

# Linux ABIの仕組み

Linuxバイナリを自作OS上でそのまま動かす

nuta@seiya.me

# 内容

- Linuxバイナリをそのまま動かす「ABIエミュレーション」の解説
- 先行事例
- Linuxプログラムの初期化处理
- Linuxシステムコール処理
- Busyboxのシェルを動かすのに必要なもの

デモ

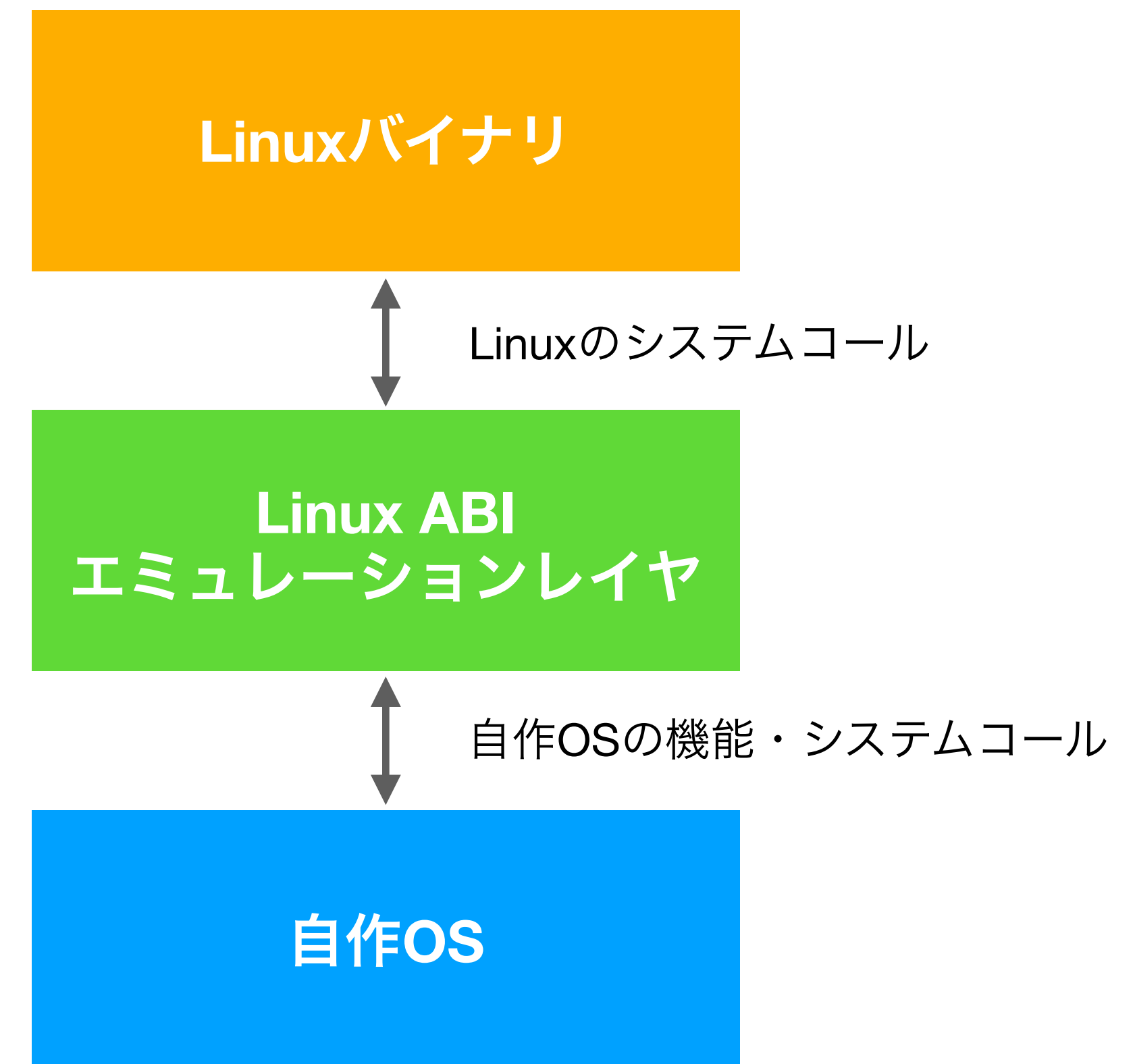
# 先行事例

- **Windows Subsystem for Linux (WSL)**
  - WSL1ではバイナリエミュレーションしていたらしい
- **FreeBSD / NetBSD**
  - NetBSDはLibre OfficeのLinuxバイナリが動くレベルらしい
- **OSv**
  - Linux ABIを実装したユニカーネル

Linuxバイナリ対応に  
(最低限) 必要なもの

# ABIエミュレーションの概要

- Linuxバイナリが使うもの（システムコール等）を提供するのが目的
- メインはLinuxシステムコールを自作OSの機能を使って実装すること
- 自作OS側でシステムコールへのフック機構が必要



# Linuxプログラムの初期化処理

初期化は特に大変ではない:

## 1.Linuxバイナリ（ELFファイル）のロード

- .bssセクションのゼロクリアし忘れに気をつけよう

## 2.コマンドライン引数, 環境変数, 補助ベクタのコピー

- 補助ベクタはlibcが内部で使う。AT\_NULLだけおけば（とりあえず）動く。

## 3.ユーザプロセスの開始

# Linuxシステムコールの処理 (x86\_64)

- System V ABIのドキュメントの付録に書いてある
  - SYSCALL命令で呼ばれる。引数は全てレジスタ経由。
  - **RCX/R11以外の汎用レジスタはシステムコールハンドラが上書きしてはだめ（復元してあげる必要がある）**
  - [https://www.uclibc.org/docs/psABI-x86\\_64.pdf](https://www.uclibc.org/docs/psABI-x86_64.pdf)
- 自作OSが独自のシステムコール体系を持っている場合は、何らかのフック機構が必要
- ここができればシステムコールをひたすら実装していくだけ。



# 必要なシステムコール（アセンブリのHello World）

- write
  - 画面出力をする
- exit
  - プロセスの終了

```
.code64
.intel_syntax noprefix
.global _start
_start:
    ; write(stdout, "Hello, World from Linux ABI!\n", 29)
    mov rax, 1
    mov rdi, 1
    lea rsi, [rip + msg]
    mov rdx, 29
    syscall

    ; exit(0)
    mov rax, 60
    mov rdi, 0
    syscall

    ud2

.section .rodata
msg:
    .ascii "Hello World from Linux ABI!\n"
```

# 必要なシステムコール (CのHello World)

- arch\_prctl
  - FSレジスタのベースアドレスの設定
- set\_tid\_address
  - マルチスレッディング関連  
(とりあえず0を返せばOK)
- ioctl
  - 確か標準出力がttyがチェック  
(とりあえず0を返せばOK)
- writev
  - putsが使う

```
// musl-gcc -static -o hello hello.c
int main(void) {
    puts("Hello World from Linux ABI!");
}
```

# 必要なシステムコール (Busyboxのcatコマンド)

- open
    - ファイルを開く
  - read
    - ファイルの読み込み
  - close
    - ファイルを閉じる
- (とりあえず0を返せばOK)

[illegible]

# 必要なシステムコール (Busyboxのechoコマンド)


- brk
  - ヒープ領域の拡張

A screenshot of a QEMU terminal window. The window title bar shows 'QEMU' and standard macOS window controls (red, yellow, grey buttons). The terminal output shows the BusyBox version and build information, followed by the execution of the 'echo' command which prints 'hello!'.

```
QEMU
BusyBox v1.31.1 (2020-05-10 01:48:42 UTC) built-in shell (a
$ echo hello!
hello!
$ _
```

# 必要なシステムコール (Busyboxのシェル)

- getpid
  - シェルのPIDを返す
- read (キーボード入力)
- stat
  - \$PATH内の探索に使う  
(加えてst\_modeで実行可能か見ている)
- fork / exec / wait4
  - プロセスの作成と待機
- rt\_sigaction / rtsigprocmask
  - シグナル関連  
(とりあえず0を返せばOK)



```
QEMU
BusyBox v1.31.1 (2020-05-10 01:48:42 UTC) built-in shell (ash)
$ echo hello!
hello!
$ _
```

# まとめ

- Linuxバイナリ互換に必要なものをざっくり説明した
- `uname(1)` が動くと感動する
- 簡単なプログラムを動かすだけなら、それ程大変ではないので皆さんもやりましょう
- ここでもう少し詳しく説明しています
  - <https://seiya.me/implementing-linux-abi>