SUNBEAM

Sunbeam Institute of Information Technology
Pune and Karad

Embedded Linux Device Driver

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Kernel module internals

- Kernel module is represented by struct module in the Linux kernel.
    - Variable of struct module is created & initialized in .mod.c file, with name __this_module. This can be accessed in the module source code using macro THIS_MODULE.

    - After module is loaded kernel keep this variable in a kernel linked list. All kernel modules info can be accessed via /sys/module or /proc/modules or "lsmod" command.

    - struct module members:
        - enum module_state state; — COMING → 1, LIVE → 0, GOING → 2
        - struct list_head list; — | *next | *prev |
        - char name[MODULE_NAME_LEN]; ← name of module
        - int (*init)(void); } function pointers – stores addresses of entry point functions
        - void (*exit)(void); } init_module & cleanup_module resp
        - void *module_init; – info used to initialize the module
        - void *module_core; – info used to control execution of module
        - atomic_t refcnt; – count of modules which are using this module

Makefile - Assignment

* Normal Assignment (early assignment)
  obj-m := hello.o ← value is assigned at the time of assignment only

* Recursive Assignment (late assignment)
  obj-m = hello.o ← value will be assigned at the use of variable

* Conditional Assignment
  obj-m ?= hello.o ← value will be assigned if variable is empty.

* Additive Assignment
  obj-m += hello.o ← more values will be added into variable.

```
CSRC= main.c
CSRC += add.c sub.c
```

[ ] ← do not compile → obj-n
[*] ← compile as static component → obj-y
[m] ← compile as dynamic component → obj-m

Loading and Unloading of Kernel Module

insmod
↓
init_module()
↓
sys_init_module()
↓
check for the permissions of
user & process
↓
load_module()
↓
struct module mod = {...}
↓
return mod
↓
add_list(module_list, mod)
↓
init_module()
↓
State is changed to LIVE

rmmod
↓
delete_module()
↓
sys_delete_module()
↓
check for permissions
↓
check for module is in use or not
↓
module state is changed to going
↓
cleanup_module( )
↓
free_module( )
↓
delete all references of module
↓
release space module

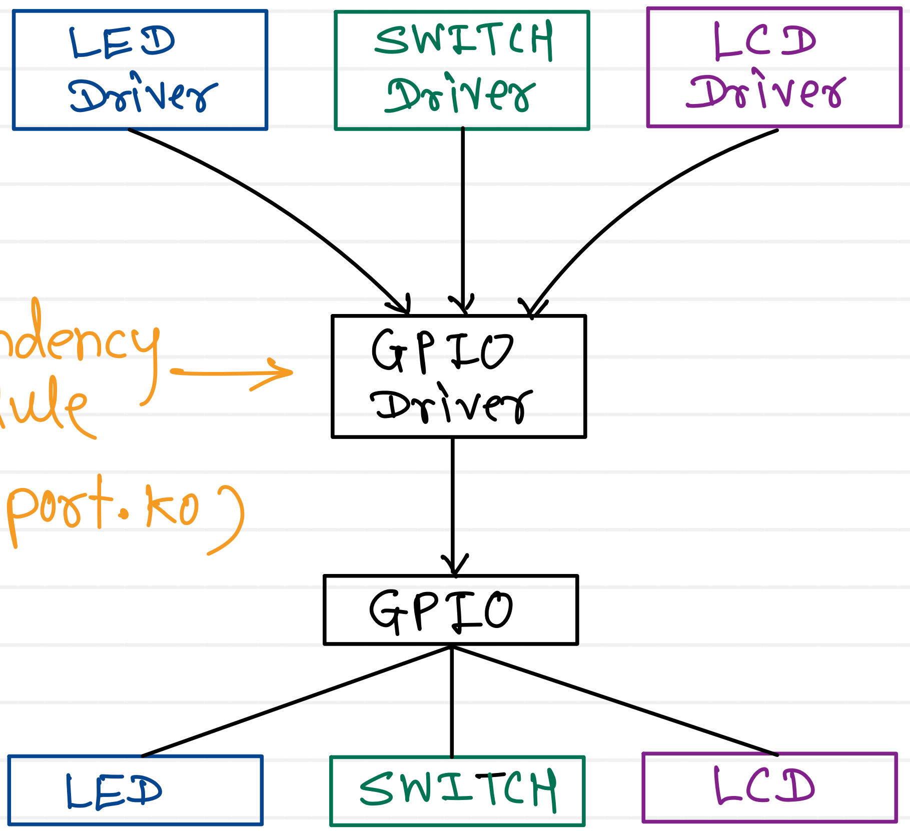
Module stacking

- One module is calling functions/variables from another module.

will use exported symbols of dependency module ← dependent module (import.ko) →

will export some ← dependency module (export.ko) →

$\underbrace{\text{functions \& variables,}}_{\text{symbols}}$



Module stacking

- A kernel module can export symbols using macros:
    - EXPORT_SYMBOL(symbol)
        - it exports the given symbol, so that it can be used by any other kernel module.
    - EXPORT_SYMBOL_GPL(symbol)
        - Similar to EXPORT_SYMBOL() for exporting symbols with "_gpl" tag.
        - These symbols can only be used by GPL licensed modules.

- Compiling multiple modules using Makefile.

```
obj-m := export.o import.o
```

- Loading & unloading kernel modules:
    - terminal> sudo insmod export.ko
    - terminal> sudo insmod import.ko
    - terminal> sudo rmmod import.ko
    - terminal> sudo rmmod export.ko

$$\begin{tikzcd}
\text{export.c} \arrow[r] \arrow[d] & \text{export.o} \arrow[dr] & & & \\
\text{export.mod.c} \arrow[r] & \text{export.mod.o} \arrow[r] & \text{export.ko} & \text{export.mod} & \text{modules.order} \\
\text{import.c} \arrow[r] \arrow[d] & \text{import.o} \arrow[dr] & & & \text{Module.symvers} \\
\text{import.mod.c} \arrow[r] & \text{import.mod.o} \arrow[r] & \text{import.ko} & \text{import.mod} &
\end{tikzcd}$$

Module stacking internals

EXPORT_SYMBOL(symbol):
* Creates a name of symbol as "static const char[]" and store it in special section "__ksymtab_strings".

* Also create a struct "kernel_symbol" variable that stores address of the variable and its name in another special section "__ksymtab".

* If Linux kernel version control is enabled, CRC is also stored in special section "__kcrctab". e.g. if a variable by name "get_rms" is exported using "EXPORT_SYMBOL", then this macro internally produces following code:

* static const char __kstrtab_get_rms[] __attribute__((section("__ksymtab_strings"))) = "get_rms";

* static const struct kernel_symbol __ksymtab_get_rms __attribute_used__ __attribute__((section("__ksymtab" "") , unused)) = { (unsigned long)&get_rms, __kstrtab_get_rms };

* During loading a module these special sections (__ksymtab) are read from .ko file and information in that is added into kernel symbol table.

Module stacking

- Module loading sequence can be automated using modprobe tool.
- modprobe tool loads modules from /lib/modules/`uname -r`.
  The module dependency is read from modules.dep file. This file is built using depmod tool.

- Steps to use modprobe:

```
terminal> sudo cp import.ko export.ko /lib/modules/`uname -r`/kernel
terminal> sudo depmod
```

- Loading module using modprobe

```
terminal> sudo modprobe import
```

- Unloading module using modprobe

```
terminal> sudo modprobe i[redacted]t -r import
terminal> sudo modprobe t -r export
```

insmod	modprobe
1. not able to resolve dependency(s) if any	1. able to resolve dependency(s) if any
2. can also insert module which is out side of the kernel module tree	2. can not insert module which is out side of the kernel module tree

- Modules can be loaded automatically while boot by making entry into /etc/modules.

Kernel module parameters

- Module parameters are used to give configuration options while loading the module.
- Some of these configurations may be modified via /sys/module entry.
- The module params are declared as static global variables in the module.

- Supported data types are: bool, invbool, charp, int, short, long, uint, ushort, ulong

- Module param syntax

```
module_param(varname, datatype, permissions);
```

- Module param array syntax

```
module_param_array(varname, datatype, ele_count, permissions);
```

- Module param can be exposed with different name

```
module_param_named(user_name, varname, datatype, permissions);
```

- Module param permissions can be given using macros:

  S_IRUSR, S_IWUSR, S_IXUSR,
  S_IRGRP, S_IWGRP, S_IXGRP,
  S_IROTH, S_IWOTH, S_IXOTH.

Kernel module parameters internals

- Internally module parameters are stored in a struct kernel_param.

  It has following members:

```
        const char *name;
        param_set_fn set;
        param_get_fn get;
        union {
                void *arg;
                const struct kparam_string *str;
                const struct kparam_array *arr;
        };
```

function pointers $\rightarrow$ addresses of default callbacks

- The kernel_param struct variables are added into "__params" section.
- The param name and type information is added into ".modinfo" section.
- The param names & params are exported from module and become part of kernel symbol table.
- Module param value can be accessed or modified using
  /sys/module/mod_name/parameters/param_name.

Kernel module parameters

- module param callback syntax

```
module_param_cb(name, &kernel_param_ops, &name, perm)
```

- This macro used to register the callback whenever the argument (parameter) got changed.

- callbacks

```
struct kernel_param_ops {
        int (*set)(const char *val, const struct kernel_param *kp);
        int (*get)(char *buffer, const struct kernel_param *kp);
        void (*free)(void *arg);
};
```

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com