urandom vol.3

ファミコンミニとLinux

Secure Grouping Protocol Using Mental Poker

urandom出版技術部活動報告

目次

1

ファミコンミニとLinux

1　はじめに

身の回りには様々な計算機が溢れていますが、全ての計算機はリバースエンジニアリングの対象たり得ます。2016 年 11 月 10 日に発売されたニンテンドークラシックミニファミリーコンピューター（ファミコンミニ）も例外ではありません。ファミコンミニの中身は U-boot + Linux という組み込み機器にありがちな構成で、ほどほどに未知の部分もあり、簡単な組み込み機器解析の練習題材としてちょうどいいのではないかと思います。カーネル以下のレイヤーが全て GPLv2 ライセンスのソフトウェアで構成されていてソースコードが公開されている点も、練習題材としてのメリットです。

この記事では、組み込み機器解析の一例として、ファミコンミニで自前ビルドの Linux カーネルを起動するまでに筆者（op）が辿った紆余曲折を記します。「部品・完成品ベンダーが提供するドキュメントとか SDK は無いけど、既知の情報と Open Source Software のソースコード（OSS ソースコード）を解析してどうにか自前ビルドの Linux を起動する」というものなので、本来の起動手順や用語と比べると正しくない記述が含まれるかもしれませんが、動かした者勝ちという精神でやっていきます。

2　警告

この記事に書かれているファミコンミニの分解等の行為を実践すると、製品保証が無効になったり、ファミコンミニが故障したりする可能性があります。記事内容の実践は、内容を理解できる人が、自己責任で行って下さい。

3　ソースコードの引用とログ出力等の編集について

この記事では、任天堂 Web サイトの OSS ソースコード配布ページ[*1]で配布されている NintendoEntertainmentSystemNESClassicEdition_OSS.zip[*2]からソースコードを引用します。レイアウトの関係上、ソースコードの意味が変わらない範囲で、一部インデントや空白文字を編集して引用しているものがあります。

この記事ではコンソールからのログ出力やソフトウェアの実行結果を引用しますが、長すぎる場合には途中の行を省略しています。省略した行は`.....`で置換しています。ソフトウェアの実行結果中の製品シリアル番号を筆者の意向で隠した部分がありますが、その部分は`<REMOVED>`で置換しています。

4　下調べと事前準備

4.1　ハードウェア構成

まずはハードウェア構成をおさらいします。手元のファミコンミニを分解して出て来た基板の写真（図 1）を示します。

図 1: ファミコンミニの基板写真

この基板上の主要部品から読み取れる型番と、型番で検索して分かった簡単なス

[*1] https://www.nintendo.co.jp/support/oss/
[*2] SHA-1: dbe31c88f2b8b96cb8aa9866e6c59a8ad8fbb628

6.6　Linux が起動しない問題 - 調査編

ログ（リスト 21）によると、異常に大きいメモリー領域（約 1536MiB）を確保しようとして死んでいることが分かります。コールスタックによれば、メモリーを確保している箇所は arch/arm/mach-sunxi/sys_config.c の 540 行目周辺です（リスト 22）。メモリー確保の失敗は、関数 `int __init script_init(void)`の中で `origin_main[i].sub_cnt`の値がおかしくなっていることが原因のようです。

リスト 22: Linux arch/arm/mach-sunxi/sys_config.c

```
538         /* allock memory for sub-keys */
539         main_key->subkey = SCRIPT_MALLOC(origin_main[i].sub_cnt*sizeof
                (script_sub_key_t));
540         main_key->subkey_val = SCRIPT_MALLOC(origin_main[i].sub_cnt*
                sizeof(script_item_u));
```

`origin_main[i].sub_cnt`の値が何に由来するのか調べると、同じファイルの 498 行目で定義している `script_hdr`に行き着きます。この `script_hdr`は仮想アドレス `SYS_CONFIG_MEMBASE`をそのままマップしたものです。

リスト 23: Linux arch/arm/mach-sunxi/sys_config.c

```
491 /*
492  * init script
493  */
494 int __init script_init(void)
495 {
496     int     i, j, count;
497
498     script_origin_head_t        *script_hdr = __va(SYS_CONFIG_MEMBASE
            );
```

次に `SYS_CONFIG_MEMBASE`の定義を調べます。`SYS_CONFIG_MEMBASE`で Linux のソースコード全体を検索すると複数ヒットしますが、ファミコンミニでの正解は arch/arm/mach-sunxi/include/mach/sun8i/memory-sun8iw5p1.h の定義です（リスト 24）。これを見ると、`SYS_CONFIG_MEMBASE == 0x43000000`であることが分かります。

リスト 24: Linux arch/arm/mach-sunxi/include/mach/sun8i/memory-sun8iw5p1.h

```
27 #define SYS_CONFIG_MEMBASE       (PLAT_PHYS_OFFSET + SZ_32M + SZ_16M)
            /* 0x43000000 */
```

2

Secure Grouping Protocol Using Mental Poker

1　はじめに

離れた所にいるプレイヤーがポーカーなどを公平な第三者なしにプレイするための方法として、*Mental Poker*[1] というものがあります。Mental Poker はシャッフル・ドロー・カードの公開という 3 つの操作を公平な第三者なしに提供する暗号技術です。一方で、Mental Poker ではグループ分けが必要である「人狼」といったゲームは公平な第三者（ゲームマスター）なしでプレイできないと考えられていました。人狼は「村人」と「人狼」にプレイヤーをグループ分けして行うゲームです。村人になったプレイヤーは他のプレイヤーが村人か人狼のどちらに所属しているのか判定できてはならない一方で、人狼になったプレイヤーはどのプレイヤーが人狼であってどのプレイヤーが村人であるかを識別できます。このような柔軟なグループ分けを公平な第三者なしで実行する方法がなかったため、人狼は従来 Mental Poker の道具だけでは実行できないと考えられていました。

ところが最近になって秘匿グループ分けプロトコル（*Secure Grouping Protocol*）[2] が発明されました。この秘匿グループ分けプロトコルを用いることで、人狼のように複雑なグループ分けが必要なゲームを公平な第三者なしで実現できるのではないだろうかと研究が進められています。

本稿ではまず Mental Poker について説明し、次に置換群の基礎を解説し、この置換群の性質を利用した秘匿グループ分けプロトコルについて説明します。そして Mental Poker と秘匿グループ分けプロトコルの関係について述べます。

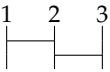
ただし、たとえばアリスの鍵 k_A などがボブに知られてはいけません。本稿ではゼロ知識証明の具体的な方法については割愛しますので、興味がある方は [3] や [4] などをご覧ください。

4　置換と置換群

秘匿グループ分けプロトコルの説明に入る前の準備として、まずは並び換えについて詳しく説明したいと思います。なお、ここに書かれている内容は [5] を参考にしました。

4.1　置換の定義と演算

並び換えは置換とも言い、これの直感的なイメージは次のような「あみだくじ」です。



上のあみだくじでは $1 \to 3, 2 \to 1, 3 \to 2$ となる置換です。以後、このようなあみだくじの図は番号を省略して次のように書きます。

$X := \{1, 2, \ldots, n\}$ のような有限集合 X が与えられたとき、この集合から X への全単射の写像 σ を X の n 次の置換と呼びます。また、$1 \to i_i, 2 \to i_2, \ldots n \to i_n$ という置換を次のように書きます。

$$\begin{pmatrix} 1 & 2 & \ldots & n \\ i_1 & i_2 & \ldots & i_n \end{pmatrix}$$

さて、このようなあみだくじが 2 つあるとします。

(1)

(2)

式（1）を σ、式（2）を ρ として、置換の積 $\sigma * \rho$ を次のように定義します。

$$(\sigma * \rho)(i) := \rho(\sigma(i))$$

3

urandom出版技術部活動報告

1　はじめに

本稿は urandom vol.2 の出版より導入した、出版向けの CI 的なシステムを紹介するものです。

そもそもそういった仕組みを導入するモチベーションは「いつでも、誰でも、最新の成果物を、最終的に入稿するものと全く同様の形で確認したい」というところにありました。urandom はこれまで（そして今回も）TEX を使って記事を執筆していますが、vol.1 の時点では今回紹介する仕組みは整えておらず、執筆者各自が手元の環境でビルドを試し、確認している状態でした。これによる問題点はいくつかありますが、特に問題となるのは環境によって結果が変わってしまうことです。例えば vol.1 で記事を書いた op、mayth、yyu の 3 名のうち、op だけ Linux を使用しており、それ以外は Mac を使っていました。Mac を使っている 2 名はフォントに Mac 同梱のヒラギノを使用していましたが、一方で op は IPA フォントを使用しました。これによって改ページ位置にズレが生じるなど、微妙な差異が出てきます。そして、入稿に使用しているのはヒラギノを使ってビルドした原稿です。そのため、op がビルドした原稿は実際に入稿される原稿とは異なるものになります。逆に言えば、mayth と yyu の 2 名だけが入稿される版をビルドすることができました。これは、この 2 名がビルド作業可能か否かによって入稿候補版をビルドできるかどうかが左右されてしまうという問題に繋がりました。出版 CI 的なシステムはこの問題を解決し、出版作業を円滑に進めるために導入したものです。

5　ビルドの自動化

5.1　全体の流れ

ここまでの手順で前提となる環境はできたので、これを使って自動ビルドシステムを構成していきます。

自動ビルドの流れとしては以下の通りです。

1. ユーザーが push すると GitBucket から Jenkins に通知される
2. 通知を受けた Jenkins はジョブを起動する
3. 起動されたジョブは push されたリポジトリを clone し、ビルドする
4. 生成された PDF を所定のサーバーにアップロードする
5. アップロードした PDF ファイルへの URL とビルド結果を Slack に投稿する

5.2　事前準備

GitBucket ではビルド対象となるリポジトリを用意します。まだビルドする中身がない場合は、テスト用の Makefile を用意しておきましょう（Listing 9）。

Listing 9: ビルド試験用 Makefile

```
.PHONY: hello
hello:
    @echo Hello world
```

Jenkins ではジョブを作成しておきます。新規ジョブ作成を選択し、ジョブ名を入力します。プロジェクトのタイプはフリースタイル・プロジェクトのビルドを選択します。適当な説明を入力しておいて、一旦これで完了しておきます。

5.3　Jenkins への通知とビルドの開始

GitBucket から Jenkins への通知には Service Hook の仕組みを使います。ビルド対象となるリポジトリの Settings を開き、Service Hooks タブを開きます。Add webhook ボタンをクリックし、Webhook を追加します。

Payload URL には `https://jenkins.example.com/github-webhook/` を設定し、Content type と Security Token はそのままにします。トリガーとなるイベントとしては Push があれば十分です。Test Hook をクリックすると Webhook のテ

urandom vol.3

発行者	urandom
表紙デザイン	秋弦めい
発行日	2016 年 12 月 31 日
バージョン	1.00（2016-12-12 01:38:39+09:00）
連絡先	https://blog.urandom.team/
印刷所	株式会社栄光

2016/12/31 COMIC MARKET 91
team urandom