CAS in Sandbox on mobile devices

福岡教育大学　　藤本光史 *1
Mitsushi Fujimoto
University of Teacher Education Fukuoka

Abstract

Sandbox is an environment for running software in a protected area and used to test unverified software. Termux for Android and iSH for iOS are distributed as terminal emulators with a sandbox environment. These provide a Linux CLI environment for mobile devices. In this article, we show that a computer algebra system Risa/Asir can be running on Android/iOS using them.

1　はじめに

私の研究目標の一つは教育現場で文房具として利用可能な数式処理システム (CAS) を搭載したモバイルデバイスの開発であり、その実現のためにこれまで様々なモバイルデバイスに数式処理システム Risa/Asir を移植してきた。移植対象として選んだプラットフォームはモバイルデバイスにおいて合わせて 9 割以上の世界シェアを持つ Android と iOS である。

Android については 2012 年から作業を開始し、CodeSourcery [1]、Scratchbox [2]、Arm rootfs + QEMU + chroot [3] と開発環境を模索し、arm64 対応 [4] も行った。これらは gc や pari などの外部ライブラリを静的リンクしたものであった。現在の Risa/Asir は gmp/mpfr/mpc という LGPL ライセンスのライブラリを動的リンクしており、この開発手法には問題がある。また、Android には多くのターミナルエミュレータアプリが存在しており、どれを用いたら良いのかという利用上の問題もある。

一方、iOS については 2007 年にリリースされた iPhoneOS 1.0 の時代から調査を開始した。iOS を通常の UNIX のように利用するには Jailbreak と呼ばれる改造行為が必要であり、これは開発元のアップルが認めていないため一般のユーザーには使えないという問題がある。

これらの問題に対する一つの解決策として、Sandbox 環境の利用を提案する。

2　Sandbox 環境

Sandbox とは「砂場を意味し、アプリケーションを保護された領域で動作させることによって、システムが不正に操作されるのを防ぐセキュリティモデルのこと」[5] であり、外部へのアクセスが厳しく制限された領域のことを指す。使用用途としてはマルウェア対策やソフトウェアの動作検証などが挙げられる。

この Sandbox を提供するものが「Sandbox 環境」であり、一種の仮想環境と考えて良い。ただし、Virtualbox などで作成される「仮想マシン」と異なり、ホスト OS のカーネルを利用する場合が多く、OS を丸ごとエミュレートするわけではないので高速であるという特徴がある。Sandbox 環境の例として、Web アプリケーション開発環境として利用されている Docker やソフトウェアテスト環境の Windows Sandbox などある。

*1 〒 811-4192 宗像市赤間文教町 1-1　E-mail: fujimoto@fukuoka-edu.ac.jp

モバイルデバイスの OS である Android と iOS は、カーネルとしてそれぞれ Linux と Darwin という UNIX 系カーネルを利用している。そのため、これらのモバイルデバイス上で UNIX 系のコマンドラインアプリを利用したい、Web アプリケーションの開発環境を構築したい、ssh によるサーバー管理を行いたい、というニーズが少なからずある。しかし、どちらの OS も root 権限はユーザーに開放されておらず、通常の Linux マシンのように利用することは想定されていない。そこで、このようなニーズに応えるために Sandbox 環境を利用したターミナルエミュレータアプリが開発された。以下の節で Android と iOS のそれぞれについて、Sandbox タイプのターミナルアプリの詳細とその上で Risa/Asir を動作させるための手法について解説する。

3 Android の場合

3.1 Android の Sandbox アプリ

Android には、Termux [6] というターミナルアプリが Google の公式アプリストアの GooglePlay で配信されている。Android の Linux カーネルを利用した Sandbox ターミナルエミュレータであり、root 化と呼ばれる機能制限の解除をすることなくコマンドライン版の Linux 環境を提供するものである。`/etc` や `/var` などへのアクセスは不可であったり、`/data/data/com.termux/files/` をルートディレクトリとするという制限はあるもののメジャーな Linux ディストリビューションの Debian と同じ apt によるパッケージ管理機能を備え、通常の Linux マシンと同等に扱うことが可能である。以下ではこの Termux を用いることとする。

3.2 Termux のインストール

Android デバイスに Termux をインストールする手順は以下の通りである。

1. GooglePlay から Termux をインストールする。
2. Termux を開く。
3. `termux-setup-storage` を実行する[1)]。
4. `pkg update` を実行する。
5. `pkg upgrade` を実行する。

3.3 Termux 用ビルド環境の構築

Termux で動作するコマンドラインアプリをビルドする方法として次の 2 つがある。

方法 (a) 公式 Docker イメージを用いてクロスビルド

方法 (b) Android デバイス上でセルフビルド

方法 (a) の方が環境構築は簡単であるが、方法 (b) は実行環境と同じ環境であるためビルド時のエラー原因の究明が簡単という利点がある。よって、ここでは方法 (b) を採用する。

Termux を開き、以下のように pkg コマンドでビルドに必要なパッケージをインストールする。

[1)] これによって `home` ディレクトリに内部ストレージへのシンボリックリンクを集めた `storage` ディレクトリが作成される。

```
$ pkg install autoconf automake bison bzip2 clang cmake \
> coreutils diffutils flex gawk grep gzip libtool make \
> patch perl sed tar termux-exec wget file
$ cd ../usr/bin
$ ln -s busybox uuencode
$ ln -s busybox uudecode
```

3.4 Risa/Asir のビルド

Risa/Asir の最新のソースコードをダウンロードして解凍する。

```
$ wget http://air.s.kanazawa-u.ac.jp/ohara/openxm/openxm-head.tar.
  gz
$ tar xf openxm-head.tar.gz
```

ビルドの前に次のように環境変数を設定する。

```
$ export PREFIX=/data/data/com.termux/files/usr
$ export LD_PRELOAD=${PREFIX}/lib/libtermux-exec.so
$ export CC=clang
$ export CXX=clang++
$ export CFLAGS="-O2 -std=gnu89 -fsigned-char -D ANDROID
-DNO_GETCONTEXT -Wno-macro-redefined -Wno-builtin-macro-redefined"
```

以上の準備の後、configure を実行して gc をビルドする。

```
$ cd OpenXM/src
$ make configure-asir
$ cd gc
$ make all
$ make install
$ cd ..
```

これで OpenXM/lib に libpari.a, libasir-gc.so などが生成される。そして、次のように gmp をビルドする。

```
$ make configure-gmp
$ cd gmp
$ make all
$ make install
$ cd ..
```

これと同様に、mpfr/mpc/mpfi/ox_toolkit/ox_pari/ox_cpp をビルドして、最後に Risa/Asir を以下のようにビルドしてインストールする。

```
$ cd ../../OpenXM_contrib2/asir2018
$ ./configure --prefix=/data/data/com.termux/files/home/OpenXM
$ make
$ make install
$ make install-openxm
```

3.5 Risa/Asir の起動

以下の設定後、Risa/Asir を Termux 内で起動できる。

```
$ export OpenXM_HOME=/data/data/com.termux/files/home/OpenXM
$ export PATH="${OpenXM_HOME}"/bin:$PATH
$ export LD_LIBRARY_PATH="${OpenXM_HOME}"/lib:"$LD_LIBRARY_PATH"
$ asir
This is Risa/Asir, full GMP Version 20200210 (Kobe Distribution).
Copyright (C) 1994-2000, all rights reserved, FUJITSU LABORATORIES
   LIMITED.
Copyright 2000-2019, Risa/Asir committers, http://www.openxm.org/.
GC 7.4.2 copyright 1988-2015, H-J. Boehm, A. J. Demers, Xerox, SGI
  , HP.
[0]
```

3.6 Android 版 Risa/Asir の注意事項

Android で使用されている C ライブラリは一般的な glibc と異なり、Bionic libc [7] というものである。このため、エラーなくビルドするためにはソースコードの修正が必要であった。以下に主な問題とその対策例を挙げる。

- FILE 構造体のメンバーが普通の Linux と異なる。
  → ヘッダーファイルを辿って調査して定義を追加。
- `strings.h` に index 関数や rindex 関数がない。
  → strchr 関数や strrchr 関数を利用するように変更。
- `stdio.h` に getw 関数がない。
  → fread 関数を利用するように変更。
- `/tmp` ディレクトリがない。
  → `/data/data/com.termux/files/usr/tmp` を指定。

また、Android はカーネルが Linux であるため、ソースコード中のマクロ定義の `#if defined(linux)` の部分が読み込まれてしまう。そこで、Android 独自の対策が必要な箇所には、

```
#if defined(ANDROID)
#define FP_DATA_IS_AVAILABLE(fp) ((fp)->_r)
#elif defined(linux)
...
```

のように `#if defined(linux)` の前に `#if defined(ANDROID)` を挿入して記述する必要があった。

3.7 Asir on Android の配布

上で述べたビルド方法で作成したバイナリは次の URL で配布している。

`http://ww1.fukuoka-edu.ac.jp/~fujimoto/asiroid`

この配布パッケージには、asir のフロントエンドとして利用できる fep も同梱している。

Android デバイスに Termux をインストールした後、以下のように asir をインストールする。

```
$ pkg install wget
$ wget http://ww1.fukuoka-edu.ac.jp/~fujimoto/android/asir_aarch64
  .tar.gz
$ tar xf asir_aarch64.tar.gz
$ cp OpenXM/rc/feprc .feprc
$ cp ../usr/etc/profile .profile
$ echo "source ./OpenXM/rc/bashrc" >> .profile
$ exit
```

Termux を再起動後、以下の操作で Risa/Asir を起動できる。

```
$ fep asir
```

4 iOS の場合

4.1 iOS の Sandbox アプリ

iOS には、LibTerm [8] と a-Shell [9] という Sandbox タイプのターミナルアプリがアップルの公式アプリストアの AppStore で配信されている。どちらも iOS の Darwin カーネルを利用したものであり、UNIX コマンドの利用だけでなくスクリプト言語の Python や Lua によるプログラミングも可能である。さらに、clang コマンドによって C 言語のソースコードもコンパイルできる。しかし、このコンパイルで生成されるのはネイティブコードではなく中間コードの LLVM IR であり、実行はインタプリタの lli コマンドで行わなければならない。このため C 言語で書かれたプログラムの実行速度は遅く、一般的なソースコードからの configure と make によるビルドもサポートされていないことから、通常の UNIX 系 OS として利用するのは難しい[2)]。

一方、iSH Shell [10] はベータ版アプリ配信サービスである TestFlight を通して配信されているターミナルアプリである。LibTerm や a-Shell と異なり、ユーザーモードの x86 エミュレーションとシステムコー

[2)] iOS のためのコマンドラインツールについては以下のサイトが詳しい：
`http://maverick.inria.fr/~Nicolas.Holzschuch/ios_shell.html`

ル変換の実装により 32bit の Linux Sandbox を実現している。使用している Linux ディストリビューションは Docker でも利用されている軽量な Alpine Linux であり、パッケージ管理コマンドの apk を用いて多くのコマンドラインアプリケーションが利用可能である。以下では iSH Shell を用いることとする。

4.2 iSH Shell のインストール

iOS デバイスに iSH Shell をインストールする手順は以下の通りである。

1. AppStore から TestFlight をインストールする。
2. iOS デバイスの Web ブラウザ Safari から以下のサイトにアクセスする。
   `https://testflight.apple.com/join/97i7KM8O`
3. 「同意する」をクリックする。
4. iSH Shell を開く。
5. `apk update` を実行する。
6. `apk upgrade` を実行する。

4.3 iSH Shell 用ビルド環境の構築

Android の場合でも述べたように、ビルド時のエラー原因の究明が簡単なのはセルフビルドである。iSH Shell にはパッケージ管理コマンド apk があるので、ビルド環境の構築は比較的簡単である。しかし、iSH Shell は Termux と異なりエミュレータのため動作速度が遅い。また、メモリ管理機構が不完全であるため gmp などの大きなライブラリをビルドしようとするとメモリリークが発生して iSH Shell がクラッシュしてしまうことがある。

そこで、iSH Shell の環境に近い Alpine Linux x86 VIRTUAL 版 (3.11) [11] の仮想マシンを Virtualbox で作成してビルド環境として用いる。そして、仮想マシン内で以下のように apk コマンドでビルドに必要なパッケージをインストールする。

```
# apk add gcc g++ build-base autoconf automake bison make cmake \
> coreutils diffutils findutils sharutils \
> file flex gawk git grep libtool patch perl sed \
> bzip2 gzip tar curl wget ncurses-dev
```

4.4 Risa/Asir のビルド

Risa/Asir の最新のソースコードをダウンロードして解凍する。

```
# mkdir local
# cd local
# wget http://air.s.kanazawa-u.ac.jp/ohara/openxm/openxm-head.tar.
  gz
# tar xf openxm-head.tar.gz
```

ビルドの前に次のように環境変数を設定する。

```
# export ABI=32
# export CFLAGS='-O2 -D_GNU_SOURCE -DNO_GETCONTEXT -DUSE_MMAP
-mtune=pentiumpro -march=pentiumpro'
```

ABI の設定は gmp の configure を通すために必要である。CFLAGS の -D_GNU_SOURCE -DNO_GETCONTEXT -DUSE_MMAP は Alpine Linux で動作するガベージコレクタ gc のビルドに必要なものである。以上の準備の後、configure を実行して gc をビルドする。

```
# cd OpenXM/src
# make configure-asir
# cd gc
# make all
# make install
# cd ..
```

これで OpenXM/lib に libpari.a, libasir-gc.so などが生成される。

次に gmp/Makefile をエディタで開き、configure-others の所のオプションを以下に修正する。このオプションを指定しないと、iSH Shell での実行時に illegal instruction (ILL) が発生してしまう。

```
--enable-cxx --host=i586-alpine-linux-musl
```

そして、次のように gmp をビルドする。

```
# make configure-gmp
# cd gmp
# make all
# make install
# cd ..
```

これと同様に、mpfr/mpc/mpfi/ox_toolkit/ox_pari/ox_cpp をビルドして、最後に Risa/Asir を以下のようにビルドしてインストールする。

```
# cd ../../OpenXM_contrib2/asir2018
# ./configure --prefix=/root/local/OpenXM
# make
# make install
# make install-openxm
```

4.5 Risa/Asir の起動

以下の設定後、Risa/Asir を仮想マシン内で起動できる。

```
# export OpenXM_HOME=/root/local/OpenXM
# export PATH="${OpenXM_HOME}"/bin:$PATH
# export LD_LIBRARY_PATH="${OpenXM_HOME}"/lib:"$LD_LIBRARY_PATH"
```

```
# asir
This is Risa/Asir, full GMP Version 20200210 (Kobe Distribution).
Copyright (C) 1994-2000, all rights reserved, FUJITSU LABORATORIES
   LIMITED.
Copyright 2000-2019, Risa/Asir committers, http://www.openxm.org/.
GC 7.4.2 copyright 1988-2015, H-J. Boehm, A. J. Demers, Xerox, SGI
  , HP.
[0]
```

以上の操作によって作成したバイナリは iOS デバイス上の iSH Shell で動作する。

4.6 iSH Shell の注意事項

iSH Shell の Alpine Linux で使用されている C ライブラリは一般的な glibc と異なり、musl libc [12] である。このため、エラーなくビルドするためにはソースコードの修正が必要であった。具体的には、FILE 構造体へのインターフェースとして、`stdio_ext.h` の `__freadable()` と `__fpurge()` を利用するようにした。

また、現在の iSH Shell は整数演算や浮動小数演算の CPU 命令セットの MMX や SSE に未対応なので、エミュレートする CPU は Pentium Pro 相当である。そのため、CPU 命令セットを用いる gmp の configure には `--host=i586-alpine-linux-musl` のオプション指定が必要であった。

4.7 Asir on iOS の配布

上で述べたビルド方法で作成したバイナリは次の URL で配布している。

`http://ww1.fukuoka-edu.ac.jp/~fujimoto/iOS`

この配布パッケージには、asir のフロントエンドとして利用できる fep も同梱している。

iOS デバイスに iSH Shell をインストールした後、以下のように asir をインストールする。

```
# wget http://ww1.fukuoka-edu.ac.jp/~fujimoto/ios/asir_ios2.tar.gz
# tar xf asir_ios2.tar.gz
# cp OpenXM/rc/feprc .feprc
# cp /etc/profile .profile
# echo "source ./OpenXM/rc/bashrc" >> .profile
# apk add ncurses-dev
# exit
```

iSH Shell を再起動後、以下の操作で Risa/Asir を起動できる。

```
# fep asir
```

5 スクリーンショット

Android と iOS で Risa/Asir が動作している画面のスクリーンショットを以下に示す。

図 1: Termux で動作する Risa/Asir

図 2: iSH Shell で動作する Risa/Asir

6 ベンチマーク

本稿で解説した手法によって作成した Risa/Asir バイナリの計算速度について述べる。以下の表は、Risa/Asir の `nd_gr(C, vars(C), 31991,0)$` コマンドによるグレブナー基底計算 (cyclic-7, cyclic-8) のベンチマーク結果（単位は秒）である。参考として、Windows マシンでの計測結果も掲載しておく。

	Nexus 7(2013) Android 6.0.1 Snapdragon S4 1.5GHz 2GB memory	Phab 2 Pro Android 6.0.1 Snapdragon 652 1.8GHz 4GB memory	Pixel 3a XL Android 10 Snapdragon 670 2.0GHz 4GB memory	iPad mini5 iOS 13.3.1 Apple A12 2.5GHz 3GB memory	ThinkPad T470s Windows 10 Core i7-7600U 3.9GHz 40GB memory
Cyclic 7	16.52	3.17	2.39	28.94	1.28
Cyclic 8	467.30	107.50	58.99	670.5	34.30

7 今後の課題

本稿で述べた手法により、モバイルデバイス用の数式処理システムとしてフルスペックの安定したバージョンを作成できるようになった。今後に取り組むべき課題としては以下のものが考えられる。

- Termux 用 deb パッケージの作成
- iSH Shell 用 apk パッケージの作成
- Qt Quick または Kivy などのマルチプラットフォームの GUI ツールキットを使ったモバイルデバイス用 GUI の作成

謝　辞

本研究は JSPS 科研費 JP17K01133 の助成を受けたものである。

参 考 文 献

[1] Mitsushi Fujimoto, The current state of computer algebra system on tablet devices, Developments in Computer Algebra Research, RIMS Kokyuroku vol.1930 (2015) pp.93–100.

[2] Scratchbox, http://www.scratchbox.org/

[3] 藤本光史, タブレット端末への数式処理システムの実装手法, 京都大学数理解析研究所講究録 1927, 「数式処理研究の新たな発展」(2014) pp.89–102.

[4] 藤本光史, 数式処理システムの arm64 対応について, 数式処理 Vol.23, No.2 (2017) pp.128–131.

[5] 総務省 国民のための情報セキュリティサイト, https://www.soumu.go.jp/main_sosiki/joho_tsusin/security/

[6] Termux, https://termux.com/

[7] Bionic libc, https://android.googlesource.com/platform/bionic/

[8] LibTerm, https://libterm.app/

[9] a-Shell, https://holzschu.github.io/a-Shell_iOS/

[10] iSH Shell, https://github.com/tbodt/ish

[11] Alpine Linux ISO images, https://www.alpinelinux.org/downloads/

[12] musl libc, https://musl.libc.org/