Jacek Matulewski
http://www.fizyka.umk.pl/~jacek/

Hooks

Wersja α

Toruń, 15 grudnia 2006

Najnowsza wersja dokumentu dostępna pod adresem
http://www.fizyka.umk.pl/~jacek/dydaktyka/cpp/cpp-hooks.pdf

Źródła programów z tego skryptu (C++Builder 6):
http://www.fizyka.umk.pl/~jacek/dydaktyka/cpp/cpp-dll2.zip

Hook procedure

Możliwe jest monitorowanie komunikatów lokalne (na poziomie jednego procesu, wątku) oraz globalne (przechwytywanie komunikatów przesyłanych między dowolnymi aplikacjami). W tym pierwszym przypadku funkcja zakotwiczona (*hook procedure*) może znajdować się wewnątrz modułu, którego dotyczy np. zdefiniowana w pliku *.exe*, w którym ustawiany jest hak (*hook*) i do którego przesyłane są komunikaty. W drugim przypadku funkcja ta musi być umieszczona w bibliotece DLL, żeby mogła być ładowana do przestrzeni adresowej innych aplikacji i tam uruchamiana.

Zajmiemy się hakami na komunikatach dotyczących naciskania klawiszy identyfikowanych przez stałą `WH_KEYBOARD`. Są to dwa komunikaty: `WM_KEYDOWN` i `WM_KEYUP` (klawisz naciśnięty i zwolniony). Monitorowane są komunikaty wysłane do kolejki komunikatów.

Zgodnie z dokumentacją WinAPI funkcja zakotwiczona dla tego haka powinna mieć sygnaturę zgodną z:

```
LRESULT CALLBACK KeyboardProc(
    int code,    // hook code
    WPARAM wParam,      // virtual-key code
    LPARAM lParam       // keystroke-message information
   );
```

Pierwszy parametr informuje o tym, czy ****

Znaczenie kolejnych parametrów jest identyczne, jak w komunikatach `WM_KEYDOWN` i `WM_KEYUP`. Parametr `wParam` informuje o kodzie naciśniętego klawisza (włączając w to klawisze specjalne). Parametr lParam zawiera zestaw informacji o kontekście naciśnięcia klawisza, m.in. ostatni bit (31) informuje o tym, czy klawisz jest naciskany, czy zwalniany.

Local hook w pliku .exe

Tworzymy projekt aplikacji. Na formie umieszczamy dwa przycisku `TButton` i komponent `TLabel` (zob. źródła).

Listing

```
HHOOK uchwytHooka=NULL;


void __fastcall TForm1::Button1Click(TObject *Sender)
{
    //deklaracja typu funkcji obslugujacej hooka zwiazanego z komunikatem o nacisnietym
klawiszu
    LRESULT CALLBACK KeyboardProc(
        int code,       // hook code
        WPARAM wParam,  // virtual-key code
        LPARAM lParam   // keystroke-message information
      );


    //SetWindowHookEx(typ hhoka,adres funkcji obslugujacej,uchwyt biblioteki z funkcja
obsl.,id. watku)
    //pilnujemy, zeby ustawiac tylko jeden hak
    if (uchwytHooka==NULL)
uchwytHooka=SetWindowsHookEx(WH_KEYBOARD,(HOOKPROC)KeyboardProc,HInstance,0);
    if (uchwytHooka==NULL) MessageBox(NULL,"Założenie hooka nie powiodło
się","KeyHook",MB_OK | MB_ICONERROR);
```

```
    else MessageBox(NULL,"Założenie hooka udało się","KeyHook",MB_OK |
MB_ICONINFORMATION);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    bool wynik=UnhookWindowsHookEx(uchwytHooka);
    uchwytHooka=NULL;
    if (wynik) MessageBox(NULL,"Usunięcie global hook udało się", "KeyHook", MB_OK |
MB_ICONINFORMATION);
    else MessageBox(NULL,"Usunięcie global hook nie powiodło się", "KeyHook", MB_OK |
MB_ICONERROR);
}
//---------------------------------------------------------------------------

LRESULT CALLBACK KeyboardProc(
                    int code,  // hook code
                    WPARAM wParam,     // virtual-key code
                    LPARAM lParam      // keystroke-message information
                    )
{
    //ShowMessage("Hook!");

    Form1->Label2->Caption=IntToStr(wParam);
    if((lParam & 0x80000000)==0) Beep(150,50);
    else Beep(50,50);

    return CallNextHookEx(uchwytHooka,code,wParam,lParam); //przekazywanie hooka dalej
}
```

Pilnujemy, aby tworzyć tylko jeden hak, bo inaczej dźwięk byłby odtwarzany w serii haków (wszystkie wskazują na `KeyboardProc`) wiele razy.

Z jakiegoś powodu hak przestaje działać, gdy aplikacja traci fokus!!!

Global hook w bibliotece DLL

W przypadku *global hook* odpowiednią funkcję należy umieścić w bibliotece DLL tak, żeby mogła być wciągnięta do przestrzeni adresowej aplikacji, która otrzymać komunikat i wykonana przed wywołaniem funkcji obsługującej komunikaty (`WndProc`). Biblioteka ładowana jest do przestrzeni adresowej każdej aplikacji, która otzyma komunikat związany z klawiaturą. Z tego powodu funkcja zwrotna `DllMain`/`DllEntryPoint` nie jest dobrym miejscem do ustawiania haka – byłby on niepotrzebnie powielany przy zmianie focusu aplikacji. Zdefiniujemy wobec tego i wyeksportujemy funkcje `ZalozHak` i `UsunHak`, które będą umożliwiały ustawianie i usuwanie haka, a które będą używały funkcji `KeyboardProc` zdefiniowanej w tej samej bibliotece. Funkcja ta będzie zresztą niemal identyczna, jak w przypadku haka lokalnego.

Uwaga. Biblioteka z funkcją zakotwiczoną (*hook procedure*) wczytywana jest do przestrzeni adresowej każdej aplikacji otrzymującej monitorowane komunikaty. Dlatego nie należy ustawiać i usuwać haków w `DllMain`/`DllEntryPoint`.

Tworzymy projekt biblioteki DLL.

Listing

```
void __stdcall ZalozHak(void)
{
    if (MessageBox(NULL,"Czy założyć global hook?","KeyHook",MB_YESNO |
MB_ICONQUESTION)==ID_YES)
    {
        //deklaracja funkcji obslugujacej hooka zwiazanego z komunikatem o nacisnietym
klawiszu
        LRESULT CALLBACK KeyboardProc(
            int code,   // hook code
            WPARAM wParam,     // virtual-key code
            LPARAM lParam      // keystroke-message information
            );

        //SetWindowHookEx(typ hhoka,adres funkcji obslugujacej,uchwyt biblioteki z
funkcja obsl.,id. watku)

uchwytHooka=SetWindowsHookEx(WH_KEYBOARD,(HOOKPROC)KeyboardProc,uchwytDLL/*biezaca
DLL*/,NULL);
        if (uchwytHooka==NULL) MessageBox(NULL,"Założenie hooka nie powiodło
się","KeyHook",MB_OK | MB_ICONERROR);
        else MessageBox(NULL,"Założenie hooka udało się. Bedzie dzialal dotad, dopoki
biblioteka bedzie w pamieci","KeyHook",MB_OK | MB_ICONINFORMATION);
    }
}

void __stdcall UsunHak(void)
{
    if (MessageBox(NULL,"Czy usunąć global hook?","KeyHook",MB_YESNO |
MB_ICONQUESTION)==ID_YES)
    {
        bool wynik=UnhookWindowsHookEx(uchwytHooka);
        if (wynik) MessageBox(NULL,"Usunięcie global hook udało się", "KeyHook", MB_OK |
MB_ICONINFORMATION);
        else MessageBox(NULL,"Usunięcie global hook nie powiodło się", "KeyHook", MB_OK
| MB_ICONERROR);
    }
}
```

Funkcje te należy wyeksportować:

```
extern "C" __declspec(dllexport) void __stdcall ZalozHak(void);
extern "C" __declspec(dllexport) void __stdcall UsunHak(void);
```

Do działania funkcje potrzebują dwóch zmiennych globalnych:

```
HHOOK uchwytHooka=NULL;
HINSTANCE uchwytDLL=NULL;
```

Drugą z nich możemy zainicjować w metodzie wywoływanej po załadowaniu biblioteki:

Listing 2

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fwdreason, LPVOID lpvReserved)
{
  switch(fwdreason)
  {
      case DLL_PROCESS_ATTACH:
```

```
            MessageBox(NULL,"Biblioteka KeyHook.dll załadowana do pamięci","Inicjacja
KeyHook",MB_OK | MB_ICONINFORMATION);
            uchwytDLL=hinstDLL;
            break;
        case DLL_PROCESS_DETACH:
            MessageBox(NULL,"Usuwanie biblioteki KeyHook.dll z pamięci","Kończenie
KeyHook",MB_OK | MB_ICONINFORMATION);
             break;
    }

    return 1;
}
```

Musimy jeszcze zdefiniować funkcję zwrotną wywoływaną w przypadku wykrycia monitorowanych komunikatów. Możemy ją skopiować z poprzedniego paragrafu usuwając jedynie linie odwołującą się do komponentu `Label2`.

Listing 3

```
LRESULT CALLBACK KeyboardProc(
                    int code,  // hook code
                    WPARAM wParam,     // virtual-key code
                    LPARAM lParam      // keystroke-message information
                    )
{
    // MessageBox(NULL,"Hook!","",MB_OK);
    if (code>=HC_ACTION)
    {
        if((lParam & 0x80000000)==0) Beep(150,50);
        else Beep(50,50);
    }
    return CallNextHookEx(uchwytHooka,code,wParam,lParam); //przekazywanie hooka dalej
}
```

Zgodnie z informacją w dokumentacji WinAPI funkcja nie powinna być wykonana i jej działanie natychmiast przekazane do następnej funkcji zakotwiczonej jeżeli parametr `code` jest mniejszy od zera (tj. od stałej `HC_ACTION`).

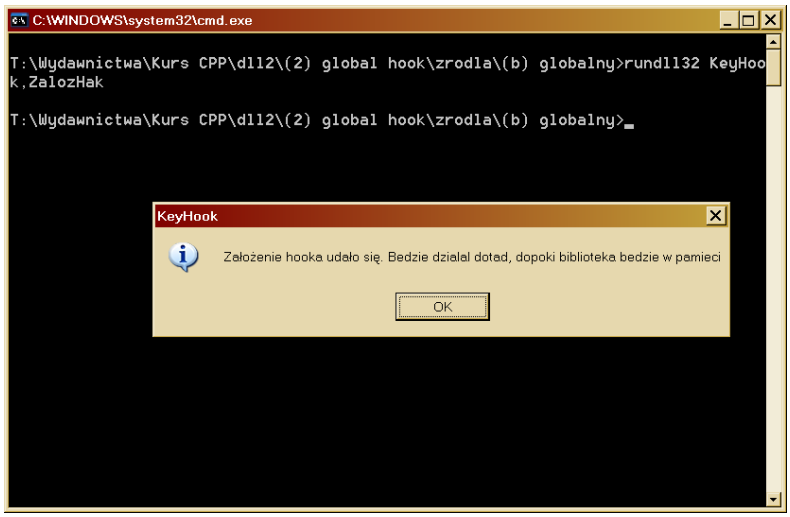
Skompilujmy projekt (*Ctrl+F9*).

Moglibyśmy teraz przygotować aplikację ładującą tak przygotowaną bibliotekę do pamięci i wywołującą funkcję `ZalozHak`, ale możemy też po prostu uruchomić tę funkcje z linii komend poleceniem:

```
rundll32 KeyHook,ZalozHak
```

Pojawi się okno informujące o wczytaniu biblioteki do pamięci. Następnie pytanie o to, czy wczytać hak – odpowiadamy tak. I wreszcie zobaczymy informację z rysunku 1. Nie klikajmy na razie OK, bo spowodowałoby to usunięcie biblioteki z hakiem z pamięci. Póki to okno jest widoczne, czyli dopóki biblioteka jest w pamięci, to każde naciśnięcie klawisza powoduje wyemitowanie krótkiego sygnału dźwiękowego o tonie zależącym od tego, czy klawisz jest przyciskany, czy zwolniony.

Jeżeli zmienimy aktywną aplikację na inną i naciśniemy klawisz, to nasza biblioteka zostanie załadowana i do jej przestrzeni adresowej (dowodzi tego komunikat o załadowaniu biblioteki).



Generator liczb losowych

Generowanie liczb losowych w komputerach, maszynach z natury w pełni deterministycznych, jest problemem nie do pokonania. W zamian korzystamy z szybko zmiennych funkcji, które inicjowane bieżącym czasem generują tzw. liczby pseudolosowe. Z punktu widzenia szyfrowania danych lub innych zastosowań związanych z bezpieczeństwem może to jednak być niewystarczające. Jest jednak jeden element, który w systemach komputerowych nie jest w pełni przewidywalny – użytkownik. Ta sztuczka, podobnie jak w dowodzie twierdzenia Gödla-Malcewa, pozwala przerzucenie pomostu między dwoma, z pozoru całkowicie odrębnymi światami.

Poniższy projekt nie jest w pełni działającą realizacją generatora liczb losowych, a jedynie szkicem pomysłu, w jaki sposób taki generator zrealizować. Jego zadaniem będzie zapełnianie wskazanego pliku liczbami, co do których nie może być wątpliwości, że są całkowicie losowe. Zapisywanie tych liczb nie będzie jednak tak szybkie, aby starczyło np. przy wielkich obliczeniach numerycznych.

Pomysł jest następujący: założymy hak na zdarzenia związane z klawiaturą, a dokładnie na zwalnianie klawiszy (aby zapobiec wykorzystaniu komunikatów związanych z automatycznym powtarzaniem przy stale naciśniętym klawiszu). Następnie do pliku będziemy zapisywali ostatnią cyfrę w ilości milisekund, jaka w momencie naciśnięcia klawisza minęła od włączenia komputera (znana już nam funkcja `GetTickCount`). Milisekundy (tysięczne części sekundy) biegną tak szybko, że człowiek nie jest zupełnie w stanie kontrolować palców w tej skali czasu.

Oczywiście istnieją możliwości oszukiwania naszego generatora: można wyobrazić sobie aplikację, która będzie sztucznie generować komunikaty. Ale to zagadnienie do zupełnie innej historii.

Tworzymy projekt biblioteki DLL (C++; Use VCL, VC++ Style DLL).

Umieszczamy w niej i eksportujemy funkcje `ZalozHak` i `UsunHak` o identycznych sygnaturach i niemal identycznej zawartości jak, w poprzednim projekcie (usuwamy jedynie wywołania funkcji `MessageBox`). Podobnie funkcje `DllMain`. Zresztą one są podobne w większości bibliotek DLL związanych z hakami.

```
void __stdcall ZalozHak(void)
{
```

```
    //deklaracja funkcji obslugujacej hooka zwiazanego z komunikatem o nacisnietym
klawiszu
    LRESULT CALLBACK KeyboardProc(
        int code,       // hook code
        WPARAM wParam,  // virtual-key code
        LPARAM lParam   // keystroke-message information
        );


    //SetWindowHookEx(typ hhoka,adres funkcji obslugujacej,uchwyt biblioteki z funkcja
obsl.,id. watku)
    uchwytHooka=SetWindowsHookEx(WH_KEYBOARD,(HOOKPROC)KeyboardProc,uchwytDLL/*biezaca
DLL*/,NULL);
    if (uchwytHooka==NULL) MessageBox(NULL,"Hook setting
failed","TrueRandomNumbersGenerator.dll",MB_OK | MB_ICONERROR);
}


void __stdcall UsunHak(void)
{
    bool wynik=UnhookWindowsHookEx(uchwytHooka);
    if (!wynik) MessageBox(NULL,"Hook unhooking failed",
"TrueRandomNumbersGenerator.dll", MB_OK | MB_ICONERROR);
}
```

Inne zadania będzie jednak realizować funkcja zakotwiczona. Jednak zanim przejdziemy do jej definiowania przygotujmy osobny moduł *Kolejka*, w którym zdefiniujemy klasę implementującą kolejkę. Do przechowywania jej elementów wykorzystamy zwykły plik tekstowy. O ile dodawanie elementów na koniec pliku nie stanowi żadnego problemu, to zdejmowanie elementu z początku tak pomyślanej kolejki jest poważnym problemem:

Plik nagłówka modułu:

```
//---------------------------------------------------------------------------


#ifndef KolejkaH
#define KolejkaH
//---------------------------------------------------------------------------


#include <windows.h>


//typedef unsigned char byte;
typedef unsigned short TCyfra;


class Kolejka
{
    public:
        virtual void WstawNaKoniec(const TCyfra element) = 0;
        virtual TCyfra ZdejmijZPoczatku(void) = 0;
        virtual unsigned __int64 IleZostalo(void) const = 0;
};



class KolejkaPlik : public Kolejka
{
```

```
    public:
        KolejkaPlik(const char nazwaPliku[MAX_PATH])
            :Kolejka()
        {
            strncpy(this->nazwaPliku,nazwaPliku,MAX_PATH);
        };
        void WstawNaKoniec(const TCyfra);
        TCyfra ZdejmijZPoczatku(void);
        unsigned __int64 IleZostalo(void) const;
    private:
        char nazwaPliku[MAX_PATH];
};

class KolejkaPamiec : public Kolejka
{
//vector<char>
};

#endif
```

Plik źródłowy modułu:

```
//---------------------------------------------------------------------------

#pragma hdrstop

#include "Kolejka.h"

//---------------------------------------------------------------------------

#include <fstream.h>
#include <SysUtils.hpp>

#pragma package(smart_init)

void KolejkaPlik::WstawNaKoniec(const TCyfra cyfra)
{
    //MessageBox(NULL,nazwaPliku,nazwaPliku,MB_OK);
    ofstream plik_wy;plik_wy.open(nazwaPliku,ios::out | ios::app);
    plik_wy << cyfra;
    plik_wy.close();
}

TCyfra KolejkaPlik::ZdejmijZPoczatku(void)
{
    if (!FileExists(nazwaPliku))
    {
        throw Exception("File with random numbers storage does not exist!");
```

```
        //return -1;
    }
    char nazwaPliku_new[MAX_PATH]; strcpy(nazwaPliku_new,nazwaPliku);
strcat(nazwaPliku_new,".tmp");
    char nazwaPliku_bak[MAX_PATH]; strcpy(nazwaPliku_bak,nazwaPliku);
strcat(nazwaPliku_bak,".bak");
    ifstream plik_we(nazwaPliku,ios::in);
    ofstream plik_wy(nazwaPliku_new,ios::out);
    char char_random='-';
    plik_we.get(char_random); //tu trzeba zrobic kasowanie tej cyfry
    for (;plik_we;)
    {
        char znak;
        plik_we.get(znak);
        if (!plik_we.eof()) plik_wy << znak;
    }
    plik_we.close();
    plik_wy.close();
    //DeleteFile(nazwaPliku_bak);
    MoveFileEx(nazwaPliku,nazwaPliku_bak,MOVEFILE_REPLACE_EXISTING);
    MoveFile(nazwaPliku_new,nazwaPliku);
    //MessageBox(NULL,((AnsiString)char_random).c_str(),"",MB_OK);
    return char_random-48;
}

unsigned __int64 KolejkaPlik::IleZostalo(void) const
{
  HANDLE hFile = CreateFile(
        nazwaPliku,
         0,
         FILE_SHARE_READ,
         NULL,
         OPEN_EXISTING,
         FILE_ATTRIBUTE_NORMAL,
         NULL);

    DWORD dlugoscPlikuHigh=0;
  DWORD dlugoscPlikuLow = GetFileSize(hFile, &dlugoscPlikuHigh);
    CloseHandle(hFile);
    unsigned __int64 dlugoscPliku=0x100000000*dlugoscPlikuHigh+dlugoscPlikuLow;
    return dlugoscPliku;
}
```

Obiekt reprezentujący kolejkę tworzymyw funkcji DllMain:

```
HHOOK uchwytHooka=NULL;
HINSTANCE uchwytDLL=NULL;
```

```
enum TMiejscePrzechowywania {mpPlik,mpPamiec} miejscePrzechowywania=mpPlik;
Kolejka* kolejka;

#pragma argsused
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fwdreason, LPVOID lpvReserved)
{
  switch(fwdreason)
  {
      case DLL_PROCESS_ATTACH:
            //MessageBox(NULL,"Biblioteka TrueRandomNumbersGenerator.dll załadowana do
pamięci","Inicjacja TrueRandomNumbersGenerator.dll",MB_OK | MB_ICONINFORMATION);
            uchwytDLL=hinstDLL;
            switch(miejscePrzechowywania)
            {
                case mpPlik:
                    char nazwaPliku[MAX_PATH];
                    GetTempPath(MAX_PATH,nazwaPliku); //katalog TEMP=
                    strcat(nazwaPliku,"TrueRandom.txt");
                    //MessageBox(NULL,nazwaPliku,nazwaPliku,MB_OK);
                    kolejka=new KolejkaPlik(nazwaPliku);
                    break;
            }
            break;
        case DLL_PROCESS_DETACH:
            //MessageBox(NULL,"Usuwanie biblioteki TrueRandomNumbersGenerator.dll z
pamięci","Kończenie TrueRandomNumbersGenerator.dll",MB_OK | MB_ICONINFORMATION);
             break;
    }

    return 1;
}
```

Nasza funkcja zakotwiczona będzie dokładać elementy do kolejki:

```
LRESULT CALLBACK KeyboardProc(
                    int code,  // hook code
                    WPARAM wParam,     // virtual-key code
                    LPARAM lParam      // keystroke-message information
                    )
{
    //MessageBox(NULL,"Hook!","Inicjacja KeyHook",MB_OK | MB_ICONINFORMATION);
    if (code>=HC_ACTION & ((lParam & 0x80000000)!=0))
    {
        long ticks=GetTickCount();
        TCyfra random=ticks-10*(ticks/10);

        kolejka->WstawNaKoniec(random);
        //if((lParam & 0x80000000)==0) Beep(150,50);
```

```
            //else Beep(50,50);
        }
        return CallNextHookEx(uchwytHooka,code,wParam,lParam); //przekazywanie hooka dalej
    }
```

Zdefiniujemy również w bibliotece funkcję, która będzie zdejmowała z początki kolejki liczbę i prezentować ją użytkownikowi jako liczbę w pełni losową:

```
TCyfra __stdcall TrueRandom(void)
{
    return kolejka->ZdejmijZPoczatku();
}
```

Poza tym zdefiniujemy metodę, która będzie zwracała ilość liczb losowych w magazynie:

```
unsigned __int64 __stdcall TrueRandomNumbersAmount(void)
{
    return kolejka->IleZostalo();
}
```

Korzystamy w niej z metody zdefiniowanej w klasie kolejki, która sprawdza po prostu wielkość pliku, w którym przechowywane są liczby. Obydwie funkcje muszą oczywiście zostać udostępnione:

```
extern "C" __declspec(dllexport) TCyfra __stdcall TrueRandom(void);
extern "C" __declspec(dllexport) unsigned __int64 __stdcall
TrueRandomNumbersAmount(void);
```

Przydałby się jeszcze osobny program – zwykła aplikacja, która wczyta bibliotekę, wywoła funkcję zakładającą hak i testującą liczby losowe udostępniane przez bibliotekę. Poniżej jest zbiór metod zdarzeniowych realizujących poszczególne zadania (należy zdefiniować pole `uchwytDLL` typu `HINSTANCE`):

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if (OpenDialog1->Execute())
    {
        Edit1->Text=OpenDialog1->FileName;
        Button2->Enabled=true;
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
    uchwytDLL=LoadLibrary(Edit1->Text.c_str());
    if (uchwytDLL==NULL)
    {
        Label3->Caption="Załadowanie biblioteki nie powiodło się";
        Label3->Font->Color=clRed;
    }
    else
    {
        Label3->Caption="Załadowanie biblioteki udało się";
        Label3->Font->Color=clGreen;
    }
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    if (FreeLibrary(uchwytDLL))
    {
        Label3->Caption="Usunięcie biblioteki z pamięci udało się";
        Label3->Font->Color=clGreen;
    }
    else
    {
        Label3->Caption="Usunięcie biblioteki z pamięci nie powiodło się";
        Label3->Font->Color=clRed;
    }
    uchwytDLL=NULL;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{
    if (uchwytDLL!=NULL) Button3Click(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
    typedef bool (*TZalozHak)(void);
    TZalozHak ZalozHak=(TZalozHak)GetProcAddress(uchwytDLL, "ZalozHak");
    if (ZalozHak==NULL) ShowMessage("Uruchomienie funkcji ZalozHak nie jest możliwe");
    else
    {
        if (ZalozHak())
        {
            Label3->Caption="Założenie haka udało się";
            Label3->Font->Color=clGreen;
        }
        else
        {
            Label3->Caption="Założenie haka nie powiodło się";
            Label3->Font->Color=clRed;
        }
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
```

```
{
    typedef short (*TTrueRandom)(void);
    TTrueRandom TrueRandom=(TTrueRandom)GetProcAddress(uchwytDLL,"TrueRandom");
    if (TrueRandom==NULL) ShowMessage("Uruchomienie funkcji TrueRandom nie jest
możliwe");
    else ShowMessage("Liczba losowa: "+IntToStr(TrueRandom()));
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button6Click(TObject *Sender)
{
    typedef bool (*TUsunHak)(void);
    TUsunHak UsunHak=(TUsunHak)GetProcAddress(uchwytDLL, "UsunHak");
    if (UsunHak==NULL) ShowMessage("Uruchomienie funkcji UsunHak nie jest możliwe");
    else
    {
        if (UsunHak())
        {
            Label3->Caption="Usunięcie haka udało się";
            Label3->Font->Color=clGreen;
        }
        else
        {
            Label3->Caption="Usunięcie haka nie powiodło się";
            Label3->Font->Color=clRed;
        }
    }
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button7Click(TObject *Sender)
{
    typedef unsigned __int64 (*TIleZostalo)(void);
    TIleZostalo IleZostalo=(TIleZostalo)GetProcAddress(uchwytDLL,
"TrueRandomNumbersAmount");
    if (IleZostalo==NULL) ShowMessage("Uruchomienie funkcji TrueRandomNumbersAmount nie
jest możliwe");
    else ShowMessage(IleZostalo());
}
```