自作OS用デバッガの開発

学習駆動コース　坂井ゼミ　村井公哉

1. 作品の概要

・対象者

趣味でOS自作をしている人やOS自作に挑戦したい人.

・目的

自作OSのデバッグを簡単化したい("OS自作について"を参照)

・自作OS用デバッガとは

CPUをエミュレートし,その上でOSを動かしてデバッグするツールのこと. すべてPythonを用いてコーディングされており, 環境構築も簡単に行える.

作品を作ろうと思ったきっかけ

中学生の頃の私

「あれー, ブートローダがうまく動作しない！自分が書いたコード見ても何が原因かさっぱりわからない！」

SecHack365
参加前の私

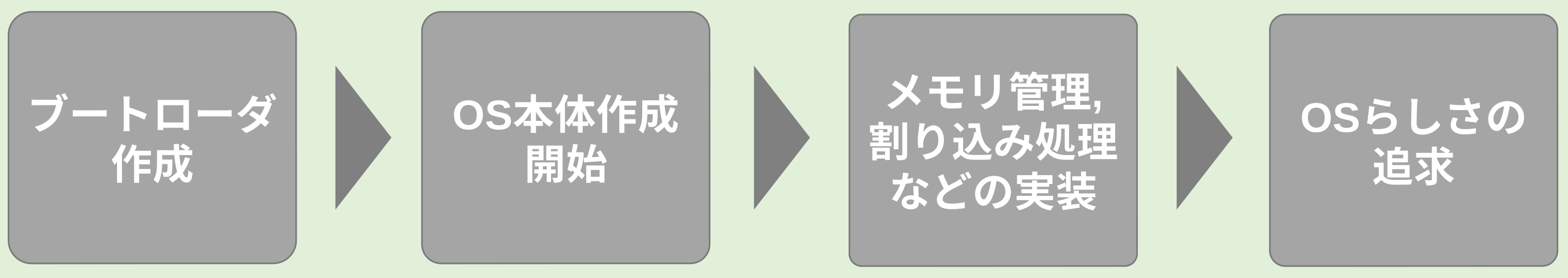
「あーそういえば中学生の頃に自作OSのデバッグで悩んでたなあ. よし, OS自作をサポートする作品を作ろう！」

自作OS用デバッガの作成へ

2. OS自作について

・OS自作のおおまかな流れ

OS自作は主にブートローダの作成, メモリ管理や割り込み処理の実装を行う.


ブートローダ作成 ▶ OS本体作成開始 ▶ メモリ管理, 割り込み処理などの実装 ▶ OSらしさの追求

使用する言語...アセンブリ言語やC言語など

ブートローダー...OSをロードするためのプログラム. PC起動後に処理される.

割り込み...マウスのクリック, ネットワークパケット到着などの不定期的に発生するイベント

・OS自作のメリット

1からOSを制作することにより, 割り込み処理やメモリ管理, ページングやスケジューリングなどの重要な知識が得られるので, 低レイヤーに少し強くなれる. また, 楽しいので趣味になる.

・OS自作のデメリット

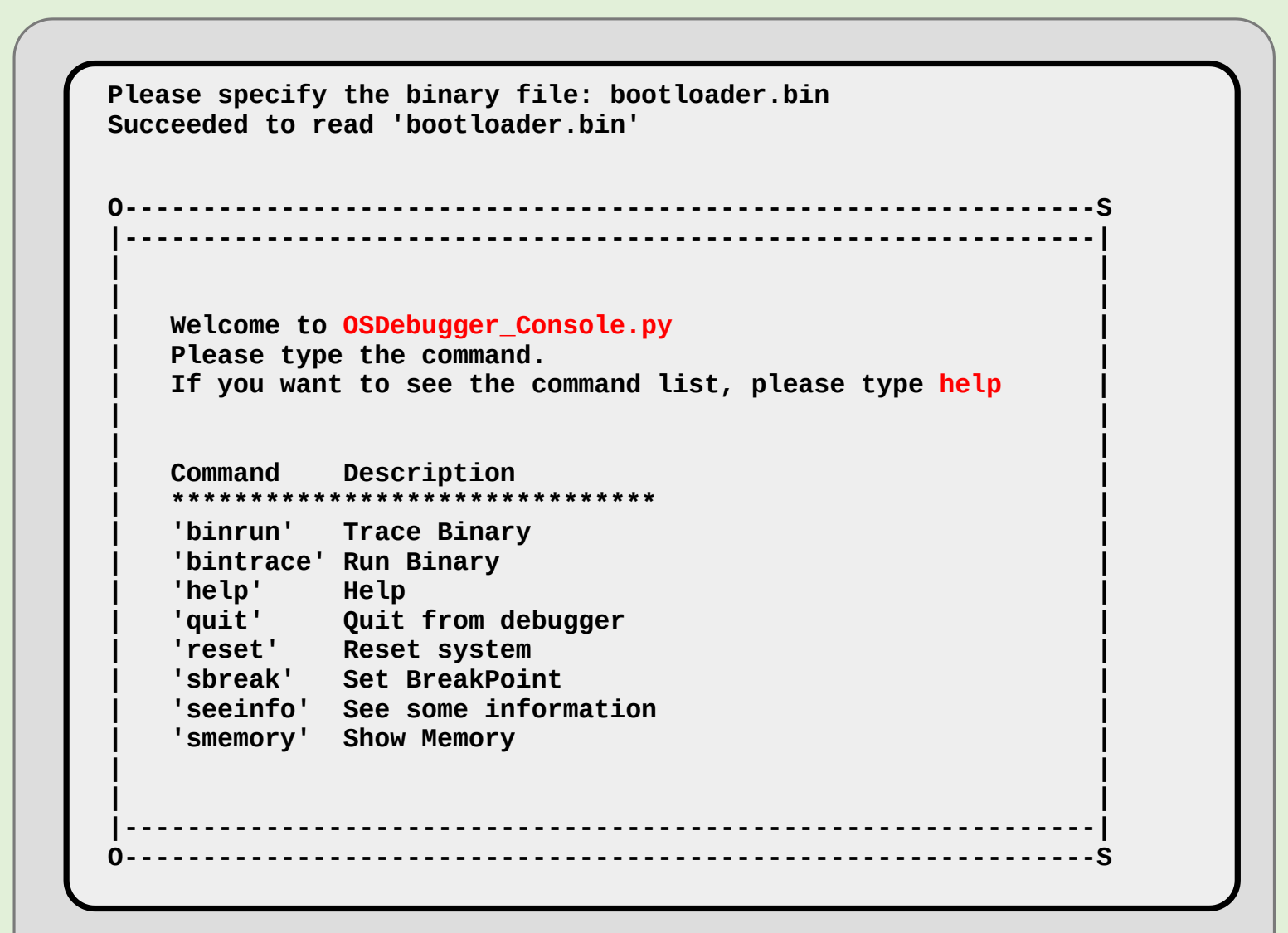
自作したOSを実際に動かし, 不具合が生じてもエラーメッセージが表示されないことが多い. これではデバッグが非常に困難となる. デバッグが困難であれば, 単純なプログラムミスによる不具合であったとしても, 原因の特定や解決に多くの時間を割くことになる. このように, OS自作においてデバッグが一番のボトルネックとなるため, **デバッグの簡単化をすることが必要であると考えられる.**

3. デバッガの詳細

・デバッガの特徴

自作OSのデバッグの簡単化を目的として, 様々な機能を持ったデバッガを作成した. 特徴は以下の通り.

- x86に対応
- GDTやIDT, PIC, ビデオの状態を簡単に確認可能
- ディスプレイをサポート(Pygletを使用)
- コンソール版とブラウザ版がある
- よく使われていそうな命令のみをエミュレート
- OSのイメージファイルを読み込ませるだけで実行可能
  つまり, エミュレータを準備する必要はない



```
Please specify the binary file: bootloader.bin
Succeeded to read 'bootloader.bin'


O---------------------------------------------------------------S
|---------------------------------------------------------------|
|                                                               |
|                                                               |
|   Welcome to OSDebugger_Console.py                            |
|   Please type the command.                                    |
|   If you want to see the command list, please type help       |
|                                                               |
|                                                               |
|   Command    Description                                      |
|   ********************************                            |
|   'binrun'   Trace Binary                                     |
|   'bintrace' Run Binary                                       |
|   'help'     Help                                             |
|   'quit'     Quit from debugger                               |
|   'reset'    Reset system                                     |
|   'sbreak'   Set BreakPoint                                   |
|   'seeinfo'  See some information                             |
|   'smemory'  Show Memory                                      |
|                                                               |
|                                                               |
|---------------------------------------------------------------|
O---------------------------------------------------------------S
```

起動した時の画面. 最初にコマンド一覧が表示されるため, はじめて触る人にも優しいツールとなっている

よくある例		この機能で解決！
突然画面が真っ黒になった！！	▶	トレース機能, ビデオ確認
割り込みが正常か確認したい！	▶	IDT, PICの確認
保護モードに移行できない！	▶	GDTの確認
OS本体をうまく実行できない！	▶	トレース機能

3. デバッガの詳細(続き)

・デバッガの仕組み

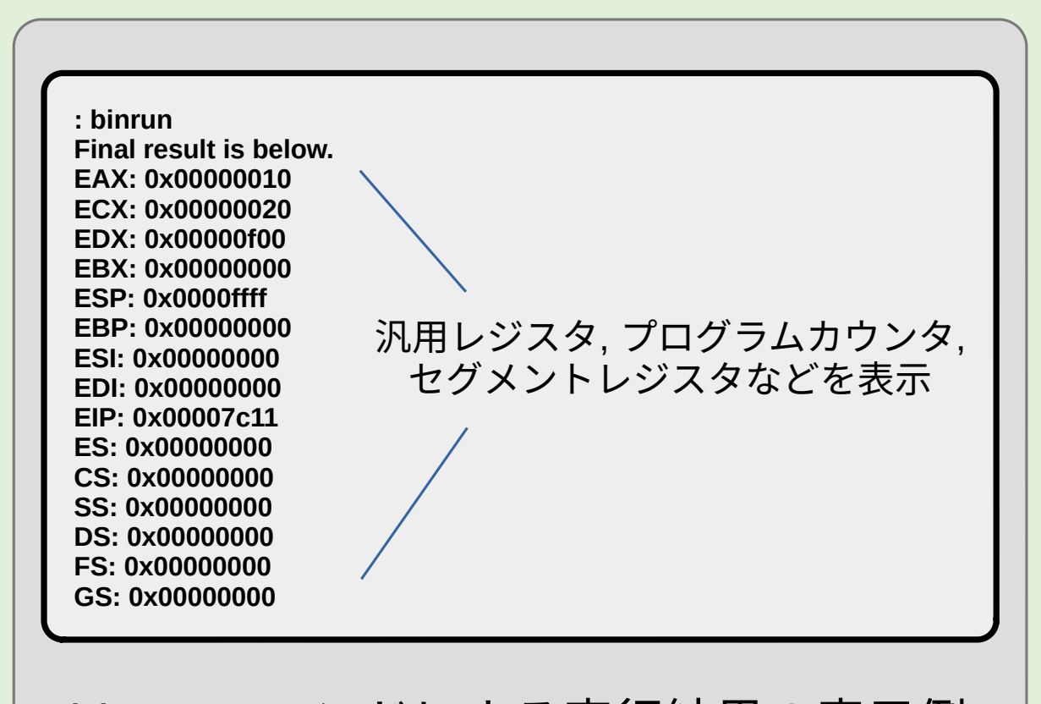
今回作成したデバッガは, 大まかに3つのプログラムに分けられる. 主にトレースやBreakPointの制御を行うプログラム, 命令フェッチと実行を行うプログラム, 操作などのユーザーインターフェース関連の制御を行うプログラムである. ディスプレイ機能については別のスレッドで実行するようにしている. ブラウザ版では, トレースやBreakPointの制御を行うプログラムも別のスレッドで実行し, ブラウザに結果を返すようにしている.

・実行例

binrunコマンド

デバッガに読み込ませたOSを, **"binrun"**というコマンドによってデバッガ上で動かすことができる. hlt命令の実行によって処理の 完了とみなし, その時のレジスタの値を表示する. 命令フェッチ, 実行などはすべてデバッガ側で行われている.

右の図は, 実際にbinrunコマンドを実行し, 結果を表示している(ここでは結果の途中部分までを載せている)



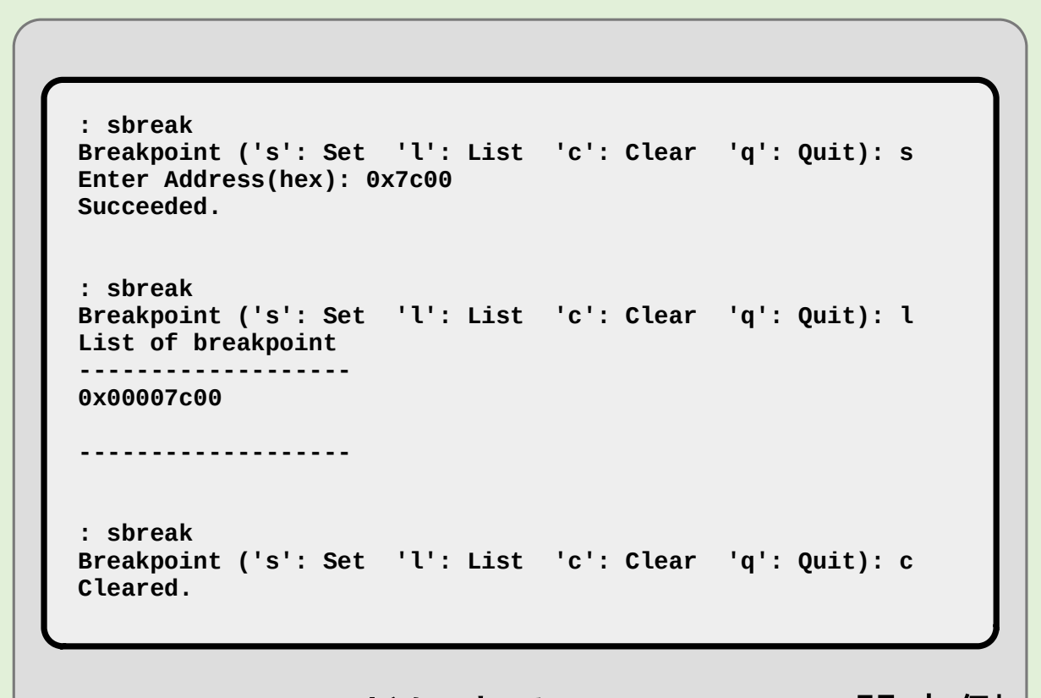
```
: binrun
Final result is below.
EAX: 0x00000010
ECX: 0x00000020
EDX: 0x00000f00
EBX: 0x00000000
ESP: 0x0000ffff
EBP: 0x00000000
ESI: 0x00000000
EDI: 0x00000000
EIP: 0x00007c11
ES: 0x00000000
CS: 0x00000000
SS: 0x00000000
DS: 0x00000000
FS: 0x00000000
GS: 0x00000000
```

binrunコマンドによる実行結果の表示例

BreakPointの設定

デバッガにはBreakPointの設定を行う**sbreak**コマンドがある. BreakPointとは, 実行の一時停止をする場所のことであり, コマンドを実行すると, BreakPointの設置やクリア, 設置したアドレスのリスト表示を行うことができる. リスト表示により, どこにBreakPointを設置したかの一覧を一目で確認することが可能となる. binrunコマンドを組み合わせることで, OSのデバッグを効率的に行うことができる.

右の図は, sbreakコマンドを実行したときのものである.



```
: sbreak
Breakpoint ('s': Set  'l': List  'c': Clear  'q': Quit): s
Enter Address(hex): 0x7c00
Succeeded.


: sbreak
Breakpoint ('s': Set  'l': List  'c': Clear  'q': Quit): l
List of breakpoint
-------------------
0x00007c00

-------------------


: sbreak
Breakpoint ('s': Set  'l': List  'c': Clear  'q': Quit): c
Cleared.
```

sbreakコマンドによるBreakPointの設定例

3.でご紹介したすべての作品の画面は, 実際のキャプチャ画面ではありません.

4. セキュリティ面

Buffer Overflow AttackやFormat String Attackにより, スタック上にある戻りアドレスが書き換えられてしまうと, 任意コード実行によって権限が奪取されることがある. 事前に防止策を施すことは重要である.

今回は, 戻りアドレス書き換えによる任意コード実行を防止する機能をデバッガに実装した(下図)

説明

1. call命令実行時に, "戻りアドレス"のアドレスと, 戻りアドレスのペアをデバッガ側で記憶
2. ret命令実行時に, 先ほど記憶した情報をプログラムカウンタとスタックにセットする

call命令とは関数を呼び出すための命令, ret命令とは関数から呼び出し元へ戻る際に実行する命令のことである. こうすることによって, たとえ戻りアドレスがBuffer Overflow AttackまたはFormat String Attackによって書き換えられたとしても, 正しい呼び出し先に戻ることができる.

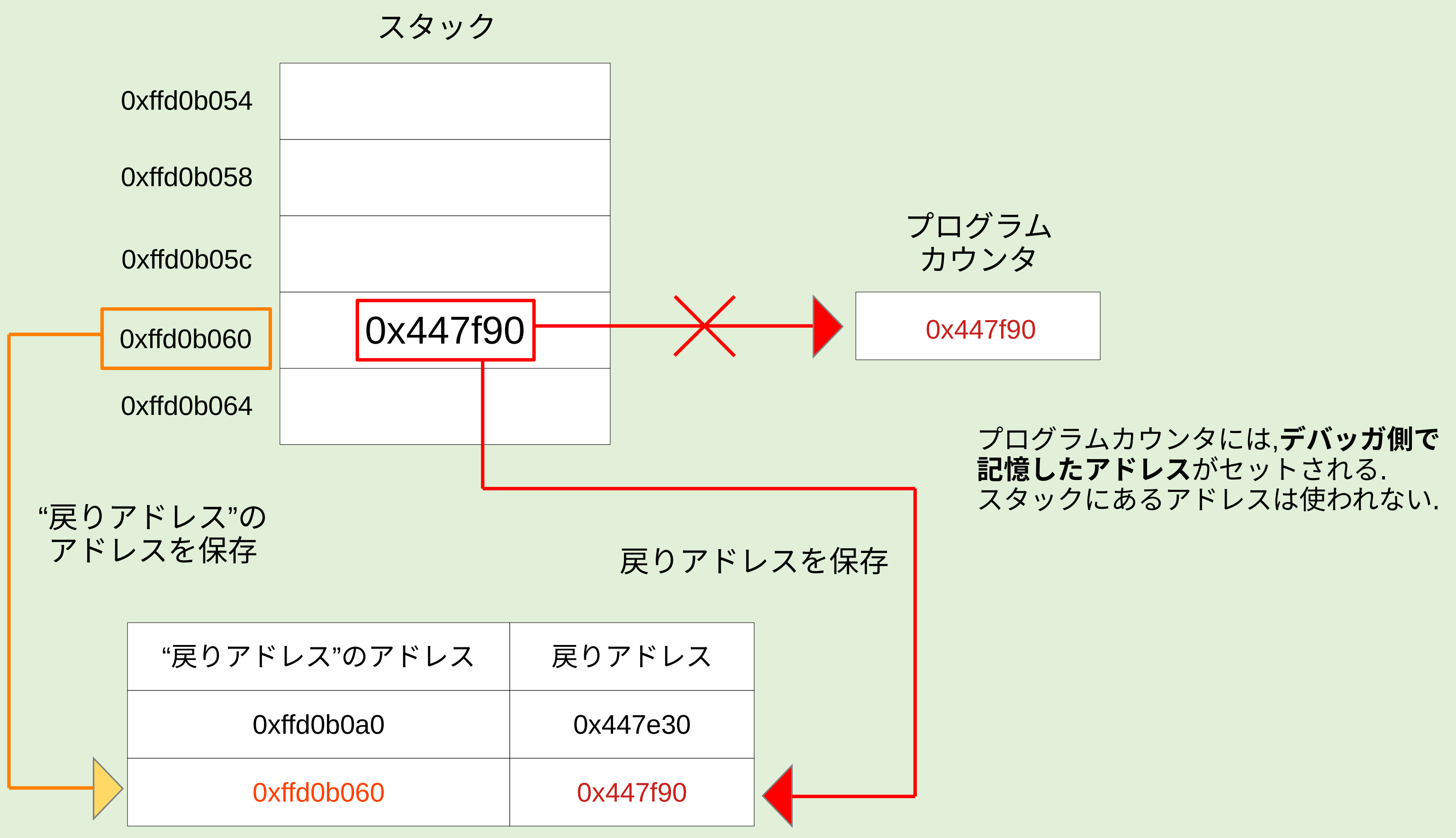


図: セキュリティ機能について

5. SecHack365の活動と今後について

SecHack365での活動は非常に充実したものとなっている. 開発着手時は完成させることができるのだろうかという不安があったが, 形になっていくにつれ, 開発を楽しむようになった. また, 多くの方々と交流して刺激を受けたことがとても良い経験となった.

今後, 自作OS用デバッガの規模はまだまだ大きくすることができると考えられるため、逆アセンブラ機能の実装, さらに多くの割り込み処理に対応させること, ユーザーインターフェースの改善などを行っていきたい.