Hotswapping Linux kernel modules
Yueh-Freng Lee, Ruei-Chuan Chang
National Chaio Tung University
Outline

- Introduction
- Hotswaping and Linux modules
- Design and implementation
- Evaluation
- Conclusion
Introduction

- Modern operating system kernels are normally extensible.
  - Linux kernel modules (LKM).

- However, upgrade kernel extension involves
  - shutting down the applications.
  - Removing old kernel extension.
  - Installing the new one.
  - Restarting applications.

- Old kernel extension’s state is lost.
Introduction

- Dynamic update facility can solve these problems.
  - Can replace kernel extensions without affecting the dependent applications.

- Dynamic update can be implemented in two ways.
  - Indirect swapping.
  - Direct swapping.
Introduction

Fig. 1. (a) Indirect swap and (b) direct swap.
Hotswapping and Linux modules

- General requirements of hotswapping
  - Component state transfer.
  - Component boundaries.
  - Mutually consistent states.
  - External references.
Design and implementation

Component state transfer.

- **Static address sections**
  - `static_new`
    - Variable placed in the section, then its content is overwritten by new module.
  - `static_old`
    - Content is not change.
Design and implementation
Generally, programmers can treat global variables as the module state.

Transfer global variables by prefixing their names with _old_.

- \( i \Rightarrow \_old\_i \)
Design and implementation

```c
struct struct_a
{
    int i;
    int j;
}
static struct struct_a a;
```

```c
struct old_struct_a
{
    int i;
    int j;
}
struct struct_a
{
    int x;
    int i;
    int j;
}
static struct struct_a a;
extern struct old_struct_a _old_a;
```

```c
static void replace_handler
{
    a.x = 100;
    a.i = _old_a.i;
    a.j = _old_a.j;
}
```

```c
```
Design and implementation

- **Dynamic resolution and relocation**
  - Can links a symbol statically many times.

- **Replace_handle structure**
  - Hold the addresses of the backup object file.
  - And important sections.

External references.
Design and implementation

- Example

```
A
B'
B
```
Design and implementation

Mutually consistent states.

- Preserve mutual consistency using stack trace.
  - A module is in the quiescent state when
    - none of its functions are in any process stack.
  - Stack tracing code examines each frame of each process’s kernel stack.
Evaluation

Apache server

File system driver (vfat)

partition
struct inode_operations vfat_dir_inode_operations
__attribute__((section(".static_new")))) = {
    .create = vfat_create,
    .lookup = vfat_lookup,
    .unlink = vfat_unlink,
    .mkdir = vfat_mkdir,
    .rmdir = vfat_rmdir,
    .rename = vfat_rename,
    .setattr = fat_notify_change,
};

static struct file_system_type vfat_fs_type
__attribute__((section(".static_old")))) = {
    .owner = THIS_MODULE,
    .name = "vfat",
    .get_sb = vfat_get_sb,
    .kill_sb = kill_block_super,
    .fs_flags = FS_REQUIRES_DEV,
};

static int replace_module(void)
{
    vfat_fs_type.name="vfat";
    vfat_fs_type.get_sb=vfat_get_sb;
    vfat_fs_type.kill_sb=kill_block_super;
    return 0;
}
## Evaluation

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Loading times of the vfat module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original system (μs)</strong></td>
<td><strong>Hotswap system (μs)</strong></td>
</tr>
<tr>
<td>2119</td>
<td>2192</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Loading and hotswapping times of the vfat module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module loading</strong></td>
<td><strong>Hotswap system (μs)</strong></td>
</tr>
<tr>
<td></td>
<td>2192</td>
</tr>
<tr>
<td><strong>Hotswapping</strong></td>
<td>3042</td>
</tr>
<tr>
<td><strong>Ratio</strong></td>
<td>138.77%</td>
</tr>
</tbody>
</table>
Evaluation

Table 3
The most time-consuming steps on hotswapping the vfat module

<table>
<thead>
<tr>
<th></th>
<th>Hotswap system (μs)</th>
<th>Percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>1476</td>
<td>48.52</td>
</tr>
<tr>
<td>Quiescence detection</td>
<td>933</td>
<td>30.67</td>
</tr>
<tr>
<td>Others</td>
<td>633</td>
<td>20.81</td>
</tr>
<tr>
<td>Total</td>
<td>3042</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4
Module and symbol status of the experimental system

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Module number</td>
<td>36</td>
</tr>
<tr>
<td>Module symbols</td>
<td>584</td>
</tr>
<tr>
<td>Module GPL symbols</td>
<td>1</td>
</tr>
<tr>
<td>Kernel symbols</td>
<td>2166</td>
</tr>
<tr>
<td>Kernel GPL symbols</td>
<td>240</td>
</tr>
</tbody>
</table>
# Evaluation

## Table 5
Execution times of the init_module and replace_module functions

<table>
<thead>
<tr>
<th></th>
<th>Module loading (µs)</th>
<th>Hotswapping (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_module or replace_module function</td>
<td>6.637</td>
<td>0.583</td>
</tr>
<tr>
<td>Total</td>
<td>2192</td>
<td>3042</td>
</tr>
<tr>
<td>Ratio</td>
<td>0.30%</td>
<td>0.02%</td>
</tr>
</tbody>
</table>

## Table 6
Normal resolution and dynamic resolution times of the vfat module

<table>
<thead>
<tr>
<th></th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic resolution</td>
<td>517</td>
</tr>
<tr>
<td>Normal resolution</td>
<td>1476</td>
</tr>
<tr>
<td>Ratio</td>
<td>35.03%</td>
</tr>
</tbody>
</table>
Conclusion

- Proposes a Linux-based hotswap module system.