SUNBEAM

Sunbeam Institute of Information Technology
Pune and Karad

Embedded Linux Device Driver

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Kernel module implementation

- Kernel modules are binary files containing code & data (like user-space applications) which are dynamically linked to the kernel at runtime.

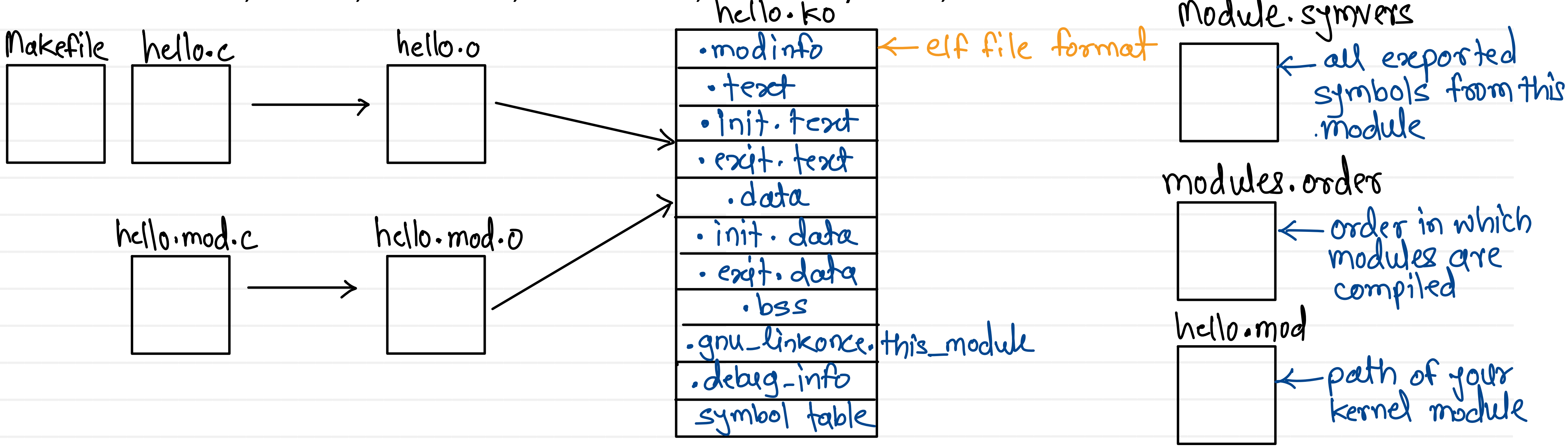
- Each kernel module have at least two entry point functions i.e. init and exit.
  - Traditionally their names as init_module() and cleanup_module().
  - Programmer may choose different names using module_init() & module_exit() macros
  - These functions are marked with __init and __exit attributes.

- Each module also have information associated with it using MODULE_XYZ() macros.
  - These macros will expands to MODULE_INFO() macros.
  - All this metadata is added into .modinfo section of .ko file, which can be inspected using modinfo command.
  - It also stores kernel version (of kernel against which module is built). This version is verified while loading it into the kernel. If version mismatch, module loading fails.

- Kernel modules can access functions exported by the kernel or other kernel module.

Kernel module compilation

- Create Makefile for compiling kernel module.
  - obj-m = hello.o
- Compile the kernel module using kernel Makefile.
  - make -C /lib/modules/`uname -r`/build M=`pwd` modules
- Generated files
  - hello.mod.c, hello.o, hello.mod, hello.mod.o, Module.symvers, modules.order



elf file format

all exported symbols from this module

order in which modules are compiled

path of your kernel module

Kernel module structure

- Kernel module must be compiled against the kernel in which it is to be loaded. For this we should have access to kernel headers and kernel build system (Makefile, ...)

- Compiled kernel modules (.ko) are sectioned binary like ELF.

- terminal> objdump -f hello.ko

- terminal> objdump -h hello.ko
  - .text, .data, .rodata, .bss
  - .init.text
  - .exit.text
  - .gnu.linkonce.this_module

- terminal> objdump -t hello.ko
  - All unresolved symbols (e.g. printk()) are resolved at the time of loading that module (i.e. insmod) from kernel symbol table.
  - This table can be viewed via /proc/kallsyms.

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);
    - void (*exit)(void); } - function pointers - stores addresses of entry point functions 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

Compiling multi-file modules

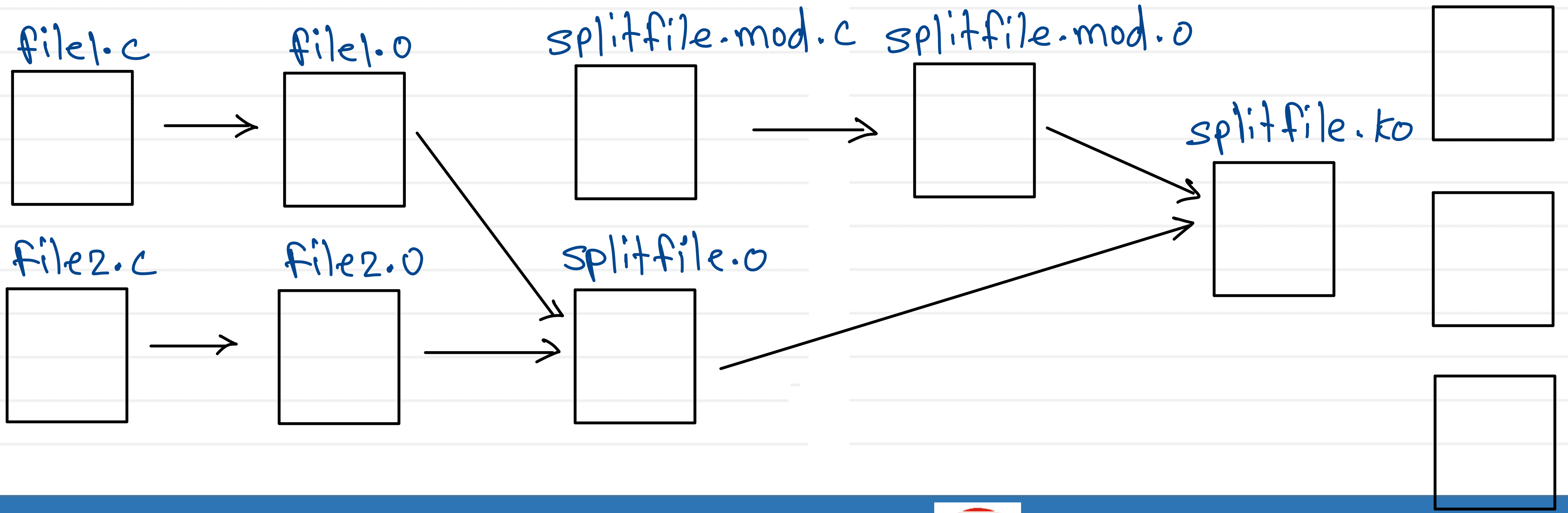
- To make kernel modules code maintainable, it is common practice to divide the code into multiple source files.

- To compile such files, Makefile should be updated.

- Makefile:

```
        splitfile-objs = file1.o file2.o
        obj-m = splitfile.o
```

- Compilation flow:



SUNBEAM

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com