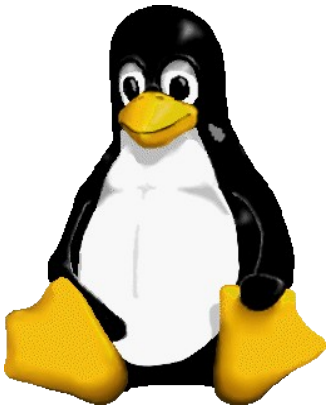
Il kernel Linux



Andrea Righi <andrea@betterlinux.com>

Overview

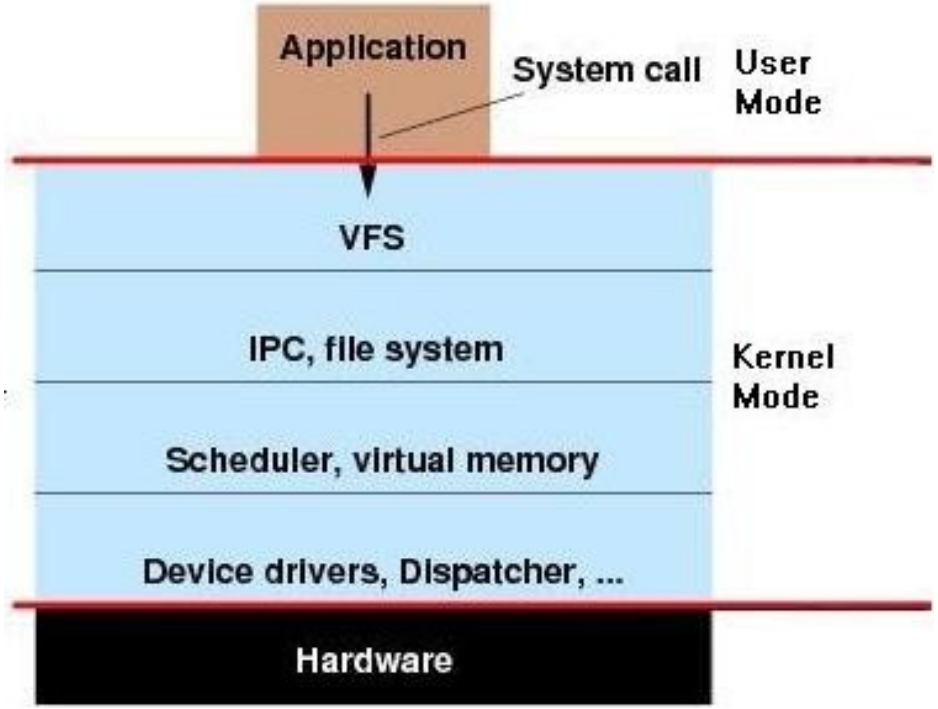
Andrea Righi <andrea@betterlinux.com>

Cos'è il kernel?

- Il kernel rappresenta la componente software che ha il compito di amministrare le risorse presenti sul nostro sistema, rendendole disponibili alle applicazioni dell'utente
- Fornisce un livello di astrazione per l'accesso alle risorse hardware

Kernel monolitico

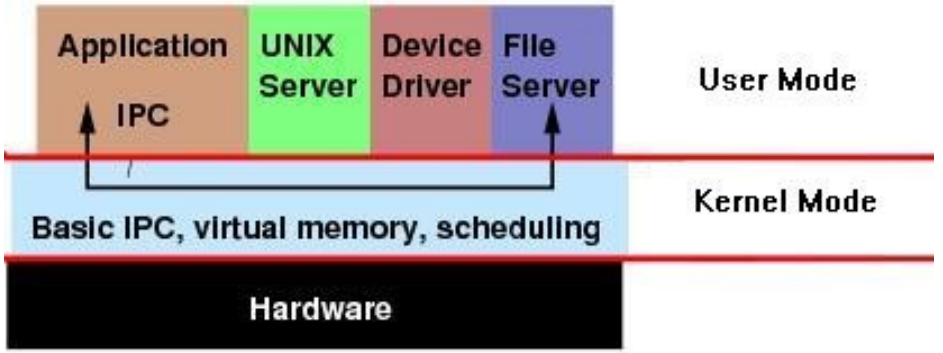
- Il kernel risiede nella stessa area di memoria di ciascun processo utente
- Chiamate di sistema simili a normali chiamate a funzione



Andrea Righi <andrea@betterlinux.com>

Microkernel

- Il kernel mette a disposizione solo un layer minimale di accesso all'hardware e meccanismi di comunicazione tra processi (IPC)
- Chiamate di sistema realizzate tramite IPC

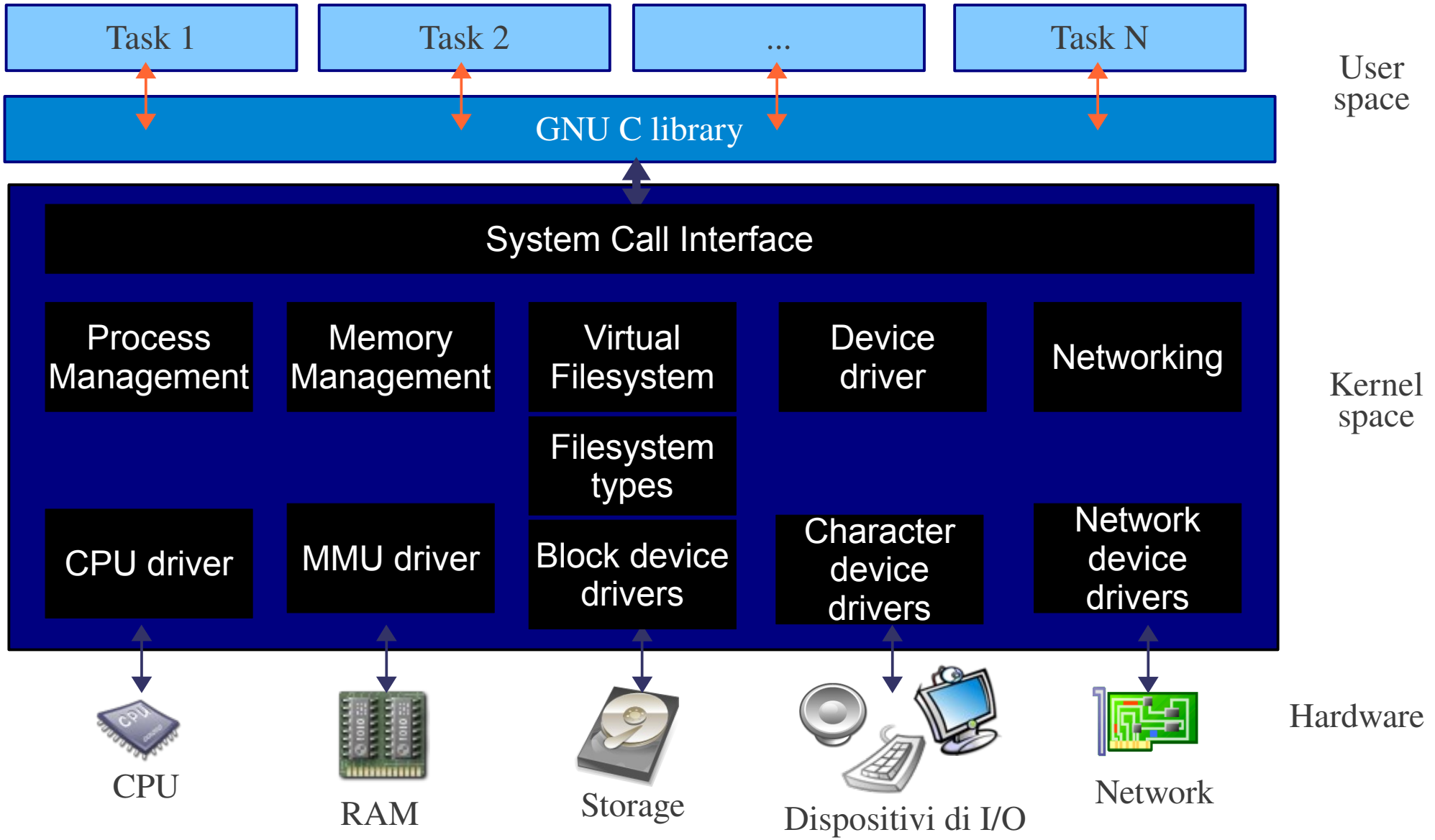


Andrea Righi <andrea@betterlinux.com>

Kernel Linux: monolitico (modulare)

- «*I still maintain the point that designing a monolithic kernel in 1991 is a fundamental error. Be thankful you are not my student. You would not get a high grade for such a design*»
  - Andrew S. Tanenbaum to Linus Torvalds

Linux kernel architecture



Andrea Righi <andrea@betterlinux.com>

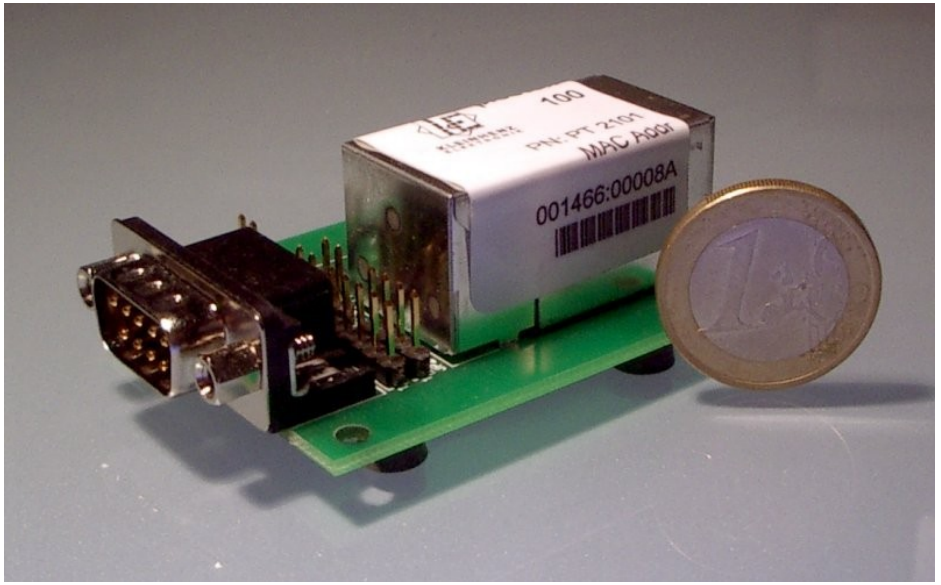
Linux kernel key features

- Portabilità
- Scalabilità
- Compatibile con lo standard UNIX
- Sicurezza, Stabilità, Affidabilità (processo di review del codice molto severo)
- Modularità (possibilità di caricare/scaricare funzionalità a runtime)
- Disponibilità del codice (licenza GPLv2)

Portabilità & Scalabilità

Andrea Righi <andrea@betterlinux.com>

Picotux



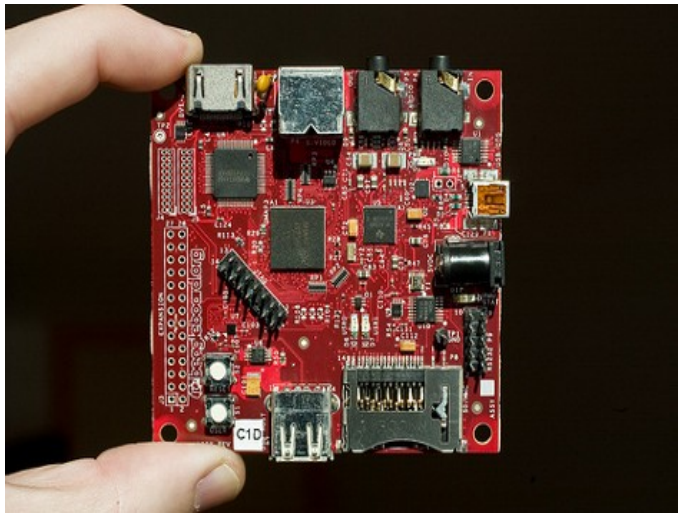
- Il più piccolo computer al mondo su cui gira Linux
- 55 MHz 32-bit ARM7 Netsilicon NS7520
- 2MB flash
- 8MB SDRAM
- Kernel: Linux
- Consumo: ~800 mW

Andrea Righi <andrea@betterlinux.com>

'K' computer

- Il computer più potente al mondo
- 68.544 2GHz 8-core SPARC64 processors
- Speed: >8 petaflops
- Ranking: 1st (Top500, June 2011)
- Kernel: Linux
- Consumo: 9.89MW

Andrea Righi <andrea@betterlinux.com>



Andrea Righi <andrea@betterlinux.com>

Open source

Andrea Righi <andrea@betterlinux.com>

GPLv2 (in breve)

- Diritti:
  - L'utente può eseguire il software per qualsiasi scopo
  - L'utente può aprire e studiare il funzionamento del programma e, se occorre, modificarlo
  - L'utente può fare copie del software e distribuirlo liberamente
- Doveri:
  - Le modifiche al codice devono anche essere essere rilasciate con licenza GPL *(v2 o superiore...)*

Statistiche sul codice sorgente (3.1.0-rc9)

```
Totals grouped by language (dominant language first):
ansic:      9550623 (96.53%)
asm:         271689 (2.75%)
xml:          40082 (0.41%)
perl:         14791 (0.15%)
sh:            4205 (0.04%)
cpp:           3527 (0.04%)
yacc:          2978 (0.03%)
python:        2784 (0.03%)
lex:           1718 (0.02%)
awk:            708 (0.01%)
pascal:         231 (0.00%)
lisp:           218 (0.00%)
sed:             30 (0.00%)


Total Physical Source Lines of Code (SLOC) = 9,893,584
```

Generate con SLOCCount: http://www.dwheeler.com/sloccount/

Kernel space o User space?

Andrea Righi <andrea@betterlinux.com>

Kernel space

- C e assembly (no C++)
- Tool di debugging specifici (kgdb, UML, ...)
- Un bug può bloccare l'intero sistema
- La memoria kernel non può essere swappata
- Kernel stack size limitato (4K / 8K)
- Uso di floating point non permesso
- NO librerie userspace
- Il codice kernel deve essere portabile (dai supercomputer ai piccoli dispositivi embedded)
- I moduli closed-source “sporcano” il kernel

User space

- Ampia disponibiltà di strumenti di debugging (gdb)
- L'accesso diretto alla memoria fisica non è possibile
- Non è possibile interagire direttamente con le routine di interrupt
- I tempi di risposta sono più lenti (context switch, scheduler, syscall, ...)

Come scrivere un modulo del kernel (hello, world!)

Andrea Righi <andrea@betterlinux.com>

Makefile

```
ifndef KERNELRELEASE
PWD := $(shell pwd)
all:
    $(MAKE) -C /lib/modules/`uname -r`/build SUBDIRS=$(PWD) modules
clean:
    rm -f *.o *.ko *.mod.* .*.cmd Module.symvers modules.order
    rm -rf .tmp_versions
else
    obj-m := hello.o
endif
```


Andrea Righi <andrea@betterlinux.com>

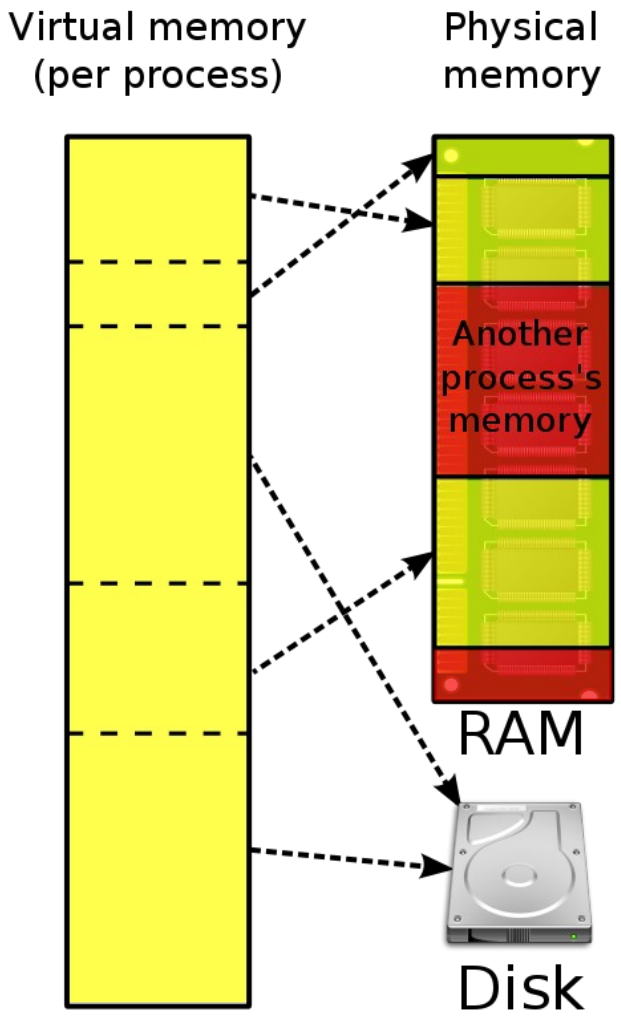
hello.c

```
#include <linux/init.h>
#include <linux/module.h>

static int __init hello_init(void)
{
    printk(KERN_INFO "hello, world!\n");
    return 0;
}

static void __exit hello_exit(void)
{
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Andrea Righi <andrea@betterlinux.com>");
MODULE_DESCRIPTION("LinuxDay hello world example");
```

Scrittura del driver “rawmem”

Andrea Righi <andrea@betterlinux.com>

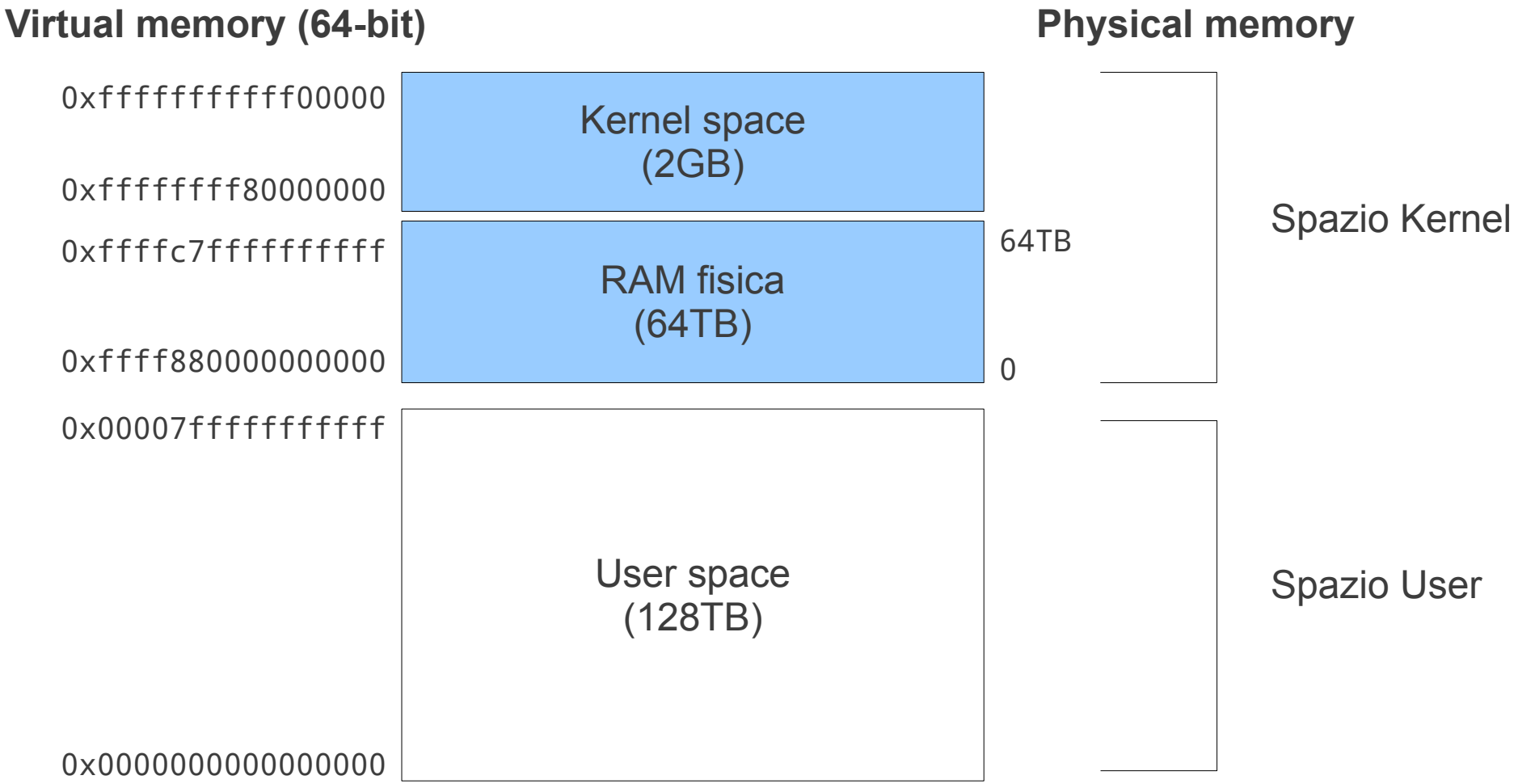
Memoria virtuale



- Virtualizza lo spazio di indirizzamento dei processi
- Ogni processo “vede” virtualmente più memoria della memoria fisica presente sul sistema

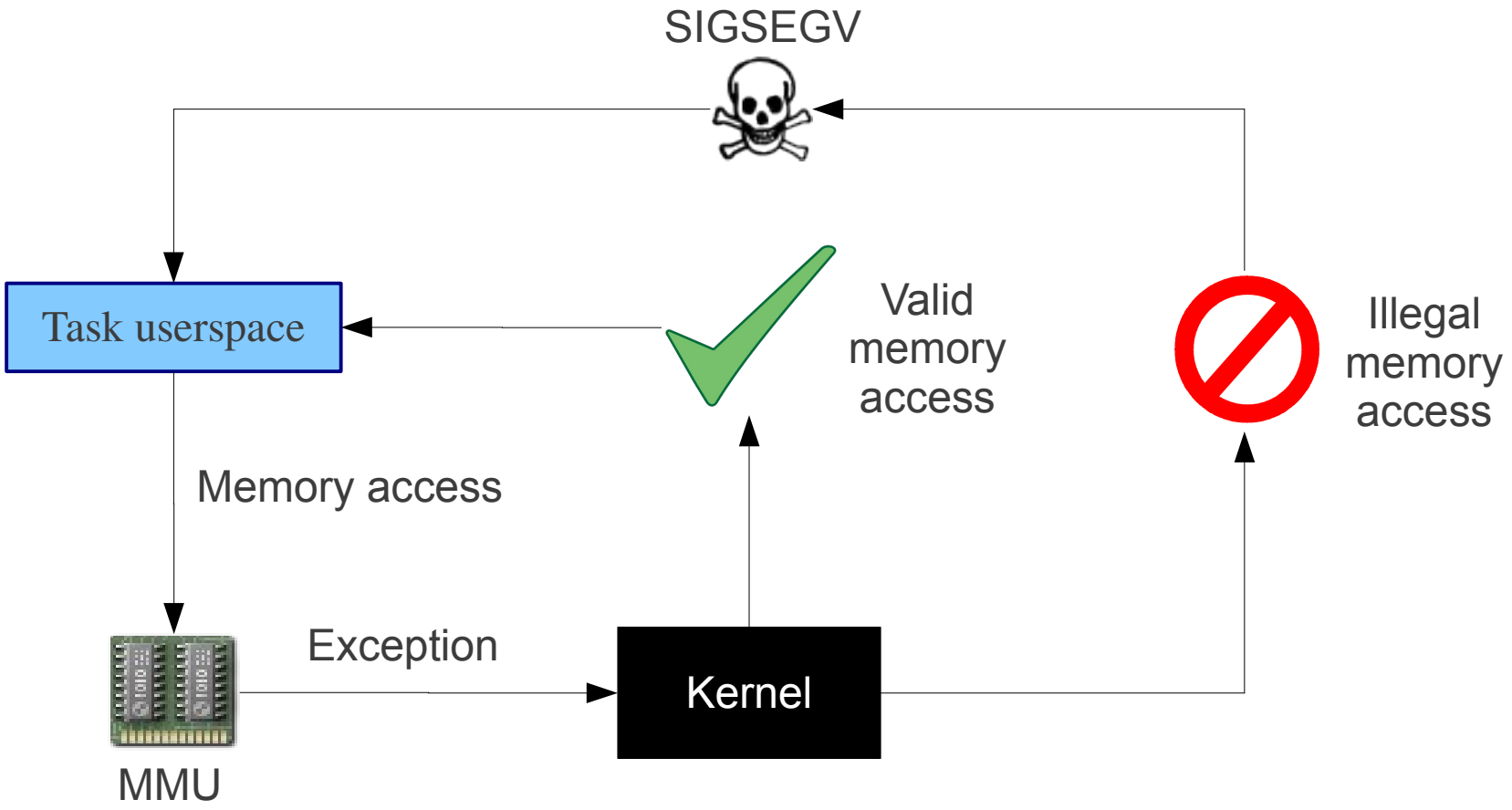
Andrea Righi <andrea@betterlinux.com>

Memoria virtuale: overview
(x86_64)



Andrea Righi <andrea@betterlinux.com>

Accesso alla memoria



Andrea Righi <andrea@betterlinux.com>

Character device files

- Accesso ad una risorsa tramite flusso di bytes
- Esempi:
  - tastiera, mouse, terminale, audio, video, ...

```
$ ls -l /dev/ | grep ^c
crw-rw----  1 root video      10, 175 2011-10-16 18:07 agpgart
crw-------  1 root root         5,   1 2011-10-16 18:07 console
crw-rw----  1 root video      29,   0 2011-10-16 18:07 fb0
crw-rw-rw-  1 root root         1,   7 2011-10-16 18:07 full
crw-rw-rw-  1 root fuse        10, 229 2011-10-16 18:07 fuse
crw-------  1 root root      251,   0 2011-10-16 18:07 hidraw0
crw-------  1 root root      251,   1 2011-10-16 18:07 hidraw1
crw-------  1 root root      251,   2 2011-10-16 18:07 hidraw2
crw-------  1 root root       10, 228 2011-10-16 18:07 hpet
crw-------  1 root root        1,  11 2011-10-16 18:07 kmsg
...
```

Block device files

- Accesso ad una risorsa tramite blocchi di dimensione fissa
- Esempi:
  - hard disk, partizioni disco, ram disk, loop device, ...

```
$ ls -l /dev/ | grep ^b
brw-rw----  1 root disk       7,   0 2011-10-16 18:07 loop0
brw-rw----  1 root disk       7,   1 2011-10-16 18:07 loop1
brw-rw----  1 root disk       7,   2 2011-10-16 18:07 loop2
brw-rw----  1 root disk       7,   3 2011-10-16 18:07 loop3
brw-rw----  1 root disk       7,   4 2011-10-16 18:07 loop4
brw-rw----  1 root disk       7,   5 2011-10-16 18:07 loop5
brw-rw----  1 root disk       7,   6 2011-10-16 18:07 loop6
brw-rw----  1 root disk       7,   7 2011-10-16 18:07 loop7
brw-rw----  1 root disk       1,   0 2011-10-16 18:07 ram0
brw-rw----  1 root disk       1,   1 2011-10-16 18:07 ram1
brw-rw----  1 root disk       1,  10 2011-10-16 18:07 ram10
brw-rw----  1 root disk       1,  11 2011-10-16 18:07 ram11
brw-rw----  1 root disk       1,  12 2011-10-16 18:07 ram12
brw-rw----  1 root disk       1,  13 2011-10-16 18:07 ram13
brw-rw----  1 root disk       1,  14 2011-10-16 18:07 ram14
brw-rw----  1 root disk       1,  15 2011-10-16 18:07 ram15
brw-rw----  1 root disk       1,   2 2011-10-16 18:07 ram2
brw-rw----  1 root disk       1,   3 2011-10-16 18:07 ram3
brw-rw----  1 root disk       1,   4 2011-10-16 18:07 ram4
brw-rw----  1 root disk       1,   5 2011-10-16 18:07 ram5
brw-rw----  1 root disk       1,   6 2011-10-16 18:07 ram6
brw-rw----  1 root disk       1,   7 2011-10-16 18:07 ram7
brw-rw----  1 root disk       1,   8 2011-10-16 18:07 ram8
brw-rw----  1 root disk       1,   9 2011-10-16 18:07 ram9
brw-rw----  1 root disk       8,   0 2011-10-16 18:07 sda
brw-rw----  1 root disk       8,   1 2011-10-16 18:07 sda1
brw-rw----  1 root disk       8,   2 2011-10-16 18:07 sda2
brw-rw----  1 root disk       8,   5 2011-10-16 18:07 sda5
```


Andrea Righi <andrea@betterlinux.com>

Major/minor numbers

- Usati per legare un device driver a un device file
  - Major: identifica il device driver
  - Minor: identifica un particolare dispositivo che fa uso del device driver
- Esempi:
  - /dev/sda = block (8,1)
    - major: 8 (SCSI driver)
    - minor: 0 (primo disco rilevato)
  - /dev/zero = character (1,5)
  - /dev/null = character (1,3)

Obiettivo

- Fornire allo userspace un'interfaccia di accesso a ***qualsiasi*** area di memoria fisica

Andrea Righi <andrea@betterlinux.com>

Implementazione

- Registrazione di un nuovo character device (/dev/rawmem)
- Implementazione della system call mmap() fatta su /dev/rawmem:
  - mmap(X) = Y
    - X: indirizzo fisico
    - Y: indirizzo virtuale mappato sulla pagina X

Codice sorgente (core)

```
/* Implementazione della nostra mmap() su /dev/rawmem */
static int rawmem_mmap(struct file *file, struct vm_area_struct *vma)
{
	return remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff,
			vma->vm_end - vma->vm_start, vma->vm_page_prot);
}

/* Operazioni su file di /dev/rawmem */
static struct file_operations rawmem_fops = {
	.mmap = rawmem_mmap,
};
```


Andrea Righi <andrea@betterlinux.com>

Conclusioni

- Linux è un kernel monolitico che ha un gestore della memoria virtuale
- Abbiamo visto un semplice esempio di come interagire con questo gestore per mappare una qualsiasi pagina di memoria nello spazio utente
- Idea di progetto: /dev/rawmem potrebbe essere usato come base per la scrittura di device driver in userspace
  - Vantaggi / svantaggi...

Domande / risposte

Andrea Righi <andrea@betterlinux.com>

Andrea Righi <andrea@betterlinux.com>