# .NET Coreによる Raspberry Pi プログラミング

川口直也◉著

CUTT
カットシステム

■サンプルファイルのダウンロードについて

本書掲載のサンプルファイルは、一部を除いてインターネット上のダウンロードサービスからダウンロードすることができます。詳しい手順については、本書の巻末にある袋とじの内容をご覧ください。

なお、ダウンロードサービスのご利用にはユーザー登録と袋とじ内に記されている番号が必要です。そのため、本書を中古書店から購入されたり、他者から貸与、譲渡された場合にはサービスをご利用いただけないことがあります。あらかじめご承知おきください。

・本書の内容についてのご意見、ご質問は、お名前、ご連絡先を明記のうえ、小社出版部宛文書（郵送またはE-mail）でお送りください。
・電話によるお問い合わせはお受けできません。
・本書の解説範囲を越える内容のご質問や、本書の内容と無関係なご質問にはお答えできません。
・匿名のフリーメールアドレスからのお問い合わせには返信しかねます。

本書で取り上げられているシステム名／製品名は、一般に開発各社の登録商標／商品名です。本書では、™ および ® マークは明記していません。本書に掲載されている団体／商品に対して、その商標権を侵害する意図は一切ありません。本書で紹介しているURLや各サイトの内容は変更される場合があります。

# はじめに

本書は .NET Core で Raspberry Pi 用のアプリケーションを開発するための解説書です。.NET Core は .NET のサブセットの開発・実行環境であり、ARM/Linux 環境のような様々なプラットフォーム上で .NET アプリケーションを実行することができます。また、Raspberry Pi は今や言わずと知れた世界的に有名な組み込み開発のプラットフォームです。本書では Raspberry Pi 用のアプリケーションを開発するうえで必要な各種センサの制御や、ネットワーク機能、RFID リーダーの制御といった実践的なテクニックを解説しています。

本書が、C# で Web アプリケーションや業務アプリケーションを開発しているエンジニアの皆さんにとって、Raspberry Pi を用いた電子工作の世界に踏み出すきっかけになれば幸いです。

**献辞**

本書の出版にあたり、私にこの本を執筆する機会を与えてくださったカットシステムの石塚さんに御礼申し上げます。また、本書の編集を手掛けてくださった同社編集部の皆さんに感謝いたします。

最後に、私が休日も机に向かって本書の執筆に取り組むことを快く承諾してくれた妻に感謝します。

平成 31 年春　桜花の舞う公園を眺めながら

川口　直也

## ■本書の方針

筆者は本書の執筆にあたり下記の方針を立てました。

- 最小限の環境で Raspberry Pi の開発を行うこと。
  - ・ディスプレイ不要、キーボード不要
- 開発言語は基本的に C# とすること
  - ・ネイティブコードのラッパーを用意して開発
- GPIO や I2C といった I/O インターフェースの制御について解説すること
- Raw ソケットや ioctl といった Linux 固有の API を用いた開発を実践すること

このうち、最小限の環境で Raspberry Pi の開発を行う方針は、Raspberry Pi 用のディスプレイやキーボードを用意しなくても開発ができることを意味します。Raspberry Pi はとても安価なハードウェアで、5000 円程度で入手できますが、ディスプレイやキーボードを別途用意すると結構な出費になるはずです。しかし、Raspberry Pi でやりたいことが GPIO の制御といった低レベルのプログラミングである場合、単にセットアップのためだけにディスプレイなどを用意するのはもったいないでしょう。本書では dd コマンドと SSH だけで操作できるよう解説しています。

また、C# を開発言語として選定したのは、.NET のような超高級言語の世界と Raspberry Pi の低レベル I/O の世界を結びつける目的からです。しかしながら、.NET の API だけで Raspberry Pi を操作することは残念ながらできませんでした。筆者は、この問題をネイティブコードを呼び出して制御することで回避しました。したがって、本書で紹介するコードは純粋な .NET のコードではないことに留意してください。

最後に、本書は Raspberry Pi のハードウェアの話題だけではなく、Raw ソケットや USB 接続の Ferica リーダーといったアプリケーション寄りの話題についても解説しています。こうした技術を組み合わせることで、例えば特定のパケットを検出した際に LED を点灯させたり、SUICA と I2C 液晶を組み合わせた出退勤のタイムカードシステムを構築したりすることが可能になるでしょう。

## ■ .NET Core について

.NET Core は .NET Framework のサブセットです。Raspberry Pi を含む ARM/Linux 環境のような、Windows 以外の環境で .NET プログラムを実行することができます。

.NET Core は .NET Framework のサブセットのため、プログラムを実装する際は専用のプロジェクトとして開発する必要があります。.NET Core プロジェクトでは Windows フォームのような GUI が使用できなかったり、導入できる Nuget パッケージに制限があったりします。

.NET Core を用いることで、従来 Windows 環境に閉じていた C# プログラムの開発・運用環境を Linux や MacOS のような様々な環境に広げることができます。

### ■ Mono との違いについて

Mono は .NET Framework のオープンソース実装です。Mono は主に Linux 環境で動作します。Mono は .NET Framework のフルセットの実装を目指しているため、Windows フォームのような通常の .NET プログラムが動作します。

実は、Raspberry Pi（Raspbian）においても Mono は動作します。したがって、本書に掲載している全てのプログラムが、実は .NET Core のランタイムではなく、Mono を用いても実行することができます。

それでは、Mono ではなく、.NET Core のランタイムを使用する優位性は何なのでしょうか。筆者は、.NET Core ランタイムが Microsoft がオフィシャルに提供している開発・実行環境である点にあると考えています。

### ■ .NET MF との違いについて

.NET MF（.NET Micro Framework）は、.NET Core よりもさらに限られたリソースで動作する .NET の処理系です。

.NET MF の特徴はターゲットのデバイスに特化した処理系を、ユーザがある程度カスタマイズできる点にあります。例えば、ターゲットのデバイスが提供するハードウェアリソースに合わせて .NET MF の処理系をカスタマイズして配備します。これにより、上位層の .NET プログラムからハードウェアへのアクセス機能を提供することができます。

本書で扱う Raspberry Pi は .NET MF が対象とするハードウェアよりも、もっとパワフルで高機能なため、より高水準の .NET Core を用いて開発することができます。

## ■Raspberry Piについて

Raspberry Piは、ラズベリーパイ財団が開発した、主にホビーユースのシングルボードコンピュータです。Raspberry PiのプロセッサはARMであり、主に使用されるOSはLinux（Raspbian）です。日本では5000円前後の低廉な価格で販売されており、根強い人気があります。

本書執筆時点（2019年3月）の最新ハードウェアは、Raspberry Pi 3 Model B+です。本書は「Raspberry Pi 3 Model B」を対象に執筆していますが、上位版の「Raspberry Pi 3 Model B+」でもほとんどプログラムが動作します。

付録のWindows IoT CoreはRaspberry Pi Model 3B+では動作しませんでした。Model 3Bを使用してください。

## ■Raspberry Pi 3 Model Bについて

Raspberry Pi 3 Model Bは、2016年に発売されたRaspberry Piデバイスです。CPUに「ARM Cortex-A53」の1.2 GHzを搭載し、メインメモリも1 GB搭載、無線LANと有線LANのI/Fを搭載し、IoTデバイスとしては非常にリッチな環境を提供しています。

なお、Raspberry PiにはModel B以外にModel Aのボードも存在します。Model Aも基本的なハードウェア構成は同じですが、ボードのサイズが異なります。

## ■ 本書が想定する動作環境

本書の内容は下記の動作環境に基づいています。

| | |
|---|---|
| ハードウェア | Raspberry PI model 3B |
| OS | Raspbian Stretch with desktop and recommended software（Linux raspberrypi 4.14.71-v7+） |
| ホストマシン | CPU : x64<br>OS : Windows 10<br>SSH クライアント : TeraTerm<br>開発環境 : Visual Studio 2016<br>mDNS : Bonjour または iTunes |

# 目次

はじめに ..... iii

## 第1章 初期設定 ..... 1

1.1 ホストマシンの設定 ..... 1
- 1.1.1 MSYS2 の導入 ..... 1
- 1.1.2 開発環境の整備 ..... 2
- 1.1.3 SSH クライアントの導入 ..... 4
- 1.1.4 Bonjour による mDNS の利用 ..... 4

1.2 Raspberry PI の初期設定 ..... 5
- 1.2.1 OS イメージのダウンロード ..... 5
- 1.2.2 OS イメージの書き込み ..... 5
- 1.2.3 SSH サーバーの起動設定 ..... 9

1.3 Raspberry Pi の起動後の設定 ..... 10
- 1.3.1 Raspberry Pi の起動 ..... 10
- 1.3.2 SSH で Raspberry PI に接続 ..... 10
- 1.3.3 Raspberry Pi の IP アドレスを調べる ..... 12
- 1.3.4 samba の導入 ..... 14
- 1.3.5 VNC サーバー ..... 16
- 1.3.6 RDP サーバーの導入 ..... 23
- 1.3.7 Raspberry Pi への .NET Core の導入 ..... 25

## 第2章 .NET Core によるデバイス制御プログラミング ..... 31

2.1 オンボード LED の点灯 ..... 31
- 2.1.1 コマンドによる LED 点灯 ..... 31
- 2.1.2 C# プログラムによるオンボード LED 点灯 ..... 33

2.2 GPIO による LED の点灯 ..... 35
- 2.2.1 GPIO による LED 点灯プログラム ..... 35
- 2.2.2 配線 ..... 39
- 2.2.3 プログラムの実行 ..... 39

2.3 サーボモーター制御 ..... 40
- 2.3.1 サーボモーター制御の概要 ..... 40
- 2.3.2 ハードウェア PWM ..... 41

2.3.3 Wiring Pi について ..... 41
2.3.4 PWM 制御の解説 ..... 42
2.3.5 PWM 制御の実装 ..... 44
2.3.6 配線 ..... 46
2.3.7 実行 ..... 46
2.4 DC モーター制御 ..... 47
2.4.1 配線 ..... 47
2.4.2 コマンドラインによる配線のテスト ..... 49
2.4.3 プログラムによる DC モーター制御 ..... 50
2.4.4 PWM による回転数制御 ..... 53
2.5 ステッピングモーター制御 ..... 56
2.5.1 ステッピングモーターの配線 ..... 57
2.5.2 プログラムの実装 ..... 58
2.5.3 プログラムの実行 ..... 61
2.6 エンコーダ付き DC モーター ..... 61
2.6.1 配線 ..... 62
2.6.2 プログラムの実装 ..... 63
2.6.3 プログラムの実行 ..... 66
2.7 ソフトウェア PWM ..... 67
2.7.1 Wiring Pi によるソフトウェア PWM ..... 67
2.7.2 LED 点灯プログラム ..... 67
2.7.3 配線 ..... 69
2.7.4 プログラムの実行 ..... 70
2.8 タクトスイッチによる割込み制御 ..... 71
2.8.1 タクトスイッチの配線 ..... 71
2.8.2 プログラムの実装 ..... 72
2.8.3 プログラムの実行 ..... 74
2.9 LCD キャラクタディスプレイ ..... 75
2.9.1 I2C とは ..... 75
2.9.2 I2C LCD について ..... 76
2.9.3 プログラムの実装 ..... 76
2.9.4 配線作業 ..... 83
2.9.5 プログラムの実行 ..... 84
2.10 RTC（リアルタイムクロック） ..... 85
2.10.1 秋月電子製 RTC-8564NB のキットについて ..... 86
2.10.2 配線 ..... 87
2.10.3 コマンドによる RTC 制御 ..... 88
2.10.4 C# から RTC 制御 ..... 90
2.11 GPS による位置情報の読み取り ..... 93
2.11.1 GPS モジュールの接続 ..... 93
2.11.2 シリアルポートの設定 ..... 94

2.11.3　コマンドラインからの動作確認 ..... 96
2.11.4　プログラムの実装 ..... 97
2.11.5　プログラムの実行 ..... 100

## 第 3 章　ASP.NET Core による Web サービスプログラミング ..... 103

3.1　Web サービスの開発環境の構築 ..... 103
3.2　ソリューションの作成 ..... 104
3.3　ピン情報クラスの実装 ..... 106
3.4　サービスクラスの実装 ..... 110
3.5　Web サービスクラスの実装 ..... 113
3.6　デバッグ実行時の設定 ..... 114
3.7　プロジェクトの実行 ..... 115
3.8　リモートホストからのアクセス ..... 116

## 第 4 章　ネットワークプログラミング ..... 119

4.1　レイヤ 2 フレームキャプチャ ..... 119
4.1.1　プログラムの実装 ..... 119
4.1.2　プログラムの実行 ..... 126

## 第 5 章　他のマイコンボードとの連携 ..... 127

5.1　MicroBit との連携 ..... 127
5.1.1　MicroBit の接続 ..... 127
5.1.2　REPL スケッチの書き込み ..... 128
5.1.3　REPL スケッチのテスト実行 ..... 130
5.1.4　.NET Core から MicroBit 制御 ..... 132
5.2　Arduino との連携 ..... 139
5.2.1　Arduino とは ..... 139
5.2.2　Arduino 制御について ..... 140
5.2.3　事前準備 ..... 140
5.2.4　Firmata ファームウェアの書き込み ..... 141
5.2.5　C# プログラムの実装 ..... 142

## 第 6 章 Arduino 経由での Grove デバイスの制御 ..... 149

6.1 arduino-cli の導入 ..... 151
6.2 Grove-4-Digit Display（TM1637）の制御 ..... 154
6.2.1 Grove-4-Digit Display Library の追加 ..... 155
6.2.2 Arduino スケッチの作成 ..... 156
6.2.3 Arduino スケッチの転送 ..... 159
6.2.4 Arduino スケッチのテスト実行 ..... 159
6.2.5 C# プログラムの実装 ..... 160
6.3 Grove - LED Bar の制御 ..... 164
6.3.1 arduino-cli にライブラリを追加 ..... 164
6.3.2 Arduino スケッチの作成 ..... 165
6.3.3 Arduino デバイスへスケッチを転送 ..... 167
6.3.4 スケッチのテスト実行 ..... 167
6.3.5 C# プログラムの実行 ..... 168
6.4 Grove 温度センサ ..... 169
6.4.1 Arduino スケッチの実装 ..... 169
6.4.2 Arduino スケッチの転送 ..... 171
6.4.3 スケッチのテスト ..... 171
6.4.4 C# プログラムの実装 ..... 172

## 第 7 章 USB デバイス ..... 175

7.1 RFID リーダー ..... 175
7.1.1 RFID について ..... 175
7.1.2 初期設定 ..... 176
7.1.3 プログラムの実装 ..... 177
7.1.4 プログラムの実行 ..... 181
7.2 Web カメラ ..... 183
7.2.1 Web カメラの接続 ..... 183
7.2.2 コマンドラインから画像を取得 ..... 184
7.2.3 ライブラリの設定 ..... 185
7.2.4 プログラムの実装 ..... 189

## 付録 A dotnet コマンド ..... 191

A.1 dotnet コマンドのオプション（「dotnet --help」の出力結果を整形） ..... 191
A.2 特に重要なオプション ..... 193

付録 B Mono ..... 201

B.1 Mono について ..... 201
B.2 Mono のインストール ..... 202
B.2.1 apt-get による Mono のインストール ..... 203
B.3 オンボード LED 点灯プログラム（.NET Core） ..... 204
B.4 GPIO 操作 GUI ツール（.NET Framework） ..... 204

付録 C Windows 10 IoT Core ..... 209

C.1 Windows 10 IoT Core のインストール ..... 209
C.1.1 Windows 10 IoT Core Dashboard のインストール ..... 209
C.1.2 OS イメージの書き込み ..... 211
C.2 Windows 10 IoT Core の操作 ..... 213
C.2.1 PowerShell ..... 215
C.2.2 デバイスポータル ..... 216
C.2.3 ファイル共有 ..... 218
C.3 プログラムの実行 ..... 219
C.3.1 PowerShell とファイル共有によるリモート実行 ..... 221
C.3.2 Windows Device Portal の Run Command ページでの実行 ..... 224
C.4 ハードウェアの制御 ..... 225
C.4.1 .NET Core IoT Libraries ..... 225

索引 ..... 229

# 第1章

# 初期設定

ここではホストマシンと Raspberry Pi の初期設定について解説します。「はじめに」で述べたとおり、筆者は本書の執筆にあたり、Raspberry Pi 用のディスプレイやキーボードを用意しなくても開発ができることを志向しました。ここでは実際に、ディスプレイやキーボードの用意がなくても Raspberry Pi を設定できるよう解説していきます。

## 1.1 ホストマシンの設定

ホストマシン（開発マシン）の設定について解説します。

### 1.1.1 MSYS2 の導入

MSYS2 は Windows 上で UNIX のコマンドを使用できるようにするツールチェインです。本書では、後述する dd コマンドにより OS イメージを書き込むために使用します。

MSYS2 のインストールは次の Web サイトからインストーラを入手して実行してください。

- MSYS2 の Web サイト（MSYS2 homepage）
  https://www.msys2.org/

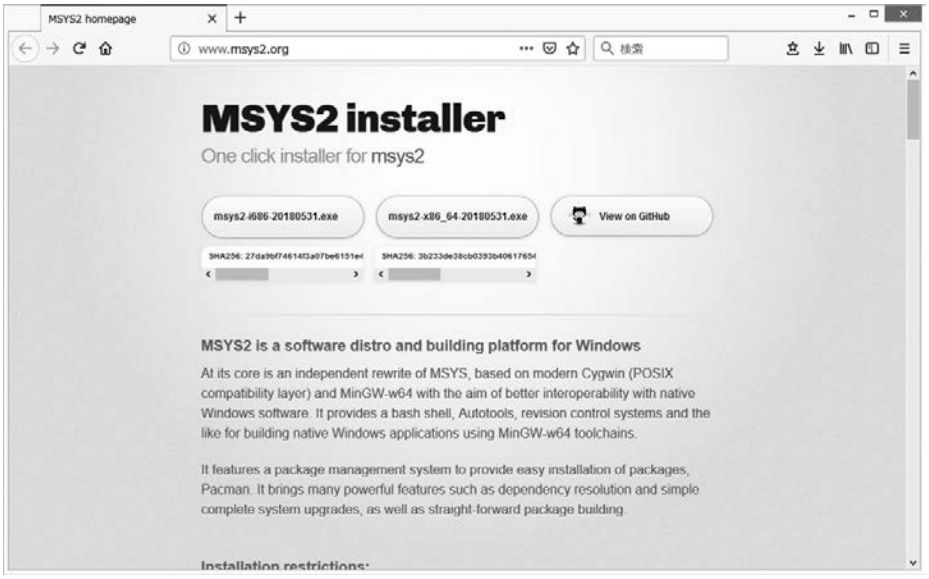


**図2.1●MSYS2のWebサイト**

インストールが完了すると、デスクトップ上に msys2.exe へのショートカットが生成されています。

## 1.1.2 開発環境の整備

本書では、Visual Studio 2017 を対象に解説を行います。ここでは、無償版の Visual Studio 2017 Community Edition を対象に解説を行います。

Visual Studio 2017 Community Edition は下記のサイトからインストーラを実行してください。

- Visual Studio のダウンロードサイト（Visual Studio IDE）
  https://visualstudio.microsoft.com/ja/downloads/

インストールの途中で次の選択画面が表示されます。本書で扱う .NET Core の開発環境を構

築するために、「ユニバーサル Windows プラットフォーム開発」を選択してインストールしてください。

**図2.2●インストールの選択画面**

また、すでに Visual Studio がインストール済みの場合は、Visual Studio のメニューから、「ツールと機能を取得」を選択して機能を追加してください。

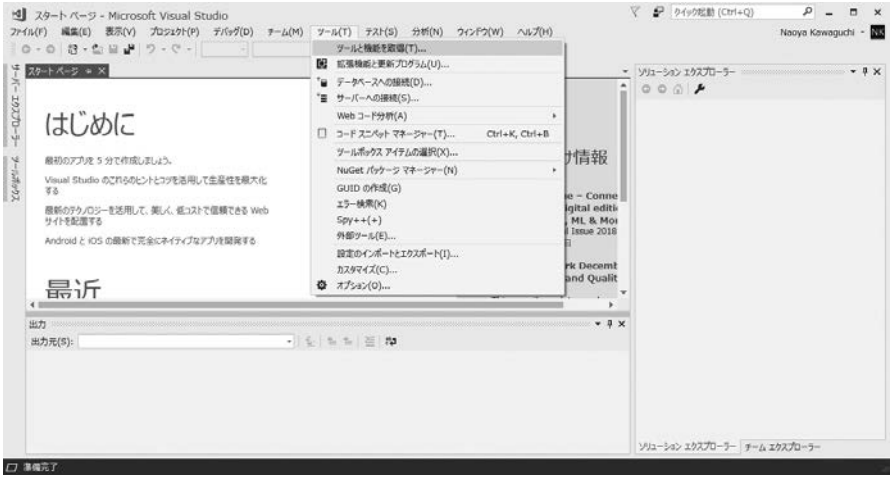


**図2.3●「ツールと機能を取得」メニュー項目**

## 1.1.3 SSH クライアントの導入

Raspberry Pi をリモート操作するために、SSH クライアントを導入します。筆者は TeraTerm を使用しました。TeraTerm は窓の杜などのサイトで配布されているので、任意のバージョンを使用してください。

## 1.1.4 Bonjour による mDNS の利用

Raspberry Pi は mDNS により、DNS の設定を行うことなく、xxx.local（xxx はホスト名）といったホスト名でアクセスすることができます。この mDNS は通常、ホストマシンに iTunes などをインストールすることで利用することができます。

ここでは、iTunes をインストールせず、同じく Apple が無償で配布している Bonjour Print Services for Windows をインストールすることで、mDNS を利用する方法について解説します。

### Bonjour Print Services for Windows のダウンロード

下記のサイトから、Bonjour をダウンロードします。

- ダウンロード - Bonjour Print Services（Windows）
  https://support.apple.com/kb/DL999?viewlocale=ja_JP&locale=en_US

上記のサイトから、BonjourPSSetup.exe ファイルをダウンロードします。

### Bonjour のインストール

ダウンロードした BonjourPSSetup.exe ファイルをダブルクリックしてインストールを実行します。

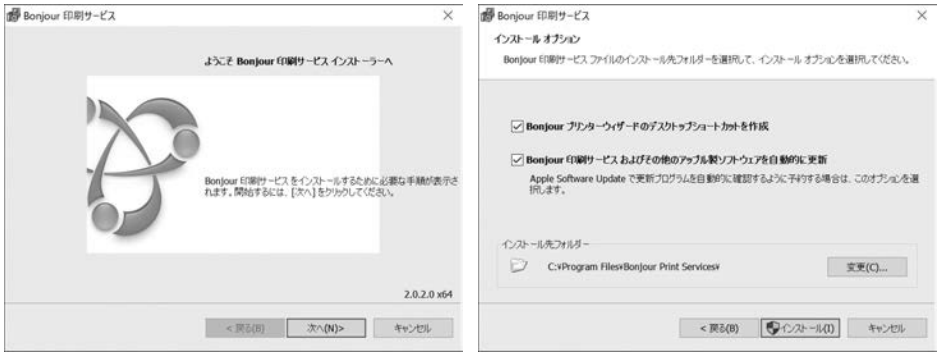


**図2.4●Bonjourのインストール画面**

# 1.2 Raspberry PI の初期設定

ここでは Raspberry Pi の初期設定について解説します。

## 1.2.1　OS イメージのダウンロード

Raspbian の OS イメージは下記のサイトから入手できます。筆者は本書の執筆にあたり「Raspbian Stretch with desktop and recommended software」の 2018/11/13 リリース版を使用しました。

- Raspbian の配布サイト（Download Raspbian for Raspberry Pi）
  https://www.raspberrypi.org/downloads/raspbian/

## 1.2.2　OS イメージの書き込み

本書では OS イメージの書き込みに、msys2 の dd コマンドを使用します。なお、WSL（Windows Subsystem for Linux）の dd コマンドは使用できませんのでご注意ください。

まずは、SD カードを PC に接続し、ドライブ名を確認します。次図の例では E ドライブと

して認識されています。

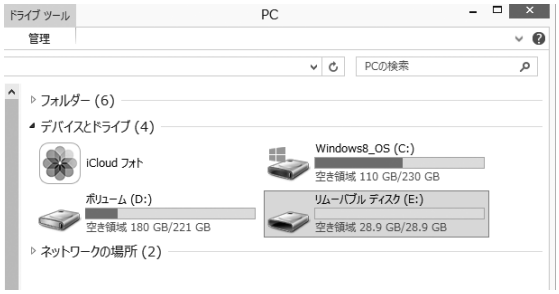


**図2.5●SDカードのドライブ名を確認**

次に、msys2 のコンソールを管理者権限で起動します。msys2 のショートカットを右クリックして、「管理者として実行」を選択してください。

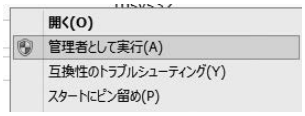


**図2.6●msys2のコンソールを管理者権限で起動**

なお、msys2.exe はデフォルトインストールの場合は下記のディレクトリに存在します。

C:\msys32\msys2.exe

下記のコマンドでパーティションの一覧を確認します。

```
$ cat /proc/partitions
```

## ■ 結果

```
$ cat /proc/partitions
major minor  #blocks  name   win-mounts

    8     0 488384512 sda
```

```
8     1   1024000 sda1
8     2    266240 sda2
8     3   1024000 sda3
8     4    131072 sda4
8     5 241426432 sda5   C:\
8     6    466944 sda6
8     7 232508416 sda7   D:\
8     8  11534336 sda8
8    16  30375936 sdb
8    17  30371840 sdb1   E:\
```

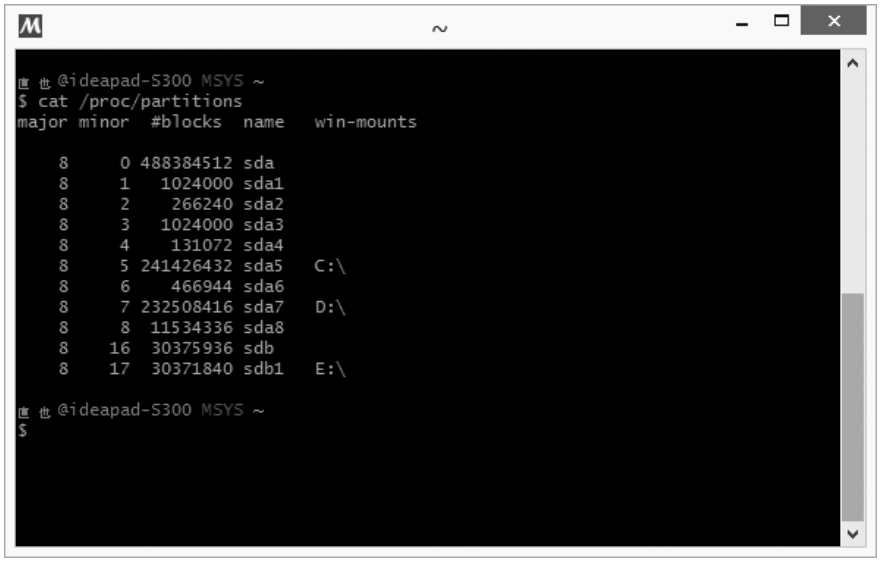


**図2.7●パーティション一覧を確認**

先ほどの E ドライブが、sdb デバイスとして認識されていることがわかります。

次の dd コマンドの実行を間違えると、ホストマシンの Windows システムが壊れたり、データが破損する可能性があるので注意してください。特に、of パラメータで指定するデバイス名は絶対に正しい値にする必要があります。

## ■ dd コマンドのパラメータ

```
dd bs=65536 if=<OSイメージファイル(.img)> of=/dev/<デバイス名>
```

bs　ブロックサイズ
if　入力ファイル（デバイス）
of　出力ファイル（デバイス）

最近の環境ではブロックサイズに 1m などを指定できますが、msys2 付属の dd コマンドでは 1m は使用できません。65536（64KB）を指定してください。

## ■ dd コマンドの実行例

```
# dd bs=65536 if=2018-10-09-raspbian-stretch.img of=/dev/sdb

63104+0 レコード入力
63104+0 レコード出力
4135583744 bytes (4.1 GB, 3.9 GiB) copied, 330.299 s, 12.5 MB/s
```

筆者の環境では、上記のコマンドの実行に約 10 分ほどかかりました。

書き込みが終わってから、Windows 上で SD カードをアンマウントし、再度マウントすると、次図のように boot パーティションが表示されます。

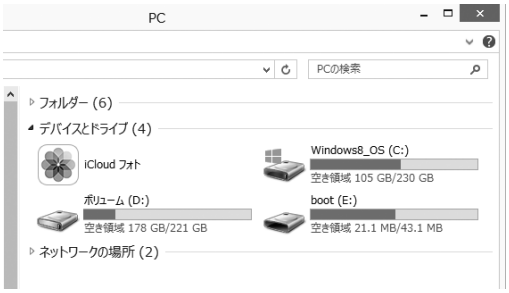


**図2.8●SDカードを再マウントするとbootパーティションが表示される**

## 1.2.3　SSH サーバーの起動設定

Raspberry Pi が起動する際に自動で SSH サーバーが起動するように設定を行います。SSH サーバーを自動起動するには、boot パーティション上に ssh ファイルを作成します。

Windows 上で boot パーティションを開き、ファイル名「ssh」の空ファイルを作成してください。

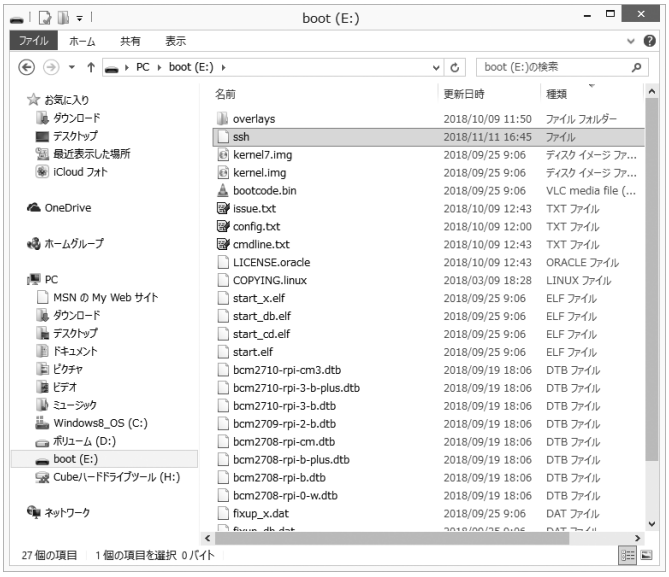


**図2.9●bootパーティション上にsshファイルを作成する**

以上で、SSH サーバーの起動設定は完了です。

# 1.3 Raspberry Pi の起動後の設定

ここからは、Raspberry Pi を起動して設定を行います。

## 1.3.1 Raspberry Pi の起動

まずは、RaspberryPi に先ほどデータを書き込んだ SD カードを挿入してください。次に、PC と Raspberry Pi をイーサネットで接続してください。なお、無線 LAN の I/F は Raspberry Pi 側の SSID 設定が完了していませんので使用できません。最後に、Raspberry Pi の Micro USB ポートに電源を供給し、Raspberry Pi を起動してください。

なお、最初の起動では Raspberry Pi の初期化が行われるため、後述する SSH 接続などができるまで時間がかかります。30 分ほどは様子を見た方がよいでしょう。

## 1.3.2 SSH で Raspberry PI に接続

TeraTerm を起動して下記のように Raspberry Pi に接続します。接続先のホスト名は raspberrypi.local を指定します。

なお、DNS の設定をしていないのに、raspberrypi.local で接続できるのは mDNS という機能のおかげです。

図2.10●TeraTermで接続する