第1章 高性能で安全なプログラムを効率よく書ける

C/C++の後継的言語 Rustの登場

中林 智之

背景

● マイコン/組み込みプログラムはC/C++以外なかった

Rustは，C/C++（どちらかというとC++）と真っ向から競合するプログラミング言語です．C/C++の安全性に関わる問題を人類の英知で克服しようとする正統派な言語だと感じることから，筆者の推しプログラミング言語はRustです．まず，Rustの特徴を紹介します．

現在，多くの組み込みシステムはC/C++で書かれています．これは，ハードウェアを直接制御する組み込み開発において，ポインタ操作が必要不可欠という観点があります．

また，性能や効率，バイナリ・サイズの小ささが組み込みシステムでは重視されるからです．

ハードウェア・リソースが比較的リッチな組み込みシステムにおいて，JavaやPython（MicroPython）といった言語が使われる場合もあります．しかし，ハードウェア・リソースが乏しく，低消費電力性やリアルタイム性が求められるようなマイコン開発では，実質的にC/C++以外の選択肢がない状態でした．

● C/C++の課題…メモリ操作やセキュリティの責任はプログラマに

近年，IoTが注目を集め，従来はスタンドアロンで動作していた組み込みシステムが，ネットワークに接続して動作するのが当たり前になりつつあります．そこで問題になってくるのがセキュリティです．末端の組み込みシステムがハッキングされるとネットワーク経由で重大な事故につながる可能性があります．

残念ながら，C/C++（特にC）の言語仕様はセキュリティを考慮しておらず，肥大化する組み込みシステムにおいて脆弱性のないコードを書くことが難しい状況です．これは，C/C++が開発された当時，セキュアなプログラムを作ることよりもコンパイラの軽さやターゲット環境での性能発揮が重要であったためです．

しかしIoTでは，組み込みシステムにおいてもセキュアなソフトウェア・システム構築が求められています．

C/C++において脆弱性の原因となるのは，未定義動作，特にメモリ操作に関連するものです．C/C++では，メモリの確保，解放，境界の管理などメモリ操作を正しく行う責任が，全てプログラマにあります．しかし，完璧にメモリ操作を行うことは，歴戦のプログラマでも困難です．C++では，C++11以降，メモリの確保，解放をほとんど自動化できるようになりましたが，未定義動作が発生する余地が残ったままです．

● C/C++以外の選択肢Rustの登場

マイコン開発では実質的にC/C++以外の選択肢がない状態でしたが，状況は変化しつつあります．組み込み開発においてもC/C++に代わる選択肢が生まれようとしています．その筆頭が今から紹介するRustです．

Rustは2015年にバージョン1.0が出たコンパイル型のプログラミング言語です（**図1**）．

ここ1年は特に，Rustを使った数々のプロダクトがリリースされ，C/C++を代替し得るプログラミング言語として，注目と期待を集めています．Rustは，C/C++のようなパフォーマンスを発揮しながら，ポインタのような低レベルな操作を用いてハードウェアを直接制御できます．

さらにC/C++と決定的に異なる点は，安全性を重視していることです．Rustでは脆弱性につながる多くのバグを，「コンパイル時」に検出します．Rustで書いたプログラムでは，メモリ破壊のようなバグにほとんど遭遇しません．バグをコンパイル時に検出することで，実行時のオーバヘッドなしに，安全な実行バイナリを生成できます．Rustはオブジェクト指向言語や関数型言語のパラダイムを取り入れており，効率的にソフトウェアが構築できるのも大きな特徴です．