

Esercitazione: Sviluppo di moduli per il kernel Linux



Corso di Laurea in Ingegneria Informatica
Università degli Studi di Napoli Federico II
Anno Accademico 2023/2024, Canale San Giovanni



Sommario

- Sommario
 - Cenni sull'architettura del kernel Linux
 - Configurazione, compilazione e installazione del kernel
 - Sviluppo di un modulo del kernel
 - Sviluppo di una chiamata di sistema
- Riferimenti
 - The Linux Kernel Module Programming Guide,
<http://tldp.org/LDP/lkmpg/2.6/html/>
<https://sysprog21.github.io/lkmpg/>
 - R. Love, Linux Kernel Development (3^a ed.), cap. 2, 5, 17, 18



Struttura e compilazione del kernel Linux

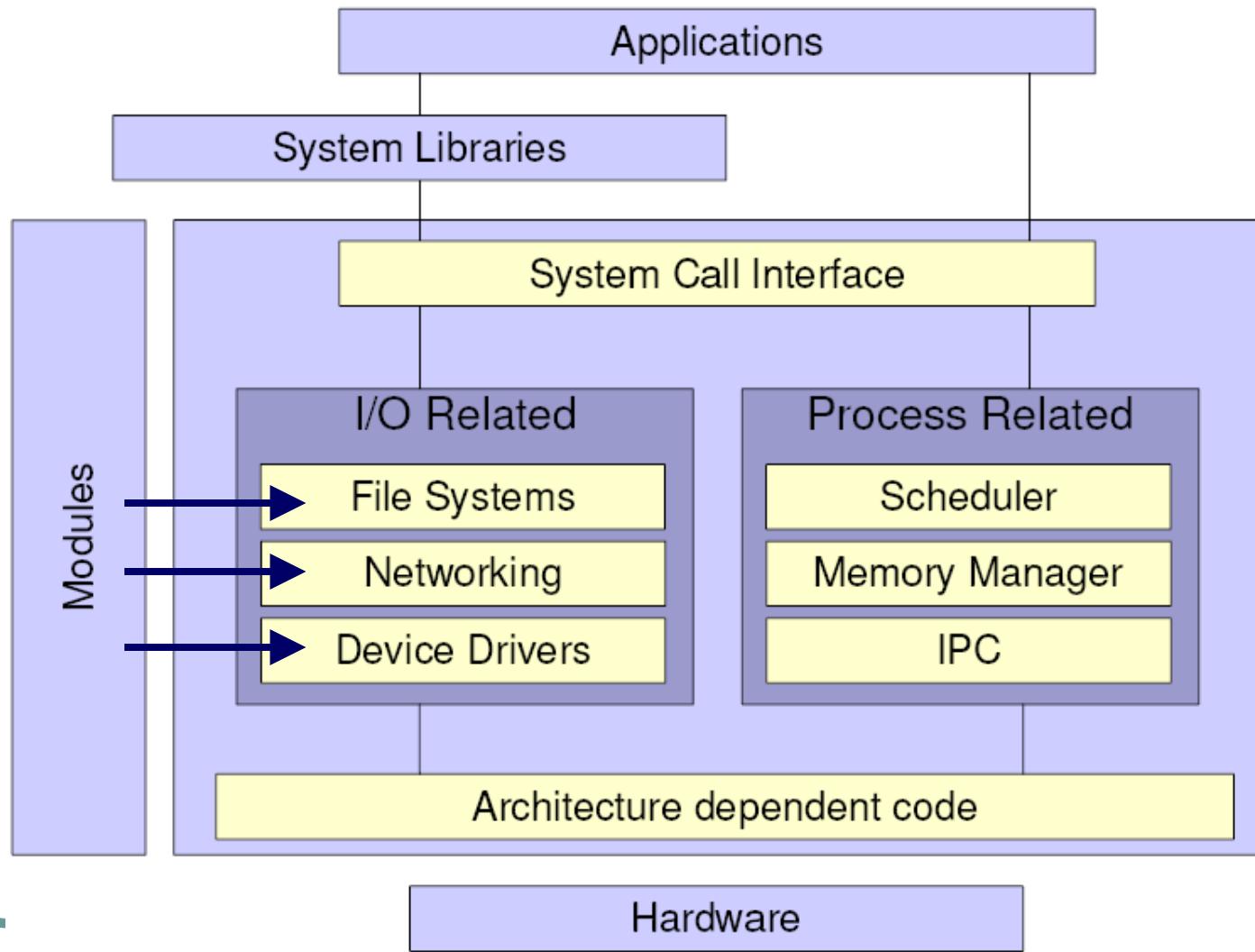


Il kernel Linux

- Linux è un sistema operativo **monolitico** e **modulare**
 - Tutto il codice del kernel è contenuto in un **unico file eseguibile**
 - Utilizza un unico spazio di indirizzamento
 - Permette di caricare ulteriore codice da **"moduli"**, a tempo di esecuzione



Architettura concettuale





Moduli kernel

```
$ lsmod
```

Module	Size	Used by
xt_conntrack	16384	1
nft_chain_nat	16384	3
xt_MASQUERADE	20480	1
...		
e1000	159744	0
scsi_transport_spi	36864	1 mptspi
mptscsih	45056	1 mptspi
mptbase	110592	2 mptspi, mptscsih

Device driver per schede di rete
Intel® Gigabit Ethernet Network Connection



Alcuni esempi di parti del kernel

- Versione "navigabile" del codice sorgente del kernel Linux: <https://elixir.bootlin.com/>

The screenshot shows a web-based interface for navigating kernel source code. The top navigation bar includes a dropdown for 'linux' (set to v5.15), a search bar, and a 'Search Identifier' button. The main content area displays the file '/ include / linux / sched.h'. The code shown is:

```
722  
723 struct task_struct {  
724 #ifdef CONFIG_THREAD_INFO_IN_TASK  
725 /*  
726 * For reasons of header soup (see current_thread_info()), this  
727 * must be the first element of task_struct.  
728 */  
729 struct thread_info thread_info;  
730 #endif  
731 unsigned int __state;  
732  
733 #ifdef CONFIG_PREEMPT_RT  
734 /* saved state for "spinlock sleepers" */  
735 unsigned int saved_state;  
736 #endif  
737
```

The left sidebar shows a tree view of kernel versions and specific commits. At the bottom, it says 'linux v5.15.6' and 'powered by Elixir 2.1'.



Alcuni esempi di parti del kernel

- **Process control block (task_struct):**
 - <https://elixir.bootlin.com/linux/v5.15.6/source/include/linux/sched.h#L723>
 - **int prio**
 - **struct mm_struct *mm**
 - **struct list_head tasks**
 - **struct task_struct *parent**
 - **char comm[TASK_COMM_LEN]**
- **System call fork():**
 - <https://elixir.bootlin.com/linux/v5.9.11/source/kernel/fork.c#L2415>
- **ISR di un device driver per tastiera:**
 - <https://elixir.bootlin.com/linux/v5.9.11/source/drivers/input/keyboard/atkbd.c#L409>



Compilazione del kernel

- Perché compilare il kernel?
 - Minore occupazione di memoria e storage (es. sistemi embedded)
 - Abilitare driver e funzionalità specifiche
 - Installare modifiche di terze parti
 - Ottimizzazione (es. selezione CPU family, SMP, layout memoria)
 - Utilizzo del **sistema di build per sviluppare un nuovo modulo kernel**
 - Studio del kernel!



KernelNewbies.org



- <https://kernelnewbies.org>
- Kernel hacking howtos
- Kernel changelog
- Kernel Janitors project
- Kernel mentors
- Job postings, career advice



Versione del kernel

```
$ uname -a
```

```
Linux Ubuntu-SO 5.15.0-53-generic #59-Ubuntu SMP Mon  
Oct 17 18:53:30 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
```

```
$ uname -r
```

```
5.15.0-53-generic
```

È raccomandabile installare la **stessa versione (o simile) del kernel** già utilizzata dalla propria distribuzione



Dipendenze e sorgenti

- Prima di iniziare, occorre copiare i **sorgenti**, e installare librerie e tool necessari alla compilazione
 - Es. Ubuntu: <https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel>

```
$ sudo gedit /etc/apt/sources.list
# togliere il cancelletto dalle righe con "deb-src"

$ sudo apt update

$ sudo apt build-dep linux linux-image-$(uname -r)

$ sudo apt install qt5-default

$ wget https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.4.0.tar.gz

# Oppure:
# git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

```
1 #deb cdrom:[Ubuntu 20.04.1 LTS _Focal Fossa]
2
3 # See http://help.ubuntu.com/community/Upgr...
4 # newer versions of the distribution.
5 deb http://it.archive.ubuntu.com/ubuntu/ fo...
6 # deb-src http://it.archive.ubuntu.com/ubuntu...
7
```



Cartella di lavoro

```
so@Ubuntu-SO: /usr/src/linux-5.4.210
so@Ubuntu-SO: /usr/src$ sudo tar zxf linux-5.4.210.tar.gz
so@Ubuntu-SO: /usr/src$ sudo chown -R root:so linux-5.4.210
so@Ubuntu-SO: /usr/src$ sudo chmod -R g+w linux-5.4.210
so@Ubuntu-SO: /usr/src$ cd linux-5.4.210/
so@Ubuntu-SO: /usr/src/linux-5.4.210$ ls
arch    COPYING  Documentation  include  Kbuild   lib      Makefile  README  security  usr
block   CREDITS  drivers       init     Kconfig  LICENSES  mm      samples  sound   virt
certs   crypto    fs           ipc     kernel   MAINTAINERS  net      scripts  tools
so@Ubuntu-SO: /usr/src/linux-5.4.210$
```



Configurazione del kernel

- La fase di configurazione permette di **selezionare le funzionalità e i moduli** da compilare, escludendo ciò che non serve
- Per modificare la configurazione, usare uno tra i seguenti comandi:

`make xconfig`

(grafico, consigliato)

`make menuconfig`

(linea di comando)

`make config`

(linea di comando)



Configurazione del kernel

- Per creare una **configurazione di partenza**:
make localmodconfig
- Inizializza la configurazione in base ai **moduli in uso** nel sistema



Configurazione grafica

The screenshot shows the Linux kernel configuration interface (menuconfig) with a blue header bar containing the title "Configurazione grafica". Below the header is a toolbar with icons for file operations like Open, Save, and Help. The main window has a "File Edit Option" menu at the top. The left pane displays a tree view of configuration options under the heading "Option". The "Processor type and features" option is currently selected and highlighted with a dark grey background. The right pane shows the detailed configuration for the selected option. It includes several checkboxes and radio buttons. One radio button, labeled "PC-compatible", is selected and highlighted with a black background. A descriptive text below the radio buttons states: "Choose this option if your computer is a standard PC or compatible." At the bottom of the right pane, there is a link to "Documentation for this section".

File Edit Option Help

Option

- Code maturity level options
- General setup
 - Configure standard kernel features (for small systems)
- Loadable module support
- Block layer
- Processor type and features**
- Firmware Drivers
- Power management options (ACPI, APM)
 - CPU Frequency scaling
- Bus options (PCI, PCMCIA, EISA, MCA, ISA)
 - PCCARD (PCMCIA/CardBus) support
 - Support for PCI Hotplug (EXPERIMENTAL)
- Executable file formats
- Networking
- Device Drivers
 - Generic Driver Options
 - Connector - unified userspace <-> kernelspace interface
- Memory Technology Device (MTD) support**
 - RAM/ROM/Flash chip drivers
 - Mapping drivers for chip access
 - Self-contained MTD device drivers
 - NAND Device Support
 - OneNAND Device Support
 - UBI - Unsorted block images

Option

- Tickless System (Dynamic Ticks)
- High Resolution Timer Support
- Symmetric multi-processing support

Subarchitecture Type

- PC-compatible**
- AMD Elan
- Voyager (NCR)
- NUMAQ (IBM/Sequent)
- Summit/EXA (IBM x440)
- Support for other sub-arch SMP systems with more than 8 CPUs

PC-compatible (X86_PC)

Choose this option if your computer is a standard PC or compatible.



Opzioni di configurazione

- Ciascuna opzione può essere
 - compilata all'interno del kernel (**Y**, simbolo ✓)
 - compilata in un modulo separato (**M**, simbolo •)
 - esclusa dalla compilazione (**N**, casella vuota)
- Alcune opzioni richiedono valori numerici o stringhe



Opzioni di configurazione

- **LOCALVERSION** è il nome assegnato al kernel che compileremo
- **LOCALVERSION="-CorsoSO"**: il nostro kernel sarà indicato con "**5.4.0-CorsoSO**"

The screenshot shows two panels of the Linux kernel configuration tool. The left panel displays a tree of configuration options under 'Compiler: gcc (Ubuntu 11.3.0-1ubuntu1)'. The 'General setup' option is selected. The right panel shows the 'Option' screen where the 'Local version - append to kernel release' field is set to '-CorsoSO'. Below this, the 'Local version - append to kernel release (LOCALVERSION)' and 'CONFIG_LOCALVERSION:' sections are visible, with a note explaining the purpose of appending an extra string to the kernel version.

Compiler: gcc (Ubuntu 11.3.0-1ubuntu1)

General setup

- IRQ subsystem
- Timers subsystem
- CPU/Task time and stats account
- RCU Subsystem
- Scheduler features
- Control Group support
 - CPU controller
 - Namespaces support
 - Configure standard kernel
 - Kernel Performance Events API
- 64-bit kernel
- Processor type and features
 - Linux guest support
 - Supported processor vendor
 - Performance monitoring
- Power management and ACPI options

Option

- Compile also drivers which will not load
- Local version - append to kernel release: **-CorsoSO**
- Automatically append version information to the version string

Build ID Salt:

Kernel compression mode

- Gzip
- Bzip2
- LZMA

-CorsoSO

Local version - append to kernel release (LOCALVERSION)

CONFIG_LOCALVERSION:

Append an extra string to the end of your kernel version.
This will show up when you type uname, for example.



Alcuni esempi di opzioni

- Modalità di preemption del kernel
- Frequenza del timer
- Ottimizzazioni per la CPU (SMP, HyperThreading, famiglia di CPU,...)



Personalizzare la configurazione

- Molte opzioni riguardano il **supporto hardware** della macchina su cui il kernel sarà eseguito
- Per informazioni sul proprio hw:

```
lspci -vvv
```

```
cat /proc/cpuinfo
```

```
hal-device-manager
```

```
lsmod
```



Configurazione su una macchina virtuale VMware

```
$ lspci
```

.....

```
00:07.2 USB Controller: Intel Corporation 82371AB/EB/MB PIIX4 USB
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:0f.0 VGA compatible controller: VMware Inc [VMware SVGA II] PCI Display
Adapter
00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-
MPT Dual Ultra320 SCSI (rev 01)
00:11.0 PCI bridge: VMware Inc Unknown device 0790 (rev 02)
02:00.0 Ethernet controller: Advanced Micro Devices [AMD] 79c970 [PCnet32
LANCE] (rev 10)
02:01.0 Multimedia audio controller: Ensoniq ES1371 [AudioPCI-97] (rev 02)
02:02.0 USB Controller: VMware Inc Unknown device 0770
```



Esempio: opzioni per driver SATA

The screenshot shows the Linux kernel configuration interface (menuconfig) with a focus on SATA driver options. The left pane displays a tree view of device driver categories, and the right pane shows detailed options for the selected category.

Left Pane (Device Drivers):

- Device Drivers
 - Generic Driver Options
 - Connector - unified userspace <-> kernelspace li
 - [+] Memory Technology Device (MTD) support**
 - RAM/ROM/Flash chip drivers
 - Mapping drivers for chip access
 - Self-contained MTD device drivers
 - [+] NAND Device Support**
 - [+] OneNAND Device Support**
 - UBI - Unsorted block images
 - Parallel port support
 - Plug and Play support
 - Block devices
 - Misc devices
 - [+] ATA/ATAPI/MFM/RLL support**
 - [+] SCSI device support**
 - SCSI Transports
 - SCSI low-level drivers
 - PCMCIA SCSI adapter support
 - [+] Serial ATA (prod) and Parallel ATA (experimental)**
 - Old CD-ROM drivers (not SCSI, not IDE)
 - Multi-device support (RAID and LVM)
 - Fusion MPT device support
 - IEEE 1394 (FireWire) support

Right Pane (Option details):

- ...
- [+] AHCI SATA support**
- [+] ServerWorks Frodo / Apple K2 SATA support
- [+] Intel ESB, ICH, PIIX3, PIIX4 PATA/SATA support
- [+] Marvell SATA support (HIGHLY EXPERIMENTAL)
- [+] NVIDIA SATA support
- [+] Pacific Digital ADMA support
- [+] Pacific Digital SATA QStor support
- [+] Promise SATA TX2/TX4 support
- [+] Promise SATA SX4 support

Description for AHCI SATA support (SATA_AHCI):

This option enables support for AHCI Serial ATA.
If unsure, say N.



File di configurazione

- La configurazione viene salvata nel file **".config"** nella cartella dei sorgenti

```
so@Ubuntu-S0:/usr/src/linux-5.4.210$ ls -la
totale 1176
drwxrwxr-x  24 root so      4096 dic  3 17:41 .
drwxr-xr-x  12 root root    4096 dic  3 17:22 ..
drwxrwxr-x  27 root so      4096 ago 11 10:57 arch
drwxrwxr-x  3 root so      4096 dic  3 17:44 block
drwxrwxr-x  2 root so      4096 dic  3 17:33 certs
-rw-rw-r--  1 root so     15318 ago 11 10:57 .clang-format
-rw-rw-r--  1 root so       59 ago 11 10:57 .cocciconfig
-rw-rw-r--  1 so   so    139590 dic  3 17:32 .config
```

NOTA: I file che **iniziano con il punto (".")** sono detti **file invisibili**.
Non sono mostrati dal comando "ls", ma occorre il comando "ls -a".



Compilazione

- Per rimuovere i file di una precedente compilazione:

```
make clean
```

- Per compilare sia il kernel sia i moduli:

```
make
```

- Per compilare in maniera parallela, utile per sfruttare SMP (es. 9 jobs se si hanno 8 core):

```
make -j 9
```



A terminal window titled "so@so-vbox: ~/linux-5.4" showing the output of a "make -j 9" command. The output lists various compilation steps: AR, LD, GEN, CHK, LD, MODPOST, MODINFO, LD, KSYM, LD, KSYM, LD, SORTEX, SYMSMAP, and Building modules, stage 2.

```
+ so@so-vbox: ~/linux-5.4
AR      drivers/xen/built-in.a
LD [M]  drivers/xen/xen-privcmd.o
AR      drivers/built-in.a
GEN     .version
CHK     include/generated/compile.h
LD      vmlinux.o
MODPOST vmlinux.o
MODINFO modules.builtin.modinfo
LD      .tmp_vmlinux1
KSYM   .tmp_kallsyms1.o
LD      .tmp_vmlinux2
KSYM   .tmp_kallsyms2.o
LD      vmlinux
SORTEX vmlinux
SYMSMAP System.map
Building modules, stage 2.
```



Installazione

- Installare l'**immagine** del kernel nella cartella `/boot`

```
sudo cp arch/x86_64/boot/bzImage /boot/vmlinuz-VERSIONE
```

- Copiare i **moduli** kernel nella cartella `/lib/modules/`

```
sudo make modules_install
```

- Salvare una copia del file di configurazione:

```
sudo cp .config /boot/config-VERSIONE
```

Sostituire "VERSIONE" (es. con "5.4.0-CorsoSO")



Cartella /boot

```
so@Ubuntu-SO:/usr/src$ ls -l /boot/
totale 195632
-rw-r--r-- 1 root root 261841 ago 10 07:49 config-5.15.0-47-generic
-rw-r--r-- 1 root root 261861 ott 13 07:40 config-5.15.0-52-generic
-rw-r--r-- 1 root root 261837 ott 17 18:36 config-5.15.0-53-generic
-rw-r--r-- 1 root root 139590 dic  3 18:02 config-5.4.210-CorsoSO
drwxr-xr-x 5 root root 4096 dic  5 16:14 grub
lrwxrwxrwx 1 root root 28 nov 22 12:33 initrd.img -> initrd.img-5.15.0-53-generic
-rw-r--r-- 1 root root 32745364 set 15 12:37 initrd.img-5.15.0-47-generic
-rw-r--r-- 1 root root 32750951 ott 26 12:46 initrd.img-5.15.0-52-generic
-rw-r--r-- 1 root root 32746358 nov 22 12:38 initrd.img-5.15.0-53-generic
-rw-r--r-- 1 root root 35900964 dic  5 16:14 initrd.img-5.4.210-CorsoSO
lrwxrwxrwx 1 root root 28 nov 22 12:33 initrd.img.old -> initrd.img-5.15.0-52-generic
-rw----- 1 root root 6245466 ago 10 07:49 System.map-5.15.0-47-generic
-rw----- 1 root root 6249017 ott 13 07:40 System.map-5.15.0-52-generic
-rw----- 1 root root 6250186 ott 17 18:36 System.map-5.15.0-53-generic
lrwxrwxrwx 1 root root 25 nov 22 12:33 vmlinuz -> vmlinuz-5.15.0-53-generic
-rw----- 1 root root 11530816 ago 11 07:44 vmlinuz-5.15.0-47-generic
-rw----- 1 root root 11543392 ott 13 07:49 vmlinuz-5.15.0-52-generic
-rw----- 1 root root 11548224 ott 17 18:41 vmlinuz-5.15.0-53-generic
-rw-r--r-- 1 root root 11864384 dic  5 16:13 vmlinuz-5.4.210-CorsoSO
lrwxrwxrwx 1 root root 25 nov 22 12:33 vmlinuz.old -> vmlinuz-5.15.0-52-generic
so@Ubuntu-SO:/usr/src$
```

La cartella **/boot** raccoglie i file del kernel

- config
- initrd.img
- vmlinuz
- System.map (opzionale)



Cartella /lib/modules

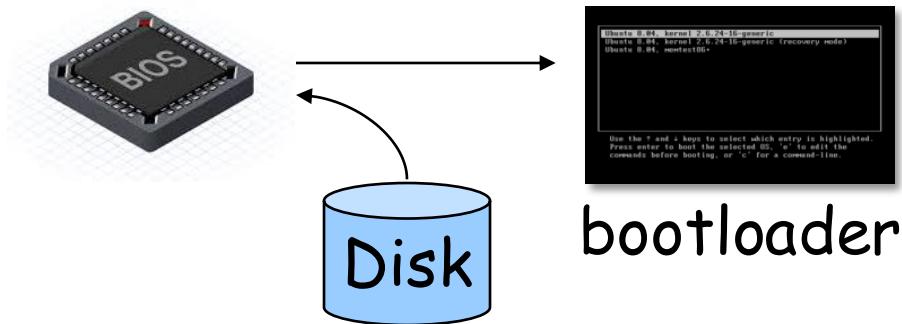
```
so@Ubuntu-S0:/usr/src$ ls -l /lib/modules
totale 24
drwxr-xr-x 6 root root 4096 set 15 12:37 5.15.0-47-generic
drwxr-xr-x 3 root root 4096 ott 26 12:47 5.15.0-48-generic
drwxr-xr-x 3 root root 4096 nov 22 12:39 5.15.0-50-generic
drwxr-xr-x 6 root root 4096 ott 26 12:46 5.15.0-52-generic
drwxr-xr-x 6 root root 4096 nov 22 12:34 5.15.0-53-generic
drwxr-xr-x 4 root root 4096 dic  5 16:13 5.4.210-CorsoS0
so@Ubuntu-S0:/usr/src$
```

La cartella **/lib/modules** raccoglie i moduli del kernel (una sottocartella per ogni versione)



Bootstrap del nuovo kernel

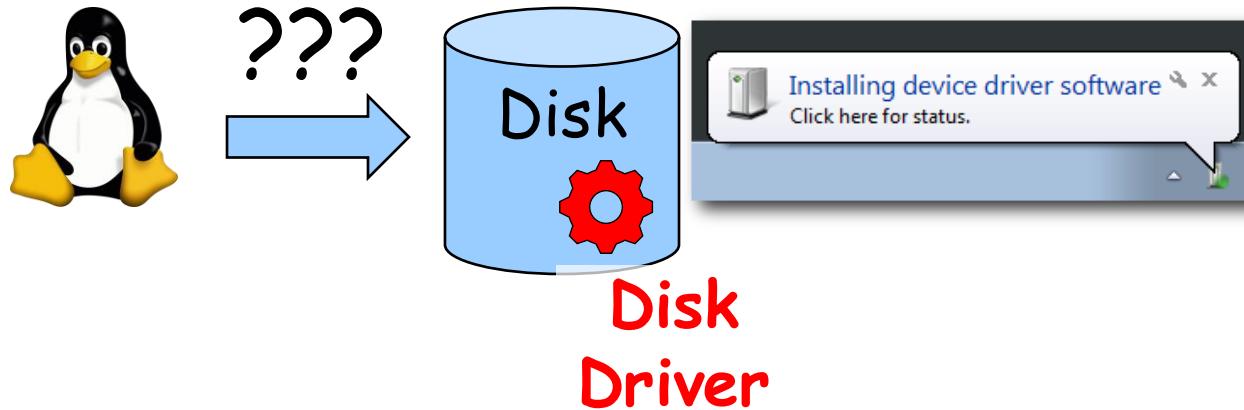
- Il **bootloader** è un piccolo programma che viene eseguito all'avvio del computer
- Viene prelevato dal **BIOS** dai primi settori del disco
- Il bootloader può interagire con l'utente per selezionare quale sistema operativo avviare





Caricamento Driver di I/O

- Il kernel ha bisogno dei **driver di I/O** (es., disco, rete, filesystem) per **caricare i moduli** al **boot**
- ...e se a loro volta anche i **driver di I/O sono moduli?**





Caricamento Driver di I/O

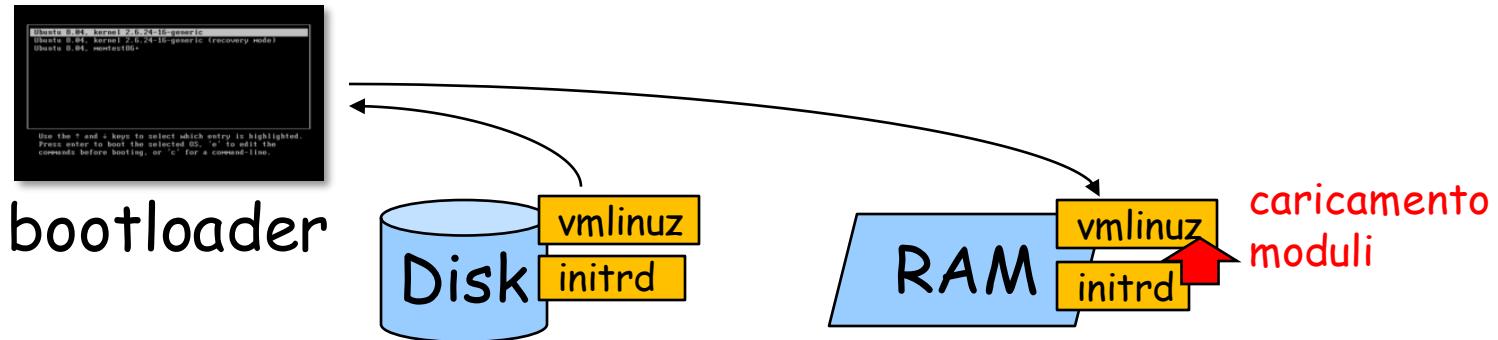
- È il caso tipico delle distribuzioni Linux
 - Le distribuzioni includono il **maggior numero possibile di driver di I/O**, per maggiore compatibilità
 - I driver sono **compilati come moduli**
 - Evita che il file vmlinuz diventi eccessivamente grande

```
$ find /lib/modules/$(uname -r) -name "*.ko" | wc -l  
1022
```



Initial RAM Disk (initrd)

- I moduli strettamente necessari sono copiati in un file detto “**initial RAM disk**”
- Il **bootloader** carica questo file su RAM
- Il kernel preleva da lì il driver del disco





Creazione Initial RAM Disk

- Il comando va eseguito **dopo aver installato i moduli con `make modules_install`**

- Per creare l'initial RAM disk:

```
sudo mkinitramfs -o /boot/initrd.img-VERSIONE VERSIONE
```

- Ad esempio:

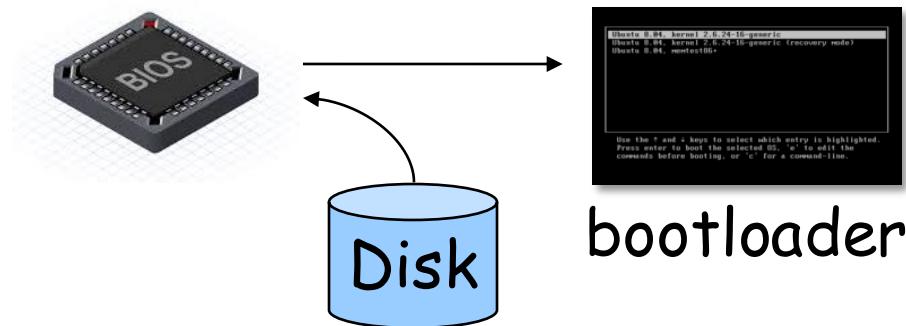
```
sudo mkinitramfs  
        -o /boot/initrd.img-5.4.0-CorsoSO 5.4.0-CorsoSO
```

- Altre distribuzioni utilizzano il programma `mkinitrd`



Configurazione bootloader

- Occorre **configurare il bootloader** per utilizzare il kernel appena compilato
- **GRUB** è il bootloader più comune tra le distribuzioni Linux





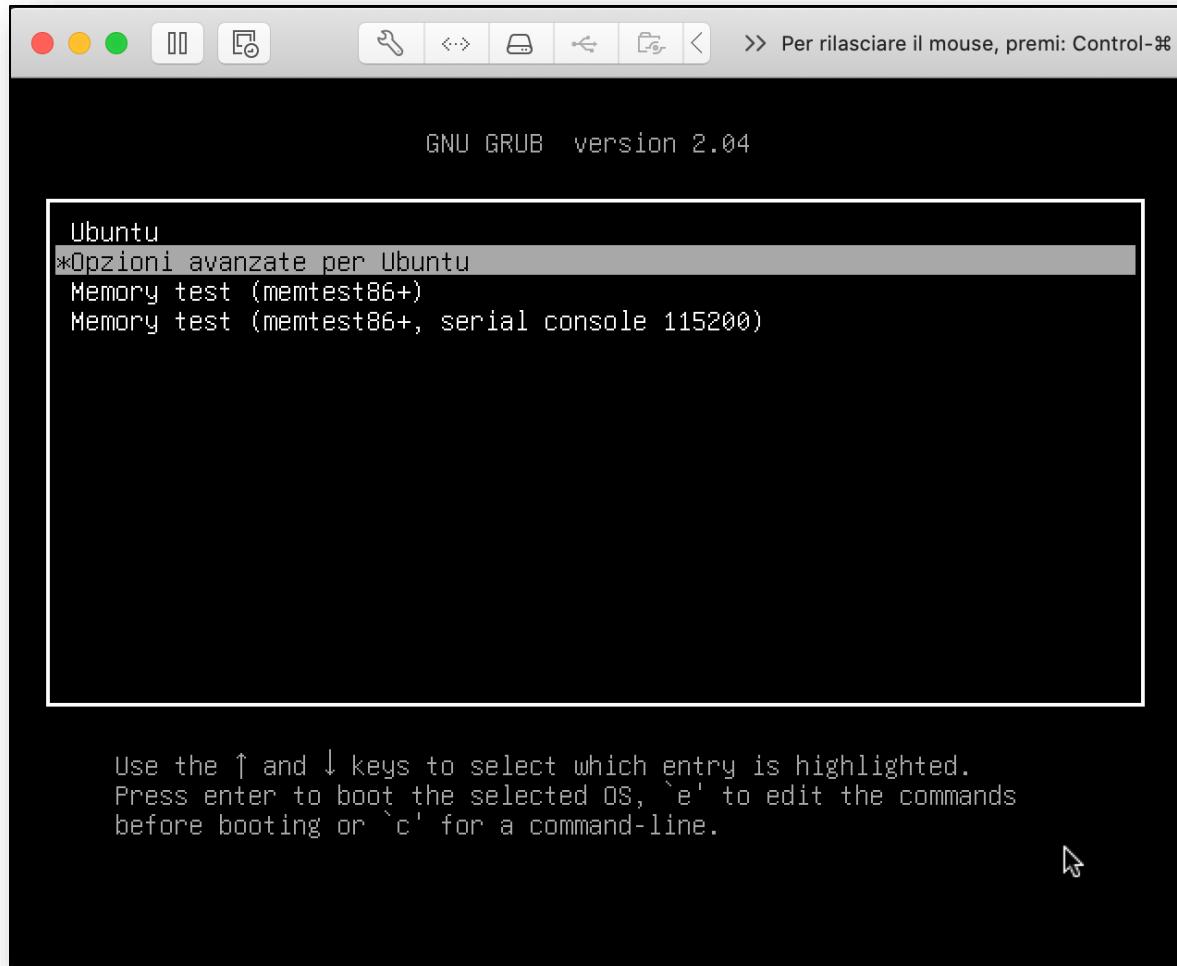
GRUB

- Modificare il file `/etc/default/grub`, per fare apparire il menù di GRUB all'avvio
 - Aggiungere `#` all'inizio della riga `GRUB_TIMEOUT_STYLE=hidden`
 - Impostare `GRUB_TIMEOUT` a un valore maggiore di 0
- Aggiornare GRUB:
`sudo update-grub`
- Al prossimo riavvio, si potrà scegliere di avviare il nuovo kernel invece di quello di default

NOTA: il comando “`make install`” copia automaticamente i file `vmlinuz` e `initrd` in `/boot`, ed aggiorna il bootloader



GRUB





...ricapitolando (procedura semplificata)

1. Scaricare i sorgenti (es., dal sito www.kernel.org)
2. Effettuare la configurazione
 - **`sudo make localmodconfig`** (seleziona automaticamente i moduli kernel in uso nel sistema)
 - Modificare la configurazione, settando il parametro **`LOCALVERSION`** (es., la stringa "-CorsoSO")
3. Lanciare la compilazione
 - **`sudo make`**
4. Installare i moduli
 - **`sudo make modules_install`**
5. Installare vmlinuz e aggiornare il bootloader
 - **`sudo make install`**



Utilizzo del nuovo kernel

- Comando per verificare la versione del kernel in uso:
`uname -a`
- Se si usa un nuovo kernel in una VM, può occorrere installare nuovamente i **vmware-tools**
 - Essi includono dei moduli kernel, che vanno ricompilati per poter essere eseguiti nel nuovo kernel



Errori comuni

- La **configurazione** del kernel non include alcuni moduli necessari all'avvio
 - es., i driver del disco oppure il filesystem
- La **creazione dello Initial RAM Disk** è fallita
 - non bisogna ignorare eventuali warning durante la creazione
- Errori nelle **convenzioni dei nomi**
 - es., l'uso della stringa LOCALVERSION all'interno dei nomi dei file



Note (VM del corso)

- Prima di creare lo **Init RAM Disk**, cancellare il file:

```
/etc/default/grub.d/50-cloudimg-settings.cfg
```

- Prima di lanciare **update-grub**, modificare il file:

```
/etc/initramfs-tools/initramfs.conf
```

configurando **COMPRESS=bzip2**



Linux Kernel Modules (LKM)



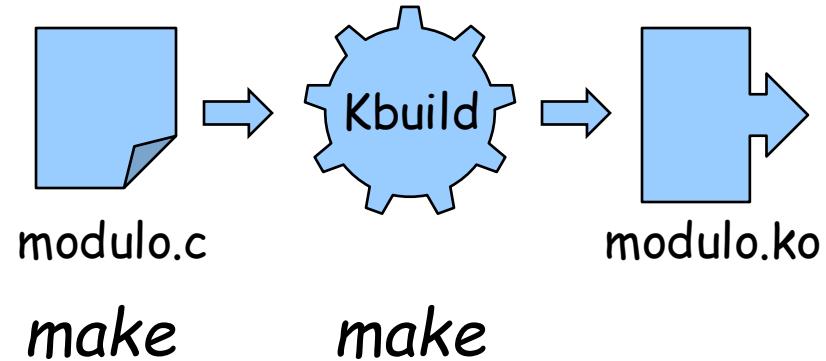
Moduli del kernel Linux

- I **moduli del kernel** permettono di rendere il sistema più flessibile:
 - Installazione di software di terze parti (es., device driver)
 - Caricamento di funzionalità on-demand
 - Sviluppo e collaudo di nuovo codice senza dover riavviare il sistema



Ciclo di vita di un modulo

- I sorgenti del modulo vengono compilati sfruttando i **tool (Kbuild)** e gli **header file** inclusi nei **sorgenti del kernel**
- Il modulo viene caricato **dinamicamente**, su richiesta dell'utente, oppure dai processi di sistema





Esecuzione del modulo

- Un modulo è una collezione di funzioni ("entry point")
- Eseguite quando si verificano:
 - **Caricamento e rimozione** del modulo
 - **Chiamate di sistema**
(o altre interazioni tra user- e kernel-mode)
 - **Interruzioni**



Un semplice modulo

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */

MODULE_LICENSE("GPL");

int init_module(void) {
    printk(KERN_INFO "Hello world 1.\n");

    return 0;
}

void cleanup_module(void) {
    printk(KERN_INFO "Goodbye world 1.\n");
}
```

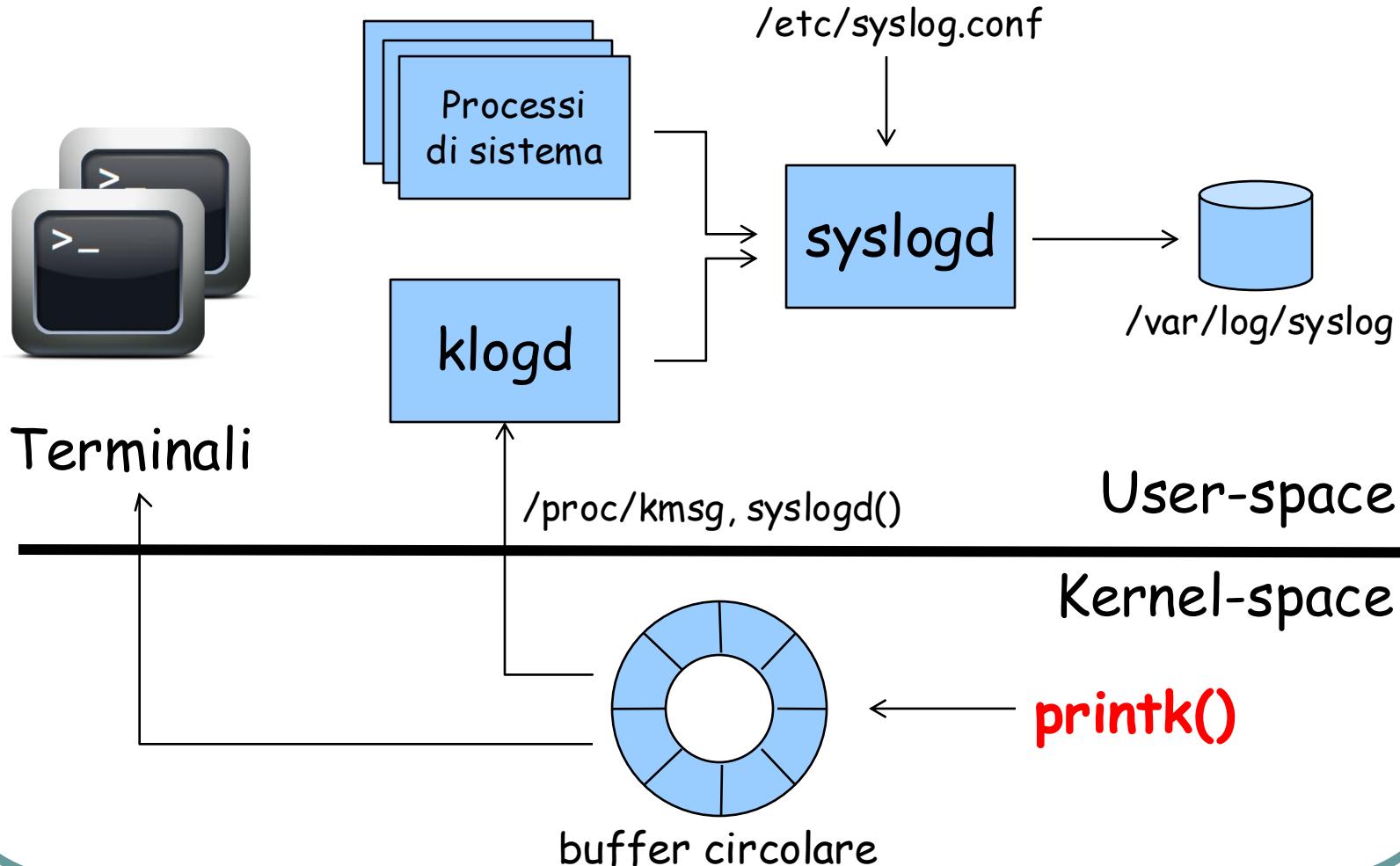


La funzione printk

- Il codice del kernel **non ha associato un terminale** su cui far apparire messaggi!
- **printk()** scrive su un buffer circolare, e i messaggi sono rediretti verso un processo di sistema (**syslogd**)
- Consultabili via:
 - il comando **dmesg**
 - il file di log (**/var/log/syslog**)
 - in tutti i terminali (nei casi da **alta severità**)



Logging nel kernel





Livelli di severità dei log

Loglevel	Description
KERN_EMERG	An emergency condition; the system is probably dead.
KERN_ALERT	A problem that requires immediate attention.
KERN_CRIT	A critical condition.
KERN_ERR	An error.
KERN_WARNING	A warning.
KERN_NOTICE	A normal, but perhaps noteworthy, condition.
KERN_INFO	An informational message.
KERN_DEBUG	A debug message—typically superfluous.



Makefile di un modulo

- La compilazione si basa sul sistema **Kbuild** di compilazione del kernel (Makefile e script per la shell)
- I sorgenti ed il Makefile del nuovo modulo possono risiedere in qualunque directory

```
obj-m += hello-1.o  
  
VERSIONE_KERNEL=$(shell uname -r)  
  
all:  
    make -C /lib/modules/$(VERSIONE_KERNEL)/build M=$(PWD) modules  
clean:  
    make -C /lib/modules/$(VERSIONE_KERNEL)/build M=$(PWD) clean
```

Directory corrente

È un link simbolico verso la cartella
dei sorgenti del kernel (creato da
"make modules_install")



Compilazione ed utilizzo

- Compilazione:

```
$ make
make -C /lib/modules/5.4.0-56-generic/build M=/home/so/so_esempi/moduli-kernel/hello-1 modules
make[1]: ingresso nella directory «/usr/src/linux-headers-5.4.0-56-generic»
  CC [M]  /home/so/so_esempi/moduli-kernel/hello-1/hello-1.o
Building modules, stage 2.
MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/so/so_esempi/moduli-kernel/hello-1/hello-1.o
see include/linux/module.h for more information
  CC [M]  /home/so/so_esempi/moduli-kernel/hello-1/hello-1.mod.o
  LD [M]  /home/so/so_esempi/moduli-kernel/hello-1/hello-1.ko
make[1]: uscita dalla directory «/usr/src/linux-headers-5.4.0-56-generic»
```

- Caricamento:

```
sudo insmod ./hello-1.ko
```

- Visualizzazione elenco moduli:

```
lsmod
```

- Rimozione:

```
sudo rmmod hello-1
```

- Informazioni sul modulo:

```
modinfo ./hello-1.ko
```

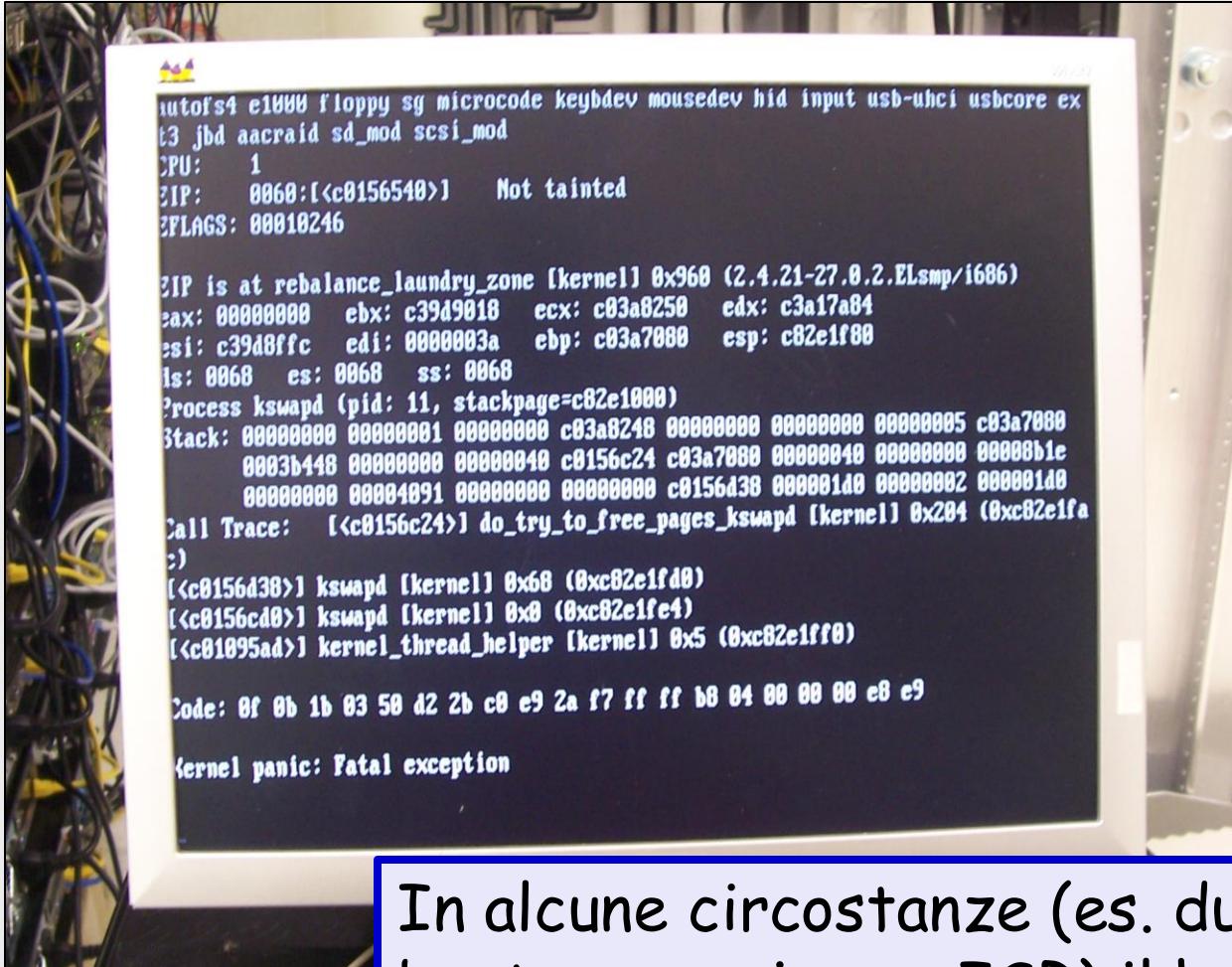


Avvertenze per la programmazione kernel-space

- A differenza dei processi, un bug nella gestione della memoria nel kernel (es. indirizzo invalido) **rende instabile l'intero sistema**
- Un “**oops**” è un messaggio diagnostico prodotto a seguito di eccezioni occorse in kernel-mode
 - Il kernel prova ad uccidere il processo corrente e a continuare l'esecuzione



Kernel panic



In alcune circostanze (es. durante il boot oppure in una ISR) il kernel si ferma (stato di **kernel panic**)



Kernel panic

[3448075.307991] ? system call +0x6B/0x6D
[3448075.307991] ? system call +0x0/0x6D
[3448075.307991]
[3448075.307991]
[3448075.307991] Code: 30 fa 5B 80 4C 39 2c 88
75 04 0f 0b eh fe 48 c7 cb 48 fa 5B 00 ch
1f 65 48 8b 04 25 10 00 00 00 66 f7 80 44
e0 f1 // 00 f1 25 04 <0f> 0b eh fe 48 c7
c0 30 fa 88 80 48 e8 1c 08 48 83 3b 00
74 04
[3448015.307991] R.I.P. [~~fffff8037/c7c~~] xen_spin_
wait +0x90/0x139
[3448015.307991] RSP <~~fffff80595e28~~>
[3448015.307991] --- [end trace 604/bc4ae1a8e]
[3448015.307991] Kernel panic - not syncing
Aiee, killing interrupt handler!





Avvertenze per la programmazione kernel-space

- Il kernel **non può utilizzare le funzioni di libreria C**
- Si utilizzano invece dei sostituti:
 - **Logging:** printk()
 - **Allocazione memoria:** kmem_cache_alloc(), kmalloc()
 - **Sincronizzazione:** spin_lock()/unlock(), wake_up_interruptible()
 - ...



Avvertenze per la programmazione kernel-space

- L'aritmetica **floating point** non è di semplice utilizzo (per via della gestione delle eccezioni)
- Gli **stack del kernel** sono limitati (1 o 2 pagine)
- Occorre gestire la **sincronizzazione** con altre parti del kernel (system call, ISR, kernel threads, ...) che accedono a dati comuni (ad esempio il PCB)



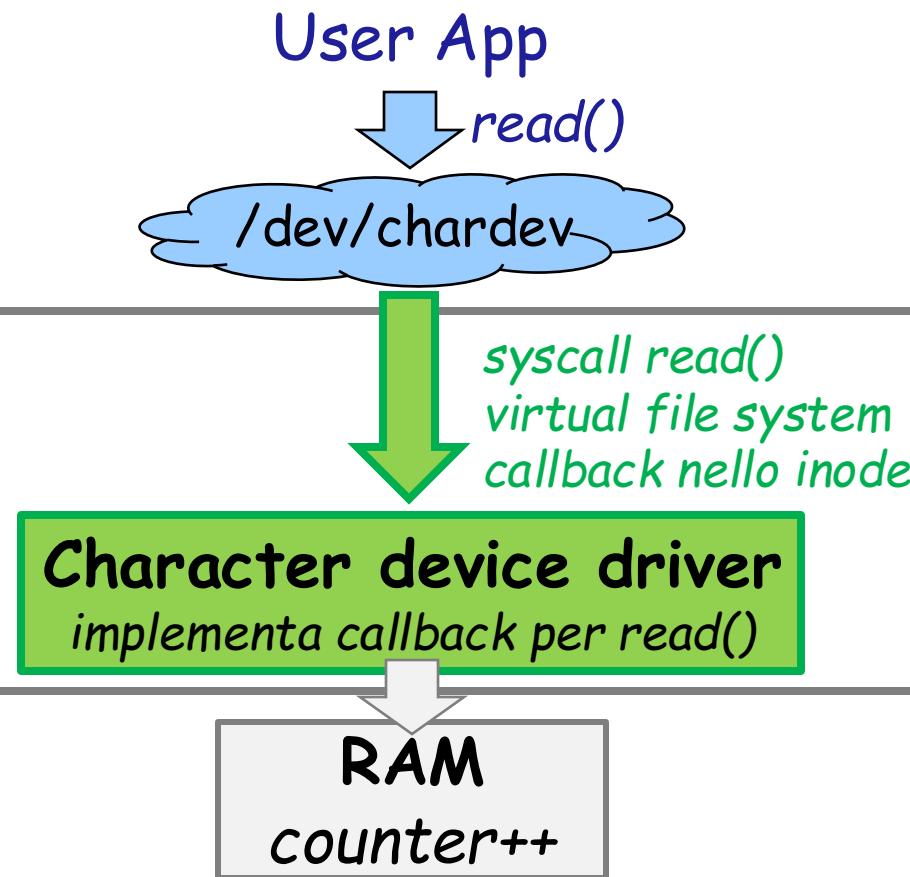
Un device driver fittizio

- Esempio: device driver per **dispositivo a caratteri** (fittizio)

User space

Kernel space

Hardware





Un device driver fittizio

- Un processo può leggere/scrivere il file virtuale **/dev/chardev**
- Il SO richiama le funzioni del device driver
 - Lettura (system call **open + read**):
il device driver **copia una stringa** con il numero di volte che il file è stato acceduto
 - Scrittura (system call **write**):
ignorate, scrive un log e ritorna un errore al processo chiamante



Esempio chardev

```
# insmod chardev.ko
# dmesg
...
I was assigned major number 249

# mknod /dev/chardev c 249 0

# cat /dev/chardev
I already told you 0 times Hello world!
# cat /dev/chardev
I already told you 1 times Hello world!
...
```



Esempio chardev

```
# insmod chardev.ko
```

```
# dmesg
```

```
...
```

```
I was assigned major number 249
```

```
# mknod /dev/chardev c 249 0
```

```
# cat /dev/chardev
```

```
I already told you
```

```
# cat /dev/chardev
```

```
I already told you 1 times Hello world!
```

```
...
```

Il comando **mknod** crea un **device file**, e lo abbina al modulo appena caricato in base ad un "**major number**"



Major e minor number

- Il **major number** è un numero identificativo del driver che deve gestire un device file
 - Il major number può essere scelto dallo sviluppatore, oppure **scelto dinamicamente** dal kernel
- Il **minor number** è un numero secondario, nel caso un driver gestisca più device file



Major e minor number

- Il **major number** essenzialmente identifica una classe di dispositivi
 - es. i driver SCSI avranno lo stesso major number
 - sono documentati nei sorgenti del kernel Linux (*Documentation/admin-guide/devices.txt*)

```
$ ls -l /dev

...
brw-rw---- 1 root disk      8,      0 dic 10 01:47 sda
brw-rw---- 1 root disk      8,      1 dic 10 01:47 sda1

...
crw--w---- 1 root tty       4,      0 dic 10 01:47 tty0
```

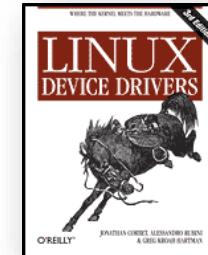


Registrazione del driver

- Il modulo del device driver deve
 - registrarsi come tale presso il kernel
 - registrare le funzioni del modulo atte a gestire le operazioni sul device file (apertura, lettura, scrittura, ...)

```
int register_chrdev( unsigned int major,
                      const char *name,
                      struct file_operations *fops );
```

- Esistono funzioni analoghe per altre **classi di dispositivi** (es., USB, SATA, Ethernet, ...)



Linux Device Drivers, 3rd Ed.
<https://lwn.net/Kernel/LDD3/>



Operazioni su device a caratteri

```
struct file_operations {
    struct module *owner;
    ...
    int (*open) (struct inode *, struct file *);
    ssize_t (*read) ( struct file *, char __user *,
                      size_t, loff_t * );
    ssize_t (*write) ( struct file *, const char __user *,
                      size_t, loff_t * );
    ...
};
```



Struttura del modulo

```
int init_module(void);
void cleanup_module(void);
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);

static int Major;           /* Major number assigned to our device driver */
static int Device_Open = 0;  /* Is device open?
                             * Used to prevent multiple access to device */
static char msg[BUF_LEN];   /* The msg the device will give when asked */
static char *msg_Ptr;

static struct file_operations fops = {
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release
};
```



Inizializzazione del modulo

```
int init_module(void)
{
    Major = register_chrdev(0, DEVICE_NAME, &fops);

    if (Major < 0) {
        printk(KERN_ALERT "Registering char device failed with %d\n", Major);
        return Major;
    }

    printk(KERN_INFO "I was assigned major number %d. To talk to\n", Major);
    printk(KERN_INFO "the driver, create a dev file with\n");
    printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
    printk(KERN_INFO "Try various minor numbers. Try to cat and echo to\n");
    printk(KERN_INFO "the device file.\n");
    printk(KERN_INFO "Remove the device file and module when done.\n");

    return SUCCESS;
}
```



Operazione di apertura

```
/*
 * Called when a process tries to open the device file, like
 * "cat /dev/mycharfile"
 */

static int device_open(struct inode *inode, struct file *file)
{
    static int counter = 0;

    if (Device_Open)
        return -EBUSY;

    Device_Open++;
    sprintf(msg, "I already told you %d times Hello world!\n", counter++);
    msg_Ptr = msg;

    try_module_get(THIS_MODULE);

    return SUCCESS;
}
```



Operazione di lettura

```
static ssize_t device_read(
    struct file *filp,          /* see include/linux/fs.h */
    char *buffer,               /* buffer to fill with data */
    size_t length,              /* length of the buffer */
    loff_t * offset ) {

    int bytes_read = 0; // Number of bytes actually written to the buffer

    if (*msg_Ptr == 0) // If we're at the end of the message,
        return 0; // return 0 signifying end of file

    while (length && *msg_Ptr) {
        /* The buffer is in the user data segment, not the kernel segment, so
         * "*" assignment won't work. We have to use put_user which copies
         * data from the kernel data segment to the user data segment.*/
        put_user(* (msg_Ptr++) , buffer++);
        length--;
        bytes_read++;
    }
    return bytes_read;
}
```



Chiamate di sistema



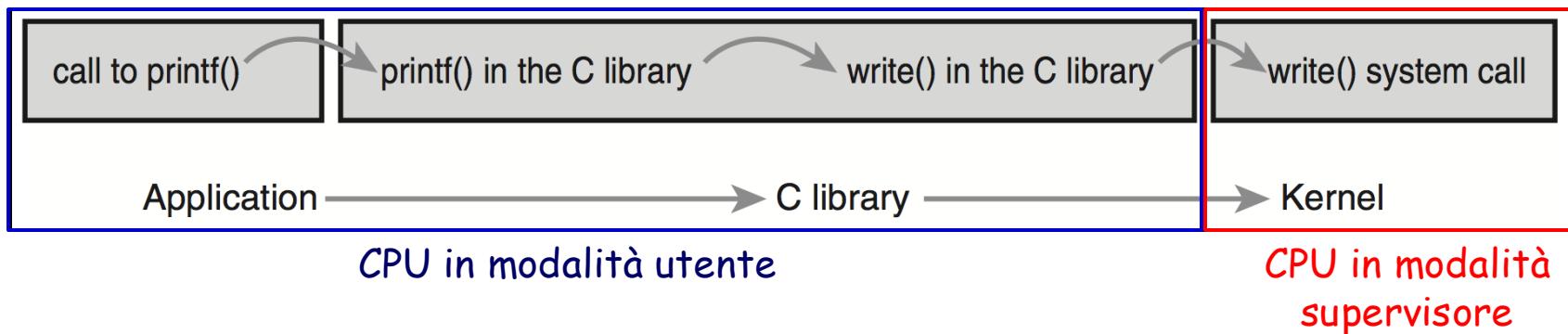
Chiamate di sistema

- Le chiamate di sistema sono l'**interfaccia** tra i processi e il SO
- Passaggio della CPU dalla **modalità utente** (esecuzione dei processi) alla **modalità kernel** (esecuzione del codice del kernel)
- Linux fornisce oltre 300 chiamate di sistema (in parte basate sullo standard **IEEE POSIX**)



La libreria C

- Le chiamate di sistema sono incapsulate ("wrapped") dalla **libreria C**





Esempio di chiamata di sistema

- Le funzioni nel kernel che implementano syscall hanno il prefisso "**sys_**"

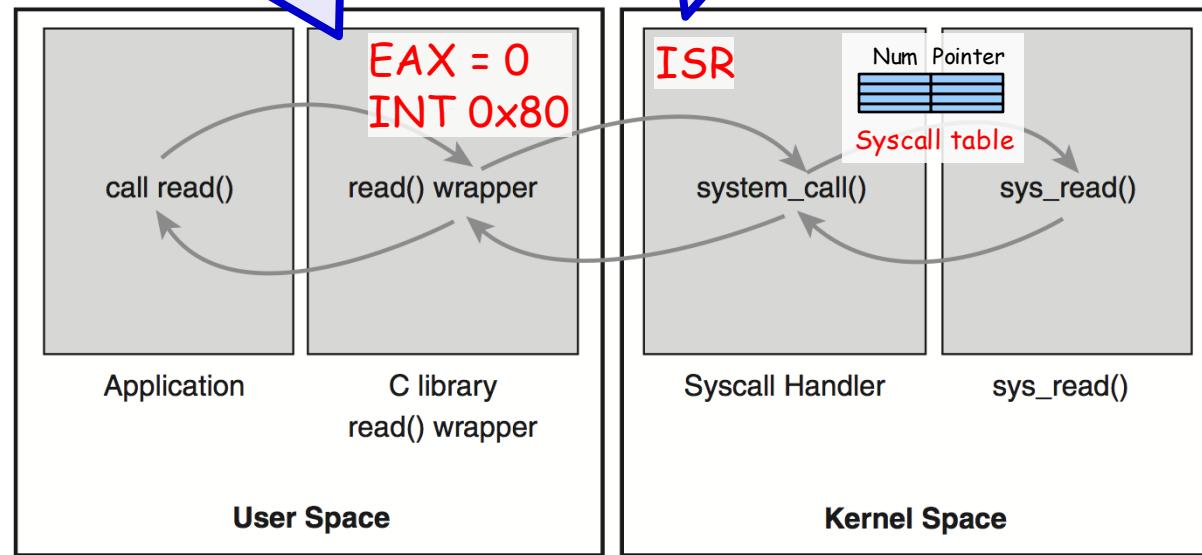
```
asmlinkage long sys_getpid(void) {  
    return current->tgid;  
}
```

- "**current**" è un puntatore globale al PCB del processo in esecuzione
- In Linux, il PID di un processo è anche detto "**Thread Group ID**" (TGID)



Invocazione di una chiamata di sistema

Scrive il **syscall number** su un registro (es. 0 per read()), esegue **supervisor call**



La ISR (**syscall handler**) cerca il syscall number nella **syscall table**, e trova il puntatore alla funzione da eseguire (es. **sys_read()**)



System call handler

Identificativo
della system call

```
_visible void do_syscall_64(unsigned long nr, struct pt_regs *regs)
{
    struct thread_info *ti;

    enter_from_user_mode();
    local_irq_enable();
    ti = current_thread_info();

    if (likely(nr < NR_syscalls)) {
        nr = array_index_nospec(nr, NR_syscalls);
        regs->ax = sys_call_table[nr](regs);
    }

    syscall_return_slowpath(regs);
}
```

Questo array contiene
dei puntatori a funzione
alle system call



Application Binary Interface

- La **Application Binary Interface (ABI)** è un insieme di convenzioni tra programmi utente e SO
- Es., in x86, il passaggio di parametri è via:
 - System Call number: **EAX**
 - Parametri di ingresso: **EBX, ECX, EDX, ESI, EDI, EBP**
 - Parametro di uscita: **EAX**
- Il kernel ne garantisce la stabilità ad ogni versione



Implementazione di una system call

1. Inserire la syscall in un **nuovo file** all'interno dei sorgenti

es.: <sorgenti>/kernel/mysyscall.c

2. Modificare il **Makefile** nella cartella del nuovo file

es. (in <sorgenti>/kernel/Makefile): obj-y += mysyscall.o

3. Aggiungere una nuova riga alla **system call table**

es. (per x86): <sorgenti>/arch/x86/entry/syscalls/syscall_64.tbl

4. Ricompilare il kernel



Esempio di system call (lato kernel)

```
SYSCALL_DEFINEn(mysyscall, int, PARAMETRO1, char *, PARAMETRO2, ....)
{
    // .... implementazione della system call ....
    return <VALORE LONG>;
}
```

Utilizzare la macro **SYSCALL_DEFINEn** (invece della sintassi classica del linguaggio C) per dichiarare la chiamata di sistema

Serve per esportare la definizione ad altri tool esterni



Esempio di system call (lato kernel)

```
SYSCALL_DEFINE3(mysyscall, pid_t, pid, char __user *, buffer, size_t, buf_size)
{
    // .... implementazione della system call ....
    return <VALORE LONG>;
}
```

Esempio:
System call per **cercare un processo con il PID indicato** come parametro
Ritorna una stringa con il nome del processo



Esempio di system call (lato kernel)

```
SYSCALL_DEFINE3(mysyscall, pid_t, pid, char __user *, buffer, size_t, buf_size)
{
    struct task_struct * mytask;
    char nome_processo[TASK_COMM_LEN];
    size_t dim;

    mytask = find_task_by_vpid(pid);                                // ricerca del PCB, in base al PID

    if(!mytask) {
        printk(KERN_DEBUG "Processo non trovato: %d\n", pid);      // ricerca fallita
        return -ESRCH;
    }

    __get_task_comm(nome_processo, TASK_COMM_LEN, mytask);          // legge il nome del processo
    printk(KERN_DEBUG "Processo trovato: %d = %s\n", pid, nome_processo);

    dim = (buf_size < TASK_COMM_LEN) ? buf_size : TASK_COMM_LEN;

    if( copy_to_user(buffer, nome_processo, dim) ) {

        printk(KERN_DEBUG "Errore di accesso alla memoria: %p\n", buffer); // copy_to_user() fallita
        return -EFAULT;
    }

    return 0;
}
```

<sorgenti>/kernel/mysyscall.c



Esempio di system call (lato kernel)

<sorgenti>/kernel/Makefile

The screenshot shows a terminal window with a dark theme. The title bar says "Makefile" and "~/linux-5.4/kernel". The file content is a Makefile for the kernel. Line 137 is highlighted in red, indicating it is the target of modification. The code includes various shell commands and variables, such as `\${obj}` and `\${call if_changed,gzip}`.

```
126 $(call if_changed,gzip)
127
128 ${obj}/kheaders.o: ${obj}/kheaders_data.tar.xz
129
130 quiet_cmd_genikh = CHK      ${obj}/kheaders_data.tar.xz
131     cmd_genikh = $(BASH) $(srctree)/kernel/gen_kheaders.sh $@
132 ${obj}/kheaders_data.tar.xz: FORCE
133     $(call cmd,genikh)
134
135 clean_files := kheaders_data.tar.xz kheaders.md5
136
137 obj-y += mysyscall.o
138
```

Makefile Larg. tab.: 4 Rg 137, Col 21 INS

- Copiamo il file "mysyscall.c" nei sorgenti del kernel, nella sottocartella "**kernel**"
- Modifichiamo "**kernel/Makefile**"



System call table

System Call Table (Linux 5.4)

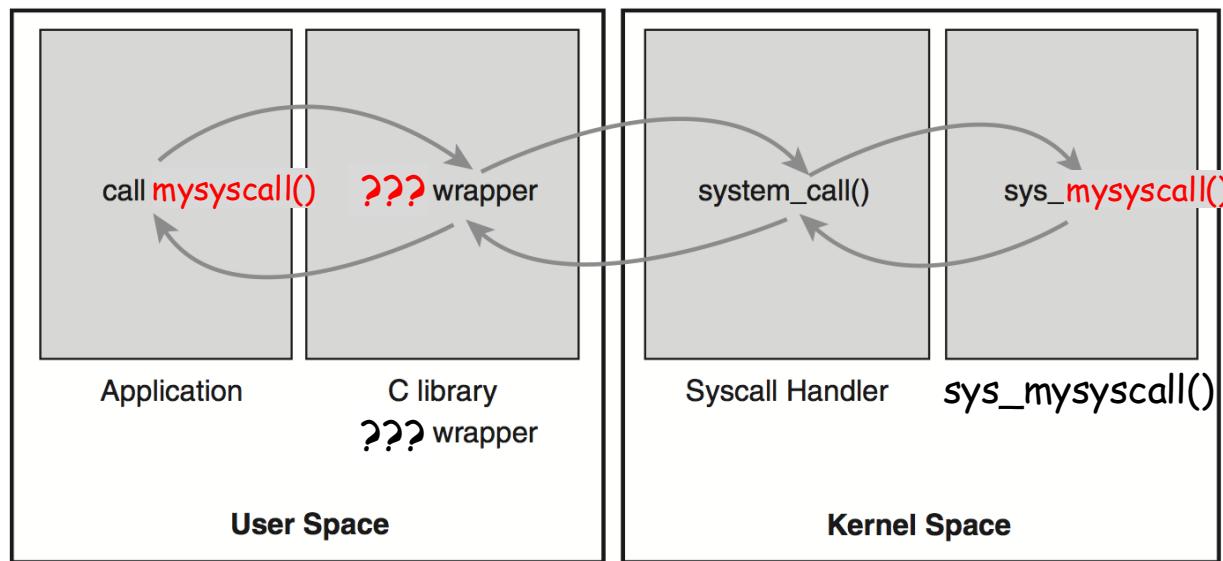
<sorgenti>/arch/x86/entry/syscalls/syscall_64.tbl

```
#  
# 64-bit system call numbers and entry vectors  
  
#  
# The format is:  
# <number> <abi> <name> <entry point>  
  
#  
# The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls  
  
#  
# The abi is "common", "64" or "x32" for this file.  
  
#  
0      common  read           __x64_sys_read  
1      common  write          __x64_sys_write  
2      common  open           __x64_sys_open  
...  
  
332    common  statx         __x64_sys_statx  
333    common  io_pgetevents __x64_sys_io_pgetevents  
334    common  rseq           __x64_sys_rseq  
  
# La nuova system call  
335    common  mysyscall    __x64_sys_mysyscall  
...
```



Accedere alla system call da user-space

- Per la **nuova system call**, non esiste alcuna funzione nelle librerie di sistema!
- Occorre scrivere una funzione “**wrapper**” apposita per la nuova system call





Accedere alla system call da user-space

- La libreria C fornisce un **wrapper generico** denominato "**syscall()**"

```
#define __NR_read 0

ssize_t my_read(int fd, void *buf, size_t count)
{
    return syscall(__NR_open, fd, buf, count);
```



Esempio di system call (lato user)

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/syscall.h>

#define __NR_mysyscall 335      // NOTA: Utilizzare lo stesso valore usato in syscall_64.tbl

long mysyscall(pid_t pid, char * buffer, size_t buf_size) {

    return syscall(__NR_mysyscall, pid, buffer, buf_size);
}

int main() {

    long err;
    char nomeprocesso[16];

    err = mysyscall( getpid(), nomeprocesso, 16 );

    if(err) {
        perror("mysyscall fallita");
        return -1;
    }

    printf("Risultato mysyscall: %s\n", nomeprocesso);

    return 0;
}
```



Avvertenze

- Le chiamate di sistema devono **verificare la validità dei parametri di ingresso** dal processo chiamante
 - Ad esempio, se si passa in ingresso un PID, occorre accertarsi che il PID esista veramente nel sistema!
 - Altrimenti, è a rischio la **sicurezza** del kernel
- In particolare, occorre garantire che i **puntatori di memoria** passati siano validi
 - Prevenire che la system call legga/scriva aree di memoria non autorizzate
 - Utilizzare **copy_from_user()** e **copy_to_user()** per garantire la copia sicura dei dati da/verso il processo



Un esempio di modulo kernel
malevolo (rootkit)



Linux kernel rootkit

<https://github.com/m0nad/Diamorphine>

Diamorphine

Diamorphine is a LKM rootkit for Linux Kernels 2.6.x/3.x/4.x/5.x/6.x (x86/x86_64 and ARM64)

Features

- When loaded, the module starts invisible;
- Hide/unhide any process by sending a signal 31;
- Sending a signal 63(to any pid) makes the module become (in)visible;
- Sending a signal 64(to any pid) makes the given user become root;
- Files or directories starting with the MAGIC_PREFIX become invisible;
- Source: <https://github.com/m0nad/Diamorphine>

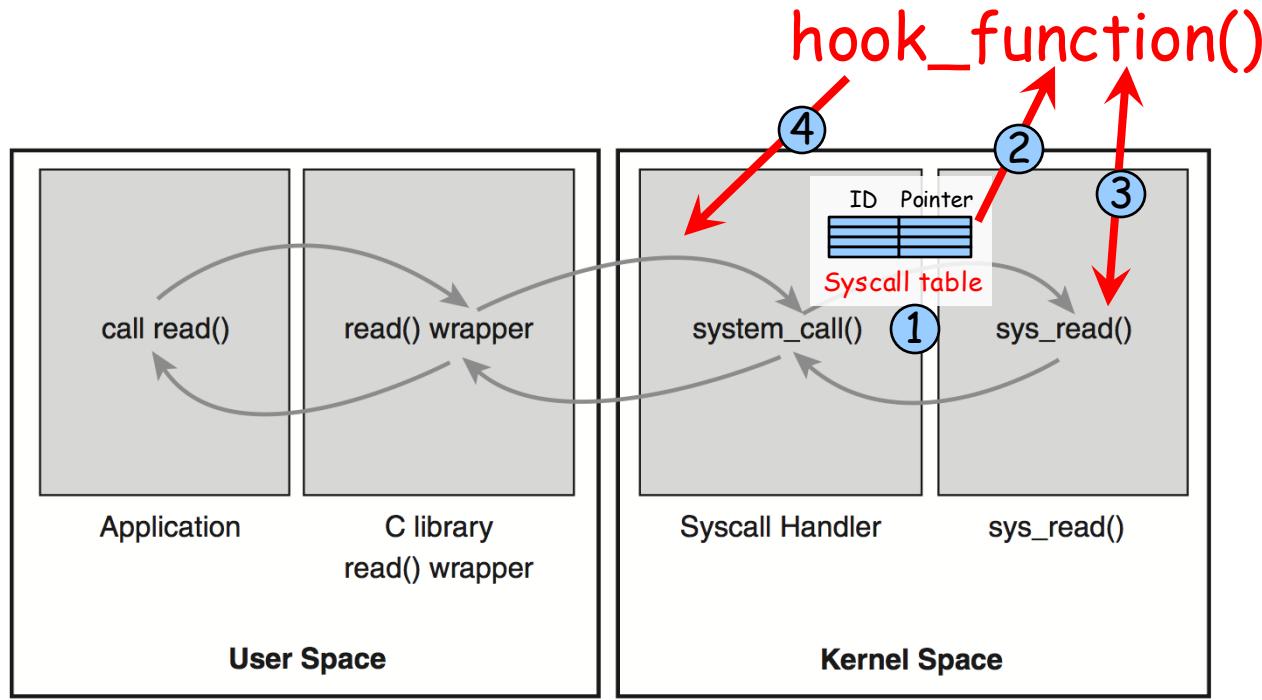


Linux kernel rootkit

- Nascondere processi
 - Nascondere file e cartelle
 - Nascondere connessioni
 - Otttenere una shell di root
 - Backdoor
-
- **Attenzione! Può danneggiare il sistema,
procedere con cautela**



Linux kernel rootkit



Tecnica dello **hooking**:

1. Nella syscall table, si inserisce un **puntatore ad una funzione malevola**
2. Quando un processo chiama la syscall vittima (es. `read`), il kernel segue la tabella e chiama la funzione malevola
3. La funzione malevola chiama la system call
4. La funzione malevola **modifica i risultati della system call vittima** (es. omette un file o un processo, per nasconderlo)



DRM rootkit

The screenshot shows a Microsoft Internet Explorer window displaying an Amazon.com product page. The product is 'Get Right with the Man [CONTENT/COPY-PROTECTED CD]' by Van Zant. The title is highlighted with a yellow box. The page includes a 'MUSIC INFORMATION' sidebar with links to buying info, samples, reviews, and rating options. The main content area shows the album cover, a price of \$14.99, and a 'List Price: \$18.98'. A 'Super Saver Shipping' offer is available. The 'READY TO BUY?' section offers to add it to the shopping cart or sign in for one-click ordering. Other buying choices include 105 used & new items from \$6.79 and a club price of \$9.99.

Sony, Rootkits and Digital Rights Management Gone Too Far

<https://techcommunity.microsoft.com/t5/windows-blog-archive/sony-rootkits-and-digital-rights-management-gone-too-far/ba-p/723442>