令和元年度 卒業論文

CbCによるxv6のFileSystemの書き換え

琉球大学工学部情報工学科

165723C 坂本昂弘

指導教員 河野真治

目 次

図 目 次

ソースコード目次

第1章　xv6 の OS の信頼性保証

第2章　Continuation based C

2.1　Continuation based Cの概要

Continuation based C [1] (以下 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当研究室で開発されたプログラミング言語である。図 2.1 は CodeGear 間の継続する際の処理の流れを示している。

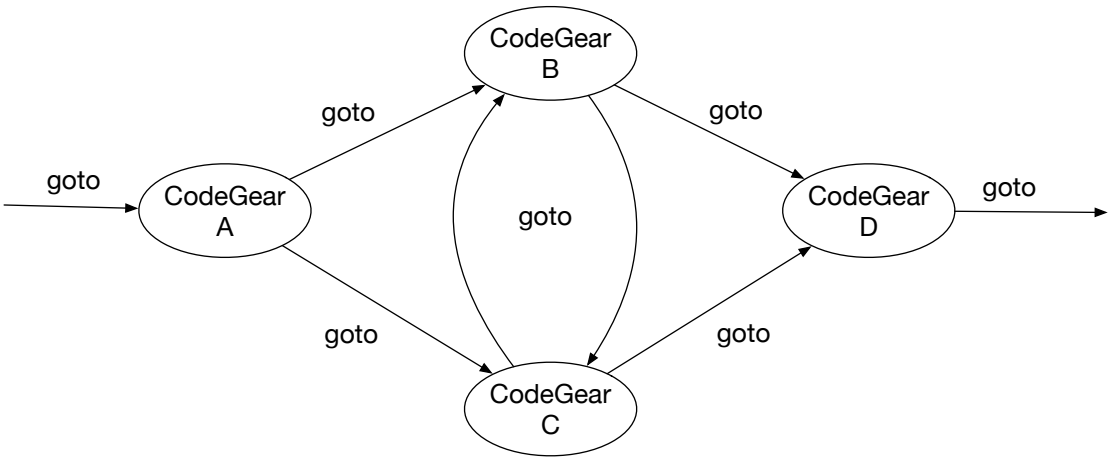

図 2.1: CodeGear 間の継続

現在 CbC は C コンパイラである GCC[2] [3] 及び LLVM[4] [5] をバックエンドとした clang 上で実装されている。本研究では、このプログラミング言語を用いて xv6 の Filesystem を書き換える。

2.2　CodeGear

CodeGear は CbC における基本的な処理単位である。以下のソースコード 2.1 は CodeGear の継続の例である。

ソースコード 2.1: CodeGear の継続の例

```
1 __code cg0(Integer a, Integer b){
2  int a_v = a->value;
3  int b_v = b->value;
4  Integer c = {a_v + b_v};
5   goto cg1(c);
6 }
7 __code cg1(Integer c){
8   goto cg2(c);
9 }
```

CodeGear は__code CodeGear 名 (引数) の形で記述される。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。ソースコード 2.1 の 5 行目の goto cg1(c); や 8 行目の goto cg2(c); などがこれにあたる。図 2.2 はソースコード 2.1 の状態遷移を表している。

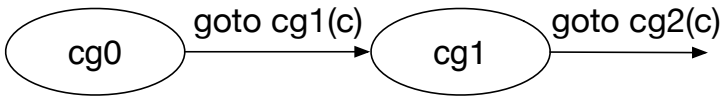

図 2.2: ソースコード 2.1 が表している CodeGear の状態遷移

また CbC における CodeGear 間の継続にはスタックが使用されず、呼び出し元の環境などを持たない為軽量継続と呼ぶ。この CbC における CodeGear 間の継続にスタックが使用されない性質は信頼性の高い OS の開発に適している。

2.3 DataGear

DataGear は CbC におけるデータの基本的な単位である。CodeGear は Input DataGear、Output DataGear を引数に持ち、図2.3で示したように遷移する際に任意の Input DataGear を参照し、Output DataGear を書き出す。

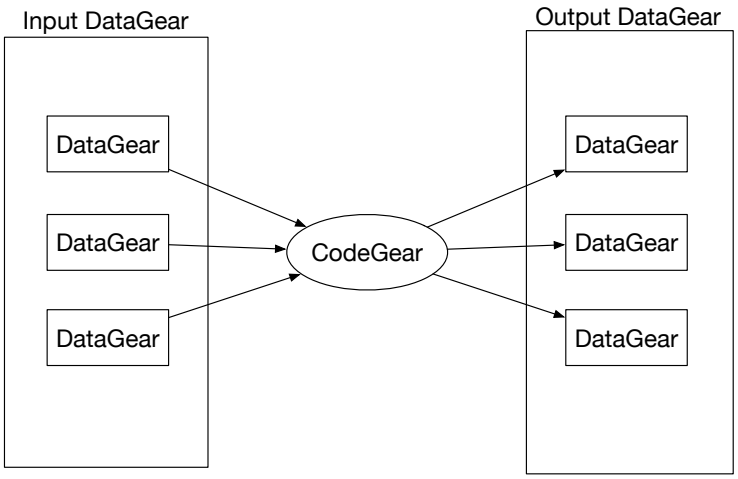

図 2.3: CodeGear と DataGear

第3章　GearsOS

3.1　GearsOS の概要

Gears OS [6] は CbC によって記述されており、CodeGear と DataGear の単位を用いて開発されている OS である。Gears OS は 一連の実行が行われる際に使用される CodeGear と DataGear を全て持つ Context と呼ばれるものを持っている。Gears OS は CodeGear 間の継続などの際、常に context を持ち歩いており CodeGear と DataGear の参照が必要になる場合、この Context を通して参照される。

3.2　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。 ノーマルレベルで見た際の CodeGear、DataGear および context の関係を以下の図 3.1 に簡潔に示す。

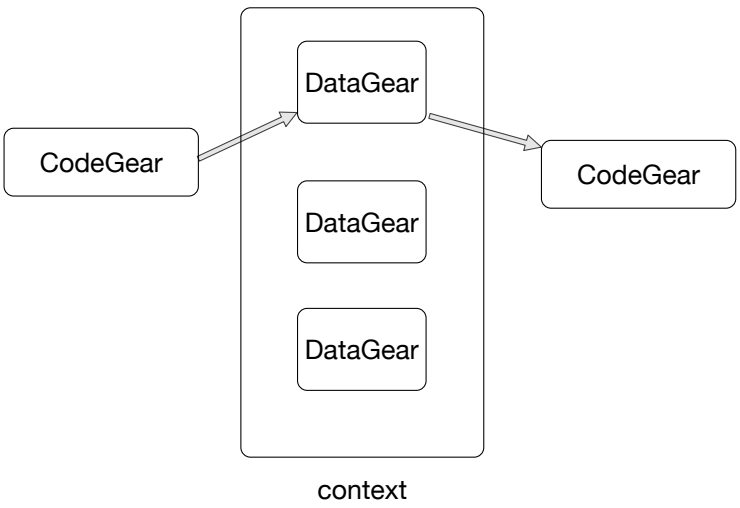

図 3.1: CodeGear、DataGear、contxt の関係図

3.3 Inetrface

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。図 3.2 は Stack の Interface とその実装を表したものである。

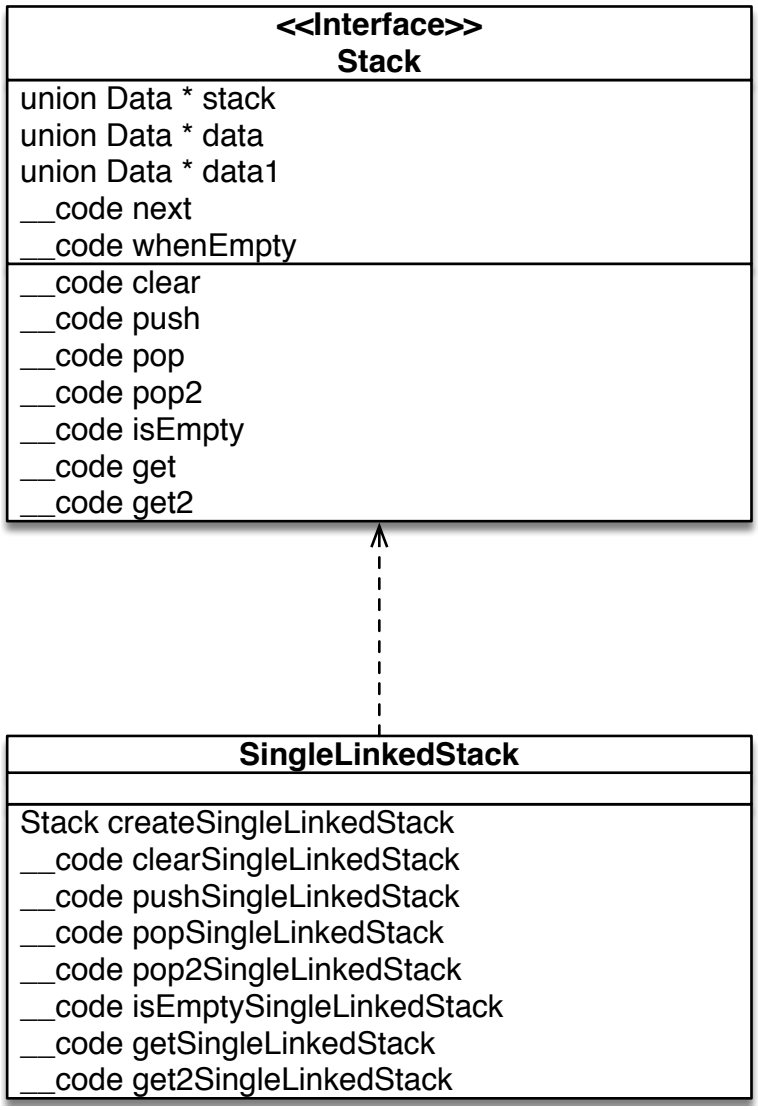

図 3.2: Stack の Interface とその実装

第4章　xv6

4.1　xv6 の概要

xv6 [7] 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。本研究で使われているのは ARM[8] 上で動作する Raspberry Pi 用に改良された xv6 を使用する。

4.2　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、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 FileSystem も存在する。 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。

4.3　xv6 の FileSystem

.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 FileSystem を使用している。xv6 の FileSystem は図 4.1 のように 7 つの階層によって構成されている。

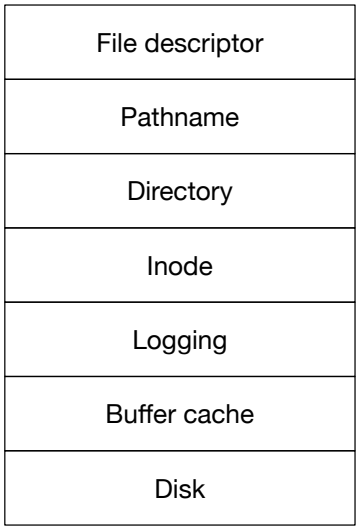

図 4.1: xv6 の FileSystem 構造

4.4 FilleSystem の API

FileSystem について記述している fs.c ではファイルを操作、管理する際に様々な関数がプロセスやデバイスなどから呼び出され使用されている。fs.c に存在している関数とその挙動に関して具体的に以下に示す。

- readsb
  ブロックのファイルサイズやデータブロックの数、inode の数、log 中のブロック数などが格納されている super block を読み込む。
  log.c で呼び出されて使用している。

- iinit
  main.c で呼び出されて使用している。

- ialloc
  デバイスで指定されたタイプを新しい inode に割り当てる。
  mkfs.c で呼び出されて使用している。

- iupdate
  変更されたメモリ内の inode をディスクにコピーする。
  fs.c で呼び出されて使用している。

- idup
  fs.c で呼び出されて使用している。

- ilock
  指定した inode をロックする。またその際に必要であるならば、ディスクから inode を読み込む。
  fs.c と exec.c で呼び出されて使用している。

- iunlock
  指定された inode のロックを解除する。
  fs.c と exec.c で呼び出されて使用している。

- iput
  メモリ内の inode への参照を削除する。
  fs.c で呼び出されて使用している。

- iunlockput
  指定された inode のロックを解除してから iput を実行する。
  fs.c と exec.c で呼び出されて使用している。

- stati
  inode から ファイルに関する統計情報を複製する。
  fs.c で呼び出されて使用している。

- readi
  inode からデータを読み込む。
  fs.c と exec.c と vm.c で呼び出されて使用している。

- writei
  inode へデータを書き込む。
  fs.c で呼び出されて使用している。

- namecmp
  fs.c で呼び出されて使用している。

- dirlookup
  fs.c で呼び出されて使用している。

- dirlink
  fs.c で呼び出されて使用している。

- namei
  fs.c と exec.c で呼び出されて使用している。

- nameiparent
  fs.c で呼び出されて使用している。

第5章　CbCによるFileSystemの書き換え

5.1　書き換え方針

5.2　FileSystem の Interface の定義 (fs.dg)

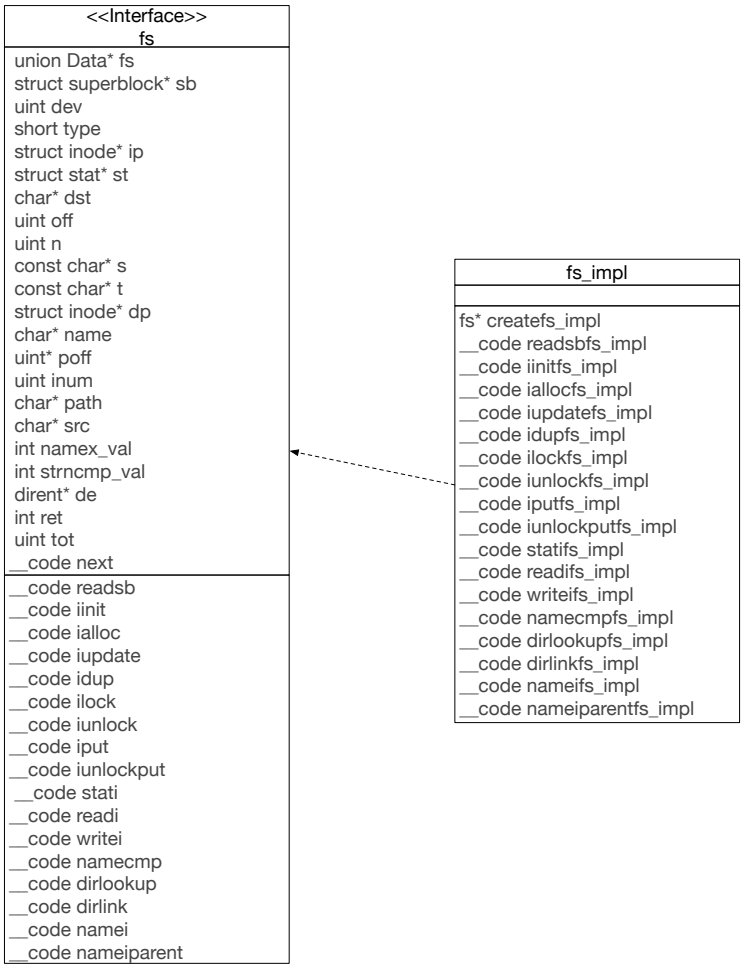

図 5.1: xv6 FileSystem の Interface と実装

FileSystem の Interface を記述したコードをソースコード 5.1 に示す。

ソースコード 5.1: FileSystem の Interface

```
typedef struct fs<Type,Impl> {
    union Data* fs;
    struct superblock* sb;
    uint dev;
    short type;
    struct inode* ip;
    struct stat* st;
    char* dst;
    uint off;
    uint n;
    const char* s;
    const char* t;
    struct inode* dp;
    char* name;
    uint* poff;
    uint inum;
    char* path;
    char* src;
    int namex_val;
    int strncmp_val;
    dirent* de;
    int ret;
    uint tot;
    __code readsb(Impl* fs, uint dev, struct superblock* sb, __code next(...));
    __code iinit(Impl* fs, __code next(...));
    __code ialloc(Impl* fs, uint dev, short type, __code next(...));
    __code iupdate(Impl* fs, struct inode* ip, __code next(...));
    __code idup(Impl* fs, struct inode* ip, __code next(...));
    __code ilock(Impl* fs, struct inode* ip, __code next(...));
    __code iunlock(Impl* fs, struct inode* ip, __code next(...));
    __code iput(Impl* fs, struct inode* ip, __code next(...));
    __code iunlockput(Impl* fs, struct inode* ip, __code next(...));
    __code stati(Impl* fs , struct inode* ip, struct stat* st, __code next(...));
    __code readi(Impl* fs, struct inode* ip, char* dst, uint off, uint tot, uint n,
        __code next(int ret, ...));
    __code writei(Impl* fs, struct inode* ip, char* src, uint off, uint tot, uint n,
        __code next(int ret, ...));
    __code namecmp(Impl* fs, const char* s, const char* t, __code next(int
        strncmp_val, ...));
    __code dirlookup(Impl* fs, struct inode* dp, char* name, uint off, uint* poff,
        dirent* de, __code next(int ret, ...));
    __code dirlink(struct fs_impl* fs, struct inode* ip, struct dirent* de, struct
        inode* dp, char* name, uint off, uint inum, __code next(...));
    __code namei(Impl* fs, char* path, __code next(int namex_val, ...));
    __code nameiparent(Impl* fs, char* path, char* name, __code next(int namex_val,
        ...));
    __code next(...);
} fs;
```

5.3 FileSystem の Interface の実装 (fs_impl.cbc)

ソースコード 5.2: FileSystem の Interface の実装

```
#interface "Err.h"
#interface "fs.dg"

fs* createfs_impl(struct Context* cbc_context) {
    struct fs* fs = new fs();
    struct fs_impl* fs_impl = new fs_impl();
```

```
7     fs->fs = (union Data*)fs_impl;
8     fs_impl->fs_impl = NULL;
9     fs_impl->sb = NULL;
10    fs_impl->ret = 0;
11    fs_impl->dev = 0;
12    fs_impl->type = 0;
13    fs_impl->bp = NULL;
14    fs_impl->dip = NULL;
15    fs_impl->inum = 0;
16    fs_impl->dp = NULL;
17    fs_impl->name = NULL;
18    fs_impl->off = 0;
19    fs_impl->poff = NULL;
20    fs_impl->de = NULL;
21    fs_impl->tot = 0;
22    fs_impl->m = 0;
23    fs_impl->dst = NULL;
24    fs_impl->n = 0;
25    fs_impl->src = NULL;
26    fs_impl->allocinode = C_allocinode;
27    fs_impl->allocinode_loop = C_allocinode_loop;
28    fs_impl->allocinode_loopcheck = C_allocinode_loopcheck;
29    fs_impl->allocinode_noloop = C_allocinode_noloop;
30    fs_impl->lockinode1 = C_lockinode1;
31    fs_impl->lockinode2 = C_lockinode2;
32    fs_impl->lockinode_sleepcheck = C_lockinode_sleepcheck;
33    fs_impl->iput_check = C_iput_check;
34    fs_impl->iput_inode_nolink = C_iput_inode_nolink;
35    fs_impl->readi_check_diskinode = C_readi_check_diskinode;
36    fs_impl->readi_loopcheck = C_readi_loopcheck;
37    fs_impl->readi_loop = C_readi_loop;
38    fs_impl->readi_noloop = C_readi_noloop;
39    fs_impl->writei_check_diskinode = C_writei_check_diskinode;
40    fs_impl->writei_loopcheck = C_writei_loopcheck;
41    fs_impl->writei_loop = C_writei_loop;
42    fs_impl->writei_noloop = C_writei_noloop;
43    fs_impl->dirlookup_loopcheck = C_dirlookup_loopcheck;
44    fs_impl->dirlookup_loop = C_dirlookup_loop;
45    fs_impl->dirlookup_noloop = C_dirlookup_noloop;
46    fs_impl->dirlink_namecheck = C_dirlink_namecheck;
47    fs_impl->dirlink_loopcheck = C_dirlink_loopcheck;
48    fs_impl->dirlink_loop = C_dirlink_loop;
49    fs_impl->dirlink_noloop = C_dirlink_noloop;
50    fs->readsb = C_readsbfs_impl;
51    fs->iinit = C_iinitfs_impl;
52    fs->ialloc = C_iallocfs_impl;
53    fs->iupdate = C_iupdatefs_impl;
54    fs->idup = C_idupfs_impl;
55    fs->ilock = C_ilockfs_impl;
56    fs->iunlock = C_iunlockfs_impl;
57    fs->iput = C_iputfs_impl;
58    fs->iunlockput = C_iunlockputfs_impl;
59    fs->stati = C_statifs_impl;
60    fs->readi = C_readifs_impl;
61    fs->writei = C_writeifs_impl;
62    fs->namecmp = C_namecmpfs_impl;
63    fs->dirlookup = C_dirlookupfs_impl;
64    fs->dirlink = C_dirlinkfs_impl;
65    fs->namei = C_nameifs_impl;
66    fs->nameiparent = C_nameiparentfs_impl;
67    return fs;
68 }
69
70 typedef struct superblock superblock;
71 __code readsbfs_impl(struct fs_impl* fs, uint dev, struct superblock* sb, __code
      next(...)) { //:skip
```

```
72
73      struct buf* bp;
74
75      bp = bread(dev, 1);
76      memmove(sb, bp->data, sizeof(*sb));
77      brelse(bp);
78
79      goto next(...);
80  }
81
82  __code iinitfs_impl(struct fs_impl* fs, __code next(...)) {
83
84       initlock(&icache.lock, "icache");
85
86      goto next(...);
87  }
88
89  __code iallocfs_impl(struct fs_impl* fs, uint dev, short type, __code next(...)) {
90      goto allocinode(fs, dev, sb, next(...));
91  }
92
93  __code iupdatefs_impl(struct fs_impl* fs, struct inode* ip, __code next(...)) {
94
95      struct buf *bp;
96      struct dinode *dip;
97
98      bp = bread(ip->dev, IBLOCK(ip->inum));
99
100     dip = (struct dinode*) bp->data + ip->inum % IPB;
101     dip->type = ip->type;
102     dip->major = ip->major;
103     dip->minor = ip->minor;
104     dip->nlink = ip->nlink;
105     dip->size = ip->size;
106
107     memmove(dip->addrs, ip->addrs, sizeof(ip->addrs));
108     log_write(bp);
109     brelse(bp);
110
111     goto next(...);
112 }
113
114 __code idupfs_impl(struct fs_impl* fs, struct inode* ip, __code next(...)) {
115
116     acquire(&icache.lock);
117     ip->ref++;
118     release(&icache.lock);
119
120     goto next(ip, ...);
121 }
122
123 __code ilockfs_impl(struct fs_impl* fs, struct inode* ip, __code next(...)) {
124
125     goto lockinode1(fs, ip, bp, dip, next(...));
126 }
127
128 __code iunlockfs_impl(struct fs_impl* fs, struct inode* ip, __code next(...)) {
129
130     if (ip == 0 || !(ip->flags & I_BUSY) || ip->ref < 1) {
131        char* msg = "iunlock";
132        struct Err* err = createKernelError(&proc->cbc_context);
133        Gearef(cbc_context, Err)->msg = msg;
134        goto meta(cbc_context, err->panic);
135     }
136
137     acquire(&icache.lock);
```

```
138     ip->flags &= ~I_BUSY;
139     wakeup(ip);
140     release(&icache.lock);
141 
142     goto next(...);
143 }
144 
145 __code iputfs_impl(struct fs_impl* fs, struct inode* ip, __code next(...)) {
146     if (next == C_iputfs_impl) {
147         next = fs->next2;
148     }
149     goto iput_check(fs, ip, next(...));
150 }
151 
152 __code iunlockputfs_impl(struct fs_impl* fs, struct inode* ip, __code next(...)) {
153     fs->next2 = next;
154     goto iunlockfs_impl(ip, fs->iput, ...);
155 }
156 
157 typedef struct stat stat;
158 __code statifs_impl(struct fs_impl* fs , struct inode* ip, struct stat* st, __code
        next(...)) { //:skip
159     st->dev = ip->dev;
160     st->ino = ip->inum;
161     st->type = ip->type;
162     st->nlink = ip->nlink;
163     st->size = ip->size;
164     goto next(...);
165 }
166 
167 __code readifs_impl(struct fs_impl* fs, struct inode* ip, char* dst, uint off, uint
        tot, uint n, __code next(int ret, ...)) {
168     if (ip->type == T_DEV) {
169         goto readi_check_diskinode(fs, ip, dst, n, next(...));
170     }
171 
172     if (off > ip->size || off + n < off) {
173         ret = -1;
174         goto next(ret, ...);
175     }
176 
177     if (off + n > ip->size) {
178         n = ip->size - off;
179     }
180     Gearef(cbc_context, fs)->tot = 0;
181     goto readi_loopcheck(fs, tot, m, dst, off, n, next(...));
182 }
183 
184 __code writeifs_impl(struct fs_impl* fs, struct inode* ip, char* src, uint off, uint
         tot, uint n, __code next(int ret, ...)) {
185     if (ip->type == T_DEV) {
186         goto writei_check_diskinode(fs, ip, src, n, next(...));
187     }
188 
189     if (off > ip->size || off + n < off) {
190         ret = -1;
191         goto next(ret, ...);
192     }
193 
194     if (off + n > MAXFILE * BSIZE) {
195         ret = -1;
196         goto next(ret, ...);
197     }
198     Gearef(cbc_context, fs)->tot = 0;
199     goto writei_loopcheck(fs, tot, m, src, off, n, next(...));
200 }
```

```
201
202 __code namecmpfs_impl(struct fs_impl* fs, const char* s, const char* t, __code next(
        int strncmp_val, ...)) {
203     strncmp_val = strncmp(s, t, DIRSIZ);
204     goto next(strncmp_val, ...);
205 }
206
207 __code dirlookupfs_impl(struct fs_impl* fs, struct inode* dp, char* name, uint off,
        uint* poff, dirent* de, __code next(...)) { //:skip
208     if (dp->type != T_DIR) {
209        char* msg = "dirlookup␣not␣DIR";
210        struct Err* err = createKernelError(&proc->cbc_context);
211        Gearef(cbc_context, Err)->msg = msg;
212        goto meta(cbc_context, err->panic);
213     }
214     Gearef(cbc_context, fs)->off = 0;
215     goto dirlookup_loopcheck(fs, dp, name, off, poff, de, next(...));
216 }
217
218 __code dirlinkfs_impl(struct fs_impl* fs, struct inode* ip, struct dirent* de,
        struct inode* dp, char* name, uint off, uint inum, __code next(...)) { //:skip
219
220     if ((ip = dirlookup(dp, name, 0)) != 0) {
221         goto dirlink_namecheck(fs, ip, next(...));
222     }
223     Gearef(cbc_context, fs)->off = 0;
224     goto dirlink_loopcheck(fs, de, dp, off, next(...));
225 }
226
227 __code nameifs_impl(struct fs_impl* fs, char* path, __code next(int namex_val, ...))
        {
228     char name[DIRSIZ];
229     namex_val = namex(path, 0, name);
230     goto next(namex_val, ...);
231 }
232
233 __code nameiparentfs_impl(struct fs_impl* fs, char* path, char* name, __code next(
        int namex_val, ...)) {
234
235     namex_val = namex(path, 1, name);
236     goto next(namex_val, ...);
237
238 }
```

第6章　まとめと今後の課題

今回の研究では xv6 の FileSystem 部分について CbC を用いて書き換えを行った。しかし、 xv6 は Gears OS を開発する前段階として開発しているので今後は書き換えた xv6 を Gears OS に適応した形に改良していく必要がある。xv6 の FileSystem 部分書き換え後 make し build することはできたが、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、動かなかった場合修正を行い 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] GNU Compiler Collection (GCC) Internals. *http://gcc.gnu.org/onlinedocs/gccint/*.

[3] 大城信康, 河野真治. Continuation based c の gcc4.6 上の実装について. 第 53 回プログラミング・シンポジウム予稿集, 第 2012 巻, pp. 69–78, jan 2012.

[4] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[5] Tokumori Kaito and Kono Shinji. Implementing continuation based language in llvm and clang. *LOLA 2015, Kyoto*, July 2015.

[6] 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.

[7] 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).

[8] ARM Architecture Reference Manual. *http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/index.html*.

[9] 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.

[10] 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.

謝辞

本研究の遂行，また本論文の作成にあたり、御多忙にも関わらず終始懇切なる御指導と御教授を賜わりました河野真治准教授に深く感謝したします。

数々の貴重な御助言と細かな御配慮を戴いた並列信頼研究室の hoge 氏に深く感謝致します。

また一年間共に研究を行い、暖かな気遣いと励ましをもって支えてくれた並列信頼研究室の hoge 君、hoge 君、hoge さんに感謝致します。

最後に、有意義な時間を共に過ごした情報工学科の学友、並びに物心両面で支えてくれた両親に深く感謝致します。

2020 年 2 月
坂本昂弘