Linux Device Drivers

dr. Andrea E. Naimoli

Università degli Studi di Trento
•Dipartimento di Ingegneria e Scienza dell'Informazione
via Sommarive 14
I - 38050 Trento - Povo, Italy

Prima di tutto…

<APRIAMO UN TERMINALE>

// Posizioniamoci nella propria cartella
cd /sysop/<login>

// Creazione di un nuovo disco delle differenze
qemu-img create -b /sysop/_shared/disk.img
-f qcow2 ./diskdiff.img

// Avvio della macchina virtuale
qemu -hda ./diskdiff.img
-boot c -localtime –usb -m 975 -redir tcp:2222::22

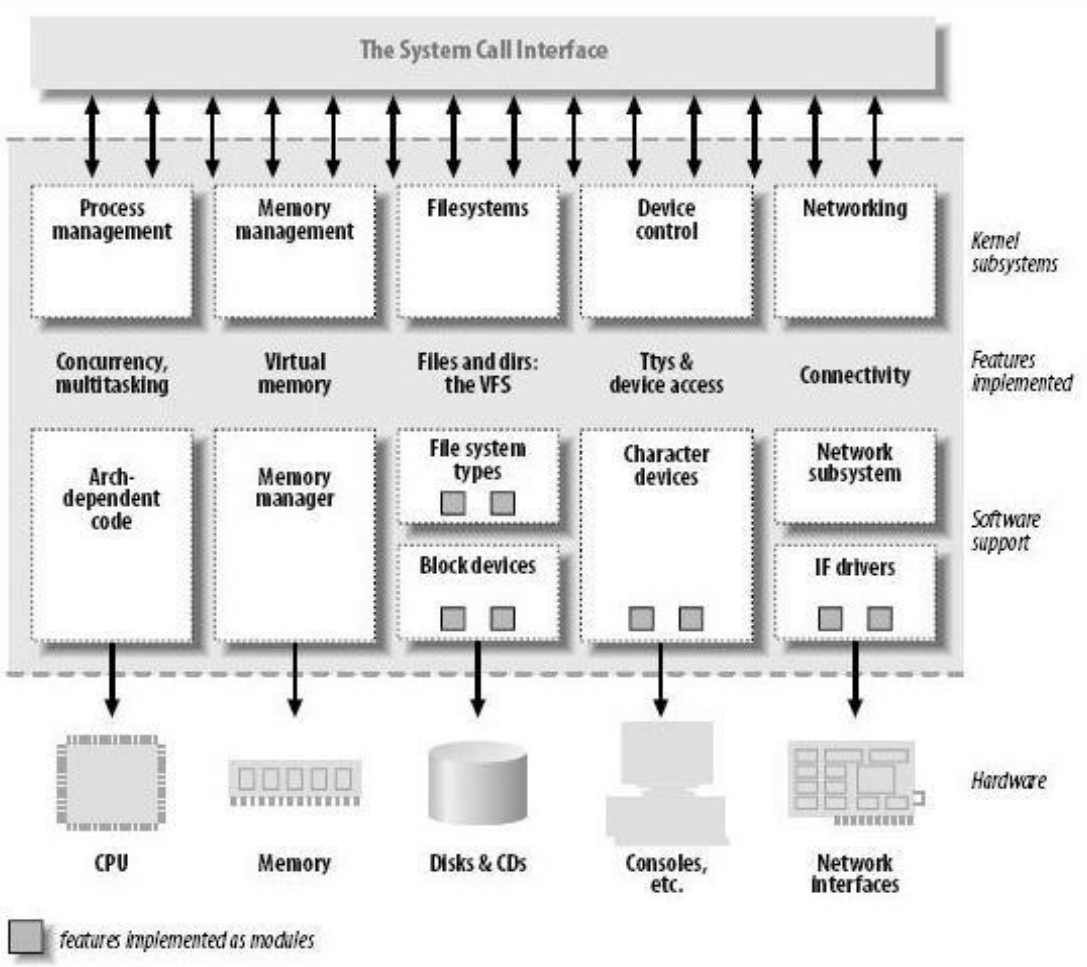
<APRIAMO UN ALTRO TERMINALE>

// Colleghiamoci alla macchina virtuale
ssh –p2222 root@localhost

Moduli Linux (1/3)

- Una delle caratteristiche migliori dei sistemi Linux è la possibilità di estendere a runtime le funzionalità offerte dal kernel
- L'utente ha la possibilità di aggiungere o rimuovere funzionalità mentre il sistema e` in esecuzione
- Ogni parte di codice e` chiamata **modulo**
- Ciascun modulo è formato da codice che può essere dinamicamente caricato o rimosso al kernel eseguito
- I comandi usati per aggiungere e rimuovere i moduli sono **modprobe**, **insmod** e **rmmod**

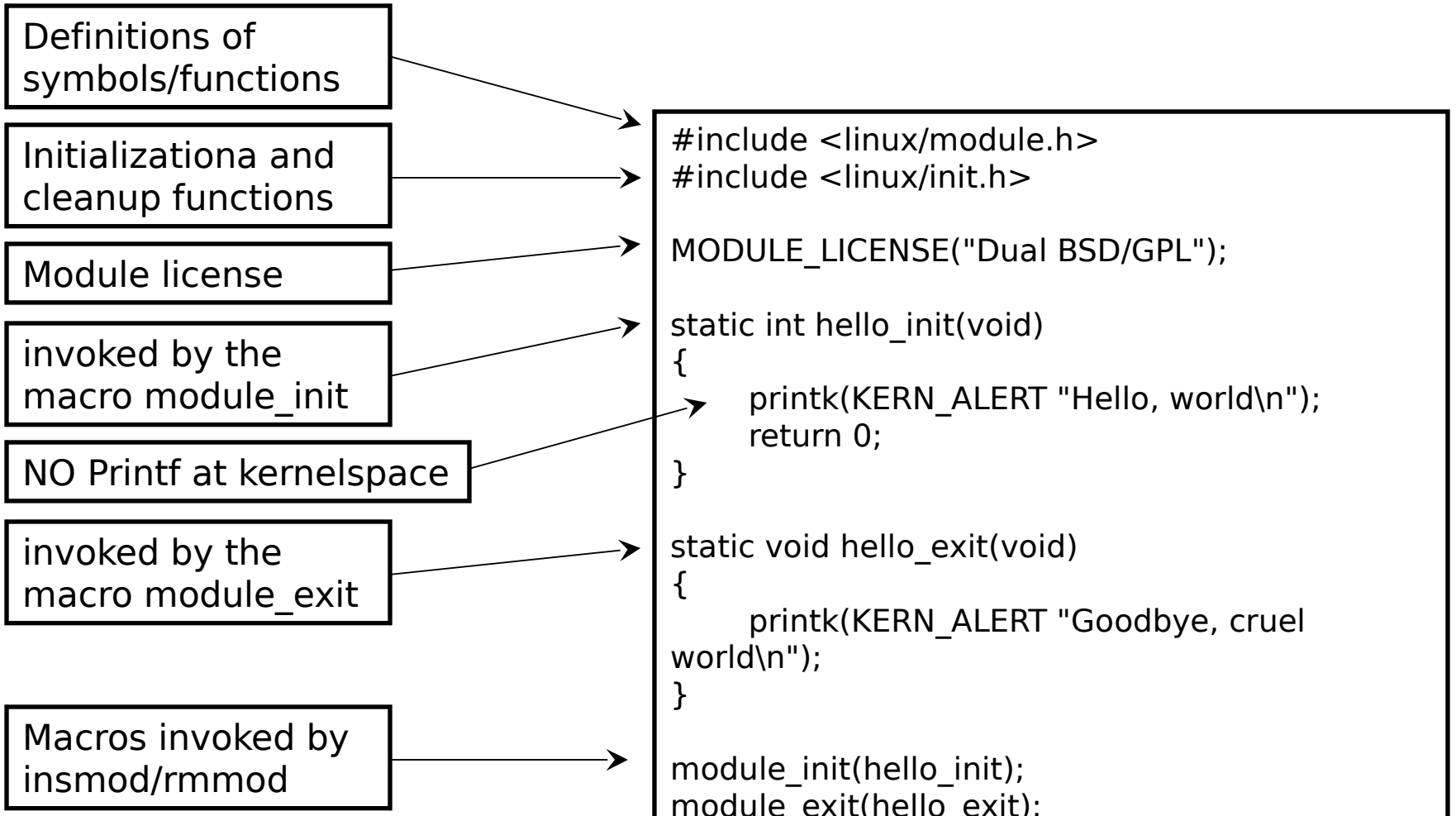
Moduli Linux (2/3)



Moduli Linux (3/3)

- I moduli possono gestire:
- *char devices*: come stream di bytes
  (/dev/ttyS[0..4], /dev/console, mouse)
- *block devices*: come insieme di blocchi
  (e.g. hard disks)
- *dispositivi di rete*:
  (e.g., ethernet or WiFi cards)
- *filesystems*:
  (e.g. ext2/ext3, reiserfs, FAT)
- *abstract layers*:
  (e.g. usbcore, cryptography)

Creazione modulo - helloworld



Compilazione (1/3)

- Per compilare correttamente il modulo, dovete creare un Makefile contenente almeno la seguente istruzione
  - ✓ **obj-m := hello.o**
- Questa istruzione indica che c'è un file oggetto pronto per essere costruito come modulo del kernel
- Per compilare il modulo d'esempio
  - ✓ **make -C /usr/src/linux M=$(pwd) modules**
  - ✓ Go to **/usr/src/linux**
  - ✓ Find the kernel makefile
  - ✓ Pass the old-current directory to the “kernel makefile”
  - ✓ Build the modules target (obj-m variable)

Compilazione (2/3)

- **make -C /usr/src/linux M=$(pwd) modules**
- L'opzione **-C** cambia la directory prima di leggere il Makefile, eseguendolo dentro la directory contenente i sorgenti del kernel
- Quindi torna dentro la directory corrente dove c'è il sorgente del modulo, prima di tentare la costruzione di **modules**
- Il target **modules** fa riferimento alla lista di moduli indicati in **obj-m**
- L'output è il modulo **hello.ko**

Compilazione (3/3)

- Define **Makefile** for module (beware of “spaces”/“tabs”!)
- Set correct folders

```
ifneq ($(KERNELRELEASE),)
# kbuild part of makefile ---------------------

obj-m := hello.o

else
# normal makefile ----------------------------

KDIR ?= /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

# ------------------------------------------------
endif
```

(Un)Loading dei moduli

- root può caricare:

  insmod ./hello.ko (modprobe hello.ko)
- root può rimuovere

  rmmod hello (modprobe -r helloworld)
- L'utente può controllare se il modulo è caricato:

  lsmod | grep hello
- e verificare se o meno è stato caricato correttamente.
- **lsmod** formatta il contenuto di **/proc/modules**
- Altrimenti, in base al sistema di log, l'output può essere scritto su **/var/log/syslog**

Esercizio (15 min.)

- Caricarlo
  - insmod hello.ko
- Visualizzare i messaggi del kernel
  - dmesg | tail
  - [1774.371847] Hello, world
  - [1774.371847] The process is insmod (pid 2622)
  - [1774.371847] Parameters assigned by insmod: 1, world
- Rimuovere il modulo e visulizzare i messaggi del kernel
  - rmmod hello

The kernel symbol table (1/2)

- The table contains the addresses of global kernel items—functions and variables—that are needed to implement modularized drivers
- When a module is loaded, any symbol exported by the module becomes part of the kernel symbol table
- The Linux kernel header files provide a convenient way to cope with the visibility of your symbols
- There are two MACROs to accomplish this goal:
- **`EXPORT_SYMBOL(name)`**
- **`EXPORT_SYMBOL_GPL(name)`**
- They must be exported in the global part of the file, outside any functions

The kernel symbol table (2/2)

- New modules can use symbols exported by your module, and you can stack new modules on top of other modules
- Module stacking is implemented in the mainstream kernel sources as well
- The msdos filesystem relies on symbols exported by the **`fat`** module, and each input USB device module stacks on the **`usbcore`** and **`input`** modules
- When using stacked modules, it is helpful to be aware of the **`modprobe`** utility

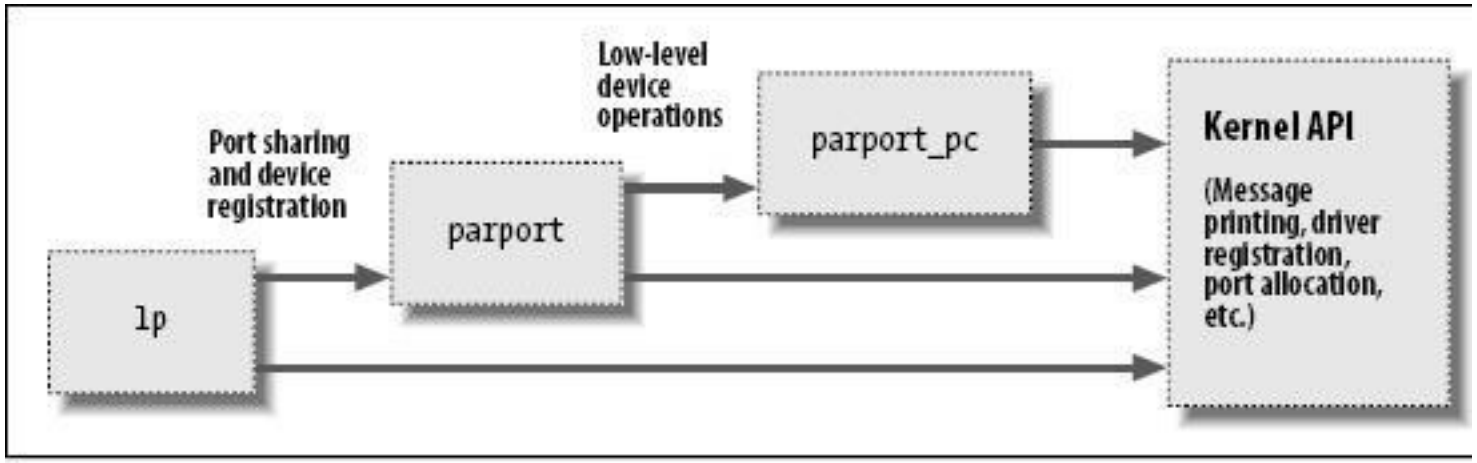


Figure 2-2. Stacking of parallel port driver modules

Modules Parameters

- Module parameters at runtime: how to?
- while inserting the module:

```
insmod hellop.ko howmany=4 whom="sysop"
```

- how the module makes them available:

```
#include<linux/moduleparam.h>
static char * whom = "sysop";
static int howmany = 1;
module_param(howmany,int,S_IRUGO);
module_param(whom,charp,S_IRUGO);
```

- available types: `bool/invbool, charp, int/long/short/uint/ulong/ushort`
- `cat /sys/module/hellop/parameters/whom`
- check url: https://www.linux.com/learn/linux-training/28065-the-kernel-newbie-corner-everything-you-wanted-to-know-about-module-paramete)

printk priorities

```
printk(<priority_level> "<message>"
[,<parameters>]);
```

KERN_EMERG: emergency before a crash

KERN_ALERT: requires immediate action

KERN_CRIT: serious hw or sw failures

KERN_ERR: error condition (e.g., hw difficulties)

KERN_WARNING: low impact problems

KERN_NOTICE: normal but relevant to say (e.g., security)

KERN_INFO: informational messages (e.g., hw detection)

KERN_DEBUG: debugging messages

Bibliography

- Linux Device Drivers, Ed. 3 – J. Corbet, A. Rubini, G. K-Hartman
- http://lwn.net/Kernel/LDD3