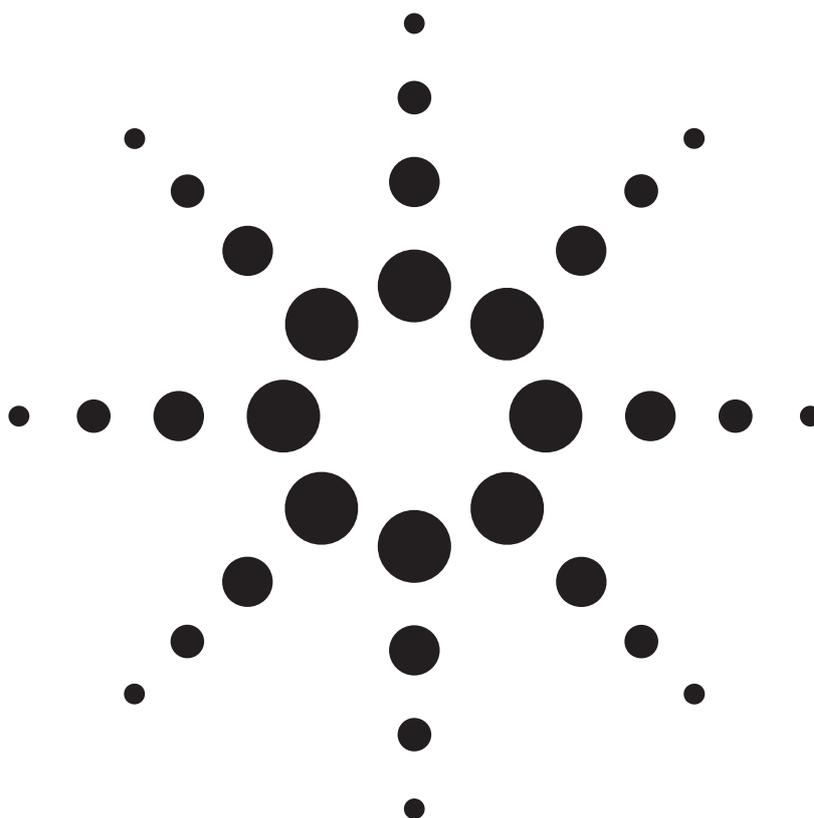


Linux を使用した USB 測定器の制御

Application Note AN 1465-30



Agilent Open では、I/O インタフェースとして、PC 標準の I/O インタフェースを採用しています。これにより、ハードウェア、I/O、ソフトウェア・ツールを柔軟に組み合わせてシステムの構築、拡張、保守が可能になります。たとえば、OS として Linux を使用している場合、LAN や USB インタフェースを有効活用できます。

本アプリケーション・ノートでは、Linux 環境でテスト機器を制御する方法を解説しています。

目次

概要：USBTCM 測定器と USB ベース測定器	2
基本的な USB 用語	2
USBTCM 測定器との通信	3
USB コアへの登録	5
ユーザ空間からドライバへの アクセス	6
USBTCM ドライバのコンパイルと インストール	8
USBTCM ドライバの使用	9
まとめ	10



Agilent Technologies

概要： USBTCM 測定器と USB ベースの測定器

Agilent は、テスト・システム用測定器のインタフェースを簡素化するために Agilent Open プログラムにより、PC インタフェース(特に LAN と USB) への移行を提唱しています。最新の Agilent 測定器の多くは、イーサネット、USB、GPIB をサポートしています。

USB のサポートは、USB チップセットを制御する低レベル USB ドライバ(カーネル・モジュール)という形で現在の Linux カーネルに組み込まれています(図1を参照)。しかしこれらのドライバには、ユーザ(ユーザ空間で動作するアプリケーション)に対する低レベル・プログラミング・インタフェースがありません。ドライバは通常、(カーネル空間で)ポインティング・デバイス(マウスなど)、USB ディスク・ドライブなどの特定のタイプの USB デバイスをサポートする他のカーネル・モジュールから呼び出されます。ほとんどの場合、USB デバイスを利用するには、対応するデバイス・クラスをサポートするカーネル・ドライバが必要です。

電子計測器も例外ではありません。Agilent をはじめとする多くの測定器ベンダが、USB-IF と連携して、ベンダに依存しない USB ベースの測定器の規格として、USBTCM(USB Test and Measurement Class) 規格を策定し、2002 年に公表しました。現在市販されている USB 測定器(特に Agilent 製品)のほとんどが、USBTCM 規格に準拠しています。

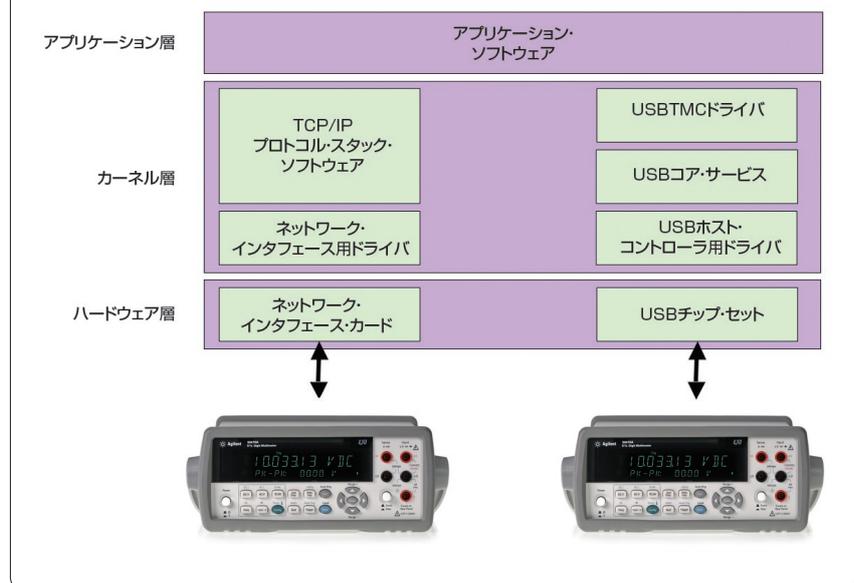
このアプリケーション・ノートでは、USB 測定器を制御するために、USBTCM カーネル・モジュールを作成する方法について説明します。以下で説明するドライバ用のサンプル・ソース・コードは、各ディストリビューションやカーネル・バージョンでのコンパイルのために、Agilent の Web サイト <http://www.agilent.co.jp/find/linux> からダウンロードすることができます。ソース・コードの作成とテストは、openSUSE 10.2 に基づいて行われていますが、最新ディストリビューションでもほとんど変更せずに動作します。

基本的な USB の用語

USB は、独自の方法を使って帯域幅を共有し、バス上で通信を論理的に構築します。USB で最も重要な概念は、エンドポイントという概念です。1つの物理デバイスは通常、複数の論理エンドポイントを使用して、それぞれがデータを送信する(入力エンドポイント)か、データを受信します(出力エンドポイント)。エンドポイントには、方向のほか、タイプの違いもあります。

制御エンドポイントは、デバイスの設定と初期セットアップに使用されます。バルク・エンドポイントは、大量のデータをデバイスとやりとりする際に使用されます。例えば、USBTCM デバイスは、バルクアウト・エンドポイントを使用して SCPI コマンドを受け取り、バルクイン・エンドポイントを使用して測定結果を転送します。

図 1. イーサネットと比較した USBTCM ドライバの構造



割り込みエンドポイントは、時間的な制約のある少量データの転送（例えば、USB マウスの移動）への対応に使用されます。同様に、アイソクロナス・エンドポイントは、大量のデータ用（例えば、音声／ビデオ・アプリケーションのストリーミング）に帯域幅を連続して予約するために使用されます。

インタフェースとは、デバイスの特定のサブ機能を提供するためにグループ化されたエンドポイントの集まりです。例えば、USBTMC デバイスは、セットアップには制御エンドポイントを、通常の通信にはバルクイン／バルクアウト・エンドポイントを使用します。これらのエンドポイントだけで USBTMC デバイスを制御することができ、これらのエンドポイントが1つの（論理）USB インタフェースの一部となります。音声出力／入力用の USB サウンド・カードなど複数のインタフェースを使用する USB デバイスもあります。

USBTMC では通常用いられませんが、1つのインタフェースに対して異なる設定（構成）を複数行うことができます。例えば、同一デバイスに対して異なる物理 USB インタフェース速度または異なるアプリケーション・タイプを設定できます。

USB に関するもう1つの重要な概念が、ユニバーサル・リクエスト・ブロック (URB) です。URB は、デバイス・クラス・ドライバが USB コア・ドライバに USB デバイスとのデータ通信を指示するために使用するデータ構造です。URB には、転送するデータと、リクエストを正しく処理するために必要なすべてのアドレス指定情報が含まれています。

USBTMC 測定器との通信

USBTMC デバイスとの通信のほとんどは、SCPI コマンドの送信と問い合わせコマンドに対する測定器の応答の読み取りです。この動作のしくみを、例を使って説明します。1 個の SCPI コマンドを送信し、通信の詳細を調べます。

USBTMC 規格では、測定器コマンドを送信するために、DEV_DEP_MSG_OUT メッセージを定義しています。このメッセージは、測定器のバルクアウト・エンドポイントに送信されます。メッセージには、SCPI コマンド自体のほか、多くのフィールドが含まれています(表1を参照)。

表 1. USBTMC DEV_DEP_MSG_OUT メッセージの構造 (*RST コマンドの例)

オフセット (バイト)	フィールド	サイズ (バイト)	値	概要
0	MsgID	1	1	DEV_DEP_MSG_OUT メッセージ (コマンド文字列を測定器へ送信するために使用)
1	bTag	1	x	転送識別子。この ID は、転送ごとに増分され、メッセージの紛失を測定器が検出できます。
2	bTagInverse	1	~x	bTag (転送識別子) の1の補数
3	Reserved	1	0x00	予約
4~7	TransferSize	4	5	転送するバイトの数 (測定器コマンド)
8	bmTransferAttributes	1	0x01	メッセージの終了。ビット0が1に設定されている場合は、測定器メッセージはこの転送で終了です。それ以外の場合は、次の転送でメッセージの続きが送信されます。その他のビットはすべて反転されず (0 に設定されます)。
9~11	Reserved	3	0x000000	予約。0x000000 に設定されます
12~16	Instrument Command	5	""RST\n"	測定器コマンド

Linux 用の VISA IO ライブラリ

Agilent が提供する MS Windows 環境用の IO ライブラリ・スイートに精通しているユーザであれば、Linux の VISA プログラミングにも興味をお持ちのはずです。VISA は、測定器制御用のマルチベンダ規格です。TAMS (www.tamsinc.com) から入手可能な Red Hat Linux 用の VISA/SICL ライブラリは、ほとんどのインタフェース・タイプ (GPIB、USB、LXI、VXI、GPIO、RS-232) に対応しています。VISA を Windows 環境と Linux 環境の両方で使用する場合は、VISA/SICL ライブラリから互換性のあるコマンド・セットが得られます。このように、Linux オペレーティング・システムの内蔵機能を使用するか (このアプリケーション・ノートで説明)、より精通したコマンド・セットを提供する VISA ライブラリをロードするかの選択が可能です。詳細については、www.tamsinc.com をご覧ください。製品番号は 82091 です。

現在使用可能な VISA インプリメンテーションはオープン・ソースではないため、他の Linux ディストリビューションにトランスポートできません。そのため、VISA インプリメンテーションが選択肢として適切ではありません。

SCPI コマンドを測定器に送信するために、USBTCM ドライバは、コマンドを以下に示すようにメッセージ構造でラップし、USB コア・ドライバにメッセージを処理するように指示します (すなわち、メッセージをデバイスのバルクアウト・エンドポイントに送信します)。図 2 に、対応するコードのサンプルを示します。

以下に示すコードに関していくつかの注意すべき点があります。関数 `copy_from_user()` は、ユーザ空間からカーネル・メモリにデータをコピーするカーネル関数です。この関数は `memcpy()` とは異なり、ページングの問題 (メモリでのページの消失) に対応しています。

コードの次の数行は、必要に応じてアライメント・バイトを追加するためのものです。USBTCM 規格に従って、メッセージ内のバイトの総数は、4 の倍数でなければなりません。

図 2. サンプル・コード：DEV_DEP_MSG_OUT メッセージ経由で SCPI コマンドを送信する方法

```
// DEV_DEP_MSG_OUT メッセージ用の IO バッファをセットアップ
usb_tmc_buffer[0]=1; // DEV_DEP_MSG_OUT
usb_tmc_buffer[1]=bTag; // 転送 ID (bTag)
usb_tmc_buffer[2]=~(bTag); // bTag の反転
usb_tmc_buffer[3]=0; // 予約
usb_tmc_buffer[4]=command_length&255; // 転送サイズ (1 番目のバイト)
usb_tmc_buffer[5]=(command_length>>8)&255; // 転送サイズ (2 番目のバイト)
usb_tmc_buffer[6]=(command_length>>16)&255; // 転送サイズ (3 番目のバイト)
usb_tmc_buffer[7]=(command_length>>24)&255; // 転送サイズ (4 番目のバイト)
usb_tmc_buffer[8]=1; // この転送でメッセージが終了
usb_tmc_buffer[9]=0; // 予約
usb_tmc_buffer[10]=0; // 予約
usb_tmc_buffer[11]=0; // 予約

// 書き込みバッファ (測定器コマンド) を USBTCM メッセージにアペンド
if(copy_from_user(&(usb_tmc_buffer[12]),command_buffer,command_length)) {
    // アドレス指定問題が発生
    return -EFAULT;
}

// 4 バイト・アライメントを実現するために 0 バイトを追加
n_bytes=12+command_length;
if(command_length%4) {
    n_bytes+=4-command_length%4;
    for(n=12+command_length;n<n_bytes;n++) usb_tmc_buffer[n]=0;
}

// バルク出力転送用のパイプを作成
pipe=usb_sndbulkpipe(usb_dev,bulk_out);

// バルク URB を送信
retval=usb_bulk_msg(usb_dev,pipe,usb_tmc_buffer,n_bytes,
    &actual,USBTCM_USB_TIMEOUT);
```

関数 `usb_sndbulk_pipe()` は、使用するエンドポイントの情報を作成し、`usb_bulk_msg()` がカーネルにメッセージの処理を依頼します。後の2つの関数は、USB コア層によって提供されるサービスの一部です。

測定器からのデータの読み取りも同様に動作します。最初に、`DEV_DEP_MSG_IN` メッセージがバルクアウト・エンドポイントに送信され、次の読み取りトランザクションでデータの送信を測定器に依頼します。次に、データが測定器のバルクイン・エンドポイントから読み取られます。詳細については、USBTMC 規格を参照し、このアプリケーション・ノートに付属のサンプル・コードを調べてください。

USB コアへの登録

図1に示す USB コアは、単に上位レベル・ドライバの代わりに USB メッセージを処理するだけではなく、接続された USB デバイスとインストールされたさまざまな上位レイヤ・サービス間の動作を支援します。USB デバイスのホットプラグの管理も行います。

USB コアと情報をやりとりする場合（特にデバイスを接続していて、デバイスを識別中であることを通知する場合）は、上位レベル・ドライバを USB コアに登録する必要があります。

この登録プロセスの重要な要素は、デバイスが使用可能になったときに上位レベル・ドライバのサービスの対象がどのデバイスであるかを USB コアに通知することです。必要なデバイスは、デバイスのベンダ ID、製品 ID、デバイス・クラスなど、さまざまな属性によりフィルタリングできます。USBTMC のコンテキストでは、デバイス・クラス（アプリケーション固有）と USBTMC サブクラスによるフィルタリングが最適です。これにより、USBTMC ドライバはベンダや製品コードに関係なく、USBTMC 互換デバイスが接続されているときに通知を受けとります。

図3に、このアプリケーション・ノートに付属のサンプル・ドライバを USB コアに登録する方法を示します。

図3：サンプル・コード：USB 上位レベル・ドライバを USB コア層に登録する方法

```
// このリストで、このドライバのサービスの対象となるデバイスを定義します。このドライバは
// USBTMC デバイスを処理するので、対応するクラス（アプリケーション
// 固有）とサブクラス（USBTMC）を探します
static struct usb_device_id usbtmc_devices[] = {
    {.match_flags=USB_DEVICE_ID_MATCH_INT_CLASS |
     USB_DEVICE_ID_MATCH_INT_SUBCLASS,
     // システムが通知を行うためには、デバイス・クラスとサブクラスが一致している必要があります
     .bInterfaceClass=254, // 254 = アプリケーション固有
     .bInterfaceSubClass=3}, // 3 = 電子計測クラス（USBTMC）
    { } // エントリの終了
};

// この構造体にはドライバの登録情報が含まれています
// 情報は、ドライバの init 関数で呼び出された
// usb_register() 経由でシステムに渡されます
static struct usb_driver usbtmc_driver;
// この構造体を使用して、この USB ドライバに関する情報を
// USB コアに（usb_register 経由で）渡します
static struct usb_driver usbtmc_driver = {
    .name="USBTMC", // ドライバ名
    .id_table=usbtmc_devices, // ドライバのサービス対象のデバイス
    .probe=usbtmc_probe, // Probe 関数（デバイスが接続されたときに呼び出されます）
    .disconnect=usbtmc_disconnect // Disconnect 関数
};
// USB ドライバを USB コアに登録します
if((retcode=usb_register(&usbtmc_driver))) {
    printk(KERN_ALERT "USBTMC: Unable to register driver\n");
    goto exit_usb_register;
}
```

ここでも、コードに関していくつかの注意すべき点があります。最初のセクションでは、どのタイプのデバイスに興味があるのかを USB コアに通知する構造体のリストを作成します。この例では、リストに USBTMC デバイス用の 1 個のエントリがあります。

次に、タイプ `usb_driver` の構造体を定義します。構造体は、USB コア層が上位レベル・ドライバを登録するために必要な情報を保持しています。構造体には、前述のフィルタに対するポインタのほか、`probe()` 関数と `disconnect()` 関数のアドレスが含まれています。

`probe()` は、より上位のレイヤのドライバに新しく接続されたデバイスを通知するために USB コアから呼び出されます。ドライバは `probe()` により、メモリの割り当て、内部データ構造の初期化、新しいデバイスへのサービスの準備が可能になります。

`disconnect()` は、デバイスがもう使用できなくなったことをドライバに通知するために呼び出されます。ドライバは通常、内部データ構造をクリーンアップし、`probe()` 関数の実行中に割り当てたメモリやその他のリソースを開放します。

ユーザ空間からドライバへのアクセス

USBTMC 互換測定器は通常（必ずではない）、SCPI 規格に従ってテキスト・コマンドを用いて制御されます。同様に、測定結果やその他のデータは、人間が読めるテキストとして返されます。すなわち、USB 測定器との通信は、ストリーム指向で、テキスト・ファイルに対する読み書きと類似しています。こうしたテキスト・ベースのデバイスでは、ユーザ空間へのアクセス方法としてキャラクタ・デバイス・ドライバがよく使用されます。

キャラクタ・デバイス・ドライバの利点は、通常のテキスト・ファイルのように動作する点です。したがって、デバイスとのデータのやりとりは、標準のファイル I/O システム・コールを使用することができます。同様に、コンソール・アプリケーションの出力をデバイスにリダイレクトできます。このようにキャラクタ・デバイス・ドライバにより高い柔軟性が得られます。

キャラクタ・デバイス・ドライバは、ドライバの背後のデバイスと情報をやりとりするためにシステムが呼び出す多くのエントリ・ポイントを実装する必要があります。最も基本的

なものは `open()`、`read()`、`write()`、`release()` で、それぞれシステム・コール `open(2)`、`read(2)`、`write(2)`、`close(2)` に対応します。

USBTMC のコンテキストでは、`write()` エントリ・ポイントは、書き込む文字列を取り込み、それを USBTMC `DEV_DEP_MSG_OUT` メッセージでラップします。同様に、`read()` エントリ・ポイントは、`DEV_DEP_MSG_IN` メッセージを使用して、デバイスからのデータの読み取り、戻りデータからの測定器メッセージ部分の抽出、供給されたユーザ・バッファへの抽出データのコピーを行います。

デバイス・ドライバに関する重要な概念が、メジャー番号とマイナー番号です。デバイス・ファイルは `mknod(1)` コマンドを使用して作成され、指定されたメジャー番号は（任意の）デバイス・ファイル名の後にあるキャラクタ・ドライバを表します。マイナー番号は通常、同一ドライバにサービス対象のデバイスが複数ある場合は、ドライバが制御するデバイスを指定するために使用されます。

キャラクタ・デバイス・ドライバをカーネルにロードする場合は、最初にそのメジャー番号とマイナー番号をカーネルに登録し、エントリ・ポイントを公開する必要があります。図4に、このアプリケーション・ノートに付属のサンプル・ドライバの対応する行を示します。

以下に示すコードの最初のセクションでは、ドライバとともに使用するために、空いているメジャー番号とある範囲のマイナー番号を動的に割り当てます。

ドライバがファイル I/O に対して公開するさまざまなエントリ・ポイントのアドレスを保持するために、タ

イプ `file_operations` の構造体が初期化されます。次のコードで新しいキャラクタ・ドライバを記述する `cdev` 構造体を割り当てて、最後に `cdev_add()` 関数を使用して新しいドライバをアクティブにします。ドライバはこの時点で、先に公開したエントリ・ポイントをいつでも呼び出すことができます。

図4: サンプル・コード：キャラクタ・デバイス・ドライバの登録方法

```
// キャラクタ・ドライバのメジャー／マイナー番号を動的に割り当てます
if((retcode=alloc_chrdev_region(&dev, // 使用する最初のメジャー／マイナー番号
0, // 最初のマイナー番号
USBTMC_MINOR_NUMBERS, // 予約するマイナー番号の数
"USBTMCCHR" // キャラクタ・デバイス・ドライバ名
))) {
    printk(KERN_ALERT "USBTMC: Unable to allocate major/minor numbers\n");
    goto exit_alloc_chrdev_region;
}

// この構造体は、キャラクタ・デバイス・ドライバの関数を公開するために使用されます
static struct file_operations fops = {
    .owner=THIS_MODULE,
    .read=usbtmc_read,
    .write=usbtmc_write,
    .open=usbtmc_open,
    .release=usbtmc_release,
    .ioctl=usbtmc_ioctl,
    .llseek=usbtmc_llseek,
};

// このキャラクタ・デバイスの cdev 構造体を初期化します
cdev_init(&cdev,&fops);
cdev.owner=THIS_MODULE;
cdev.ops=&fops;

// メジャー番号とマイナー番号を結合します
printk(KERN_NOTICE "USBTMC: MKDEV\n");
devno=MKDEV(MAJOR(dev),n);

// キャラクタ・デバイスをカーネル・リストに追加します
printk(KERN_NOTICE "USBTMC: CDEV_ADD\n");
if((retcode=cdev_add(&cdev,devno,1))) {
    printk(KERN_ALERT "USBTMC: Unable to add character device\n");
    goto exit_cdev_add;
}
```

USBTMC ドライバのコンパイルとインストール

このアプリケーション・ノートに付属のサンプル・ドライバは、<http://www.agilent.co.jp/find/linux> から、TAR アーカイブとして入手できます。アーカイブを適切な(空の)ディレクトリにコピーし、コマンド `tar -x an1465-30.tar` を使用して解凍します。

解凍されたファイルには、ソース・ファイルと `makefile` が含まれています。`make(1)` コマンドを使用してドライバをコンパイルします。これにより、`usbttmc.ko` ファイル(カーネル・オブジェクト・ファイル)が作成されます。ドライバをコンパイルするには、システムにカーネル・ソース・ツリーがインストールされている必要があります。これは通常、ディストリビューションのメディアから入手できますが、デフォルトでインストールされていない場合がよくあります("kernelsource" という名前のパッケージを探してください)。

この時点で、コマンド `insmod ./usbttmc.ko` を使用して動作中のカーネルにドライバ・モジュールをインストールできます。同様に、`rmmod usbttmc` を使用してモジュールをカーネルからアンロードできます。これらのコマンドを実行するには `root` 権限が必要です。

ドライバを使用するには、最初に適切なデバイス・ファイルを作成します。作成するには、ドライバが使用するメジャー番号を知っている必要があります(メジャー番号はドライバをインストールする際、すなわち `insmod` を実行する際に、初期化ルーチンに動的に割り当てられます)。この情報を取得する一番簡単な方法は、コマンド `cat /proc/devices | grep USBTMCCHR` を使

用して `/proc/devices` を読み取る方法です。

上記コマンドにより返されたメジャー番号を使って、`mknod/dev/usbttmc0 c 253 0` (または同様のコマンド)を使用してデバイス・ファイルを作成できます。ここで 253 は、ドライバにより割り当てられたメジャー番号です。最後に、`chmod(1)`

を使用して読み取り/書き込みビットを適切に設定します。

ドライバには、上記のステップを自動化する `usbttmc_load` という名前のシェル・スクリプトが付属しています。これを図 5 に示します。

図 5: モジュール・ロード・スクリプト

```
#!/bin/sh

module="usbttmc"

# カーネルからモジュールを削除 (動作中である可能性があるため)
/sbin/rmmod $module

# モジュールをインストール
/sbin/insmod ./module.ko

# 使用するメジャー番号を検出
major=$(cat /proc/devices | grep USBTMCCHR | awk '{print $1}')
echo Using major number $major

# 古いデバイス・ファイルを削除
rm -f /dev/${module}[0-9]

# 新しいデバイス・ファイルを作成
mknod /dev/${module}0 c $major 0
mknod /dev/${module}1 c $major 1
mknod /dev/${module}2 c $major 2
mknod /dev/${module}3 c $major 3
mknod /dev/${module}4 c $major 4
mknod /dev/${module}5 c $major 5
mknod /dev/${module}6 c $major 6
mknod /dev/${module}7 c $major 7
mknod /dev/${module}8 c $major 8
mknod /dev/${module}9 c $major 9

# アクセス・モードを変更 (RW アクセス)
chmod 666 /dev/${module}0
chmod 666 /dev/${module}1
chmod 666 /dev/${module}2
chmod 666 /dev/${module}3
chmod 666 /dev/${module}4
chmod 666 /dev/${module}5
chmod 666 /dev/${module}6
chmod 666 /dev/${module}7
chmod 666 /dev/${module}8
chmod 666 /dev/${module}9
```

USBTMC ドライバの使用

サンプル USBTMC ドライバは、接続された各 USBTMC デバイスに次の空いている（使用されていない）マイナー番号を動的に発行します。USB コアが、この順番で新しい USB デバイスの存在をドライバに通知します。測定器と通信するには、デバイスが使用しているマイナー番号を知る必要があります。サンプル USBTMC ドライバでは、その情報は、マイナー番号 0 から読み取ることにより得られます。すなわち、マ

イナー番号 0 は、USBTMC ドライバ自体との通信のために予約されています。

USB デバイスを接続した後（または測定器がすでに接続されたシステムをブートした後）、`cat/dev/usbttmc0` を使用してデバイスのリストを読み取ることができます。これにより、各デバイスの製品番号、メーカー ID、シリアル番号、マイナー番号が返ってきます。

コマンド文字列をそのデバイス・ファイルにリダイレクトすることにより、SCPI コマンドをデバイスに送信できます。例えば、`echo *RST>/dev/usbttmc1` を使用して最初の USBTMC デバイスをリセットできます。

同様に、`cat` を使用して USBTMC デバイスから読み取ります。例えば、`echo *IDN?>/dev/usbttmc1`、と次の `cat/dev/usbttmc1` は、デバイスの ID 文字列を出力します（図 6 を参照）。

図 6：echo と cat を使用した対話型の測定器制御

```
skopp@A0071584:~/Projects/usbtmc/src> make
make -C /lib/modules/2.6.18.2-34-default/build SUBDIRS=/home/skopp/Projects/usbtmc/src modules
make[1]: Entering directory `/usr/src/linux-2.6.18.2-34-obj/i386/default'
make -C ../../../../linux-2.6.18.2-34 O=../../linux-2.6.18.2-34-obj/i386/default modules
  CC [M] /home/skopp/Projects/usbtmc/src/usbtmc.o
  Building modules, stage 2.
  MODPOST
  LD [M] /home/skopp/Projects/usbtmc/src/usbtmc.ko
make[1]: Leaving directory `/usr/src/linux-2.6.18.2-34-obj/i386/default'
skopp@A0071584:~/Projects/usbtmc/src> su
Password:
A0071584:/home/skopp/Projects/usbtmc/src # ./usbtmc_load
ERROR: Module usbtmc does not exist in /proc/modules
Using major number 253
A0071584:/home/skopp/Projects/usbtmc/src # cat /dev/usbttmc0
Minor Number    Manufacturer    Product Serial Number
001      Agilent Technologies    34980A Switch Measure Unit    MY44003719
A0071584:/home/skopp/Projects/usbtmc/src # echo *RST>/dev/usbttmc1
A0071584:/home/skopp/Projects/usbtmc/src # echo *IDN?>/dev/usbttmc1
A0071584:/home/skopp/Projects/usbtmc/src # cat /dev/usbttmc1
Agilent Technologies,34980A,MY44003719,2.19-2.19-2.07-1.05
```

テスト・アプリケーションでは、ファイル IO システム・コールを使用して適切な SCPI コマンド文字列をデバイスに送信します（または応答をリードバックします）。図 7 に、基本的な例を示します。

まとめ

現在使用可能な USB 測定デバイスのほとんどは、USBTMC 規格に準拠しています。これらのデバイスを使用するには、USBTMC デバイス・ドライバが必要です。このアプリケーション・ノートでは、現在の Linux ディストリビューションおよびバージョンと一緒に使用できるジェネリック・ドライバを作成するための方法について説明しています。ドライバはキャラクタ・デバイス・ドライバとして実装され、ファイル I/O の場合は、出力のリダイレクトと簡単なシステム・コールによる測定器アクセスが可能です。

図 7：ファイル IO システム・コールを使用したプログラムによる測定器制御

```
#include <stdio.h>
#include <fcntl.h>

main()
{

    int myfile;
    char buffer[4000];
    int actual;
    myfile=open("/dev/usbtmc1",O_RDWR);
    if(myfile>0)
    {

        write(myfile,"*IDN?\n",6);
        actual=read(myfile,buffer,4000);
        buffer[actual]=0;
        printf("Response:\n%s\n",buffer);
        close(myfile);
    }
}
```

¹ Linux Device Drivers, Jonathan Corbet/
Alessandro Rubini/Greg Kroah-Hartman, O'
REILLY

² USB Test and Measurement Class
Specifications, USB Implementers Forum,
http://www.usb.org/developers/devclass_docs#approved

Agilent の関連カタログ

1465 シリーズ・アプリケーション・ノートでは、テスト・システムの構築、テスト・システムで有効に LAN/WLAN/USB を使用する方法、RF/マイクロ波テスト・システムの最適化と拡張についての豊富な情報を提供しています。

テスト・システム開発

- 『システム開発者ガイド:テスト・システムでの LAN の使用:基礎』 AN 1465-9 (カタログ番号 5989-1412JA) <http://cp.literature.agilent.com/litweb/pdf/5989-1412JA.pdf>
- 『テスト・システムでの LAN の使用:ネットワークの設定』 AN 1465-10 (カタログ番号 5989-1413JA) <http://cp.literature.agilent.com/litweb/pdf/5989-1413JA.pdf>
- 『システム開発ガイド テスト・システムでの LAN の使用:PC の設定』 AN 1465-11 (カタログ番号 5989-1415JA) <http://cp.literature.agilent.com/litweb/pdf/5989-1415JA.pdf>
- 『システム開発ガイド 計測環境での USB 使用』 AN 1465-12 (カタログ番号 5989-1417JA) <http://cp.literature.agilent.com/litweb/pdf/5989-1417JA.pdf>
- 『システム開発ガイド SCPI + ダイレクト I/O、ドライバの使用法』 AN 1465-13 (カタログ番号 5989-1414JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-1414JAJP.pdf>

- 『システム開発ガイド テスト・システムにおける LAN の使用法:アプリケーション』 AN 1465-14 (カタログ番号 5989-1416JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-1416JAJP.pdf>
- 『システム開発ガイド テスト・システムでの LAN の使用:システム I/O のセットアップ』 AN 1465-15 (カタログ番号 5989-2409JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-2409JAJP.pdf>

- 『LXI による次世代テスト・システム』 AN 1465-16 (カタログ番号 5989-2802JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-2802JAJP.pdf>

RF/マイクロ波テスト・システム

- 『RF/マイクロ波テスト・システムの構成要素の最適化』 AN 1465-17 (カタログ番号 5989-3321JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-3321JAJP.pdf>
- 『RF/マイクロ波テストシステムのテスト品質向上のための 6 ヒント』 AN 1465-18 (カタログ番号 5989-3322JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-3322JAJP.pdf>
- 『システムの信号経路の校正:ベクトルおよびスカラー補正法による測定精度の向上』 AN 1465-19 (カタログ番号 5989-3323JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-3323JAJP.pdf>

LXI (LAN eXtensions for Instrumentation)

- 『次世代 LXI テスト・システム』 AN 1465-20 (カタログ番号 5989-4371JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-4371JAJP.pdf>
- 『LXI に移行する 10 の理由』 AN 1465-21 (カタログ番号 5989-4372JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-4372JAJP.pdf>
- 『GPIB から LXI への移行』 AN 1465-22 (カタログ番号 5989-4373JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-4373JAJP.pdf>

- 『PXI、VXI、LXI によるハイブリッド・テスト・システムの構築』 AN 1465-23 (カタログ番号 5989-4374JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-4374JAJP.pdf>

- 『テスト・システムにおけるシンセティック測定器の使用法:利点とトレードオフ』 AN 1465-24 (カタログ番号 5989-4375JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-4375JAJP.pdf>

- 『GPIB から LXI への移行 (システム・ソフトウェア編)』 AN 1465-25 (カタログ番号 5989-4376JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-4376JAJP.pdf>

- 『LAN/LXI を組み込むための GPIB システムの変更』 AN 1465-26 (カタログ番号 5989-6824JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-6824JAJP.pdf>

テスト・システムでの Linux の使用

サンプル・コードは、<http://www.agilent.co.jp/find/linux> からダウンロードできます。

- 『Linux を使用したテスト・システム:Linux の基礎』 AN 1465-27 (カタログ番号 5989-6715JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-6715JAJP.pdf>
- 『Linux を使用した LXI 測定器の制御:VXI-11 の使用』 AN 1465-28 (カタログ番号 5989-6716JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-6716JAJP.pdf>
- 『Linux を使用した LXI 測定器の制御:TCP の使用』 AN 1465-29 (カタログ番号 5989-6717JAJP) <http://cp.literature.agilent.com/litweb/pdf/5989-6717JAJP.pdf>

www.agilent.co.jp/find/open



電子計測UPDATE

www.agilent.co.jp/find/emailupdates-Japan
Agilentからの最新情報を記載した電子メールを無料でお送りします。



Agilent Direct

www.agilent.co.jp/find/agilentdirect
測定器ソリューションを迅速に選択して、使用できます。



www.agilent.co.jp/find/open
Agilentは、テスト・システムの接続とプログラミングのプロセスを簡素化することにより、電子製品の設計、検証、製造に携わるエンジニアを支援します。Agilentの広範囲のシステム対応測定器、オープン・インダストリ・ソフトウェア、PC標準I/O、ワールドワイドのサポートは、テスト・システムの開発を加速します。



www.lxistandard.org
LXIIは、GPIBのLANベースの後継インタフェースで、さらに高速かつ効率的なコネクティビティを提供します。Agilentは、LXIコンソーシアムの設立メンバーです。

Remove all doubt

アジレント・テクノロジーでは、柔軟性の高い高品質な校正サービスと、お客様のニーズに応じた修理サービスを提供することで、お使いの測定機器を最高標準に保つお手伝いをしています。お預かりした機器をお約束どおりのパフォーマンスにすることはもちろん、そのサービスをお約束した期日までに確実にお届けします。熟練した技術者、最新の校正試験プログラム、自動化された故障診断、純正部品によるサポートなど、アジレント・テクノロジーの校正・修理サービスは、いつも安心して信頼できる測定結果をお客様に提供します。

また、お客様それぞれの技術的なご要望やビジネスのご要望に応じて、

- アプリケーション・サポート
- システム・インテグレーション
- 導入時のスタート・アップ・サービス
- 教育サービス

など、専門的なテストおよび測定サービスも提供しております。

世界各地の経験豊富なアジレント・テクノロジーのエンジニアが、お客様の生産性の向上、設備投資の回収率の最大化、測定器のメインテナンスをサポートいたします。詳しくは：

www.agilent.co.jp/find/removealldoubt

アジレント・テクノロジー株式会社
本社 〒192-8510 東京都八王子市高倉町 9-1

計測お客様窓口

受付時間 9:00-19:00 (土・日・祭日を除く)

FAX、E-mail、Webは24時間受け付けています。

TEL ■■■ 0120-421-345
(042-656-7832)

FAX ■■■ 0120-421-678
(042-656-7840)

Email contact_japan@agilent.com

電子計測ホームページ
www.agilent.co.jp

● 記載事項は変更になる場合があります。
ご発注の際はご確認ください。

Copyright 2008
アジレント・テクノロジー株式会社



Agilent Technologies

April 9, 2008
5989-6718JAJP
0000-00DEP