<https://blog.csdn.net/lqglqglqg/article/details/48469793>

http://www.cnblogs.com/moodlxs/archive/2012/10/15/2724318.html

**目录索引：**

复制代码

一、LibCurl基本编程框架

二、一些基本的函数

三、curl\_easy\_setopt函数部分选项介绍

四、curl\_easy\_perform 函数说明（error 状态码）

五、libcurl使用的HTTP消息头

六、获取http应答头信息

七、多线程问题

八、什么时候libcurl无法正常工作

九、关于密码

十、HTTP验证

十一、代码示例

1.基本的http GET/POST操作

2 获取html网页

3 网页下载保存实例

4 进度条实例显示文件下载进度

5 断点续传实例

复制代码

**一、LibCurl基本编程框架**  
[libcurl](http://curl.haxx.se/)是一个跨平台的网络协议库，支持http, https, ftp, gopher, telnet, dict, file, 和ldap 协议。libcurl同样支持HTTPS证书授权，HTTP POST, HTTP PUT, FTP 上传, HTTP基本表单上传，代理，cookies,和用户认证。想要知道更多关于libcurl的介绍，可以到官网 <http://curl.haxx.se/>上去了解，在这里不再详述。

**win32版的libcurl下载地址:**<http://curl.freeby.pctools.cl/download/libcurl-7.18.0-win32-msvc.zip>

在基于LibCurl的程序里，主要采用callback function （回调函数）的形式完成传输任务，用户在启动传输前设置好各类参数和回调函数，当满足条件时libcurl将调用用户的回调函数实现特定功能。下面是利用libcurl完成传输任务的流程：

1.       调用**curl\_global\_init()**初始化libcurl  
2.       调用**curl\_easy\_init()**函数得到 easy interface型指针  
3.       调用**curl\_easy\_setopt()**设置传输选项  
4.       根据**curl\_easy\_setopt()**设置的传输选项，实现回调函数以完成用户特定任务  
5.       调用**curl\_easy\_perform()**函数完成传输任务  
6.       调用**curl\_easy\_cleanup()**释放内存  
在整过过程中设置**curl\_easy\_setopt(**)参数是最关键的，几乎所有的libcurl程序都要使用它。

**二、一些基本的函数**  
**1.CURLcode curl\_global\_init(long flags);**  
描述：  
这个函数只能用一次。(其实在调用curl\_global\_cleanup 函数后仍然可再用)  
如果这个函数在curl\_easy\_init函数调用时还没调用，它讲由libcurl库自动调用，**所以多线程下最好主动调用该函数以防止在线程中curl\_easy\_init时多次调用。**

**注意：虽然libcurl是线程安全的，但curl\_global\_init是不能保证线程安全的，所以不要在每个线程中都调用curl\_global\_init，应该将该函数的调用放在主线程中。**  
参数：flags  
CURL\_GLOBAL\_ALL                      //初始化所有的可能的调用。  
CURL\_GLOBAL\_SSL                      //初始化支持 安全套接字层。  
CURL\_GLOBAL\_WIN32            //初始化win32套接字库。  
CURL\_GLOBAL\_NOTHING         //没有额外的初始化。

**2 void curl\_global\_cleanup(void);**  
描述：在结束libcurl使用的时候，用来对curl\_global\_init做的工作清理。类似于close的函数。

**注意：虽然libcurl是线程安全的，但curl\_global\_cleanup是不能保证线程安全的，所以不要在每个线程中都调用curl\_global\_init，应该将该函数的调用放在主线程中。**

**3 char \*curl\_version( );**  
描述: 打印当前libcurl库的版本。

**4 CURL \*curl\_easy\_init( );**  
描述:  
curl\_easy\_init用来初始化一个CURL的指针(有些像返回FILE类型的指针一样). 相应的在调用结束时要用curl\_easy\_cleanup函数清理.  
一般curl\_easy\_init意味着一个会话的开始. 它会返回一个easy\_handle(CURL\*对象), 一般都用在easy系列的函数中.

**5 void curl\_easy\_cleanup(CURL \*handle);**  
描述:  
这个调用用来结束一个会话.与curl\_easy\_init配合着用.   
参数:  
CURL类型的指针.

**6 CURLcode curl\_easy\_setopt(CURL \*handle, CURLoption option, parameter);**  
描述: 这个函数最重要了.几乎所有的curl 程序都要频繁的使用它.它告诉curl库.程序将有如何的行为. 比如要查看一个网页的html代码等.(这个函数有些像ioctl函数)参数:  
1 CURL类型的指针  
2 各种CURLoption类型的选项.(都在curl.h库里有定义,man 也可以查看到)  
3 parameter 这个参数 既可以是个函数的指针,也可以是某个对象的指针,也可以是个long型的变量.它用什么这取决于第二个参数.  
CURLoption 这个参数的取值很多.具体的可以查看man手册.

**7 CURLcode curl\_easy\_perform(CURL \*handle);**

描述:这个函数在初始化CURL类型的指针 以及curl\_easy\_setopt完成后调用. 就像字面的意思所说perform就像是个舞台.让我们设置的  
option 运作起来.参数:  
CURL类型的指针.

**三、 curl\_easy\_setopt函数部分选项介绍**  
本节主要介绍curl\_easy\_setopt中跟http相关的参数。该函数是curl中非常重要的函数，curl所有设置都是在该函数中完成的，该函数的设置选项众多，注意本节的阐述的只是部分常见选项。

1.     CURLOPT\_URL   
设置访问URL

2.       CURLOPT\_WRITEFUNCTION，CURLOPT\_WRITEDATA  
回调函数原型为：**size\_t function( void \*ptr, size\_t size, size\_t nmemb, void \*stream);**函数将在libcurl接收到数据后被调用，因此函数多做数据保存的功能，如处理下载文件。CURLOPT\_WRITEDATA 用于表明CURLOPT\_WRITEFUNCTION函数中的stream指针的来源。

如果你没有通过CURLOPT\_WRITEFUNCTION属性给easy handle设置回调函数，libcurl会提供一个默认的回调函数，它只是简单的将接收到的数据打印到标准输出。你也可以通过 CURLOPT\_WRITEDATA属性给默认回调函数传递一个已经打开的文件指针，用于将数据输出到文件里。

3.      CURLOPT\_HEADERFUNCTION，CURLOPT\_HEADERDATA  
回调函数原型为 size\_t function( void \*ptr, size\_t size,size\_t nmemb, void \*stream); libcurl一旦接收到http 头部数据后将调用该函数。CURLOPT\_WRITEDATA 传递指针给libcurl，该指针表明CURLOPT\_HEADERFUNCTION 函数的stream指针的来源。

4.       CURLOPT\_READFUNCTION CURLOPT\_READDATA  
libCurl需要读取数据传递给远程主机时将调用CURLOPT\_READFUNCTION指定的函数，函数原型是：size\_t function(void \*ptr, size\_t size, size\_t nmemb,void \*stream). CURLOPT\_READDATA 表明CURLOPT\_READFUNCTION函数原型中的stream指针来源。

5.       CURLOPT\_NOPROGRESS，CURLOPT\_PROGRESSFUNCTION，CURLOPT\_PROGRESSDATA  
跟数据传输进度相关的参数。CURLOPT\_PROGRESSFUNCTION 指定的函数正常情况下每秒被libcurl调用一次，为了使CURLOPT\_PROGRESSFUNCTION被调用，CURLOPT\_NOPROGRESS必须被设置为false，CURLOPT\_PROGRESSDATA指定的参数将作为CURLOPT\_PROGRESSFUNCTION指定函数的第一个参数

6.       CURLOPT\_TIMEOUT，CURLOPT\_CONNECTIONTIMEOUT:  
CURLOPT\_TIMEOUT 由于设置传输时间，CURLOPT\_CONNECTIONTIMEOUT 设置连接等待时间

7.       CURLOPT\_FOLLOWLOCATION  
设置重定位URL

8.       CURLOPT\_RANGE: CURLOPT\_RESUME\_FROM:  
断点续传相关设置。CURLOPT\_RANGE 指定char \*参数传递给libcurl，用于指明http域的RANGE头域，例如：  
表示头500个字节：bytes=0-499  
表示第二个500字节：bytes=500-999  
表示最后500个字节：bytes=-500  
表示500字节以后的范围：bytes=500-  
第一个和最后一个字节：bytes=0-0,-1  
同时指定几个范围：bytes=500-600,601-999  
CURLOPT\_RESUME\_FROM 传递一个long参数给libcurl，指定你希望开始传递的 偏移量。

**四、 curl\_easy\_perform 函数说明（error 状态码）**  
该函数是完成curl\_easy\_setopt指定的所有选项，本节重点介绍curl\_easy\_perform的返回值。返回0意味一切ok，非0代表错误发生。主要错误码说明：  
1.    CURLE\_OK   
    任务完成一切都好  
2     CURLE\_UNSUPPORTED\_PROTOCOL  
    不支持的协议，由URL的头部指定  
3     CURLE\_COULDNT\_CONNECT  
    不能连接到remote 主机或者代理  
4     CURLE\_REMOTE\_ACCESS\_DENIED  
    访问被拒绝  
5     CURLE\_HTTP\_RETURNED\_ERROR  
    Http返回错误  
6     CURLE\_READ\_ERROR  
读本地文件错误

要获取详细的错误描述字符串，可以通过**const char \*curl\_easy\_strerror(CURLcode errornum )** 这个函数取得.

**五、libcurl使用的HTTP消息头**  
    当使用libcurl发送http请求时，它会自动添加一些http头。我们可以通过CURLOPT\_HTTPHEADER属性手动替换、添加或删除相应 的HTTP消息头。  
    Host  
    http1.1（大部分http1.0)版本都要求客户端请求提供这个信息头。  
    Pragma  
    "no-cache"。表示不要缓冲数据。  
    Accept  
    "\*/\*"。表示允许接收任何类型的数据。  
    Expect  
    以POST的方式向HTTP服务器提交请求时，libcurl会设置该消息头为"100-continue"，它要求服务器在正式处理该请求之前，返回一 个"OK"消息。如果POST的数据很小，libcurl可能不会设置该消息头。  
**自定义选项**  
    当前越来越多的协议都构建在HTTP协议之上（如：soap），这主要归功于HTTP的可靠性，以及被广泛使用的代理支持（可以穿透大部分防火墙）。 这些协议的使用方式与传统HTTP可能有很大的不同。对此，libcurl作了很好的支持。  
    自定义请求方式(CustomRequest)  
    HTTP支持GET, HEAD或者POST提交请求。可以设置CURLOPT\_CUSTOMREQUEST来设置自定义的请求方式，libcurl默认以GET方式提交请求：  
    **curl\_easy\_setopt(easy\_handle, CURLOPT\_CUSTOMREQUEST, "MYOWNREQUEST");   
  
修改消息头**  
    HTTP协议提供了消息头，请求消息头用于告诉服务器如何处理请求；响应消息头则告诉浏览器如何处理接收到的数据。在libcurl中，你可以自由的添加 这些消息头：

复制代码

struct curl\_slist \*headers=NULL; /\* init to NULL is important \*/

headers = curl\_slist\_append(headers, "Hey-server-hey: how are you?");

headers = curl\_slist\_append(headers, "X-silly-content: yes");

/\* pass our list of custom made headers \*/

curl\_easy\_setopt(easyhandle, CURLOPT\_HTTPHEADER, headers);

curl\_easy\_perform(easyhandle); /\* transfer http \*/

curl\_slist\_free\_all(headers); /\* free the header list \*/

复制代码

对于已经存在的消息头，可以重新设置它的值：

headers = curl\_slist\_append(headers, "Accept: Agent-007");

headers = curl\_slist\_append(headers, "Host: munged.host.line");

**删除消息头**  
对于一个已经存在的消息头，设置它的内容为空，libcurl在发送请求时就不会同时提交该消息头：

headers = curl\_slist\_append(headers, "Accept:");

**六、获取http应答头信息**

    发出http请求后，服务器会返回应答头信息和应答数据，如果仅仅是打印应答头的所有内容，则直接可以通过curl\_easy\_setopt(curl, CURLOPT\_HEADERFUNCTION, 打印函数)的方式来完成，这里需要获取的是应答头中特定的信息，比如应答码、cookies列表等，则需要通过下面这个函数：  
**CURLcode curl\_easy\_getinfo(CURL \*curl, CURLINFO info, ... );**  
    info参数就是我们需要获取的内容，下面是一些参数值:  
    1.CURLINFO\_RESPONSE\_CODE  
    获取应答码  
    2.CURLINFO\_HEADER\_SIZE  
    头大小  
    3.CURLINFO\_COOKIELIST  
    cookies列表

    除了获取应答信息外，这个函数还能获取curl的一些内部信息，如请求时间、连接时间等等。

    更多的参数可以参考API文档。

**七、多线程问题**  
    首先一个基本原则就是：绝对不应该在线程之间共享同一个libcurl handle(CURL \*对象)，不管是easy handle还是multi handle（本文只介绍easy\_handle）。一个线程每次只能使用一个handle。  
    libcurl是线程安全的，但有两点例外：信号(signals)和SSL/TLS handler。 信号用于超时失效名字解析(timing out name resolves)。libcurl依赖其他的库来支持SSL/STL，所以用多线程的方式访问HTTPS或FTPS的URL时，应该满足这些库对多线程 操作的一些要求。详细可以参考：  
    OpenSSL: <http://www.openssl.org/docs/crypto/threads.html#DESCRIPTION>

    GnuTLS: <http://www.gnu.org/software/gnutls/manual/html_node/Multi_002dthreaded-applications.html>

    NSS: 宣称是多线程安全的。

**八、什么时候libcurl无法正常工作**  
    传输失败总是有原因的。你可能错误的设置了一些libcurl的属性或者没有正确的理解某些属性的含义，或者是远程主机返回一些无法被正确解析的内容。  
    这里有一个黄金法则来处理这些问题：将CURLOPT\_VERBOSE属性设置为1，libcurl会输出通信过程中的一些细节。如果使用的是http协 议，请求头/响应头也会被输出。将CURLOPT\_HEADER设为1，这些头信息将出现在消息的内容中。  
    当然不可否认的是，libcurl还存在bug。  
    如果你对相关的协议了解越多，在使用libcurl时，就越不容易犯错。

**九、关于密码**  
    客户端向服务器发送请求时，许多协议都要求提供用户名与密码。libcurl提供了多种方式来设置它们。  
    一些协议支持在URL中直接指定用户名和密码，类似于： protocol://user:password@example.com/path/。libcurl能正确的识别这种URL中的用户名与密码并执行 相应的操作。如果你提供的用户名和密码中有特殊字符，首先应该对其进行URL编码。  
    也可以通过CURLOPT\_USERPWD属性来设置用户名与密码。参数是格式如 “user:password ”的字符串：  
   **curl\_easy\_setopt(easy\_handle, CURLOPT\_USERPWD, "user\_name:password")**;   
    有时候在访问代理服务器的时候，可能时时要求提供用户名和密码进行用户身份验证。这种情况下，libcurl提供了另 一个属性CURLOPT\_PROXYUSERPWD：  
    **curl\_easy\_setopt(easy\_handle, CURLOPT\_PROXYUSERPWD, "user\_name:password");**   
    在UNIX平台下，访问FTP的用户名和密码可能会被保存在$HOME/.netrc文件中。libcurl支持直接从这个文件中获取用户名与密码：  
    **curl\_easy\_setopt(easy\_handle, CURLOPT\_NETRC, 1L);**  
    在使用SSL时，可能需要提供一个私钥用于数据安全传输，通过CURLOPT\_KEYPASSWD来设置私钥：  
   **curl\_easy\_setopt(easy\_handle, CURLOPT\_KEYPASSWD, "keypassword");**

**十、HTTP验证**  
    在使用HTTP协议时，客户端有很多种方式向服务器提供验证信息。默认的 HTTP验证方法是"Basic”，它将用户名与密码以明文的方式、经Base64编码后保存在HTTP请求头中，发往服务器。当然这不太安全。  
    当前版本的libcurl支持的验证方法有：basic, Digest, NTLM, Negotiate, GSS-Negotiate and SPNEGO。（译者感叹：搞Web这么多年，尽然不知道这些Http的验证方式，实在惭愧。）可以通过CURLOPT\_HTTPAUTH属性来设置具体 的验证方式：  
    **curl\_easy\_setopt(easy\_handle, CURLOPT\_HTTPAUTH, CURLAUTH\_DIGEST);**  
    向代理服务器发送验证信息时，可以通过CURLOPT\_PROXYAUTH设置验证方式：  
   **curl\_easy\_setopt(easy\_handle, CURLOPT\_PROXYAUTH, CURLAUTH\_NTLM);**  
    也可以同时设置多种验证方式（通过按位与）， 使用‘CURLAUTH\_ANY‘将允许libcurl可以选择任何它所支持的验证方式。通过CURLOPT\_HTTPAUTH或 CURLOPT\_PROXYAUTH属性设置的多种验证方式，libcurl会在运行时选择一种它认为是最好的方式与服务器通信：  
   **curl\_easy\_setopt(easy\_handle, CURLOPT\_HTTPAUTH, CURLAUTH\_DIGEST|CURLAUTH\_BASIC);**  
**// curl\_easy\_setopt(easy\_handle, CURLOPT\_HTTPAUTH, CURLAUTH\_ANY);**

**十一、代码示例**

**下载的libcurl中自带了很多示例代码，在docs\examples目录下，建议下载该库后好好阅读一下这些代码。**

**libcurl的API文档在docs\libcurl\index.html中。**

**1.基本的http GET/POST操作**

复制代码

#include <stdio.h>

#include <curl/curl.h>

bool getUrl(char \*filename)

{

CURL \*curl;

CURLcode res;

FILE \*fp;

if ((fp = fopen(filename, "w")) == NULL) // 返回结果用文件存储

return false;

struct curl\_slist \*headers = NULL;

headers = curl\_slist\_append(headers, "Accept: Agent-007");

curl = curl\_easy\_init(); // 初始化

if (curl)

{

//curl\_easy\_setopt(curl, CURLOPT\_PROXY, "10.99.60.201:8080");// 代理

curl\_easy\_setopt(curl, CURLOPT\_HTTPHEADER, headers);// 改协议头

curl\_easy\_setopt(curl, CURLOPT\_URL,"http://www.baidu.com");

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, fp); //将返回的http头输出到fp指向的文件

curl\_easy\_setopt(curl, CURLOPT\_HEADERDATA, fp); //将返回的html主体数据输出到fp指向的文件

res = curl\_easy\_perform(curl); // 执行

if (res != 0) {

curl\_slist\_free\_all(headers);

curl\_easy\_cleanup(curl);

}

fclose(fp);

return true;

}

}

bool postUrl(char \*filename)

{

CURL \*curl;

CURLcode res;

FILE \*fp;

if ((fp = fopen(filename, "w")) == NULL)

return false;

curl = curl\_easy\_init();

if (curl)

{

curl\_easy\_setopt(curl, CURLOPT\_COOKIEFILE, "/tmp/cookie.txt"); // 指定cookie文件

curl\_easy\_setopt(curl, CURLOPT\_POSTFIELDS, "&logintype=uid&u=xieyan&psw=xxx86"); // 指定post内容

//curl\_easy\_setopt(curl, CURLOPT\_PROXY, "10.99.60.201:8080");

curl\_easy\_setopt(curl, CURLOPT\_URL, " http://mail.sina.com.cn/cgi-bin/login.cgi "); // 指定url

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, fp);

res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

}

fclose(fp);

return true;

}

int main(void)

{

getUrl("/tmp/get.html");

postUrl("/tmp/post.html");

}

复制代码

编译**gcc  get\_post.c  -o get\_post –lcurl**

**./ get\_post**

**2 获取html网页**

复制代码

#include <stdio.h>

#include <curl/curl.h>

#include <stdlib.h>

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

{

CURL \*curl; //定义CURL类型的指针

CURLcode res; //定义CURLcode类型的变量，保存返回状态码

if(argc!=2)

{

printf("Usage : file <url>;\n");

exit(1);

}

curl = curl\_easy\_init(); //初始化一个CURL类型的指针

if(curl!=NULL)

{

//设置curl选项. 其中CURLOPT\_URL是让用户指 定url. argv[1]中存放的命令行传进来的网址

curl\_easy\_setopt(curl, CURLOPT\_URL, argv[1]);

//调用curl\_easy\_perform 执行我们的设置.并进行相关的操作. 在这 里只在屏幕上显示出来.

res = curl\_easy\_perform(curl);

//清除curl操作.

curl\_easy\_cleanup(curl);

}

return 0;

}

复制代码

编译**gcc  get\_http.c  -o get\_http –lcurl**

**./ get\_http**[**www.baidu.com**](http://www.baidu.com/)

**3 网页下载保存实例**

复制代码

// 采用CURLOPT\_WRITEFUNCTION 实现网页下载保存功能

#include <stdio.h>;

#include <stdlib.h>;

#include <unistd.h>;

#include <curl/curl.h>;

#include <curl/types.h>;

#include <curl/easy.h>;

FILE \*fp; //定义FILE类型指针

//这个函数是为了符合CURLOPT\_WRITEFUNCTION而构造的

//完成数据保存功能

size\_t write\_data(void \*ptr, size\_t size, size\_t nmemb, void \*stream)

{

int written = fwrite(ptr, size, nmemb, (FILE \*)fp);

return written;

}

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

{

CURL \*curl;

curl\_global\_init(CURL\_GLOBAL\_ALL);

curl=curl\_easy\_init();

curl\_easy\_setopt(curl, CURLOPT\_URL, argv[1]);

if((fp=fopen(argv[2],"w"))==NULL)

{

curl\_easy\_cleanup(curl);

exit(1);

}

////CURLOPT\_WRITEFUNCTION 将后继的动作交给write\_data函数处理

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, write\_data);

curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

exit(0);

}

复制代码

编译gcc  save\_http.c  -o save\_http –lcurl  
./ save\_http [www.baidu.com](http://www.baidu.com/)  
  /tmp/baidu

**4 进度条实例显示文件下载进度**

复制代码

// 采用CURLOPT\_NOPROGRESS， CURLOPT\_PROGRESSFUNCTION CURLOPT\_PROGRESSDATA 实现文件传输进度提示功能

//函数采用了gtk库，故编译时需指定gtk库

//函数启动专门的线程用于显示gtk 进度条bar

#include <stdio.h>

#include <gtk/gtk.h>

#include <curl/curl.h>

#include <curl/types.h> /\* new for v7 \*/

#include <curl/easy.h> /\* new for v7 \*/

GtkWidget \*Bar;

////这个函数是为了符合CURLOPT\_WRITEFUNCTION而构造的

//完成数据保存功能

size\_t my\_write\_func(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream)

{

return fwrite(ptr, size, nmemb, stream);

}

//这个函数是为了符合CURLOPT\_READFUNCTION而构造的

//数据上传时使用

size\_t my\_read\_func(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream)

{

return fread(ptr, size, nmemb, stream);

}

//这个函数是为了符合CURLOPT\_PROGRESSFUNCTION而构造的

//显示文件传输进度，t代表文件大小，d代表传 输已经完成部分

int my\_progress\_func(GtkWidget \*bar,

double t, /\* dltotal \*/

double d, /\* dlnow \*/

double ultotal,

double ulnow)

{

/\* printf("%d / %d (%g %%)\n", d, t, d\*100.0/t);\*/

gdk\_threads\_enter();

gtk\_progress\_set\_value(GTK\_PROGRESS(bar), d\*100.0/t);

gdk\_threads\_leave();

return 0;

}

void \*my\_thread(void \*ptr)

{

CURL \*curl;

CURLcode res;

FILE \*outfile;

gchar \*url = ptr;

curl = curl\_easy\_init();

if(curl)

{

outfile = fopen("test.curl", "w");

curl\_easy\_setopt(curl, CURLOPT\_URL, url);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, outfile);

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, my\_write\_func);

curl\_easy\_setopt(curl, CURLOPT\_READFUNCTION, my\_read\_func);

curl\_easy\_setopt(curl, CURLOPT\_NOPROGRESS, 0L);

curl\_easy\_setopt(curl, CURLOPT\_PROGRESSFUNCTION, my\_progress\_func);

curl\_easy\_setopt(curl, CURLOPT\_PROGRESSDATA, Bar);

res = curl\_easy\_perform(curl);

fclose(outfile);

/\* always cleanup \*/

curl\_easy\_cleanup(curl);

}

return NULL;

}

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

{

GtkWidget \*Window, \*Frame, \*Frame2;

GtkAdjustment \*adj;

/\* Must initialize libcurl before any threads are started \*/

curl\_global\_init(CURL\_GLOBAL\_ALL);

/\* Init thread \*/

g\_thread\_init(NULL);

gtk\_init(&argc, &argv);

Window = gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL);

Frame = gtk\_frame\_new(NULL);

gtk\_frame\_set\_shadow\_type(GTK\_FRAME(Frame), GTK\_SHADOW\_OUT);

gtk\_container\_add(GTK\_CONTAINER(Window), Frame);

Frame2 = gtk\_frame\_new(NULL);

gtk\_frame\_set\_shadow\_type(GTK\_FRAME(Frame2), GTK\_SHADOW\_IN);

gtk\_container\_add(GTK\_CONTAINER(Frame), Frame2);

gtk\_container\_set\_border\_width(GTK\_CONTAINER(Frame2), 5);

adj = (GtkAdjustment\*)gtk\_adjustment\_new(0, 0, 100, 0, 0, 0);

Bar = gtk\_progress\_bar\_new\_with\_adjustment(adj);

gtk\_container\_add(GTK\_CONTAINER(Frame2), Bar);

gtk\_widget\_show\_all(Window);

if (!g\_thread\_create(&my\_thread, argv[1], FALSE, NULL) != 0)

g\_warning("can't create the thread");

gdk\_threads\_enter();

gtk\_main();

gdk\_threads\_leave();

return 0;

}

复制代码

编译export PKG\_CONFIG\_PATH=/usr/lib/pkgconfig/  
gcc progress.c –o progress ` pkg-config --libs –cflags gtk+-2..0` -lcurl –lgthread-2.0  
./ progress  <http://software.sky-union.cn/index.asp>  
  
  
**5 断点续传实例**

复制代码

//采用CURLOPT\_RESUME\_FROM\_LARGE 实现文件断点续传功能

#include <stdlib.h>

#include <stdio.h>

#include <sys/stat.h>

#include <curl/curl.h>

//这个函数为CURLOPT\_HEADERFUNCTION参数构造

/\* 从http头部获取文件size\*/

size\_t getcontentlengthfunc(void \*ptr, size\_t size, size\_t nmemb, void \*stream) {

int r;

long len = 0;

/\* \_snscanf() is Win32 specific \*/

// r = \_snscanf(ptr, size \* nmemb, "Content-Length: %ld\n", &len);

r = sscanf(ptr, "Content-Length: %ld\n", &len);

if (r) /\* Microsoft: we don't read the specs \*/

\*((long \*) stream) = len;

return size \* nmemb;

}

/\* 保存下载文件 \*/

size\_t wirtefunc(void \*ptr, size\_t size, size\_t nmemb, void \*stream)

{

return fwrite(ptr, size, nmemb, stream);

}

/\*读取上传文件 \*/

size\_t readfunc(void \*ptr, size\_t size, size\_t nmemb, void \*stream)

{

FILE \*f = stream;

size\_t n;

if (ferror(f))

return CURL\_READFUNC\_ABORT;

n = fread(ptr, size, nmemb, f) \* size;

return n;

}

// 下载 或者上传文件函数

int download(CURL \*curlhandle, const char \* remotepath, const char \* localpath,

long timeout, long tries)

{

FILE \*f;

curl\_off\_t local\_file\_len = -1 ;

long filesize =0 ;

CURLcode r = CURLE\_GOT\_NOTHING;

int c;

struct stat file\_info;

int use\_resume = 0;

/\* 得到本地文件大小 \*/

//if(access(localpath,F\_OK) ==0)

if(stat(localpath, &file\_info) == 0)

{

local\_file\_len = file\_info.st\_size;

use\_resume = 1;

}

//采用追加方式打开文件，便于实现文件断点续传工作

f = fopen(localpath, "ab+");

if (f == NULL) {

perror(NULL);

return 0;

}

//curl\_easy\_setopt(curlhandle, CURLOPT\_UPLOAD, 1L);

curl\_easy\_setopt(curlhandle, CURLOPT\_URL, remotepath);

curl\_easy\_setopt(curlhandle, CURLOPT\_CONNECTTIMEOUT, timeout); // 设置连接超时，单位秒

//设置http 头部处理函数

curl\_easy\_setopt(curlhandle, CURLOPT\_HEADERFUNCTION, getcontentlengthfunc);

curl\_easy\_setopt(curlhandle, CURLOPT\_HEADERDATA, &filesize);

// 设置文件续传的位置给libcurl

curl\_easy\_setopt(curlhandle, CURLOPT\_RESUME\_FROM\_LARGE, use\_resume?local\_file\_len:0);

curl\_easy\_setopt(curlhandle, CURLOPT\_WRITEDATA, f);

curl\_easy\_setopt(curlhandle, CURLOPT\_WRITEFUNCTION, wirtefunc);

//curl\_easy\_setopt(curlhandle, CURLOPT\_READFUNCTION, readfunc);

//curl\_easy\_setopt(curlhandle, CURLOPT\_READDATA, f);

curl\_easy\_setopt(curlhandle, CURLOPT\_NOPROGRESS, 1L);

curl\_easy\_setopt(curlhandle, CURLOPT\_VERBOSE, 1L);

r = curl\_easy\_perform(curlhandle);

fclose(f);

if (r == CURLE\_OK)

return 1;

else {

fprintf(stderr, "%s\n", curl\_easy\_strerror(r));

return 0;

}

}

int main(int c, char \*\*argv) {

CURL \*curlhandle = NULL;

curl\_global\_init(CURL\_GLOBAL\_ALL);

curlhandle = curl\_easy\_init();

//download(curlhandle, "ftp://user:pass@host/path/file", "C:\\file", 0, 3);

download(curlhandle , "http://software.sky-union.cn/index.asp","/work/index.asp",1,3);

curl\_easy\_cleanup(curlhandle);

curl\_global\_cleanup();

return 0;

}

复制代码

编译gcc  resume.c  -o resume –lcurl  
./ resume  