強い型によるOSの開発手法の提案

岡部 究[1,a)]　水野 洋樹[2,b)]　瀬川 秀一

概要：現在でも OS は C 言語によって設計されている．一方アプリケーションは強い型付けの言語を用いた安全な設計手法が確立されている．本稿では OS の安全な設計手法として，C 言語によって設計された OS のソースコードを元に少しずつ型推論をそなえた言語による実装に置き換えるスナッチ設計という手法を提案する．また当該手法を小規模 OS に対して適用し，その結果を考察する．

キーワード：プログラミング・シンポジウム，プログラミング言語，コンパイラ，Haskell，OS

1. はじめに

筆者らは実用可能な OS を開発可能な型推論をそなえたコンパイラを目指し，Ajhc Haskell コンパイラ [1] を開発している．本稿では Ajhc を使った OS の開発手法を提案し，その手法を小規模 OS に対して適用/評価する．最後に今後の研究計画について述べる．

2. OS 開発における既存手法の問題

ソフトウェア開発において実行時エラーの削減は重要なテーマであり，様々な手法が提案されている．その 1 つとして開発に強い型付けの言語を用いる手法が提案されており，アプリケーション領域では実用化されている．一方 Linux や BSD のような実用化された OS の開発において主に使われているプログラミング言語は C 言語である．C 言語は ML や Haskell のような型推論をそなえた言語より弱い型付けであるためにしばしば実行時エラーを引き起こす．研究レベルにおいて，型推論をそなえた強い型付け言語で OS を設計する試みは複数存在する [2][3][4]．しかしこれらの OS は Linux や BSD のようにデスクトップ/サーバ用途として実用化されていない．筆者らは上記の OS が実用化されていない原因は主に 3 つに大別できると考える．

1 つ目は，実用化に辿り着くまでプロジェクト参加者の気力が継続できないということである．実用化された Linux のような OS は Linux kernel の動作するデスクトップで Linux kernel 自体を開発している．彼等はまた開発以外の日常のデスクトップ用途にも Linux kernel を使用する．このような開発スタイルはドッグフード開発と呼ばれている．この開発サイクルによって自然に実用上のテストが行なわれ，OS の品質は向上する．OS 開発ではこのドッグフード開発にすばやく辿り着く必要がある．

2 つ目は，ハードウェア割込をポーリングで検出していることである．再入可能な言語で開発されていないため，ハードウェア割込は言語のランタイムで受け取り，OS 実装側は定期的にランタイムにためられたイベント通知を引き上げる必要がある．実用化された OS のほとんどは割込をイベントドリブンで OS 実装が直接引き上げる．そのため UNIX 誕生から長年つちかわれた OS 設計のノウハウを捨てて，まったく新しい設計を行なわなければならない．

3 つ目は，既存の C 言語デバイスドライバと共存させることができないことである．OS の存在意義はアプリケーションを動作させることであるが，そのためにはコンピュータに接続されたデバイスを抽象化してアプリケーション側に見せる必要がある．世界には膨大な種類のデバイスが存在し，それぞれに異なるドライバ実装が必要である．これらの OS では必要なドライバを全て再実装する必要がある．

3. 本稿で提案する開発手法

筆者らは上記の問題を解決するために**スナッチ設計**という手法を提案する．スナッチ設計では OS をゼロから設計せず，既存の C 言語によって設計された OS のソースコードを元に少しずつ型推論をそなえた言語による実装に置き換える．この手法によってドッグフード開発をしながら強い型を持つ言語による OS 開発が可能で，さらに既存のデバイスドライバを再利用することもできる．

既存の再入可能でないコンパイラでは，イベントドリブンで設計された OS をスナッチ設計することができない．そこでスナッチ設計を行なうためのコンパイラを作成する基礎となるコンパイラを選定するために “hoge” と印字する

1 Metasepi Project
2 ocaml-nagoya
a) kiwamu@debian.or.jp
b) mzp@ocaml.jp

表 1 “hoge” と印字するプログラムに見るコンパイラの特性

コンパイラ実装	サイズ	未定義シンボル	依存ライブラリ
GHC-7.4.1	797228 B	144 個	9 個
SML#-1.2.0	813460 B	134 個	7 個
OCaml-4.00.1	183348 B	84 個	5 個
MLton-20100608	170061 B	71 個	5 個
jhc-0.8.0	21248 B	20 個	3 個

だけのプログラムをコンパイルして比較評価した（**表 1**）．各評価値が小さいほど POSIX 依存度が低いことを示している．評価対象の実装の中では jhc[5] が良い特性を持っていることがわかる．しかし jhc は再入可能なバイナリを扱えず，スレッドさえサポートされていない．そこで筆者らは jhc にスレッドと再入可能サポートを追加した Ajhc コンパイラを開発した [6]．

4. 開発手法の評価と考察

スナッチ設計と Ajhc の組合せを Linux や BSD のような実用的な OS の開発に投入する前に手法の評価を行なった．

最初に，RAM サイズが 40kB のマイコン上で OS を搭載せずに直接 Haskell コードを動作させた [7]．この評価では通常コンテキストとハードウェア割込コンテキストの両方を Haskell 言語で実装した．コンテキスト間の通信はポインタを経由するが Ajhc の再入可能拡張が正常に動作することを実証した．また jhc の小さなバイナリを生成する特性が OS の省メモリ化に寄与することも実証できた．

次に，組込み向け OS の上に小さな TCP/IP プロトコルスタックを搭載し，その上で動作するネットワークアプリケーションを Haskell で作成した．OS とプロトコルスタックの C 言語実装と Haskell 実装が協調動作させることが可能で，まだ Haskell 化が進んでいない段階でもドッグフード開発に移行できることを実証できた．

最後に，組込み向け OS の一つである ChibiOS/RT[8] のスレッドを使って Haskell 言語のスレッドを実装した [9]．Ajhc の Haskell スレッド実装を，ランタイム側に用意されている forkOS_createThread API を独自ものに差し換えれば POSIX スレッド以外の OS 側スレッド実装を使って実現できるように変更した（図 1）．Ajhc が再入可能であるだけではなくスレッドセーフであることを実証した．

5. 結論と今後の課題

スナッチ設計と Ajhc の組合せによって 2 章での OS 実用化の 3 つの障壁を解決する見込みがたった．しかしその過程でいくつかの問題が見つかった．1 つ目に，C 言語の構造体のように構造の一部を局所的に更新する簡単な方法がないことがある．これは GHC で使われている vector[*1] ライブラリを Ajhc に移植することで解決できると考えられる．2 つ目に，コンテキスト間の状態共有方法がポインタのみでしか許されず，型を使った状態共有ができない．これは MVar[*2] を Ajhc で使用可能にすべきである．

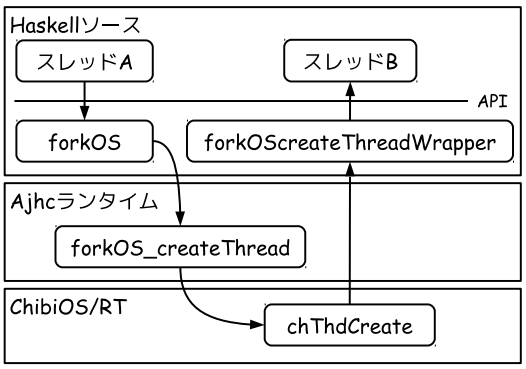

図 1 RTOS のスレッドを利用した Haskell スレッド実装

今後は上記 2 つの問題を解決した後，スナッチ設計によって NetBSD kernel の一部のデバイスドライバを Haskell 化する予定である．また jhc 以外のスナッチ可能なコンパイラとして有望に思われる ATS 言語 [10] の調査を行なう．

謝辞 誰もが望みを捨ててしまっていた OS 領域にかすかな光をもたらした John Meacham に感謝する．

参考文献

[1] team, M.: Ajhc - Haskell everywhere, Metasepi Project (online), available from ⟨http://ajhc.metasepi.org/⟩ (accessed 2013-03-01).

[2] funk team, T.: Funk The Functional Kernel, The funk team (online), available from ⟨http://home.gna.org/funk/⟩ (accessed 2005-09-16).

[3] jessica.l.hamilton: snowflake-os An O’Caml Operating System, snowflake-os members (online), available from ⟨https://code.google.com/p/snowflake-os/⟩ (accessed 2012-05-23).

[4] Hallgren, T. et al.: A Principled Approach to Operating System Construction in Haskell, *ICFP 2005* (2005).

[5] Meacham, J.: Jhc Haskell Compiler, (online), available from ⟨http://repetae.net/computer/jhc/⟩ (accessed 2013-11-18).

[6] 岡部究：めたせぴ☆ふぁうんでーしょん，簡約! λカ娘， Vol. 5, pp. 3–44 (2013).

[7] team, M.: Ajhc demo for Cortex-M3/4 board, (online), available from ⟨https://github.com/ajhc/demo-cortex-m3⟩ (accessed 2013-11-20).

[8] Sirio, G. D.: ChibiOS/RT Homepage, (online), available from ⟨http://www.chibios.org/⟩ (accessed 2013-11-20).

[9] team, M.: Snatch ChibiOS/RT using Ajhc, (online), available from ⟨https://github.com/metasepi/chibios-arafura⟩ (accessed 2013-11-20).

[10] Xi, H.: The ATS Programming Language, Boston University (online), available from ⟨http://www.ats-lang.org/⟩ (accessed 2013-11-18).

[*1] http://hackage.haskell.org/package/vector

[*2] http://hackage.haskell.org/package/base/docs/Control-Concurrent-MVar.html