CbC による xv6 の FileSystem の書き換え
Rewriting xv6 FileSystem by CbC

学籍番号：165723C 氏名：坂本昂弘 指導教員：河野真治

2020 年 2 月 17 日

Abstract

The OS is required to guarantee reliability and expandability. Gears OS is being developed using Continuation based C (CbC) with the goal of guaranteeing reliability for normal-level computation and achieving scalability by meta-level computation. Although it is simple as a previous step, it rewrites xv6, which is an OS that has the basic structure of Unix such as process, virtual memory, separation of kernel and user, interrupt, file system, etc., with CbC. In this paper, we focused on the xv6 FileSystem and rewrote it with CbC.

1 OS に対する信頼性の保証

OS には信頼性の保証と拡張性の実現が求められている。信頼性をノーマルレベルの 計算に対して保証し、拡張性をメタレベルの計算で実現することを目標に Continuation based C[1] (CbC) を用いて Gears OS[2] を開発中である。CbC は Code Gear という基本的な処理の単位と Data Gear というデータの単位を用いる。Code Gear に対して入力の Data Gear と 出力の Data Gear が存在し、入力に対して期待される出力がされてるか検査することで信頼性を保証する。CbC の Interface は Gears OS のモジュール化の仕組みである。Interface を使うことで検証や機能の入れ替えによる拡張が可能となる。前段階としてシンプルであるがプロセス、仮想メモリ、カーネルとユーザーの分離、割り込み、ファイルシステムなど Unix の基本的な構造を持っている OS である xv6[3] を CbC で書き換えている。本論文では、xv6 の FileSystem を CbC によって書き換えることにより 複雑な処理である FileSystem を明確化させ信頼性を保証、 Interface を使用可能とすることで拡張性を実現することを目標とする。

2 xv6

xv6 とは MIT のオペレーティングコースの教育目的で 2006 年に開発されたオペレーティングシステムである。xv6 はオリジナルである v6 が非常に古い C 言語で書かれている為、ANSI-C に書き換えられ x86 に再実装された。 xv6 は read や write などの systemcall、プロセス、仮想メモリ、カーネルとユーザーの分離、割り込み、ファイルシステムなど Unix の基本的な構造を持っている。

3 Continuation based C

xv6 kernel 上で interface を実装する際、当研究室で開発されたプログラミング言語 Continuation based C (以下 CbC) を用いる。CbC は基本的な処理単位を CodeGear として定義し、CodeGea 間で遷移するようにプログラムを記述する C 言語と互換性のあるプログラミング言語である。CodeGear は返り値を持たない為、関数内で処理が終了すると呼び出し元の関数に戻ることがなく別の CodeGear へ遷移する。以下の Code1 に CodeGear 遷移時のコード例を示す。

```
__code cg0(Integer a, Integer b){
 int a_v = a->value;
 int b_v = b->value;
 Integer c = {a_v + b_v};
  goto cg1(c);
}
__code cg1(Integer c){
  goto cg2(c);
}
```

Code 1: CodeGear の継続の例

4 context

context とは一連の実行が行われる際に使用される CodeGear と DataGear の集合である。従来のスレッドやプロセスに対応する。Context は接続可能な CodeGear、Data Gear のリスト、Data Gear を確保するメモリ空間、実行される Task への Code Gear 等を持っている。CodeGear が別の CodeGear に遷移する際、必ず context を参照し enum で定義された CodeGear の番号を指定し遷移する。

5 CbC の Interface

先述した通り、CbC の 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。この Interface は、Java の Interface や Haskell の型クラスに対応し、導入することで仕様と実装に分けて記述することが出来る。

6 xv6 の FileSystem

FileSystem とは、コンピュータの資源を操作するための OS が持つ機能のことである。 ファイルといえば記憶装置内に格納されている情報を指すが、xv6 の FileSystem は、デバ イスやプロセス、カーネル内の処理をする際の情報などをファイルとして扱う。OS ごとに 利用している FileSystem は異なるが、一部の OS を除きほとんどの OS には FileSystem が存在する。

7 CbC による xv6 FileSystem の書き換え

xv6 FileSystem の CbC による書き換え方針は、FileSystem が記述されている fs.c で定義されている 関数を CbC によって書き換えることにより処理の明確化と信頼性の保証をしたい。そのために今回は、Basic Block 単位に書き換えを行った。また、CbC の Interface を用いることにより仕様と実装を分離し拡張性を実現する。まず、Filesystem Interface を定義し、それに対応した CodeGear を用いることによって Interface を実装していく。Interface を実装する際に明示的に次の遷移先が決まっている場合、Java の private メソッドのように扱うため別に分けて private 実装してやる。その際 interface の時と同じようにヘッダーファイルを作成し、使う CodeGear を全て定義する必要がある。どのような処理の流れをするか図 1 に示す。

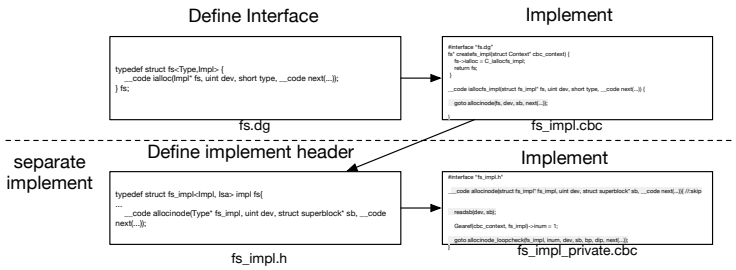

図 1

8 まとめと今後の課題

本研究では xv6 の FileSystem 部分について CbC を用いて書き換えを行った。実際に FileSystem を CbC で書き換えることによって、if 文と for 文を切り出してやるができた。 さらに、FileSystem を Interface とその実装に書き換えることによって仕様と実装に分け、拡張性を高めることができた。xv6 の FileSystem 部分書き換え後 、デバックをまだ行っていないため正常に動くかどうか確認することが求められる。 また、正常に動作しなかった場合は修正を行い、OS として機能しているか再確認する必要がある。]

参考文献

[1] Kaito TOKKMORI and Shinji KONO. Implementing continuation based language in llvm and clang. *LOLA 2015*, July 2015.

[2] 河野真治, 伊波立樹, 東恩納琢偉. Code gear、data gear に基づく os のプロトタイプ. 情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS), May 2016.

[3] Russ Cox, M Frans Kaashoek, and Robert Morris. Xv6, a simple unix-like teaching operating system, 2011. (2020 年 2 月 7 日閲覧).

[4] 伊波立樹, 河野真治. Gears os の並列処理. 琉球大学工学部情報工学科平成 30 年度学位論文 (修士), 2018.

[5] 宮城光希, 河野真治. 継続を基本とした言語による os のモジュール化. 琉球大学工学部情報工学科平成 31 年度学位論文 (修士), 2019.