搞了几个跨Linux与Windows的项目，碰到很多问题，记下来，以供查考。另外，因为水平的原因，肯定错误在所难免，请读者一定指正。

　　如无其它说明，本文所指Linux均表示2.6内核Linux，GCC编译器，Windows均表示Windows XP系统，Visual Studio 2005 sp1编译环境。

　　下面大概分几个方面进行罗列：

**socket**

　　Linux要包含

#include <sys/socket.h>  
#include <netinet/in.h>  
#include <netdb.h>  
#include <arpa/inet.h>  
等头文件，而windows下则是包含 #include <winsock.h>  
。

　　Linux中socket为整形，Windows中为一个SOCKET。

　　Linux中关闭socket为close，Windows中为closesocket。

　　Linux中有变量socklen\_t，Windows中直接为int。

　　因为linux中的socket与普通的fd一样，所以可以在TCP的socket中，发送与接收数据时，直接使用read和write。而windows只能使用recv和send。

　　设置socet选项，比如设置socket为非阻塞的。Linux下为

flag = fcntl (fd, F\_GETFL);  
fcntl (fd, F\_SETFL, flag | O\_NONBLOCK);  
，Windows下为 flag = 1;  
ioctlsocket (fd, FIONBIO, (unsigned long \*) &flag);  
。

　　当非阻塞socket的TCP连接正在进行时，Linux的错误号为EINPROGRESS，Windows的错误号为WSAEWOULDBLOCK。

**file**

　　Linux下面，文件换行是"\n"，而windows下面是"\r\n"。

　　Linux下面，目录分隔符是"/"，而windows下面是"\"。

　　Linux与Windows下面，均可以使用stat调用来查询文件信息。但是，Linux只支持2G大小，而Windows只支持4G大小。 为了支持更大的文件查询，可以在Linux环境下加\_FILE\_OFFSET\_BITS=64定义，在Windows下面使用\_stat64调用，入参为 struct \_\_stat64。

　　Linux中可根据stat的st\_mode判断文件类型，有S\_ISREG、S\_ISDIR等宏。Windows中没有，需要自己定义相应的宏，如

#define S\_ISREG(m) (((m) & 0170000) == (0100000))  
#define S\_ISDIR(m) (((m) & 0170000) == (0040000))

　　Linux中删除文件是unlink，Windows中为DeleteFile。

**time**

　　Linux中，time\_t结构是32位的无符号整形。而windows中，time\_t结构是64位的整形。如果要在windows始time\_t为32位无符号整形，可以加宏定义，\_USE\_32BIT\_TIME\_T。

　　Linux中，sleep的单位为秒。Windows中，Sleep的单位为毫秒。即，Linux下sleep (1)，在Windows环境下则需要Sleep (1000)。

　　Windows中的timecmp宏，不支持大于等于或者小于等于。

　　Windows中没有struct timeval结构的加减宏可以使用，需要手动定义：

#define MICROSECONDS (1000 \* 1000)  
  
#define timeradd(t1, t2, t3) do { \  
(t3)->tv\_sec = (t1)->tv\_sec + (t2)->tv\_sec; \  
(t3)->tv\_usec = (t1)->tv\_usec + (t2)->tv\_usec % MICROSECONDS; \  
if ((t1)->tv\_usec + (t2)->tv\_usec > MICROSECONDS) (t3)->tv\_sec ++; \  
} while (0)  
  
#define timersub(t1, t2, t3) do { \  
(t3)->tv\_sec = (t1)->tv\_sec - (t2)->tv\_sec; \  
(t3)->tv\_usec = (t1)->tv\_usec - (t2)->tv\_usec; \  
if ((t1)->tv\_usec - (t2)->tv\_usec < 0) (t3)->tv\_usec --, (t3)->tv\_usec += MICROSECONDS; \  
} while (0)

**process**

　　Linux下可以直接使用system来调用外部程序。Windows最好使用WinExec，因为WinExec可以支持是打开还是隐藏程序窗口。用WinExec的第二个入参指明，如SW\_SHOW/SW\_HIDE。

**杂项**

　　Linux为srandom和random函数，Windows为srand和rand函数。

　　Linux为snprintf，Windows为\_snprintf。

　　同理，Linux中的strcasecmp，Windows为\_stricmp。

**错误处理**

　　Linux下面，通常使用全局变量errno来表示函数执行的错误号。Windows下要使用GetLastError ()调用来取得。

**Linux环境下仅有的**

　　这些函数或者宏，Windows中完全没有，需要用户手动实现。

　　atoll

long long  
atoll (const char \*p)  
{  
int minus = 0;  
long long value = 0;  
if (\*p == '-')  
{  
minus ++;  
p ++;  
}  
while (\*p >= '0' && \*p <= '9')  
{  
value \*= 10;  
value += \*p - '0';  
p ++;  
}  
return minus ? 0 - value : value;  
}

gettimeofday

#if defined(\_MSC\_VER) || defined(\_MSC\_EXTENSIONS)  
#define EPOCHFILETIME 11644473600000000Ui64  
#else  
#define EPOCHFILETIME 11644473600000000ULL  
#endif  
  
struct timezone  
{  
int tz\_minuteswest;  
int tz\_dsttime;  
};  
  
int  
gettimeofday (struct timeval \*tv, struct timezone \*tz)  
{  
FILETIME ft;  
LARGE\_INTEGER li;  
\_\_int64 t;  
static int tzflag;  
  
if (tv)  
{  
GetSystemTimeAsFileTime (&ft);  
li.LowPart = ft.dwLowDateTime;  
li.HighPart = ft.dwHighDateTime;  
t = li.QuadPart; /\* In 100-nanosecond intervals \*/  
t -= EPOCHFILETIME; /\* Offset to the Epoch time \*/  
t /= 10; /\* In microseconds \*/  
tv->tv\_sec = (long) (t / 1000000);  
tv->tv\_usec = (long) (t % 1000000);  
}  
  
if (tz)  
{  
if (!tzflag)  
{  
\_tzset ();  
tzflag++;  
}  
tz->tz\_minuteswest = \_timezone / 60;  
tz->tz\_dsttime = \_daylight;  
}  
  
return 0;  
}

**编译相关**

　　当前函数，Linux用\_\_FUNCTION\_\_表示，Windows用\_\_func\_\_表示。