Linux使用手册：



Linux基础：

**Ubuntu16.04换国内源：**<http://t.zoukankan.com/masbay-p-10887571.html>

**Linux Qt cannot find -lGL错误完美解决方案：**

Qt 找不到 OpenGL 的动态链接库（libGL.so）

Qt 默认在 /usr/lib/ 目录下查找动态链接库，但是很多 Linux 发行版将 OpenGL

链接库放在其它目录；只要我们把 libGL.so 拷贝到 /usr/lib/ 目录，或者在

/usr/lib/ 目录下为 libGL.so 创建一个链接，就能解决问题。显然第二种办法更好

#查找 libGL 所在位置

[root@localhost ~]# locate libGL

/usr/lib64/libGL.so

/usr/lib64/libGL.so.1

/usr/lib64/libGL.so.1.2.0

/usr/share/doc/mesa-libGL-9.2.5

/usr/share/doc/mesa-libGL-9.2.5/COPYING

#创建链接

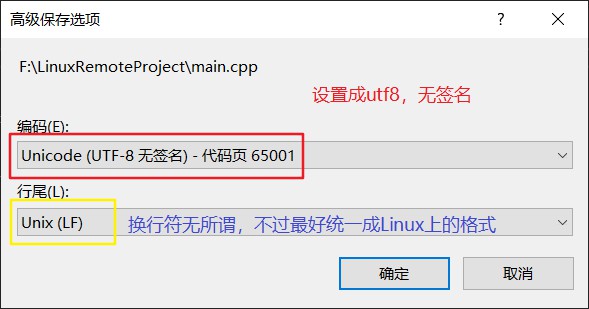
[root@localhost ~]# ln -s /usr/lib64/libGL.so.1 /usr/lib/libGL.so

Linux 系统中可能存在多个版本的 libGL.so，为任意一个版本创建链接即可。普通

用户没有权限创建链接，所以我使用了 root 用户；完成以上操作，再次启动 Qt，

然后编译或者运行，就不会出现“cannot find -lGL”错误了。

**使用VS2019进行Linux远程开发中文乱码问题：**



**服务器常用Linux，客户端常用Windows：因为Linux开源免费，安全性更高；**

**Linux框架结构：**

裸机+内核Kernel+外核Shell（向上提供接口）+应用程序+外围工具（应用程序）

**Linux设备管理：**所有设备都能看做是特殊的文件

**Linux文件：**

后缀为\*：可执行文件

前缀为d：文件夹

前缀为-：文件

bin：存储Linux常用命令

boot：系统盘,存储操作系统引导程序

dev：Linux系统中使用的外部硬件设备映射文件

etc：系统管理的各种配置文件&主目录

/etc/passwd 用户信息

/etc/shadow 用户密码(已加密)

/etc/group 组信息

home：主目录

lib：存放系统动态连接共享库，几乎所有应用程序都会用

lost+found：存储系统强制关闭时丢失的文件

mnt：分区的挂载点

proc：虚拟目录，系統内存的映射，可获得系统信息

root：系统管理员的主目录

usr：用户很多应用程序&库都放在这

**Linux用户类别:**

root：最高权限;

owner：实际拥有文件的用户;

group：用户组;

world：其他用户；

**Linux文件权限位：**

共10位：r读w写x执行-没有该权限

文件类型：普通文件- 目录d 链接文件(快捷方式)l

第一位:文件类型；后三位: owner的权利；后三位: group的权利；后三位: word的权利;

**LinuxCRT中文乱码问题：**

在windows里面记事本方式打开cpp，另存为UTF-8编码，在ubuntu里面打开就

不会乱码了；再放到ubuntu里面去

**用CRT远程连接Linux开发：**

服务器不常在身边，要在服务器的操作系统上进行开发就必须远程连接

CRT远程连接使用UTF-8编码方式

**静态库&共享库：**

静态库（.a）：程序在编译链接的时候把库的代码链接到可执行文件中

共享库（.so或.sa）：程序在运行的时候才去链接共享库的代码，多个程序共享使用库的代码

**Linux系统调用内核函数 & API函数库调用函数：**

系统调用：直接请求操作系统内核服务，执行代码效率更高；每种操作系统内核

函数各不相同，具有局限性

API函数库调用：封装在程序内的函数；调用兼容性、可移植性更好；调用开销

更小；

Linux操作命令：

**Linux切换root用户：**

首先，要设置root用户的密码： sudo passwd root

设置root用户密码

切换root用户：su root

输入密码切换

退出root用户：exit

**常用快捷键：**

挂起终端进程：ctrl+z

强制中断程序：ctrl+c

终端清屏命令：clear

查看隐藏文件：ctrl+h

**文件操作：**

切换目录：cd 要切换到的目录（.当前路径 ..上一级路径 ~home路径）

输出目录下所有文件信息：ls（输出文件名） or ll（输出文件详细信息）

文件创建：touch

文件拷贝：cp 文件 目标目录

文件移动：mv 文件 目标目录

文件删除：rm 文件 目标目录

文件夹创建：mkdir 文件夹

文件夹删除：rmdir 文件夹

删除非空目录：rm -R 目录

查看文本内容：cat 文件（瞄一眼）

**Linux设备&进程指令：**

ps -aux：查看任务管理器

pstree：查看进程树状图

**文件权限管理：**

chmod：给创建文件的用户赋予更多权限r-4 w-2 x-1 u-该文件拥有者g-该档案拥有者同组成员o-其他人

**用户&组的创建删除**：

创建用：adduser xxx (创建时会自动创建同名组) ;

创建组：groupadd xxx;

用户添加到组：adduser 用户名 组;

删除用户：deluser 用户名 组名;

删除组：delgroup 组名;

**修改文件所属用户&组**

chown 所属用户:所属组 文件名

**删除用户**

deluser -remove-home 用户名

**Vim操作：**

安装：apt install vim

Vim打开文件进行编辑：vim 文件

进入insert编辑模式：点i

编辑完成退出insert模式：Esc

保存退出文件：:wq

退出不保存文件：:q!

Shell Bash：通配符，上下键记忆命令，Tab自动补全，输出/入重定向：>,>>,<, 管道：I

**gedit操作：**工具，类似Windows的记事本；

**Linux文件IO指令：**

**终端指令—man 2/3... 指令名：查看API使用手册—Linux Programmer's Manual**

umask()：设置权限掩码，默认0022 rwx421——rwxrwxrwx777

perror()：输出报错信息，头文件stdio.h中

open()：打开文件

close()：关闭文件

write()：文件写入数据

read()：文件读取数据

获取读写数据大小：

strlen—针对纯文本文件：txt cpp h

sizeof—针对二进制文件：zip mp4 mp3

复制拷贝文件的时候推荐：write长度就是read的返回值

Linux编译工具：

**Linux编译工具：**gcc（编译C）& g++（编译C++）

gcc -E源文件-o文件名.i 预处理文件

gcc -S文件名.i-o文件名.s 汇编文件

gcc -C文件名.s-o文件名 可执行文件

也可一步到位：gcc源文件-o文件名

**Linux生成动态链接库**（公用化，方便今后重新调用）：

g++源文件名cpp -shared -fpic -o lib源文件名.so

-shared和-fpic要空格分开

动态链接库移动至/usr/li

使用动态链接库

g++ main.cpp -o libmain -l源文件名（不需要文件后缀）

**Linux gdb代码调试工具：**

1.生成可执行文件 -g

gcc \*.c -g -o 可执行文件

2 gdb ./可执行文件 进入调试模式

l (list)：查看代码

r (run)：运行

b n (break n)：在n行打断点

c (continue)：执行到下一个断点

n (next)：下一步

watch变量名：观察变量

**Linux make工具:**

make自动推导:

make工具通过一个称为makefile的文件来完成并自动维护编译工作

只要make看到一 个[.o]文件,它就会自动的把[.c]文件加在依赖关系中

makefile格式:

目标文件：依赖文件

命令

目标文件1：依赖文件1

命令

clear：

rm -f目标文件 目标文件1...

makefile最终版：

ELF=最终要生成的目标文件名

src=$(wildcard \*.cpp) //wildcard函数: 当前目录下匹配模式的文件

object=$(src: .cpp=.o) //src..cpp=.o：模式匹配替换

$(ELF) : $(object)

g++ $(object) -o $@ /$@：规则的目标文件名

$(object) :

clean:

rm -f $(object) $(ELF)

自动编译make

删除目标文件make clear

**Makefile工作原理：**

make会在当前目录下找名字叫makefile的文件

如果找到，它会找文件中的第一个目标文件，并且把这个文件作为最终的目标文件

如果该文件不存在，或是其所依赖的后面的.o文件的文件修改时间要比其新，它就会执行后面所定义的命令来生成新的第一个目标文件

如果第一个目标文件所依赖的\*\*.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，根据之前的新旧或是否存在的规则，来生成新的.o文件

Linux内核编程：

**Linux系统调用内核函数& API函数库调用函数：**

**系統调用：**直接请求操作系统内核服务，执行代码效率更高；每种操作系统内核函

数各不相同，具有局限性

**API函数库调用：**封装在程序内的函数；调用兼容性、可移植性更好；调用开销更

小；

**Linux指令：**

**终端指令一man 23... 指令名**；查看AP|使用手册一Linux Programmer's Manual

umask()：设置权限掩码,默认0022 rwx421-- -rwxrwxrwx777

perror()：输出报错信息，头文件stdlio.h中

open()：打开文件

close()：关闭文件

write()：文件写入数据

read()：文件读取数据

获取读写数据大小: .

strlen-针对纯文本文件：txt cpp h

sizeof-针对二进制文件：zip mp4 mp3

Linux char数组初始化:

memset()函数 或 bzero()函数

**Linux进程：**

进程ID(PID)：每个Linux进程都有一个唯一的数字标识符; 总是非负整数

进程控制块(PCB)：进程存在的唯一标志; =代码段+数据段+堆栈段

init进程：PID为1的进程，在自举过程结束时由内核调用，绝不会终止

孤儿进程：父进程比子进程先执行结束，父进程将子进程交付系统进程管理

僵尸进程：子进程比父进程先执行结束，子进程执行逻辑结束，但内存没有收回

解决方案：父进程比子进程先执行结束，子进程是否结束应该给父进程反馈，父进

程依据反馈判断是否结束

**Linux进程编程：**

pid\_t getpid(void)：返回调用进程的进程ID

pid\_t fork(void)：返回：子进程中为0，父进程中为子进程ID，出错为-1;

原理：fork开出子进程，子进程复制父进程代码，因此一次调用会有两次返回

**Linux进程代码框架：**

pid = fork()

if(pid > 0) .

{

//父进程

wait(x); //为了避免出现孤儿进程，父进程再逻辑执行完以后使用wait(x)函数

//等待子进程结束

}

else if(pid == 0)

{

//子进程

exit(0); //为了避免出现僵尸进程，子进程结束以后要调用exit函数

}

**Linux设备&文件IO：**

DIR目录

struct dirent目录信息结构体

DIR\* opendir(char \*pathname);

打开一个目录

返回：成功，返回一个目录指针；失败，返回0

struct dirent\* readdir ( DIR \*dirptr);

访问目录中的文件

返回一个指向dirent结构的指针,它包含指定目录中下一个连接的细节;

没有更多连接时，返回NULL

**Linux信号机制：**

用于实现进程间通信

进程可以生成信号、捕捉并响应信号或屏蔽信号

是Qt信号与槽机制的祖宗

**信号分类：**

不可靠信号：1~31 若连续发送，信号会被忽略；但是发送一次休息一会再发

送则信号有效

可靠信号：34~64 无论如何发送，信号一定会被执行

**kill终端指令：**

Linux指令，用于发送信号

Kill-几号信号 进程PID

**Linux信号两种写法：**

**无参信号：**

signal函数≈Qt connect函数

函数signal(哪个信号，函数指针)：绑定信号和执行函数

kill函数：发送信号给指定进程

函数kill(接收信号的进程，哪个信号)

**带参信号：**

sigaction函数=Qt connect函数，需要sigaction结构体变量作为参数

sigaction结构体：信号是否可携带参数状态位，处理函数，屏蔽信号集

sigqueue函数：发送信号给指定进程，需要union sigval联合体变量作为

参数

union sigval联合体：**信号携带的参数(现在只能为int)**

处理函数(自定义)：函参列表与sigaction结构体中存储处理函数的元素的

函参列表要一致

**信号屏蔽(防止进程接收到不相关信号直接结束)：**

如果一个进程正在运行，没有绑定任何信号和处理函数。当运行中进程收到一个不认识的信号不知道如何处理就会中断进程执行。这个时候我们就需要考虑增加信号屏蔽确保程序的健壮性

定义信号集：sigset\_t数据类型

信号集初始化：sigemptyset函数

信号集添加：sigaddset函数

信号集删除：sigdelset函数

添加全部信号：sigfillset函数

信号集启动屏蔽：igprocmask函数

**信号冲突：**

情况1：

一个进程正在执行信号处理函数，还在执行没有结束，这个时候进程又收

到同种信号

第一次执行结束后，再执行第二次执行函数

情况2：

一个进程正在执行信号处理函数，还在执行没有结束，这个时候进程又收

到异种信号

第一次执行函数打断，执行异种函数处理，异种函数处理结束后，回到第

一次还没处理完的执行函数

但时间上存在问题

**结论：异种信号打断是会造成逻辑、数据出错**

**口诀：异种打断，同种排队**

解决方案：一个信号在处理的时候不要被其他信号打扰，所以在这个信号绑定

的时候设置黑名单

struct sigaction属性sa\_ mask

**信号从传递数据角度来审视信号技术，性价比很低**

**信号的最佳实践：做进程通知**

**Linux管道机制：**

管道用于进程通信

特点：半双工通信

**匿名管道(无名管道)：**

只用在亲缘关系进程之间

1.创建管道：

pipe函数，需要file\_ descriptor[2]数组作为参数，file\_ descriptor[0]表示读(入)端，file\_ descriptor[1]表示写(出)端

2.进行write read读入写出

3.删除对应端：close函数，父进程中close读(入)端，子进程中close写(出)端

**命名管道(有名管道)：**

不相关进程间交换数据，利用FIFO管道文件，进程通过访问FIFO文件进行数

据通信

1.创建命名文件：mkfifo函数

2.打开管道文件：open函数

3.进行write read读入写出

**Linux IPC-进程间通信：**

Linux ipcs终端指令：查看消息队列、共享内存段、信号量数组

Linux ipcrm -a终端指令：清除用户创建的所有消息队列、共享内存段、信号量数组

**Linux消息队列：**

msgget函数：创建消息队列

msgsnd函数：发送消息至消息队列，参数需要消息体结构体

msgbuf (mtype：消息类型，用于接收；mtext：消息数据)

msgrcv函数：接收消息从消息队列，参数需要消息体结构体

msgbuf (mtype：消息类型，用于接收；mtext：消息数据),

接收消息类型：同结构体msgbuf的mtype一致

**Linux消息队列传输结构体数据：**

利用memcpy函数

存入：将结构体数据传输给msgbuf里的mtext char数组

取出：将msgbuf里的mtext char数组传输给结构体数据

**Linux共享内存：**

IPC通信效率最高，由操作系统管理

一个进程创建，所有进程可连接，共享数据

shmget函数：创建共享内存

shmat函数：进程连接共享内存，返回共享内存指针

利用memcpy将数据考进共享内存中

利用memcpy将数据从共享内存中考出

拿出数据后，利用memset重置共享内存段数据

shmdt函数：进程和共享内存断开连接

**Linux IPC消息队列、IPC共享内存：**

**结合使用，共享内存用于数据通信，消息队列用于消息传递**

**Linux 进程信号量：**

≈线程互斥量&线程信号量

信号量：解决多个进程同时访问一个内存区域导致的数据安全问题

semget函数：创建一个新的信号量或者取得一个现有信号量的键字

semop函数：改变信号量的键值

semctl函数：允许我们直接控制信号量的信息；

p操作：-1加锁；v操作：+1解锁

**Linux POSIX多线程技术：**

进程是资源分配的基本单位，线程是CPU调度的基本单位

一切进程至少都有一个执行线程

新的执行线程会拥有自己的堆栈(有自己的局部变量)，但要与它的创建者共享全局量、文件描述符、信号处理器和当前的子目录状态.

VS2019 POSIX线程使用：

链接线程函数库：右键项目->属性->链接器->命令行->键入"-pthread"->确认

**线程优缺点：**

优点：比创建新进程代价小；切换时间短；占用资源少

缺点：调试多线程程序比调试单线程程序困难

**解决任务并发运行问题；但给每个客户端开线程，资源消耗巨大**

**Linux线程编程：**

引入线程函数库:文件<pthread.h>

pthread\_ create函数：

作用：创建一个新的线程;

函参列表：新线程标识符，一般为NULL，线程启动要执行的函数地址(≈ QThread run函数)，传给线程启动要执行的函数的参数

**Linux线程同步：**

两个(或多个)线程同时执行时，经常需要访问到公共资源或代码的关键部分

防止多个线程同时访问公共资源导致的数据出错问题

在对公共资源进行访问操作前后进行开关锁操作来实现

**线程信号量:**

sem\_t sem; //二进制线程信号量，只有0和1两个值

sem\_init(&sem, 0, 1); //线程信号量初始化，初始化为1

sem\_wait(&sem); //信号量值-1，“关锁"一阻塞函数

sem\_post(&sem); //信号量值+1，“开锁”

**线程互斥量:**

pthread\_mutex\_t mutex; //线程互斥量锁对象

pthread\_mutex\_init(&mutex, NULL); //互斥量锁初始化

pthread\_mutex\_lock(&mutex); //互诉量关锁"—阻塞函数

pthread\_mutex\_unlock(&mutex); //互斥量开锁”

Linux网络编程：

服务器常用Linux，客户端常用Windows：Linux开源免费，安全性高

Linux网络基础：

网络三要素：

IP地址：网络中查找对方计算机

端口号：确定计算机中某一个应用程序 10000以下系统+默认程序使用

通信协议：双方共同约定和遵循的协议(数据打包、解包规则)

计算机网络体系结构：详见计网

TCP/IP协议族：详见计网

**Linux网络编程：**

socket概述：

一套网络通信程序的API函数标准

linux中的网络编程通过socket接口实现；

Socket既是一种特殊的IO，也是一种文件描述符

**基于流套接字(socket)的编程流程：**

socket函数：网络初始化，判断是否可以搭建网络

bind函数：绑定端口和IP地址(其中要将sockaddr\_ in结构体转化成sockaddr)

sockaddr\_ in结构体

.sin\_ sin\_ family：网络连接协议族

.sin\_ \_addr.s\_ \_addr：本地联网IP

.sin\_ port：开放服务器端口，一般10000以上；用htos函数将

主机字节顺序转换为网络字节顺序，防止数据由于大小端模式的

不同而出错

listen函数：监听是否有户上线

accept函数：阻塞函数,等待用户连接

connect函数：客户端函数，参数类似bind函数，客户端连接服务器

**在qt中使用linux socket函数的connect：使用::connect()即可解决**

read|write函数：客户端和服务器端之间数据的读入写出

**socket-IO复用技术：**

端口复用一避免出现address already be used:

int optval= 1;

setsockopt( socketfd，

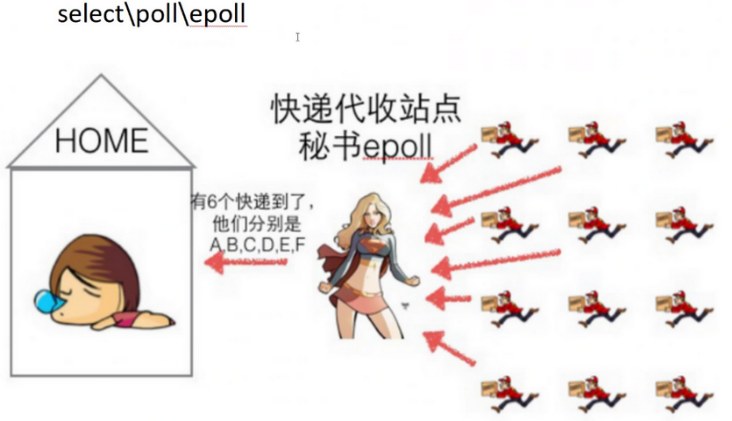
SOL\_SOCKET，

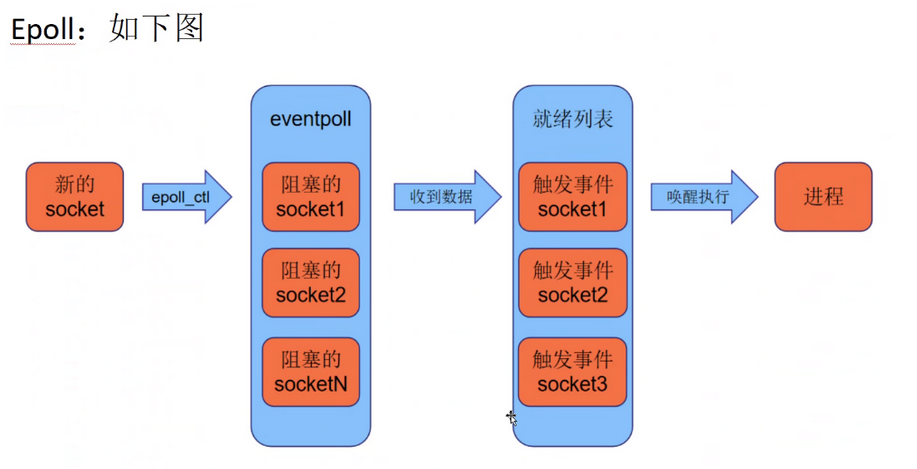
SO\_REUSEADDR，

(const void\*)&optval，

sizeof(optval))；

**IO复用技术：**





Epoll = 事件列表 + 就绪列表

Linux 网络编程API里的epoll只有eventpoll

struct epoll\_event：epoll事件结构体

int epoll\_create(int size)：创建epoll

int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event)：

epoll事件队列添加or删除事件

int epoll\_wait(int epfd, struct epoll\_event\* events, int maxevents, int timeout); 等待事件队列事件发生

**epoll解决同时接收N个客户端连接fd问题;**

**但无法解决任务并发，客户端排队问题**

线程池技术：见网络小总结

高并发服务器架构：见网络小总结

