MPUを活用したセキュアな組み込みOSの開発

学習駆動コース　坂井ゼミ　黒須紀行

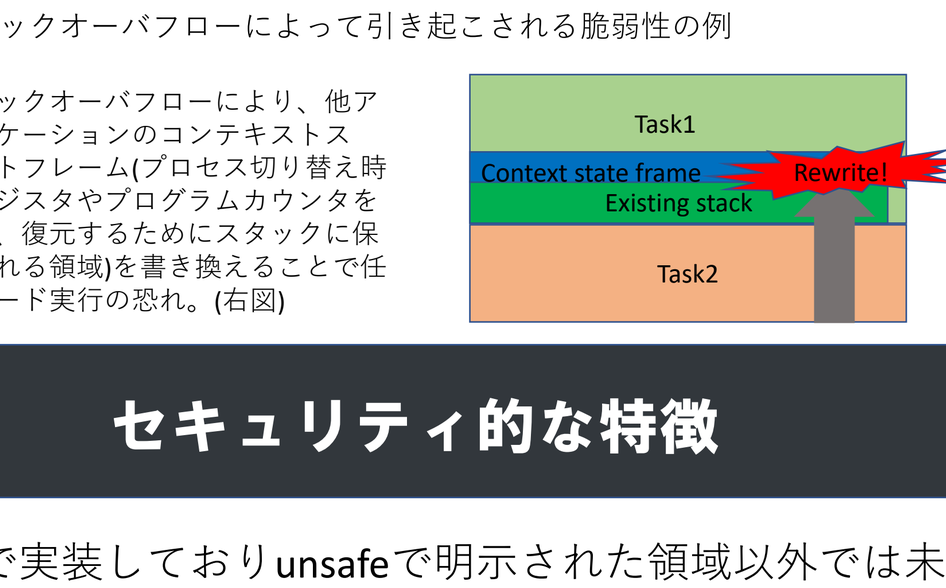
開発動機

- １２ステップで作る組込みOS自作入門を見て組み込みOSを書いていた。
- 最近Rustが組み込み分野でも徐々に使われるようになった。

Rustと組み込みソフトウェア両方興味がでてきた！
Rustで組み込みOS書きたい！

実装機能

- **コンテキストスイッチ**
  - 複数のプロセスが１つのCPUを共有できるようにレジスタを退避、復帰する仕組み。
- **タスクスケジューラ**
  - ラウンドロビン式と固定優先度式の２種類のスケジューラを実装した。
  - ラウンドロビン式：１タスクがCPUを使用する時間を定め、順々に使用権を与えていくスケジュール方法。
  - 固定優先度式：プロセスに優先度をつけることで、優先度順に使用権を与えていくスケジュール方法。
- **MPU(メモリ保護機構)を用いたシステムコール**
  - 開発で使用しているマイコン(STM32F401)ドキュメント読んだ時、メモリのアクセスを管理できるユニット、MPUを見つけて、活用できないか考えた。
  - 実験的実装として、アプリケーション実行時(非特権状態時)は、ペリフェラル領域にはシステムコール経由でないとアクセスできないようにした。

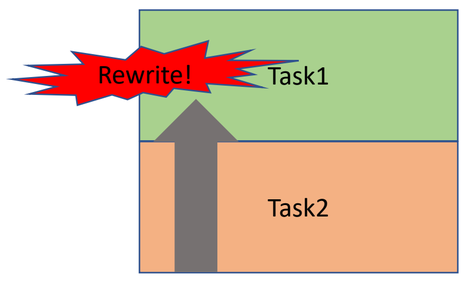


- **MPUを用いたスタックオーバフロー検知機構**
  - スタックオーバフローとは？

    事前に割り当てられたスタックサイズを超えたデータプッシュにより文字通りスタックを超過してしまうこと。

    例）タスク２実行時に再帰関数呼び出し過ぎによりスタックオーバフローが発生。タスク１のスタック領域を書き換えてしまい、未定義動作を引き起こしたりクラッシュする。(右図)

スタック領域のメモリマップ



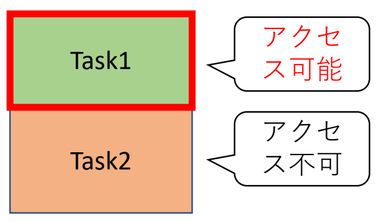
  - スタックオーバフローによって引き起こされる脆弱性の例

    スタックオーバフローにより、他アプリケーションのコンテキストステートフレーム(プロセス切り替え時にレジスタやプログラムカウンタを退避、復元するためにスタックに保存される領域)を書き換えることで任意コード実行の恐れ。(右図)

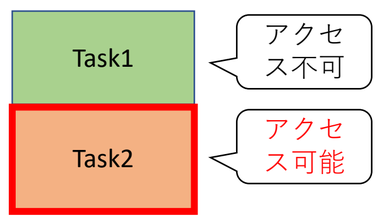
- OSでMPUを操作しスタックオーバフローを検知する

  プロセス切り替え時にMPUを操作し、アプリケーションレイヤ(非特権状態)では、次に実行するタスクのスタック領域のみアクセスを許可する。

タスク１実行時



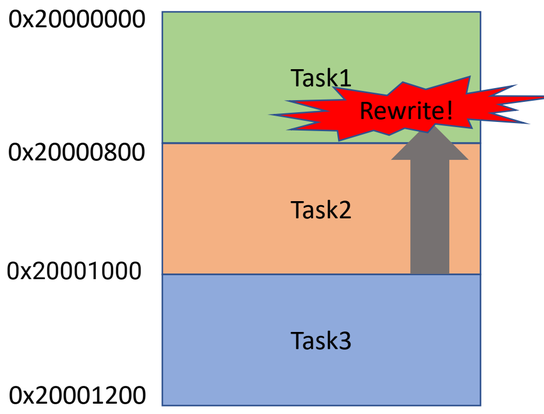
タスク２実行時



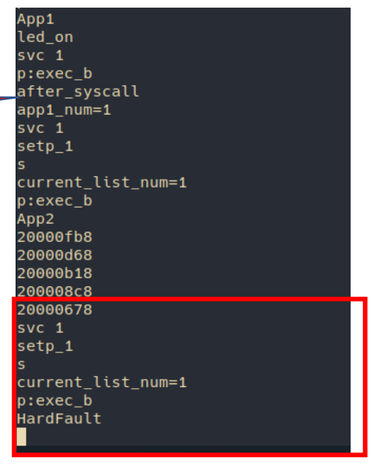
- 実際にスタックオーバフローを検知できるか検証する

  Task2(App2)で内部で配列のローカル変数を定義する再帰関数をfor文で呼び出して意図的にスタックオーバフローさせる。

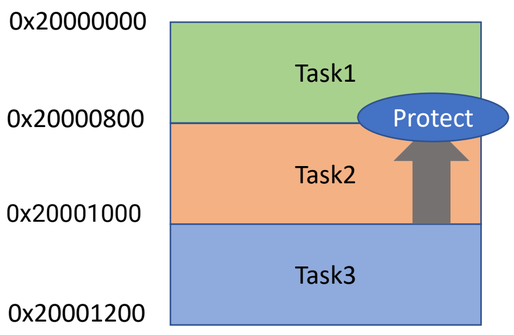
スタック領域のメモリマップ



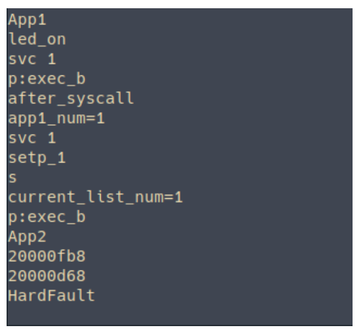
スタックオーバフロー
検出機構無効時



Task1のスタックまで書き込んでしまっており、スタックオーバフローしてしまっている。



スタックオーバフロー
検出機構有効時



Task2のスタックオーバーフローを検知、直後に例外処理が行えている。

セキュリティ的な特徴

- Rustで実装しておりunsafeで明示された領域以外では未定義動作を引き起こさないことを保証できる。言語仕様上、Cよりメモリ安全性を担保しやすく、**セキュリティに関するバグが混入しづらい実装**となっている。
- MPUを活用してスタックオーバフローを検知する機構を導入することで、他タスクのスタック領域への書き込みを防いだり、**スタックフレーム書き換えによる任意コード実行を防ぐ。**

感想と今後

- Rustと組み込みOSの学習が同時に行えて非常に勉強になった。Rustでの循環参照が難易度高く、スケジューラの実装に少し手こずった。
- 今後は以下の機能を追加実装していきたい。
  - タスク間通信
  - タスク処理時間管理
  - ヒープアロケータ
- TPMを用いたセキュアブート並びにブートローダの実装や、自作組み込みOSを搭載したマイクロマウス制作も行ってみたい。
- 今後の開発は以下のブログにまとめていく予定。
  - https://tosonshirley.hatenablog.com/