Linux C编程

笔记本：	C++		
创建时间：	2015/12/15 21:34	**更新时间：**	2015/12/18 9:15
标签：	C++Base		

Linux C编程

2015年12月15日
21:34

- **一 gcc，gdb和make**

1，gcc处理文件扩展名

扩展名	文件类型
.c,.C	C语言
.cxx,.cc	C++语言
.m	面向对象的C
.i	预处理之后的C语言源文件
.ii	预处理之后的C++源文件
.s,.S	编译生成的汇编语言文件
.h	头文件
.o	编译生成的目标代码文件，机器代码
.a	库文件

2，gcc常用模式

- 源文件直接编译成可执行程序：**gcc －o test test.c**。test.c为C源文件，-o为输出选项。多个源文件时可以并列多个源文件，所有文件里必须有且只有一个main函数。
- 源文件生成目标文件（即机器代码）：**gcc －c test.c －o test.o**。
- **Gcc –wall**：输出检测到的警告。
- 编译C++程序时使用的是g++，而不是gcc

3，gcc常用选项

常用选项	意义
-x langage	标识源文件所有的语言
-x none	把上面的langage都关掉
-c	编译成目标代码，机器代码
-C	同-E配合，让预处理的结果把注释保留
-S	编译成汇编代码
-E	输出预处理的结果，不进行编译
-o file	指明输出文件名为file
-v	把整个编译过程的输出信息打印出来
-include file	相当于#include<file>语句
-Idir	宏include需要搜寻的目录
-Ldir	搜寻库文件的路径
-ansic	设置程序使用ANSIC标准
-I目录	添加头文件的路径

4, gdb调试器

功能：启动需要调试的程序，自定义方式运行；设置断点；程序暂停时的变量检查；动态改变程序的执行环境。**需要调试的程序在编译时(gcc)要添加-g选项**。

- **启动选项：gdb 执行文件 [core | PID]**:core表示调试一个运行程序和core文件，core是程序非法执行产生的文件。PID指定服务程序的进程号或调试正在运行的程序。其他选项：-s <file>:从指定文件读取

符号表;-se file :读取符号表信息，并用在可执行文件中；-c <file>:调试core文件；-d <directory>加入一个源文件的搜索目录。

- **帮助信息**：help：列出命令种类；help <class>:产看种类中的命令；help breakpoint：查看断点相关；help <command>查看命令。
- Shell <**命令**>:在gdb中执行shell命令。

5，运行程序(r)

- **程序运行参数**：set args；show args：指定运行时的参数。
- **运行环境**：path <**路径**>设定程序的运行路径；show paths查看程序运行路径；set/show environment **变量名**[=**值**] 设置环境变量。
- 工作目录：cd <dir>切换目录；pwd查看当前目录
- **程序输入输出**：info terminal显示程序使用的终端模式；>重定向控制输出；tty写输入输出的终端设备。Info program查看程序是否在运行，以及进程号和暂停的原因。

6，暂停/恢复运行

- 设置断点：break **函数/行号**；break +/-offset在当前行号前/后停住。Break **文件名.行号**：在源文件的某行停住。Break if
- 设置观察点：watch <expr>变量值变量时停住程序；rwatch <expr>当变量被读时停住；awatch <expr>当变量被读写时停住；info watchpoint列出当前设置的所有观察点。
- 设置捕捉点：catch <event>当event(throw或catch的异常)发生时停住程序；tcatch <event>捕捉点只起一次作用。
- t设置在函数上的停止点；clear <**文件:行号**>清除设置在该行的停止点；delete [**断点号**][**断点号范围**]删除指定断点；disable/enable [**断点号**][**断点号范围**]禁用/启用指定断点。
- 恢复程序运行和单步调试：continue/c/fg [**忽略断点次数**]恢复运行；step <count>单步跟踪，会进入函数调用；next <count>单步跟踪，不会进入函数调用；finish运行程序直到当前函数完成返回。

7，其他命令

- 查看栈信息：backtrace/bt当前函数调用栈的信息；bactrace/bt <n>n为正，则打印栈顶n层信息；n为负，则打印栈底n层信息。
- 查看源代码：list **行号/函数**查看某行或某函数的代码；list当前行后的代码；list - 当前行前的代码。Search <regexp>向前搜索；forward- search <regexp>向后搜索；reverse-search <regexp>全部搜索；
- 指定源文件路径：dir <**路径**>加一个源文件路径到当前路径；directory清除所有自定义的路径；show directories显示定义的路径；
- 查看运行时数据：print/<f> <expr>,f表示输出格式，比如/x。
- 修改变量值：print **变量名=值**；
- 跳转执行:jump <**行号**>/<**代码行内存地址**>指定下一条语句的运行点；最好在同一函数内跳转；
- 产生信号量：signal <**信号**>信号在1~15之间，产生一个信号量给被调试的程序。
- 强制函数返回:return [<expression>]强制函数忽略没有执行的语句返回，表达式的值会被认为函数返回值。
- 强制调用函数：call <expr>轻质调用函数并显示函数的返回值。

8，make工程管理器

可以根据时间戳自动发现更新过的文件而减少编译工作量，通过读入**makefile**文件内容执行大量编译工作。**Make target** 输入目标名即可建立指定目标。Make读取makefile后会建立一个描述数据库，记录了各个文件间的相互关系和他们的关系描述。

9，makefile描述规则

```
Target...: prerequisites...
    Command
    .....
```

- **target**为最后要生成的文件名或中间过程文件名，也可以是一个make动作如clean，称为伪目标。**Preeminents** 规则的依赖，生成规则目标所需要的文件名列表。**Command** 规则的命令行，任意的shell命令或可在shell下执行的程序。每行命令必须以[tab]字符开头。
- 较长的行用\分解为多行，注意\之后不能有空格。
- 在shell下输入make，make会读取**当前目录下**的makefile文件并将文件中第一个目标作为终极目标。
- 指定变量：可以定义一个变量(object)代替文件列表，在使用文件列表时用**$(object)**表示。
- Makefile中的系统变量：**$^**——所有的依赖文件；**$@**——生成目标；**$<**——第一个依赖文件；@——取消回显。
- 自动推到规则：make编译.c源文件时，编译源文件命令可以不明确给出，make会自动完成cc -c的任务，这样简化makefile

```
#samlpe MakeFile
edit:main.o kbd.o command.o display.o insert.o search.o files.o\ utls.o
    cc -o edit main.o kbd.o command.o display.o insert.o search.o\
    files.o utls.o
main.o:main.c defs.h
    cc -c main.c
kbd.o:kbd.c defs.h command.h
    cc -c kbd.c
command.o:command.c defs.h command.h
    cc -c command.c
display.o:display.c defs.h buffer.h
    cc -c display.c
insert.o:insert.c defs.h buffer.h
    cc -c insert.c
search.o:search.c defs.h buffer.h
    cc -c search.c
files.o:files.c defs.h buffer.h command.h
    cc -c files.c
utls.o:ults.c defs.h
    cc -c ults.c

clean:
    rm edit main.o kbd.o command.o desplay.o insert.o search.o\ files.o
    utls.o
```

```
#简化后的makefile
Objecs= main.o kbd.o command.o desplay.o insert.o search.o\ files.o
utls.o
Edit:$(objects)
    Cc -o edit $(objects)
main.o: defs.h
kbd.o: defs.h command.h
command.o: defs.h command.h
display.o: defs.h buffer.h
insert.o: defs.h buffer.h
search.o: defs.h buffer.h
files.o: defs.h buffer.h command.h
utls.o: defs.h

.PHONY:clean    --声明为伪目标
Clean:
    -rm edit $(objects)    --前面加-表示忽略rm的执行错误
```

二 linuxC常用类型和修饰符

1，**sizeof**（）以字节为单位返回大小；**typeof 类型 别名**：为一个已有的类型设置别名；逗号表达式的值为最后一个表达式的值。

2，**printf**（）：%d十进制；%o八进制；%x十六进制；%u无符号十进制；%f单双精度，6位小数；%e 指数形式浮点数；%g自动选择%f或%e格式。

- %mds字符串占m列，若串长大于m，则全部输出；若串长小于m，左补空格。%-ms 串长小于m，字符串向左靠，右补空格。
- %m.ns：输出占m列，只取字符串左端n个字符。

3,变量存储类型

- **Auto变量**:不专门声明为static的局部变量都是动态分配空间的，调用函数时分配，调用结束自动释放。函数形参和函数内变量都是auto变量。**Auto int b**；
- **Static变量**：函数调用结束不消失仍保留原值。静态局部变量在函数调用后仍然存在，但不能引用它。
- **Register变量**：也称寄存器变量，将局部变量的值放在CPU寄存器中，提高了存取速度。**Register int b**。如果没有可以分配的寄存器，编译器就可能使用内存空间。
- **Extern变量**：在函数外定义的全局变量，到文件末尾。Extern可以再一个文件中声明，也可以在多个文件中声明(其作用域为多文件)。**Extern 变量**，即可使用别的文件里同名的变量。某些外部变量只限于本文件使用，不能被其他文件引用，可以再定义外部变量时加static：**file1.c static int A;**

4,指向指针的指针：指向指针数据的指针变量。定义：**类型******名称**；

函数指针：指向函数的指针，用来存放函数的入口地址，把哪个函数的地址赋给它，它就指向哪个函数。定义：**类型 (*名称)()。赋值时只需给出函数名。**

5,共用体

Union 名称{ 类型 成员名....}；在某一个时刻，只有一个成员的值有意义，定义变量分配空间时按照最大的数值类型分配，共用体比结构体要节省空间，但是**访问速度慢**。

- 两个相同类型的共用体变量可以相互赋值
- 共用体变量可以作为参数传递给被调用函数，可以作为函数的返回值。
- 共用体在初始化时只需要对其中一个成员赋值。

6，修饰符

- **Const**：定义类型常量。**Const int*a** 为指向常量的指针，指针的指向可以修改；**int *const a** 为常指针，指针的指向不能修改。
- **Volatile**：表名某个变量值可能在外部被修改，对该变量的存取不能缓存到寄存器，每次使用要重新存取。即不能依赖变量值。
- **Short和long**：**short int** 使int和char具有相同大小的位；**long int** 使int和正常的int一样的位。主要用来统一位数。

三 LinuxC预处理器和重要函数

1,预处理器

- **预定义宏**：**#define 标 识符 字符串**；**#define 标识符(参数列表) 字符串**；宏定义可以是单个字符串，也可以是函数代码，可以提高执行的效率。解除宏定义：**#undef 标识符**。被解除的宏定义，在该语句之后不再起作用。
- **文件包含**：**#include <文件名>;#include "文件名"**.也可以将自己常用的符号常量，类型定义和带参数的宏定义、自编函数放到.h文件中，用#include引用。
- **条件编译**：在编译带有条件编译的文件时，用**-D +参数**来给文件传

递判断条件，以完成条件编译。条件表达式中不包含sizeof()、强制类型和枚举常量。

#ifdef 标识符 语句1 #else 语句2 #endif	#if 条件表达式 语句1 #else 语句2 #endif

- **行号控制：#line number[“filename”]**,用于表明文件中当前的行号，主要用于调试和其他特殊用途。

2,字符串处理

字符测试函数：<ctype.h>

Isalnum	是否是英文或数字	Int isalnum（int c）
Isalpha	是否是英文字母	Int isalpha（int c）
Isascii	是否是ASCII码字符	Int isascii（int c）
Isdigit	是否是数字	Int isdigit（int c）
Iscntrl	是否是ASCII码的控制字符	Int iscntrl（int c）
Isgraph/isprint	是否是可打印字符	Int isgraph（int c）
Islower	是否为小写字母	Int islower（int c）
Isupper	是否是大写字母	Int isupper（int c）
Isspace	是否是空格	Int isspace（int c）
Inpunct	是否是标点或特殊字符	Int ispunct（int c）
Isxdigit	是否是十六进制数字	

字符串转换函数：<stdlib.h>

Double **Atof**（char*）	字符串转换为浮点数
Int **Atoi**(char*)	字符串转换为整型
Long **Atil**(char*)	字符串转换为长整形
Char * **Ecvt**(double,size_t,char*)	浮点数转换为字符串
Double **Strtod**(char*,char**,int base)	字符串转换为浮点，可设置进制
Long **Strtol**(char*,char**,int base)	转换为长整形，设置进制
Unsigned long **Strtoul**(char*,char**,int base)	转换为无符号长整形，设置进制
Int **Toascii**(int c)	转换为ASCII码字符
Int **Tolower**(int c)	转换为小写
Int **Toupper**(int c)	转换为大写

字符串比较：<string.h>

Int **bcmp**(s1,s2,int n)	比较前n个字符是否相同，相同返回0，否则返回非0	
Int **Memcmp**(s1,s2,size_t)	比较前n个字符是否相等，返回第1个不相等字符差值	
Int **strncasecmp**(s1,s2,size_t)		忽略大小写，其余和memcmp相同

字符串复制：<string.h>

Bcopy(src,dest,int n)	将src的前n个字符复制到dest
Char* **Strcpy**(dest,src)	将src复制到dest，返回指向dest的指针
Char***strncpy**(dest,src,siez_t)	将src前n个字符复制到dest，返回dest的指针

字符串清理与填充：<string.h>

Bzero(void *s,int n)	将s的前n个字符清0，变成NULL
Memset(s,c,size_t n)	将s的前n个字符填充为c
Strset(s,c)	将s的所有字符替换成c

字符串查找:<string.h>

Char***index**(s,c)	查找s中第一次出现c位置
Char***rindex**(s,c)	茶盅s中最后一次出现c的位置
Char***Memchr**(s,c,size_t n)	在s的前n个字符中查找c，返回位置指针
Char***strchr/strrchr**(s,c)	查找s中第一次/最后一次出现c的位置
Int **Strcspn**（char*，char*）	在串1中找串2中出现的字符，返回第一个出现的字符的下标值
Char* **Strpbrk**(char*，char*)	在串1中找串2中出现的字符，返回第一个出现字符的指针
Char* **Strstr**(char*,char*)	在串1查找串2第一次出现的位置

字符串的连接和分割<string.h>

Char***strcat**(dest,src)	将src的内容添加到dest的后面
Char***Strtok**(s,char*delim)	Delim为分割的标记，要调用两次，第1次会将标记改为NULL，后面再调用会依次返回分割后的字符串指针。

Strdup（char*）：复制字符串。**Strrev（char*）**：将字符串的顺序颠倒。

3，文件输入输出函数(stdio.h)

- **Fp=fopen(文件名，使用文件方式)**：打开文件。使用文件方式：**r**(只读),**r+**(读写)，**w**(只写)，**w+**(读写)，**a**(追加)，**a+**(读写)，**rb**(只读二进制)，**rb+**(读写二进制)，**wb**(只写二进制)，**wb+**(读写二进制)，**ab**(追加二进制)，**ab+**(读写二进制)。
- **FILE* freopen(char* path,char* mode,FILE*stream)**:关闭stream指针的文件，然后打开path代表的文件，返回文件指针。
- **FILE* fdopen(int fd,const char * mode)**:将open打开的文件编号转换为文件指针返回。Mode为打开方式要与open里的方式一致。
- **Fclose(指针)**：关闭文件。
- **Fputc(char,fp)**:把一个字符写到磁盘文件，成功则返回输出的字符，失败返回EOF；**fgetc(fp)**：从文件读入一个字符，遇到文件结束符则返回EOF。**Feof(fp)**：判断文件是否结束，结束返回1。
- **Fputs(str，fp)**：把一个字符串写到磁盘文件，成功返回0。**fgets(str，n，fp)**：从文件中读取n-1个字符，放入str中，遇到换行符或EOF则停止。
- **Fread(buffer,size,count,fp)，fwrite(buffer，size，count，fp)**:buffer为数据存放地址，size为一次读写的字节数，count为读写的次数。
- **Fprintf(文件指针，格式，输出列表)，fscanf(指针，格式，输入列表)**：在内存和磁盘频繁交换数据时，最好使用fread和fwrite。
- **Rewind(指针)**：使指针重新返回文件的开头。**Fseek(指针，位移量，起始点)**：改变文件指针位置，起始点用0,1,2表示，0为文件开始，1为当前位置，2文件末尾。
- **Char* getcwd(char*buf,size_t size)**:返回当前路径到buf，如果buf太小则返回-1。<unistd.h>

4,内存操作函数

分配和释放内存(stdlib.h)

- **Malloc**(size_t size),**calloc**(size_t num,size_t size):动态分配内存，size为分配内存的大小，num为内存块的个数。Malloc一次申请一个内存区并且不初始化，calloc可以申请多个内存区并初始化为0。
- **Free(*p)**:动态释放内存，p为分配时返回的指针。Malloc/calloc与free要成对出现。

内存块操作(string.h)

- **Memset（void*buffer，char/int ch，int count）**：初始化指定的内存空间，将buffer的前count个字节设置成ch。
- **Memcpy（void *dest，void*src，unsigned int count）**：复制内存块，由src所指的内存区域复制count个字节到dest。Memcpy只复制

内存空间，不处理空间重叠问题。

- **Memmove(void*dest，void*src，unsigned int count)**：复制或者说移动内存空间，memmove会处理空间重叠的问题，若有重叠可以正确处理，但是src内容会变化。

位操作

&	**按位取**与
|	按位取**或**
^	按位取**异或**
~	按位取**反**
<<	**左移**
>>	**右移**

五，时间函数

1,**time_t time(time_t *)**:返回当前时间值，日历时间以秒为单位。

2，**struct tm* gmtime(time_t*)** ：将time_t表示的秒数转换为tm结构体类型数据。**Time_t mktime(tm*)**:将tm类型数据转换为time_t类型。

3，**char* ctime(time_t*)**：将time_t时间转换成可以识别的字符串。**char* asctime (struct tm*)** ：将tm时间转化为字符串格式。

4，**struct tm *localtime(time_t)**：返回tm格式的本地时间。

5，**int gettimeofday (struct timeval*, struct timezone*)** ：从今日凌晨到现在的时间差，返回微秒级的时间，返回到两个结构体指针上，处理成功返回值为1，否则为0。**int settimeofday (struct timeval*, struct timezone*)** ：设置当前的系统时间。

6，**unsigned int sleep(unsigned int sec)**:使程序睡眠sec秒。**Void usleep(unsigned long usec)** ：使程序睡眠usec微秒。

六，目录与文件(依赖于linux系统)

1，**错误定义**：在linux中已经把所有的错误定义为不同的错误号和错误常数，存放在**/usr/include/asm-generic/errno-base.h**和**/usr/include/asm-generic/errno.h**两个错误定义文件中。使用这些错误时要引用该两个头文件。

Char* strerror(错误常数/错误序号)：将错误序号或错误常数表示的错误信息返回到char指针。

2，**创建和删除目录**（sys/types.h,sys/stat.h）

Int mkdir(char*,mode_t):创建目录，char*为linux下的路径，mode_t为八进制数字表示权限。创建成功会返回0，否则返回-1。

Int rmdir(char*):删除一个空目录，char*为目录路径，删除成功返回0。**Extern int errno**设置一个errno来捕获错误。

3，**创建和删除文件**(sys/types.h,sys/stat.h,fcntl.h)

Int creat(char*,mode_t):在指定路径下创建空文件，mode为权限。创建成功返回文件标号，否则返回-1。

Int remove(char*):删除一个文件，删除成功返回0，否则返回-1。

Int mkstemp(char*):常见一个临时文件，用于程序运行时存储中间数据，计算机重启后自动删除。文件名最后6个字符必须是XXXXXX，成功返回文件标号，否则返回-1。**这里的char*必须是char path[]格式，不能是指针格式**。

4,**文件的打开关闭**(sys/types.h,sys/stat.h,fcntl.h)

Int open (char*, int flags) , int open(char*, flags, mode):打开成功返回文件编号，失败返回0。mode为打开不存在文件创建时的权限。flags表示文件

的打开方式：

- O_RDONLY:只读方式；O_WRONLY:只写方式；O_RDWR：读写方式
- O_CREAT:自动创建文件；O_EXCL:如果O_creat设置了，会自动创建文件，否则打开失败；O_NOCTTY:文件为终端设备，不会讲该终端机当做进程控制终端机。

Int close(int fd):将数据写入磁盘，关闭文件，fd为文件编号。<unistd.h>。成功返回0。

5，文件读写

Size_t write(int fd,void*buf,size_t count):写入文件，buf为需要写入的字符串，count为写入的字符个数。返回实际写入的字节数，失败时返回-1。

Size_t read(int fd，void*buf，size_t）：读取文件，返回值表示读取到的字符个数，返回0表示文件结尾；返回-1读取失败。

Off_t lseek(int fd,off_t offset,int whence):文件读写位置的移动。Whence的取值为：

- **SEEK_SET**：offset即为新读写位置；**SEEK_CUR**：当前位置加上offset；**SEEK_END**：读写位置指向结尾再加上offset。

Int sync(void):将程序打开的所有文件内容保存到磁盘。

Int fsync(int fd):将指定文件内容写入磁盘。成功返回0，失败返回-1。

Int access(const char*path,int mode):判断文件是否可以进行某操作。Mode取值：R_OK,W_OK,X_OK,F_OK(是否存在）。符合返回0，反之返回1。

6，文件锁定，移动

Int flock(int fd,int oper):锁定文件，使其他用户当前不能操作。成功返回0，失败返回-1。Oper的取值：

- **LOCK_SH**:共享锁定，其他程序可以访问。
- **LOCK_EX**:互斥锁定，其他用户不能同时访问
- **LOCK_UN**:接触文件锁定。
- **LOCK_NB**：无法建立锁定时，马上返回进程，不会被阻断。

Int rename(char* ,char*):在同一分区移动文件，前面为旧路径，后面为新路径。成功返回0，失败返回-1。

七，网络编程

1，网络基本概念

- **端口**：每个程序访问网络，会分配标识符表示这一网络数据属于某程序，即为端口。16位整数，0~65535，低于256的端口是系统保留端口号。
- **ICMP**：消息控制协议，与IP一层，传送IP的控制信息，提供通向目的地址的路径信息，ping就是其应用。
- **TCP**：面向连接的网络传输方式，可以理解为打电话，传输过程复杂，占用较多的网络资源。
- **UDP**：面向无连接的传输方式，理解为邮寄邮件，不可靠，但对传输要求不高。
- **套接字(socket）**：用来描述计算机中不同程序与其他计算机程序的通信方式。**套接字=传输层协议(TCP)+端口号+IP地址**。
- **Sockaddr**：结构体用于保存套接字信息，sa_family指定通信的地

址类型，sa_data保存ip地址和端口信息。

- **Sockaddr_in**：保存套接字信息，sin_family：通信类型；sin_port：端口号；sin_addr：IP地址；sin_zero：未使用字段。

2,套接字类型

- **流套接字**：TCP协议，无差错、无重复发送、按顺序接收
- **数据报套接字**：UDP协议，不能保证数据正确传输，不能保证按顺序接收，可能出现数据丢失。
- **原始套接字**：没有处理的IP数据包，可以按照自己要求封装。

3，基本网络函数：

- **Int socket(int domain,int type,int proto)**:建立一个通信接口，proto为协议编号，一般设置为0。成功返回套接字编号，失败返回-1。domain表示地址类型。
- **Int bind(int sockid,sockaddr*addr,int addtlen)**:设置sockfd，相当于socket一个名称，定位。
- **Int getsockopt(int s,int level,int optname,void*optval,socklen_t*optlen)**:取得一个socket的参数，成功返回0，反之返回-1。s为socket编号，level为需要设置的网络层(SOL_SOCKET)，optname是需要获取的选项，常用值
  - SO_DEBUG：打开或关闭排错模式
  - SO_REUSEADDR：允许在bind函数中本地地址可重复使用
  - SO_TYPE：返回socket形态
  - SO_ERROR：返回socket已发生的错误原因
  - SO_DONTROUTE：送出的数据包不用路由设备传送
  - SO_BROADCAST：使用广播方式传送

Optval为是取得某个参数的返回值指针，optlen是optval的内存长度。

- **Int setsockopt(int s,int level,int optname,void*optval,socklen_t optlen)**:设置一个socket状态，成功返回0，反之返回-1，level一般设置为SOL_SOCKET。

4，服务器和客户机的信息函数

字符转换函数

- **ulong htonl(ulong hostlong)**:32位hostlong 转换成网络字符顺序
- **Ushort htons(ushort hostshort)**：16位hostshort转换成网络字符顺序
- **long ntohl(ulong netlong)**：32位netlong转换成主机字符顺序
- **ushort ntohs(ushort netshort)**：16位netshort转换成主机字符顺序

域名和IP地址转换(netdb.h,sys/socket.h)

- **Struct hostent *gethostbyname(const char* name)**：域名转换为对应的IP，name为域名，返回值为一个主机地址结构体，包括**主机名、别名、主机名类型、地址和长度信息**。

- **Struct hostent*gethostbyaddr(const void*addr,socklen_t len,int type)**:IP地址转换到域名，addr为IP地址，len为IP长度，type一般为**AF_INET**。

32位IP与点分IP转换

- **Char* inet_ntoa(struct in_addr in)**:将32位IP转换为a.b.c.d。In_addr为表示IP的结构体。
- **Int ient_aton(char*cp,struct in_addr*inp)**:将a.b.c.d转化成32位的IP。

服务信息函数

- **Struct servent* getservent()**:取得系统所支持的网络服务，返回一个servent结构体，包括**服务名、别名、端口、使用的协议**等信息。系统支持的服务在/etc/services文件中
- **Struct servent* getservbyname(char* name,char*proto)**:用服务名取得一个服务。Name为服务名，proto为使用的协议。
- **Struct servent* getservbyport(int port,char*proto)**:从一个端口取得一个服务。Port为端口号，需要用**Htons(port)**转换，proto表示协议。
- **Struct protoent* getprotobyname(char*)**:根据名称取得一个协议的数据。Char*为协议名称字符串，返回值为protoent结构体，结构体中包括**协议名称、别名、序号信息**。<netdb.h>。
- **Struct protoent* getprotobynumber(int proto)**:根据协议的编号取得协议的信息。Proto为协议的编号。<netdb.h>。
- **Struct protoent* getprotoent(void)**:取得系统中所支持的所有协议。系统的协议是记录在/etc/protocols文件中。结束后返回NULL。<netdb.h>

5，错误处理<netdb.h>

- **Void herror (const char*s)** ：显示上一个网络函数发生的错误。会先输出char*，然后直接输出错误信息。
- **Extern int h_errno**：网路程序中可以使用该语句不活发生错误的编号。捕获之后输出错误信息：**char* hstrerror(h_errno)**。

八，TCP和UDP通信

1，**TCP**通信服务器和客户端的配置

服务器端	**客户端**
1，创建socket，用socket() 2,绑定IP、端口到socket，用Bind() 3,设置最大连接数，listen() 4，等待来自客户端的连接请求，accept() 5，收发数据,send()和recv(),或read()和write() 6，关闭连接	1，创建socket 2，设置要连接的服务器IP和端口 3，连接服务器，用connect() 4,收发数据,send()和recv(),或read()和write() 5,关闭连接

2，**UDP**通信服务器和客户端的配置

服务器端	**客户端**
1，创建socket，用socket()	1，创建socket，用socket()

2,绑定IP、端口到socket，用Bind() 3，循环接收数据，用recvfrom() 可以向客户端发送信息，sendto() 4，关闭连接	2,绑定IP、端口到socket，用Bind() 3,设置对方的IP地址和端口等属性 4，发送数据，用sendto() 可以接收服务器信息，recvfrom() 5，关闭连接

3,服务器模型

- 循环服务器：在同一时刻能够响应一个客户端请求
- 并发服务器：在同一时刻可以响应多个客户端请求

Udp循环服务器	tcp循环服务器	Tcp并发服务器

Socket();
Bind();
While(1)
{ recvfrom();
Process();
Sendto();
}
	Socket();
	Bind();
	Listen();
	While(1)
	{ accept();
	Process();
	Close();
	}
		Socket();
		Bind();
		Listen();
		While (1) {
		Accep();
		If(fork()==0){
		Process();
		Close();
		Exit();}
		Close(

九，进程控制

1，进程相关概念

- 进程：具有独立功能的程序的一次运行。具有以下特点：**动态性、并发性、独立性、异步性**。进程间通过ID号区分。
- 进程互斥：争夺共享资源。即临界资源只允许一个进程访问。临界区：访问临界资源的程序代码。
- 进程同步：一组并发进程按照一定的顺序执行，也为合作进程。
- 进程调度：分为抢占式和非抢占式，具体算法有先来先服务、短进程优先、高优先级优先、时间片轮转。
- 死锁：进程彼此竞争资源形成僵局，需要外力。

2，进程编程<unistd.h>

- Pid_t getpid(void):获取本进程的ID。Pid_t getppid(void):获取父进程ID。
- Pid_t fork(void):创建子进程，在父进程中返回新创建的子进程的ID，在子进程中返回0。出现错误返回负值。Fork（）之后的代码会被执行两次，但执行顺序不确定。**代码共享，数据不共享**。
- Pid_t vfork(void):创建子进程，**数据和代码都共享**，并且子进程先运行，父进程后运行。

Exec函数族：启动一个新程序，替换原有的进程，ID号不变。

- **Int execl(char*path,char*arg1-n)**:path为被执行的程序路径，arg1-n为被执行程序所需要的命令行参数，程序名也要包含在其中。以**NULL表示结束**。
- **Int execlp(char*file,char*arg1-n)**:path为程序名，不带路径，会从PATH环境变量中查找该程序。其余和execl相同。
- **Int execv(char*path,char*const argv[])**:path为程序路径，argv[]为命令行参数组成的数组。
- **Int system(const char*string)**：调用fork产生子进程，由子进程调用**/bin/sh -c string**来执行参数string代表的命令。
- **Pid_t wait(int* status)**:堵塞该进程，直到其某个子进程退出或中断，返回退出的子进程的ID，status为子进程的结束状态值。

3，进程通讯概述

为什么需要进程通信？

- 数据传输、资源共享、通知事件、进程控制

Linux的进程通信方式(IPC)

- 管道/有名管道、信号、消息队列、共享内存、信号量、套接字。

4，管道通信

管道是**单向、先进先出**。**无名管道**只能用于父进程和子进程之间的通信，**有名管道**可用于运行同一系统的任意两个进程间的通信。

使用read和write函数读写管道里的数据，就像操作文件一样。

- **Int pipe(int filedis[2])**：创建一个无名管道，成功返回0，失败返回-1。filedis[0]为读管道，filedis[1]为写管道。关闭管道只需要用close关闭两个管道filedis。**必须在fork之前调用pipe创建管道**。
- **Int mkfifo(char*path,mode_t mode)**:创建命名管道(FIFO)，命名管道相当于一个文件，path为文件路径，mode为权限。打开FIFO时，非阻塞标志**O_NONBLOCK**，即读空FIFO时返回不会阻塞。

5，信号通信

常见信号：

- Sighup：从终端发出的结束信号；
- Sigint：来自键盘的中断信号，ctrl+c；
- Sigkill：该信号结束接收信号的进程；
- Sigterm：kill命令发出的信号；
- Sigchld：标志子进程停止或结束的信号；
- Sigstop：来自键盘ctrl-Z或调试程序的停止执行信号。

几个常用发送信号的函数

- **Int kill(pid_t pid,int signo)**:可以向自身发送信号，也可以向其他进程发送信号。**Pid>0**:发送给pid的进程；**pid==0**：发送给同组进程；**pid<0**：发送给进程组ID等于pid绝对值的进程；**pid==-1**：发送给所有进程。
- **Int raise(int signo)**:只能向自己发送信号。

- **Unsigned int alarm(unsigned int second)**:将SIGALRM信号发送给自己，在设置的时间到来时，second为时间(秒)。默认是终止进程。
- **Int pause(void)**:将进程挂起直到捕捉到一个信号。

信号的处理：

- **Void (*signal(int signo,void(*func)(int)))(int)**:func可能的值：SIG_LGN(忽略)、SIG_DFL(系统默认)或信号处理函数名。

6，共享内存通信（用的多）

步骤：创建共享内存、映射共享内存

- **Int shmget(key_t key,int size,int shmflg)**:创建共享内存，key标识其键值：0/IPC_PRIVATE(创建新内存),为0并且shmflg为IPC_PRIVATE也是创建新内存。成功返回内存标识符，失败-1.
- **Int shmat(int shmid,char*shmaddr,int flg)**:映射内存，shmid为内存标识符，flg一般为0，shmaddr为内存的指针，通过它操作共享内存。
- **Int shmdt(char*shmaddr)**:解除映射，从进程地址空间脱离。
- **Int shmctl(int shmid,int cmd,struct shmid_ds*buf)**:共享内存操作，cmd取值：IPC_STAT获得内存信息复制到buf；IPC_SET设置共享内存信息；IPC_RMID删除内存段；SHM_LOCK锁定，只有root可以;SHM_UNLOCK解锁内存，只有root。

7，消息队列通信<sys/types.h><sys/ipc.h><sys/msg.h>

消息的链表，具有特定的格式，而管道没有格式。内核重启或人工删除才会消除，每个消息队列对应唯一的键值。

- **Key_t ftok(char*path,char proj)**:返回文件名对应的键值。Proj为项目名(不为0即可）。
- **Int msgget(key_t key,int msgflg)**:返回与键值key对应的消息队列描述字。Msgflg为标志位：**IPC_CREAT**创建新的消息队列；**IPC_EXCL**一般与IPC_CREAT同用；**IPC_NOWAIT**无法读写时，不堵塞。
- **Int msgsnd(int msqid,struct msgbuf*msgp,int msgsz,int msgflg)**:向消息队列发送一条消息，msgbuf为消息的特定格式。
- **Int msgrcv(int msqid,struct msgbuf*msgp,int msgsz,long msgtyp,int msgflg)**:从消息队列中读取一个消息，存入msgp。注意msgsz应该是msgbuf的大小减去msgtyp的大小。
- **Int msgctl(int msqid,int cmd,struct msqid_ds*buf)**:消息控制，cmd为：**IPC_STAT**获取队列信息存入buf；**IPC_SET**设置消息队列属性，属性存于buf；**IPC_RMID**删除消息队列。

8，信号量通信

信号量(信号灯)主要用于保护临界资源。进程可以根据它判断是否能够访问某些共享资源，用于访问控制和进程同步。

- 二值信号量：只能取0或1，类似于互斥锁。

- 计数信号量：值可以取任意非负值，直到减到0为止。
- **Int semget(key_t key,int nsems, int semflg)**：打开/创建信号量集，key为ftok获取的键值，nsems为信号量集中信号量个数，semflg为标识，和消息队列一样。
- **Int semop(int semid,struct sembuf*sops,unsigned nsems)**:对信号量进行控制。Sops表示进行什么操作，nsems为sops指向的信号量集信号量个数。Semflg：**IPC_NOWAIT**不堵塞；**IPC_UNDO**程序结束释放信号量。

十，多线程程序设计

1，线程，和进程相比是一种**非常节俭**的多任务操作方式。线程间彼此切换所需的时间远远小于进程间切换的时间。编写linux多线程，需要**pthread.h**和**libpthread.a**库文件。

代码段，数据段都是共享的。全局变量是在数据段里，不用传递在线程里就可以使用，而局部变量不可以，要经过传递。

因为pthread库不是linux系统的库，在编译时候要加上-lpthread

- **Int pthread_create(pthread_t*tidp,const pthread_arrt_t*attr,void*(*start_rtn)(void),void*arg)**:创建线程，tidp为获取的线程id，attr为线程属性(一般为空),start_rtn 线程要执行的函数，arg为函数参数。

线程正常退出方式：1，从启动进程中返回；2，被另一个进程终止；3，线程自己调用函数终止。

- **Void pthread_exit(void *rval_ptr)**:线程自己终止，rval_ptr是线程退出返回值的指针。
- **Int pthread_join(pthread_t tid,void**rval_ptr)**:线程等待。阻塞调用线程，直到指定的线程终止。Tid为等待的线程id，rval_ptr为线程退出返回值指针。
- **Pthread_t pthread_self(void)**:返回线程的ID。
- **Void pthread_cleanup_push(void(*rtn)(void*),void*arg)**:将清除函数压入清除栈，rtn为清除函数，arg为参数。
- **Void pthread_cleanup_pop(int execute)**:将清除函数弹出清除栈，execute表示是否在弹出同时执行函数，0不执行。清除函数用于pthread_exit和异常终止，不包括return。

2，修改线程属性：

属性结构为pthread_arrt_t,属性对象包括是否绑定，是否分离，堆栈地址，堆栈大小，优先级等。对属性的设置修改要在create之前

轻进程：系统对线程资源的分配控制都是通过轻进程进行，绑定即某个线程固定在一个轻线程上。

- **int pthread_attr_init(pthread_attr_t*tattr)**:初始化
- **Int pthread_attr_setscope(pthread_attr_t*tattr,int scope)**:绑定，第一个参数为指向属性结构的指针，scope为绑定类型：PTHREAD_SCOPE_STSTEM绑定；PTHREAD_SCOPE_PROCESS非绑定。成功返回0。

分离状态决定线程以什么方式终止自己，分离线程没有被其他线程等待，运行结束马上释放资源。

- Int pthread_attr_setdetachstate(pthread_attr_t*tattr,int detachstate):分离，detachstate取值：PTHREAD_CREATE_DETACHED分离线程；PTHREAD_CREATE_JOINABLE非分离线程。成功返回0；

优先级存放在sched_param结构中，调度参数为sched_priority

- Intpthread_attr_setschedparam(pthread_attr_t*,sched_param):定义调度参数
- Intpthread_attr_getschedparam(pthread_attr_t*,sched_param)：获取当前的调度参数。