GearsOSの分散ファイルシステムの設計

一木貴裕[,a)]　河野 真治[,b)]

概要：本研究室では gear というプログラミング概念を持つ, 分散フレームワーク Christie を開発している. Christie はノード同士が Datagear と呼ばれる変数データを送信しあうことにより, 簡潔に分散プログラムの記述を行うことができる. この Christie の仕組みを, 同様に本研究室が開発している GearsOS に組み込み, ファイルシステムを構築したい. GearsOS はノーマルレベルとメタレベルを分けて記述できる Continuation based C(CbC) で構成されており、Christie と近い仕様をもつ.

Designing a Distributed File System for GearsOS

1. GearsOS のファイルシステムの開発

当研究室では OS の信頼性の検証を目的とした OS である GearsOS を開発している. GearsOS はユーザレベルとメタレベルを分離して記述が行える言語である Continuation based C(以下 CbC) で記述されており, Gear というプログラミング概念を持つ.

GearsOS は現在開発途上であるため, 現在は言語フレームワークとしてしか動作しない。OS として起動するためにこれから実装が必要な機能が多く存在しており, その中の一つとして分散ファイルシステムが挙げられる. GearsOS の分散ファイルシステムを構成するために、当研究室が開発している分散フレームワーク Christie の仕組みを用いようと考えた.

Chrsitie は GearsOS のもつ Gear という概念とよく似た, 別の Gear というプログラミング概念を持っており, DataGear と呼ばれる変数データを接続されたノード同士が送信しあうことで分散処理を簡潔に記述することができる. DataGear は指定された型と名前を持つ key に対応しており, プログラムが必要な key にデータが揃ってから初めてプログラムが処理される. また, Chrisite は Topology-Manager と呼ばれる機能を持っており, 任意の形でノード同士の配線を行い Topology を形成する機能を持っている.

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，琉球大学理工学研究科情報工学専攻
Presently with Johoshori Uniersity
[a)] ikki-tkhr@cr.ie.u-ryukyu.ac.jp
[b)] kono@ie.u-ryukyu.ac.jp

2. 現代のファイルシステムについて

3. Continuation based C

GearsOS は C 言語の下位言語である Continuation based C を用いて記述されている. CbC は関数呼び出しでなく, 継続を導入しており, スタック領域を用いず jmp 命令でコード間を移動することにより軽量な継続を実現している. CbC ではこの継続を用いて for 文などのループの代わりに再起呼び出しを行う. 実際の OS やアプリケーションを記述する際には GCC または LLVM/clang の CbC 実装を用いる.

CbC では関数の代わりに CodeGear という単位でプログラミングを行う. CodeGear は`__code` で宣言を行い, 各 CodeGear は DataGear と呼ばれる変数データを入力として受け取り, その結果を別の DataGear に書き込む. 特に入力の DataGeat を InputDataGear, 出力される DataGear を OutputDataGear と呼ぶ. CodeGear と DataGear の関係図を図 1 に示す. CodeGear は関数呼び出しのスタックを持たないため, 一度 CodeGear を遷移すると元の処理に戻ってくることができない.

CbC コードの例をソースコード 1 に示す.

```
void syscall(void)
#include <stdio.h>
__code CG2(){
  int i = 10;
  printf("i␣=␣%d\n", i);
}
```

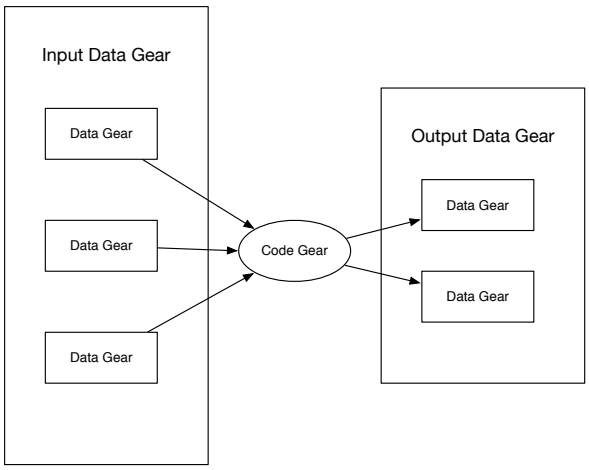

図 1: CodeGear と入出力の関係図

```
__code CG1(){
  printf("Hello\n");
  goto CG2();
}

int main(){
  goto CG1();
}
```

Code 1: CbC の例題

4. CbC を用いた OS の記述

CodeGear の遷移はノーマルレベルから見ると単純に CodeGear が DataGear を Input, Output をのみ繰り返し, コードブロックを移動しているように見える. CodeGear が別の DataGear に遷移する際の DataGear との関係性を図 2 に示す. ノーマルレベルでは DataGear を受け取った CodeGear を実行, 実行結果を DataGear に書き込み別の CodeGear に継続していると見える.

しかし, 実際には CodeGear から別の CodeGear への遷移にはデータの整合性の確認などのメタ計算が必要となる. コード間の遷移に必要となるメタ計算は, MetaCodeGear と呼ばれる CodeGear ごとに実装された CodeGear で行う. MetaCodeGear で参照される DataGear を MetaDataGear 呼び, また, CodeGear の直前に実行される MetaCodeGear を StubCodeGear と呼ぶ. これら Meta 計算部分を含めた CodeGear の遷移と DataGear の関係性を図示すると図 2 の下段の形に表せる. CordGear の実行前後に実行される MetaCodeGear や入出力の DataGear を MetaDagaGear から取り出すなどのメタ計算が加わる.

MetaCodeGear は詳細な処理の変更や, スクリプトに問題がある場合を除き, プログラマが直接実装する必要がなく, GearsOS が持つ Perl スクリプトにより, GearsOS がビルドされる際に生成される.

CodeGear の遷移に重要な役割を持つ MetaDataGear と

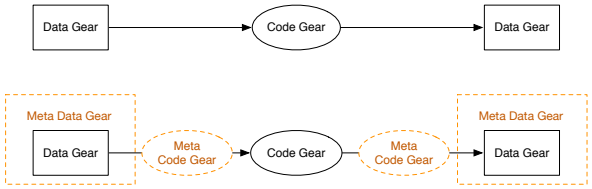

図 2: CodeGear と MetaCodeGear の関係図

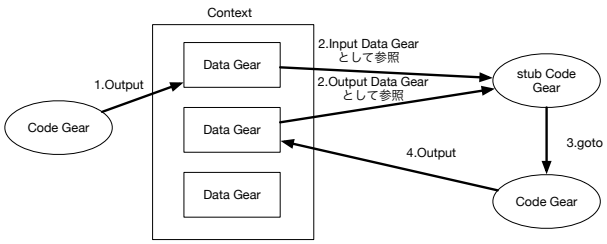

図 3: Context を介した CodeGear の継続

して context が存在する。context は遷移先の CodeGear と MetaDataGear の紐付けや, 計算に必要な DataGear の保存や管理を行う. 加えて context は処理に必要になる CodeGear の番号と MetaCodeGear の対応表や, DataGear の格納場所を持つ. context と各データ構造の役割を図 3 に示す. 計算に必要なデータ構造と処理を持つデータ構造であることから, context は従来の OS のプロセスに相当し, ユーザープログラムごとに context が存在している.

5. 分散フレームワーク Christie

Christie は当研究室で開発されている java 言語で記述された, 分散フレームワークである. Christie は CbC と似ているが異なる仕様を持つ Gear というプログラミング概念を持つ.

- `CodeGear`（以下 CG）
- `DataGear`（以下 DG）
- `CodeGearManager`（以下 CGM）
- `DataGearManager`（以下 DGM）

CodeGear はクラスやスレッドに相当する. DataGear は変数データであり, CodeGear 内で java のアノテーションを用いて記述する.

DataGear は Key と必ず対応しており, CodeGear 内の全ての Key に DataGear が揃った際に初めて CodeGear が動作するという仕組みになっている.

CodeGearManager はいわゆるノードに相当し, CodeGear, DataGear, DataGearManager を管理する. 複数の CodeGearManager 同士が配線され, DataGear を送信し合うことで分散処理を実現している.

DataGearManager は DG を管理しているもので変数プールに相当し, CodeGearManager の持っている DataGear の key と put されたデータの全てを所持している. DataGearManager は Local と Remote に区分することができ, Local-

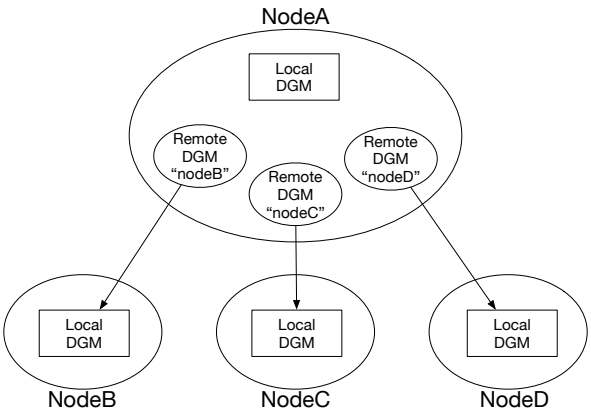

図 4: RemoteDataGear と接続ノードの関係図

DataGearManager は CodeGearManager 自身が所持する DataGear(key) のプールであり, Local に put することにより自身の持つ key に DataGear を送ることができる. 対する RemoteDataGearManager は CodeGearManager が配線されている別の CodeGearManager が持つ DataGear のプールである. つまり, 任意の接続された RemoteDataGear に DataGear を put すると対応したノードが持つ key に DataGear が送信される. RemoteDataGear に DataGear を put する処理が分散処理の肝となっている. RemoteDataGear の仕組みを図 4 に示す.

Christie の要となる DataGear の key は java のアノテーション機能が使われている. アノテーションには以下の 4 つが存在する.

Take 先頭の DG を読み込み，その DG を削除する. DG が複数ある場合，この動作を用いる.

Peek 先頭の DG を読み込むが, DG が削除されない. そのため，特に操作をしない場合は同じデータを参照し続ける.

TakeFrom(Remote DGM name) Take と似ているが，Remote DGM name を指定することで，その接続先 (Remote) の DGM から Take 操作を行える.

PeekFrom(Remote DGM name) Peek と似ているが，Remote DGM name を指定することで，その接続先 (Remote) の DGM から Peek 操作を行える.

コード 2, 3 は Christie で記述した Hello World のプログラムである. ユーザープログラムは StartCodeGear クラスを継承したクラス (コード 2) から開始する. CodeGearManager はポート番号を指定した上で creatCGM メソッドを呼び出すことにより生成される. 生成された CodeGearManager は CGM 名.setup にて CGM に処理させたいスレッド, つまり CodeGear を持たせることができる.

コード 3 は HeloWorldCodeGear の記述である. HelloWorldCodeGear では key: helloWorld に put された文字列を print 出力するという単純な処理を記述している. CGM 名.getLocalDGM().put(”Keyname”, 変数データ) にて key に変数データを紐付け (put し), CodeGear に設定されている全ての key がデータを受け取った際に初めて CodeGear は処理される. HelloWorldCodeGear では String 型の helloWorld という key が Take 型で設定されている.

以下の HelloWorld プログラムを実行した際の流れを説明する. まずポート 10000 番の CodeGearManager を生成し, HelloWorldCodeGear を setup させる. この時点では必要な key(key 名: helloWorld) にデータが揃っていないので CodeGear は実行されない. cgm.getLocalDGM().put(”helloWorld”,”hello”); にて helloWorldkey に文字列”hello”を put すると, HelloWorldCodeGear に必要な DataGear が揃い, print 表示が行われる. プログラム中では key:helloWorld への put は文字列”hello”と”world”の二回が行われ, print 出力結果は hello world と表示される.

```
public class StartHelloWorld extends StartCodeGear {

    public StartHelloWorld(CodeGearManager cgm) {
        super(cgm);
    }

    public static void main(String[] args){
        CodeGearManager cgm = createCGM(10000);
        cgm.setup(new HelloWorldCodeGear());
        cgm.getLocalDGM().put("helloWorld","hello");
        cgm.getLocalDGM().put("helloWorld","world");
    }
}
```

Code 2: Christie における CGM と CG の setup

```
public class HelloWorldCodeGear extends CodeGear {
    @Take
    String helloWorld;

    @Override
    protected void run(CodeGearManager cgm) {
        System.out.print(helloWorld + "␣");
        cgm.setup(new HelloWorldCodeGear());
    }
}
```

Code 3: HelloWorldCodeGear

Christie には Topology を形成するための機能 TopologyManager が備わっている. Topology に参加するノードに対して名前を与え, 必要とあればノード間の配線を行う.

TopologyManager の Topology 形成方法として静的 Topology と動的 Topology がある. 静的 Topology はプログラマが任意の形の Topology とノードの配線を dot ファイルに記述し, TopologyManager に参照させることで自由な形の Topology が形成できる. 現時点では静的 Topology での Topology 形成は dot ファイルに記述した参加ノード数に実際に参加するノードの数が達していない場

合, 動作しないという制約が存在している. 動的 Topology は参加を表明したノードに対し, 自動的にノード同士の配線を行う. 例えば Tree を構成する場合, 参加したノードから順番に root から近い役割を与える.

謝辞 A4 横型に対するガイドを基に，本稿を作成した．クラスファイルの作成においては，京都大学の中島　浩氏にさまざまなご教示を頂き，さらに BiBTEX 関連ファイルの利用についても快諾頂いたことを深謝する．また，A4 横型に対するガイドを作成された当時の編集委員会の担当者に深謝する．

参考文献

[1] 奥村晴彦：改訂第 5 版 LATEX 2ε 美文書作成入門，技術評論社 (2010).
[2] Goossens, M., Mittelbach, F. and Samarin, A.: *The LaTeX Companion*, Addison Wesley, Reading, Massachusetts (1993).
[3] 木下是雄：理科系の作文技術，中公新書 (1981).
[4] Strunk W. J. and White E.B.: *The Elements of Style, Forth Edition*, Longman (2000).
[5] Blake G. and Bly R.W.: *The Elements of Technical Writing*, Longman (1993).
[6] Higham N.J.: *Handbook of Writing for the Mathematical Sciences*, SIAM (1998).
[7] 情報処理学会論文誌ジャーナル編集委員会：投稿者マニュアル (online)，入手先 ⟨http://www.ipsj.or.jp/journal/submit/manual/j_manual.html⟩ (2007.04.05).
[8] 情報処理学会論文誌ジャーナル編集委員会：べからず集 (online)，入手先 ⟨http://www.ipsj.or.jp/journal/manual/bekarazu.html⟩ (2011.09.15).