# Introduzione ai Device Drivers in Linux

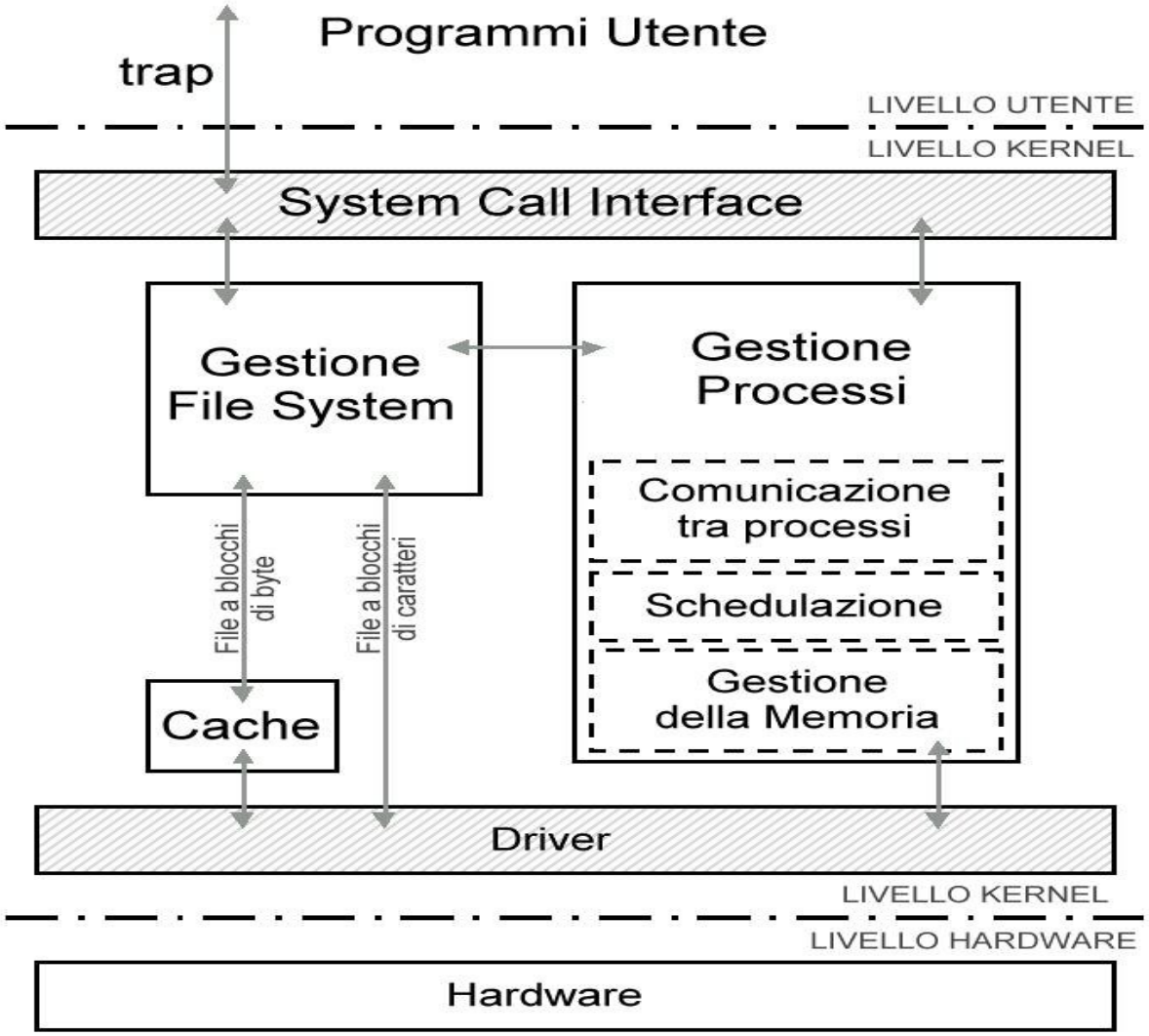
**E.Mumolo, DEEI**

**mumolo@units.it**

# *Device Drivers*

- Moduli software che gestiscono le periferiche attraverso il file system
- Tutte le periferiche sono viste come file speciali
- In Linux ogni periferica è gestita mediante un Device Driver
- Tipi di periferiche:
  - A blocchi → blocchi di dati ad accesso casuale
  - A caratteri → byte singoli o a blocchi ad accesso sequenziale

# Struttura del kernel



# File speciali di tipo Device Driver

- Memorizzati in /dev
- Esempio:

```
$ls -l /dev
   crw-------  1 root root      5,   1 2011-12-06 08:50 console
   lrwxrwxrwx  1 root root          11 2011-12-06 08:50 core -> /proc/kcore
   crw-rw----  1 root root     10,  58 2011-12-06 08:50 cpu_dma_latency
   drwxr-xr-x  6 root root         120 2011-12-06 09:49 disk
   crw-------  1 root root    108,   0 2011-12-06 08:50 ppp
   brw-rw----  1 root disk      8,   0 2011-12-06 08:50 sda
   crw-rw-rw-  1 root tty       5,   0 2011-12-06 08:50 tty
   crw-rw----  1 root root    252,   0 2011-12-06 08:50 usbmon0
```


c = a carattere; b= a blocchi

# File speciali di tipo Device Driver

- File speciali di tipo DeviceDriver possono essere creati solo da **root** con la chiamata di sistema

```
mknod(path,tipo,major,minor)
```

oppure il comando di linea

```
mknod file tipo major_number minor_number
```

- Major number: tipo di periferica
- Minor number: periferica dello stesso tipo
- Valori di major/minor sono contenuti nell'Inode
- Accesso e gestione delle periferiche mediante chiamate di accesso ai file: open, read, write, ioctl ...

# Esempio: il file /dev/dsp è il driver audio

Operazioni fondamentali:

```
void main(int argc, char **argv)
{
     ...
     int speed=11025;
     format=AFMT_S16_LE; //formato: 16 bit  L endian
     if ((audio_fd = open("/dev/dsp", O_RDWR, 0)) == -1)  exit(Errrorcode); // apri
     if (ioctl(audio_fd, SNDCTL_DSP_SETFMT, &format)==-1) exit(Errorcode); // formato
     if (ioctl(audio_fd, SNDCTL_DSP_STEREO, &stereo)==-1) exit(Errorcode); //nr canali
     if (ioctl(audio_fd, SNDCTL_DSP_SPEED, &speed)==-1)  exit(Errorcode); //campionamento
     ...
     n=write(audio_fd, &audio_buffer[nwrite], nwrite);
     ...
}
```

# Struttura di un Device Driver

- Il Device Driver è un MODULO del kernel
- Ogni driver ha associata la lista dei puntatori alle funzioni di gestione:

```
struct file_operations fops{
        int (*lseek) ( );
        int (*read) ( );
        int (*write) ( );
        int (*ioctl) ( );
        int (*open) ( );
        void (*release) ( );
        …
}
```

- La init restituisce al kernel i puntatori alle funzioni di gestione
- La init del driver registra il driver:
  - `register_chrdev(major, "nome", &fops);`

# Struttura di un Device Driver

- La registrazione del driver salva la coppia (major number, puntatore) nella Character Device table
- Al boot il SO attiva la inizializzazione di ogni dispositivo installato
- Interfaccia SO e driver:
  - Block Device Table: tabella driver per i dispositivi a blocchi
  - Character Device Table: tabella per i dispositivi a carattere
- Ogni dispositivo occupa una riga della Tabella corrispondente che indirizza alla tabella delle operazioni del driver corrispondente

# *Indirizzamento delle routine di gestione di un driver*

Il processo apre il driver: **`fd=open(/dev/tty, …)`**

↘

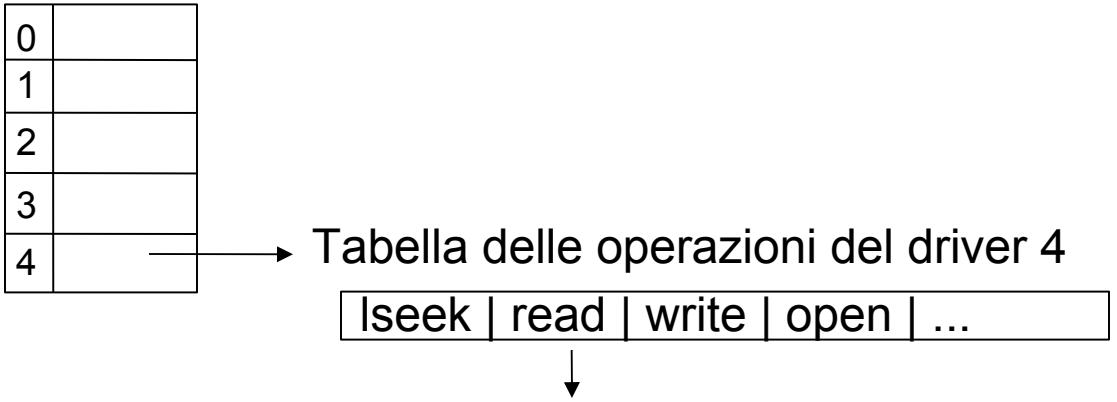
Il SO cerca l'INODE di /dev/tty

↘

dall' I-node del file speciale a caratteri → major=4, minor=1

↘

dalla Character Device Table:



# Esempio di driver a caratteri

```
#include <linux/init.h>
#include <linux/module.h>
…
struct file_operations memory_fops = {
  .owner =     THIS_MODULE,
  .read =      memory_read,
  .write =     memory_write,
  .open =      memory_open,
  .ioctl =     memory_ioctl,
  .release =   memory_release
};
char *memory_buffer; int Buffer_size = 1;

int memory_init(void) {
  register_chrdev(memory_major, "memory", &memory_fops);
  memory_buffer = kmalloc(Buffer_size, GFP_KERNEL);
  memset(memory_buffer, 0, Buffer_size); //  Setta la grandezza di memory_buffer
  return 0;
}

void memory_exit(void) {
  unregister_chrdev(memory_major, "memory");
}

ssize_t memory_read(struct file *filp, char *buf,size_t count, loff_t *f_pos) {
    copy_to_user(buf,memory_buffer,Buffer_size);
    if (*f_pos == 0) { *f_pos+=Buffer_size; return Buffer_size; } else { return 0; }
}
```

# Esempio di driver a caratteri

```
int memory_open(struct inode *inode, struct file *filp)
{
 return 0;
}

int memory_release(struct inode *inode, struct file *filp)
{
  return 0;
}

int memory_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
{
     Buffer_size = cmd;
     return 0;
}

ssize_t memory_write( struct file *filp, char *buf, size_t count, loff_t *f_pos)
{

  char *tmp;

  tmp=buf+count-Buffer_size;
  copy_from_user(memory_buffer,tmp,Buffer_size);
  return Buffer_size;
}
```

# Lettura e scrittura del driver

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <string.h>

int main()
{
     char *buffer; int Buffer_Length, fd, Byte_letti;
     printf("Inserisci numero Bytes: "); scanf("%d", &Buffer_Length);
     char dati[Buffer_Length];
     printf("Inserisci dati: "); scanf("%s", dati);

     buffer=(char *)malloc((Buffer_Length)*sizeof(char));
     strcpy(buffer, dati);

     fd = open("/dev/memory",O_RDWR);
     ioctl(fd, Buffer_Length);

     write(fd, buffer, Buffer_Length, NULL);
     Byte_letti = read(fd, buffer, Buffer_Length, NULL);

     printf("Ho letto: %d Bytes\n", Byte_letti);
     printf("Contenuto: %s\n\n", buffer);
     free(buffer);
     close(fd);
}
```

# Operazioni finali

- Makefile

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

- Scelta del major
  - Per selezionare un major non usato: cat /proc/devices
  - Selezione automatica:

```
major = register_chrdev(0, "...",&fops);
```

- Creazione device a carattere

```
mknod /dev/nomedriver c 254 0
```



Esempio di major number