UT-VPN

Developer's Guide

このドキュメントは、ソフトウェアの開発の知識のある方が UT-VPN ソフトウェアのソースコードを研究したり、ソースコードの一部を改造して自分専用の VPN ソフトウェアを開発したり、それを第三者に対して配布したりするための操作方法や基礎知識について解説しています。

このドキュメントは、現在日本語でのみ記述されています。後日、UT-VPN の英語版をリリースする際には、このドキュメントも英語に翻訳される予定です。このドキュメントをお読みいただくことにより、UT-VPN のソフトウェアをビルドし実行する方法、インストーラ等のパッケージファイルを作成する方法、および自己の好みに応じて好きなように改造する方法を理解いただくことができます。

1. UT-VPN のソフトウェアの構成

実行可能ファイルの一覧

UT-VPN には以下の 5 つのソフトウェアが含まれています。誰でも、UT-VPN のソースコードを改変して、自分専用の以下のそれぞれの実行可能ファイルをビルドすることができます。

ソフトウェア名	実行ファイル名	注意点
UT-VPN Server	【Windows 環境】 utvpnserver.exe (32-bit) utvpnserver_x64.exe (64-bit) 【UNIX 環境】 utvpnserver	
UT-VPN Client	【Windows 環境】 utvpnclient.exe (32-bit) utvpnclient_x64.exe (64-bit) 【UNIX 環境】 utvpnclient	
UT-VPN コマンドライン管理ユーティリティ (utvpncmd)	【Windows 環境】 utvpncmd.exe (32-bit) utvpncmd_x64.exe (64-bit) 【UNIX 環境】 utvpncmd	
UT-VPN Server 管理マネージャ	【Windows 環境】 utvpnsmgr.exe (32-bit) utvpnsmgr_x64.exe (64-bit)	この GUI ツールは、Windows 版のみ提供されています。
UT-VPN Client 接続マネージャ	【Windows 環境】 utvpncmgr.exe (32-bit) utvpncmgr_x64.exe (64-bit)	この GUI ツールは、Windows 版のみ提供されています。

hamcore.utvpn ファイル

hamcore.utvpn ファイルは UT-VPN の上記 5 個のプログラムから共通で参照されるデータファイルです。hamcore.utvpn は独自の ZIP ファイルとよく似た形式のアーカイブファイルで、中に複数のファイルを含んでいます。hamcore.utvpn ファイルは実行可能ファイルと同一のパスに設置しなければなりません。

hamcore.utvpn ファイルの生成 (改変) 方法については後述します。

2. UT-VPN のソースコード

UT-VPN のソースコードは、Web サイト http://utvpn.tsukuba-ac.jp/ において GPLv2 ライセンスで配布されており、誰でもダウンロード可能です。ソースコードは、「Windows 用ソースコード」と「UNIX 用ソースコード」の 2 種類が配布されています。

Windows 用ソースコードについて

Windows 用ソースコードを用いると、Windows 用のバイナリを生成することができます。Windows 用ソースコードのビルドには、Windows XP 以降の Windows と Visual Studio 2008 が必要です。これらの環境があれば Windows 用ソースコードのビルドは非常に簡単で、極端に言えば、BuildAll.cmd というバッチファイルをダブルクリックするだけでビルドすることができます。

Windows 用ソースコードには BuildUtil プログラム (C#で記述されています) が含まれています。BuildUtil は、Visual Studio 2008 を内部的に呼び出して UT-VPN をビルドし、すべての実行可能ファイルおよび hamcore.utvpn ファイルを動的に生成します。そして、これらのファイルをもとに、Windows Installer XML Toolkit (WiX) という Windows Installer の.msi ファイルを作成するためのツール (このツールは Common Public License で Microsoft 社から供給されており、UT-VPN のソースコード中に含まれています。なお、このツールは GPLv2 ライセンスで供給されている訳ではないことにご注意ください。) を呼び出し、.msi ファイル (Windows Installer ファイル) を自動的にビルドします。

BuildUtil プログラムは最終的に C:\TMP\UTVPN_TMP にリリースファイルを出力します (C: の部分はお使いの Windows のシステムドライブです。たとえば Z: ドライブがシステムドライブの場合は Z:\TMP\UTVPN_TMP に出力されます)。もし Windows Vista 以降をお使いで BuildUtil を、UAC の下で、一般ユーザー権限で起動する場合は、予め C:\TMP ディレクトリを一般ユーザーでも書き込み可能なように手動で作成して ACL を適切に設定しておいてください。

BuildUtil プログラムを使用すると、このように、ソースコードからインストーラを自動生成することができます。しかし、BuildUtil プログラムを使用する場合は Visual Studio 2008 を用いた UT-VPN のデバッグを行うことはできません。UT-VPN のデバッグ (つまり、コードを少しずつ改良しながら動作確認をする作業) を行うためには、Visual Studio 2008 を起動し、UTVPN.sln ソリューションファイルを開いてください。そして、他の多くの Visual Studio 2008 用ソースコードをビルドするのと同じ手順で、単純にビルドを行ってください。プラットフォームとして Win32 と x64 が選択できます。選択したほうの CPU に適合した EXE ファイルが bin ディレクトリに出力されます。

Windows 用ソースコードから UNIX 用ソースコードの tar.gz ファイルを生成する

Windows 用ソースコードは UNIX 用ソースコードの源泉となるべきソースコードです。Windows 用ソースコードに含まれる BuildUtil プログラムにより、Windows 用ソースコードをもとにして、UNIX 用ソースコードのファイル集が自動生成されます。

この際に、すべてのソースコード (.c ファイルや.h ファイル等) は BOM 付き UTF-8 ファイルから BOM 無し UTF-8 ファイルに自動変換されます。なぜこのような作業が必要になるかというと、Visual Studio 2008 のコンパイラやエディタは BOM 付き UTF-8 ファイルが必要であり、逆に、UNIX のコンパイラである gcc は BOM 無し UTF-8 ファイルが必要だからです。UT-VPN のソースコードは日本人

が開発したので、ソースコード中のコメントには日本語文字列が多数含まれています。この部分が仮に正しく UTF-8 形式として認識されず、オペレーティングシステムの標準の文字コードとして認識されてしまうと、おかしな警告メッセージがたくさん出たり、コンパイルエラーの原因になったりします。そこで、UT-VPN の開発者は、標準的な開発環境として Windows および Visual Studio 2008 を採用し、そこで編集・開発したコードをもとに UNIX 版のソースコードファイル集を自動生成するという仕組みを構築しました。

したがって、お勧めの開発方法としては、Windows 上で Visual Studio 2008 を用いて UT-VPN を改造・拡張し、Windows 上である程度の動作チェックを行ってから、BuildUtil を用いて UNIX 版のソースコードを自動生成し、その UNIX 版のソースコードを UNIX コンピュータに転送（または Windows 上のディスクを CIFS や NFS で UNIX コンピュータからマウント）して Makefile を用いて make するという方法が推奨されます。

しかし、開発者は、上記の少し変わった開発方法に従う義務はありません。たとえば、UNIX 用ソースコードを直接 emacs 等でいつものように編集・改造してそのまま make することもできます。いずれの方法でもソースコードを編集できます。

UNIX 用ソースコードについて

UNIX 用ソースコードは前述のように Windows 用ソースコードの BuildUtil から自動生成されます (Makefile も自動生成されます)。そしてそれらのファイルはすべて tar.gz 形式にパッケージされます。

UNIX 用のソースコードをビルドする場合は、当該 tar.gz ファイルを、tar を用いて展開し、そのディレクトリ上でまず configure を実行します。すると、OS や CPU の種類を選択するよう求めるプロンプトが表示されますので、指示に従って選択してください。

一般的な UNIX 用のプログラムは、autoconf というソフトウェアを用いて configure を生成します。しかし、UT-VPN では、時間がなかったので、autoconf は使わず、代わりに OS ごとの相違点を記述した OS ごとの Makefile を自動生成するようにしています。OS ごとの Makefile は OS ごと、CPU ごとに存在し、makefiles というサブディレクトリに格納されています。configure の実行することは、単に makefiles ディレクトリから目的の OS や CPU に適合した Makefile を取得してカレントディレクトリにコピーする処理だけです。

Makefile が configure と同一のディレクトリにコピーされたら、普通のソフトウェアをビルドするときと同様に、make コマンドでプログラムをビルドすることができます。

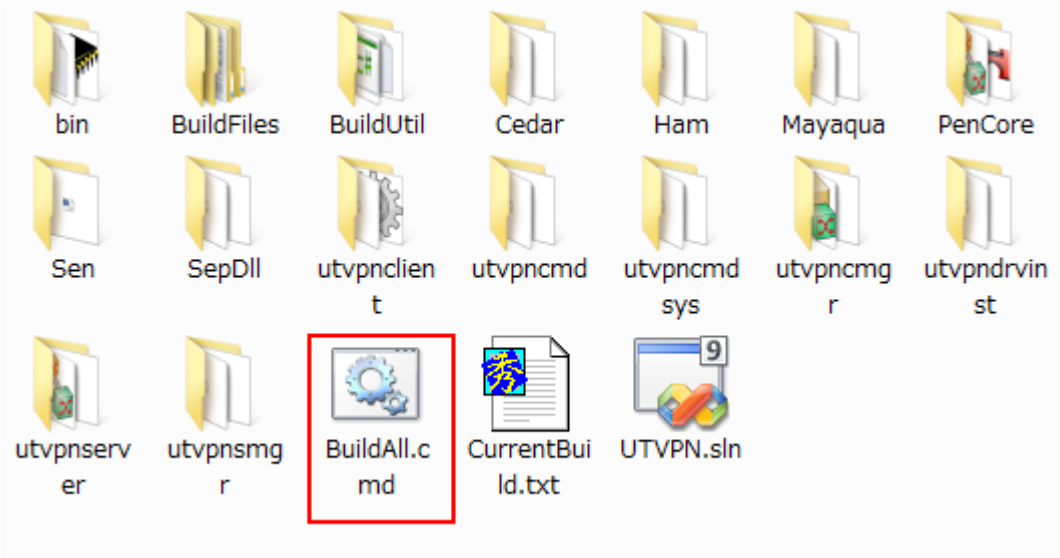
3. Windows 用ソースコードのビルド、デバッグ、インストーラ作成

Windows 用ソースコードのダウンロードと展開

まず、任意のディレクトリに Windows 用ソースコードをダウンロードします。Windows 用ソースコードのファイル名は、たとえば「utvpn-src-win32-v100-7093-beta-2010.06.26.zip」のように「utvpn-src-win32」で始まっています。この zip ファイルを展開すると、「Main」というディレクトリの中に大量のファイルがあることが分かります。

とりあえずフルビルドをしてみる

まずは試しにソースコードをフルビルドし、インストーラを生成してみましょう。フルビルドを行うには、「BuildAll.cmd」を起動してください。



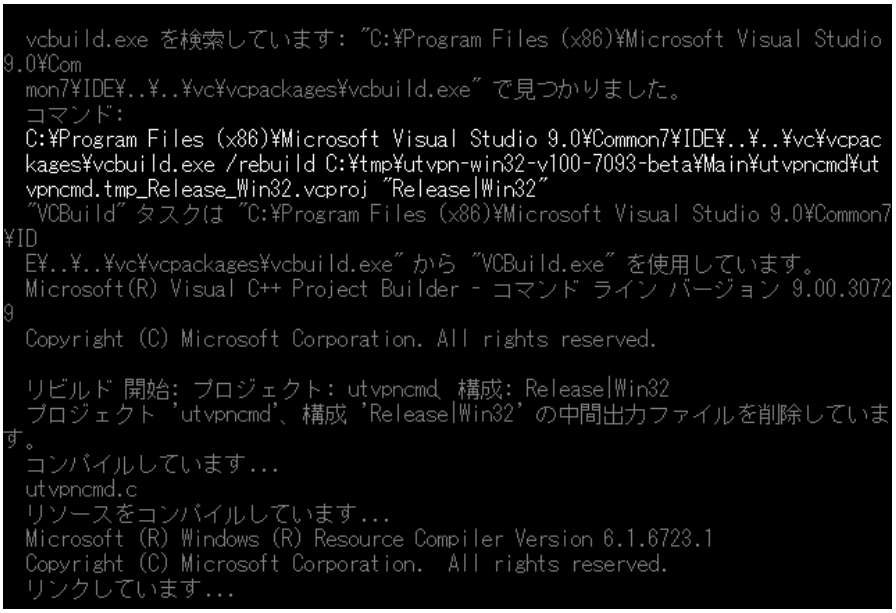
BuildAll.cmd を起動すると、まず、内部的に BuildUtil をビルドします。次に、以下のようなメッセージが表示されます。

```
BuildUtil コマンド - UT-VPN Build Utility

Copyright (C) 2004-2010 SoftEther Corporation.
Copyright (C) 2004-2010 University of Tsukuba, Japan.
Copyright (C) 2003-2010 Daiyuu Nobori.
All Rights Reserved.

All コマンド - Builds all sources and releases all packages.
Increments build number (y/n) ?
```

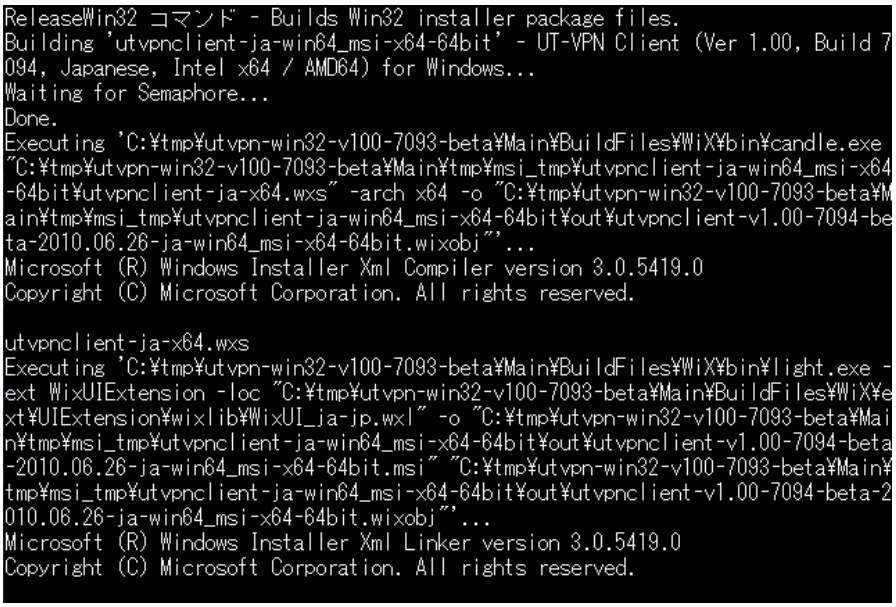
この「Increments build number?」とは、「ソースコードのビルド番号をインクリメントしますか?」という意味です。ここで「y」を押すと、現在のソースコードのビルド番号に 1 が加算されます。「n」を押すとビルド番号は変更されません。

次に「Normalizes source codes?」と表示されます。これは「ソースコード中のビルド番号を統一しますか?」という意味です。ここで「y」を押すと、UT-VPN のいくつかのソースコード中のビルド番号やビルド日時を記載した部分がすべて新しいビルド番号および本日のビルド時刻で上書きされます。たとえば、Windows のリソースファイル (.rc ファイル) 中にはビルド番号を記載する欄がありますが、この欄のビルド番号が上書きされます。

次に、しばらくの間、Visual Studio 2008 のコンパイラが内部的に起動され、コンパイル処理が行われます。コンピュータの処理速度によって所要時間は変動しますが、数分でコンパイルやリンクは終わります。



さらに放っておくと、Windows Installer XML Toolkit (WiX) が起動し、Windows Installer のビルドを開始します。



最後に、UNIX 用ソースコードへの変換処理が行われ、すべての処理が完了すると、「C:\tmp\UTVPN_TMP\20100626-v1.00-7094-beta-2010.06.26-PC25-username に出力しました。」というようなメッセージが表示されます。

このメッセージが表示された後に、「C:\tmp\UTVPN_TMP」を開き、リリース名のディレクトリの下に「Public」というディレクトリができており、その中のサブフォルダ内に以下のように Windows 用のインストーラ (.msi ファイルや.zip ファイル) や UNIX 用のソースコードの tar.gz ファイルが生成されていることを確認してください。

そして、試しにいずれかの.msi ファイルをダブルクリックして起動すると、UT-VPN のインストーラが起動することが確認できるはずです。

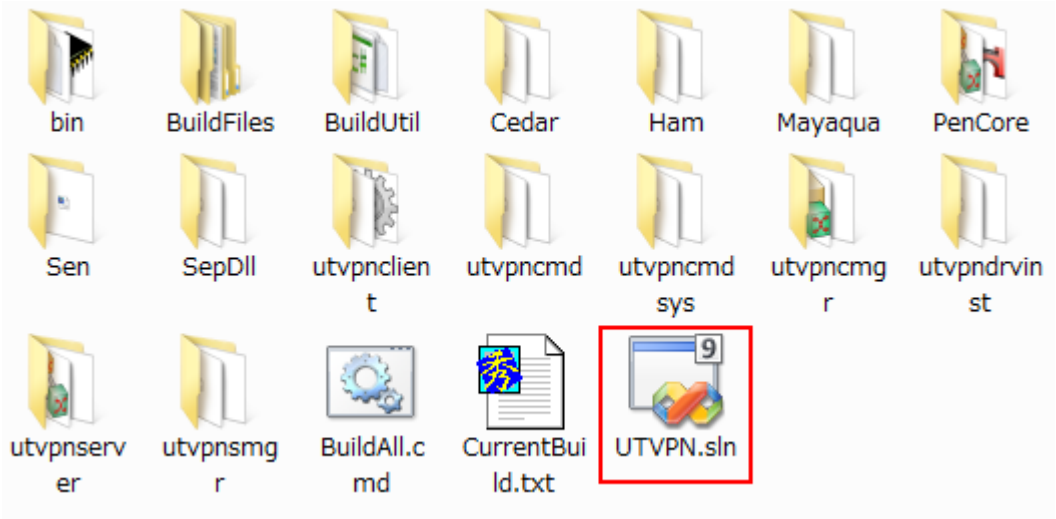
UT-VPN のソースコードはすべて GPLv2 形式でライセンスされていますので、GPLv2 に従い、誰でも、改造したソースコードだけではなく、改造後のインストーラやバイナリパッケージを第三者に配布することができます（バイナリを配布する場合は、ソースコードも配布する義務が生じます)。

また、ここで生成された tar.gz ファイルは UNIX 用のソースコード集です。これを UNIX マシンにコピーして make することで、UNIX 版をビルドすることもできます。

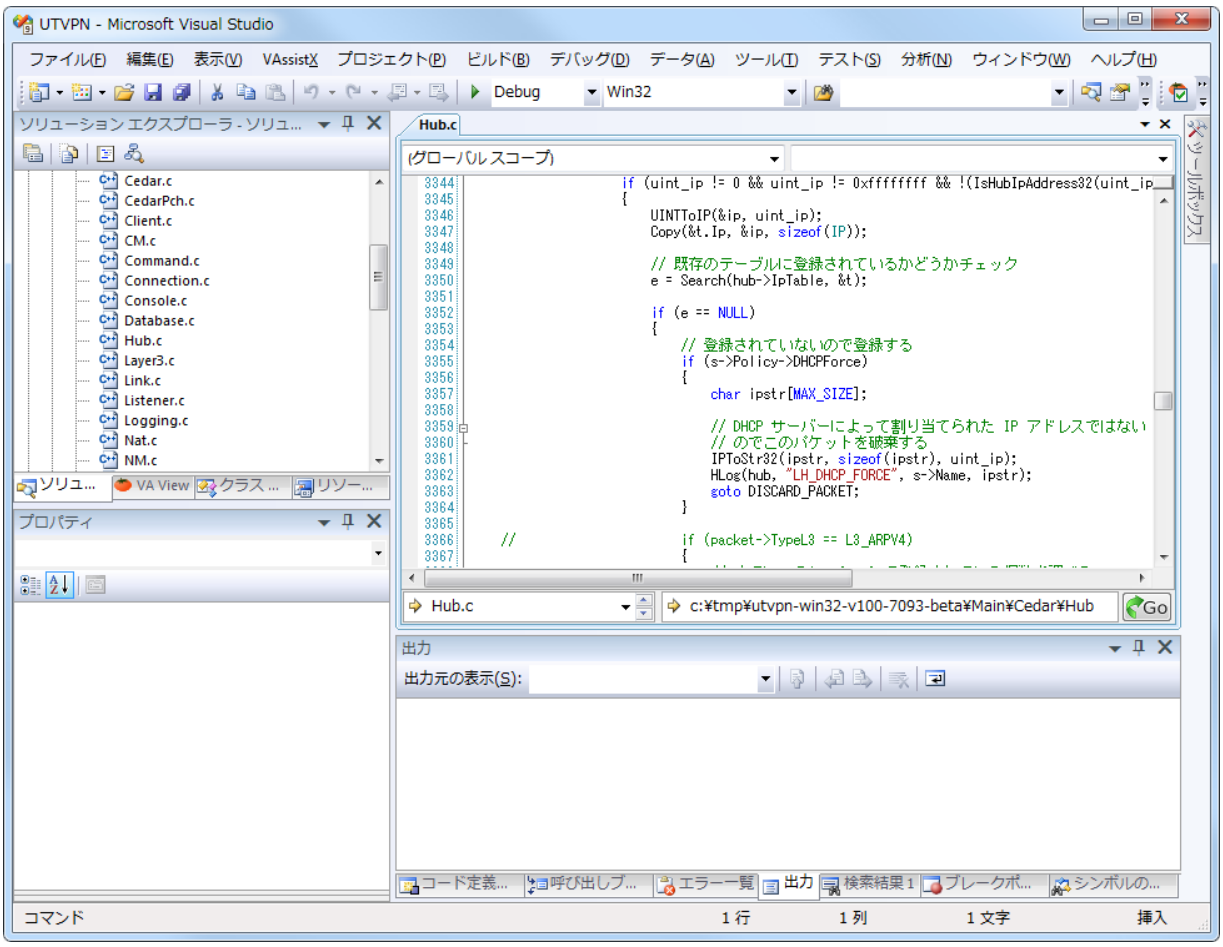
Visual Studio 2008 で UTVPN.sln ファイルを開いてデバッグする

上述の BuildAll.cmd を用いてすべてのインストーラをビルドする方法では、ソースコードを少しずつ改造しながら動作テスト（デバッグ）を行うことは困難です。デバッグを行いながら UT-VPN を改良する作業のためには、Visual Studio 2008 の IDE を使用するのが最も良い方法です。

まず、Visual Studio 2008 用の UTVPN.sln ファイルをダブルクリックして開いてください。

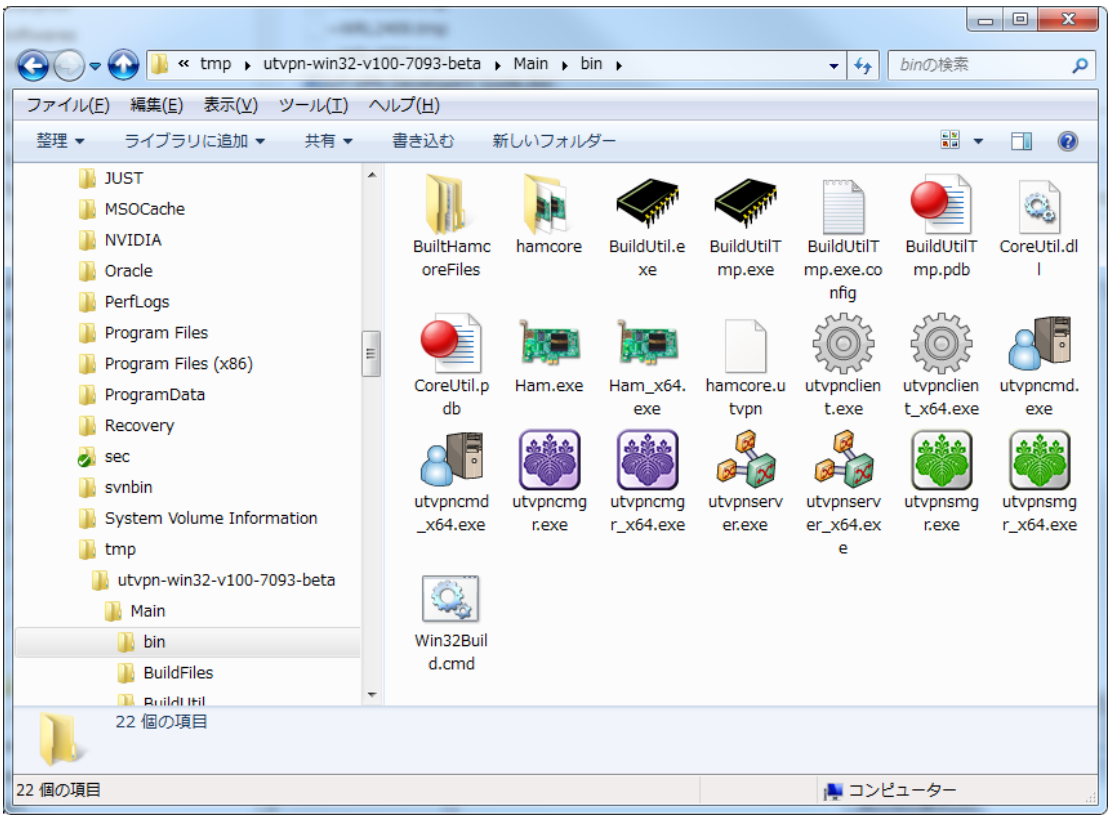


UTVPN.sln を開くと、以下のように Visual Studio の IDE が起動します。



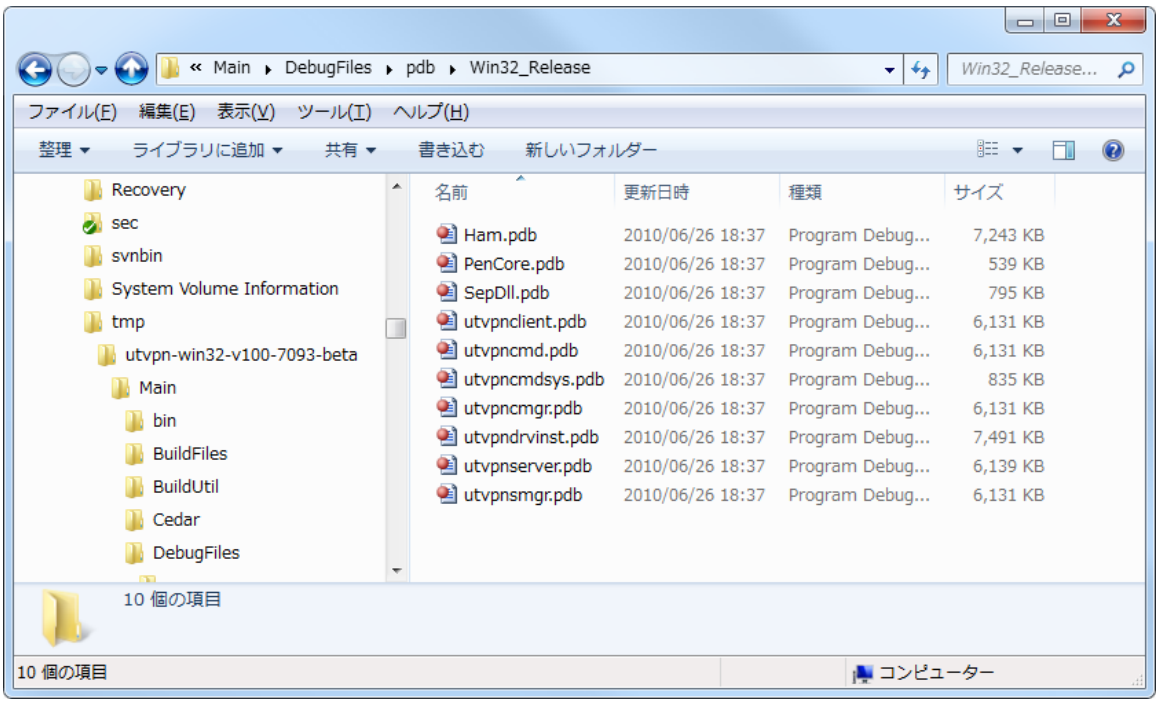
ここで、普通の Visual Studio のプログラムをビルドするのと同様の方法で「ソリューションのビルド」をクリックすると、ビルドが開始されます。ビルドされたファイルは、bin ディレクトリの下に保存されます。

なお、ビルドを行う前に、プラットフォームを「Win32」または「x64」のいずれかから選択してください。また、ビルドモードは「Debug」または「Release」から選択できます。Debug を選択すると最適化が OFF になり、デバッグに適しています。Release を選択すると最適化が ON になり、動作が高速になりますが、デバッグが難しくなります。



これらの.exe ファイルを単にダブルクリックするだけでも動作テストを行うことができます。また、Visual Studio 2008 を用いて適切なプロジェクトを選択し (スタートアッププロジェクトとして選択する)、デバッガを起動することで、普通のプログラムのデバッグと同様にデバッグを行うことができます。

「Debug」モード、「Release」モードのいずれでビルドを行ったときも、出力された.exe ファイルに対応する.pdb ファイル（デバッグ情報ファイル）および.map ファイル（マップファイル）が DebugFiles ディレクトリに保存されます。 .exe ファイルおよびこれらの.pdf および.map ファイルをどこかに保管しておくことにより、.exe ファイルを配布した先のコンピュータでバグが発生したときに、その際のクラッシュ・ダンプを用いてデバッグを行うことができます。



hamcore ディレクトリと hamcore.utvpn の説明

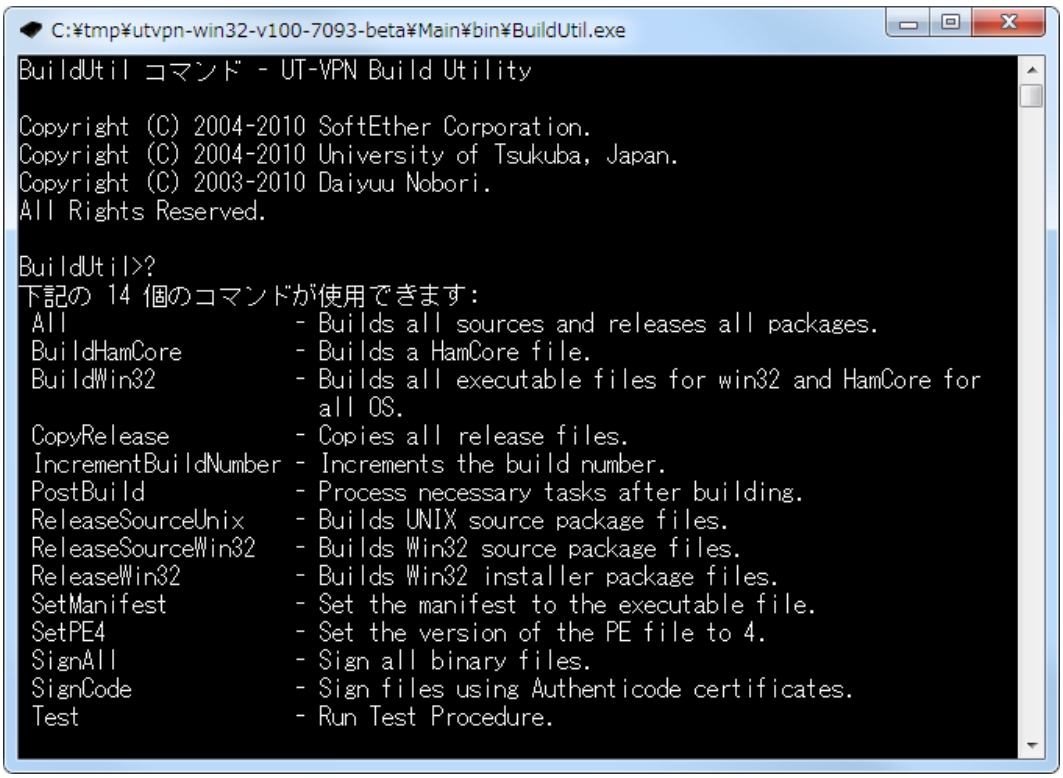
.exe ファイルと同一のディレクトリに hamcore.utvpn ファイル（データファイル）が存在する必要があります。しかし、.exe ファイルと同一のディレクトリに「hamcore」ディレクトリがあり、hamcore ディレクトリ内にファイルが存在する場合は、hamcore.utvpn ファイルにアーカイブされた当該ファイル名のファイルは、hamcore.utvpn から読み込まれず、代わりに hamcore ディレクトリ内にある物理的なファイルから読み込まれます。

たとえば、文字列テーブル「strtable.stb」ファイルは「bin」ディリレクトリ内の「hamcore」ディレクトリ中に保存されています。したがって、デバッグ時に strtable.stb を頻繁に書き換える場合、hamcore.utvpn をその都度再生成する必要はありません。単に「hamcore\strtable.stb」を書きかえればそれで OK です。

hamcore.utvpn ファイルは BuildUtil から生成することができます。「BuildUtil.exe」を起動し、次に「BuildHamcore」と入力して Enter キーを押すと、「BuiltHamcoreFiles」ディレクトリが生成され、その中に Windows 用および UNIX 用の hamcore.utvpn ファイルが自動生成されます。

なお、Windows 用の hamcore.utvpn ファイルにはビットマップやリソースデータ等の大きなファイルがたくさん含まれており、それを削減し必要最小限のファイルのみにしたのが UNIX 用の hamcore.utvpn ファイルです。BuildHamcore コマンドは、Windows 用の hamcore.utvpn ファイルを.exe ファイルと同一のディレクトリに自動的にコピーします。

BuildUtil プログラムはその他にもいくつかの便利な機能を提供しています。詳しくは、BuildUtil を起動して「?」または「help」と入力してみてください。BuildUtil プログラムは C#で記述されており、ソースコードは BuildUtil ディレクトリにあります。

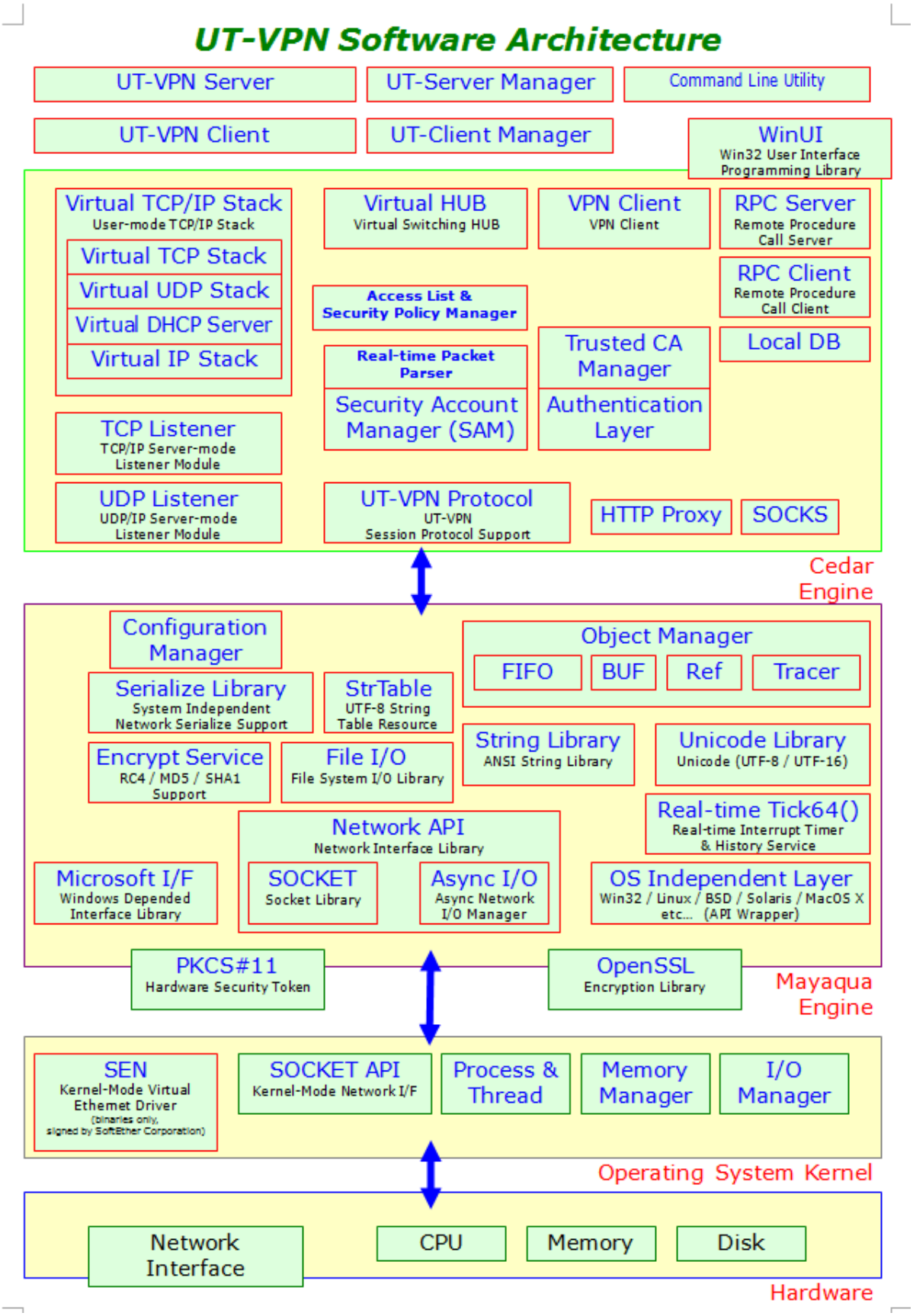


4. UNIX 版ソースコードのビルド

UNIX 版ソースコードをビルドする方法については、UNIX 版ソースコードの.tar.gz ファイルを展開すると出現する「UT-VPN_Startup_Guide_for_UNIX.pdf」を参照してください。

5. 参考資料 – UT-VPN の内部構造

以下の図が UT-VPN の内部構造を理解する上で参考になるかも知れません。



6. 参考資料 – UT-VPN のソースコードを理解するにあたって

Ham.exe の起動

生成された utvpnserver.exe や utvpnclient.exe などのプログラムは Windows のサービスプログラムであり、バックグラウンドで動作するため、入出力用のコマンドラインインターフェイスなどのテストやデバッグに使える機能は持っていません。したがって、通常実装した機能のテストやデバッグなどは utvpnserver.exe や utvpnclient.exe を直接使用することはあまりありません。

その代わり、Ham.exe というプログラムをテスト (動作確認など) に使用します。Ham.exe は UT-VPN Server や UT-VPN Client など UT-VPN に含まれている色々な機能が合体したプログラムで、起動すると DOS プロンプトのような文字入力画面が表示されます。その画面上でいくつかの事前定義されたコマンドを入力することにより、utvpnserver.exe や utvpnclient.exe などで UT-VPN Server や UT-VPN Client などのサービスを実行したときと同じモジュールが起動し、UT-VPN Server / UT-VPN Client / UT-VPN Server Manager / UT-VPN Client Manager / utvpncmd のすべての機能を呼び出してテストすることができます。

Ham.exe を起動する (Visual Studio 2008 で[デバッグ]メニューから[デバッグなしで開始]をクリックするか、bin\Ham.exe を直接開く) と、プロンプト画面が表示されます。
ここで、下記のようなコマンドを入力すると、各種機能が起動します。

コマンド	機能および注意点
ss	UT-VPN Server の機能が起動します。つまり、通常 UT-VPN Server サービスとして動作しているプログラムが、サービスではなく Ham.exe の中で一般のプログラムとして動作します。もちろん Ham.exe を終了すると UT-VPN Server としての動作も停止してしまいます。また、Enter キーを押すと正常終了 (シャットダウン動作) を行い、プロンプトに戻ります。
cc	UT-VPN Client の機能が起動します。つまり、通常 UT-VPN Client サービスとして動作しているプログラムが、サービスではなく Ham.exe の中で一般のプログラムとして動作します。もちろん Ham.exe を終了すると UT-VPN Client としての動作も停止してしまいます。また、Enter キーを押すと正常終了 (シャットダウン動作) を行い、プロンプトに戻ります。
sm	UT-VPN サーバー管理マネージャが起動します。
cm	UT-VPN クライアント接続マネージャが起動します。
utvpncmd	utvpncmd (コマンドライン管理ユーティリティ) が起動します。

なぜ Ham.exe が必要か

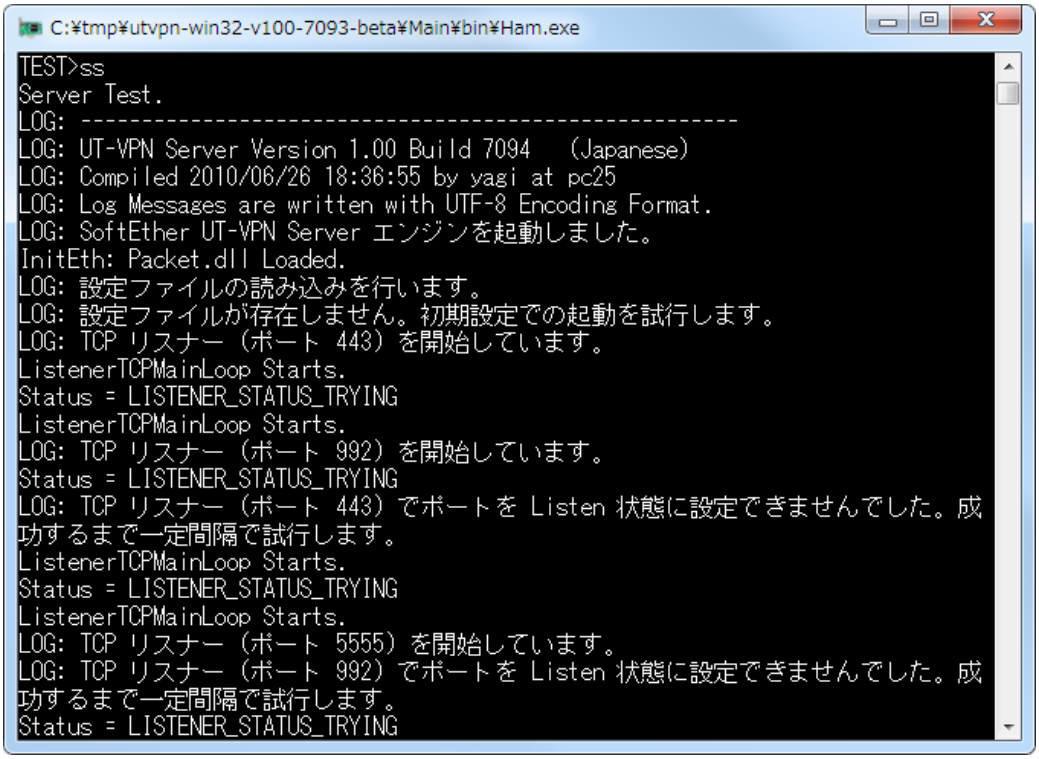
utvpnserver.exe や utvpnclient.exe などのプログラムはサービスとして動作しますので、そのままではデバッグしたり、デバッグのためのメッセージを printf などで表示したりすることは困難です。そこで Ham.exe にすべてのコードをリンクして、Ham.exe を通常のコンソールアプリケーションとして

呼び出せばすべての機能のテストやデバッグができるようにしたのです。

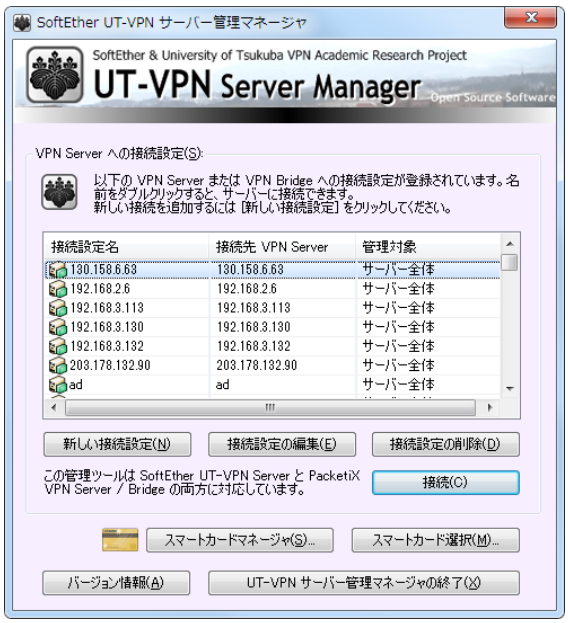
重要: Ham.exe や生成した他の EXE ファイルで UT-VPN Server / UT-VPN Client のコードをテストするときは、必ずローカルコンピュータで既に実行されている UT-VPN Server / UT-VPN Client サービスがある場合は、それらのサービスを停止してください。そうしなければ、ポート番号などの競合が発生してうまくテストできません。

テスト

Ham.exe が起動した後、ss と入力すると UT-VPN Server プログラムモジュールの起動処理が行われ、起動が完了します。下記のような画面が表示されれば成功です。



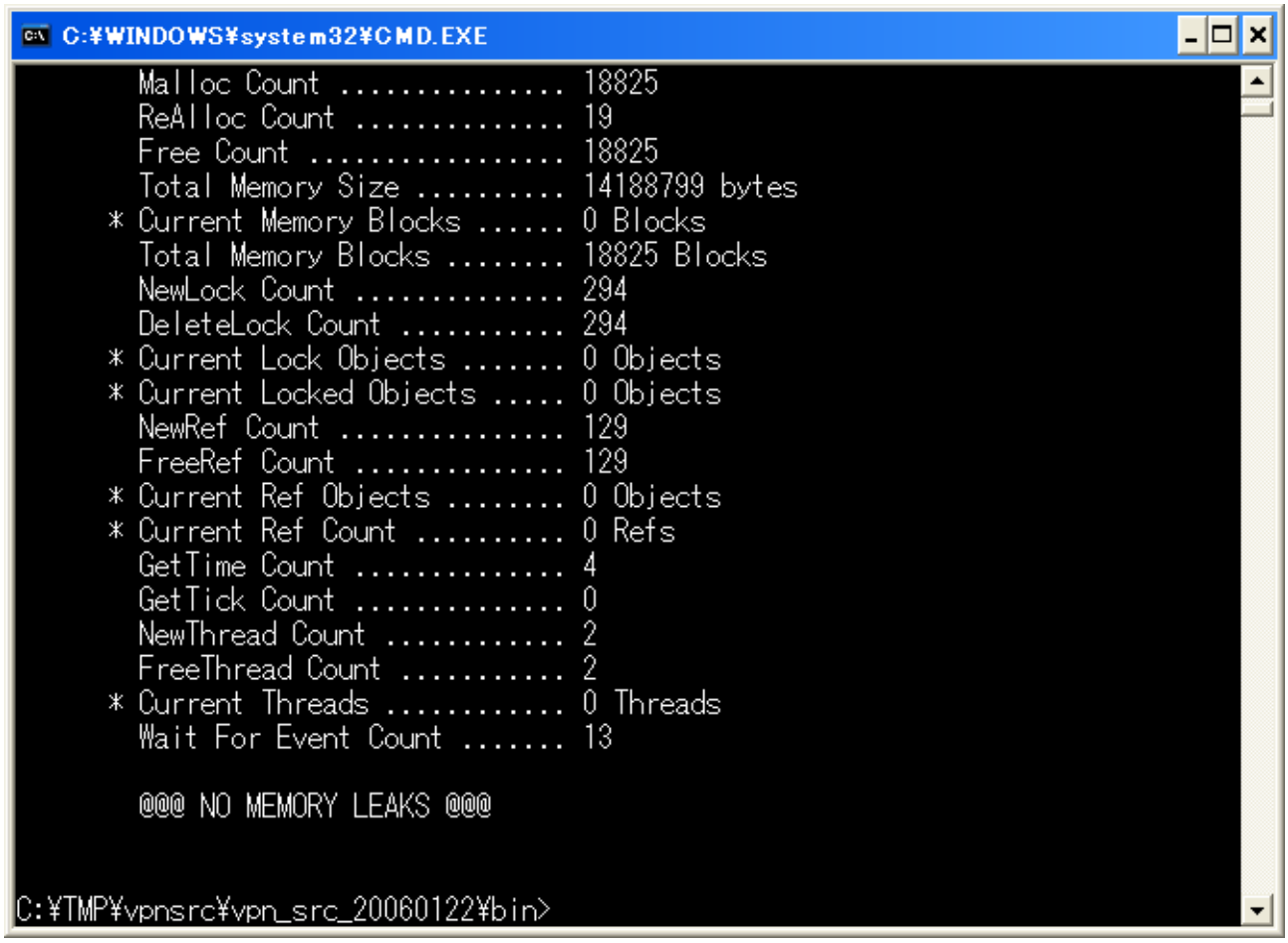
この状態で、もう一つ Ham.exe を起動し、sm と入力すると UT-VPN Server 管理マネージャが起動します。



そして、ここで localhost に対して接続すると、Ham.exe 上で動作している UT-VPN Server に接続することができます。その際、Ham.exe 側にデバッグ用メッセージが表示されますので、よく確認してみてください。

Ham.exe の終了時のメモリリークチェック

Ham.exe としてプログラムを起動した場合は、すべてのメモリ確保/解放、その他のリソースの確保/解放などの操作の際にメモリリークチェックが行われ、Ham.exe が起動してから終了するまでの間に確保したリソースと解放したリソースが一致するかどうかチェックされます。Debug モードでビルドした Ham.exe が終了する際には、下記のような画面が表示され、メモリリークが検出されなかったことが確認できます。



もしメモリリークなどがあれば、ここでメモリリークの個数などが表示されるため、デバッグすることになります。

ソースコードの構造

UT-VPN のソースコードは、主に下記の 3 つのディレクトリに収められています。

ディレクトリ名	内容物
Mayaqua	「Mayaqua Kernel」と呼ばれるコードで、「低レベルレイヤ」と「中レベルレイヤ」の 2 つのレイヤを実装している。主に OS に依存するコードや、そのコードをラッピング (カプセル化) して上位レイヤから容易に扱えるようにする実装のためのレイヤである。
Cedar	「Cedar Communication Module」と呼ばれるコードで、「高レベルレイヤ」と「応用レベルレイヤ」の 2 つのレイヤを実装している。機能自体は Cedar の内部ですべて定義されている。Cedar は Mayaqua を呼び出すことによって、OS の機能 (同期、スレッド、通信、メモリ管理など) を使用することにより VPN 通信機能を実現している。VPN のアルゴリズムやコード自体は Cedar 内部ですべてコーディングされているが、Cedar は原則として Mayaqua の関数のみを呼び出し、直接 OS の機能を呼び出さないようにしている。これにより OS への依存度が下がり、移植性が高くなる。
PenCore	Windows 版で使用されているリソース (Windows リソースのこと。ダイアログボック

	スやアイコン、ビットマップなど) を格納している DLL をビルドするためのプロジェクト。

Mayaqua と Cedar

Mayaqua および Cedar はそれぞれ「スタティックリンクライブラリ」 (Windows では .lib ファイル) を生成するプロジェクトであり、大量の公開関数を含んでいますが、そのままでは実行できません。つまり、UT-VPN Server / UT-VPN Client のコード自体は Mauaqua と Cedar の内部で実装されているのです。したがって、ソースコードを読解する上での大半の作業は Mayaqua と Cedar の読解ということになります。

ソースコードの読解

UT-VPN に含まれるソースコードのプロジェクトの構造・上下関係をこれまでに説明しました。実際に UT-VPN のコードについて理解を深めるためには、ソースコードを隅々まで読むことを推奨します。UT-VPN のソースコードは、大学 1 年生で習う程度の C 言語の知識があればすべて理解することができます。ただし、各コードが実際に何のための処理をしているのかということは、その関数の呼び出し元や呼び出し先との関係やデータ構造を調査しながら読む必要があるため、理解には時間がかかる可能性があります。ソースコード中にはコメントが多くあります。これらのコメントは完全なものではありませんが、ソースコードの内容を理解する上で助けになります。

Mayaqua ソースコード

Mayaqua プロジェクトには下記のソースコードがあり、それぞれの役割は次のとおりです。

ファイル名	役割
Mayaqua.c	Mayaqua Kernel プログラムのうち主要な部分。
Kernel.c	Mayaqua が実装する機能のうち、OS の機能に近い部分のコード。たとえば時刻管理やスレッド管理など。ただし、Kernel.c は API を呼び出すのではなく、OS.c を経由して呼び出すので、Kernel.c は OS 非依存である。
OS.c	OS が提供する各種システムコールを呼び出すコードは Win32.c と Unix.c に分離されているが、これらの関数一覧のテーブルを元に現在動作している OS 用のコードを呼び出すような仕組みが実装されている。実際には関数テーブルがあって、それを呼び出すスタブが実装されているだけ。
Win32.c	Win32 API が使用できるシステム上で Win32 API を呼び出すコード。Windows 版 VPN をビルドする際にリンクされる。
Unix.c	POSIX API が使用できるシステム上でシステムコールを呼び出すコード。Linux 版などの UNIX 版 VPN をビルドする際にリンクされる。
Cfg.c	コンフィグレーションマネージャ。Windows のレジストリのようなツリー状の構成データを実現するためのコード。

Encrypt.c	暗号化や電子署名などのセキュリティを提供するコード。X.509 証明書や RSA 暗号を上位レイヤから簡単に扱うことができるようなカプセル化の役割を持っている。実際には OpenSSL に依存している。
FileIO.c	ファイル入出力や OS 間の名前空間の違いを吸収するモジュールコードが含まれている。
Memory.c	メモリ管理およびストレージ用データ構造が定義されている。
Network.c	TCP/IP、UDP/IP および SSL 通信を行うためのモジュール。これらの呼び出し規則は OS によって異なるので、このコードで吸収している。
Object.c	イベント、ロック、参照カウンタなどのシステムオブジェクトを実現しているコード。
Pack.c	構造体や配列などのデータ形式をカプセル化 (シリアライズ) してバイナリデータに変換し、それをネットワーク上に流すためのモジュール。汎用データ転送コード。RPC で使用している。
Secure.c	スマートカードなどのセキュリティトークンにアクセスする PKCS#11 モジュールをカプセル化するコード。
Str.c	文字列処理コード。ANSI 文字列 (1 バイト文字列) を扱う。StrCpy、StrLen など。
Internat.c	文字列処理コード。Str.c が ANSI 文字列用であるのに比べて、Internat.c は Unicode 文字列用である。また Unicode 文字列と ANSI 文字列との変換、UTF-8 エンコーディングなども担当する。
Table.c	文字列テーブル (hamcore\strtable.stb) の読み込みとキャッシュのためのモジュール。
Tick64.c	リアルタイム時刻取得および逆戻りしない一貫性を持った 1ms・64bit 精度のシステムタイマ管理コード。
Tracking.c	Mayaqua 中で確保した OS リソースの確保場所を行番号・コールスタックとともに追跡し、終了時に解放していないものを検出するためのトラッキングコード。メモリリークの発見などに役に立つ。
Microsoft.c	Windows に依存したコードのうち、Unix 用には無いものが多い。そこで Windows を呼び出す必要があるコードで Win32.c に含むことが適当でないものを Microsoft.c に実装した。Windows プラットフォーム用の色々な処理を行う。

Cedar ソースコード

Cedar プロジェクトには下記のソースコードがあり、それぞれの役割は次のとおりです。

ファイル名	役割
Cedar.c	CEDAR オブジェクトを管理するための重要なコード。また主要なデータ構造についても実装している。
Account.c	VPN Server の仮想 HUB のユーザー、グループやセキュリティポリシ

	ーなどを管理するコード。
Admin.c	VPN サーバー管理マネージャまたは vpncmd が VPN Server を制御する際の、呼び出し側 RPC スタブと呼び出される側の RPC サーバー関数定義。
Bridge.c	ローカルブリッジなどに使用される、Ethernet デバイスへの低レベルアクセス用コード。内部では BridgeWin32.c または BridgeUnix.c を呼び出す。
Client.c	VPN Client のサービスとしての機能およびクライアント接続マネージャまたは vpncmd がサービスに接続して制御する際の、呼び出し側 RPC スタブと呼び出される側の RPC サーバー関数定義。
CM.c	Win32 用 VPN クライアント接続マネージャのコード。
Command.c	vpncmd コマンドライン管理ユーティリティのコード。
Connection.c	コネクションオブジェクトの管理コード。
Console.c	コンソールサービス。Win32 および UNIX ではコンソール（コマンドプロンプト）の制御方法が異なるので、それを吸収すると共に、便利なユーティリティ関数を提供する。Command.c によって使用される。
Hub.c	VPN Server の仮想 HUB のコード。
Layer3.c	VPN Server の仮想レイヤ 3 スイッチのコード。
Link.c	仮想 HUB 間カスケード接続機能のコード。
Listener.c	TCP/IP リスナーポートコード。
Logging.c	ロギングサービス。ログデータを中間形式で保存用キューに入れて、遅延評価してディスクに書き出す。
NAT.c	SecureNAT 機能の一部の管理用インターフェイスなどを定義するコード。
NullLan.c	テスト用の NULL デバイス（仮想 LAN カードのようなインターフェイスをソフトウェア的に持つが、パケットは送信しない。ただし乱数でランダムなパケットを送信しているようにすることができる）のコード。
Protocol.c	VPN プロトコルのうち、主にネゴシエーション部分の実装。ネゴシエーションが完了した後、VPN セッション上の通信に移ると、制御は Session.c に移行される。
Remote.c	Remote Procedure Call クライアントおよびサーバー用コード。
Sam.c	Security Account Manager。VPN Server でのユーザー認証などのインターフェイスを共通化し、各種ユーザー認証方法を一元呼び出しにて用いることができる。
SecureNAT.c	SecureNAT 機能のセッション生成および Virtual.c へのデータの受け渡しコード。
Server.c	VPN Server のサービスとしての実装。また、クラスタリング機能の制御もこのコードで行われている。

Session.c	VPN セッション管理コード。また VPN セッションでの通信のメインループもここで実装されている。非常に重要。
SM.c	VPN サーバー管理マネージャのコード。
TcpIp.c	TCP/IP プロトコルの解析モジュール。Ethernet パケットを解析し、TCP/IP のヘッダ情報やさらに上位レイヤのアプリケーションデータを読み取る。
UT.c	VPN クライアント接続マネージャの「ネットワークデバイスの状態」機能の実装。
Virtual.c	仮想 TCP/IP スタックおよび仮想 NAT をユーザーモードで実装したコード。SecureNAT 機能の仮想 NAT 機能および仮想 DHCP サーバー機能で使用されている。単一のスレッドで TCP/IP のような多重処理を行うのは大変技術的に複雑なので、ソースコードの行数も多い。
VLan.c	仮想 LAN カードデバイスドライバとの通信を行うコード。VLanWin32.c と VLanUnix.c を呼び出す。
Win32Html.cpp	唯一の C++ソースコード。COM を呼び出す必要があるコードがここで記述されている。特に重要な処理はない。
WinUi.c	Windows 用 GUI コードのうち、各モジュールから頻繁に呼ばれるような処理を一元化したライブラリコード。

似非オブジェクト指向プログラミング

UT-VPN のような複雑なコードを記述する上では、オブジェクト指向でのプログラミングが必須でした。しかしながら、C 言語ではクラスの概念が無いため、大部分を構造体とその構造体のインスタンスを生成 / 破棄 / 操作するコードとして実装してあります。

たとえば、SESSION という名前の構造体がある場合、これをインスタンス化したものを「SESSION オブジェクト」と呼びます。

なお、ここでの「オブジェクト」とは C++等のオブジェクト指向プログラム言語におけるオブジェクトと比較すると大きく劣ります。どちらかというと、Windows Kernel および Win32 API におけるオブジェクトと似たような意味でこう呼んでいます。

オブジェクト関係の操作関数の命名規則

共通の関数名の命名規則として、オブジェクトを作成する場合は「NewSession」などのように「New オブジェクト名」という関数を使用することにしています。

例えば、他に USER オブジェクトというものを作成する関数を探したい場合は、NewUser 関数を呼び出すことによってその種類のオブジェクトを生成しています。

また、オブジェクトを解放する場合は、「Free オブジェクト名」や「Release オブジェクト名」または「Delete オブジェクト名」というような関数名をデストラクタとして使用しています。Free / Delete と Release の違いは、次のセクションで説明します。

自動解放機能無しオブジェクト

オブジェクトの中には、「New オブジェクト」のような関数で生成したインスタンスを、それほど複雑でない状況で使用および破棄するようなものがあり、このような種類のオブジェクトは「Free オブジェクト名」や「Delete オブジェクト名」というような名前の関数で破棄します。

たとえば、NewLock 関数は新しい LOCK オブジェクトを作成しますが、これは DeleteLock 関数で破棄します。

```
LOCK *neko = NewLock();
    :
DeleteLock(neko);
```

自動解放機能付きオブジェクト (参照カウンタ)

オブジェクトの中には、複雑な状況 (特にマルチスレッドプログラミングが絡んでいて、そのオブジェクトを作成したスレッド以外のスレッドでもそのオブジェクトのインスタンスが複数参照される可能性があるなどの理由によって、どのタイミングで破棄すれば良いかを事実上事前に決定することができないような状況) で作成されるものを前提としているものもあります。そのようなオブジェクトは、「New オブジェクト名」の関数で作成しますが、「Release オブジェクト名」の関数で解放します。また、REF オブジェクト (参照カウンタオブジェクト) をメンバ変数として持っており、ref という名前 (違う名前の場合もあります) でそのオブジェクトの参照カウントを管理しています。オブジェクトを作成したときは、参照カウンタの値は 1 です。そのインスタンスを別のスレッドにも渡して処理させる場合は、参照カウンタを AddRef で増加させてやります。逆に参照していたスレッドがそのオブジェクトへのポインタを解放したい場合は「Release オブジェクト名」を呼ぶと参照カウンタが 1 減ることになります。参照カウンタが 0 になると、そのオブジェクトが実際に破棄されるようなコードが組んであります。このような仕組みによって、マルチスレッド環境で複雑な状況で使用される可能性があるオブジェクトのコードでのメモリリークの発生や、逆にまだ参照しているにもかかわらずインスタンスが破棄されてしまうことなどを防いでいます。

Mayaqua に含まれるオブジェクト一覧

Mayaqua モジュールには下記のようなオブジェクトの定義が含まれている。なお、下記は主要なものをリストアップしただけであり、実際には他にも存在する。

オブジェクト名	参照カウンタ付き	説明
BUF		バッファ (自動サイズ拡張機能付きのバイナリデータストア)
CANCEL	はい	キャンセレーションオブジェクト。Select 関数でのブロッキング中にそれを別スレッドからキャンセル解除するために使う。
CANDIDATE		文字列候補オブジェクト。
DIRLIST		ディレクトリエントリリスト。特定のディレクトリ内の

		ファイルおよびディレクトリ一覧、ファイル名、情報などのリストが格納されている。
EVENT	はい	イベントオブジェクト。Win32 におけるイベントと同等の同期機能を提供する。UNIX でも同等に動作する。
FIFO		FIFO バッファ。バッファの末尾から新しいデータを追記でき、また先頭からデータを読み出して切り詰めることができる。
K		RSA の公開鍵または秘密鍵オブジェクト。
X		X.509 証明書オブジェクト。
P12		PKCS#12 オブジェクト。
LIST	はい	リストオブジェクト。複数のデータ項目をリストに追加したり、削除したり、ソートしたりすることができる。
LOCK		ロックオブジェクト。複数のスレッドが同一ロックを同時に取得できないようにする。
OBJECT		リソースリークの検出用のリソース確保履歴保存オブジェクト。
PACK		Pack.c で使用するデータ構造を保持するオブジェクト。
QUEUE	はい	キューオブジェクト。
SK	はい	スタックオブジェクト。
REF		参照カウンタオブジェクト。他の参照カウンタ付きオブジェクトの内部で利用されている。
SOCK	はい	TCP/IP または UDP/IP ソケット通信におけるソケットを抽象化したオブジェクト。
SOCK_EVENT	はい	複数のソケットを束ねることによりイベントを生成し、そのイベントオブジェクト (特殊なイベントオブジェクト) を監視することにより、どれか 1 つのソケットで送受信が発生するまでコードをブロックできる。複数のソケットを非同期通信する処理を 1 つのスレッドで行うために便利。
SOCKSET		複数のソケットをリストにしたものである。Select 関数で SOCKSET を指定することによって、SOCKSET に含まれているすべてのソケットを監視することができる。SOCK_EVENT と似たような仕組みである。
THREAD	はい	スレッドを抽象化したもの。実際には内部的にスレッドプールを管理しており、スレッドの初期化・解放などのオーバーヘッドを低減している。

Cedar に含まれるオブジェクト一覧

Cedar モジュールには下記のようなオブジェクトの定義が含まれている。なお、下記は主要なものをリストアップしただけであり、実際には他にも存在する。

オブジェクト名	参照カウンタ付き	説明
AC		IP アクセス制御リストのエントリ。
ACCESS		アクセスリスト (パケットフィルタリングルール) のエントリ。
ACCOUNT		VPN Client の接続設定 (内部的には「アカウント」と呼んでいる) のエントリ。
ADMIN		VPN Server 側で、サーバー管理モードとして接続してきた VPN サーバー管理マネージャや vpncmd からの接続セッションに対応して生成されるコンテキスト。その管理セッションの権限 (管理することができる仮想 HUB の名前、接続モードなど) が格納されている。
ADMIN_OPTION		仮想 HUB の管理オプションのエントリ。
ARP_ENTRY		SecureNAT 機能で内部的に保持している ARP テーブルのエントリ。
ARP_WAIT		SecureNAT 機能の内部で ARP 解決を行おうとする際、ARP リクエストを送信した後に一旦キューに格納し、応答があるまで待機する。そのための待機リストのエントリ。
AUTHNT		VPN Server での NT ドメイン認証のためのユーザー認証設定データ格納用構造体。
AUTHPASSWORD		VPN Server でのパスワード認証のためのユーザー認証設定データ格納用構造体。
AUTHRADIUS		VPN Server での Radius 認証のためのユーザー認証設定データ格納用構造体。
AUTHROOTCERT		VPN Server での署名済み証明書認証のためのユーザー認証設定データ格納用構造体。
AUTHUSERCERT		VPN Server でのユーザー固有証明書認証のためのユーザー認証設定データ格納用構造体。
BLOCK		Ethernet フレームを一時的に格納しておくことができるパケットバッファのエントリ。内部データが圧縮されている場合もある。
BRIDGE		ローカルブリッジセッションオブジェクト。
LOCALBRIDGE		ローカルブリッジ定義オブジェクト。これをもとにしてローカルブリッジセッションが生成される。
CAPSLIST		サーバー能力を示すオブジェクト。

CEDAR	はい	CEDAR オブジェクトは VPN Client / VPN Server の機能を実現する上で最も重要なオブジェクトである。非常に重要な情報が記録されたり、内部的に生成するほかのオブジェクトのインスタンスを保持したりするためのコンテキストである。
CLIENT	はい	VPN Client を実現するためのコンテナオブジェクト。
CLIENT_AUTH		VPN Client 側で VPN Server への接続設定の一部として認証データ (ユーザー認証に使用するパラメータ) を指定するが、そのための認証データを保持するためのオブジェクト。
CLIENT_CONFIG		VPN Client の設定情報を保持するオブジェクト。
CLIENT_OPTION		VPN Client の接続設定や VPN Server のカスケード設定などにおける接続先の VPN Server や仮想 HUB 名などの主要な情報を保持するオブジェクト。
CONNECTION	はい	VPN セッションのコネクションを定義し必要な情報を保持するオブジェクト。SESSION から参照される。VPN Server に対して接続してきた論理的な TCP/IP コネクションはすべて CONNECTION オブジェクトのインスタンスにマッピングされ管理される。その際 CONNECTION オブジェクトは CEDAR の保持リストの管理下にある。その後、セッションの初期化が完了すると SESSION オブジェクトが CONNECTION オブジェクトへのリンクを持つようになり、CEDAR からの直接リンクは切れる。
CONSOLE		コンソールサービスを提供するオブジェクト。Windows / UNIX ではコンソール入出力の API が異なるので、この CONSOLE オブジェクトの内部にコンソール入出力のための関数ポインタが格納され、それを通じてコンソールを操作することをサポートする。
CRL		無効な証明書のエントリを保持するオブジェクト。
CT		コンソール上にテーブル状のデータを出力する際にテーブルの列一覧 (CTC) と行一覧 (CTR) のデータ定義を保持するオブジェクト。
DHCP_LEASE		SecureNAT 機能の仮想 DHCP サーバー機能でリースした DHCP エントリを記憶するオブジェクト。
DHCP_OPTION		SecureNAT 機能の仮想 DHCP サーバー機能で DHCP クライアントに対して送信する DHCP オプションの内容を記憶するオブジェクト。
DHCP_OPTION_LIST		SecureNAT 機能の仮想 DHCP サーバー機能で DHCP

		クライアントに対して送信する DHCP オプションの内容を記憶するオブジェクト。
ERASER		VPN Server などでログファイルを消去する機能のためのコンテキストを保持するオブジェクト。
ETH		Ethernet アダプタ (LAN カード) に対して直接アクセスする際に経由するオブジェクト。
FARM_CONTROLLER		クラスタメンバがクラスタコントローラへ接続する際に生成される接続状況を保持するオブジェクト。
FARM_MEMBER		クラスタコントローラがクラスタメンバからの接続を受け付けた際にクラスタメンバの情報と接続状況を保持するオブジェクト。
FARM_TASK		クラスタコントローラがクラスタメンバに対してタスク (要求) を投げるが、そのタスクの内容を保持するオブジェクト。
HUB	はい	VPN Server における仮想 HUB を管理するオブジェクト。
HUB_LOG		仮想 HUB のログ保存設定を保持するオブジェクト。
HUB_OPTION		仮想HUBのオプション設定を保持するオブジェクト。
HUB_PA		仮想HUB用パケットアダプタ情報を保持する構造体。パケットアダプタとは、セッション側から見たパケット処理部分であり、セッション側からは仮想 HUB も仮想 LAN カードも同一のインターフェイス (パケットアダプタと呼んでいる) でその先のパケット処理モジュールにパケットを投げることができるようにしている。
HUBDB		仮想 HUB 内のユーザーリストや証明書データベースなどのリスト状のデータを保持するオブジェクト。
IP_COMBINE		SecureNAT 機能で使用されている、複数に分割された IP パケットを結合するために一時的にデータを保持する構造体。
IP_TABLE_ENTRY		仮想 HUB 内で保持している IP アドレステーブルのエントリ。
IP_WAIT		SecureNAT 機能で使用されている、IP パケットを送信しようとしているがまだ ARP 解決が完了していないため一時的に IP パケットの内容を記憶しておくオブジェクト。
KEEP		インターネット接続の維持機能を実現するためのコンテキストを保持するオブジェクト。
L3ARPENTRY		仮想レイヤ3スイッチ機能で内部で保持している ARP

		テーブルのエントリを表現するオブジェクト。
L3ARPWAIT		仮想レイヤ 3 スイッチ機能で解決待ち状態の ARP リクエストを保持するオブジェクト。
L3IF		仮想レイヤ 3 スイッチ機能の仮想インターフェイスを実現するオブジェクト。
L3PACKET		仮想レイヤ 3 スイッチ機能で、IP パケットを送信しようとしているがまだ ARP 解決が完了していないため一時的に IP パケットの内容を記憶しておくオブジェクト。
L3SW		仮想レイヤ 3 スイッチを定義するオブジェクト。
L3TABLE		仮想レイヤ 3 スイッチが保持しているルーティングテーブルのエントリを表現するオブジェクト。
LINK	はい	カスケード接続の機能を提供するためのコンテキストを保持するオブジェクト。
LISTENER	はい	TCP/IP リスナーの機能を提供するためのコンテキストを保持するオブジェクト。
LOG		ログ保存システムを管理するためのオブジェクト。
NAT		SecureNAT 機能を管理するためのオブジェクト。SecureNAT 機能が内部で使用するデータを保持するコンテキストである。
NAT_ENTRY		SecureNAT 機能のうち仮想 NAT 機能が内部で保持している NAT アドレス変換エントリを表現するオブジェクト。
NODE_INFO		VPN Server が VPN 接続元のコンピュータの情報を管理するためのオブジェクト。
PACKET_ADAPTER		パケットアダプタ (前述) におけるパケット入出力および非同期イベント通知のための関数ポインタを含むオブジェクト。
PACKET_LOG		VPN Server でログ保存するパケットの内容を保持するオブジェクト。
POLICY		セキュリティポリシーの設定値を保持するオブジェクト。
ROUTE_TRACKING		VPN Client でルーティングテーブルの状態遷移を管理するコンテキスト保持用オブジェクト。
RPC		VPN Server / VPN Client に対する管理用接続 (RPC) を管理するオブジェクト。RPC クライアントおよび RPC サーバーとして使用する。
SERVER	はい	VPN Server を実現するためのオブジェクト。
SESSION	はい	VPN セッションを管理するためのオブジェクト。VPN

		サーバー側 / VPN クライアント側の両方でこのオブジェクトが共通に使われる。内部にはスレッドへのリンクや同期オブジェクトなどが含まれており、VPN 通信において最も重要なオブジェクトである。
SNAT		SecureNAT 機能のために必要なコンテキストを保持するオブジェクト。
STORM		ブロードキャストストームの発生を防ぐためのブロードキャストパケット数を記録しておくオブジェクト。
TCP		VPN セッションを構成する TCP/IP コネクションを管理するオブジェクト。複数本の TCP/IP コネクションによって構成される。同様に UDP というオブジェクトもあったが、現在のバージョンの VPN 2.0 では UDP を用いた通信はサポートしていないため、未使用である。
TCPSOCK		VPN 通信を行うための TCP ソケットを保持するためのオブジェクト。その他にも必要なコンテキストデータを保持している。TCPSOCK は、TCP オブジェクトがリストとして保有している。
TICKET		クラスタコントローラで接続認証が完了した VPN 接続について発行されるチケットをクラスタメンバに対して送信するとともに、接続元 VPN クライアントに対してチケットを渡す。チケットを受け取ったクラスタメンバは一定時間チケットを保持する。そのためのデータを保持するオブジェクト。
TRAFFIC		トラフィックデータを保持するオブジェクト。パケットの送受信数や合計サイズなどが含まれている。
TRAFFIC_DIFF		トラフィック差分データを保持するオブジェクト。クラスタメンバでは一定時間ごとに前回送信時を 0 としてそれ以降に処理したパケットの送受信サイズなどのトラフィックデータを VPN Server に対して送信することにより、VPN Server に対して完全にリアルタイムではないがある程度頻繁な間隔で最新のトラフィック差分情報を通知する。
TRAFFIC_LIMITER		通信速度のスロットリングに使用されるデータを保持するオブジェクト。
TTC		スループット測定ツールのクライアント側のオブジェクト。
TTS		スループット測定ツールのサーバー側のオブジェク

		ト。
USER	はい	VPN Server の仮想 HUB に登録されるユーザーオブジェクト。
USERGROUP	はい	VPN Server の仮想 HUB に登録されるグループオブジェクト。
VH	はい	仮想ホスト (SecureNAT の仮想 IP ホスト) を保持するコンテキストオブジェクト。
VH_OPTION		仮想ホストの設定データを保持するオブジェクト。
VLAN		仮想 LAN カードを管理するオブジェクト。

7. UT-VPN ソースコードに関する練習問題集

これまでの説明で、UT-VPN のソースコードを読解するために必要な前提知識の大半を解説しました。後は、通常の C 言語に関する知識と、ごくわずかの Win32 API およびバークレーSocket などの知識を持っているだけで、UT-VPN のソースコードのすべての部分を理解するとともに、自ら好きな機能を追加したり、バグを発見したりすることができるようになるはずです。

実際に UT-VPN ソースコードに関する十分な理解力を身につけることができたかどうかは、下記のいくつかの質問について解答してみてください。もしわからない点があれば、ソースコードを実際に読んで調べてみてください。このようにすることで、UT-VPN のソースコードに関する理解がより深まります。

Mayaqua ライブラリ（低レベルライブラリ）に関する質問

1. hamcore.utvpn ファイルの役割は何ですか?
2. hamcore. utvpn ファイルが見つからない場合の既定の動作は何ですか?
3. UT-VPN ソースコード中で現在の 64bit システム時刻（tick 値）を取得するために頻繁に呼ばれる関数がありますが、その関数の名前は何ですか?
4. Tick64.c では新しいスレッドを生成しています。そのスレッドが果たしている重要な役割を 2 つ答えてください。
5. 64bit システム時刻（tick 値）は、システム時計がユーザーまたは自動システム時刻調整ソフト(NTP クライアントなど）によって突然逆戻りさせられても、常に前方向に向かって進むようになっています。UT-VPN プログラムでは、ユーザーまたは NTP クライアントなどが何度かシステム時計を変化させても、ある瞬間の tick 値が生成されたときの、その時点での正確なシステム時計が指していた時刻を後から取得することができるようになっています。そのための関数が Tick64ToTime64 (TickToTime) 関数ですが、これを利用している UT-VPN 内の重要な機能と、その機能が tick 値をその時点での正確なシステム時計の値に変換する必要性を説明してください。
6. Win32 では 64bit システム時刻を取得するために timeGetTime 関数を用いています。timeGetTime 関数は非常に高速であり、呼び出す際のオーバーヘッドが全く無いため、Win32 で Tick64 を呼び出すとその都度 timeGetTime で最新の tick 値を取得しています。しかし、Linux やその他の UNIX のタイマはそれほど性能が良くなく、gettimeofday 関数などでその都度タイマを取得すると、カーネルモード呼び出しのためのオーバーヘッドが無視できず、たとえば毎秒 1 万個のパケットについて tick 値を取得して記録するというような用途に使えません。そこで Linux やその他の UNIX 上で Tick64 関数を呼び出した場合はどのような仕組みでオーバーヘッドの問題を解決しようとしているかを解説してください。
7. BUF オブジェクトはどのような目的のために存在しているのか解説してください。
8. Mayaqua ライブラリが提供するファイル入出力関数では、ファイル名として文字列を渡すとき、その文字列の最初の 1 文字目が'|'である場合、および'@'である場合に特別扱いをします。どのような特別扱いをしますか?
9. UNIX 上で UT-VPN のコードをリリースモードでコンパイルしても、メモリ確保のための関数としては malloc が呼ばれるだけですが、Win32 版ではより最適化が施されており、malloc ではなく別

の関数を呼んでパフォーマンスを向上しています。その最適化手法について解説してください。

10. hamcore ファイルシステムは FileOpen などのファイル操作系関数から通常のファイルと同様に内部ファイルを読み込むことができますが、そのような操作ができるようになっている理由および実装上の工夫について解説してください。
11. Windows では自分自身（exe ファイル）のフルパスを取得するのは簡単ですが、UNIX 上では標準的な方法がありません。そこで UT-VPN の UNIX 版コードでは自分自身の実行可能ファイル名を力技で取得しようと試みます。ほとんどの場合、それは成功しますが、どのようなアルゴリズムでそれを実現しているか解説してください。また、その方法が失敗すると思われる例を少なくとも 2 つ説明してください。
12. Mayaqua ライブラリを使うと、たとえば /usr/local/bin/../../local/bin/../bin/../../lib/ というような冗長なパス指定を、自動的に /usr/lib/ というように正規化することがとても簡単にできますが、それを行うためにはどの関数を呼べば良いでしょうか?
13. Format 関数の役割について説明してください。
14. Format 関数と UniFormat 関数の違いについて説明してください。また、Format 関数や UniFormat 関数は C 言語標準ライブラリなどの sprintf 系関数で代用できるような気がしますが、なぜ複雑な方法で実現しているのかを推測してください。
15. ParseToken 関数が内部でロックを用いている理由を推測してください。
16. SearchStr および ReplaceStr 関数は非常に効率の悪いアルゴリズムで実現されていますが、これは実装時に時間が無かったためです。これをより効率の良いアルゴリズムに置き換えるとしたら、どのアルゴリズムが最適でしょうか?
17. Windows 上と UNIX 上で Unicode 文字列を扱う場合、動作結果は全く同一ですが、内部的に行っている処理は全く異なります。UNIX では libc による Unicode サポートが貧弱であるため、ある有名なライブラリを呼び出して処理していますが、そのライブラリは何ですか?
18. Windows の Win32 スレッドと UNIX の pthread では、API が全く違うにもかかわらず、UT-VPN のソースコード中では両者で全く同様に動作するようなコーディングが可能になっています。そのためにどのような工夫を Mayaqua ライブラリが行っているかについてできるだけ詳しく解説してください。
19. WaitThreadInit 関数は何のために存在しますか?
20. Windows の Win32 スレッドは WaitForSingleObject によって複数スレッドから終了を待機することができますが、UNIX の pthread では pthread_join で待機することができる呼び出し元スレッドは必ず 1 つだけに限定されます。しかし、多くの状況では複数スレッドが特定スレッドの正常終了を待機する必要があります。そのため Mayaqua ライブラリではある工夫をして複数スレッドから特定スレッドに対する WaitThread を呼び出すことができるようにしていますが、その方法について解説してください。
21. HashInstanceName 関数の役割と存在する必要性を解説してください。
22. FileToBuf 関数、BufToFile 関数は何のために存在しますか? DumpBuf 関数、ReadBuf 関数と比べて何が違うのでしょうか?
23. FIFO オブジェクトは何のために存在しますか? また、リングバッファを用いない理由を推測してください。

24. リスト (LIST) データ形式はそのアルゴリズム上のメリットとデメリットがあります。つまりある特定の状況では利益がある (コーディングが簡単だったり、速度が高速だったりする) が、別の状況では不利益がある (速度が低速になる) のですが、それらについて解説してください。
25. リストデータ形式内の特定番目のデータを取得するコードは高速化のためマクロを用いて書きますが、どのように書きますか?
26. たとえばリストデータ内のすべての項目を走査して、特定条件に一致したものだけをそのリストから削除するというようなコードは UT-VPN 内で多く発見されますが、いずれも少し回りくどい手法を使ってコーディングしています。その手法はどのような手法ですか?
27. ListKeyToPointer 関数は何のために存在していますか?
28. QUEUE データ型は別のデータ型を内部的に利用しています。その別のデータ型は何ですか?
29. Windows でも Linux / UNIX でも LOCK オブジェクトによるロックは再帰ロック (あるスレッドがロックを取得したあと、もう一度同一ロックを取得しようとしても、そこでデッドロックせずに内部的にカウンタを操作すること) が可能であることが Mayaqua ライブラリによって保証されていますが、それはどのような仕組みによるものなのか、Windows 版と UNIX 版について解説してください。
30. コンフィグレーションマネージャ (.config ファイルを管理・操作するモジュール) は、既定ではテキスト形式で .config ファイルを読み書きしますが、ユーザーがある設定を行うとバイナリ形式で .config ファイルを書き出すことができ、パフォーマンスが向上します。そのための具体的なユーザー操作を答えてください。
31. コンフィグレーションマネージャ (.config ファイルを管理・操作するモジュール) は、読み込もうとしている .config ファイルがバイナリ形式かテキスト形式かをどのような方法で判別していますか?
32. Windows 版 UT-VPN でも Windows NT カーネルと Windows 9x カーネルで API が異なるため、内部的に動的に切り分けを行っていますが、そのためにどのような手法でそれを実現していますか?
33. MsShutdown 関数と MsShutdownEx 関数の違いは何ですか?
34. Windows で仮想 LAN カードをシステムにインストールする際、テンプレートとなる inf ファイルの内容を動的に書き換えて、それをシステムに渡しています。その際にどのような手法を用いているか解説してください。
35. Windows 2000 以降で仮想 LAN カードをシステムにインストールする際に「ドライバが署名されていません」という警告ダイアログが出ないような工夫が行われていますが、その方法について詳しく解説してください。
36. GetNetBiosName 関数は非常に特殊な処理を行っていますが、どのような仕組みを使ってこの方法で指定した IP アドレスのコンピュータ名を取得しているのでしょうか?
37. GetHostName 関数は IP アドレスからコンピュータ名を取得する関数ですが、なぜか内部でスレッドを用いています。内部でスレッドを用いている理由を推測してください。
38. connect_timeout 関数は connect システムコールでタイムアウトがサポートされていないシステムでもタイムアウトを発生させますが、どのような仕組みでそれを実現していますか?
39. Mayaqua ライブラリによって SSL 通信か通常のソケット通信かに関わらず常に呼び出し元コードは Send または Recv を呼び出せば通信が可能になっており大変便利ですが、SSL 通信を行う場合

は必ず呼び出さなければならない関数があります。その関数は何ですか?

40. SSL 通信中に突然暗号化せずにデータを送受信したくなった場合はどのようにすべきでしょうか?
41. Send と SendAll 関数の違い、Recv と RecvAll 関数の違いは何ですか?
42. Select による SOCKSET に登録されたソケットの待機と、WaitSockEvent による SOCK_EVENT に登録されたソケットの待機の 2 つは良く似ていますが、システム的 (特に Win32 上) では内部実装は異なっています。どのように実装が異なっているか解説してください。また SOCKSET が使用されている代表的な場面と、SOCK_EVENT が使用されている代表的な場面について解説してください。
43. CANCEL オブジェクトは、Win32 と UNIX ではそれぞれどのような方法で実現されていますか?
44. EVENT オブジェクトは通常 Object.c の関数を経由してアクセスします。ただし、Win32 用に限って、少々乱暴ではあるが、直接 EVENT オブジェクトのメンバにアクセスしているようなコードが Cedar 内にいくつかあります。そのようなものを探してきて、なぜその必要があったのかを推測してください。
45. Pack.c は何のために存在しますか?
46. Secure.c は PKCS#11 (仕様書については google で検索して入手してください) のラッパーですが、なぜこのようなラッパーが必要になっているのでしょうか?
47. 文字列テーブル strtable.stb ファイルは、初回起動時には hamcore 内から読み込まれますが、2 回目以降はキャッシュされたファイルから読み込まれます。なぜ高速化のためにキャッシュが必要になるのかについて解説してください。
48. ソースコードのどこか一部を改変し、Malloc などを呼び出した後そのメモリを解放せずに ham.exe を終了してみて、メモリリークが検出されることを確認してください。次に、ham.exe に何かのパラメータ (コマンドライン引数) を渡して起動すると、実際にメモリリークが発生している箇所とメモリ確保の際のコールスタックが表示されますが、実際にそれが可能であることを試してみてください。最後に、どのような手段でこれらのメモリリーク追跡のためのデバッグ機能を実現しているかを解説してください。
49. UNIX 上で utvpnserver や utvpnclient などのデーモン化可能プロセスを実行 (utvpnserver start など) すると、内部的にはどのような動作が行われてデーモン化するのかを解説してください。
50. ファイルから X.509 証明書を読み込むために使用する関数は何ですか?
51. メモリ上のバイト列から X.509 証明書を読み込むために使用する関数は何ですか?
52. OpenSSL はスレッドセーフですが、事前にスレッド処理のためのコールバック関数を指定する必要があります。それはどこで指定していますか?
53. OpenSSL の SHA 関数と SHA1 関数の違いは何ですか?
54. CloneX 関数や CloneK 関数は何のために使用していますか?
55. なぜ K とか X というような名前のオブジェクト (構造体) が存在する必要があるのですか?
56. Win32 サービスモードで動作するコードを書く場合、サービスを終了 (クリーンアップ) する場合は、もしクリーンアップ中にデッドロックなどが発生してしまった場合でも Windows のシャットダウンがいつまでたっても終了しないというような事態を避けるため、ある安全機能が含まれています。その機能について解説してください。
57. Windows 9x にはサービスモードとしてプログラムを動作させる方法がないため、Microsoft.c では

別の方法であたかもサービスとして Windows 9x 上でプログラムが動作しているかのように見せかけています。それはどのような方法でしょうか?

58. UT-VPN のソースコード中では、決してスレッドを別のスレッドから強制終了するような危険なコードはありません。なぜそのような行為が危険であるのかを解説してください。また、原則として UT-VPN ソースコード中ではあるスレッドの仕事を中断させるために別のスレッドがどのような方法で終了指示 (通知) を行っているかを調べて、解説してください。

Cedar 通信コードに関する質問

1. NewCedar 関数で渡す server_x と server_k は何ですか?
2. EnableDebugLog 関数を呼び出すと何が起きますか? 詳細に解説してください。
3. InitNetSvcList 関数は何のために存在しますか?
4. Cedar が直接保持する Connection リストに含まれているコネクションにはどのようなものがありますか?
5. VPN Client の Cedar には Connection リスト内に 1 つ以上のエントリが含まれる可能性はありますか?
6. Listener が存在する理由は何故ですか? 必要な都度、Accept を用いるコードを書けば良いのではないでしょうか?
7. Listener.c の 154 行目付近について、なぜ新しいスレッドを作成してそのスレッドでこのような処理を実行しようとせずに、Accept したスレッドでこのような処理を実行しているのでしょうか?
8. utvpnserver/utvpnserver.c 内の StartProcess 関数は、InitCedar、StInit および StStartServer の 3 つの関数を順番に呼び出すだけの大変単純なものですが、これらによって実際は大変複雑な VPN Server の初期化処理が背後で行われ、VPN Server が起動します。ここで行われているような処理内容をできるだけ詳しく解説してください。
9. VPN セッションは、最初は 1 本の TCP/IP コネクションで構成されますが、接続設定の CLIENT_OPTION で指定されている条件によっては後ほど別の TCP/IP コネクションを追加していき、複数本の TCP/IP コネクションで通信を行うようになっています。そこで既存のセッションに対して新しい TCP/IP コネクションのソケットを動的に追加するためにどのようなプログラムになっているか解説してください。
10. VPN セッションに対する 2 本目以降の TCP/IP コネクションを受け取った VPN Server はそれがどの VPN セッションに対する追加コネクションであるのかをどのようにして識別するか説明してください。
11. パケットアダプタ (PACKET_ADAPTER) とは何で、何のためにこのようなインターフェイスがあるのかを解説してください。
12. VPN Server / VPN Client のいずれにおいても CANCEL オブジェクトは大変重要です。その理由は何でしょうか?
13. なぜ SESSION と CONNECTION という 2 つのオブジェクトが存在しているのでしょうか?
14. 仮想 HUB 間リンク (カスケード接続) 機能はどのような仕組みで実装されているか、できるだけ詳しく解説してください。
15. ローカルブリッジ機能はどのような仕組みで実装されているか、できるだけ詳しく解説してくだ

さい。

16. 仮想レイヤ 3 スイッチ機能はどのような仕組みで実装されているか、できるだけ詳しく解説してください。
17. SecureNAT 機能はどのような仕組みで実装されているか、できるだけ詳しく解説してください。
18. ClonePacket 関数は何のために存在し、どのような場面で使用されていますか?
19. Cedar、Session、Connection、User、Group などのオブジェクト間では相互に参照し合っている場合が多くありますが、どのようにして循環参照による参照カウンタがいつまでも解消されない問題 (オブジェクトリークの発生) を回避していますか?
20. PrivacyFilter セキュリティポリシーはどのように実装されていますか?
21. NoServer セキュリティポリシーはどのように実装されていますか?
22. DHCPNoServer セキュリティポリシーはどのように実装されていますか?
23. DHCPForce セキュリティポリシーはどのように実装されていますか?
24. NoBridge セキュリティポリシーはどのように実装されていますか?
25. NoRouting セキュリティポリシーはどのように実装されていますか?
26. モニタリングモードセッション (MonitorPort セキュリティポリシーが有効なセッション) は仮想 HUB 内を流れるすべてのフレームを受信できますがどのような仕組みでそれが実装されていますか? また、モニタリングモードセッションのクライアント側から仮想 HUB に対してパケットが送信されたら、そのパケットはどのように扱われますか?
27. DECLARE_RPC_EX マクロ、DECLARE_RPC マクロ、DECLARE_SC_EX マクロ、DECLARE_SC マクロ、CHECK_RIGHT マクロは何のために存在し、どのような場面で利用されていますか?
28. CAPSLIST は何のために存在し、どのような場面で利用されていますか?
29. GetEthDeviceHash 関数は何のために存在していますか?
30. Linux ではどのような手段でローカルブリッジ機能のための Ethernet アダプタへの直接アクセスを実現していますか?
31. Windows 版クライアント接続マネージャは VPN Client サービスで何かの状態変化が発生すると、即座にそれを受け取り、変更をユーザーインターフェイスに反映させます。それはどのような仕組みで実施されていますか?
32. Windows 版のユーザーインターフェイス機構はまず PenCore.dll ファイルを hamcore から読み出し、次にそれを何らかの方法で LoadLibrary してリソースを抜き出しています。それはどのような方法でしょうか?
33. Windows 版のユーザーインターフェイス機構のうち、ダイアログボックス (ウインドウ) やメニューについてはリソースデータ内には一切日本語文字列を含まず、strtable.stb から動的に文字列を読み込んで表示しています。①その利点は何ですか? ②実際には WinUI.c コードのどの部分でそのような処理が行われていますか?
34. Windows 版クライアント接続マネージャでは、どのような仕組みで接続設定のインポート / エクスポート機能を実現していますか?
35. 最もソースコード行数が長いファイルは Command.c です。このファイルは一体何をしているファイルでしょうか?
36. Connection.c ソースコード中で、最もモジュール間の機能分割がうまく進んでおらずソースコー

ドの拡張性が低下している部分があります。その部分はどこでしょうか? また解決策にはどのような手法がありますか?

37. ConnectionSend 関数と ConnectionReceive 関数の役割について詳しく解説してください。
38. VPN セッションを構成する各 TCP/IP コネクションはある程度の間隔ごとにキープアライブパケットが送受信されます。その「ある程度の間隔」はどのようにして決められていますか? また、もしキープアライブまたは VPN 通信が設定されたタイムアウト値以上の間途切れたらどのような処理が発生してそのコネクションは破棄されますか?
39. 仮想 HUB には複数の VPN セッションが接続されますが、ある VPN セッション側から Ethernet フレームが届いた場合、その Ethernet フレームを仮想 HUB がどのような手順で正当な宛先の VPN セッションにスイッチング (ストア & フォワード) し、そのパケットの物理的な転送処理が開始されるのかについて詳しく解説してください。
40. 仮想 HUB が送信元セッションおよび宛先セッションにおけるセキュリティポリシーを強制する手段およびタイミングについて詳しく解説してください。
41. 仮想レイヤ 3 スイッチ機能で、ある仮想インターフェイスから入っていた IP パケットのルーティング先がその仮想インターフェイスであった場合は、どのような処理を行って送信元仮想インターフェイスにパケットを差し戻しますか?
42. クラスタコントローラがクラスタメンバに対して要求 (Task) を渡す場合、そのタスクを生成したスレッドが直接ソケットを用いてクラスタメンバに対して要求を送信するのではなく、別のワーカースレッドが常にそのスレッドに対して投入されてきたタスクを処理するようなモデルでプログラミングされています。その処理は具体的にどのように行われていますか?
43. クラスタコントローラ内の仮想 HUB に対してセッション一覧を要求すると、そのクラスタ全体のセッション一覧が返ってきます。その処理はどのような仕組みで行われていますか?
44. クラスタメンバがクラスタコントローラに対して管理接続しようとしてうまく接続できない場合や、管理接続が切れてしまった場合はどのようにして回復を試みますか?
45. スタンドアロンモードの VPN Server 上で、オンライン状態であった仮想 HUB がオフラインになった場合、VPN Server はオフライン化の過程でどのような作業を行いますか? 具体的かつ詳細に解説してください。
46. クラスタコントローラの VPN Server 上で、オンライン状態であった仮想 HUB がオフラインになった場合、VPN Server はオフライン化の過程でどのような作業を行いますか? 具体的かつ詳細に解説してください。
47. 仮想 NAT 機能で、仮想 HUB 内の TCP クライアントが仮想 NAT サーバーを経由して外部の TCP サーバーに接続を試みようとした瞬間から、実際に TCP/IP 通信が行われ、また最後に TCP/IP コネクションが切断されるまで、NAT 構造体のコンテキストが生成され、それが状態機械として状態遷移を行いつつ適切なパケット処理を行います。これは大変複雑なプロセスですが、できるだけ詳しく仮想 NAT 機能の動作方法について解説してください。これは大変難解な問題であり、Virtual.c および TcpIp.c の読解を必要とします。
48. SecureNAT 機能で仮想サーバーに割り当てられている IP アドレスに対して MTU (1500 バイト) を超える大きなサイズの ICMP を投げつけても、正しくすべてのペイロードが返ってきます。これは何故でしょうか? また仮想レイヤ 3 スイッチ機能の仮想インターフェイスに割り当てられてい

る IP アドレスに対して約 1500 バイト以上の ICMP を投げつけても、今度は先頭のあるバイト数しか返ってきません。この違いはなぜ発生するのでしょうか?