Rust の高信頼組込みシステムへの適用性評価

2020SC101 内田裕貴
指導教員：本田晋也

1　はじめに

現在，様々なプログラミング言語で組込みシステムの開発が行われている．その中で、最も広く使用されている言語は C 言語である．

しかし，C 言語には安全性の問題があると言われている．例えば，定義した配列の範囲外へアクセスする処理を記述できてしまい，実際にコンパイルや実行もできてしまう．この場合，配列の範囲外のメモリにある他の重要な変数を上書きしたり，プログラムが異常終了したりと，どのような動作をするか分からないプログラムになってしまう．

このような C 言語の問題点を解決するために，これまで様々な言語が開発されている．Java や Python は仮想マシン上で実行することで安全性を実現するが，C 言語と比較して実行性能が低いという問題がある [1]．近年，これらの問題を解決し C 言語に置き換わる安全性を重視した言語として Rust が開発されている．Rust には，所有権やライフタイムなどの，安全性を確保するための独自の概念がある．また C 言語と比較して実行速度が変わらないという特徴を持つ [2]．

組込みシステムの高性能と安全性を実現するために C 言語から Rust に置き換える方法が考えられる．しかしながら，組込みシステム開発は，通常のプログラム開発と異なり，特殊な開発ツールが必要となることから，これまで Rust の適用例は少なく，適用性評価はなされていない．

本研究では，C 言語で記述された組込みシステムを Rust で実現し，両者の記述を比較する．

2　背景技術

2.1　組込みシステムの安全性

組込みシステムでは，ソフトウェアの不具合が機器の重大な事故やトラブルに繋がるため，高い安全性が要求される．アリアン 5 型ロケットでは，システムの仕様と設計に齟齬が生じた結果，水平速度に関する演算を行なっている際にオーバーフローし，最終的には空中で爆発した．

2.2　Rust

Rust は性能やメモリ安全性を高め，C 言語に代わることを目指して開発されたプログラミング言語である．Rust は独自の構文構造や制約, 概念を持ち, コンパイラのチェック機能が強力であるため，コンパイル時に欠陥を発見しやすく，バグが発生しにくい．具体的には，C 言語のポインタの記述を Rust では借用を用いることで，適切なタイミングでメモリが解放され，メモリリークやバッファオーバーランが起こらないようなメモリ安全なプログラムを書くことができる．また，グローバルな変数への代入は直接は記述できず，危険なコードであることを示す unsafe ブロック内にのみ記述することが可能である．unsafe ブロックによりテストやデバッグの際に，重点的に確認する必要となる箇所が明確となる．

```
 1 unsigned char Event;
 2 unsigned int Slide1;
 3 ...
 4 void main(void) {
 5 ...
 6         switch_process();
 7 ...
 8 }
 9
10 void switch_process(void){
11     unsigned char sw;
12
13     sw = gpio_get(SW_SLIDE_PIN);
14     if (Slide1 != sw) {
15         if (sw == 1) {
16             Event |= EVT_TIMER_START;
17         } else {
18             Event |= EVT_TIMER_STOP;
19         }
20         Slide1 = sw;
21     }
22 ...
23 }
```

図 1　キッチンタイマーの C プログラム

3　記述性の評価

3.1　キッチンタイマーの仕様

キッチンタイマーを対象とする組込みシステムとした．仕様の一部を次に示す．

- スライドスイッチ 1 が ON になると，設定時間 30 秒でタイマをスタートする．
- スライドスイッチ 1 が OFF になると，タイマをストップする．
- タイマの動作中にプッシュスイッチ 1 が ON になると，LED2 を点灯させ，設定時間を 30 秒延長する．
- 設定時間が経過すると，LED4 を 15 秒点滅させた後，消灯する．この点滅も 0.25 秒間隔とする．

3.2　評価環境

マイコンは Raspberry Pi Pico を用いた．コンパイラは arm-none-eabi-gcc（Ver.9.2.1）と rustc（Ver.1.70.0）を用いた．

3.3　C 言語での記述

C 言語記述のプログラム片を図 1 に示す．スイッチの状態によってグローバル変数 Event を更新する関数である switch_process() を作成し，main 関数内の無限ループ内で呼び出している．スイッチの状態は SDK に用意されて

```
1 static mut Event: u32 = 0;
2 ...
3 struct SlideProcess{switch:DynPin, sw:u32, slide1:u32,}
4 ...
5 impl SlideProcess {
6     ...
7     fn slide_process(&mut self) {
8         self.sw = if self.is_high().unwrap() {1} else {0};
9         if self.slide1 != self.sw {
10            if self.sw == 1 {
11                unsafe {Event |= EVT_TIMER_START;}
12            } else {
13                unsafe {Event |= EVT_TIMER_STOP;}
14            }
15            self.slide1 = self.sw;
16        }
17    }
18 }
19 ...
20 #[entry]
21 fn main() -> ! {
22     ...
23     let mut slide_process = SlideProcess::new(pins.gpio5.
          into());
24     ...
25     slide_process.slide_process();
26     ...
27 }
28 ...
```

図 2 キッチンタイマーの Rust プログラム

いる gpio_get() 関数にピンの ID を指定して読み出す.

3.4 Rust での記述

適用の第一段階として Rust 固有の機能は使わず，可能な限り C 言語と構造等を似通わせて記述した．Rust では、GPIO ピンをオブジェクトとして定義する必要がある．この定義は main 関数でしか行えなかったため，switch_process() には，借用という値を参照するための仕組みを用いて引数として渡している．さらに，Rust ではグローバル変数の読み書きは unsafe であるため，変数 Slide1 及び Event を読み書きする箇所は unsafe ブロック内に記述している

3.5 クラス化

3.4 のプログラムをクラス化して記述したプログラム片を図 2 に示す．まずオブジェクトを作成するために，3 行目で構造体を定義している．5∼18 行目では，構造体上で動作する関数を定義している．23 行目では new 関数を呼び出すことでオブジェクトを作成し，25 行目では構造体に実装した関数を呼び出している．クラス化をしたことで変数 slide1 はメンバ変数になったが，変数 Event は複数の関数で読み書きする必要がある．そのためグローバル変数とする必要があり，読み書きをする処理は unsafe 内に記述する必要がある．

3.6 Mutex 記述

グローバル変数を使いつつ unsafe を使わずに記述するため，Mutex を使った．プログラムの構造は図 2 と同じである．グローバル変数を Mutex 型とすることで，グローバル変数を読み書きする記述がやや煩雑にはなるが，unsafe を使わずに記述することができた.

3.7 評価

評価結果を表 1 に示す．表 1 の安全性は，unsafe ブロックの使用数のことである．記述量は関数記述が一番少なく，クラス化記述と Mutex 記述はほとんど差がなかった．その一方で，関数記述はグローバル変数を更新する処理が多いため，unsafe ブロックを多く使う必要があり，安全性については他の記述よりも低いと言える．また，関数記述は引数などの記述が煩雑である点や，クラス化記述や Mutex 記述は main 関数のループ内の記述が簡潔である点から，クラス化記述や Mutex 記述の方が関数記述より可読性が高いと評価した.

表 1 評価結果

	記述量 (行)	安全性 (個)	可読性
関数記述	215	19	△
クラス化記述	338	8	○
Mutex 記述	339	0	○

4 実行性能評価

C 言語と Rust の性能評価のためのテストプログラムとして，表 2 中の 2 つを作成し，実行時間や応答時間の測定を行い，最悪値を求めた．計測結果は表 2 の通りである.

表 2 計測結果 (ns)

評価項目	C	Rust
スイッチプログラムの応答時間	112	112
LED トグルプログラムの実行時間	32	36

C 言語と Rust の実行性能には大きな差はなかった．Rust は，C 言語のようにガベージコレクションを使わず，またコンパイル時に機械語に変換されて実行時にチェックが行われないためであると考えられる.

5 おわりに

今回は，複数の記述方法の記述性を評価した．クラス化を行うことで，行数は増えたものの，煩雑な記述を排除し，さらに unsafe を使わずに記述することができた.

参考文献

[1] 板田怜子, 藤田一希, 横山哲郎, 本田 晋也,”マイクロコントローラ向けスクリプト言語 MicroPython の組込みシステムへの適用性評価”, 情報処理学会研究報告, Vol.2021-EMB-58,No.4, pp. 1-8, 2021.

[2] I. Plauska, A. Liutkevičius and A. Janavičiūtė,”Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller”, Electronics, 12(1), 143, 2023.