(int sockfd) {
    struct sockaddr_in addr;
    int addrlen;
    char msg[MSG];

    addrlen = sizeof(struct sockaddr);
    /* 一个无限循环，等待服务请求并响应 */
    for(;;) {
        /* 接收请求数据 */
        n = recvfrom(sockfd, msg, MSG, 0, (struct sockaddr *)&addr, &addrlen);
        /* 返回响应数据 */
        sendto(sockfd, msg, n, 0, addr, addrlen);
    }
}

int main() {
    int sockfd;
    struct sockaddr_in addr;
    /* 创建一个 socket 以便通信 */
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd < 0) {
```

```
    fprintf(stderr, "Create socket error\n");
    return(1);
}
bzero(&addr, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(PORT);
/* 绑定 socket */
if(bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    fprintf(stderr, "Error when bind\n");
    return(1);
}
/* 调用用户函数执行服务 */
udps_respon(sockfd);
close(sockfd);
}
```



## 实验 7：编程与调试：线程编程

### 目的

1. 学习和掌握线程基本原理和常用的系统调用；
2. 对比进程和线程，学习掌握线程的同步与互斥；
3. 学习掌握多线程编程的基本方法。

### 1. 练习 1

**要求：**用 vim 编辑创建下列文件，用 GCC 编译工具，生成可调试的可执行文件，记录并分析执行结果，分析遇到的问题和解决方法。

**示例：**用线程共享数据的方法实现两个线程同步：定义一个共享字符串，一个线程接收用户输入，然后通知另一个线程，另一个线程接到通知后，显示该字符串，否则显示正在等待接收。

```
#include <pthread.h>
```

```
#include <stdio.h>
```

```
void *thread_function(void *arg);
```

```
char message[10];
```

```
int done = 0;
```

```
int main()
```

```
{
```

```
    int tid;
```

```
    pthread_t a_thread;
```

```
    pthread_attr_t attr;
```

```
    // Initialize the thread attributes
```

```
    pthread_attr_init(&attr);
```

```
    // Create another thread
```

```
    tid = pthread_create(&a_thread, &attr, thread_function, message);
```

```
    if (tid != 0) {
```

```
    perror("Thread creation failed");
    return(1);
}

// Get input string
scanf("%s", message);
done = 1;
sleep(1);
}

void *thread_function(void *arg) {
    printf("thread_function is waiting for input\n");
    while (!done) sleep(1);
    printf(message);
    printf("\n");
}
```

## 2. 练习 2

**要求：** 用多线程方法编程实现多客户/服务器模式的聊天室。可以启动多个客户端，每个客户接收用户输入的字符串发给服务器，服务器为每一个客户创建一个线程，显示客户发来的信息，并向客户发回应答字符。

**示例：**

```
// this example includes two programs: thread5_server.c and thread5_client.c
// thread5_server.c
#include <pthread.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/un.h>
#include <sys/ioctl.h>
#include <unistd.h>

#define MAXMSG      512

int makeSocket();
```

```
void *receive_thread(void *);

int main()
{
    int sock;
    int result;
    fd_set readfds;
    int fd;
    struct sockaddr_un client_address;
    int client_len;
    int client_sock;
    pthread_t a_thread;
    pthread_attr_t attr;

    // Initialize the thread attributes
    pthread_attr_init(&attr);

    // create the socket
    sock = makeSocket();

    // create a connection queue
    if (listen(sock, 5) < 0) {
        perror("Listen failed");
        return(1);
    }

    while (1) { /* Main server loop - forever */
        // initialize readfds to handle input from sock
        FD_ZERO(&readfds);
        FD_SET(sock, &readfds);

        // wait for clients and requests
        result = select(FD_SETSIZE, &readfds, NULL, NULL, NULL);
        if (result < 1) {
            perror("Select failed");
            return(1);
        }
    }
}
```

```
for (fd = 0 ; fd < FD_SETSIZE; fd++) {
    if (FD_ISSET(fd, &readfds)) {
        if (fd == sock) { /* a new connection request */
            client_len = sizeof(client_address);
            client_sock = accept(sock, (struct sockaddr *)&client_address,
                                &client_len);

            printf("A client joined on fd %d\n", client_sock);

            // Create a thread to handle the chat with the client
            result = pthread_create(&a_thread, &attr, receive_thread,
                                    (void *)&client_sock);
            if (result != 0) {
                perror("Thread creation failed");
                return(1);
            }
        }
    }
}

int makeSocket()
{
    int sock;
    struct sockaddr_un serv_addr;

    // remove any old socket
    unlink("server_socket");

    // create the socket
    sock = socket(PF_UNIX, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Create socket failed");
        return(1);
    }

    // name the socket
```

```
serv_addr.sun_family = PF_UNIX;
strcpy(serv_addr.sun_path, "server_socket");
if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    perror("Bind failed");
    return(1);
}

return sock;
}

void *receive_thread(void *arg)
{
    char buffer[MAXMSG];
    char reply_buffer[MAXMSG];
    int sock;
    int result;
    int fd;
    fd_set readfds;
    int i;

    sock = *(int*)arg;

    while (1) {
        // initialize readfds to handle input from sock
        FD_ZERO(&readfds);
        FD_SET(sock, &readfds);

        result = select(FD_SETSIZE, &readfds, NULL, NULL, NULL);
        if (result < 1) {
            perror("Select failed");
            return(1);
        }

        for (fd = 0; fd < FD_SETSIZE; fd++) {
            if (FD_ISSET(fd, &readfds)) {
                ioctl(fd, FIONREAD, &result);
            }
            if (result == 0) { /* the client closed the connection */
                close(fd);
            }
        }
    }
}
```

- 53 -

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/un.h>
#include <sys/ioctl.h>
#include <unistd.h>

#define MAXMSG 512

void writeToServer(int sock);

int main(int argc, char *argv[])
{
    int sock;
    struct sockaddr_un serv_addr;

    // create the socket
    sock = socket(PF_UNIX, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Create socket failed");
        return(1);
    }

    // name the socket
    serv_addr.sun_family = PF_UNIX;
    strcpy(serv_addr.sun_path, "server_socket");

    // connect to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("Connect failed");
        return(1);
    }

    // send data to the server
    writeToServer(sock);
    close(sock);
}

void writeToServer(int sock)
```

```
{
    char buffer[MAXMSG];
    int result;
    int i;

    while (1) {
        scanf("%s", buffer);

        // send message to server
        result = write(sock, buffer, strlen(buffer));
        if (result < 0) {
            perror("Write failed");
            return(1);
        }

        // clear the buffer
        for (i = 0; i < MAXMSG; i++)
            buffer[i] = '\0';

        // receive reply from server
        result = read(sock, buffer, MAXMSG);
        if (result < 0) {
            perror("Read failed");
            return(1);
        }

        printf("%s\n", buffer);
    }
}
```



# 综合实验

## 1. 题目：

基于 Linux 平台的 C/S 结构即时通信系统设计与实现

## 2. 目的

在本课程前半部分 Linux 操作实验的基础上，学生基本掌握了 Linux 平台上的用户和文件管理、系统管理、设备管理、网络管理及进程管理等基本操作。在此基础上进行 Linux 平台上的编程和系统设计能力的训练，提高学生的 Linux 平台系统编程能力，培养学生分解问题、解决问题的能力，逐步设计实现一个模型系统。

该综合实验是本实验指导手册中实验 3 到实验 7 实验内容的综合和扩展，学生在完成这些实验的基础上，有针对性地应用所学的知识，进行扩展性的应用系统设计和实现的训练，供学有余力的学生选择和参考。

## 3. 实验要求与评价

### 1) 基本功能要求：

本项目设计一个基于 Linux 平台的 C/S 结构即时通信系统原型。多个客户端和一个服务器端位于网络上的不同主机，利用 UDP 协议进行通信。客户端通过 Socket 接口向服务器发出短信，服务器端监听客户端发来的请求，创建子进程处理该客户端请求，包括转发客户端的消息给其他客户端，并把消息写入本地后备存储中（文件或者数据库）。

### 2) 设计与实现要求：

- 在 Linux 平台上设计并实现本系统。
- 根据基本功能要求进行系统分析与设计，提交系统结构图设计
- 可以在基本要求基础上，添加客户信息管理、数据库支持等功能。
- 可以用线程代替进程实现服务器端的设计，并比较两者的性能。

### 3) 评定标准（根据学生的实际情况进行调节）

- 合格：基本实现项目中的单个实验的基本要求
- 中等：实现项目中的单个实验的基本要求
- 良好：实现整个系统的基本要求

- 优秀：在实现系统基本功能的基础上，实现自己设计的附加功能

## 4. 实验内容及学时安排

实验题目和要求在课程前期布置给学生，要求学生进行系统功能分析和设计。在本综合实验开始时，要求学生提交项目分析和设计报告。

本项目的实施过程包括 6 次小实验，每次 2 小时，分别涉及 Shell 编程及 GCC 编程环境，进程创建和执行，进程间通信，内存申请和分配，文件访问，socket 网络编程。每次完成大题目中相关内容的设计与实现。

### 1) 实验 1: shell 编程和 GCC 编程环境

- 熟悉 GCC 编程环境和 Shell 脚本。
- 根据项目分析和设计报告，设计源程序目录树，编写实现本项目的 Makefile 文件和相关 Shell 脚本。

### 2) 实验 2: 进程创建和执行

- 利用 Fork、execv、wait、return 等系统调用，设计并实现服务器端的父进程创建子进程的过程。
- 设计客户端用户界面和屏幕功能区划分，实现接受用户输入，并显示在屏幕上。

### 3) 实验 3: 进程间通信

- 利用管道、消息通信或共享内存方式，实现父进程和子进程的通信
- 在服务器端，父进程接受用户输入的信息，转发给子进程，两者分别显示输出，以验证父进程与子进程通信功能。

### 4) 实验 4: 内存申请和释放

- 在子进程中，实现内存的动态申请与释放功能，为将来的数据处理做准备。

### 5) 实验 5: 文件访问

- 在父进程中，实现文件的创建、打开、关闭功能
- 在子进程中，实现文件的读写操作，并验证功能正确性。

### 6) 实验 6: Socket 网络编程

- 利用 socket 编程接口，实现服务器端和客户端的通信。
- 客户端：接受用户输入后，显示在屏幕上的同时，发送给服务器。
- 服务器端：接受客户端请求，转发给子进程，由子进程完成信息显示和存储。
- 设计客户端和服务器的简单通信协议，基本协议包括客户端注册、发送消息协议。

根据简单协议实现：

1. 注册：客户端发出注册请求，服务器端在本地记录客户端 IP、ID 号等信息
2. 发送消息：客户端发送信息给服务器，服务器根据消息目标地址，确定要转发的目标客户端，如果没有指定客户端，则广播。