Loadable Kernel Module による システムコール拡張を容易にする領域特化言語

情報・通信工学科 中野研究室 1211084 徐振宇

1 背景

Linux カーネルに新たな機能を追加するためには，システムコールの追加や拡張が必要不可欠である．システムコールを拡張 (追加) する方法には，以下の 2 つがある．

1 つめは，カーネルのソースコードに拡張内容を記述することである．そして，カーネル全体をビルドし，再起動することにより，カーネルに新たに追加した機能を初めて利用できるようになる．この方法は，一般的に利用されているが，ビルドの時間が長い点と，再起動が必要となるという問題点がある．

2 つめは, LKM を用いることである．Loadable Kernel Module (LKM) は，カーネルの機能を拡張するためのオブジェクトファイルである．その特徴は，実行中のカーネルに動的にロードさせることである．このため，OS を再起動させる必要がない．ビルドの対象は LKM の部分だけなので，カーネル全体をビルドするより短い時間で済む．

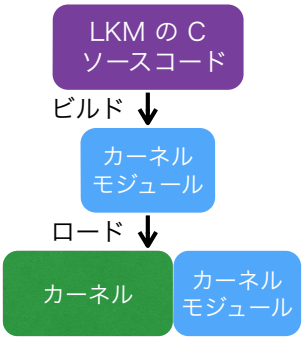

図 1 LKM を用いたカーネル拡張の流れ

しかしながら，LKM を用いてカーネルを拡張することは容易ではない．何故なら，LKM のコードは定型的な処理が多く，繁雑なフォーマットに従わなければならないからである．

2 LKM の実装例とその問題点

図 2 のプログラムは，open システムコールが呼び出される前に，実引数を先に表示するように拡張するための LKM の実装例である．

具体的な拡張内容は，拡張機能のコードと拡張機能が行うタイミングの部分だけであるにもかかわらず，LKM では定型的なコードが多いため，プログラムが繁雑になり，見通しが悪い．

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <asm/unistd.h>

Module_LICENSE("GPL");
extern void* sys_call_table[];

asmlinkage static int (*original_open)
     (const char* pathname, int flags);

asmlinkage static int my_open
      (const char* pathname, int flags){
 printk(KERN_INFO "my_open(\"%s\",%d)\n",
 pathname, flags);
 return original_open(pathname, flags);
}

static int on_init(void) {
 printk(KERN_INFO "on_init\n");
 original_open = sys_call_table[__NR_open];
 sys_call_table[__NR_open] = my_open;
 rteurn 0;
}

static void on_exit(void) {
 printk(KERN_INFO "on_exit\n");
 sys_call_table[__NR_open] = original_open;
}

module_init(on_init);
module_exit(on_exit);
```

図 2 LKM の実装例

3 目的と方針

本研究では，LKM を用いたシステムコールの拡張を支援し，ユーザの負担を軽減するためシステムの設計と実装を目指す．

基本方針として，領域特化言語 (DSL) を提供し，ユーザが記述した領域特化言語のコードを LKM のコードに自動的に変換する．ここで DSL とは，特定の領域に特化した機能を提供するプログラミング言語である．

DSL を用いた処理の流れは，図 3 に示すように，記述した DSL のプログラムを処理系に渡して，LKM のコードが生成されるようにする．生成されたコードを図 1 と同様にコンパイルし，カーネルモジュールとする．

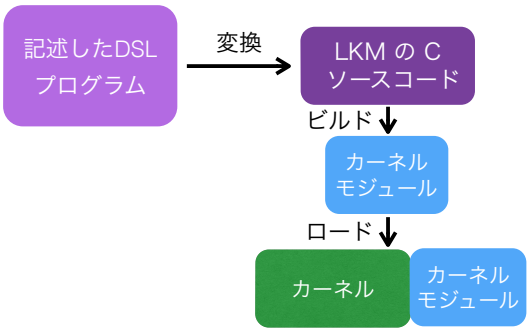

図 3　DSL を用いた LKM 開発の流れ

4　提案 DSL による記述例

システムコールの拡張の代表的なパターンには，指定したコードを元の動作の前に行う，元の動作の後に行う，指定したコードで元の動作を置き換える，の 3 種類が考えられる．先の LKM の例を，提案 DSL を用いて記述すると，図 4 のようになる．

```
before(OPEN, Func);
int Func(const char* pathname, int flags){
 printk(KERN_INFO "my_open(\"%s\",%d)\n",
 pathname, flags);
 return 0;
}
```

図 4　提案 DSL による記述例

まず，拡張機能の行うタイミングが元の動作の前なので，`before` を用いる．`before` の第一引数は，拡張したいシステムコールを表す定数．第二引数は，元の動作の前に挿入される動作を行う関数へのポインタである．最後に，拡張機能のコードを関数として記述する．

5　現状と今後

これまでは，LKM について調査を行い，システムコールを拡張する LKM を実装した．更に，LKM のコードに現れるパターンを調査した．

今後は，まず DSL の仕様を決定し，処理系の実装を行う．また，既存の LKM コードを提案 DSL で書き直した場合の記述量を比較することにより，評価を行う．

参考文献

[1] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., Irwin, J. (1997). Aspect-oriented programming

[2] Peter Jay Salzman, Michael Burian, Ori Pomerantz. (2007). The Linux Kernel Module Programming Guide