継続を用いたxv6 kernelの書き換え

坂本昂弘[†1]　桃 原 優[†1]　河野真治[†1]

概要：xv6 は MIT で教育用の目的で開発されたオペレーティングシステムで基本的な Unix の構造を持っている。当研究室で開発している Continuation based C (CbC) は継続を中心とした言語で、用いることにより検証しやすくなる。xv6 を CbC で書き換えることで、 オペレーティングシステムの基本的な機能に対する検証を行うために書き換えを行う。

1. はじめに

OS はさまざまなコンピュータの信頼性の基本である。OS の信頼性を保証する事自体が難しいが、時代とともに進歩するハードウェア、サービスに対応して OS 自体が拡張される必要がある。さらに非決定的な実行を持つ為、モデル検査は無限の状態ではなくても巨大な状態を調べることになり、状態を有限に制限したり状態を抽象化したりする方法が用いられている。 xv6 はシンプルで扱いやすい基本的な構造を持つ OS である。この xv6 から Stack を排除することで OS の状態を有限化させることが可能となる。当研究室で開発している Continuation based C を用いて xv6 kernel を書き換えることで OS の状態を有限化可能であるかを検討する。

2. Continuation based C

2.1 CbC の概要

CbC は処理を Code Gear とした単位を用いて記述するプログラミング言語である。 Code Gear 間では軽量継続 (goto 文) による遷移を行うので、継続前の Code Gear に戻 ることはなく、状態遷移ベースのプログラミングに適している。図 1 は Code Gear 間 の処理の流れを表している。

現在 CbC は C コンパイラである GCC 及び LLVM をバックエンドとした clang、MicroC コンパイラ上で実装されている。今回 xv6 の kernel の部分をこの CbC を用いて書き換える。

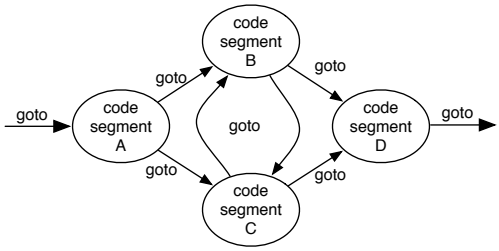

図 1: goto による code gear 間の継続

2.2 Code Gear

Code Gear は CbC における最も基本的な処理単位である。ソースコード 1 は CbC における Code Gear の一例である。

ソースコード 1: code segment の軽量継続

```
1 __code cs0(int a, int b){
2   goto cs1(a+b);
3 }
4 __code cs1(int c){
5   goto cs2(c);
6 }
```

Code Gear は__code Code Gear 名 (引数) の形で記述される。Code Gear は戻り値を持 たないので、関数とは異なり return 文は存在しない。次の Code Gear への遷移は goto Code Gear 名 (引数) で次の Code Gear への遷移を記述する。ソースコード 1 での goto cs1(a+b); がこれにあたる。この goto の行き先を継続と呼び、このときの a+b が次の Code Gear への出力となる。Scheme の継続と異なり CbC には呼び出し元の環境がない ので、この継続は単なる行き先である。したがってこれを軽量継続と呼ぶこともある。cs1 へ継続した後は cs0 へ戻ることはない。軽量継続により、並列化、ループ制御、関数コー ルとスタックの操作を意識した最適化がソースコードレベルで行えるようにする。CbC は軽量継続による遷移を行うので、継続前

[†1] 現在，琉球大学工学部情報工学科
Presently with Information Engineering, University of the Ryukyus.
[†2] 現在，琉球大学大学院理工学研究科情報工学専攻
Presently with Interdisciplinary Information Engineering, Graduate School of Engineering and Science, University of the Ryukyus.

の Code Gear に戻ることはなく、状態 遷移ベースのプログラミングに適している。

2.3 Data Gear

Data Gear は CbC におけるデータの単位である。CbC では Code Gear は Input Data Gear、Output Data Gear を引数に持ち、任意の Input Data Gear を参照し、Output Data Gear を書き出す。図 2 Code Gear はこのとき渡された引数の Data Gear 以外を参照することはない。この Data Gear の対応から依存関係の解決を行う。

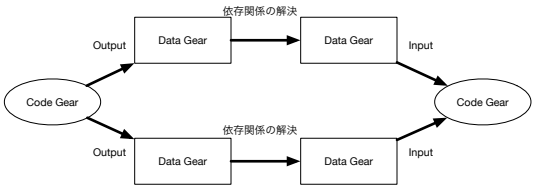

図 2: CodeGear と DataGear

3. Gears OS の概要

Gears OS は Code Gear、Data Gear の単位を用いて開発されており、CbC で記述されている。Gears OS は Context と呼ばれる使用されるすべての Code Gear、Data Gear 持っ ている Meta Data Gear を持つ。Gears OS は継続の際この Context を常に持ち歩き、必要な Code Gear、Data Gear を参照したい場合、この Context を通して参照する。

3.1 Context

Context は実行する Code Gear と Data Gear を全て持っており、通常 OS のプロセスに相当する。また Context は Data Gear を構造体で定義しており、その全てを union と定義している。Context 内で Code Gear と Data Gear は enum により番号で管理されている。

3.2 Interface

Interface は Gears OS のモジュール化の仕組みである。Interface は呼び出しの引数になる Data Gear の集合であり、そこで呼び出される Code Gear のエントリである。呼び出される Code Gear の引数となる Data Gear はここで全て定義される。Interface を定義 することで複数の実装を持つことができる。CbC は関数呼び出しと異なり、goto による継続で遷移を行う。このため CbC の継続にはスタックフレームがなく引数を格納する場所がない。Context は初期化の際に引数格納用の Data Gear の領域を確保する。Code Gear が継続する際にはこの領域に引数の Data Gear を格納する。この領域に確保された Data Gear へのアクセスは Interface の情報から行われる。

4. xv6 の CbC への書き換え

xv6 は 2006 年に MIT のオペレーティングシステムコースで教育用の目的として開発されたオペレーティングシステムである。xv6 はプロセス、仮想メモリ、カーネルとユーザ の分離、割り込み、ファイルシステムなどの基本的な Unix の構造を持つにも関わらず、シンプルで学習しやすい。CbC は 継続を中心とした言語であるため状態遷移モデルに落とし込むことができる。 xv6 を CbC で書き換えることにより、OS の機能の保証が可能となる。

4.1 書き換え

5. 結論

5.1 現状

5.2 今後

5.3

参考文献

[1] 宮城光希: **継続を基本とした言語による OS のモジュール化**. 琉球大学理工学研究科修士論文、2019