L

```
unsigned __stdcall ThreadFuncProducer(void *p)
{
    printf("Thread Producer: Start\n");
    while (nProduced < MAX_PRODUCTS) {
        EnterCriticalSection(&cs);
        if (nProducts < MAX_STORAGE) {
            nProducts++;
            nProduced++;
            Sleep(PRODUCE_TIME);
        }
        LeaveCriticalSection(&cs);
        Sleep(SPEED * (GetSysTime() % 10));
    }
    printf("Thread Producer: Finished\n");
    return 0;
}

unsigned __stdcall ThreadFuncConsumer(void *p)
{
    printf("Thread Consumer: Start\n");
    while (nConsumed < MAX_PRODUCTS) {
        EnterCriticalSection(&cs);
        if (nProducts > 0) {
            nProducts--;
            nConsumed++;
            Sleep(CONSUME_TIME);
        }
        LeaveCriticalSection(&cs);
        Sleep(SPEED * (GetSysTime() % 10));
    }
    printf("Thread Consumer: Finished\n");
    return 0;
}

BOOLEAN IsThreadRunning(HANDLE h) {
    DWORD dwExitCode = 0;
    BOOL bSuccess = GetExitCodeThread(h, &dwExitCode);
    if (bSuccess && (dwExitCode == THR_EXITCODE_ACTIVE)) {
        return TRUE;
    }
    return FALSE;
}

int main()
{
    DWORD threadid;
    HANDLE hThreadInc, hThreadDec;

    InitializeCriticalSection(&cs);
```

Windows to Linux Porting Library

(W2LPL)

www.adontec.com

ADONTEC®

This document describes how to use the W2LPL library within a custom application.

Technical Support

You can reach ADONTEC technical support by email at support@adontec.com or most secure, via http://adontec.com | Support.

Copyright Notice

Introduction

The process of porting code from Microsoft Windows* to Linux is not always an easy task. There are many differences between the APIs provided by Windows and Linux. In many cases it is not clear and by far not that easy to replace Windows specific code with equivalent Linux code.

The **Windows to Linux Porting Library** (W2LPL) allows to easily port code from Windows to Linux operating systems. The library offers access to many commonly used functions provided by the Windows API.

In order to correctly use this library please consult the documentation that follows in the next sections.

How to use?

In order to use the W2LPL library the header file *w2lpl.h* must be included into the C++ project. To compile and link the project use the library *libw2lpl.so.* Place this library to a directory that can be found by the compiler and linker. A simple example is shown below. For more details please consult the samples provided.

Example:

Consider the following makefile example. Suppose we have the following file structure:

```
/usr/home/dev/adontec/W2LPL/common/
        libw2lpl.so
        ...
/usr/home/dev/adontec/W2LPL/tests/simple_test_project/
        w2lpl.h
        test.cpp
        makefile
```

The makefile is as follows:

```
DEFS :=
OBJS := ./test.o
LIBS_PATH := "./../../common/"
LIBS_RPATH := '$$ORIGIN/../../common/'
EXE_NAME := test
LIBS := -L$(LIBS_PATH) -lw2lpl

all: $(EXE_NAME)

# Link objects
$(EXE_NAME): $(OBJS)
	g++ -Wl,-rpath=$(LIBS_RPATH) -o"$@" $(OBJS) $(LIBS)

# Compile C++ source files
%.o: ./%.cpp
	g++ $(DEFS) -Wall -c -o"$@" "$<"
```

* The standard WIN32 API of Windows (x86/x64).

To build this project open a console and go to the project folder, e.g.:

```
$ cd /usr/home/dev/adontec/W2LPL/tests /simple_test_project
```

Execute makefile:

```
$ make all
```

What to ship ?

Include the shared library *libw2lpl.so* with your application, if your are using it. Nothing to ship if using the static library.

The functions of W2LPL

Most of the functions included in this library are already documented in the Windows API manual.

The functionality of the functions is kept the same as in Windows. In some functions there may exist some limitations. For more details please read the description of each function.

Time

Sleep

Syntax:

```
void WINAPI Sleep(DWORD ms);
```

Description:

Suspends the current thread for *ms* milliseconds.

GetSysTime

Syntax:

```
TSYSTIME WINAPI GetSysTime(void);
```

Description:

The functions *GetSysTime* and *GetElapsedTime* offer time measurement with a millisecond precision.
The return value of this function is the current timestamp.

Example:

```
TSYSTIME startTime = GetSysTime();
// do some work
...
printf("Elapsed time = %d\n", GetElapsedTime(startTime));
```

GetElapsedTime

Syntax:

```
TSYSTIME WINAPI GetElapsedTime(TSYSTIME Start);
```

Description:

The functions *GetSysTime* and *GetElapsedTime* offer time measurement with a millisecond precision.
The return value of this function is the time in milliseconds that elapsed since *Start*.

Critical Section

InitializeCriticalSection

Syntax:

```
void WINAPI InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Description:

Initializes a critical section object.
For details please consult the Windows API documentation.

DeleteCriticalSection

Syntax:

```
void WINAPI DeleteCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Description:

Releases all resources used by an unowned critical section object.
For details please consult the Windows API documentation.

EnterCriticalSection

Syntax:

```
void WINAPI EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Description:

Waits for ownership of the specified critical section object. The function returns when the calling thread is granted ownership.
For details please consult the Windows API documentation.

LeaveCriticalSection

Syntax:

```
void WINAPI LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Description:

Releases ownership of the specified critical section object.
For details please consult the Windows API documentation.

TryEnterCriticalSection

Syntax:

```
BOOL WINAPI TryEnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

Description:

Attempts to enter a critical section without blocking. If the call is successful, the calling thread takes ownership of the critical section.
For details please consult the Windows API documentation.

Inifile

WritePrivateProfileString

Syntax:

```
BOOL WINAPI WritePrivateProfileString(
      LPCTSTR lpAppName,
      LPCTSTR lpKeyName,
      LPCTSTR lpString,
      LPCTSTR lpFileName);
```

Description:

Copies a string into the specified section of an initialization file.
For details please consult the Windows API documentation.

GetPrivateProfileString

Syntax:

```
DWORD WINAPI GetPrivateProfileString(
      LPCTSTR lpAppName,
      LPCTSTR lpKeyName,
      LPCTSTR lpDefault,
      LPTSTR  lpReturnedString,
      DWORD   nSize,
      LPCTSTR lpFileName);
```

Description:

Retrieves a string from the specified section in an initialization file.
For details please consult the Windows API documentation.

GetPrivateProfileInt

Syntax:

```
DWORD WINAPI GetPrivateProfileInt(
      LPCTSTR lpAppName,
      LPCTSTR lpKeyName,
      int     nDefault,
      LPCTSTR lpFileName);
```

Description:

Retrieves an integer associated with a key in the specified section of an initialization file.
For details please consult the Windows API documentation.

GetPrivateProfileSection

Syntax:

```
DWORD WINAPI GetPrivateProfileSection(
      LPCTSTR lpAppName,
      LPTSTR  lpReturnedString,
      DWORD   nSize,
      LPCTSTR lpFileName);
```

Description:

Retrieves all the keys and values for the specified section of an initialization file. For details please consult the Windows API documentation.

Handle

CloseHandle

Syntax:

```
BOOL WINAPI CloseHandle(HANDLE hObject);
```

Description:

Closes an open object handle.
For details please consult the Windows API documentation.

Event

CreateEvent

Syntax:

```
HANDLE WINAPI CreateEvent(
      PVOID lpEventAttributes,
      BOOL bManualReset,
      BOOL bInitialState,
      LPCTSTR lpName);
```

Description:

Creates or opens an event object.
For details please consult the Windows API documentation.

Remarks:

The parameters *lpEventAttributes* and *lpName* are ignored.

WaitForSingleObject

Syntax:

```
DWORD WINAPI WaitForSingleObject(HANDLE hObject, DWORD dwMilliseconds);
```

Description:

Wait on an event object.
For details please consult the Windows API documentation.

WaitForMultipleObjects

Syntax:

```
DWORD WINAPI WaitForMultipleObjects(
      DWORD    dwCount,
      const HANDLE *lpHandles,
      BOOL     bWaitAll,
      TSYSTIME dwMilliseconds);
```

Description:

Wait on multiple event objects.
For details please consult the Windows API documentation.

SetEvent

Syntax:

```
BOOL WINAPI SetEvent(HANDLE hObject);
```

Description:

Sets the specified event object to the signaled state.
For details please consult the Windows API documentation.

PulseEvent

Syntax:

```
BOOL WINAPI PulseEvent(HANDLE hObject);
```

Description:

Sets the specified event object to the signaled state and then resets it to the nonsignaled state after releasing the appropriate number of waiting threads.
For details please consult the Windows API documentation.

ResetEvent

Syntax:

```
BOOL WINAPI ResetEvent(HANDLE hObject);
```

Description:

This function sets the state of the specified event object to nonsignaled.
For details please consult the Windows API documentation.

Threads

_beginthreadex

Syntax:

```
unsigned long WINAPI _beginthreadex(
      void *security,
      unsigned stack_size,
      PSTART_ADDRESSEX pStartAddress,
      void *arglist,
      unsigned initflag,
      unsigned *thrdaddr);
```

Description:

Create a thread.
For details please consult the Windows API documentation.

Remarks:

The parameters *security* and *initflag* are ignored.

_beginthread

Syntax:

```
unsigned long WINAPI _beginthread(
      PSTART_ADDRESS pStartAddress,
      unsigned stack_size,
      void *arglist);
```

Description:

Create a thread.
For details please consult the Windows API documentation.

GetCurrentThread

Syntax:

```
HANDLE WINAPI GetCurrentThread();
```

Description:

Retrieves the handle of the calling thread.
For details please consult the Windows API documentation.

GetCurrentThreadId

Syntax:

```
DWORD WINAPI GetCurrentThreadId(void);
```

Description:

Retrieves the thread identifier of the calling thread.
For details please consult the Windows API documentation.

GetExitCodeThread

Syntax:

```
BOOL WINAPI GetExitCodeThread(HANDLE hThread, LPDWORD lpExitCode);
```

Description:

Retrieves the termination status of the specified thread.
For details please consult the Windows API documentation.

File operations

FindFirstFile

Syntax:

```
HANDLE WINAPI FindFirstFile(
      LPCTSTR lpFileName,
      LPWIN32_FIND_DATA lpFindFileData);
```

Description:

Searches a directory for a file or subdirectory with a name that matches a specific name (or partial name if wildcards are used).
For details please consult the Windows API documentation.

FindNextFile

Syntax:

```
BOOL WINAPI FindNextFile(HANDLE hFindFile, LPWIN32_FIND_DATA lpFindFileData);
```

Description:

Continues a file search from a previous call to the *FindFirstFile*.
For details please consult the Windows API documentation.

FindClose

Syntax:

```
void WINAPI FindClose(HANDLE hFindFile);
```

Description:

Closes a file search handle opened by the *FindFirstFile*.
For details please consult the Windows API documentation.

Libraries

Dynamic loadable program modules are called dynamic link libraries (.dll) in Windows and shared libraries (.so) in Linux. The following functions provide access to shared libraries (.so) under Linux using the Windows similar syntax.

LoadLibrary

Syntax:

```
HMODULE WINAPI LoadLibrary(LPCTSTR lpFileName);
```

Description:

Loads the specified shared library (.so) into the address space of the calling process. The specified module may cause other modules to be loaded.
For details please consult the Windows API documentation.

FreeLibrary

Syntax:

```
BOOL WINAPI FreeLibrary(HMODULE hModule);
```

Description:

Frees the loaded shared library (.so).
For details please consult the Windows API documentation.

GetProcAddress

Syntax:

```
FARPROC WINAPI GetProcAddress(HMODULE hModule, LPCSTR lpProcName);
```

Description:

Retrieves the address of an exported function or variable from the specified shared library.
For details please consult the Windows API documentation.

Samples

Based on the owned W2LPL license one or more samples are included to demonstrate the use of the library:

```
play_with_cs, play_with_event, test_findfile.
```