uClinux-dist
開発者ガイド

Version 1.4.3-8d87fa8
2009/07/17

株式会社アットマークテクノ [http://www.atmark-techno.com]

SUZAKU 公式サイト [http://suzaku.atmark-techno.com]

uClinux-dist
開発者ガイド

株式会社アットマークテクノ

060-0035 札幌市中央区北 5 条東 2 丁目 AFT ビル 6F
TEL 011-207-6550 FAX 011-207-6570

Version 1.4.3-8d87fa8
2009/07/17

目次

1. はじめに 9
1.1. マニュアルについて 9
1.2. フォントについて 9
1.3. コマンド入力例の表記について 9
1.4. 謝辞 10
1.5. ソフトウェアに関する注意事項 10
2. uClinux-dist について 11
3. デフォルトイメージのビルド 12
3.1. ソースコードの取得 12
3.2. ソースコードアーカイブの展開 12
3.3. 設定 12
3.4. ビルド 14
3.5. イメージ 14
3.6. まとめ 14
4. ディレクトリ構成 15
4.1. Makefile 15
4.2. config 15
4.3. tools 15
4.4. glibc と uClibc 16
4.5. user 16
4.6. vendors 16
5. Make の基本 18
6. 基本ターゲット 20
6.1. 設定（config） 20
6.1.1. テキスト画面での設定 (make config) 20
6.1.2. メニュー画面での設定 (make menuconfig) 21
6.1.3. GUI 画面での設定 (make xconfig) 21
6.2. クリーン（clean） 22
6.3. 依存関係の解決（dep） 22
6.4. デフォルト（all） 22
7. イメージファイルの作成 23
7.1. 全体の流れ 23
7.2. メニューベースコンフィギュレーションの基本操作 24
7.2.1. 移動 24
7.2.2. サブメニューの選択 24
7.2.3. リストからの選択 24
7.2.4. 有効無効の選択 24
7.3. コンフィギュレーション 24
7.3.1. Main Menu（メインメニュー） 26
7.3.2. Vendor/Product Selection（ベンダ/プロダクト選択） 26
7.3.3. Kernel/Library/Defaults Selection（カーネル/ライブラリ/デフォルト選択） 26
7.3.4. Kernel Version（カーネルの選択） 26
7.3.5. Libc Version（C ライブラリの選択） 26
7.3.6. Default all settings （デフォルトの設定に戻す） 26
7.3.7. Customize Kernel Settings（カーネル設定の変更） 26
7.3.8. Customize Vendor/User Settings（ベンダ/ユーザ設定の変更） 27
7.3.9. Update Default Vendor Settings（デフォルトベンダ設定の更新） 27
7.4. ユーザーランドの設定 27
7.4.1. Core Application 27

7.4.2. Library Configuration 27
7.4.3. Flash Tools 27
7.4.4. Filesystem Applications 27
7.4.5. Network Applications 27
7.4.6. Miscellaneous Applications 27
7.4.7. Busybox 27
7.4.8. Tinylogin 28
7.4.9. MicroWindows 28
7.4.10. Game 28
7.4.11. Miscellaneous Configuration 28
7.4.12. Debug Builds 28
7.5. カーネルソースコードの依存関係解決 28
7.6. ビルド 28
7.7. 詳細なビルドの流れ 29
7.7.1. subdirs ターゲット 29
7.7.2. romfs ターゲット 30
7.7.3. modules 31
7.7.4. modules_install 31
7.7.5. image 31
8. プロダクトディレクトリ 32
8.1. config.arch 32
8.2. config.linux-2.4.x 33
8.3. config.vendor 33
8.4. config.uClibc 33
8.5. Makefile 33
9. romfs インストールツール 34
9.1. 概要 34
9.2. ファイルのインストール 36
9.3. ディレクトリのインストール 37
9.4. リンクの作成 37
9.5. ファイルへの情報追記 38
9.6. 条件実行 38
10. 新規アプリケーションの追加方法 40
10.1. Out of Tree コンパイル 40
10.1.1. 準備 40
10.1.2. ソースコードの用意 40
10.1.3. ビルド（uClinux） 41
10.1.4. インストール 42
10.1.5. image ファイルの作成 42
10.1.6. 複数のソースコード 42
10.1.7. pthread 対応 43
10.2. プロダクト別のアプリケーション 44
10.2.1. ディレクトリの準備 44
10.2.2. ソースコードの用意 44
10.2.3. 追加アプリケーションの設定 45
10.2.4. ビルド 45
10.3. user ディレクトリへのマージ 45
10.3.1. ディレクトリの準備 46
10.3.2. ソースコードの用意 46
10.3.3. 追加アプリケーションの設定 46
10.3.4. アプリケーションの選択 46
10.3.5. ビルド 47
10.3.6. コンフィグの命名規則 47

10.3.7. 複数のアプリケーション 47
11. 新規デバイスドライバの追加方法 50
11.1. Out of Tree コンパイル 50
11.1.1. 準備 50
11.1.2. ソースコードの用意 50
11.1.3. ビルド（uClinux） 52
11.1.4. インストール 53
11.1.5. `image` ファイルの作成 53
11.2. drivers ディレクトリへのマージ 53
11.2.1. ソースコードの用意 53
11.2.2. 追加ドライバの設定 53
11.2.3. ドライバの選択 54
11.2.4. ビルド 55
12. 複数カーネルの利用 56
13. 特有なアプリケーションの説明 57
13.1. netflash 57
13.2. flatfsd 58

図目次

6.1. メニューベースコンフィグギュレーションの画面 21
6.2. GUI ベースコンフィギュレーションの画面 22
7.1. イメージ作成の流れ 23
10.1. メニューに追加された hello 47
11.1. メニューに追加された message 55

表目次

1.1. 使用しているフォント 9
1.2. 表示プロンプトと実行環境の関係 9

例目次

3.1. uClinux-dist のファイル名 12
5.1. Makefile 練習用のディレクトリを作成 18
5.2. 簡単な Makefile 18
5.3. make の実行 18
5.4. ゴールを指定して make を実行 19
5.5. 複数のゴールを指定して make を実行 19
6.1. テキストベースコンフィギュレーション 20
6.2. メニューベースコンフィギュレーションの起動コマンド 21
6.3. GUI ベースコンフィギュレーションの起動コマンド 21
7.1. make dep の実行 28
7.2. make の実行 28
7.3. user/Makefile からプロダクト Makefile が呼び出される 30
7.4. トップレベル Makefile での romfs ターゲット処理 30
7.5. user/Makefile での romfs ターゲット処理 31
9.1. romfs-inst.sh のヘルプ 35
9.2. romfs-inst.sh 構文 35
10.1. `uClinux-dist/config/config.in` の変更点 46
10.2. `uClinux-dist/user/Makefile` の変更点 46
10.3. `uClinux-dist/config/config.in` の変更点(複数アプリケーション) 48
10.4. `uClinux-dist/user/Makefile` の変更点(複数アプリケーション) 48
10.5. `Makefile`(複数アプリケーション) 49
11.1. `uClinux-dist/linux-2.4.x/drivers/char/Config.in` の変更点 54
11.2. `uClinux-dist/linux-2.4.x/drivers/char/Makefile` の変更点 54

1.はじめに

1.1. マニュアルについて

本マニュアルは、uClinux.org で配布している uClinux-dist を使用する上で必要な情報のうち、以下の点について記載されています。

- カーネルとユーザーランドのビルド
- カスタマイズ
- アプリケーション開発
- デバイスドライバ開発
- 特有のアプリケーション

また、特別な表記がないかぎり作業用のコンピュータでは Linux をベースにした OS が動作しているものと仮定します。 Windows 環境でも coLinux を使うことで簡単に Linux 環境を構築することが可能です。coLinux について詳しくは http://www.colinux.org/ を参照してください。

本マニュアルが uClinux-dist の機能を最大限引き出すためにご活用いただければ幸いです。

1.2. フォントについて

このマニュアルでは以下のようにフォントを使っています。

表 1.1. 使用しているフォント

フォント例	説明
本文中のフォント	本文
`[PC ~]$ ls`	プロンプトとユーザ入力文字列
	ソースファイルのコード、ファイル名、ディレクトリ名など
:	コマンド実行後の出力を省略

1.3. コマンド入力例の表記について

このマニュアルに記載されているコマンドの入力例は、表示されているプロンプトによって、それぞれに対応した実行環境を想定して書かれています。"/"の部分はカレントディレクトリによって異なります。ユーザホームディレクトリは"~"で表します。

表 1.2. 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
`[PC /]#`	作業用 PC 上の特権ユーザで実行
`[PC /]$`	作業用 PC 上の一般ユーザで実行
`[Target /]#`	ターゲットボード上の特権ユーザで実行
`[Target /]$`	ターゲットボード上の一般ユーザで実行

1.4. 謝辞

uClinux-dist で使用しているソフトウェアは Free Software / Open Source Software で構成されています。Free Software / Open Source Software は世界中の多くの開発者の成果によってなりたっています。この場を借りて感謝の意を示したいと思います。

uClinux は D. Jeff Dionne 氏や Greg Ungere 氏、David McCulloughu 氏、さらに uClinux development list に参加しているすべての人の成果によって支えられています。uClibc、Busybox は Eric Andersen 氏、さらにそれぞれのコミュニティによって開発・保守されています。

1.5. ソフトウェアに関する注意事項

本書に含まれるソフトウェアは、現状のまま（AS IS）提供されるものであり、特定の目的に適合することや、その信頼性、正確性を保証するものではありません。また、本製品の使用による結果についてもなんら保証するものではありません。

2.uClinux-dist について

uClinux-dist は、uClinux.org が配布するソースコードベースのディストリビューションです。最初 uClinux-dist は uClinux[1]用に作成されましたが、uClinux 専用というわけではありません。uClinux に対応するために追加変更を行ったもので、設定時に既存の Linux を選択することで i386 や ARM、PowerPC のような MMU をもった CPU にも対応しています。

uClinux では既存の Linux にはない制限がいくつか存在します。そのため、uClinux を使う場合には、既存の Linux アプリケーションを uClinux に対応させる必要がありました。uClinux に対応させたアプリケーションを、Linux カーネルのビルドシステムと組み合わせ、フラッシュメモリなどに書き込むためのファイルの生成まで自動で行ってくれるようにしたものが、uClinux-dist です。

既存の Linux ディストリビューションは、主にデスクトップやサーバ用に構成されています。このため、組み込み機器のベースとして採用するには、ファイルサイズやメモリ使用量などに問題がありました。uClinux-dist では最初から組み込み機器を想定しているため、コンパイル時に必要な機能だけを選択することが可能になっています。また、製品別に専用の設定を持つことができ、それを選択することで容易に特定製品向けのイメージファイルを作成できることや、カーネルとユーザランドの設定を同じメニューから選択でき、一つのイメージファイルとして出力するなど、多くの優れた特長があります。このため uClinux を使用する製品に限らず、MMU を搭載し Linux が動作する組み込み製品においても、利用すべきメリットがあります。

[1]uClinux とは MMU を持たないマイクロコンピュータでも動作するように作成された Linux です。通常 HDD などの大型補助記憶装置を持たない製品で利用されます。2.6 系の Linux では uClinux の成果がマージされています。

3.デフォルトイメージのビルド

はじめに、使用しているターゲットボードのデフォルトイメージを作成してみましょう。なお作業を行う際は、必ず一般ユーザで行ってください。root ユーザで行うと作業ミスなどにより、開発用マシンの環境を壊す可能性があります。

3.1. ソースコードの取得

（株）アットマークテクノ（以降、アットマークテクノと略記）が配布している SUZAKU 用の uClinux-dist は、以下の URL の dist フォルダからダウンロードすることができます。

http://suzaku.atmark-techno.com/downloads/all/

また、SUZAKU の開発者サイトや開発キットでも入手可能です。

uClinux.org で配布している オリジナルの `uClinux-dist` は、`uClinux-dist-YYYYMMDD.tar.gz` のように名前に日付が付いています。YYYY は年を、MM は月を、DD は日を表しています。SUZAKU シリーズでは、-suzakuX を追加したファイル名で配布しています。X はバージョンを意味します。

例 3.1. uClinux-dist のファイル名

```
[PC ~]$ ls
uClinux-dist-20051110-suzaku7.tar.gz
```

3.2. ソースコードアーカイブの展開

uClinux-dist は、どこに展開しても問題ありません。作業しやすく、HDD の容量に余裕のある場所を選んで展開してください。展開時で 500MB、コンパイル後には 1GB 近くの容量が必要になる場合もあります。ここでは、便宜上ユーザのホームディレクトリ(~)に uClinux-dist を展開することにします。

```
[PC ~]$ tar xzf uClinux-dist-YYYYMMDD-suzakuX.tar.gz
    :
    :
[PC ~]$ ls
uClinux-dist-YYYYMMDD-suzakuX/
```

以降、uClinux-dist-YYYYMMDD-suzakuX は、uClinux-dist と略記します。

3.3. 設定

uClinux-dist をターゲットボード用にコンフィギュレーションします。以下の例のようにコマンドを入力し、コンフィギュレーションを開始します。

```
[PC ~/uClinux-dist]$ make config
```

使用するボードのベンダー名を聞かれるので AtmarkTechno と入力してください。

```
*
* Vendor/Product Selection
*
*
* Select the Vendor you wish to target
*
Vendor (3com, ADI, Akizuki, Apple, Arcturus, Arnewsh, AtmarkTechno, Atmel, Avnet,
Cirrus,
Cogent, Conexant, Cwlinux, CyberGuard, Cytek, Exys, Feith, Future, GDB, Hitachi,
Imt,
Insight, Intel, KendinMicrel, LEOX, Mecel, Midas, Motorola, NEC, NetSilicon,
Netburner,
Nintendo, OPENcores, Promise, SNEHA, SSV, SWARM, Samsung, SecureEdge, Signal,
SnapGear,
Soekris, Sony, StrawberryLinux, TI, TeleIP, Triscend, Via, Weiss, Xilinx, senTec)
[SnapGear]
(NEW) AtmarkTechno
```

次にボード名を聞かれます。使用しているボード名を入力してください。ここでは例として SUZAKU-S.SZ130 を入力します。

```
*
* Select the Product you wish to target
*
AtmarkTechno Products (SUZAKU-S.SZ010, SUZAKU-S.SZ030, SUZAKU-S.SZ130, SUZAKU-
S.SZ130-SIL, SUZAKU-UQ-XIP, SUZAKU-V.SZ310, SUZAKU-V.SZ310-SIL) [SUZAKU-S.SZ010]
(NEW) SUZAKU-S.SZ130
```

続いて、使用する C ライブラリを指定します。使用するボードによってサポートされているライブラリは異なります。SUZAKU のライブラリは、uClibc を選択します。詳しくは、お使いの製品のソフトウェアマニュアルを参照してください。

```
*
* Kernel/Library/Defaults Selection
*
*
* Kernel is linux-2.4.x
*
Libc Version (None, glibc, uC-libc, uClibc) [uClibc] (NEW)uClibc
```

次にデフォルトの設定にするかどうか質問されます。y（Yes）を選択してください。

```
Default all settings (lose changes) (CONFIG_DEFAULTS_OVERRIDE) [N/y/?] (NEW) y
```

最後の３つの質問は n(No)と答えてください。

```
Customize Kernel Settings (CONFIG_DEFAULTS_KERNEL) [N/y/?] n
Customize Vendor/User Settings (CONFIG_DEFAULTS_VENDOR) [N/y/?] n
Update Default Vendor Settings (CONFIG_DEFAULTS_VENDOR_UPDATE) [N/y/?]  n
```

質問事項が終わるとビルドシステムが設定を行います。すべての設定が終わるとプロンプトに戻ります。

3.4. ビルド

ビルドするには以下のコマンドを入力してください。ビルドの途中でいくつか新しく質問される場合があります。その場合は Enter キーを押してください。

```
[PC ~/uClinux-dist]$ make dep all
```

3.5. イメージ

ビルドが正常終了すると、イメージファイルが images ディレクトリに生成されます。選択した製品によって、生成されるファイル数やファイルの種類が異なる場合があります。お使いの製品のソフトウェアマニュアルを参照してください。

できあがったイメージファイルを製品に書き込む方法は、お使いの製品のソフトウェアマニュアルを参照してください。

3.6. まとめ

デフォルトイメージのビルド手順を以下にまとめます。

```
[PC ~]$ ls
uClinux-dist-YYYYMMDD-suzakuX.tar.gz
[PC ~]$ tar xvzf uClinux-dist-YYYYMMDD-suzakuX.tar.gz
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ make config
  Vendor AtmarkTechno
  AtmarkTechno Products SUZAKU-S.SZ130
  Libc Version uClibc
  Default all settings y
  Customize Kernel Settings n
  Customize Vendor/User Settings n
  Update Default Vendor Settings n

[PC ~/uClinux-dist]$ make dep all
[PC ~/uClinux-dist]$ ls image/
image.bin  linux.bin  romfs.img
```

4.ディレクトリ構成

本章では、uClinux-dist のディレクトリ構成について説明します。ディレクトリ構成を理解することは、uClinux-dist をベースに開発を行う際に、非常に重要なポイントとなります。

この章では各ディレクトリの概要を説明するとともに、各ディレクトリに関連した説明がなされている章を紹介します。

```
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ tree -L 1 -F
.
|-- COPYING
|-- Documentation/
|-- Makefile
|-- README
|-- SOURCE
|-- bin/
|-- config/
|-- freeswan/
|-- glibc/
|-- include/
|-- lib/
|-- linux-2.4.x/
|-- openswan/
|-- tools/
|-- uClibc/
|-- user/
`-- vendors/

13 directories, 4 files
```

4.1. Makefile

uClinux-dist は、すべて make コマンドによって作業を行います。この uClinux-dist/Makefile をトップレベル Makefile と呼び、他のディレクトリにある Makefile とは区別することにします。トップレベル Makefile は、uClinux-dist のビルドをコントロールする大事なファイルです。ここで定義されているターゲットは「6. 基本ターゲット」の章で詳しく説明します。また make コマンドについては、「5. Make の基本」で簡単に説明します。

4.2. config

config ディレクトリには設定に必要な `script` や `Makefile` が収録されています。Makefile と config.in については、「10.2. プロダクト別のアプリケーション」の章で詳しく説明します。

4.3. tools

tools ディレクトリには、ビルドに必要ないくつかのツールが収められています。このディレクトリに収められている romfs-inst.sh はプロダクトディレクトリの Makefile でよく使います。「9. romfs インストールツール」で、詳しく説明します。

cksum は、netflash で使うことがあります。netflash については、「13.1. netflash」を参照してください。

4.4. glibc と uClibc

GNU C library (glibc)と uClibc は、uClinux-dist で採用している C ライブラリです。

4.5. user

このディレクトリにはユーザランドアプリケーションが収められています。多くのアプリケーションは、GNU/Linux 用のアプリケーションを uClinux でも使用できるように変更したものですが、uClinux 専用に開発されたものも含まれます。

uClinux-dist をベースにソフトウェア開発を行う上で特徴的なアプリケーションは「13. 特有なアプリケーションの説明」で詳しく説明します。

4.6. vendors

vendors ディレクトリは以下のようになっています。

```
[PC ~/uClinux-dist]$ tree vendors
vendors
    :
    :
|-- AtmarkTechno
|   |-- SUZAKU-S.SZ130
|   |    |-- Makefile
|   |    |-- config.arch
|   |    |-- config.linux-2.4.x
|   |        :
|   |-- SUZAKU-UQ-XUP
|   `-- SUZAKU-V.SZ310
|        |-- Makefile
|        |-- config.arch
|        |-- config.linux-2.4.x
|        |-- config.uClibc
|        |-- config.vendor
|        |-- default
|        `-- etc
    :
    :
|-- config
|   |-- arm
|   |    `-- config.arch
|   |-- armnommu
|   |    `-- config.arch
|   |-- h8300
|   |    `-- config.arch
|   |-- i386
|   |    `-- config.arch
|   |-- m68knommu
|   |    `-- config.arch
|   |-- microblaze
|   |    `-- config.arch
|   |-- powerpc
```

```
|   |    `-- config.arch
    :
    :
```

vendors ディレクトリの中には、AtmarkTechno をはじめ、ベンダー名のディレクトリが多数存在します。さらに、ベンダー名のディレクトリの中にはプロダクト用のサブディレクトリが複数収められています。AtmarkTechno ディレクトリ内には、SUZAKU-S.SZ130 をはじめ、uClinux-dist に対応しているプロダクトがそれぞれ入っています。これらのディレクトリを「プロダクトディレクトリ」と呼びます。プロダクトディレクトリには、個々の製品用のイメージをビルドするためのさまざまなファイルが含まれています。プロダクトディレクトリについては、「8. プロダクトディレクトリ」を参照してください。

ベンダー名のディレクトリ以外には、`config` というディレクトリがあります。アーキテクチャごとのデフォルト設定が `config.arch` という名前でそれぞれのディレクトリ内に保存されています。この `config.arch` はプロダクトディレクトリの `config.arch` から参照されています。詳しくは「8.1. config.arch」を参照してください。

5.Make の基本

make は、プログラムのコンパイルに広く使われているコマンドです。どのソースコードをどのようにコンパイルすれば良いかを判断し、コンパイルに必要なコマンドを実行してくれます。**make** を使うには Makefile と呼ばれるファイルを用意します。Makefile には、コンパイルされるソースコードの依存関係や、コンパイルに必要なコマンドなどの情報が書かれています。

uClinux-dist のビルドシステムでも Makefile をベースとしています。ここでは、**make** コマンドと Makefile の基本について簡単に説明します。

最初に練習用のディレクトリを作成します。ここでは、maketest という名前にします。

例 5.1. Makefile 練習用のディレクトリを作成

```
[PC ~]$ mkdir maketest
[PC ~]$ cd maketest
[PC ~/maketest]$
```

次に、maketest ディレクトリ内に簡単な Makefile を用意します。

例 5.2. 簡単な Makefile

```
hello:
         echo 'Hello World'

bye:
         echo 'Bye bye'
```

Makefile の中で、hello と bye はターゲットと呼ばれます。ターゲットは必ずコロン「：」で終わらなければいけません。ターゲットの次の行が実際に実行されるコマンドです。ここではコンパイルをせず、"Hello World"を出力するために、echo コマンドを使っています。実行されるコマンドの前はタブで始まらなければならない規則があります。

例 5.3. make の実行

```
[PC ~/maketest]$ ls
Makefile
[PC ~/maketest]$ make
echo 'Hello World'
Hello World
[PC ~/maketest]$
```

上記は、実際に make コマンドを実行した例です。make コマンドを実行すると、Makefile 中の実行されるコマンド「echo 'Hello World'」と、コマンドの結果「Hello World」が表示されます。

make コマンドは Makefile 内で定義されているターゲットを引数としてとることが可能です。指定されたターゲットをゴールと呼びます。

例 5.4. ゴールを指定して make を実行

```
[PC ~/maketest]$ make hello
echo 'Hello World'
Hello World
[PC ~/maketest]$ make bye
echo 'Bye bye'
Bye bye
[PC ~/maketest]$
```

make コマンドが引数を取らない場合は、`Makefile` 内の一番上にあるターゲットをゴールとして実行されます。このため、**make** と **make hello** では同じ動作になります。多くの `Makefile` では、一番上のターゲットは all という名前で定義されています。

複数のゴールを羅列することもできます。この場合は、左から羅列順に実行されます。

例 5.5. 複数のゴールを指定して make を実行

```
[PC ~/maketest]$ make hello bye
echo 'Hello World'
Hello World
echo 'Bye bye'
Bye bye
[PC ~/maketest]$
```

make と `Makefile` の詳しい情報は、**make** のマニュアルまたは info をご覧ください。

6.基本ターゲット

uClinux-dist でビルドする場合に、よく使うターゲットをここで説明します。uClinux-dist の目的が「ターゲットボードに書き込むためのイメージファイル作成」であることは、すでに説明したとおりです。「3. デフォルトイメージのビルド」で使用したターゲットについて詳しく説明します。

6.1. 設定（config）

設定用のターゲットには、以下の 3 種類が用意されています。表示方法が異なるだけで、すべて uClinux-dist の設定変更を行うためのターゲットです。

6.1.1. テキスト画面での設定 (make config)

config ターゲットは、「3. デフォルトイメージのビルド」の章で使用した設定方法です。デフォルトのイメージを作成するような簡単な設定をするときに便利です。

例 6.1. テキストベースコンフィギュレーション

```
[PC ~/uClinux-dist]$ make config
config/mkconfig > config.in
#
# Using defaults found in .config
#
*
* Vendor/Product Selection
*
*
* Select the Vendor you wish to target
*
Vendor (3com, ADI, Akizuki, Apple, Arcturus, Arnewsh, AtmarkTechno, Atmel, Avnet,
Cirrus, Cogent, Conexant, Cwlinux, CyberGuard, Cytek, EMAC, ESPD, Exys, Feith,
Future, GDB, Hitachi, Imt, Insight, Intel, KendinMicrel, LEOX, Mecel, Midas,
Motorola, NEC, NetSilicon, Netburner, Nintendo, OPENcores, OpenGear, Philips,
Promise, SNEHA, SSV, SWARM, Samsung, SecureEdge, Signal, SnapGear, Soekris, Sony,
StrawberryLinux, TI, TeleIP, Triscend, Via, Weiss, Xilinx, senTec) [AtmarkTechno]
  defined CONFIG_DEFAULTS_ATMARKTECHNO
*
* Select the Product you wish to target
*
AtmarkTechno Products (SUZAKU-S.SZ010, SUZAKU-S.SZ030, SUZAKU-S.SZ130, SUZAKU-
S.SZ130-SIL, SUZAKU-UQ-XUP, SUZAKU-V.SZ310, SUZAKU-V.SZ310-SIL) [SUZAKU-S.SZ130]
  defined CONFIG_DEFAULTS_ATMARKTECHNO_SUZAKU_S_SZ130
*
* Kernel/Library/Defaults Selection
*
*
* Kernel is linux-2.4.x
*
Libc Version (glibc, uC-libc, uClibc) [uClibc]
```

6.1.2. メニュー画面での設定 (make menuconfig)

menuconfig ターゲットでは、メニュー画面を使って設定を行うことができます。メニュー画面は ncurses を使って描画されます。このターゲットが指定されたときに画面などをコントロールするプログラムをビルドするため、ncurses のライブラリとヘッダファイルが必要になります。多くのディストリビューションでは、ncurses の開発用パッケージが用意されていますので、インストールしてください。

例 6.2. メニューベースコンフィギュレーションの起動コマンド

```
[PC ~/uClinux-dist]$ make menuconfig
```

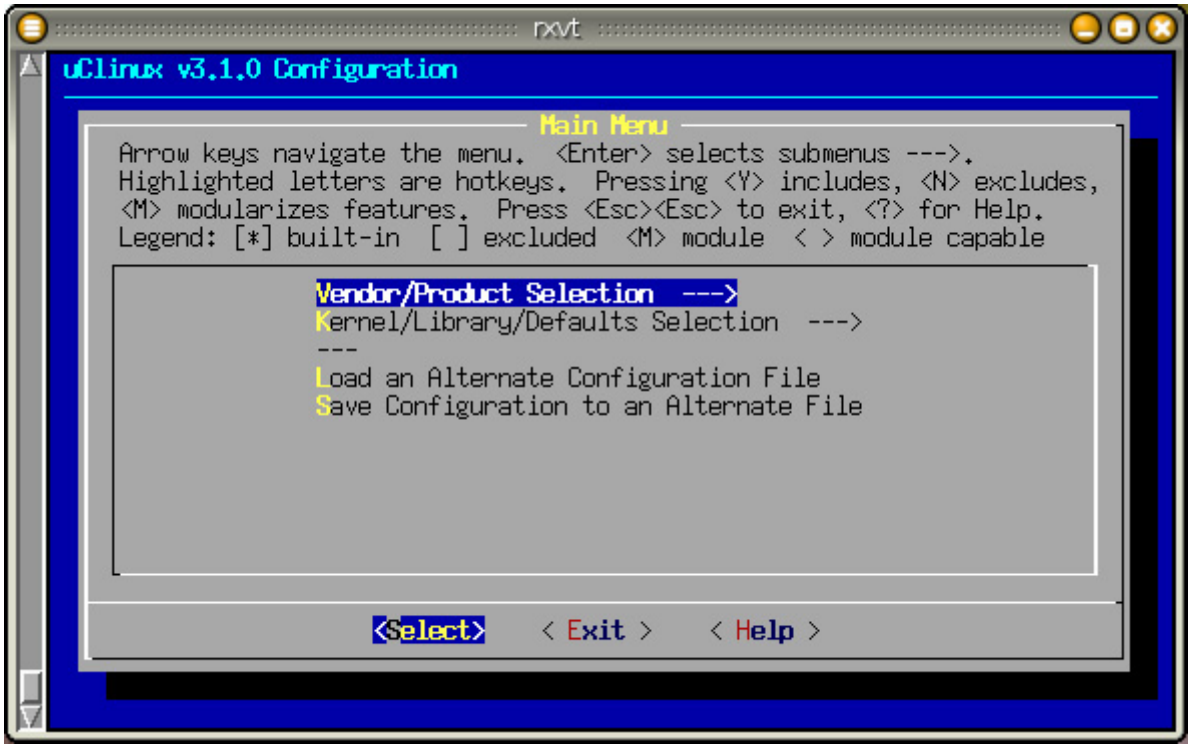


図 6.1. メニューベースコンフィグギュレーションの画面

6.1.3. GUI 画面での設定 (make xconfig)

X Window System が使える環境であれば、**make xconfig** ターゲットを使用することができます。**make xconfig** では、ターミナル中のメニュー画面と違い、マウスで操作することができます。

例 6.3. GUI ベースコンフィギュレーションの起動コマンド

```
[PC ~/atmark-dist]$ make xconfig
```

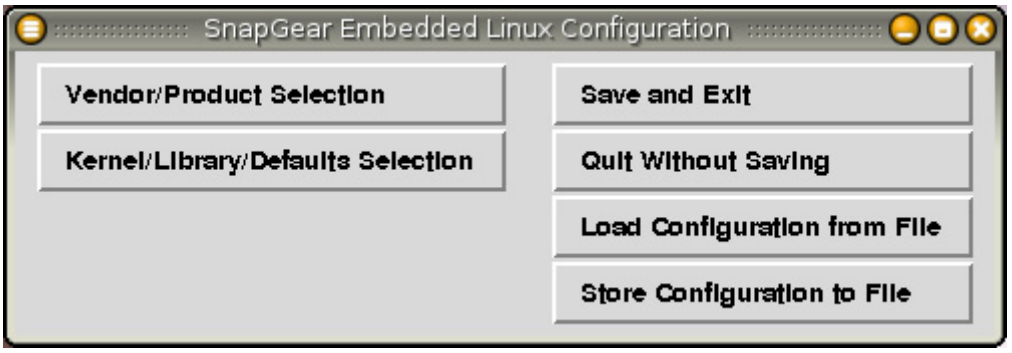


図 6.2. GUI ベースコンフィギュレーションの画面

6.2. クリーン（clean）

uClinux-dist 内をきれいにするためのターゲットです。uClinux-dist には以下の 3 種類の clean ターゲットが用意されています。

- **`make clean`**

  カーネルディレクトリの clean と、`romfs/image` ディレクトリ内のファイルの削除を行う。

- **`make real_clean`**

  `romfs/image` ディレクトリの削除や、`.config` などのコンフィグ用ファイルの削除を行う。

- **`make distclean`**

  カーネルディレクトリで、distclean ターゲットの実行などを行う。

6.3. 依存関係の解決（dep）

2.4 系までの Linux カーネルのビルドシステムでは、**`make`** 前に依存関係の解決を行わなければなりません。このターゲットは、カーネルディレクトリで依存関係の解決を行います。uClinux-dist のソースアーカイブを展開した後や **`make distclean`** を行った後に必ず実行しなければならないターゲットです。

6.4. デフォルト（all）

all ターゲットは、uClinux-dist のデフォルトターゲットです。**`make`** コマンドをオプションなしで実行することで、このターゲットが実行されます。**`make all`** では、必要なコード（カーネル、ユーザランドアプリケーション、ライブラリ）のコンパイルを行い、ターゲットボードに転送できるイメージファイルを生成します。

7.イメージファイルの作成

7.1. 全体の流れ

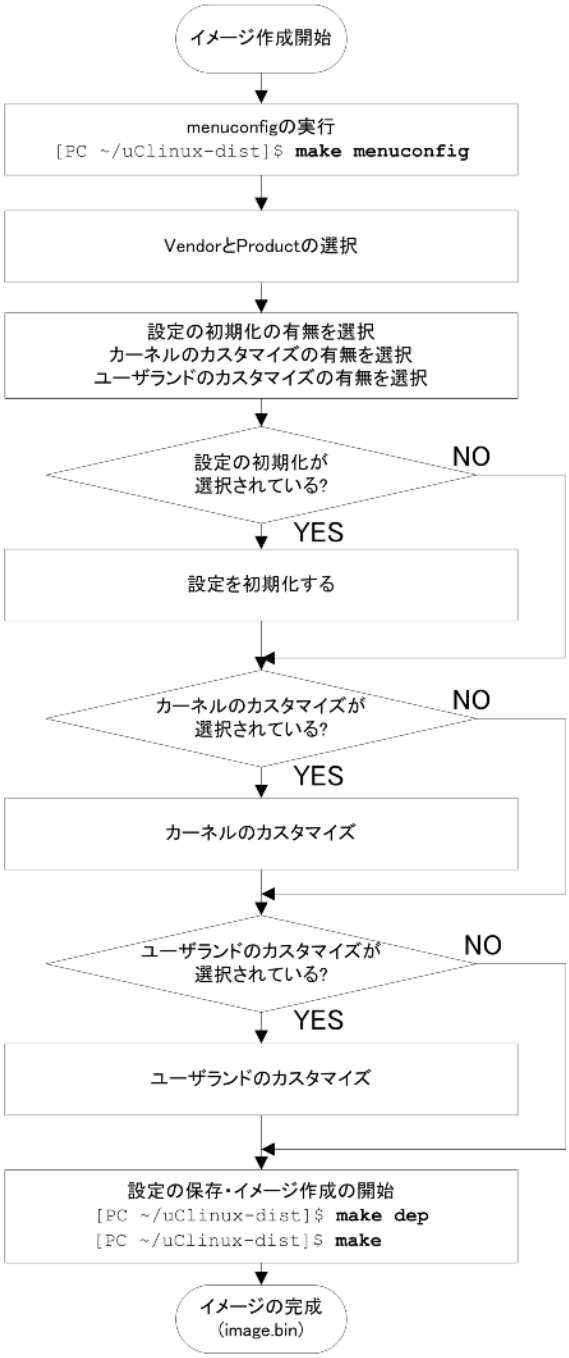


図 7.1. イメージ作成の流れ

デフォルトのイメージを作成する一連の手順を順番に説明します。一連の手順は、大きく３つに分割することができます。

- コンフィギュレーション
- カーネルソースコードの依存関係解決
- ビルド

7.2. メニューベースコンフィギュレーションの基本操作

7.2.1. 移動

カーソルキーでメニュー内の移動を行います。

7.2.2. サブメニューの選択

Enter キーを押すことで、サブメニューを選択できます。サブメニューは --->で表示されます。

7.2.3. リストからの選択

小括弧「( )」で表示されている部分は、リストから選択する部分です。Enter キーでリスト画面に移動し、上下のカーソルキーで選択対象に移動し、Enter キーで選択します。

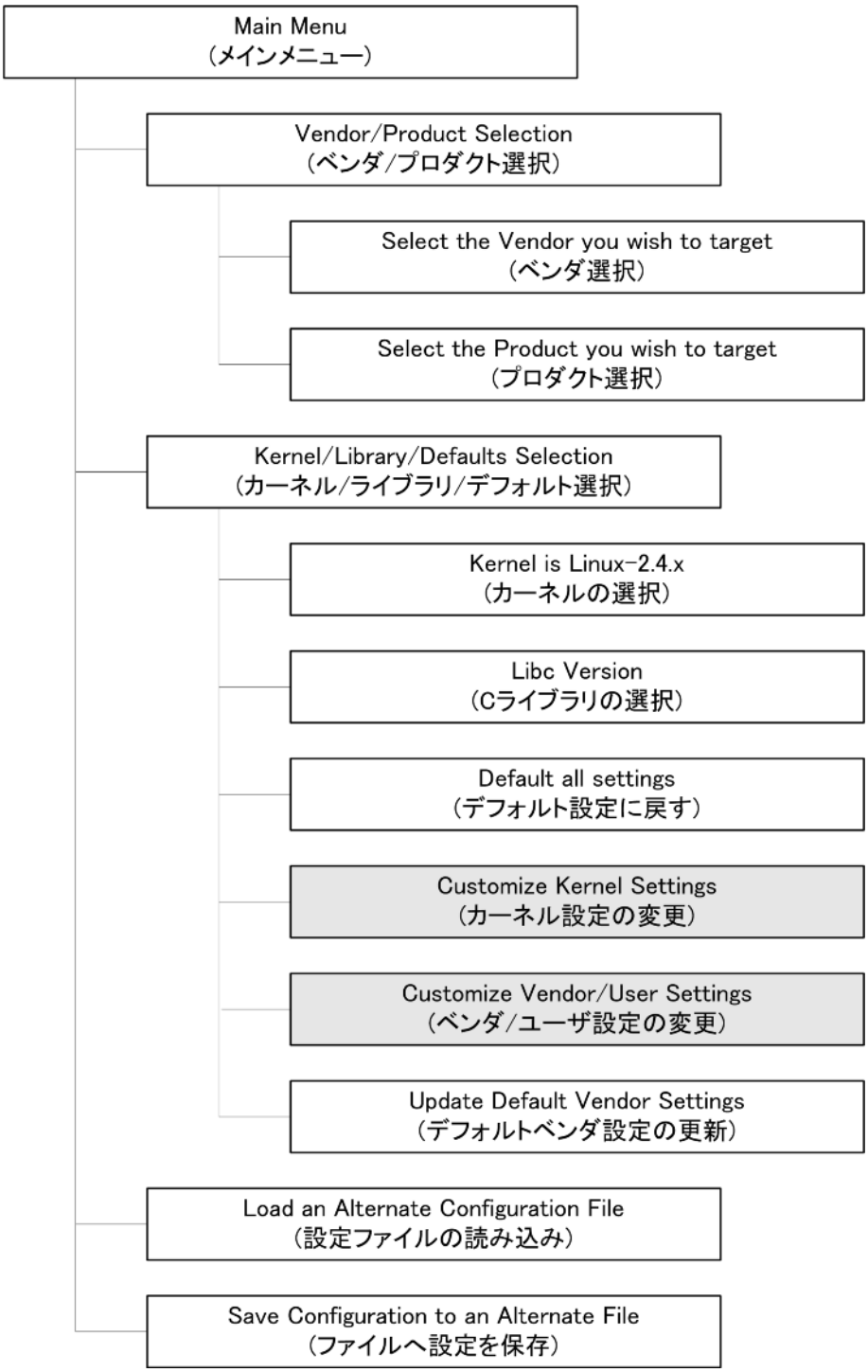
7.2.4. 有効無効の選択

大括弧「[ ]」は、有効無効の選択を表します。選択されるとアスタリスク「*」が大括弧内に表示されます。

7.3. コンフィギュレーション

「6.1. 設定（config)」で紹介したコマンドのうち、どれか一つを使ってコンフィギュレーションを行います。ここでは **`make menuconfig`** を例に説明します。

コンフィギュレーションは、大きく次のようにわけることができます。



この中でカーネル設定とベンダー/ユーザ設定は、選択後に一度メニューを終了することで、それぞれの設定画面が自動的に表示されます。

コンフィギュレーションは、以下のようにはじめます。

```
[PC ~/uClinux-dist]$ make menuconfig
```

7.3.1. Main Menu（メインメニュー）

make menuconfig を実行するとメインメニュー画面が表示されます。この画面から各サブメニューに移動することができます。

7.3.2. Vendor/Product Selection（ベンダ/プロダクト選択）

ベンダーとプロダクトを選択します。先にベンダーを選択し、その後プロダクトを選択します。

7.3.3. Kernel/Library/Defaults Selection（カーネル/ライブラリ/デフォルト選択）

カーネルやライブラリ、デフォルトの選択を行います。以降のメニューはこのサブメニュー中にあります。

7.3.4. Kernel Version（カーネルの選択）

複数のカーネルをサポートしたプロダクトの場合、ここでビルドするカーネルのバージョンを選択します。SUZAKU シリーズで現在対応しているカーネルは 1 種類だけのため、他のバージョンを選択することはできません。

7.3.5. Libc Version（C ライブラリの選択）

複数の C ライブラリをサポートしたプロダクトの場合、ここでビルドする C ライブラリを選択します。選択肢としては、以下の４つが選択対象です。

- None
- glibc-GNU C ライブラリ
- uC-libc
- uClibc

None を選択すると、すでに開発環境にインストールされている C ライブラリを使用します。そのほかの選択肢では、atmark-dist に含まれているコードをビルドします。

製品によって対応しているライブラリが異なりますので、製品のソフトウェアマニュアルを参照してください。

7.3.6. Default all settings （デフォルトの設定に戻す）

すべての設定をデフォルトの状態に変更します。変更した設定情報はすべて無効になりますので注意が必要です。

7.3.7. Customize Kernel Settings（カーネル設定の変更）

Linux カーネルのコンフィギュレーションを行うか否かを選択します。選択した場合、uClinux-dist の設定終了後に自動的にカーネルの設定画面が起動されます。

7.3.8. Customize Vendor/User Settings（ベンダ/ユーザ設定の変更）

ユーザランドのコンフィギュレーションを行うか否かを選択します。選択した場合、uClinux-dist の設定終了後に自動的にユーザランドの設定画面が起動されます。ユーザランドのメニューについては次節で説明します。

7.3.9. Update Default Vendor Settings（デフォルトベンダ設定の更新）

現在の設定で、デフォルト設定を更新します。更新した場合、古い設定に戻す方法はありませんので、注意が必要です。

7.4. ユーザーランドの設定

ユーザーランドのメニューは以下のような項目に別れています。簡単に各項目について説明します。

7.4.1. Core Application

システムとして動作するために必要な基本的なアプリケーションが入っています。システムの初期化を行う init やユーザ認証の login などがこのセクションで選択できます。

7.4.2. Library Configuration

アプリケーションが必要とするライブラリの選択ができます。

7.4.3. Flash Tools

フラッシュメモリに関係のあるアプリケーションが選択できます。以降の章で説明する netflash と呼ばれるネットワークアップデート用アプリケーションがここで選択されています。

7.4.4. Filesystem Applications

ファイルシステムに関係のあるアプリケーションが選択できます。以降の章で説明する flatfsd はここで選択することができます。その他、mount、fdisk、ext2 ファイルシステム、Reiser ファイルシステム、Samba などが含まれます。

7.4.5. Network Applications

ネットワークに関係のあるアプリケーションが選択できます。dhcpcd-new、ftpd、ifconfig、inetd、thttpd の他にも ppp やワイヤレスネットワークのユーティリティなども含まれます。

7.4.6. Miscellaneous Applications

上記のカテゴリに属さないアプリケーションが収められています。Unix の一般的なコマンド(cp、ls、rm 等)やエディタ、オーディオ関連、スクリプト言語インタプリタなどが含まれます。

7.4.7. Busybox

Busybox のカスタマイズを行います。Busybox は複数のコマンド機能をもった単一コマンドで、多くの組み込み Linux での実績があります。Busybox は多くのカスタマイズできるため、別セクションになっています。

7.4.8. Tinylogin

Tinylogin も複数コマンドの機能をもつアプケーションです。login や passwd、getty など認証に関係のある機能を提供します。多くのカスタマイズが可能なため別セクションになっています。

7.4.9. MicroWindows

MicroWindows は組み込み機器をターゲットにしたグラフィカルウインドウ環境です。LCD などを持つ機器を開発する場合に使えます。

7.4.10. Game

ゲームです。

7.4.11. Miscellaneous Configuration

いろいろな設定がまとめられています。SUZAKU の root ユーザのパスワードもここで変更できます。

7.4.12. Debug Builds

デバッグ用のオプションがまとめられています。開発中にアプリケーションのデバッグを行うときに選択します。

7.5. カーネルソースコードの依存関係解決

イメージのビルドにはカーネルのビルドも含まれています。2.4 系の Linux カーネルはビルドの前に依存関係を解決しなければなりません。依存関係の解決には、dep ターゲットを使います。これは、uClinux でも同じです。

例 7.1. make dep の実行

```
[PC ~/uClinux-dist]$ make dep
    :
    :
[PC ~/uClinux-dist]$
```

7.6. ビルド

依存関係の解決が終わったら実際にビルドします。ビルドシステムがすべてを行ってくれるため、開発者は **make** と入力するだけです。

例 7.2. make の実行

```
[PC ~/uClinux-dist]$ make
    :
    :
[PC ~/uClinux-dist]$ ls images
image.bin  linux.bin  romfs.img
```

make コマンドが終了すると、`images` ディレクトリに `image.bin` が生成されます。

7.7. 詳細なビルドの流れ

内部的にどのようにビルドが進み、最終的にイメージファイルが作られるのかを理解すると、必要な部分だけをビルドすることもできます。これによりビルドに必要な時間を大幅に短縮でき、効率的なカスタマイズを行うことができます。

デフォルトのビルドを行うとイメージファイルが作成されますが、この間に多数のターゲットが実行されています。デフォルトターゲットのビルドルールは、`uClinux-dist` ルートディレクトリの `Makefile` に以下のように記載されています。

```
ifeq (.config,$(wildcard .config))
include .config

all: ucfront subdirs romfs modules modules_install image
else
all: config_error
endif
```

デフォルトビルドを実行すると ucfront, subdirs, romfs, modules, modules_install, image の順にビルドが行われるのが分かります。

この流れに沿って、各ターゲットのビルドを説明します。

7.7.1. subdirs ターゲット

uClinux-dist ルートディレクトリの Makefile の subdirs ビルドルールを以下に示します。

```
DIRS    = $(VENDOR_TOPDIRS) include lib include user
:
:
subdirs: linux
        echo "Build start unix"
        echo $(BUILD_START_UNIX)
        for dir in $(DIRS) ; do [ ! -d $$dir ] || $(MAKEARCH_KERNEL) -C $$dir ||
exit 1 ; done
```

subdirs ターゲットをビルドすると、linux, $(VENDOR_TOPDIRS), include, lib, include, user の順にビルドが行われます。

linux ターゲットは、選択された Linux カーネルのビルドを行います。uClinux-dist では version 2.0、2.4、2.6 のカーネルに対応していますが、現在 SUZAKU では 2.4 系の Linux カーネルのみ対応しています。

VENDOR_TOPDIRS は、プロダクトディレクトリにある `config.arch` で指定することが可能ですが、このディレクトリを指定しているプロダクトは少数派のようです。VENDOR_TOPDIRS を指定している多くのプロダクトでは、boot というディレクトリを指定しています。アットマークテクノの製品では、この変数を使用していません。

lib は、ライブラリを収めたディレクトリです。uClibc と glibc は `lib` ディレクトリに入っていませんが、コンフィグ時に `lib` ディレクトリ内にシンボリックリンクを生成するようになっています。

user は、ユーザランドアプリケーションを集めたディレクトリです。`user` ディレクトリには専用の `Makefile` が用意されており、トップレベルの `Makefile` はそちらに制御を任せるようになっています。

user ディレクトリにある Makefile の中でプロダクトディレクトリの Makefile が呼び出されます。以下の例はuser/Makefile からの抜粋です。

例 7.3. user/Makefile からプロダクト Makefile が呼び出される

```
VEND=$(ROOTDIR)/vendors

#
# must run the vendor build first
#
dir_v = $(VEND)/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)/.
dir_p = $(ROOTDIR)/prop

dir_y =
dir_n =
dir_  =
    :
    :
    :
    :

all: config
    for i in $(sort $(dir_y)) $(dir_v) $(dir_p); do \
        if [ ! -d $$i ]; then \
                        touch $$i/.sgbuilt_user; \
make -C $$i || exit $$? ; \
        fi; \
    done
```

7.7.2. romfs ターゲット

romfs ターゲットでは、各ディレクトリに対して romfs ターゲットを再帰的に呼びだします。多くの場合、romfs-inst.sh を使って必要なファイルを、uClinux-dist/romfs ディレクトリにインストールします。

例 7.4. トップレベル Makefile での romfs ターゲット処理

```
DIRS    = $(VENDOR_TOPDIRS) include lib include user
:
:
:

romfs:
    for dir in vendors $(DIRS) ; do [ ! -d $$dir ] || $(MAKEARCH) -C $$dir romfs
|| exit 1 ; done
    -find $(ROMFSDIR)/. -name CVS | xargs -r rm -rf
```

ビルド時には、user ディレクトリ以下にあるディレクトリのうち、選択されたアプリケーションのビルドが最初に行われ、その後、プロダクト Makefile が呼び出されます。

しかし、romfs ターゲットの時には、プロダクトディレクトリの Makefile が最初に呼び出されます。これは、プロダクトごとに柔軟性を与えるためですが、よほど複雑なビルドを行わなければならない場合以外、気にしなくても良いことです。

例 7.5. user/Makefile での romfs ターゲット処理

```
VEND=$(ROOTDIR)/vendors

#
# must run the vendor build first
#
dir_v = $(VEND)/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)/.
dir_p = $(ROOTDIR)/prop

dir_y =
dir_n =
dir_  =
    :
    :
    :
    :

romfs:
        for i in $(sort $(dir_y)) $(dir_p) ; do \
        [ ! -d $$i ] || make -C $$i romfs || exit $$? ; \
done
```

7.7.3. modules

Linux カーネルでは、多くのドライバなどがモジュールという形で分離できるようになっています。このターゲットは Linux カーネルのビルドシステムにある modules ターゲットを実行します。

7.7.4. modules_install

上記のターゲットでビルドされたカーネルモジュールを romfs にインストールします。`romfs/lib/modules` 内にインストールされます。

7.7.5. image

image ターゲットは、プロダクト Makefile の image ターゲットを実行するためのターゲットです。多くのプロダクトでは、カーネルのイメージとユーザランドのイメージをまとめて、一つのイメージファイルにしています。直接プロダクト Makefile のターゲットが呼び出されるため、開発者が自由に処理することができます。一般的な流れとしては

1. Linux カーネルの binary file を生成 (elf から binary に変換)
2. romfs ディレクトリからイメージファイルを生成(genext2fs や genromfs を使用)
3. 上記 2 つのファイルを 1 つにまとめる
4. netflash 用にチェックサムなどを生成しバイナリファイルに添付する

と、なっています。

8.プロダクトディレクトリ

プロダクトディレクトリとは、uClinux-dist/vendors/[]/以下にあるディレクトリ群のことです。例えば uClinux-dist/venders/AtmarkTechno/以下には、アットマークテクノの製品名が並びます。プロダクトディレクトリの下には、ビルドの動作を決める Makefile や、ビルド時にデフォルト値として使われる config ファイル、アプリケーションに必要な設定ファイルなど、ビルドシステム内でプロダクトごとに異なる部分が含まれています。

ここでは、デフォルトの設定に使われる config ファイル群と Makefile について説明します。

8.1. config.arch

config.arch ファイルには、アーキテクチャに依存した設定を記述します。実際には、アーキテクチャごとにデフォルトの値がすでに用意されているため、上書きする設定だけを書くようになっています。

設定できる変数は以下のとおりです。

- CPUFLAGS

  CPU のコンパイルフラグを指定することができます。

- VENDOR_CFLAGS

  ベンダー依存の CFLAGS フラグを指定することができます。

- DISABLE_XIP

  XIP (Execute In Place)を無効にする場合、1 を指定します。

- DISABLE_SHARED_LIBS

  共有ライブラリを無効にする場合、1 を指定します。

- DISABLE_MOVE_RODATA

  読み込み専用領域の移動を無効にする場合、1 を指定します。

- LOPT

  ライブラリをコンパイルするときに、コンパイラに渡すオプションを指定します。

- UOPT

  ユーザーランドアプリケーションをコンパイルするときに、コンパイラに渡すオプションを指定します。

- CONSOLE_BAUD_RATE

  シリアルコンソールのボーレートを指定します。

8.2. config.linux-2.4.x

`config.linux-2.4.x` は、Linux カーネルのビルドシステムが生成する `.config` を、別の名前でプロダクトディレクトリに保存しています。このファイルは、カーネルコンフィギュレーションのデフォルト値として扱われます。カーネルの値を変更し、変更したものをデフォルト値としたい場合は、このファイルを上書きします。また、「7.3.9. Update Default Vendor Settings（デフォルトベンダ設定の更新）」 メニューを使うことでも可能です。

8.3. config.vendor

`config.vendor` は、ユーザランドアプリケーション等の情報が含まれています。uClinux-dist のビルドシステムが menuconfig ターゲットなどで設定された情報を保持しています。このファイルは、`uClinux-dist/config/.config` のコピーになっています。`config.linux-2.4.x` と同様に上書きすることで、デフォルトの値を変更することができます。

8.4. config.uClibc

`config.uClibc` は、uClibc の設定ファイルです。uClinux-dist のメニューから uClibc の変更をすることは、現在できません。uClibc の設定を変更する場合は、uClibc のディレクトリに移動し、uClibc 専用のコンフィグツールを使います。uClibc の設定方法も uClinux-dist と同じく Linux カーネルビルドシステムを使用しています。

8.5. Makefile

`Makefile` では、実際のイメージファイルの生成を制御します。デバイスファイルやプログラムのインストール先、プログラムが必要としている設定ファイルやデータファイルのインストール先、チェックサムの生成などです。

9.romfs インストールツール

コンパイルされたアプリケーションや各種設定ファイルは、Makefile の romfs ターゲットによって uClinux-dist/romfs ディレクトリにインストールされます。アプリケーションが必要とする設定ファイルやデータファイルなども、この時点で uClinux-dist/romfs ディレクトリにインストールされます。

ディレクトリ名に romfs という名前が使われているのは、多くの組込みシステムでは、デスクトップやサーバ用途の Linux システムで使われている ext2 や ext3、 reiserfs、xfs などではなく romfs が使われるためです。しかし、romfs ディレクトリ自体は romfs に依存しているわけではありません。後述する jffs2 などでも同じ romfs ディレクトリを使います。

romfs ディレクトリの構成は、ターゲットシステムが起動したときにターゲットシステム上で見えるディレクトリ構成と同じ構成になっています。romfs ディレクトリをルートディレクトリとして、その下に bin や dev、etc などのディレクトリが配置されます。

uClinux-dist には romfs ディレクトリにファイルを簡単にインストールするために romfs-inst.sh というファイル名のスクリプトが用意されています。このスクリプトは uClinux-dist/tools ディレクトリに入っています。romfs-inst.sh は、uClinux-dist ディレクトリにある Makefile によって ROMFSINST という変数に代入されます。このため、プロダクトディレクトリをはじめとする uClinux-dist 内の各ディレクトリの Makefile では romfs-inst.sh がどこに存在しているかを気にせず、ROMFSINST という変数で使うのが一般的な方法になっています。

9.1. 概要

romfs-inst.sh は、romfs ディレクトリを指定する環境変数 ROMFSDIR が設定されていない場合、簡易版のヘルプを出力します。

例 9.1. romfs-inst.sh のヘルプ

```
[PC ~/uClinux-dist]$tools/romfs-inst.sh
ROMFSDIR is not set
tools/romfs-inst.sh: [options] [src] dst
    -v          : output actions performed.
    -e env-var  : only take action if env-var is set to "y".
    -o option   : only take action if option is set to "y".
    -p perms    : chmod style permissions for dst.
        -d          : make dst directory if it doesn't exist
        -S          : don't strip after installing
    -a text     : append text to dst.
        -A pattern  : only append text if pattern doesn't exist in file
    -l link     : dst is a link to 'link'.
    -s sym-link : dst is a sym-link to 'sym-link'.

    if "src" is not provided,  basename is run on dst to determine the
    source in the current directory.

multiple -e and -o options are ANDed together.  To achieve an OR affect
use a single -e/-o with 1 or more y/n/"" chars in the condition.

if src is a directory,  everything in it is copied recursively to dst
with special files removed (currently CVS and Subversion dirs).
```

romfs-inst.sh のコマンド構文は以下のとおりです。

例 9.2. romfs-inst.sh 構文

```
romfs-inst.sh [options] [src] dst
```

[]の部分は省略することができます。もし、src が指定されなかった場合、**basename** コマンドが dst に適用されて、戻り値を src として使います。romfs-inst.sh はその値を現在のディレクトリ（つまりプロダクト Makefile の場合はプロダクトディレクトリ）から探します。

```
[PC ~]$ basename /foo/bar
bar
```

もし、src がディレクトリの場合は、そのディレクトリ以下すべてのファイルとディレクトリをインストールします。ただし、CVS ディレクトリはコピーされません。

以下はオプションの簡単な説明です。

- -v

  実際に実行した内容を出力

- -e env-var

  env-var が "y"のときだけ、指定されたアクションを実行

- -o option

option が "y"のときだけ、指定されたアクションを実行

- -p perms

  chmod 方式で dst のパーミッションを指定

- -a text [-A pattern]

  text を dst に追加。-A pattern が指定されているときは、pattern が dst に含まれていない場合に text を追加

- -l link

  dst で指定された名前で、link へのハードリンクを作成

- -s sym-link

  dst で指定された名前で、sym-link へのシンボリックリンクを作成

以降の章では、`romfs-inst.sh` の使用例を説明します。

9.2. ファイルのインストール

ファイルをインストールする場合は、以下のように romfs ターゲットを `Makefile` に記述します。

```
romfs:
  $(ROMFSINST) src.txt /etc/dst.txt
```

これは、プロダクトディレクトリ内にある `src.txt` を `romfs` ディレクトリの中の `/etc/dst.txt` にインストールすることを意味しています。もし、`romfs` ディレクトリが `/home/myname/uClinux-dist/romfs` であれば、

`/home/myname/uClinux-dist/romfs/etc/dst.txt` というファイルができあがります。

プロダクトディレクトリにある `src.txt` の名前を変更して `dst.txt` としておくことで、以下のように簡単に記述することができます。

```
$(ROMFSINST) /etc/dst.txt
```

上で簡単にふれましたが、src が省略されたとき `romfs-inst.sh` は dst の basename を使ってカレントディレクトリからファイルを探します。`/etc/dst.txt` の basename は、`dst.txt` なので上記の文は

```
$(ROMFSINST) dst.txt /etc/dst.txt
```

と同じ意味を持ちます。

9.3. ディレクトリのインストール

ターゲットデバイスに多くのファイルをインストールする場合は、ディレクトリごとインストールすると簡単です。たとえば、ターゲットの/etc ディレクトリに多くの設定ファイルをインストールする場合などです。

プロダクトディレクトリに etc というディレクトリを作成し、必要なファイルを置きます。そして Makefile に以下のように記述します。

```
$(ROMFSINST) /etc
```

この例でも src が省略されているので、romfs-inst.sh は dst の basename を使います。/etc の basename は etc なので、romfs-inst.sh はプロダクトディレクトリにある etc というファイルまたはディレクトリを探します。そして今回のようにディレクトリの場合、romfs-inst.sh はディレクトリ内にあるファイルも一緒にインストールしてくれます。

以下のように tree コマンドを使うと簡単に確認できます。

```
[PC ~/uClinux-dist]$ tree ~/uClinux-dist/vendors/AtmarkTechno/test/etc
:
[PC ~/uClinux-dist]$ tree ~/uClinux-dist/romfs/etc
    :
```

もちろん、保存している名前とは別の名前でインストールすることも可能です。

```
$(ROMFSINST) /etc /var
```

このコマンドでは、プロダクトディレクトリにある etc というディレクトリを romfs/var にインストールします。

9.4. リンクの作成

romfs-inst.sh を使って簡単にリンクを作成することができます。ただし、hard link と symbolic link をきちんと理解しなければいけません。シンボリックリンクを作成するときは、オプション '-s'を使います。例として a.txt へのシンボリックリンクを作成してみます。プロダクト Makefile の romfs ターゲットは以下のようになります。

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) /a.txt
    $(ROMFSINST) -s a.txt /b.txt
```

```
[PC ~/uClinux-dist]$ make romfs
    :
    :
[PC ~/uClinux-dist]$ ls -l romfs
total 0
```

```
-rw-r--r--  1 atmark atmark 0 Sep 24 05:43 a.txt
lrwxrwxrwx  1 atmark atmark 5 Sep 24 05:43 b.txt -> a.txt
```

次は hard link の例です。オプションは'-l'を使います。

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) /a.txt
    $(ROMFSINST) -l a.txt /b.txt
```

```
[PC ~/uClinux-dist]$ make romfs
    :
    :
[PC ~/uClinux-dist]$ ls -i1 romfs
6077732 a.txt
6296750 b.txt
[PC ~/uClinux-dist]$ ls -i1 vendors/AtmarkTechno/test/a.txt
6296750 vendors/ATmarkTechno/test/a.txt
```

romfs 内にできた b.txt は、romfs 内の a.txt へのハードリンクではなく、プロダクトディレクトリにある a.txt へのハードリンクということが inode の番号によってわかります。ハードリンクの使用は混乱を招きますので、よほど HDD の容量が不足していないかぎりお勧めしません。現在の uClinux-dist でも、romfs 内へのハードリンクはあまり使われていないようです。

9.5. ファイルへの情報追記

romfs-inst.sh を使うことで、すでに存在するファイルに簡単に情報を追記することができます。

構文は以下のようになります。

```
$(ROMFSINST) -a "文字列" romfs ディレクトリ内のファイル名
```

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) -a 'Hello' /a.txt
    $(ROMFSINST) -a 'World' /a.txt
```

```
[PC ~/uClinux-dist]$ make romfs
    :
    :
[PC ~/uClinux-dist]$ cat romfs/a.txt
Hello
World
```

9.6. 条件実行

romfs-inst.sh は条件による実行制御が可能です。

```
$(ROMFSINST) -e 変数名 実行するコマンド
```

変数名としてよく用いられるのは、CONFIG_ではじまる環境変数です。

```
romfs:
    [ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
    $(ROMFSINST) -e CONFIG_DEFAULTS_ATMARKTECHNO -a 'Hello' /a.txt
    $(ROMFSINST) -e CONFIG_DEFAULTS_UNKNOWN -a 'World' /a.txt
```

```
[PC ~/uClinux-dist]$make romfs
    :
    :
[PC ~/uClinux-dist]$cat romfs/a.txt
Hello
```

CONFIG_DEFAULTS_UNKNOWN は定義されていないので条件に当てはまらず、a.txt に「World」の文字列は書き出されません。CONFIG_DEFAULTS_ATMARKTECHNO は、ベンダー名で AtmarkTechno を選択すると、atmark-dist/.config に定義されます。

10.新規アプリケーションの追加方法

この章では、ターゲットボードで動作する新しくアプリケーションを作成する方法と、作成したアプリケーションを uClinux-dist 内に追加する方法を説明します。

10.1. Out of Tree コンパイル

Out of Tree コンパイルは、atmark-dist に変更を加えることなく手軽に開発を行うことができる方法です。uClinux-dist のビルドシステムを使うため、複雑な `Makefile` を書く必要もありません。uClinux-dist のディレクトリ構造を木に見たて、そのディレクトリ外でコンパイルするためにこう呼ばれています。

ここでは実際にターゲットボード用の"Hello World"を作成します。

10.1.1. 準備

Out of Tree コンパイルでは uClinux-dist に入っているビルドシステムやライブラリ群を使うために、一度ビルドされている uClinux-dist が必要です。まず、uClinux-dist がターゲットボード用にコンフィギュレーションかつビルドされていることを確認してください。

10.1.2. ソースコードの用意

次に、開発するアプリケーション用のディレクトリを uClinux-dist のディレクトリ構造の外に作ってください。この中には `Makefile` と必要な C のソースコードやヘッダファイルが入ります。

```
[PC ~]$ ls
uClinux-dist
[PC ~]$ mkdir hello
[PC ~]$ ls
hello  uClinux-dist
[PC ~]$ ls hello
Makefile  hello.c
```

`hello.c` は、以下のように記述します。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        printf("Hello World\n");
        return 0;
}
```

`Makefile` は以下を使用します。

```
#ROOTDIR=/usr/src/uClinux-dist                                    ❶
ifndef ROOTDIR
ROOTDIR=../uClinux-dist
endif
```

```
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch


EXEC = hello                                                    ❷
OBJS = hello.o                                                  ❸


all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
      $(CC) -c $(CFLAGS) -o $@ $<
```

この Makefile は"Hello World"以外のアプリケーションを開発するときにもテンプレートとして使用することができます。環境に合わせて変更する点は以下の 3 つです。

❶ ROOTDIR が指定されていない場合、現在のディレクトリと並列に uClinux-dist ディレクトリがあると仮定します。他の場所に uClinux-dist がある場合は、この行のコメントを外して適切なディレクトリ名に変更してください。

❷ 生成される実行ファイル名を指定します。ここでは、hello とします。

❸ 生成される実行ファイルが依存するオブジェクトファイルを指定します。ここでは hello.o を指定します。

10.1.3. ビルド（uClinux）

Makefile と hello.c が用意できたら、hello をビルドします。ビルドには **make** コマンドを使用します。ビルドが完了すると実行ファイル hello がディレクトリ内に生成されます（uClinux と Linux の場合では、生成されるファイルが異なります）。

```
[PC ~/hello]$ make
    :
    :
[PC ~/hello]$ ls hello*
hello  hello.c  hello.gdb  hello.o
[PC ~/hello]$ file hello*
hello:     BFLT executable - version 4 ram
hello.c:   ASCII C program text
```

```
hello.gdb: ELF 32-bit MSB executable, version 1 (SYSV), statically linked, not
stripped
hello.o:   ELF 32-bit MSB relocatable, version 1 (SYSV), not stripped
```

hello.gdb は ELF フォーマットの実行ファイルです。デバッグに使います。hello は Flat Binary フォーマットと呼ばれる uClinux 専用のバイナリフォーマットです。

10.1.4. インストール

実行ファイルを uClinux の romfs ディレクトリにインストールするために、**make** コマンドで romfs ターゲットを指定します。

```
[PC ~/hello]$ make romfs
romfs-inst.sh /bin/hello
[PC ~/hello]$ ls ../uClinux-dist/romfs/bin/hello
hello
```

10.1.5. image ファイルの作成

make romfs の後 uClinux-dist のディレクトリに移動して、**make image** を実行することで、hello を含んだ ターゲットボード用のイメージファイルが image ディレクトリに生成されます。

```
[PC ~/hello]$ cd ../uClinux-dist
[PC ~/uClinux-dist]$ make image
    :
    :
[PC ~/uClinux-dist]$ ls images
image.bin  linux.bin  romfs.img
```

image ターゲットについては、「7.7.5. image」を参照してください。

10.1.6. 複数のソースコード

実行ファイルが複数のソースコードに分割されている場合は、Makefile を変更することで対応できます。ここでは、hello.c と print.c から実行ファイル hello を生成する場合を例に説明します。

```
[PC ~/hello]$ ls
Makefile  hello.c  print.c
```

hello.c

```
#include <stdio.h>

extern void print_hello(char *string);

int main(int argc, char *argv[])
{
        print_hello("World");
        return 0;
}
```

print.c

```
#include <stdio.h>
void print_hello(char *string)
{
        printf("Hello %s\n", string);
}
```

Makefile の OBJS に print.o を追加します。

```
# ROOTDIR=/usr/src/uClinux-dist
ifndef ROOTDIR
ROOTDIR=../uClinux-dist
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch


EXEC = hello
OBJS = hello.o print.o                                    ----- print.o を追加


all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

```
[PC ~/hello]$ make
    :
    :
[PC ~/hello]$ ls
Makefile  hello  hello.c  hello.gdb  hello.o  print.c  print.o
```

10.1.7. pthread 対応

スレッドを使うアプリケーションの場合は、スレッド用のライブラリをリンクする必要があります。Makefile の一部を以下のように変更します。

```
# ROOTDIR=/usr/src/uClinux-dist
ifndef ROOTDIR
ROOTDIR=../uClinux-dist
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch


EXEC = hello
OBJS = hello.o

all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LIBPTHREAD) $(LDLIBS)     ❶

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

❶ $(LIBPTHREAD)を追加する

10.2. プロダクト別のアプリケーション

組み込み機器の場合、プロダクト固有のアプリケーション開発がよく行われます。ここでは、プロダクト固有のアプリケーションを uClinux-dist に統合する方法を紹介します。

10.2.1. ディレクトリの準備

uClinux-dist/vendors/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)/下に、アプリケーション用のディレクトリを作成します。$(CONFIG_VENDOR)と$(CONFIG_PRODUCT)は、お使いのボードにあわせて変更してください。ここでは、ベンダーを AtmarkTechno、プロダクトを SUZAKU-S.SZ130、そしてアプリケーションを hello とします。

```
[PC ~]$ mkdir uClinux-dist/vendors/AtmarkTechno/SUZAKU-S.SZ130/hello
[PC ~]$ ls -d uClinux-dist/vendors/AtmarkTechno/SUZAKU-S.SZ130/hello
hello
```

10.2.2. ソースコードの用意

C のソースコードは前項で使った hello.c を使用します。Makefile は Out of Tree コンパイルで使用したものよりもシンプルになります。

```
EXEC = hello
OBJS = hello.o


all: $(EXEC)

$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

10.2.3. 追加アプリケーションの設定

追加したアプリケーションをプロダクト Makefile に追加します。プロダクト Makefile には、SUBDIR_y と呼ばれる変数が用意されています。ここに、アプリケーションのディレクトリ名を追加します。

```
    :
    :
SUBDIR_y += hello
    :
    :
```

10.2.4. ビルド

ビルド方法は「7.6. ビルド」と同じです。all、clean、romfs ターゲットは、以下のように定義されているので、SUBDIR_y で指定したディレクトリすべてに対して、適切なコマンドを実行するようになっています。

```
all:
    for i in $(SUBDIR_y) ; do $(MAKE) -C $$i || exit $? ; done

clean:
    -for i in $(SUBDIR_y) ; do [ ! -d $$i ] || $(MAKE) -C $$i clean; done

romfs:
    :
    :
    for i in $(SUBDIR_y) ; do $(MAKE) -C $$i romfs || exit $? ; done
```

10.3. user ディレクトリへのマージ

最初はプロダクト固有として開発されたアプリケーションの中には、多くのプロダクトで使われるようになることがあります。そんな時は、アプリケーションを user ディレクトリに移動し、プロダクト間で共有することができます。

10.3.1. ディレクトリの準備

uClinux-dist/user 以下に、アプリケーション用のディレクトリを作成します。ここでは、hello とします。

```
[PC ~]$ mkdir uClinux-dist/user/hello
[PC ~]$ ls -d uClinux-dist/user/hello
hello
```

10.3.2. ソースコードの用意

C のソースコードおよび Makefile は「10.2. プロダクト別のアプリケーション」と同じものを使用します。

10.3.3. 追加アプリケーションの設定

変更個所は、uClinux-dist/config/config.in と uClinux-dist/user/Makefile です。この例では、追加するアプリケーションを Miscellaneous Application に追加します。他にならってアルファベット順に並べます。

例 10.1. uClinux-dist/config/config.in の変更点

```
--- config.in.orig    2004-04-18 04:03:57.000000000 +0900
+++ config.in    2004-05-26 17:58:38.000000000 +0900
@@ -515,6 +515,7 @@
 bool 'gdbreplay'        CONFIG_USER_GDBSERVER_GDBREPLAY
 bool 'gdbserver'        CONFIG_USER_GDBSERVER_GDBSERVER
 bool 'hd'            CONFIG_USER_HD_HD
+bool 'hello'            CONFIG_USER_HELLO_HELLO
 bool 'lcd'            CONFIG_USER_LCD_LCD
 bool 'ledcon'            CONFIG_USER_LEDCON_LEDCON
 bool 'lilo'            CONFIG_USER_LILO_LILO
```

例 10.2. uClinux-dist/user/Makefile の変更点

```
--- Makefile.orig       2004-02-20 13:22:55.000000000 +0900
+++ Makefile    2004-05-26 17:56:09.000000000 +0900
@@ -123,6 +123,7 @@
 dir_$(CONFIG_USER_GDBSERVER_GDBSERVER)      += gdbserver
 dir_$(CONFIG_USER_GETTYD_GETTYD)            += gettyd
 dir_$(CONFIG_USER_HD_HD)                    += hd
+dir_$(CONFIG_USER_HELLO_HELLO)              += hello
 dir_$(CONFIG_USER_HOSTAP_HOSTAP)            += hostap
 dir_$(CONFIG_USER_HTTPD_HTTPD)              += httpd
 dir_$(CONFIG_USER_HWCLOCK_HWCLOCK)          += hwclock
```

10.3.4. アプリケーションの選択

make の menuconfig ターゲットなどで追加したアプリケーションが Miscellaneous Application セクションに表示されるか確認してください。表示された hello を選択し、設定を保存します。

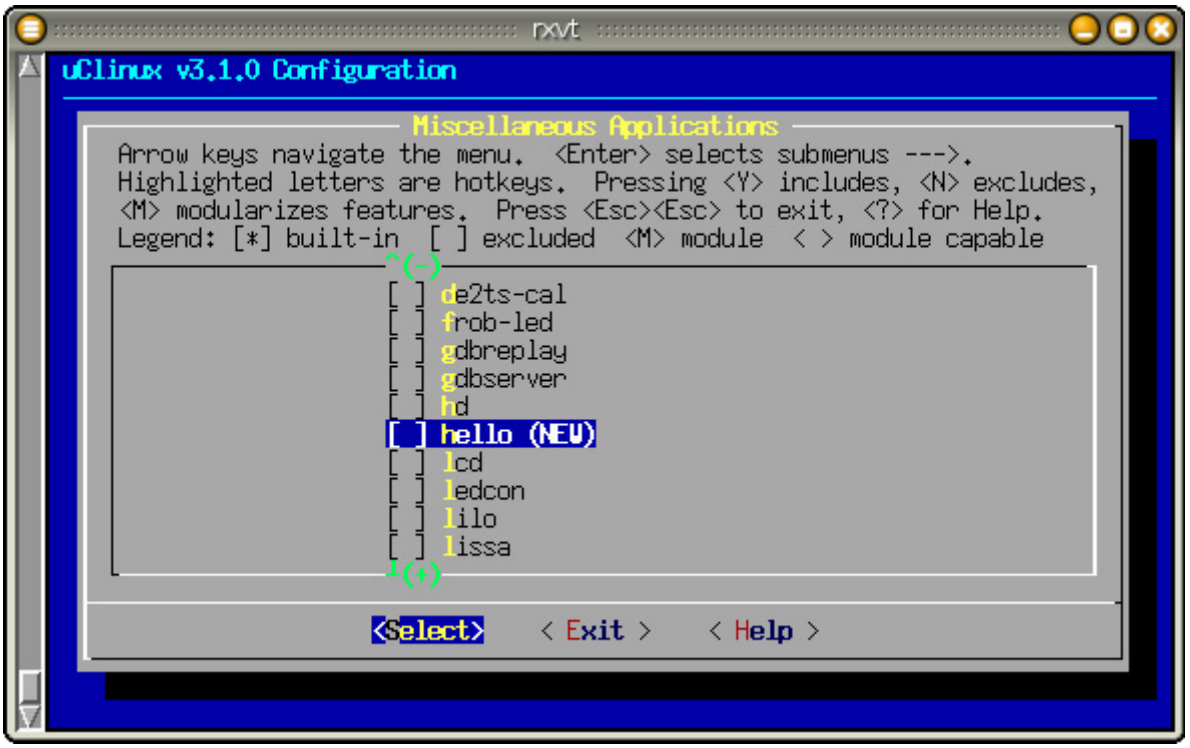


図 10.1. メニューに追加された hello

10.3.5. ビルド

In Tree コンパイルのビルド方法は「7.6. ビルド」と同じです。生成されたイメージファイルをターゲットボードに転送し、hello が動作するか確認してください。

10.3.6. コンフィグの命名規則

今回使用したコンフィグ名は CONFIG_USER_HELLO_HELLO です。

uClinux-dist ではユーザランドアプリケーションの選択にはディレクトリ名とアプリケーション名を使う決まりになっています。

- すべてのコンフィグオプションは、CONFIG_からはじめる
- uClinux-dist/user 下のものは、CONFIG_USER_からはじめる
- ディレクトリ名が hello なので、CONFIG_USER_HELLO_からはじめる
- 最後にアプリケーション名をつけて CONFIG_USER_HELLO_HELLO となる

10.3.7. 複数のアプリケーション

ひとつのディレクトリで複数のアプリケーションを開発する方法を紹介します。In Tree コンパイルに限った話ではないので、Out of Tree コンパイルでも可能です。

`uClinux-dist/user/hello` のディレクトリに `hello2` という名前のアプリケーションを追加します。コンフィグの文字列は、CONFIG_USER_HELLO_HELLO2 とします。

まずは、uClinux-dist/config/config.in と、uClinux-dist/user/Makefile の変更点です。

例 10.3. **uClinux-dist/config/config.in の変更点(複数アプリケーション)**

```
--- config.in.orig    2004-04-18 04:03:57.000000000 +0900
+++ config.in    2004-05-26 17:58:38.000000000 +0900
@@ -515,6 +515,8 @@
 bool 'gdbreplay'        CONFIG_USER_GDBSERVER_GDBREPLAY
 bool 'gdbserver'        CONFIG_USER_GDBSERVER_GDBSERVER
 bool 'hd'            CONFIG_USER_HD_HD
+bool 'hello'            CONFIG_USER_HELLO_HELLO
+bool 'hello2'            CONFIG_USER_HELLO_HELLO2
 bool 'lcd'            CONFIG_USER_LCD_LCD
 bool 'ledcon'            CONFIG_USER_LEDCON_LEDCON
 bool 'lilo'            CONFIG_USER_LILO_LILO
```

例 10.4. **uClinux-dist/user/Makefile の変更点(複数アプリケーション)**

```
--- Makefile.orig       2004-02-20 13:22:55.000000000 +0900
+++ Makefile    2004-05-26 17:56:09.000000000 +0900
@@ -123,6 +123,8 @@
 dir_$(CONFIG_USER_GDBSERVER_GDBSERVER)      += gdbserver
 dir_$(CONFIG_USER_GETTYD_GETTYD)            += gettyd
 dir_$(CONFIG_USER_HD_HD)                    += hd
+dir_$(CONFIG_USER_HELLO_HELLO)              += hello
+dir_$(CONFIG_USER_HELLO_HELLO2)             += hello
 dir_$(CONFIG_USER_HOSTAP_HOSTAP)            += hostap
 dir_$(CONFIG_USER_HTTPD_HTTPD)              += httpd
 dir_$(CONFIG_USER_HWCLOCK_HWCLOCK)          += hwclock
```

Makefile では、左側にコンフィグオプションを、+=の右側にはディレクトリ名を書きます。このため、hello も hello2 も指定するディレクトリ名は同じ hello になります。

続いて、uClinux-dist/user/hello/Makefile です。今回はアプリケーション名を変数に入れずに直接扱ってみます。

例 10.5. **Makefile**(複数アプリケーション)

```
OBJS_HELLO = hello.o
OBJS_HELLO2 = hello2.o


all: hello hello2

hello: $(OBJS_HELLO)
    $(CC) $(LDFLAGS) -o $@ $(OBJS_HELLO2) $(LDLIBS)

hello2: $(OBJS_HELLO2)
    $(CC) $(LDFLAGS) -o $@ $(OBJS_HELLO2) $(LDLIBS)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
    $(ROMFSINST) -e CONFIG_USER_HELLO_HELLO /bin/hello
    $(ROMFSINST) -e CONFIG_USER_HELLO_HELLO2 /bin/hello2


%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $<
```

romfs ターゲットで、"-e CONFIG_USER_HELLO"を使っている点に注目してください。

uClinux-dist のビルドシステムは、hello ディレクトリにあるアプリケーションがひとつでも選択されると、ディレクトリ内すべてのアプリケーションをビルドするように指定します。このため、romfs ターゲットで条件によってインストールするアプリケーションを選択しなければなりません。上記のように、コンフィグオプションによる条件分岐を行い、インストールするアプリケーションを決定します。

romfsinst.sh の詳しい説明は、「9. romfs インストールツール」の使い方を参照してください。

11.新規デバイスドライバの追加方法

この章では、ターゲットボードで動作する uCLinux カーネルにデバイスドライバを新しく作成する方法と、作成したデバイスドライバを uClinux-dist 内に追加する方法を説明します。

11.1. Out of Tree コンパイル

Out of Tree コンパイルは、uClinux-dist に変更を加えることなく手軽に開発できる方法です。uClinux-dist のビルドシステムを使うため、複雑な `Makefile` を書く必要もありません。`uClinux-dist` のディレクトリ構造を木に見たて、そのディレクトリ外でコンパイルするためにこう呼ばれています。

なお、linux-2.4.x では、デバイスドライバの Out of Tree コンパイルがサポートされていません。ここで紹介する方法は、In Tree コンパイルと同じ動作をするように `Makefile` で調整したものです。

以下に、仮想の（キャラクタ）デバイスドライバを例に作成方法を説明します。

11.1.1. 準備

Out of Tree コンパイルでは、uClinux-dist に含まれるビルドシステムやライブラリ群を使うため、一度ビルドされている uClinux-dist が必要です。まず、uClinux-dist がターゲットボード用にコンフィギュレーションかつビルドされていることを確認してください。

11.1.2. ソースコードの用意

次に、開発するデバイスドライバ用のディレクトリを `uClinux-dist` のディレクトリ構造の外に用意します。この中には、`Makefile` と必要な C のソースコードやヘッダファイルを配置します。

```
[PC ~]$ ls
uClinux-dist
[PC ~]$  mkdir smsg
[PC ~]$ ls
uClinux-dist smsg
[PC ~]$ ls smsg
Makefile  smsg.c
```

`smsg.c` は、以下を使用します。

```
/**
* Character Device Driver Sample:
* file name: smsg.c
*/
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/string.h>
#include <asm/uaccess.h>
static int driver_major_no = 0;
static char *msg = "Hello, everyone.";
MODULE_PARM(msg, "s");
```

```
/* デバイスファイルオープン時に実行 */
static int smsg_open(struct inode *inode, struct file *filp)
{
    printk("smsg_open\n");
    return 0;
}

/* デバイスファイル読み取り時に実行 */
static int smsg_read(struct file *filp, char *buff, size_t count, loff_t *pos)
{
    int len;
    printk("smsg_read: msg = %s\n", msg);
    len = strlen(msg);
    copy_to_user(buff, msg, len);
    return 0;
}

/* デバイスファイルクローズ時に実行 */
static int smsg_release(struct inode *inode, struct file *filp)
{
    printk("smsg_release\n");
    return 0;
}

/* ファイル操作定義構造体 */
static struct file_operations driver_fops = {
    .read    = smsg_read,
    .open    = smsg_open,
    .release = smsg_release,
};

/* インストール時に実行 */
int init_module(void)
{
    int ret;
    printk("smsg: init_module: msg = %s\n", msg);

    /* キャラクタ型ドライバ管理テーブルへ登録 */
    ret = register_chrdev(driver_major_no, "smsg", &driver_fops);

    /* 登録エラー */
    if (ret < 0) {
        printk("smsg: Major no. cannot be assigned.\n");
        return ret;
    }

    /* 最初に登録する場合 */
    if (driver_major_no == 0) {
        dirver_major_no = ret;
        printk("smsg: Major no. is assigned to %d.\n", ret);
    }

    return 0;
}

/* アンインストール時に実行 */
```

```
void cleanup_module(void)
{
    printk("smsg: cleanup_module\n");

    /* キャラクタ型ドライバ管理テーブルから削除 */
    unregister_chrdev(driver_major_no, "smsg");
}
```

Makfile は、以下を使用します。

```
MODULES = smsg.o                                                    ❶

ifdef UCLINUX_BUILD_KMODULE

obj-m = $(MODULES)
include $(TOPDIR)/Rules.make

else

ifndef ROOTDIR
ROOTDIR = /home/atmark/uClinux-dist                                 ❷
endif

ROMFSDIR = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_KMODULE = 1
include $(ROOTDIR)/.config
include $(ROOTDIR)/config.arch

all:
    make -C $(ROOTDIR)/linux-2.4.x SUBDIRS=`pwd` modules

romfs:
   make -C $(ROOTDIR)/linux-2.4.x INSTALL_MOD_PATH=$(ROMFSDIR) DEPMOD="$(ROOTDIR)/
user/busybox/examples/depmod.pl -k vmlinux" SUBDIRS=`pwd` MOD_DESTDIR=drivers/
char modules_install

clean:
    -rm -f $(MODULES) *~

endif
```

この Makefile は、他のデバイスドライバを開発するときにもテンプレートとして使用することができます。環境に合わせて変更する点は、以下の 2 つです。

❶ 生成されるモジュールファイル名を指定します。

❷ ROOTDIR には、uClinux-dist ディレクトリを指定します。

11.1.3. ビルド（uClinux）

Makefile と smsg.c の用意ができたら、smsg.o をビルドします。ビルドには **make** コマンドを使用します。ビルドが完了するとモジュールファイル smsg.o がディレクトリ内に生成されます。

```
[PC ~/smsg]$ make
  :
[PC ~/smsg]$ ls
Makefile  smsg.c  smsg.o
```

11.1.4. インストール

モジュールファイルを uClinux-dist の romfs ディレクトリにインストールするために、**make** コマンドで romfs ターゲットを指定します。

```
[PC ~/smsg]$ make romfs
  :
[PC ~/smsg]$ ls ../uClinux-dist/romfs/lib/modules/2.4.32-uc0/kernel/ drivers/char/
smsg.o
```

11.1.5. **image** ファイルの作成

make romfs を実行後、uClinux-dist のディレクトリに移動して、**make image** を実行することで、smsg.o モジュールを含んだターゲットボード用のイメージファイルが image ディレクトリに生成されます。

```
[PC ~/smsg]$ cd ../uClinux-dist
[PC ~/uClinux-dist]$ make image
[PC ~/uClinux-dist]$ ls images
image.bin  linux.bin  romfs.img
```

image ターゲットについては、「7.7.5. image」を参照してください。

11.2. drivers ディレクトリへのマージ

作成したデバイスドライバをディストリビューションに含める方法について説明します。デバイスドライバは、linux-2.4.x ディレクトリの drivers ディレクトリにまとめられています。この下には、デバイスをさらにカテゴリ分けし管理されています。

以下に、先に作成したキャラクタデバイスドライバ smsg を例にマージする手順を説明します。

11.2.1. ソースコードの用意

C のソースコードは、「11.1.2. ソースコードの用意」と同じものを使用します。ソースコード（smsg.c）は、uClinux-dist/linux-2.4.x/drivers/char ディレクトリに配置します。

11.2.2. 追加ドライバの設定

uClinux-dist/linux-2.4.x/drivers/char ディレクトリにある Config.in と Makefile を編集します。具体的な変更箇所は、以下のとおりです。

例 11.1. **uClinux-dist/linux-2.4.x/drivers/char/Config.in** の変更点

```
--- Config.in.orig    2007-02-14 14:20:00.000000000 +0900
+++ Config.in    2007-02-14 14:20:51.000000000 +0900
@@ -9,6 +9,8 @@
 # uClinux options
 #

+tristate 'Smsg support' CONFIG_SMSG
+
if [ "$CONFIG_SUZAKU" = "y" ]; then
      bool 'SUZAKU Starter Kit' CONFIG_SUZAKU_STARTERKIT
      if [ "$CONFIG_SUZAKU_STARTERKIT" = "y" ]; then
```

例 11.2. **uClinux-dist/linux-2.4.x/drivers/char/Makefile** の変更点

```
--- Makefile.orig    2007-02-14 14:33:00.000000000 +0900
+++ Makefile    2007-02-14 14:34:00.000000000 +0900
@@ -248,6 +248,7 @@
 #
 # uClinux drivers
 #
+obj-$(CONFIG_SMSG) += smsg.o
 obj-$(CONFIG_EB67XDIP_QUADSER) += oki_ml67x_quad.o
 obj-$(CONFIG_MACH_EB67XDIP) += oki_ml67x.o
 obj-$(CONFIG_SERIAL_ML67XXXX) += serial_ml67xxxx.o
```

11.2.3. ドライバの選択

make menuconfig などで追加したドライバが、Character devices セクションに表示されるか確認します。表示された smsg ドライバを選択し、設定を保存します。

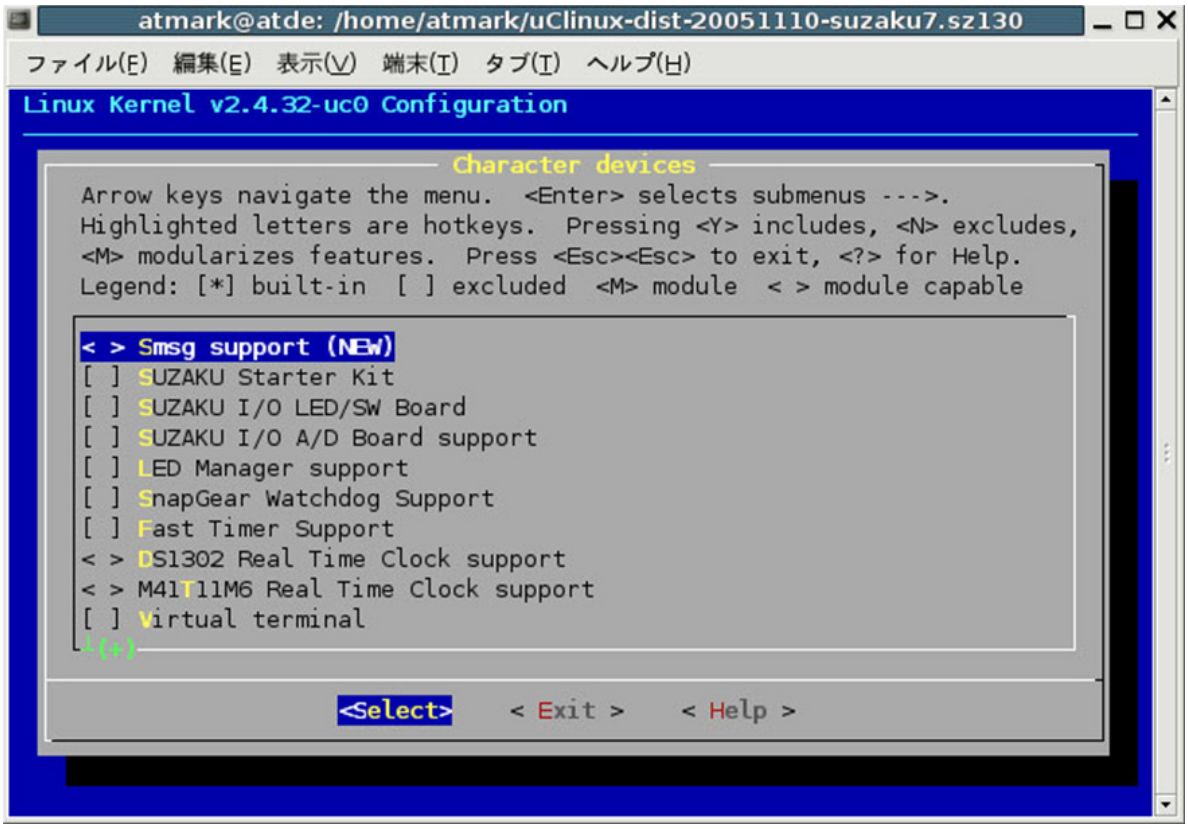


図 11.1. メニューに追加された message

11.2.4. ビルド

In Tree コンパイルのビルド方法は「7.6. ビルド」と同じです。

12.複数カーネルの利用

複数の Linux カーネルソースコードを使って開発することも可能です。その場合は、以下のようにシンボリックリンクを切り替えることで対応できます。

```
[PC ~]$ ls
kernel/   uClinux-dist/
[PC ~]$ ls kernel
linux-foo/   linux-bar/
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ ln -s ../kernel/linux-foo linux-2.4.x
[PC ~/uClinux-dist]$ ls -l linux-2.4.x
lrwxrwxrwx           linux-2.4.x -> ../kernel/linux-foo
```

13.特有なアプリケーションの説明

uClinux-dist には、一般の Linux には無いアプリケーションも含まれています。ここではその中でも特に組込み用途に適したアプリケーションを紹介します。

13.1. netflash

netflash はネットワーク経由でイメージファイルをダウンロードし、フラッシュメモリに書き込むためのアプリケーションです。組み込みシステムにおいて、ネットワーク経由によるシステムのアップグレードは、保守性とユーザの利便性の両面から非常に重宝されます。

netflash がファイルのダウンロードに利用できる通信プロトコルは、http、ftp、tftp です。このため、netflash を実行するには、http、ftp、tftp のいずれかのプロトコルで通信が行えるサーバが必要となります。

特にオプションを指定せずに netflash を実行した場合には、以下の手順でフラッシュメモリの書き込み処理が行われます。

1. 全プロセスの終了
2. 指定されたファイルを指定されたプロトコルでサーバからダウンロード
3. ダウンロードしたファイルのチェックサムを確認
4. フラッシュメモリへの書き込み
5. システムのリブート

オプションを指定する事で、これらの処理を細かく制御する事が可能です。参考として、コマンドの実行例と netflash のヘルプを以下に示します。

```
[Target /]# netflash http://embedded-server/images/image.bin
netflash: killing tasks...
...
netflash: got "http://embedded-server/images/image.bin", length=4194304
 :
 :
```

```
[Targt /]# netflash -h
usage: netflash [-bCfFhijklntuv?] [-c <console-device>] [-d <delay>] [-o <offset>]
[-r <flash-device>] [<net-server>] <file-name>

        -b      don't reboot hardware when done
        -C      check that image was written correctly
        -f      use FTP as load protocol
        -F      force overwrite (do not preserve special regions)
        -h      print help
        -i      ignore any version information
        -H      ignore hardware type information
        -j      image is a JFFS2 filesystem
```

```
        -k      don't kill other processes (or delays kill until
                after downloading when root filesystem is inside flash)
        -K      only kill unnecessary processes (or delays kill until
                after downloading when root filesystem is inside flash)
        -l      lock flash segments when done
        -n      file with no checksum at end (implies no version information)
        -p      preserve portions of flash segments not actually written.
        -s      stop erasing/programming at end of input data
        -t      check the image and then throw it away
        -u      unlock flash segments before programming
        -v      display version number
```

処理の途中でエラーが発生した場合は、フラッシュメモリへの書き込みを行わずに処理を中断します。書き込みの途中で電源断が発生した場合には、最悪システムが起動しなくなる可能性があるので、netflash を実行する際にはご注意ください。

13.2. flatfsd

組み込み機器では、設定情報を初期化することなくファームウェアだけをアップグレードする機能が良く求められます。この機能を実現するためには、ユーザ設定を保持するための小さな領域を割り当てたファイルシステムを構築するのが一般的です。

このような用途に適したファイルシステムに Flat Filesystem があります。Flat Filesystem は、1 セクタからファイルシステムを構築することができます。また、シンプルな構造であること、安定した動作実績があることが特長です。

このファイルシステムを実現するためのアプリケーションが flatfsd です。flatfsd は、ramfs にマウントされた`/etc/config` ディレクトリの内容を`/dev/flash/config` デバイスファイルに読み書きします。`/dev/flash/config` は、設定ファイルを保存すべきデバイスのメジャー番号とマイナー番号を指定して作成します。

```
[Target /]# ls -l /dev/flash/
crw-------    1 root     root      90,  14 Jan  1 09:00 config
crw-------    1 root     root      90,   6 Jan  1 09:00 image
[Target /]#
```

以前保存した設定ファイル情報を復元するためには、flatfsd にオプション(-r)を指定して実行します。-r は、`/dev/flash/config` 内の以前保存したファイル情報を読み出して、`/etc/config` にコピーするオプションです。

```
[Target /]# flatfsd -r
FLATFSD: created 6 configuration files (507 bytes)
[Target /]#
```

この時、記録されているチェックサムよりファイル情報の整合性を確認し、異常だった場合には、`/etc/default` の内容で`/etc/config` ディレクトリを初期化します。通常、システムの起動時に **`flatfsd -r`** コマンドを実行します。

設定ファイルに加えた変更をフラッシュメモリ内に記録するためには、flatfsd プロセスに SIGUSR1 シグナルを送信します。このため、起動時などに **`flatfsd`** コマンドを実行してプロセスを立ち上げておく必要があります。起動している flatfsd のプロセス ID は`/var/run/flatfsd.pid` ファイルから取得できます。以下の例では、**`killall`** コマンドを使用し、flatfsd にシグナルを送る方法です。

```
[Target /]# killall -USR1 flatfsd
[Target /]#
```

SIGUSR1 シグナルを受信した flatfsd プロセスは、/etc/config ディレクトリの内容を/dev/flash/config に書き込みます。この時、整合性の確認が行えるよう、チェックサムを計算して記録します。

uClinux-dist のコンフィギュレーションで flatfsd アプリケーションを選択すると、dhcpcd や passwd などのアプリケーションは、/etc ではなく/etc/config に設定ファイルを用意します。このため、設定の変更が次回起動時まで保存されています。

Flat Filesystem では flatfsd が/dev/flash/config デバイスにデータを書き込んでいる最中に電源が切断されると、保存していたデータを消失する可能性があります。

また、フラッシュメモリの書き込み保証回数は、およそ 10 万回ですので注意してください。

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2004/12/18	・初版作成
1.1.0	2004/12/28	・「3. デフォルトイメージのビルド」に、ビルド時のユーザ権限に関する注意事項を追加 ・「10.3.5. ビルド」のリンク先の表記を変更
1.2.0	2006/07/14	・SUZAKU シリーズに特化 ・uClinux-dist を uClinux-dist-20051110 ベースに改変
1.3.0	2007/02/16	・「11. 新規デバイスドライバの追加方法」を追加
1.4.0	2007/12/14	・uClinux-dist-20071214 の内容に合わせて修正
1.4.1	2008/09/26	・タイトルを英語表記からカタカナ表記に
1.4.2	2009/03/19	・参照先を記述する際の表記を統一 ・表記ゆれを修正
1.4.3	2009/07/17	・誤記修正 ・本文のレイアウト統一

uClinux-dist 開発者ガイド
Version 1.4.3-8d87fa8
2009/07/17

株式会社アットマークテクノ
060-0035 札幌市中央区北 5 条東 2 丁目 AFT ビル 6F TEL 011-207-6550 FAX 011-207-6570