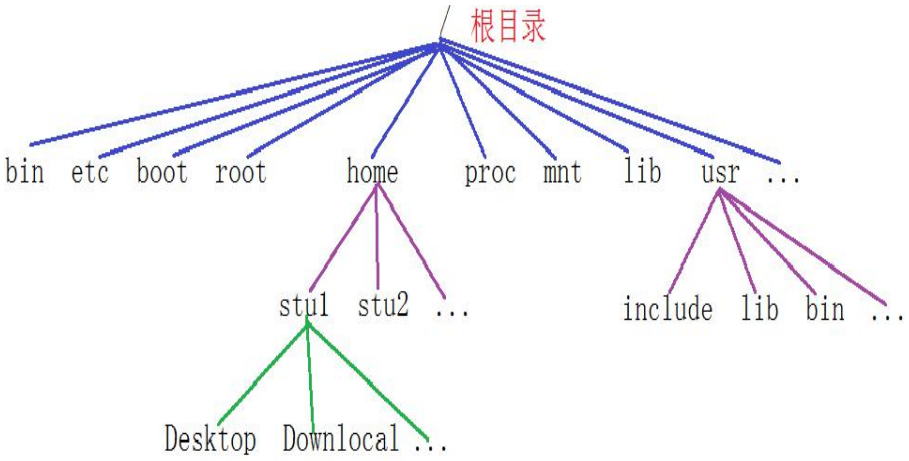
图论科技 Linux 复习大纲

第一部分：Linux 基础命令

1、Linux 文件层次结构

Linux 文件存储时，都是以根目录“/”开始的。



/bin　存放可以在单用户模式下使用的命令的可执行文件

/etc　存放系统主要的配置文件

/boot　存放系统开机所需要的核心文件和配置文件

/root　系统管理员（root）用户的家目录

/home 普通用户的家目录，用户登录上系统后默认进入自己的家目录下工作

/proc　一个虚拟文件系统，存放内存上的数据，如：系统核心、进程信息、网络状态等。

/mnt　外部设备临时挂载点

/lib　存放系统开机或者执行命令时所需要用到库文件

/usr　安装的第三方软件

参考书籍资料：

《鸟哥 Linux 私房菜--基础学习篇》第 8 章第 1,2 节

《鸟哥 Linux 私房菜--基础学习篇》第 6 章第 3 节

2、文件管理

cd + 路径 改变当前工作目录　路径可以为绝对路径或者相对路径

ls　显示当前目录下的文件	touch　新建普通文件
rm　删除普通文件	mkdir　新建目录文件
rmdir　删除空目录	rm -r　删除非空目录
cp　拷贝普通文件	cp -r　拷贝目录文件

mv	移动(剪切)文件	chmod	修改文件权限
chown	修改文件属主	chgrp	修改文件组用户
pwd	显示当前工作目录绝对路径	find	查找文件
vim	文本编辑器	wc	统计数目
nl	显示文件内容和行号	umask	显示/设置文件的缺省权限

whereis：搜索命令所在目录及帮助文档路径

more cat less head tail 查看文件内容

参考书籍资料：

《鸟哥 Linux 私房菜--基础学习篇》第 7 章

《鸟哥 Linux 私房菜--基础学习篇》第 10 章 vim 使用

3、权限管理

文件权限的划分：

rw-(文件属主--u) rw-(同组用户--g) r--(其他用户--o)

其中，r 代表有读权限，w 代表有写权限， x 代表有可执行权限(普通文件)或者能否进入权限(目录文件)。

修改文件权限的命令： chmod

a) 字符修改法：

```
chmod   u/g/o/a  +/-/=  r      filename
                         w
                         x
                         rw
                         rx
                         wx
                         rwx
```

如： chmod u+rw main.c 给 main.c 文件的属主加上读写权限

chmod g-w main.c 给 main.c 文件的同组用户去掉写权限

chmod u+w,g-r,o=r main.c 给 main.c 文件的属主加上写权限，同组用户去掉读权限，将其他用户的权限设置为只读。

b) 数字修改法：

r：4 w：2 x：1

如： chmod 664 main.c 将 main.c 文件的权限设置为属主可读可写，同组用户可读可写，其他用户可读。

参考书籍资料：

《鸟哥 Linux 私房菜--基础学习篇》第 6 章第 2 节

4、进程管理

ps　显示进程信息　　kill　pid　结束进程

kill　-stop　pid　挂起进程　　kill　-9　pid　强制结束进程

jobs　显示后台和挂起的进程任务　　&　在后台运行进程

bg　任务号　将挂起的进程放到后台执行

fg　任务号　将挂起或后台的进程放到前台执行

pstree　以树状图显示进程

参考书籍资料：

《鸟哥 Linux 私房菜--基础学习篇》第 17 章第 1,2,3,4 节

5、系统管理

top　动态显示进程信息以及系统运行统计信息

free　显示系统运行统计信息 ---　内存　buffers/cache　Swap

ipcs　-s/-q/-m　　分别显示系统的信号量、消息队列、共享内存

ipcrm　-s/-q/-m　id　根据 id 分别删除信号量、消息队列、共享内存

lsof　列出当前系统打开的所有文件描述符

mpstat：实时监测多处理器系统上的每个 CPU 使用情况

vmstat：实时输出系统各个资源的使用情况

参考书籍资料：

《Linux 高性能服务器编程》　第 13 章第 8 节

《Linux 高性能服务器编程》　第 17 章

用户管理命令：

useradd/adduser　newname　添加一个新的用户 newname

参数：　-g　执行新用户的主组

-G　将新用户添加到副组

-s　指定新用户默认使用的 shell 终端

-d　指定新用户登录默认进入的目录

passwd　username　修改用户密码

userdel　username　删除用户

参考书籍资料：

《鸟哥 Linux 私房菜--基础学习篇》第 14 章第 1,2,4,6 节

6、网络通讯

ping　测试网络连通性

ifconfig/ip　显示或设置网络设备

netstat/ss　显示网络相关信息

service　管理系统运行的服务器

mail　查看、发送电子邮件

write　给用户发信息

参考书籍资料：

《Linux 高性能服务器编程》　第 17 章

7、关机/重启

a) 系统运行级别：（root 用户可以通过 init 命令修改运行级别）

0　关机

1　单用户模式

2　不带网络的多用户模式

3　完全的多用户模式，优先进入字符界面

4　未定义

5　xll　---　图形界面模式

6　重启

b) 关机/重启命令：

init　0　关机(root 用户才能运行)

init　6　重启(root 用户才能运行)

halt　立即关机　　poweroff　立即关机

shutdown　立即关机　　reboot　立即重启

8、文件压缩

a) tar　打包　+　gzip 压缩　---》　生成 .tar.gz 压缩包

b) tar 直接压缩　---》　生成 .tgz 压缩包

参数：　c　创建新文件

f　指定目标为文件而不是设备

v 显示详细过程

t 只显示包中内容，而不真正释放

x 释放文件

z GNU 版本新添加的， 完成压缩工作

以上两个方式可以互相混用,通过 tar 压缩的文件 gzip 也可以解压,通过 gzip 压缩的文件 tar 也可以解压！！！

关于压缩与解压命令详见：

《鸟哥的 Linux 私房菜---基础篇》第九章

9、编译链接

源文件生成最终的可执行文件分：预编译、编译、汇编、链接四步：

a) 预编译

```
gcc -E hello.c /* -o hello.i */
```

b) 编译

```
gcc -S hello.i /* -o hello.s */  生成汇编指令代码
```

或者 `gcc -S hello.c /* -o hello.s */`

c) 汇编

```
gcc -c hello.s /* -o hello.o */ 生成可重定位的二进制目标文件
```

或者 `gcc -c hello.c /*-o hello.o */`

d) 链接

```
gcc -o hello hello.o  --->  生成最终的可执行文件 hello
```

或者 `gcc -o hello hello.c`

当然，在实际的使用中，一般我们可以将预编译、编译、汇编过程合并到一块（即就是执行：gcc -c hello.c),然后完成链接过程,(执行：gcc -o hello hello.o)。也可以直接通过源文件生成可执行文件(即就是执行：gcc -o hello hello.c）。

那么这四步分别完成的工作是什么呢：

预编译阶段：a) 删除所有的“#define”，并且展开所有的宏定义;

b) 处理所有的条件预编译指令，“#if”、“#ifdef”、“#endif”等;

c) 处理“#include”预编译指令，将被包含的文件插入到该预编

译指令的位置;

d) 删除所有的注释;

e) 添加行号和文件名标识，以便于编译器产生调试用的符号信息及编译时产生编译错误和警告时显示行号;

f) 保留所有的#pragma 编译器指令，因为编译器需要使用它们。

编译阶段： 词法分析、语法分析、语义分析，**代码优化，汇总符号**。

汇编阶段： 将汇编指令翻译成二进制格式，**生成各个 section，生成符号表**。

链接阶段： a) 合并各个 section，调整 section 的起始位移和段大小

b) 合并符号表，进行符号解析

c) 符号重定位

相关命令：

objdump 查看目标文件或者可执行的目标文件的构成

参考博客：http://man.linuxde.net/objdump

readelf 显示一个或者多个 elf 格式的目标文件的信息

参考博客: http://man.linuxde.net/readelf

ldd 查看可执行程序用到哪些共享库

nm 查看程序中函数和变量的逻辑地址

参考资料书籍：

《程序员的自我修养》第 2,3,4,7,10 章

10、gdb 调试

a) debug 版本

debug 版本为可调式版本，生成的可执行文件中包含调试需要的信息。我们作为开发人员，最常用的就是 debug 版本的可执行文件。

debug 版本的生成：

因为调试信息是在编译过程时加入到中间文件（.o）中的，所以必须在编译时控制其生成包含调试信息的中间文件。

```
gcc  -c  hello.c  -g     ---> 生成包含调试信息的中间文件
gcc  -o  hello  hello.o
```

或者 `gcc -o hello hello.c -g`

b) release 版本

release 版本为发行版本，是提供给用户使用的版本。用 gcc 默认生成的就是 release 版本。

首先将源代码编译、链接生成 debug 版本的可执行文件，然后通过‘gdb debug 版本的可执行文件名’进入调试模式。

a）单进程、单线程基础调试命令

l	显示 main 函数所在的文件的源代码
list filename:num	显示 filename 文件 num 行上下的源代码
b linenum	给指定行添加断点
b funname	给指点函数的第一有效行添加一个断点
info break	显示断点信息
delete 断点号	删除指定断点
disable 断点号	将断点设定为无效的，如果不加断点号，将所有断点设置为无效
enable 断点号	将断点设定为有效的，如果不加断点号，将所有断点设置为有效
r（run）	运行程序
n（next）	单步执行
c （continue）	继续执行，直接执行到下一个断点处
s	进入将要被调用的函数中执行
finish	跳出函数
q	退出调试
p val	打印变量 val 的值
p &val	打印变量 val 的地址
p a+b	打印表达式的值
p arr（数组名）	打印数组所有元素的值
p *parr@len	用指向数组的指针打印数组所有元素的值
x /nfu addr	打印 addr 表示的内存存储值

n ： 是一个正整数，表示显示内存的长度

f ： 表示显示的格式， x 十六进制； d 十进制； o 八进制 t 二进制； c 字符； f 浮点数

u： 表示从当前地址向后请求的字节数，默认 4bytes，可以指定：

b 单字节 h 双字节 w 四字节 g 八字节

display 自动显示，参数和 p 命令一样

info display 显示自动显示信息

undisplay + 自动显示编号 删除指定的自动显示

ptype val 显示变量类型

bt 显示函数调用栈

b）多进程调试命令

（gdb）set follow-fork-mode mode

mode 可以选择 parent 或者 child，即：选择调试那个进程。

注意：未被选择的进程会直接执行结束。

c）多线程调试命令

a) 利用 info threads 查看线程信息；

b) thread id 调试目标 id 指定的线程；

c) set scheduler-locking off | on | step；

“off”表示不锁定任何线程；

“on”只有当前被调试的线程继续运行；

“step”在单步执行的时候，只有当前线程会执行；

参考书籍资料：

《GDB 完全手册》

《Linux 高性能服务器编程》第 16 章第 3 节

《Linux 程序设计》第 10 章

11、makefile 文件

主要包括五方面内容：显示规则，隐晦规则，变量定义，文件指示和注释；

a）功能：关系到整个工程的编译规则，也可执行操作系统的命令，（好处）实现自动化编译整个工程，提高编译效率（只会编译修改的和依赖于修改的那些文件）；

b）简单的编写步骤：

现在有 5 个文件，分别为：main.c my1.h my2.h my1.c my2.c ；

现在编写 makefile 文件，名称为 makefile 过程如下：

a. main: main.o my1.o my2.o

gcc -o main main.o my1.o my2.o

main.o: main.c my1.h my2.h

gcc -c main.c

my1.o: my1.c my1.h

gcc -c my1.c

my2.o: my2.c my2.h

gcc -c my2.c

clean:

rm -f *.o main

b.执行命令：make 就可以生成可执行文件 main. 如果要删除可执行文件以及所有的中间目标文件，只需要执行下 make clean .

**clean 不是一个文件，它只是一个动作的名字，如 C 语言中的 lable,冒号后什么都不做；

c）文件说明

在上面的 makefile 文件当中，目标文件包括：执行文件 main 和中间目标文件（*.o），依赖文件（就是冒号后面的那些文件）。每一个 .o 文件都有一组依赖文件，而这些 .o 文件又是执行文件 main 的依赖文件。依赖关系的实质上就是说明了目标文件是由那些文件生成的。

d）make 的工作原理

Make 是解释 Make File 中指令的命令工具。

当我们输入 make 时：

i. make 首先会在当前目录下寻找 makefile 文件。

ii. 如果找到，它会找到文件中的第一个目标文件 main，并把这个文件作为最终的目标文件。

iii. 如果 main 文件不存在或者是 main 所依赖的后面的 .o 文件的修改时间要比 main 这个文件新，那么，他就会执行后面所定义的命令来生成 main 这个文件。

iv. 如果 main 所依赖的 .o 文件也存在，那么 make 会在当前文件中找目标为 .o 文件的依赖性，如果找到则再根据那一个规则生成 .o 文件。

v. 由于我的 c 文件和 h 文件都是存在的，所以 make 会生成 .o 文件，然后再用 .o

文件声明 make 的最终文件，也就是执行文件 main.

e）make 和 makefile 的联系

make 是一个命令，makefile 是一个文件，当 makefile 文件写好后，只需要一个 make 命令就可以将 makefile 文件运行起来。

参考书籍资料：

《给我一起写 makefile》

《Linux 程序设计》第 9 章第 2 节

12、库文件

库文件是预先编译好的方法的集合。比如：我们提前写好一些数据公式的实现，将其打包成库文件，以后使用只需要库文件就可以，而不需要重新编写。库文件分为：静态库和动态库（也叫共享库）两种。windows 系统静态库扩展名为：.lib 动态库扩展名为：.DLL； Linux 系统静态库扩展名为：.a 动态库扩展名为： .so

静态库

a）特点：

程序在链接的过程当中，链接器从库文件中取得所需代码，复制到生成的可执行文件当中。因此，静态库是在程序的链接阶段被复制到程序当中，和程序的执行过程没有关系。

b）创建

Eg:有两个源文件：main.c max.c

1）gcc -c main.c max.c （得到 main.o 和 max.o）

2）使用 ar 将目标文件归档

ar crv libxxx.a main.o max.o

**libxxx.a 即为我们创建的静态库。

动态库（共享库）

a）特点：

动态库在链接阶段并没有被加载到程序当中，而程序在运行时被系统动态加载到内存当中供程序使用。

b）创建

Eg:有一个源文件 max.c

a） gcc -fPIC -shared -o libxx.so max.c

上行代码等价与：gcc -c -fPIC max.c

gcc -shared -o libxx.so max.o

**libxx.so 即就是我们生成的共享库

静态库和共享库的区别

静态库的可执行文件当中包含了库代码的一份完整的拷贝，因此，当它被多次使用时就会有多份没用的拷贝在内存当中，所以比较占内存；而动态库系统只需载入一次动态库，不同的程序可以得到内存当中相同的动态库的副本，因此会节省很多内存。

参考书籍资料：

《Linux 程序设计》 第 1 章第 2 节第 4 小节

第二部分：系统编程

1、系统进程管理--PCB

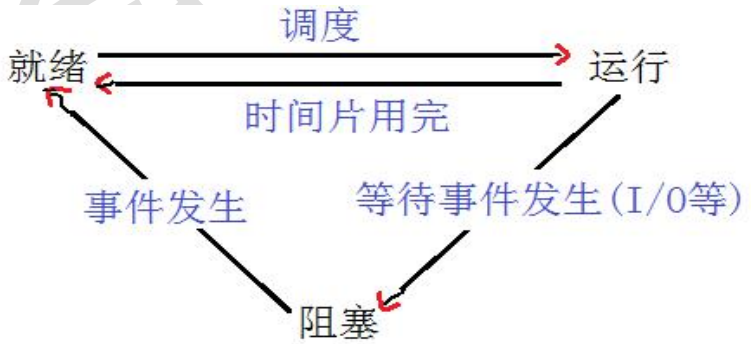
进程是执行中的程序，是一个动态的过程，好比程序是乐谱，进程就是演奏的过程。操作系统为了管理进程，因而通过一个 task_struct 结构体记录进程的信息(进程标示符、优先级、进程状态、程序计数器、程序上下文、信号、打开的文件等等)，每一个进程都有一个 task_struct 结构体变量，称之为 PCB(进程控制块)，32 位系统上，每个 PCB 大概 1.7K。操作系统通过一个双向循环链表管理所有的 PCB。

参考书籍资料：

《Linux 内核设计与实现》第 3 章第 1 节

2、进程运行状态

进程在其生命周期内可以存在多个状态，主要有：就绪 运行 阻塞状态



绪：所有资源准备完成，等待 CPU 空闲的状态；

运行： 在 CPU 上真正执行的状态；

阻塞： 等待某些事件发生，事件未发生前，不能被 CPU 调度的状态。

就绪--》运行： 系统进行进程调度；

运行--》就绪： 分配给进程的时间片用完；

运行--》阻塞： 需要某些事件的发生才能运行；（阻塞函数）

阻塞--》就绪： 等待的事件已经发生，只能转到就绪态，不能直接到运行态。

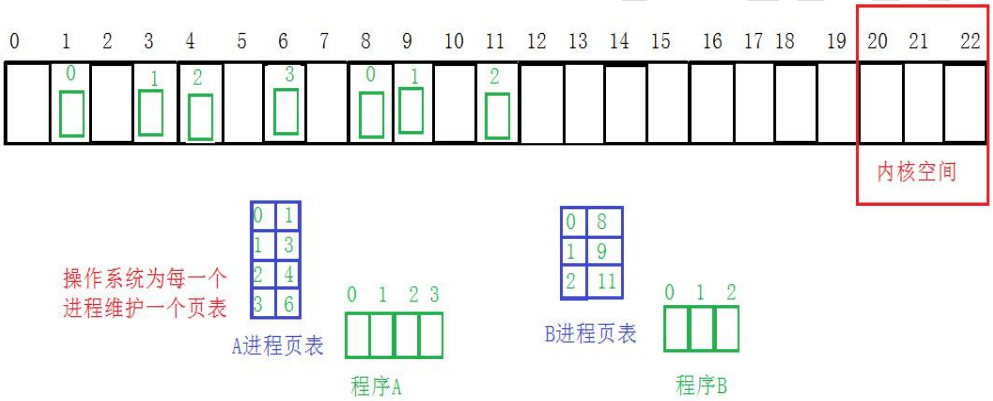
除此之外，进程还有创建、退出、就绪/挂起、阻塞/挂起等状态，

参考书籍资料：

《操作系统精髓与设计原理》第 3 章第 3 节

《从哲学层次上看操作系统》第 4 章

3、程序加载 --- 以简单分“页”为例



操作系统为每一个进程维护一个页表，所以程序加载的时候，不要求在主存上连续。

4、进程创建 --- fork

fork 函数原型： pid_t fork(void);

函数返回类型 pid_t 实质是 int 类型，Linux 内核 2.4.0 版本的定义是：

```
typedef int          __kernel_pid_t;
typedef __kernel_pid_t    pid_t;
```

fork 函数会新生成一个进程，调用 fork 函数的进程为父进程，新生成的进程为子进程。fork 函数调用一次，返回两次，在父进程中返回子进程的 pid，在子进程中返回 0，失败返回-1。fork 函数在生成子进程时，用到了写时拷贝技术：不执行一个父进程数据段、栈和堆的完全复制，这些区域由父、子进程共享，而内核将他们的访问权限改为只读的。如果父、子进程中任何一个试图修改这些区域，则内核只为修改区域的那块内存制作一个副本，并且是以虚拟存储器系统中的“一页”为单位复制。

fork 函数的内核实现原理：

Linux通过clone()系统调用实现fork()。这个调用通过一系列的参数标志来指明父、子进程需要共享的资源（关于这些标志更多的信息请参考本章后面3.3节）。fork()、vfork()和__clone ()库函数都根据各自需要的参数标志去调用clone()。然后由clone()去调用do_fork()。

do_fork完成了创建中的大部分工作，它的定义在kernel/fork.c文件中。该函数调用copy_process()函数，然后让进程开始运行。copy_process()函数完成的工作很有意思：

- 调用dup_task_struct()为新进程创建一个内核栈、thread_info结构和task_struct，这些值与当前进程的值相同。此时，子进程和父进程的描述符是完全相同的。
- 检查新创建的这个子进程后，当前用户所拥有的进程数目没有超出给他分配的资源的限制。
- 现在，子进程着手使自己与父进程区别开来。进程描述符内的许多成员都要被清0或设为初始值。进程描述符的成员值并不是继承而来的，而主要是统计信息。进程描述符中的大多数数据都是共享的。
- 接下来，子进程的状态被设置为TASK_UNINTERRUPTIBLE以保证它不会投入运行。
- copy_process()调用copy_flags()以更新task_struct的flags成员。表明进程是否拥有超级用户权限的PF_SUPERPRIV标志被清0。表明进程还没有调用exec()函数的PF_FORKNOEXEC标志被设置。
- 调用get_pid()为新进程获取一个有效的PID。
- 根据传递给clone()的参数标志，copy_process()拷贝或共享打开的文件、文件系统信息、信号处理函数、进程地址空间和命名空间等。在一般情况下，这些资源会被给定进程的所有线程共享；否则，这些资源对每个进程是不同的，因此被拷贝到这里。
- 让父进程和子进程平分剩余的时间片（第4章将作具体讨论）。
- 最后，copy_process()作扫尾工作并返回一个指向子进程的指针。

再回到do_fork()函数，如果copy_process()函数成功返回，新创建的子进程被唤醒并让其投入运行。内核有意选择子进程首先执行[6]。因为一般子进程都会马上调用exec()函数，这样可以避免写时拷贝的额外开销，如果父进程首先执行的话，有可能会开始向地址空间写入。

---

vfork 函数原型：pid_t　vfork(void);

特点：父子进程共享数据段，并且保证子进程先于父进程运行，在它调用 exec 或者_exit 时，父进程才会被运行；

僵死进程：　父进程未结束，子进程结束，并且父进程未获取子进程的退出状态。这种进程，进程主体空间已经释放，只有 PCB 还未释放。

处理办法：

a) 父进程中调用 wait 或 waitpid 获取子进程的退出状态，这种方式可能导致父进程在 wait 或 waitpid 调用出阻塞运行，直到子进程退出。

b) 父进程调用 signal(SIGCHLD,SIG_IGN)，来忽略 SIGCHLD 信号，这样子进程结束后会由内核释放资源。

c) 对子进程的退出捕获他们的退出信号 SIGCHLD,父退出信号时，在信号处理函数中调用 wait 或 waitpid 操作来释放他们的资源。

孤儿进程： 父进程结束，子进程未结束。孤儿进程会被系统守护进程 init 收养，并未他们完成状态收集工作。

守护进程：

1.守护进程有称精灵进程，常常在系统启动时自启，仅在系统关闭时才终止，生存期比较长！一般都是在后台运行。

可通过 ps -axj 命令查看常用系统守护进程，其中最为常见的 init 进程，负责各运行层次间的系统服务。

2.守护进程编程规则

（1）首先调用 umask(mode_t umask())函数将文件模式创建屏蔽字设置为 0；

（2）调用 fork(),然后使父进程退出（exit）；

（3）调用 setsid（）创建一个新会话；

（4）将当前目录更改为 根目录；

（5）关闭不再需要的文件描述符；

（6）某些守护进程打开/dev/null 使其具有文件描述符 0，1，2，这样任何一个进程就不会产生其他不好的效果；

参考书籍资料：

《Unix 环境高级编程》第 8 章第 3 节

《Unix 环境高级编程》第 8 章第 6 节 ----- wait&waitpid

《Linux 内核设计与实现》第 3 章第 2 节

《Unix 环境高级编程》第 13 章 ---- 守护进程

5、进程替换 --- exec 函数族

a) exec 函数的功能：调用它并没有产生新的进程，一个进程一旦调用 exec()函数，它本身就死亡了。就好比鬼上身了一样，身体还是你的，但是灵魂和思想已经被替换了-----系统把代码段替换成新的程序的代码，在这其中唯一保留的就是进程的 ID，对于系统而言，还是同一个进程，只不过是执行另一个程序罢了。

I.只有 fork()和 vfork()才能创建一个新的进程

II.在使用 exec()之前，首先要使用 fork(),创建一个子进程，子进程调用 exec()函数

b) exec()函数族中的函数

I.说明：

int execl（const char *path,const char *arg,...）；

*与 execv 函数的用法类似，只是在传递 argv 参数的时候每个命令行参数都声明为一个单独的参数（“.....”说明参数个数不确定），这些参数要以一个空指针结尾。

int execv（const char *path,char *const argv[]）；

*通过路径名方式调用可执行文件作为新的进程映像

int execle（const char *path,const char *arg,....,char *const envp[]）；

*与 execl 的用法类似

int execve(const char*pathname,const char *argv[],char *const envp[]);

*参数 pathname 是将要执行的程序的路径名，参数 argv,envp 与 main 函数的 argv,envp 对应

int execlp（const char *file,const char *arg,....）；

*与 execl 函数类似

int execvp（const char *file,char*const argv[] ）；

*与 execv 函数用法类似

II.区别

前四个取路径名做参数，后两个取文件名做参数

与参数表的传递有关（l 表示 list，v 表示矢量 vector）。函数 execl、execlp 和 execle 要求将新程序的每个命令行参数都说明为一个单独的参数，这中参数表以空指针结尾。而 execv、execve 和 execvp 则要先构造一个指向各参数的指针数组，然后将该数组的地址作为这三个函数的地址。

参考书籍资料：

《Unix 环境高级编程》第 8 章第 10 节

6、Linux 下文件操作函数与系统调用函数

Linux 系统下的文件操作函数：open read write close lseek stat

int open(char *path, int flag, /*mode_t mode*/)：打开一个普通文件，如果打开成功，返回文件描述符；

int read(int fd, void *buff, size_t size) ：按字节读取文件内容；

int write(int fd, void *buff, size_t size) ：按字节给文件中写入内容；

int close(int fd) ：关闭打开的文件；

int lseek(int fd, int size, int flag) ：移动文件读写偏移量。

三个 stat 函数获取文件的属性信息

```
int  stat(char  *path,  struct  stat  *st);
int  fstat(int  fd,  struct  stat  *st);
int  lstat(char  *path,  struct  stat  *st);
```

扫描目录：

opendir readdir telldir closedir seekdir

参考书籍资料：

《Unix 环境高级编程》第 3、4 章

《Linux 程序设计第四版》第 3 章第 8 节 扫描目录

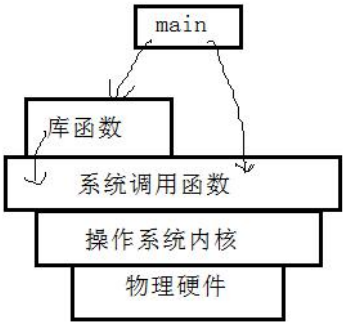
系统调用函数是系统内核抛出来给用户空间调用的接口，系统调用函数由用户态调用，在内核态执行。与之相对应的就是库函数了，库函数在函数库文件中实现，执行时只需要在用户态执行就可以了。

库函数与系统调用函数的区别：

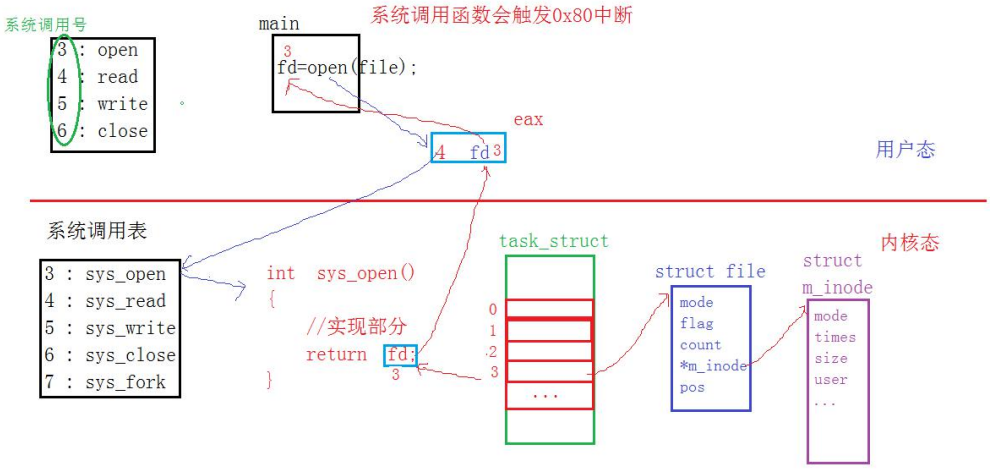
库函数： 在函数库文件中

系统调用函数： 在系统内核中实现

库函数有可能还需要转调系统调用函数，比如： fopen printf等。也有可能不需要转调系统调用函数，比如：strlen、strcpy等。



系统调用函数的实现原理： 以 open 函数为例



系统调用函数触发 0x80 中断，并且将系统调用号存储在 eax 寄存器中，然后陷入

内核，内核开始执行中断处理程序，在系统调用表中查找系统调用号对应的系统内核函数并且调用，执行完成后又将返回值通过 eax 寄存器传递回用户空间。

注意： 由于 fork 函数创建的子进程的 PCB 是拷贝父进程的，并且这块是浅拷贝，所以父子进程会共享父进程 fork 之前打开的所有文件描述符。

7、信号

信号是系统预先定义好的某些特定的事件，信号可以被产生，可以被接收，产生和接收的主体都是进程。信号有三种响应方式： 忽略 默认 自定义(捕获)

signal 函数可以修改信号的响应方式：

typedef void (*sighandle_t)(int);

sighandle_t (*signal)(int, sighandle_t);

忽略： SIG_IGN

默认： SIG_DFL

自定义： 自己写的信号处理函数

kill 函数可以向指定的进程发送指定的信号：

int kill(pid_t pid, int sig);

pid > 0 指定将信号发送个那个进程

pid == 0 信号被发送到和当前进程在同一个进程组的进程

pid == -1 将信号发送给系统上有权限发送的所有的进程

pid < -1 将信号发送给进程组 id 等于 pid 绝对值，并且有权限发送的所有的进程。

sig 指定发送信号的类型。

raise 函数将信号发送给自己：

int raise(int sig);

相当于： kill(getpid(), sig);

信号集

在实际的应用中一个应用程序需要对多个信号进行处理，为了方便，linux 系统引进了信号集的概念。信号集用多个信号组成的数据类型 sigset_t. 可用以下的系统调用设置信号集中所包含的数据。

int sigemptyset(sigset_t *set); 将 set 集合置空, 成功则返回 0，失败返回-1.

int sigaddset(sigset_t *set,int signo)将 signo 信号加入到 set 集合,成功则返回 0，失败返回-1.

int sigdelset(sigset_t *set，int signo)；从 set 集合中移除 signo 信号,成功则返回 0，失败返回-1.

int sigismember(const sigset_t *set,int signo)；signo 判断信号是否存在于 set 集合中,若真返回 1，假返回 0，失败返回-1.

信号集的使用

int sigprocmask(int how,const sigset_t *set,sigset_t *oset);用于检测或更改其信号屏蔽字，包含头文件<signal.h>,成功则返回 0，失败则返回 1.

参数:how 指示如何修改屏蔽信号

set 是一个非空指针时，根据 how 修改屏蔽信号

oset 是一个非空指针时，存放当前屏蔽信号集

若 set 为 NULL，不改变该进程的信号屏蔽字，how 也无意义

参考书籍资料：

《Unix 环境高级编程》第 10 章

《Linux 程序设计第四版》第 11 章第 4 节

8、进程间通讯 -- 管道　信号量　消息队列　共享内存

A.管道(pipe)

实现进程间通讯的原理：

就像现实中管道的两端一样，由一个进程进行写操作，其余的进程进行读操作。如果管道为空，那么 read 会阻塞；如果管道为满则 write 会阻塞。

分类：管道可以分为有名管道和无名管道两类。

区别：

I.有名管道：可以在任意进程之间进行通讯，通讯是双向的，任意一端都可读可写，但同一时间只能一端读，一端写。

II.无名管道：只能在具有亲缘关系的进程（父子进程）间通讯，不能在网络间通讯，并且是单向的，只能一端读另一端写。

特点：通讯数据遵循先进先出的原则，并且都是半双工通讯的管道。

B.消息队列(message queue)

特点:

I.是消息的链表。具有特定的格式，存放在内存当中，由消息队列标识符标识。

II.消息队列允许一个或者多个进程向他写入与读取消息。

III.消息队列可实现消息的随机查询，不一定要以先进先出的顺序读取，也可以按照类型进行读取。

相关函数

msgget()：用来创建或访问一个消息队列；

msgsnd()：用来把消息添加到消息队列中；

msgrcv()：从一个消息队列中获取消息；

msgctl()：作用与共享内存的控制函数相似；

C.信号量(semaphore)

a. 概念：用来同步进程的特殊变量；一个特殊的计数器，大于 0 时记录资源的数量，小于 0 时，记录等待资源的进程的数量。当信号量的值大于 0 时，进程总是可以获取到资源并使用，小于 0 时，进程必须阻塞等待有其他进程释放资源。

b. 操作方式：对信号量进行操作使用 p v 操作，p v 操作都是原子操作；

*p 操作：获取资源，信号量的值减 1；

*v 操作：释放资源，信号量的值加 1；

相关函数：

semget()：创建或者获取信号量的内核对象

semop()：完成对信号量的 p 操作或 v 操作

semctl()：设置信号量属性

注意： 操作系统对进程间通讯用的信号量在内核中都是以信号量集管理的，即就是通过 semget 函数获取到的是信号量集的标识符。其他函数操作时，必须指明操作的是那个信号量集中的那个信号量。类似于数组的下标。

D.共享内存(shared_memory)

实现原理：共享内存区域说白了就是多个进程共享的一块物理内存地址，只是将这块物理内存分别映射到自己的虚拟空间地址上。假设有 10 个进程将这块区域映射到自己的虚拟地址上，那么，这 10 个进程间就可以相互通信。由于是同一块区域在 10 个进程的虚拟地址上，当第一个进程向这块共享内存的虚拟地址中写入数据时，其他 9 个进程也都会看到。因此共享内存是进程间通信的一种最快的方式。但是进程

之间使用这块共享空间时，必须做同步控制。

相关函数

shmget()：创建共享内存；

shmat()：启用对第一个创建的共享内存的访问；

shmdt()：将共享内存从当前进程中分离；

shmctl()：共享内存的控制函数；

参考书籍资料：

《Unix 环境高级编程》第 15 章

《Linux 程序设计第四版》第 13、14 章

9、线程基础

a) 线程的概念： 线程是进程内部的一条执行序列，或者执行流，每一个进程至少有一条线程，称之为主线程，从代码角度看，就是 main 函数的函数体，在主线程中可以通过线程库创建其他线程(函数线程)。主线程和函数线程会同时向下运行。

b) 线程的实现方式：用户级、内核级、混合模式

用户级：线程的创建、销毁、管理都在用户空间完成，内核只会识别为一个进程，一条线程。优点是灵活性，操作系统不知道线程存在，在任何平台上都可以运行；线程切换快，线程切换在用户空间完成，不需要陷入内核；不用修改操作系统，实现容易。缺点是：编程复杂，用户必须自己管理线程，包括线程调度；如果一个线程阻塞，整个进程都会阻塞；不能使用对称多处理器。

内核级：线程的创建、销毁、管理由操作系统内核完成。内核线程使得用户编程简单，但是每次切换都得陷入内核，所以效率较低。

混合模式：即就是一部分以用户级线程创建，一部分由内核创建，是一个多对多的关系。结合用户级和内核级的优点。

参考书籍资料：

《从哲学层次上看操作系统》第 5 章

c) 线程与进程的区别

1、线程是系统调度的最小单位，进程是资源分配的最小单位。

2、线程创建、管理代价小。

10、线程库的使用 -- 线程创建

线程库包含在头文件 pthread.h 中：

线程创建函数： int pthread_create(pthread_t *id, pthread_attr_t *attr,
void*(*pthread_fun)(void*), void*arg);

pthread_create 函数会创建出一条新的函数线程，线程从 pthread_fun 函数入口地址开始执行，到 pthread_fun 函数结束。arg 参数为给 pthread_fun 函数传递的参数。attr 为线程属性。pthread_create 函数成功返回 0，失败返回错误码。

线程结束函数： int pthread_exit(void *);

注意主线程结束默认调用 exit 函数，这个函数会结束进程。进程结束，所有的线程也会随之结束。

等待线程结束函数： int pthread_join(pthread_t id, void **);

终止一个线程： int pthread_cancel(pthread_t id);

由于线程是进程中的一条执行序列，所以一个进程中的所有线程共享全局、堆区数据，包括打开的文件描述符。只有每个线程栈区的数据是线程独享的。

参考书籍资料：

《unix 环境高级编程》第 11 章

《Linux 程序设计第四版》第 12 章第 3 节

《Linux 高性能服务器编程》第 14 章第 2 节

11、线程同步 --- 信号量 互斥锁 条件变量

a) 信号量

这里和进程间用的信号量作用相似，当线程访问一些有限的共享资源时，就必须做到线程间同步访问。

信号量的使用方式：

#include <semaphore.h>

初始化 int sem_init(sem_t *sem, int shared, int val);

信号量 sem 一般被定义在线程共享的全局数据区，sem_init 函数是将信号量 sem 的初始值设置为 val，shared 参数控制这个信号量是否可以在多个进程之间共享，但是 Linux 对此不支持。

P 操作： int sem_wait(sem_t *sem);

对信号量 sem 进行-1 操作，如果结果小于 0，则此函数调用会阻塞，直到有其他线程执行 V 操作。

V 操作： int sem_post(sem_t *sem);

对信号量 sem 进行+1 操作。

销毁： int sem_destroy(sem_t *sem);

销毁信号量

参考书籍资料：

《Linux 程序设计第四版》第 12 章第 5 节第 1 小节

b) 互斥锁

互斥锁是只能在线程之间使用的一种控制临界资源访问的机制，如果一个线程要访问临界资源，则必须先加锁，用完之后解锁。这样，在一个线程访问临界资源的过程中，其他线程加锁就会阻塞，不能进入访问临界资源的临界区，直到访问临界资源的线程用完后并解锁。

互斥锁的使用方式：

#include <pthread.h>

初始化： int pthread_mutex_init(pthread_mutex_t *mutex,
pthread_mutexattr_t *attr);

互斥锁 mutex 一般被定义在线程共享的全局数据区，此函数是初始化互斥锁 mutex，attr 为锁的属性。

加锁： int pthread_mutex_lock(pthread_mutex_t *mutex);

或 int pthread_mutex_trylock(pthread_mutex_t *mutex);

解锁： int pthread_mutex_unlock(pthread_mutex_t *mutex);

销毁锁：int pthread_mutex_destroy(pthread_mutex_t *mutex);

参考书籍资料：

《Linux 程序设计第四版》第 12 章第 5 节第 1 小节

《Unix 环境高级编程》第 11 章第 6 节、第 12 章第 4 节

12、线程安全 --- 可重入函数

因为线程之间共享全局数据、静态数据。在编程的过程中，我们必须对线程访问的这些共享资源做一些同步控制，但是有些系统调用或者库函数在实现时，用到了静态的数据，当在多线程环境中调用到这些函数时，就会出现不安全的现象。对于这些函数，在多线程环境中必须使用它们的安全版本，即就是可重入版本。例如：字符串分割函数 char* strtok(char *sourstr, const char *flag) 的可重入版本为： char * strtok_r(char *sourstr, const char *flag, char **res);

不能保证线程安全的函数有：

表12-5 POSIX.1中不能保证线程安全的函数

asctime	ecvt	gethostent	getutxline	putc_unlocked
basename	encrypt	getlogin	gmtime	putchar_unlocked
catgets	endgrent	getnetbyaddr	hcreate	putenv
crypt	endpwent	getnetbyname	hdestroy	pututxline
ctime	endutxent	getnetent	hsearch	rand
dbm_clearerr	fcvt	getopt	inet_ntoa	readdir
dbm_close	ftw	getprotobyname	l64a	setenv
dbm_delete	gcvt	getprotobynumber	lgamma	setgrent
dbm_error	getc_unlocked	getprotoent	lgammaf	setkey
dbm_fetch	getchar_unlocked	getpwent	lgammal	setpwent
dbm_firstkey	getdate	getpwnam	localeconv	setutxent
dbm_nextkey	getenv	getpwuid	localtime	strerror
dbm_open	getgrent	getservbyname	lrand48	strtok
dbm_store	getgrgid	getservbyport	mrand48	ttyname
dirname	getgrnam	getservent	nftw	unsetenv
dlerror	gethostbyaddr	getutxent	nl_langinfo	wcstombs
drand48	gethostbyname	getutxid	ptsname	wctomb

参考书籍资料：

《Unix 环境高级编程》第 12 章第 5 节

13、线程中 fork 使用及其锁的继承

线程中 fork 的使用：

在多线程程序中，某一条线程调用 fork 生成子进程，在子进程中，只有调用 fork 函数的线程会被启动，其他线程不会启动（不会运行）。

子进程会继承父进程中的锁，及其状态。所以，有可能子进程发生死锁！！

解决方案：在 fork 之前调用 pthread_atfork 函数。

第三部分：网络编程

1、OSI 七层模型&TCP/IP 四层模型

OSI 七层模型：

应用层 ---------- 为应用程序提供服务

表示层 ---------- 数据格式转化，数据加密

回话层 ---------- 建立，维护和管理会话

传输层 ---------- 建立，维护和管理端到端的链接，控制数据传输的方式

网络层 ---------- 数据传输线路选择，IP 选址及路由选择

数据链路层 -------- 提供介质访问和链路管理

物理层 ---------- 以二进制形式在物理媒介上传输数据

TCP/IP 四层模型及主要协议：

应用层 传输层 网络层 数据链路层

2、TCP 编程流程及 TCP 协议特点

TCP 协议特点： 面向连接 可靠的 流式服务

编程流程：

服务器端： socket bind listen accept recv/send close

客户端： socket connect send/recv close

注意： 标识网络中通讯主角----进程的方式是：IP 地址+端口号。在编程中给定网络中进程地址的结构为：

```
struct  sockaddr_in
{
    sa_family_t  sin_family;
    short        sin_port;   //  htons  将主机字节序转网络字节序  5000 以上
    struct  in_addr  sin_addr;
};
struct  in_addr
{
    u_int32_t  s_addr;  //  将字符串表示的点分十进制转换为 int 类型
                        //    inet_addr   inet_aton
}
```

TCP 编程接口函数：

```
int bind(int listenfd,  struct sockaddr*addr,  int  addrlen);

int  listen(int  listenfd,  size_t  size);

int  accept(int  listenfd,  struct sockaddr *cliaddr,  int  *len);

int  recv(int  linkfd,  void  *buff,  size_t  size,  int  flag);

int  send(int  linkfd,  void *buff,  size_t  size,  int  flag);

int  close(int  fd);

int  connect(int sockfd,  struct sockaddr *seraddr,  int  serlen);
```

3、TCP 三次握手 四次挥手 状态图

协议特点： 无连接 不可靠 数据报服务

编程流程：

服务器： socket bind recvfrom/sendto close

客户端： socket recvfrom/sendto close

```
int recvfrom(int sockfd, void *buff, size_t len, int flag
                (struct sockaddr*)src_addr, int *addr_len);
int sendto(int sockfd, void *buff, size_t len, int flag
                (struct sockaddr*)dest_addr, int addr_len);
```

4、TCP UDP IP 比较

TCP 流式服务

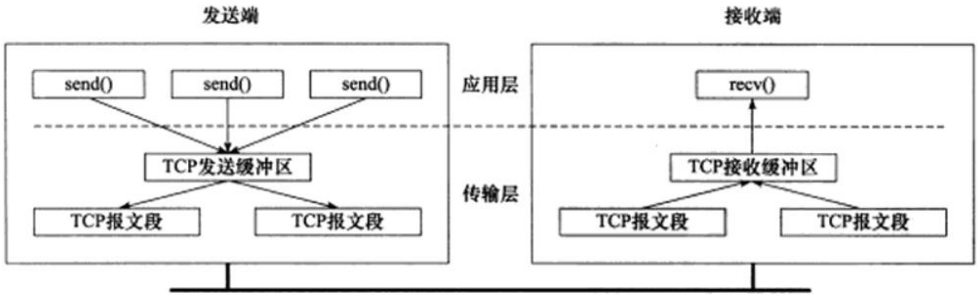


图 3-1 TCP 字节流服务

流式服务：发送端 send 的次数，与接收端 recv 的次数没有直接联系，并且 send 只是将数据写到发送缓冲区中，recv 只是从接受缓冲区中获取数据。如果 recv 一次调用将一次发送的数据获取不完，下次 recv 接着获取。

UDP 数据报服务的特点：

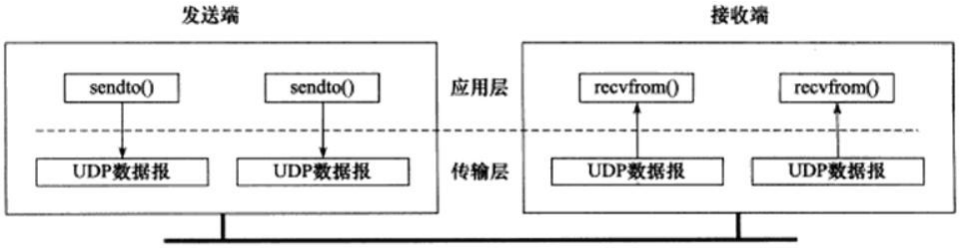


图 3-2 UDP 数据报服务

数据报服务的特点：

sendto 发送数据和对端 recvform 接受数据的次数相等。recvfrom 接受数据时必须将 sendto 发送的数据一次接受完，否则，数据丢失。sendto 一次发送的数据的长度应该小于等于 recvfrom 读取数据的大小。

各协议的特点：

TCP	面向连接	可靠的	流式服务
UDP	无连接	不可靠	数据报服务
IP	无连接	不可靠的	无状态

无状态： 数据的发送、传输、接收相互独立的，没有上下文关系。接收端接收的数据有可能重复和乱序。

IP 协议报头：

图 2-1　IPv4 头部结构

TCP 报头：

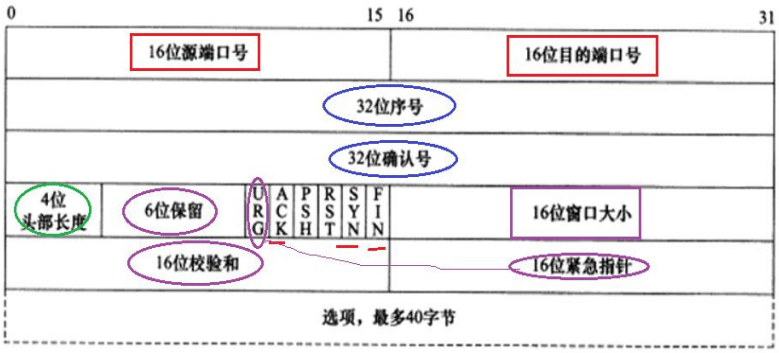


图 3-3　TCP 头部结构

UDP 报头：

5、超时重传、滑动窗口、慢启动、拥塞避免、快速重传、快速恢复

超时重传： TCP 协议在发送数据以后，每一个报文段都会有一个定时器，在定时器指定的时间内接收端对于这个报文段的确认报文如果没有到达，则会重新发送一次，并且这次的定时器时间为上次的两倍。

滑动窗口：TCP 流量控制的一种手段。这里说的窗口是指接收通告窗口(Receiver Window, RWND)。它告诉对方本端的 TCP 接收缓冲区还能容纳多少字节的数据，对方就可以控制发送数据的速度。

慢启动、拥塞避免、快速重传、快速恢复 参考下面的博客：

http://www.cnblogs.com/fll/archive/2008/06/10/1217013.html

6、DNS & Http & Https

DNS 协议： 域名解析协议

解析方式： 递归方式 迭代方式

http 协议（端口号为 80）

特点：

A1：支持客户/服务器模式;

A2：简单快速。客户向服务器请求服务时，只需传送请求方法和路径。

A3：灵活。允许传输任意类型的数据对象。

A4：无连接。其含义是限制每次连接只处理一个请求。服务器处理完客户的请求并收到客户的回应后，断开连接。

A5：无状态，即对处理事务没有记忆力；

*常用的请求方法有：

GET：请求获得资源

POST：请求附加新的数据

PUT：请求服务器存储一个资源

http 协议的响应

响应由三个部分组成：状态行，消息报头，响应正文

常用的应答状态码：

1XX：指示信息---表示请求已接收，继续处理；

2XX：成功------表示请求已经被成功接收，理解，接受

3XX：重定向-----要完成请求必须进行更近一步的操作

4XX：客户端错误----请求有语法错误或者请求无法实现

*400----客户端请求有语法错误，不能被服务器理解；

*401-----请求未经授权

*403-----服务器收到请求，但拒绝提供服务

*404-----请求资源不存在

5XX：服务器端错误----服务器未能实现合法的请求

*500---服务器发生不可预期的错误

*503----服务器当前不能处理客户端的请求，一段时间后可能恢复正常。

http 和 https 协议（网络协议）的区别:

A.https 协议需要申请 ca 证书，一般免费证书很少，需要交费；

B.http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 SSL 加密

传输协议;

C.http 和 https 使用的是完全不同的连接方式，用的端口也不一样（http：80；https：443）；

D.http 协议的连接很简单，是无状态的；

E.https 协议是由 SSL+HTTP 协议构成的可进行加密传输，身份认证的网络协议。比 http 协议安全。

第四部分：高性能服务器编程

1、多进程、多线程

多进程编程：

思想： 父进程负责监听，并接收客户连接（accept）
fork 创建子进程，子进程处理与客户端通讯

注意：

1、父子进程之间共享文件描述符，所以父进程不需要将接收链接的文件描述符传递给子进程。

2、父进程要关闭链接的文件描述符！！ 为什么？？？

原因：

1、父进程不关闭文件描述符，则后续创建的子进程会将所有的文件描述符继承下来。

2、 父进程不关闭文件描述符，则后续的链接的文件描述符不断增大，链接的客户端的数量就受一个进程最多打开的文件的限制。

多线程编程

与多进程编程对比：

1、创建多进程会消耗大量的系统资源

2、如果子进程在很短的时间内结束，系统的负担会加重

多线程：

创建线程资源消耗相对较小

线程之间数据共享更容易

线程结束释放资源比较少

思路： 主线程负责监听，并接受客户端链接
函数线程负责和客户端通讯

注意：

1、 主线程接受连接， 链接的文件描述符如何传递给函数线程？
文件描述符必须通过创建函数线程时值传递给函数线程

2、主线程能不能关闭文件描述符？？ 不可以

2、进程池、线程池

池： 初始时，申请比刚开始要使用的资源大的多的资源空间。接下来使用时，直接从池中获取资源。

线程池：

多线程存在如果客户端链接，创建一个新的线程，客户端关闭，释放线程。服务器更多时间消耗在创建线程、释放线程。对于业务逻辑的处理，就会较少。所以，我们可以用线程池来改善问题。

线程池就是在服务器运行初始时，创建 n 个线程，将这个 n 个线程用池管理起来，当有用户连接时，从线程池中选取一个线程为其服务，客户端关闭以后，服务器就将线程又放回到池中。

线程池的实现：

主线程执行先创建 3 条线程，

主线程等待客户连接，3 条函数线程因为信号量的 P 操作阻塞运行。

主线程接受到客户连接后，通过信号量的 V 操作通知一个函数线程和客户端通讯。

主线程怎样将连接的文件描述符传递给函数线程？？

全局数组作为等待函数线程处理的文件描述符的等待队列

3、I/O 复用 select poll epoll

3.1 select

函数原型： int select(int nfds, struct fd_set *readfds, struct fd_set *writefds, struct fd_set *execptfds, struct timeval * timeout);

nfds: 监听的最大文件描述符值+1；

readfd,writefd,exceptfds 分别指向可读可写和异常事件对应的文件描述符的集合。

timeout 设置 select 函数的超时时间。

返回值： 返回就绪的文件描述符数。

应用程序在调用 select 函数时，通过 readfd,writefd,exceptfds 三个参数传入自己感兴趣的文件描述符。Select 函数调用返回时，内核将修改他们来通知应用程序那些文件描述符已经就绪。fd_set 结构体只包含一个整形数组，数组每一位表示一个文件描述符。

```
#include <sys/select.h>
FD_ZERO( fd_set *fdset );              /* 清除 fdset 的所有位 */
FD_SET( int fd, fd_set *fdset );       /* 设置 fdset 的位 fd */
FD_CLR( int fd, fd_set *fdset );       /* 清除 fdset 的位 fd */
int FD_ISSET( int fd, fd_set *fdset ); /* 测试 fdset 的位 fd 是否被设置 */
```

描述符就绪条件：

a.满足下列条件之一，套接字准备好读

a1:套接字接收缓冲区当中的数据字节数大于等于套接字接收缓冲区中设置的

最最小值。（对于 TCP 和 UDP 来说默认值为 1）

a2：改连接的读半部关闭（也就是接收了 FIN 的 TCP）；

a3：该套接字是一个监听套接字，且已完成的连接数不为 0；对于这样的套接字，accept 通常不会阻塞；

a4：其上有一个套接字错误处理；

b. 满足下列任意一个条件时，套接字准备好写

b1：该套接字发送缓冲区中可用空间的大小大于等于套接字发送缓冲区当中设置的最小值时，并且或者该套接字已经连接，或者套接字不需要连接（UDP）；

b2：该连接的写半部关闭；

b3：使用非阻塞式的 connect 的套接字已建立连接，或者 connect 已经以失败告终；

b4：其上有一个套接字待错误处理；

select 的缺点：

a. 每次调用 select，都需要把 fd 集合从用户态拷贝到内核态，这个开销在 fd 很多时会很大；

b. 文件描述符就绪时，内核会修改 readfds、writefds、execptfds 结构，所以每次调用 select 之前，必须重新将文件描述符注册一遍。

c.每次调用 select 都需要在内核遍历传递进来的所有 fd,这个开销在 fd 很多时也很大；（时间复杂度为 O(N)）；

d.单个进程能够监视的文件描述符存在最大的限制。

3.2 poll

函数原型： int poll(struct pollfd *fds, int nfds, int timeout);

```
struct pollfd
{
    int fd;          // 用户关注的文件描述符
    short events;    // 用户关注的事件类型
    short revents;   // 由内核填充
}
```

poll 函数和 select 相同，也是在一定时间内轮询关注的文件描述符，测试其是否就绪。

poll 的优点：

a、将用户关注的文件描述符的事件单独表示，可关注更多的事件类型；

b、将用户传递的和内核修改的分开，每次调用 poll 之前，不需要重新设置。

c、poll 函数没有最大文件描述符的限制

poll 的缺点：

a、每次调用都需要将用户空间数组拷贝到内核空间；

b、每次返回都需要将所有的文件描述符拷贝到用户空间数组中，无论是否就绪；

c、返回的是所有的文件描述符，搜索就绪文件描述符的时间复杂度为 O(n)

3.3 epoll

I.epoll 函数的创建：int epoll_create(int size);

size 只是给内核一个提示，告诉它需要多大的事件表；

II.epoll 内核事件表的操作：

int epoll_ctl(int epdf,int op,int fd,struct epoll_event*event);

fd 为要操作的文件描述符；

op 指定操作类型（往事件表中注册，修改，删除 fd 上的注册事件）；

event 指定事件；

III.epoll 系列系统调用的主要接口：

int epoll_wait(int epfd,struct epoll_event *events,int maxevents,int timeout);

此函数如果检测到事件，就将所有就绪事件从内核事件表中（由 epfd 中的参数指定）复制到它的第二个参数 events 指定的数组中，这个数组只输出 epoll_wait 检测出的就绪事件。所以，搜索就绪文件描述符的时间复杂度为 O(1).

LT 模式：在数据到达后，无论程序是没有接收，还是接收了，但没有接收完，下一轮 epoll_wait 仍然会提醒应用程序该描述符上有数据，直到数据被接收完；

ET 模式：在数据到达后，无论程序是没有接收，还是接收了，但没有接收完，都只提醒一次，下一轮不在提醒应用程序该描述符上有数据。

所以，要求程序在收到提醒时必须将数据接收完，否则将会出现丢掉数据的可能。

三个函数的区别：

表 9-2 select、poll 和 epoll 的区别

系统调用	select	poll	epoll
事件集合	用户通过 3 个参数分别传入感兴趣的可读、可写及异常等事件，内核通过对这些参数的在线修改来反馈其中的就绪事件。这使得用户每次调用 select 都要重置这 3 个参数	统一处理所有事件类型，因此只需一个事件集参数，用户通过 pollfd.events 传入感兴趣的事件，内核通过修改 pollfd.revents 反馈其中就绪的事件	内核通过一个事件表直接管理用户感兴趣的所有事件。因此每次调用 epoll_wait 时，无须反复传入用户感兴趣的事件。epoll_wait 系统调用的参数 events 仅用来反馈就绪的事件
应用程序索引就绪文件描述符的时间复杂度	O(n)	O(n)	O(1)
最大支持文件描述符数	一般有最大值限制	65 535	65 535
工作模式	LT	LT	支持 ET 高效模式
内核实现和工作效率	采用轮询方式来检测就绪事件，算法时间复杂度为 O(n)	采用轮询方式来检测就绪事件，算法时间复杂度为 O(n)	采用回调方式来检测就绪事件，算法时间复杂度为 O(1)

第三部分&第四部分参考书籍资料：

《Linux 高性能服务器编程》全书