第6章 仕組みがカバーしてくれること/してくれないこと

Rustの安全性の考察

中林 智之

前章では，スクラッチでRustのファームウェアを書いてみました．その中で出てきたのがunsafeです．Rustの安全性とは何なのか，unsafeとは何なのか，1度ここで整理してみます．

Rustは安全なプログラミング言語です．しかし，何を持って安全なのでしょうか？

Rustではコンパイルできたプログラムに対して型安全性，メモリ安全性，スレッド安全性を保証します．

メモリ安全性は型安全性に含まれる，という考え方が多いようです．今回はメモリ安全性を強調するために，あえて分けています．

型安全性

型安全性とは何でしょうか．筆者はプログラミング言語の専門家ではないので，文献(1)の型安全性とは何か，から引用します．

この記事によると型安全性のイメージは，「正しく型付けされたプログラムは不正な動作をしない」です．少し雑な解釈ですが，コンパイルに成功したプログラムが，未定義動作を起こさないことと筆者は捉えています．

C言語は型安全なプログラミング言語ではありません．コンパイルできたC言語は，いとも簡単に未定義動作を起こします．C言語の未定義動作を1つ示します．

リスト1は，add関数の引き数の個数を間違って呼び出しています．このプログラムは警告が出るものの，コンパイルできてしまいます．C言語では，関数プロトタイプ宣言のない関数呼び出しで，実引き数の数が仮引き数の数と等しくない場合には，その動作は未定義となっています．

gcc 9.2.0を使った場合のコンパイラの出力と実行結果は**リスト2**の通りです．なお，表示される数値は実行するたびに異なります．

C言語の型システムは，このように明らかに誤ったプログラムであっても，コンパイルに成功してしまいます．Rustでは，同様のプログラムはコンパイル・エラーになります．

リスト1　C言語の型システムは間違っていてもいとも簡単にコンパイルに成功することを示すためのプログラム

```
#include <stdio.h>
#include <stdint.h>

int main(void) {
    printf("%d\n", add(1));
}

int32_t add(int32_t a, int32_t b) {
    return a + b;
}
```

リスト2　リスト1のプログラムは誤っているのにコンパイルが通って実行できてしまう

```
prog.c: In function 'main':
prog.c:5:20: warning: implicit declaration of
    function 'add' [-Wimplicit-function-declaration]
     5 |       printf("%d\n", add(1));
       |                       ^~~

-584809287
```

メモリ安全性

メモリ安全であるとは，次のような現象が発生しないことを，言語処理系が保証することを意味します．

- バッファ・オーバラン
- ヌル・ポインタのデリファレンス
- use after free
- 未初期化メモリの使用
- 不正なfree(double freeや確保されていないメモリのfree)

言語処理系が保証する，と書いたのには理由があります．全てのバグをコンパイル・エラーとして発見できるわけではなく，バグ検出の幾つかは，実行時に行われます．実行時のチェックによって上記バグを発見し，定義された挙動でプログラムがクラッシュする場合，それもメモリ安全(型安全)である，とみなされます．