Windows Internals II
Project I: Kernel-mode extensions 補足資料

金田憲二
kaneda@is.s.u-tokyo.ac.jp

平成 17 年 1 月 27 日

概 要

この課題では，Winodws Device Driver Kit (DDK) を用いて，デバイスドライバを作成する．この補足資料では，開発・実行環境の設定についてや，課題中で使用される Windows API などについて説明する．

1 はじめに

この節では，まず，環境設定や DDK の基本的な使い方について述べる．次に，Windows 上でプログラムをしたことがない人もいるかもしれないということで，Windows プログラムの基礎的な事項について説明する．

1.1 開発・実行環境の設定

このプロジェクトの課題を解く上で，以下のソフトウェアが必要となる．

Windows XP または Windows Server 2003
: この課題では，Windows のレジストリコールバックという機能を使用する．この機能は Windows XP と Windows Server 2003 でしかサポートされていないので，それ以外のバージョンの Windows を利用している学生は TA に連絡すること．

Winodws Device Driver Kit (DDK)
: デバイスドライバのコンパイル等に DDK が必要となる．なお，DDK をインストールする上で注意することが一つある．Windows XP 上でドライバのビルドなどをするならば，インストールするコンポーネントを選択する際に「Windows XP headers」と「Windows x86 libraies」を加えておく必要がある．

1.2 DDK の基本的な使い方

以下では，Windows DDK 3663 を C:\WINDDK\3663 にインストールした場合を例に，DDK の基本的な使い方について説明する．もし異なるバージョンの DDK などをインストールした場合には，適宜パス名などを変更すること．

コマンドプロンプト上からデバイスドライバなどのビルドを行うためには，Windows XP ならば

```
set DDK=C:\WINDDK\3663
%DDK%\bin\setenv.bat %DDK% chk wxp
```

と，Windows 2003 Server ならば

```
set DDK=C:\WINDDK\3663
%DDK%\bin\setenv.bat %DDK% chk wnet
```

と実行してビルド環境を設定する必要がある．この設定を行うと，以降は build などと入力することで，プログラムのビルドを行うことができる．

また，プログラムメニューから「Windows DDK 3663」⇒「Build Environments」⇒「Windows XP」⇒ 「Windows XP Checked Build Environment」などと選択しても，すでにビルド環境設定済みのコマンドプロンプトを立ち上げることができる．

1.3 Windows プログラムの基礎

windows.h という Windows アプリケーションのためのマスターインクルードファイルがあり，多くの場合これを include する．ここでは ANSI C の標準規格にはないデータ型が用意されているので（次表参照），プログラムを書く際は注意すること．

データ型	説明
BOOL	Boolean variable (shold be TRUE or FALSE)
BOOLEAN	Boolean variable (shold be TRUE or FALSE)
CHAR	8-bit Windows (ANSI) character
UCHAR	8-bit Windows (ANSI) character
WORD	16-bit unsigned integer
INT	32-bit signed integer
UINT	Unsigned INT
DWORD	32-bit unsigned integer
LONG	32-bit signed integer
ULONG	Unsigned LONG
ULONGLONG	64-bit unsigned integer
FLOAT	Floating-point variable
VOID	Any type
PWORD	WORD への (near) ポインタ
LPWORD	WORD への (far) ポインタ
HANDLE	オブジェクトのハンドル（講義資料参照）
WINAPI	システム関数のための calling convention

より詳しくは，MSDN ライブラリ中のデータ型に関する説明などを参照すること[1]．その他にも，課題中のソースコードで使われるものとして，以下のようなものがある．

- ユニコードの文字定数は L をプレフィックスとする．例えば，`L"some text"`はユニコード文字列を生成し，`"some text"`は 8-bit ANSI 文字列を生成する．
- `LPCTSTR` は，ユニコードまたは ASCII 形式の文字列定数へのポインタで，そのどちらになるかは記号定数 `UNICODE` が定義されているかによって決まる．

1.4 参考文献

MSDN ライブラリ

以下の Web ページからアクセスできる．

```
http://msdn.microsoft.com/library/default.asp
http://www.microsoft.com/japan/msdn/library
```

例えば,「windows 開発」⇒ 「windows ベースサービス」などとインデックスをたどると，Windows API に関する情報を取得することがで

[1]例えば，MSDN library（英語版）の「Development Guides」→ 「Windows API」→「Windows API Reference」→ 「Windows Data Types」

きる．英語版の方が内容が充実しており，日本語版には載っていない情報も一部ある．

Windows デバイスドライバに関する本

The Windows 2000 Device Driver Book — A Guide for Programmers — (Art Baker, Jerry Lozano 著) MICROSOFT TECHNOLOGIES SERIES

2 Trivial Driver

2.1 概要

最小限の機能しかもたない Windows のデバイスドライバ（以降 Trivial Driver と呼ぶ）を実行する．この Trivial Driver のソースは

```
http://www.i.u-tokyo.ac.jp/ss/msprojects
```

から取得できる．

`Porjects\Project1-KernelExtension\TrivialDriver\sys` 以下にある `trivial.c` が，デバイスドライバのソースコードとなっている．

`Porjects\Project1-KernelExtension\TrivialDriver\exe` 以下にある `trivialapp.c` が，そのデバイスドライバをインストールするアプリケーションのソースコードとなっている．

2.2 ビルド・実行

この TrivialDriver のビルド・実行は，以下のようにして行う．

1. 1.2 節にある手順に従ってコマンドプロンプトを立ち上げる．
2. `TrivialDriver` のあるディレクトリに移動し，

   ```
   build
   ```

   と入力し，ドライバのビルドを行う．
3. ドライバと，そのドライバをインストールするアプリケーションとを同一ディレクトリに移動する．なお，以上の 2. と 3. の手順を一括して行いたい場合には，以下のようなスクリプトを用意しておけばいい．

```
build
copy /Y exe\trivialapp.c bin\
copy /Y exe\objchk_wxp_x86\i386\trivialapp.exe bin\
copy /Y exe\objchk_wxp_x86\i386\trivialapp.pdb bin\

copy /Y sys\trivial.c bin\
copy /Y sys\objchk_wxp_x86\i386\trivial.sys bin\
copy /Y sys\objchk_wxp_x86\i386\trivial.pdb bin\
```

4. ドライバとアプリケーションの置いてあるディレクトリに移動して

```
  trivialapp.exe
```

と入力し，アプリケーションを実行する．

```
> trivialapp
Be sure driver (and service) are unloaded/removed.
Install service and driver.
Open newly installed driver.
```

と出力されれば成功．なお，プログラム中でドライバのインストールなどを行うため，実行には管理者権限を必要とする．

2.3 ソースコードの概要

2.3.1 trivial.c

`trivial.c` 中で定義されている `DriverEntry` 関数が，このドライバのエントリポイントであり，`DriverEntry` 内でドライバの初期化などを行っている．具体的には，以下のような初期化処理を行う．

- `IoCreateDevice` 関数を呼び，ドライバが I/O リクエストを処理するのに用いるデバイスデバイスを作成する．
- I/O サブシステムはドライバに I/O リクエストパケット (IRP) を送る．例えば，デバイスに関連づけられたファイルオブジェクトがオープンされると，`IRP_MJ_CREATE` というリクエストがドライバに送られてくる．その IRP が送られてきた時に呼ばれる関数を設定する．例えば，デバイスが作成された時は `TrivialCreateClose` 関数が呼ばれる．

2.3.2 trivialapp.c

TrivialDriver をインストールし，何もせずにただアンインストールする．具体的には以下のように動作する．

1. サービス制御マネージャ（Service Control Manager：SCM）のデータベースに，trivial という名のサービスを追加する．（InstallDriver 関数）．
2. そのサービスを開き，開始する．（StartDriver 関数）．
3. そのサービスを停止する．（StopDriver 関数）．
4. SCM のデータベースからサービスを削除する．（RemoveDriver 関数）．

2.4 ソースコード中で使用される Windows API

trivial.c と trivialapp.c 中で使用されるデータ型や API などについて説明する．メモリ管理や IRQL の詳細についての説明は省くので，参考文献など参照されたい．

2.4.1 NTSTATUS

多くのドライバルーチンの戻り値の型は，NTSTATUS である．返り値を確かめるにはいくつかのマクロを用いて行う．例えば，NT_SUCCESS 関数は，ドライバルーチンが成功した場合，NTSTATUS を受けとると TRUE を返す．

2.4.2 UNICODE_STRING

ユニコード文字列を定義するのに用いられる．

```
typedef struct _UNICODE_STRING {
  USHORT  Length;
  USHORT  MaximumLength;
  PWSTR  Buffer;
} UNICODE_STRING *PUNICODE_STRING;
```

メンバ

Length Buffer に格納される文字列のバイト長

MaximumLength Buffer の最長バイト長

Buffer　文字列を格納するためのバッファへのポインタ

UNICODE_STRING を初期化するには，RtlInitUnicodeString 関数を用いる．

2.4.3 DRIVER_OBJECT

カーネルモードドライバを表す．

メンバ

PDEVICE_OBJECT DeviceObject ドライバが作成したデバイスオブジェクトへのポインタ．IoCreateDevice 関数が成功するたびに自動的に更新される．

PDRIVER_EXTENSION DriverExtension ドライバ拡張へのポインタ．

PUNICODE_STRING HardwareDatabase レジストリ中のハードウェア設定情報へのパスへのポインタ．

PFAST_IO_DISPATCH FastIoDispatch ドライバの高速 I/O エントリポイントを定義している構造体へのポインタ

PDRIVER_INITIALIZE DriverInit DriverEntry ルーチンへのポインタ．

PDRIVER_STARTIO DriverStartIo StartIo ルーチンへのポインタ．

PDRIVER_UNLOAD DriverUnload Unload ルーチンへのポインタ．

PDRIVER_DISPATCH MajorFunction IRP を処理するルーチンのエントリポイントが格納される配列．

解説　DispatchXxx ルーチンは以下のように宣言される．

```
NTSTATUS
(*PDRIVER_DISPATCH) (
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    );
```

また，デバイスに関連づけられたファイルオブジェクトがアクセスされた場合に送られてくる IRP の一部を次の表に載せる．

リクエスト名	説明
IRP_MJ_CREATE	ファイルオブジェクトのオープン
IRP_MJ_READ	ファイルオブジェクトから読み込み
IRP_MJ_DEVICE_CONTROL	ファイルオブジェクトへの I/O 制御
IRP_MJ_CLOSE	ファイルオブジェクトのクローズ
IRP_MJ_CLEANUP	クリーンアップ

例えば，x をドライバオブジェクトへのポインタとするとき，生成・クローズ・クリーンアップ時にそれぞれ関数 f・関数 g・関数 h が呼ばれるように設定するには，

```
x->MajorFunction[IRP_MJ_CREATE]  = f;
x->MajorFunction[IRP_MJ_CLOSE]   = g;
x->MajorFunction[IRP_MJ_CLEANUP] = h;
```

と記述する．

2.4.4 DEVICE_OBJECT

ドライバが I/O リクエストを処理するのに用いるデバイスを表す．

メンバ（の一部）

PDRIVER_OBJECT DriverObject ドライバオブジェクトへのポインタ．DriverEntry 関数の引数．

PVOID DeviceExtension デバイス拡張へのポインタを示す．

ULONG Flags フラグ．例えばシステムとユーザ空間のバッファを I/O Manager がコピーすることによりやり取りさせたい場合，DO_BUFFERED_IO を指定．

解説

- IoCreateDevice 関数でデバイスオブジェクトを作成する．
- IoDeleteDevice 関数でデバイスオブジェクトを削除する．

2.4.5 SCM ハンドル

基本的な API の紹介など．

SCM ハンドル (型は SC_HANDLE）は以下のオブジェクトへのアクセスに用いられる．

SCM マネージャーオブジェクト サービスのインストールされたデータベースを表す．OpenSCManager 関数がこのオブジェクトへのハンドルを返すので，それを用いてサービスの追加・削除などを行う．

サービスオブジェクト インストールされたサービスを表す．CreateService 関数と OpenService 関数が，インストールされたサービスへのハンドラを返す．

データベースロック （省略）

CloseServiceHandle 関数でオブジェクトのハンドルを閉じる．

2.4.6 IRP

I/O リクエストパケット (IRP) を表す．

```
typedef struct _IRP {
  IO_STATUS_BLOCK  IoStatus;
  union {
    PVOID  SystemBuffer;
    ...
  } AssociatedIrp;
  ...
} IRP, *PIRP;
```

メンバ（の一部）

IoStatus IoCompleteRequest 関数を呼ぶ前に，ここに状態などを書き込む．

AssociatedIrp.SystemBuffer システム空間のバッファへのポインタ．

2.4.7 IO_STATUS_BLOCK

I/O リクエストの最終状態を示す．ドライバが IoCompleteRequest 関数を呼ぶ前に設定する．

```
typedef struct _IO_STATUS_BLOCK {
    union {
        NTSTATUS Status;
        PVOID Pointer;
    };
    ULONG_PTR Information;
} IO_STATUS_BLOCK, *PIO_STATUS_BLOCK;
```

メンバ（の一部）

Status　終了状態を表す．リクエストの処理に成功した場合はSTATUS_SUCCESSが，警告やエラーの場合はSTATUS_XXXが代入される．

Pointer システムによって予約されている．

Information リクエストに依存した値．

2.4.8 IO_STACK_LOCATION

個々のIRPがもつI/Oスタック位置を定義する．

```
ypedef struct _IO_STACK_LOCATION {
  UCHAR  MajorFunction;
  UCHAR  MinorFunction;
  union {
        // Parameters for IRP_MJ_READ
        //
        struct {
            ULONG Length;
            ULONG POINTER_ALIGNMENT Key;
            LARGE_INTEGER ByteOffset;
        } Read;

        //
        // Parameters for IRP_MJ_DEVICE_CONTROL
        // and IRP_MJ_INTERNAL_DEVICE_CONTROL
        //
        struct {
            ULONG POINTER_ALIGNMENT IoControlCode;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG OutputBufferLength;
            ...
        } DeviceIoControl;
        ...
    } Parameters;
  ...
} IO_STACK_LOCATION, *PIO_STACK_LOCATION;
```

メンバ（の一部）

MajorFunction I/O オペレーションの型を表す.（例えば IRP_MJ_CREATE.）

MinorFunction MajorFunction のサブファンクション.

Parameters MajorFunction へのパラメータ.

解説 IRP_MJ_READ（デバイスからシステムへのデータの転送）へのパラメータは,

Length 転送するバイト長

Key 読み込みリクエストをソートするためにドライバが用いるキー.

ByteOffset 転送処理の開始オフセット.

となっている.

IRP_MJ_DEVICE_CONTROL（I/O 制御）へのパラメータは,

IoControlCode I/O 制御コードを格納する.

InputBufferLength 入力バッファ長を示す.（ただし TransferType が METHOD_BUFFER であるとき）

OutputBufferLength 出力バッファ長を示す.（ただし TransferType が METHOD_BUFFER であるとき）

となっている.

2.4.9 CloseHandle

開いているオブジェクトハンドルを閉じる.

```
BOOL CloseHandle(
  HANDLE hObject   // オブジェクトのハンドル
);
```

パラメータ

hObject 開いているオブジェクトのハンドルを指定する.

戻り値 関数が成功すると 0 以外の値が返り，関数が失敗すると 0 が返る.

解説 ファイルやプロセスなどのハンドルを閉じる．例えば，CreateFile 関数が返したハンドルを閉じるのに用いる.

2.4.10 GetLastError

最新のエラーコードを取得する.

```
DWORD GetLastError(VOID);
```

戻り値 呼び出し側のスレッドの持つ最新のエラーコードが返る.

2.4.11 CreateFile

ファイルなどのオブジェクトを作成するか開き，そのオブジェクトをアクセスするために利用できるハンドルを返す.

```
HANDLE CreateFile(
  LPCTSTR lpFileName,           // ファイル名
  DWORD dwDesiredAccess,        // アクセスモード
  DWORD dwShareMode,            // 共有モード
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                                // セキュリティ記述子
  DWORD dwCreationDisposition, // 作成方法
  DWORD dwFlagsAndAttributes,  // ファイル属性
  HANDLE hTemplateFile          // テンプレートファイルの
                                // ハンドル
);
```

パラメータ

lpFileName 作成または開く対象のオブジェクトの名前を保持する.

dwDesiredAccess オブジェクトへのアクセスのタイプを指定する．例えば，0 でデバイス問い合わせアクセス，GENERIC_READ で読みとりアクセス，GENERIC_WRITE で書き込みアクセスを指定.

dwShareMode オブジェクトの共有方法を指定する．例えば，共有しない時は 0 を指定.

lpSecurityAttributes 取得したハンドルを子プロセスへ継承することを許可するかどうかを指定する．例えば，継承を許可しない場合は NULL を指定.

dwCreationDisposition ファイルが存在する・しない場合のファイルの扱い方を指定する．例えば，CREATE_NEW とすると，新しくファイルを作成する.(指定したファイルが既に存在する場合，この関数は失敗する)．CREATE_EXISTING とすると，ファイルが存在しない場合この関数は失敗する.

dwFlagsAndAttributes ファイルの属性とフラグを指定する．例えば，とくに属性を指定しない場合は FILE_ATTRIBUTE_NORMAL を指定.

hTemplateFile テンプレートファイルに対して GENERIC_READ アクセス権を備えているハンドルを指定する.

戻り値　関数が成功すると，指定したファイルに対する開いているハンドルが返る．関数が失敗すると，INVALID_HANDLE_VALUE が返る．

解説　CreateFile 関数が返したオブジェクトハンドルを閉じるには，CloseHandle 関数を用いる．

2.4.12　GetCurrentDirectory

現在のプロセスのカレントディレクトリを取得する．

```
DWORD GetCurrentDirectory(
  DWORD nBufferLength,  // ディレクトリバッファのサイズ
  LPTSTR lpBuffer       // ディレクトリバッファ
);
```

パラメータ

nBufferLength カレントディレクトリを取得するためのバッファの長さを TCHAR 単位で指定する．

lpBuffer バッファへのポインタを指定する．このバッファにカレントディレクトリの絶対パス名が格納される．

戻り値　関数が成功するとバッファに書き込まれた文字数が返る．関数が失敗すると 0 が返る．

2.4.13　RtlInitUnicodeString

ユニコード文字列を初期化する．

```
VOID RtlInitUnicodeString(
    IN OUT PUNICODE_STRING  DestinationString,
    IN PCWSTR  SourceString
);
```

パラメータ

DestinationString UNICODE_STRING 型の文字列へのポインタ

SourceString NULL 終端ユニコード文字列へのポインタ

戻り値

解説　SourceString に格納された内容が DestinationString にコピーされる．コピーの際に UNICODE_STRING 型への変換を行う．

なお，名前が RtlXxx の形をしている関数は，C のランタイムライブラリと同様の機能を提供する，ドライバから利用可能な関数を表す．(通常の C のランタイムライブラリはドライバから利用できないことに注意．)

2.4.14　DriverEntry

ドライバがロードされた後はじめに呼ばれる関数で，ドライバの初期化を行う．

```
NTSTATUS
  DriverEntry(
    IN PDRIVER_OBJECT  DriverObject,
    IN PUNICODE_STRING  RegistryPath
    );
```

パラメータ

DriverObject ドライバオブジェクトへのポインタ．

RegistryPath ドライバのレジストリキーを示すユニコード文字列

戻り値　関数が成功した場合は，STATUS_SUCCESS を返さなければいけない．関数が失効した場合は，ntstatus.h で定義された error status value の一つを返さなければいけない．

解説　DDK の提供するビルドツールでは DriverEntry がドライバのエントリーポイントとみなされ，I/O システムからこの関数が呼ばれる．

2.4.15　IoCreateDevice

ドライバが使用するデバイスオブジェクトを作成する．

```
NTSTATUS
  IoCreateDevice(
    IN PDRIVER_OBJECT  DriverObject,
    IN ULONG  DeviceExtensionSize,
    IN PUNICODE_STRING  DeviceName  OPTIONAL,
    IN DEVICE_TYPE  DeviceType,
    IN ULONG  DeviceCharacteristics,
    IN BOOLEAN  Exclusive,
    OUT PDEVICE_OBJECT  *DeviceObject
    );
```

パラメータ

DriverObject ドライバオブジェクトへのポインタを指定する.(DeviceEntry 関数の第一引数.)

DeviceExtensionSize デバイス拡張のために確保の必要な領域のバイト数を指定する.

DeviceName デバイスオブジェクトの名前を指定する.

DeviceType デバイスの型を指定する. ntddk.h と wdm.h で定義された型のどれとも一致しない場合, FILE_DEVICE_UNKNOWN を指定.

DeviceCharacteristics デバイスの特性を指定する.

Exclusive システムのために予約されている. FALSE を指定する.

DeviceObject デバイスオブジェクトを格納するための変数へのポインタを指定する.

戻り値 関数が成功すると STATUS_SUCCESS が返り, 関数が失敗すると NTSTATUS エラーコードが返る.

解説 不必要になったデバイスオブジェクトは IoDeleteDevice 関数で削除する.

2.4.16 IoDeleteDevice

デバイスオブジェクトを削除する.

```
VOID
  IoDeleteDevice(
    IN PDEVICE_OBJECT  DeviceObject
    );
```

パラメータ

DeviceObject 削除するデバイスオブジェクトへのポインタを指定する.

2.4.17 IoCreateSymbolicLink

デバイスオブジェクト名とユーザから見ることのできるデバイス名との間のシンボリックリンクを設定する.

```
NTSTATUS
  IoCreateSymbolicLink(
    IN PUNICODE_STRING  SymbolicLinkName,
    IN PUNICODE_STRING  DeviceName
    );
```

パラメータ

SymbolicLinkName ユーザから見ることのできるデバイス名を指定する.

DeviceName ドライバが作成したデバイスオブジェクト名を指定する.

戻り値 関数が成功すると STATUS_SUCCESS が返る.

2.4.18 IoDeleteSymbolicLink

シンボリックリンクを削除する.

```
NTSTATUS
  IoDeleteSymbolicLink(
    IN PUNICODE_STRING  SymbolicLinkName
    );
```

パラメータ

SymbolicLinkName ユーザから見ることのできるデバイス名を指定する.

戻り値　関数が成功すると STATUS_SUCCESS が返る.

2.4.19　IoGetCurrentIrpStackLocation

与えられた IRP 中のスタック位置へのポインタを返す.

```
PIO_STACK_LOCATION
  IoGetCurrentIrpStackLocation(
    IN PIRP  Irp
    );
```

パラメータ

Irp　　IRP へのポインタを指定する.

戻り値　ドライバの I/O スタック位置へのポインタを返す.

2.4.20　IoCompleteRequest

I/O リクエストを処理し終わったことを示し，IRP を I/O Manager へ返す.

```
VOID
  IoCompleteRequest(
    IN PIRP  Irp,
    IN CCHAR  PriorityBoost
    );
```

パラメータ

Irp　　IRP へのポインタを指定する.

PriorityBoost　リクエストを出したスレッドの優先度をどれだけ増加させるかを指定する．ドライバが処理をすばやく終わらせることが可能な場合，IO_NO_INCREMENT を指定.

2.4.21　OpenSCManager

指定されたコンピュータ上のサービス制御マネージャ（Service Control Manager：SCM）との接続を確立し，その SCM の指定されたデータベースを開く.

```
SC_HANDLE OpenSCManager(
  LPCTSTR lpMachineName,   // コンピュータ名
  LPCTSTR lpDatabaseName,  // SCM データベースの名前
  DWORD dwDesiredAccess    // アクセスのタイプ
);
```

パラメータ

lpMachineName 接続先コンピュータの名前を指定する．ローカルコンピュータ上の SCM に接続する場合，NULL か空文字列を指定．

lpDatabaseName SCM の開くデータベース名を指定する．

dwDesiredAccess サービス制御マネージャに割り当てるアクセス権を指定する．

戻り値 関数が成功すると，指定されたデータベースのハンドルが返る．関数が失敗すると、NULL が返る．

解説 CreateService 関数や OpenService 関数で，この関数の戻り値のハンドルを使用する．また，CloseServiceHandle 関数を呼び出すと，ハンドルを閉じることができる．

2.4.22 CreateService

サービスオブジェクトを作成し，SCM の指定されたデータベースに追加する．

```
SC_HANDLE CreateService(
  SC_HANDLE hSCManager,        // SCM データベースのハンドル
  LPCTSTR lpServiceName,       // 開始したいサービスの名前
  LPCTSTR lpDisplayName,       // 表示名
  DWORD dwDesiredAccess,       // サービスのアクセス権のタイプ
  DWORD dwServiceType,         // サービスのタイプ
  DWORD dwStartType,           // サービスを開始する時期
  DWORD dwErrorControl,        // サービスに失敗したときの深刻さ
  LPCTSTR lpBinaryPathName,    // バイナリファイル名
  LPCTSTR lpLoadOrderGroup,    // ロード順序を決定するグループ名
  LPDWORD lpdwTagId,           // タグ識別子
  LPCTSTR lpDependencies,      // 複数の依存名からなる配列
  LPCTSTR lpServiceStartName, // アカウント名
  LPCTSTR lpPassword           // アカウントのパスワード
);
```

パラメータ

hSCManager SCM のデータベースのハンドル（OpenSCManager 関数が返したハンドル）を指定する．

lpServiceName インストールするサービスを指定する．

lpDisplayName ユーザーインターフェイスプログラムがサービスを識別するために使う表示名を指定する．

dwDesiredAccess サービスに割り当てるアクセス権を指定する．

dwServiceType サービスタイプを指定する．例えば，ドライバサービスの場合は，SERVICE_KERNEL_DRIVER を指定．

dwStartType サービスを開始する時期を指定する．例えば，プロセスが StartService 関数を呼び出したときに SCM が開始するサービスを指定する場合，ERVICE_DEMAND_START を指定．

dwErrorControl 起動時にサービスを開始することに失敗した場合のエラーの深刻度を指定する．例えば，エラーをログに記録し，メッセージボックスをポップアップ表示するが，開始操作を続行する場合は，SERVICE_ERROR_NORMAL を指定．

lpBinaryPathName サービスバイナリファイルの完全修飾パスを指定する．

lpLoadOrderGroup このサービスが所属しているロード順序決定グループの名前を指定する．

lpdwTagId タグ値を受け取る変数へのポインタを指定する．

lpDependencies サービスを開始する前に開始しておかなければいけないサービスとグループを指定する．

lpServiceStartName サービスタイプが SERVICE_KERNEL_DRIVER である場合，システムがデバイスドライバをロードするために使うドライバオブジェクト名がドライバ名になる．I/O システムが作成した既定のオブジェクト名をドライバ名として使うことを予定している場合は，NULL を指定する．

lpPassword lpServiceStartName パラメータで指定されたアカウントに対するパスワードを指定する．

戻り値 関数が成功するとサービスのハンドルが返り，失敗すると NULL が返る．

解説 CloseServiceHandle 関数を呼び出すと，この関数の戻り値のハンドルを閉じることができる．

2.4.23 OpenService

既存のサービスのハンドルを開く．

```
SC_HANDLE OpenService(
  SC_HANDLE hSCManager,  // SCM データベースのハンドル
  LPCTSTR lpServiceName, // サービス名
  DWORD dwDesiredAccess  // アクセス権
);
```

パラメータ

hSCManager SCM のデータベースのハンドル．OpenSCManager 関数が返したハンドルを指定する．

lpServiceName 開くサービスの名前を指定する．

dwDesiredAccess サービスに割り当てるアクセス権を指定する．

戻り値　関数が成功するとサービスのハンドルが返る．関数が失敗すると NULL が返る．

解説　CloseServiceHandle 関数を呼び出すと，この関数の戻り値のハンドルを閉じることができる．

2.4.24　StartService

サービスを開始する．

```
BOOL StartService(
  SC_HANDLE hService,             // サービスのハンドル
  DWORD dwNumServiceArgs,         // 引数の数
  LPCTSTR *lpServiceArgVectors    // 複数の引数からなる 1 つの配列
);
```

パラメータ

hService　サービスのハンドルを指定する．OpenService 関数または CreateService 関数が返したハンドルを使う．

dwNumServiceArgs　lpServiceArgVectors パラメータが指す配列内の引数文字列の数を指定する．

lpServiceArgVectors　引数文字列へのポインタからなる配列．

戻り値　関数が成功すると 0 以外の値が返り，関数が失敗すると 0 が返る．

2.4.25　ControlService

制御コードをサービスアプリケーションへ送信する．

```
BOOL ControlService(
  SC_HANDLE hService,                // サービスのハンドル
  DWORD dwControl,                   // 制御コード
  LPSERVICE_STATUS lpServiceStatus   // ステータス情報
);
```

パラメータ

hService サービスのハンドルを指定する．OpenService 関数または CreateService 関数が返したハンドルを使う．

dwControl 要求する制御コードを指定する．サービスの停止を要求する場合には，SERVICE_CONTROL_STOP を指定．

lpServiceStatus 最新のサービスステータス情報を受け取る．SERVICE_STATUS 構造体へのポインタを指定する．

戻り値 関数が成功すると 0 以外の値が返り，関数が失敗すると 0 が返る．

2.4.26 CloseServiceHandle

OpenSCManager 関数が返した SCM オブジェクトのハンドル，または，OpenServiced 関数・CreateService 関数が返したサービスオブジェクトのハンドルを閉じる．

```
BOOL CloseServiceHandle(
  SC_HANDLE hSCObject   // サービスまたは
                        // SCM オブジェクトのハンドル
);
```

パラメータ

hSCObject 閉じる SCM オブジェクトのハンドル，またはサービスオブジェクトのハンドルを指定する．

戻り値 関数が成功すると 0 以外の値が返り，関数が失敗すると 0 が返る．

解説 CloseServiceHandle 関数は，ハンドルによって参照されている SCM のオブジェクトを破棄しない．サービスオブジェクトを破棄するためには，DeleteService 関数を呼び出す．

2.4.27 DeleteService

指定されたサービスに，SCM のデータベースから削除するためのマークを付ける．

```
BOOL DeleteService(
  SC_HANDLE hService   // サービスのハンドル
);
```

パラメータ

hService サービスのハンドルを指定する．OpenService 関数または CreateService 関数が返したハンドルを指定する．

戻り値 関数が成功すると 0 以外の値が返り，関数が失敗すると 0 が返る．

解説 削除のためのマークの付けられたサービスは，CloseServiceHandle 関数によってこのサービスの開いている全てのハンドルが閉じられるとデータベースから削除される．

2.4.28 _cdecl

関数・変数の修飾子で，C と C++用のコーリングコンベンション．

2.4.29 alloc_text

指定した関数定義がおかれるコードセクションを指定する．

```
#pragma alloc_text( "textsection", function1, ... )
```

alloc_text { init，func } と指定された関数 func は，ドライバのロードが終了後メモリから解放される．これは，メモリを節約するために用いられる．

alloc_text { page，func } と指定された関数 func は，ページングが可能なメモリ領域に格納される．これも，ページングが不可能なメモリ領域を節約するために用いられる．詳しくは参考文献を参照されたい．

2.4.30 PAGED_CODE

この関数を呼ぶスレッドが，ページングを許可する IRQL で走っていることを保証する．

```
VOID   PAGED_CODE();
```

詳しくは参考文献を参照.

3 TrivialDriver2（TrivialDriver の拡張）

3.1 TrivialDriver からの変更点について

TrivialDriver2 は，TrivialDriver に読み込みと I/O 制御の機能が付加されたドライバである．このドライバはバッファを持ち，レジストリコールバックによって得られた情報をそのバッファに更新する．

TrivialDriver2 に読み込みを行うと，バッファに格納されたデータを読むことができる．また，I/O 制御により，このバッファに格納されているデータのバイト長を取得することができる．

このドライバを使うアプリケーション (`trivialapp.c`) は定期的に以下の動作を行う．

1. デバイスドライバへ制御コードを送信し，データのバイト数を知る．
2. バイト数分だけデバイスドライバのバッファからデータを読み出す．
3. 得られたデータを端末に表示する．

3.2 ビルド・実行

2.2 節で述べられている手順と同様にビルド・実行すればよい．
例えば，実行した結果以下のような出力がでれば成功．

```
> trivialapp.exe
Be sure driver (and service) are unloaded/removed.
Install service and driver.
Open newly installed driver.
......................Bytes in kernel buffer:    73
......................Bytes read:                73
LastAttackTime
LastAttackIP
LastAttackPort
LastAttackType
LastAttackDesc
......................Bytes in kernel buffer:   125
......................Bytes read:               125
HRZR_EHACNGU
HRZR_EHACNGU:P:\JVAQBJF\flfgrz32\abgrcnq.rkr
HRZR_EHACVQY
Seed
ProgramCount
```

注意することとして，Windows XP 上でコンパイルを行う場合,「RegNtPreCreateKeyEx が定義されていない」などといってエラーがでてコンパイルに失敗してしまう．これは，RegNtPreCreateKeyEx などの実行後に通知を受けとるための機構を Windows XP が持たないからである．そのため，コンパイルするためにはソースコードを以下のように変更する必要がある．

RegNtPreCreateKeyEx	$\Longrightarrow$	RegNtPreCreateKey
RegNtPreDeleteKey	$\Longrightarrow$	RegNtDeleteKey
RegNtPreSetValueKey	$\Longrightarrow$	RegNtSetValueKey
RegNtPreDeleteValueKey	$\Longrightarrow$	RegNtDeleteValueKey
PREG_CREATE_KEY_INFORMATION	$\Longrightarrow$	PREG_PRE_CREATE_KEY_INFORMATION

3.3 ソースコードの概要

3.3.1 trivial.c

前節と異なり，

- レジストリコールバック機構により，アプリケーションのレジストリへのアクセスを捕捉する．デバイスドライバは，オープンされた時に CmRegisterCallback 関数でコールバックを追加する.（このコールバックはクリーンアップ時に削除される）．TrivialRegistryCallback 関数が呼ばれるように設定している．
- デバイスドライバはバッファ (TraceBuffer) をもつ．このバッファにレジスタコールバックによって得られた情報が格納される．
- デバイスドライバは I/O 制御と読み込みを実装している．IOCTL_TRIVIAL_GETLENGTH という制御コードが送られてくると，バッファに格納されたデータのバイト長を返す（TrivialDeviceControl 関数）．バッファへの読み込みが起こると TrivialRead 関数が呼ばれ，バッファに格納されたデータを返す．

また，バッファへのアクセスはミューテックスを用いて同期がとられている．このミューテックスは，各ドライバに依存した情報を保持しておくための領域（デバイス拡張と呼ばれる）におかれる．

3.3.2 trivialapp.c

基本的には前節の TrivialDriver と同様である．ただ，TrivialDriver では開いたデバイスドライバが何もせずに閉じられているが，ここでは，デバイスドライバから読み込みを行っている．具体的には，

1. DeviceIoControl 関数で IOCTL_TRIVIAL_GETLENGTH という制御コードを送信することにより，バッファに格納されたデータのバイト長を取得する．

2. ReadFile 関数を呼び，デバイスドライバからバイト長分だけデータを読み込み，その結果を表示する．

という動作を繰り返している．

3.4 ソースコード中で使用される Windows API

3.4.1 ミューテックス

相互排他制御に用いる．ミューテックスはシグナル状態と非シグナル状態をとり，その状態は以下のように変化する．

- KeInitializeMutex 関数で初期化されるとシグナル状態となる．
- KeWaitForSingleObject 関数が呼ばれると非シグナル状態になる．この関数は，基本的には，指定されたオブジェクトがシグナル状態になるまでブロックする．
- KeReleaseMutex 関数が呼ばれるとミューテックスはシグナル状態になる．

詳しくは参考文献を参照.

3.4.2 LARGE_INTEGER

64 ビット符合付き整数.

3.4.3 REG_NOTIFY_CLASS

RegistryCallback 関数に渡される，レジストリ操作の型を示す．例を挙げると次図のようになる．

レジストリ操作の型	説明
RegNtPreCreateKey	キーが生成される直前の通知
RegNtDeleteKey	キーが削除される直前の通知
RegNtDeleteValueKey	キーの value entry が削除される直前の通知

3.4.4 PREG_SET_VALUE_KEY_INFORMATION

レジストリキーの value entry の新規設定を表す.

```
typedef struct _REG_SET_VALUE_KEY_INFORMATION {
  PVOID  Object;
  PUNICODE_STRING  ValueName;
  ULONG  TitleIndex;
  ULONG  Type;
  PVOID  Data;
  ULONG  DataSize;
} REG_SET_VALUE_KEY_INFORMATION,
  *PREG_SET_VALUE_KEY_INFORMATION;
```

レジストリ操作が RegNtDeleteValueKey であるとき，RegistryCallback 関数の第二引数は PREG_SET_VALUE_KEY_INFORMATION 型となる．

メンバ（の一部）

Object　value entry が変更されるキーのオブジェクトへのポインタ．

ValueName value entry 名（ユニコード）へのポインタ．

TitleIndex システムに予約されている．

Type　書き込まれるデータの型．

Data　書き込まれるデータを含むバッファへのポインタ．

DataSize バッファのバイト長．

3.4.5 ReadFile

ファイルからデータを読み取る．

```
BOOL ReadFile(
  HANDLE hFile,                // ファイルのハンドル
  LPVOID lpBuffer,             // データバッファ
  DWORD nNumberOfBytesToRead,  // 読み取り対象のバイト数
  LPDWORD lpNumberOfBytesRead, // 読み取ったバイト数
  LPOVERLAPPED lpOverlapped    // オーバーラップ構造体のバッファ
);
```

パラメータ

hFile　読み取り対象のファイルのハンドルを指定する．

lpBuffer ファイルから読みとったデータを保存するためのバッファへのポインタを指定する．

nNumberOfBytesToRead 読み取りバイト数を指定する．

lpNumberOfBytesRead 読み取ったバイト数が格納するための変数へのポインタを指定する．

lpOverlapped OVERLAPPED 構造体へのポインタを指定する．hFile パラメータが FILE_FLAG_OVERLAPPED フラグを指定せずに開かれたハンドルを指していて，lpOverlapped パラメータが NULL の場合，ファイルポインタの現在の位置からファイルの同期読み取りが開始され，ReadFile は読み取りが完了すると制御を返す．

戻り値 関数が成功すると 0 以外の値が返り，関数が失敗すると 0 が返る．

3.4.6 RtlCopyMemory

指定されたバッファの内容をコピーする．

```
VOID
  RtlCopyMemory(
    IN VOID UNALIGNED  *Destination,
    IN CONST VOID UNALIGNED  *Source,
    IN SIZE_T  Length
    );
```

パラメータ

Destination コピー先のバッファへのポインタを指定する．

Source コピー元のバッファへのポインタを指定する．

Length コピーするバイト長を指定する．

解説 Source と Destination の領域は重なってはならない．

3.4.7 KeInitializeMutex

ミューテックスオブジェクトを初期化し，シグナル状態にする．

```
VOID
  KeInitializeMutex(
    IN PRKMUTEX  Mutex,
    IN ULONG  Level
    );
```

パラメータ

Mutex　ミューテックスオブジェクトへのポインタを指定する.

Level　システムによって予約されている. 0 を指定する.

3.4.8 KeReleaseMutex

指定されたミューテックスオブジェクトを解放する.

```
LONG
  KeReleaseMutex(
    IN PRKMUTEX  Mutex,
    IN BOOLEAN  Wait
    );
```

パラメータ

Mutex　ミューテックスオブジェクトへのポインタを指定する.

Wait　この関数の直後に KeWaitXxx 関数が呼ばれる場合 TRUE を指定. そうでない場合 FALSE を指定.

戻り値　ミューテックスオブジェクトが正しく解放され，かつシグナル状態になった場合，0 を返す.

3.4.9 KeWaitForSingleObject

指定されたオブジェクトがシグナル状態になるか，指定された時間が経過するまで待つ.

```
NTSTATUS
  KeWaitForSingleObject(
    IN PVOID  Object,
    IN KWAIT_REASON  WaitReason,
    IN KPROCESSOR_MODE  WaitMode,
    IN BOOLEAN  Alertable,
    IN PLARGE_INTEGER  Timeout  OPTIONAL
    );
```

パラメータ

Object　オブジェクト（ミューテックス・セマフォ）へのポインタ.

WaitReason 待機をする理由を指定する．ユーザスレッドとして走らない限りは Executive を指定する.

WaitMode カーネルモードで待つ場合は KernelMode を指定し，ユーザモードで待つ場合は UserMode を指定する.

Alertable アラートを許可するかどうかを指定する.

Timeout タイムアウト値へのポインタ．永久に待ち続ける場合 NULL を指定.

戻り値　以下のうちのどれかを返す.

STATUS_SUCCESS 関数が成功した場合.

STATUS_ALERTED アラートによってこの関数が終了した場合.

STATUS_USER_APC タイムアウト前にユーザ APC が送られてきた場合.

STATUS_TIMEOUT タイムアウトとなった場合.

3.4.10　CmRegisterCallback

レジストリコールバックルーチンを登録する.

```
NTSTATUS
  CmRegisterCallback(
    IN PEX_CALLBACK_FUNCTION  Function,
    IN PVOID  Context,
    OUT PLARGE_INTEGER  Cookie
    );
```

パラメータ

Function 登録するレジストリコールバックルーチンを指定する.

Context CallbackContext パラメタとして渡す値を指定する.

Cookie CmUnregisterCallback 関数に渡す，コールバックを識別するための LARGE_INTEGER 変数へのポインタ.

戻り値 関数が成功すると STATUS_SUCCESS が返り，関数が失敗すると NTSTATUS エラーコードが返る.

3.4.11 CmUnRegisterCallback

レジストリコールバックルーチンを削除する.

```
NTSTATUS
  CmUnRegisterCallback(
    IN LARGE_INTEGER  Cookie
    );
```

パラメータ

Cookie CmRegisterCallback 関数で取得した，コールバックを識別するための LARGE_INTEGER 値.

戻り値 関数が成功すると STATUS_SUCCESS が返り，関数が失敗すると NTSTATUS エラーコードが返る．Cookie に一致するコールバックルーチンが存在しない場合，STATUS_INVALID_PARAMETER が返る.

解説

3.4.12 DeviceIoControl

指定されたデバイスドライバへ制御コードを直接送信し，デバイスに対応する動作をさせる.

```
BOOL DeviceIoControl（
  HANDLE hDevice，// デバイス、ファイル、ディレクトリ
                  // いずれかのハンドル
  DWORD dwIoControlCode，// 実行する動作の制御コード
  LPVOID lpInBuffer，// 入力データを供給するバッファへのポインタ
  DWORD nInBufferSize，// 入力バッファのバイト単位のサイズ
  LPVOID lpOutBuffer，// 出力データを受け取るバッファへのポインタ
  DWORD nOutBufferSize，// 出力バッファのバイト単位のサイズ
  LPDWORD lpBytesReturned，// バイト数を受け取る変数へのポインタ
  LPOVERLAPPED lpOverlapped // 非同期動作を表す構造体へのポインタ
）;
```

パラメータ

hDevice デバイスのハンドルを指定する．デバイスハンドルを取得するには CreateFile 関数を呼び出す．

dwIoControlCode 制御コードを指定する．

lpInBuffer 入力データ用のバッファへのポインタを指定する．

nInBufferSize lpInBuffer が指すバッファのバイト単位のサイズを指定する．

lpOutBuffer 動作の出力データを受け取るバッファへのポインタを指定する．

nOutBufferSize lpOutBuffer が指すバッファのバイト単位のサイズを指定する．

lpBytesReturned pOutBuffer が指すバッファへ格納されるデータのバイト単位のサイズを受け取る変数へのポインタを指定する．

lpOverlapped 1 つの OVERLAPPED 構造体へのポインタを指定する．FILE_FLAG_OVERLAPPED フラグをセットせずに hDevice で指定したデバイスを開いた場合，lpOverlapped は無視される．

戻り値　関数が成功すると 0 以外の値が返り，関数が失敗すると 0 が返る．

4　プロセス起動・終了の捕捉

TrivialDriver2 のソースを変更し，レジストリコールバックルーチンをプロセスの生成･削除時に呼ばれるコールバックルーチンに変更する．PsSetCreateProcessNotifyRoutine 関数を用いて，これを実現する．

4.1　ビルド時の注意

wdm.h 中では，PsSetCreateProcessNotifyRoutine 関数と PCREATE_PROCESS_NOTIFY_ROUTINE 型[2]が定義・宣言されていない．ddk.h 中では宣言されているが，wdm.h 中で

```
#define _NTDDK_
```

とあるためこの宣言は無効となってしまっている．そこで，PsSetCreateProcessNotifyRoutine 関数を使うためには，以下のような型定義と関数宣言をソースコードに付け加えておく必要がある．

```
typedef
VOID
(*PCREATE_PROCESS_NOTIFY_ROUTINE)(
    IN HANDLE ParentId,
    IN HANDLE ProcessId,
    IN BOOLEAN Create
    );

NTSTATUS
PsSetCreateProcessNotifyRoutine(
    IN PCREATE_PROCESS_NOTIFY_ROUTINE NotifyRoutine,
    IN BOOLEAN Remove
    );
```

4.2　ソースコード中で使用される Windows API

プロセスの生成･削除時に呼ばれるコールバックルーチンを追加･削除する．

[2] PsSetCreateProcessNotifyRoutine 関数の引数の型

```
NTSTATUS
  PsSetCreateProcessNotifyRoutine(
    IN PCREATE_PROCESS_NOTIFY_ROUTINE  NotifyRoutine,
    IN BOOLEAN  Remove
    );
```

パラメータ

NotifyRoutine コールバックルーチンを指定する.

Remove ルーチンを追加する場合は FALSE を，削除する場合は TRUE を指定する.

戻り値 関数が成功すると STATUS_SUCCESS が返る．既にコールバックが登録済であるなどして関数が失敗すると，STATUS_INVALID_PARAMETER が返る.

解説 第一引数に渡す関数の型は以下のようになっている.

```
VOID
(*PCREATE_PROCESS_NOTIFY_ROUTINE) (
    IN HANDLE  ParentId,
    IN HANDLE  ProcessId,
    IN BOOLEAN  Create
    );
```

パラメータ

ParentId 親プロセスを示す.

ProcessId 子プロセスを示す.

Create プロセスが生成された場合は TRUE を，削除された場合は FALSE となる.

5 PSAPIを用いた拡張

MSDN ライブラリの「WIndows ベースサービス」→「パフォーマンスモニタ」→「SDK ドキュメント」→「パフォーマンスモニタ」→「PSAPI」に PSAPI についての情報があるので，それを参考にすること.

目 次