

Device Driver

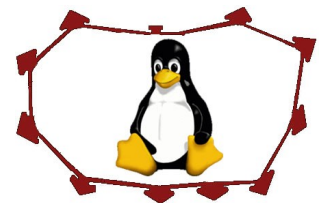
Linux Low Level -- I *Device Driver*



di

Rodolfo Giometti

`<giometti@enneenne.com>`

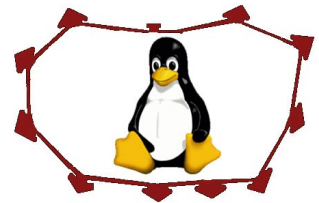


Il sistema GNU/Linux

Parliamo di *GNU/Linux* e non solo di *Linux* perché *Linux* di per se è solo il nucleo del sistema.

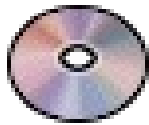


Quando parliamo di «sistema operativo» intendiamo quindi l'insieme del nucleo (*Linux*) più tutta una serie di applicazioni di gestione, utilità, ecc. (*GNU*) del sistema stesso.



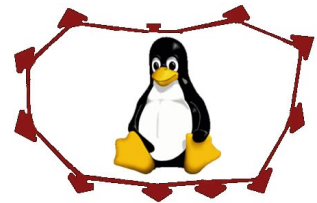
Le caratteristiche comuni

L'astrazione file: tutto all'interno del sistema è visto come un file.



Filtri: uno o più programmi semplici possono essere uniti tra loro (tramite *pipe*) fino a crearne uno più complesso:

```
$ cat /dev/framegrabber |  
    raw2jpg 800x600@16 |  
    uuencode me.jpg |  
    mail -s "See my pic" mom@home.it
```

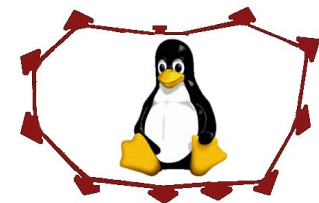
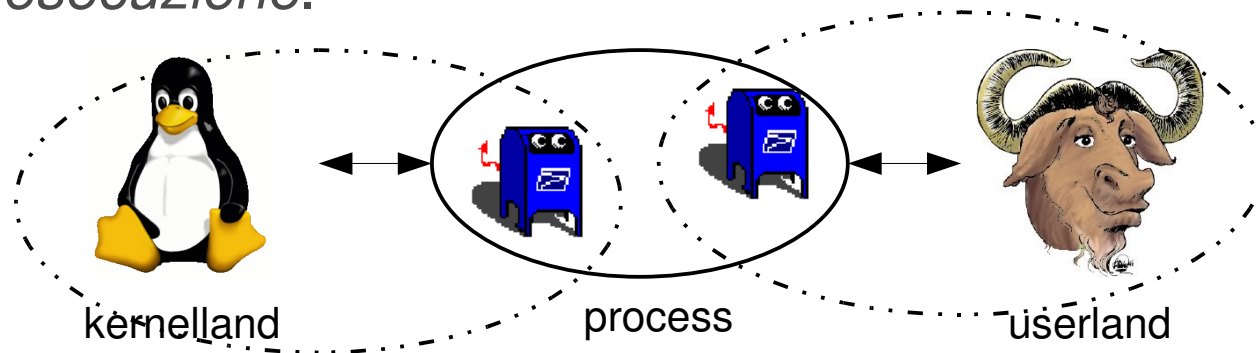


Le caratteristiche comuni (2)

Diversi filesystem: avere il supporto per più filesystem permette un'ampia possibilità di scambiare dati.

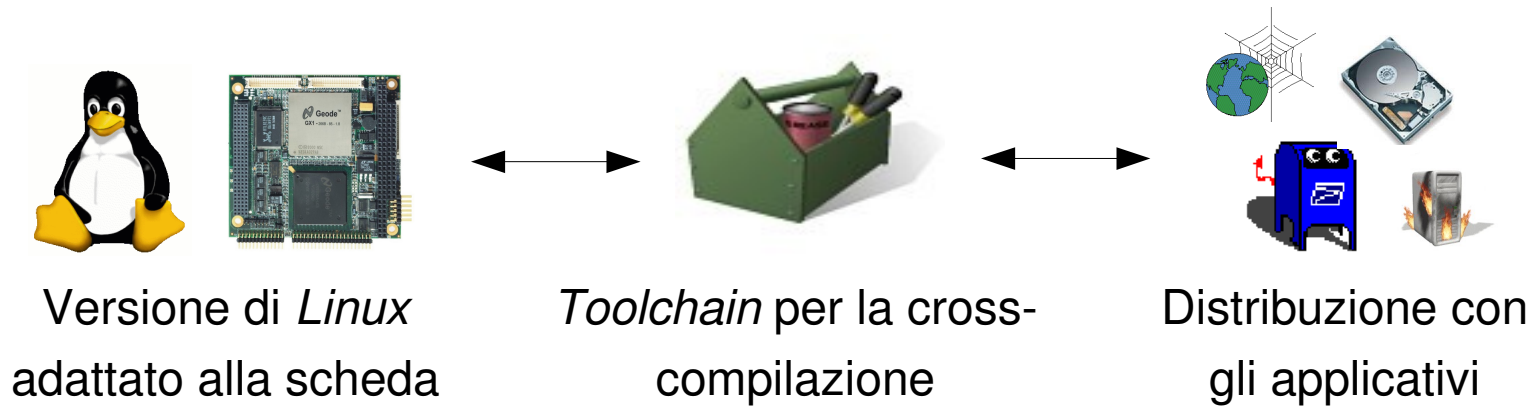


Gestione dei processi: all'interno di un sistema UNIX possono eseguire più processi «contemporaneamente» in maniera efficiente e senza possibilità di «interferenze» grazie anche al concetto di *spazio di esecuzione*.



Il sistema di sviluppo

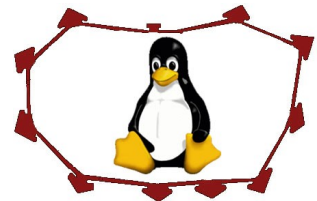
Quando vogliamo sviluppare del codice su di un sistema GNU/Linux (embedded o no) dobbiamo fornirci degli strumenti adatti.



Se questo non avviene allora occorre *arrangiarsi*:

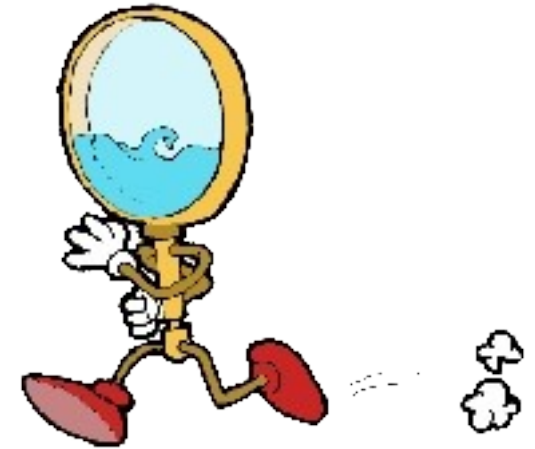


- Prendere i sorgenti
- Applicare le *patch*
- Trovare una *toolchain* & distribuzione
- Compilare, compilare, compilare...

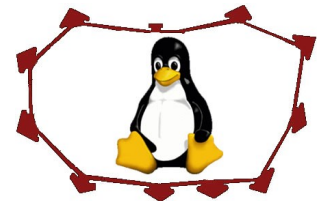


Cosa ci serve

- La versione di *Linux* che andiamo ad utilizzare deve (o dovrebbe) supportare i *driver* per tutte le periferiche del sistema. Nel caso che questo non avvenga occorre farsela fare (dal venditore o altri) o farsela in modo che si possa controllare completamente la macchina.



- Una buona *toolchain* dovrebbe includere anche altre librerie oltre alla `libc`, come le librerie grafiche (`libX`, `libgtk`, ecc.).
- Anche il bootloader è un componente importante che dovrebbe essere libero.
- Un buon supporto per lo sviluppo è dato anche dalla disponibilità o meno di tool come: `gdb`, `strace`, `netcat`, `tcpdump`, ecc..
- In fine, per la programmazione a basso livello è importante avere un JTAG.



Tecniche di sviluppo su embedded

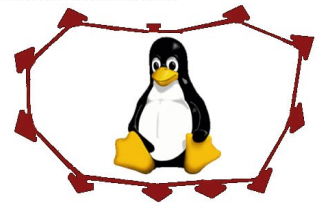
Se la nostra macchina *embedded* ha poi risorse sufficienti (una scheda di rete o porta USB-device, 32MB ed una console su LCD/Monitor) potremmo anche pensare di installare una distribuzione completa magari attraverso *NFS*! La distribuzione Debian, per esempio, essendo disponibile per molte architetture è un ottimo candidato.



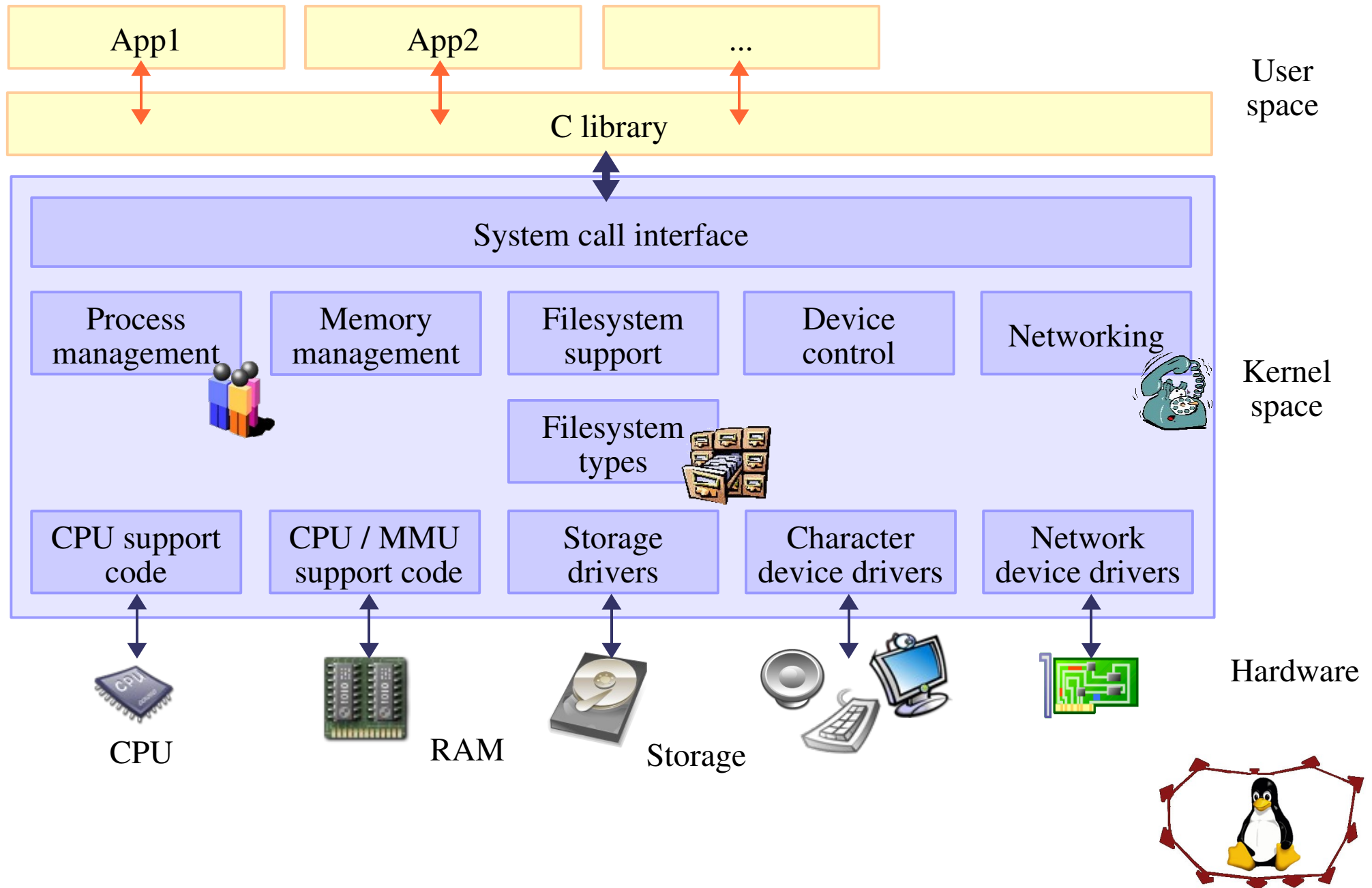
← Via NFS →



Via /etc/exports

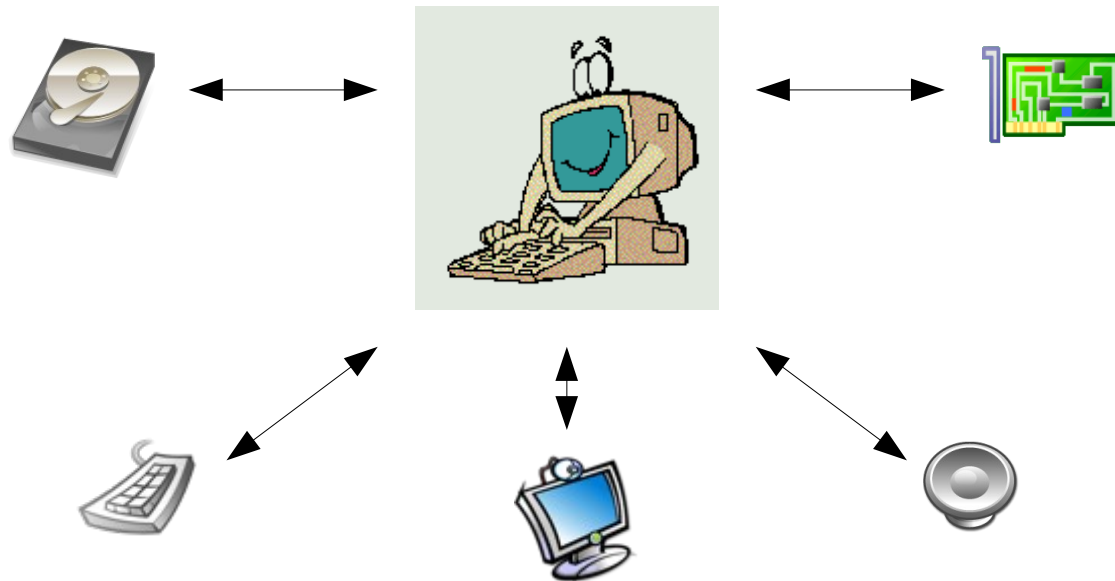


Il nucleo (o kernel)

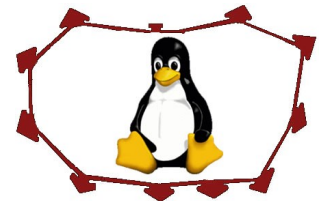


I Device Driver

Il sistema gestisce i vari dispositivi tramite i *device driver* che si occupano di trasferire dati **in maniera controllata** da e verso i processi.

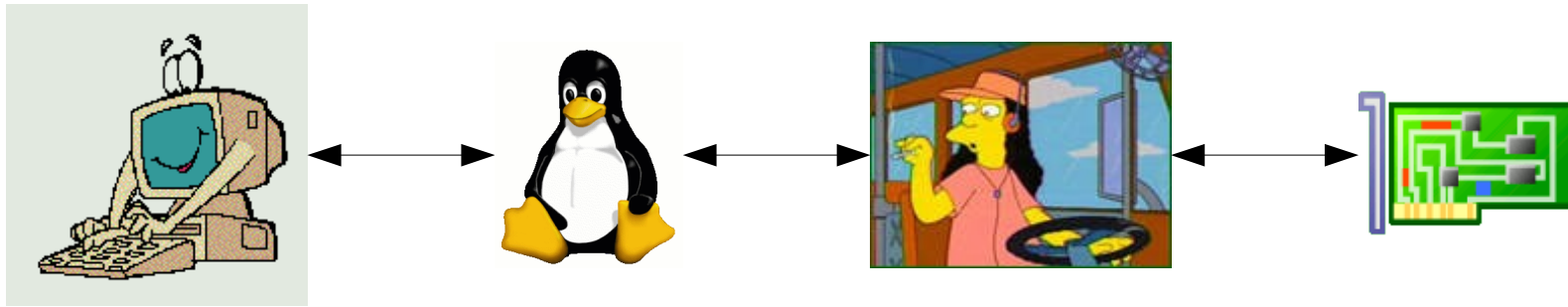


I driver sono composti da del codice particolare che si differenzia dal codice delle applicazioni (o programmi); quest'ultimo infatti non esegue dall'inizio alla fine ma viene richiamato di volta in volta quando ce n'è bisogno.

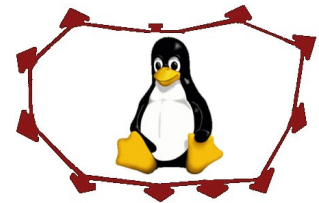


Le chiamate di sistema

Ogni qual volta che un processo vuole accedere ad un dispositivo lo fa utilizzando una *chiamate di sistema*, la quale utilizza una funzionalità del driver.



Una chiamata di sistema (*system call*) è una particolare funzione che esegue il passaggio dallo *spazio utente* (*user space*) allo spazio di nucleo (*kernel space*).



Cosa accade

- Quando un processo vuole accedere ad una periferica esso deve prima creare un descrittore di file (*file descriptor*) che «punti» ad un dispositivo:

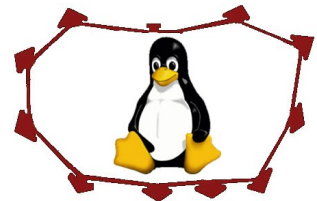
```
fd = open("/dev/sound", ...);
```

- Quindi usa una chiamata di sistema su questo descrittore di file per impostare l'operazione desiderata:

```
ret = read(fd, buff, count);
```

- A questo punto il sistema chiama la parte di codice del driver che implementa l'operazione richiesta:

```
sound_read(struct file *file,  
           char *buf, size_t count, loff_t *ppos);
```



Come è possibile

Il driver viene registrato all'interno del sistema grazie al simbolo di nucleo:

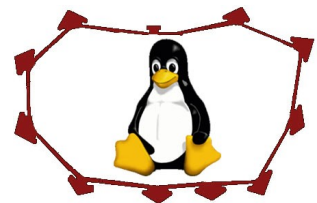
```
ret = register_chrdev(major, NAME, &sound_fops);
```

(retaggio Linux-2.4).

La variabile `major` è quella che, unitamente al «tipo» del driver, lo identifica univocamente. Nello spazio utente poi si usa:

```
mknod [OPTION]... /dev/sound c <major> 0
```

(Non vero per i dispositivi di rete o *net device*)



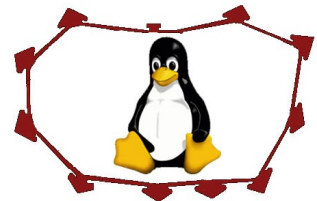
Come è possibile (2)

```
ret = register_chrdev(major, NAME, &sound_fops);
```

La struttura `sound_fops` è quella che definisce i «metodi» di accesso al dispositivo:

```
struct file_operations sound_fops = {  
    poll :          sound_poll,  
    mmap :          sound_mmap,  
    ioctl :         sound_ioctl,  
    read :          sound_read,  
    open :          sound_open,  
    release :       sound_release,  
};
```

Si parla di «metodi» in perfetta analogia con la programmazione ad oggetti; ma si programma in C!

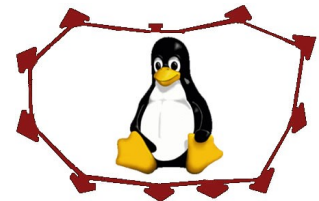


La gestione dei processi

I vari metodi poi possono gestire l'evoluzione del processo che accede al driver grazie a simboli di nucleo che ne permettono la sospensione ed il risveglio.

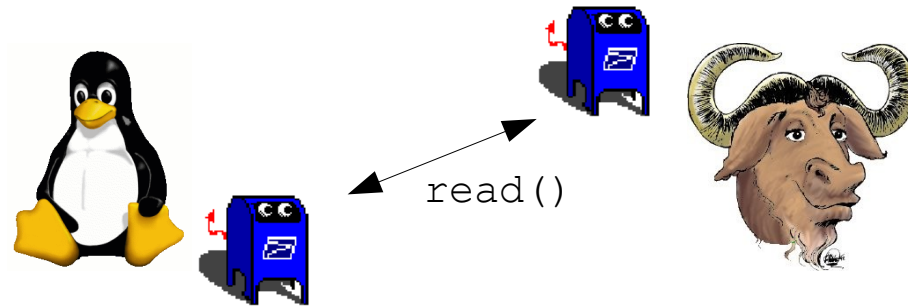
```
static ssize_t sound_read(struct file *file, char
*buf, size_t count, loff_t *ppos)
{
    ...
    ret = wait_event_interruptible(dev->queue,
        (dev->head != dev->tail) || dev->flags);
    ...
    if (copy_to_user(buf, dev->addr+
        dev->v_tail*BLOCK_SIZE, count)) {
        return -EFAULT;
    }

    *ppos += count;
    return count;
}
```

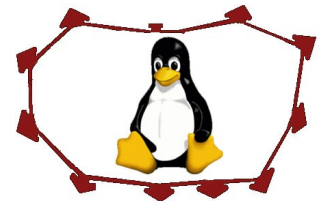
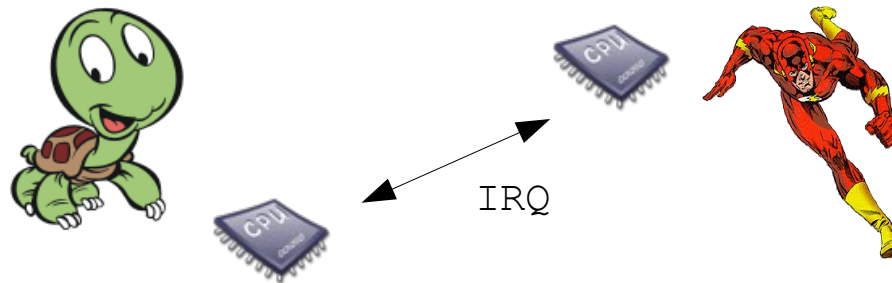


Il «contesto» di esecuzione

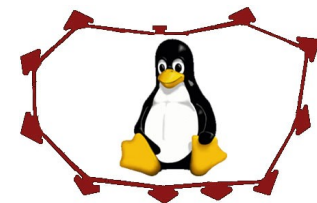
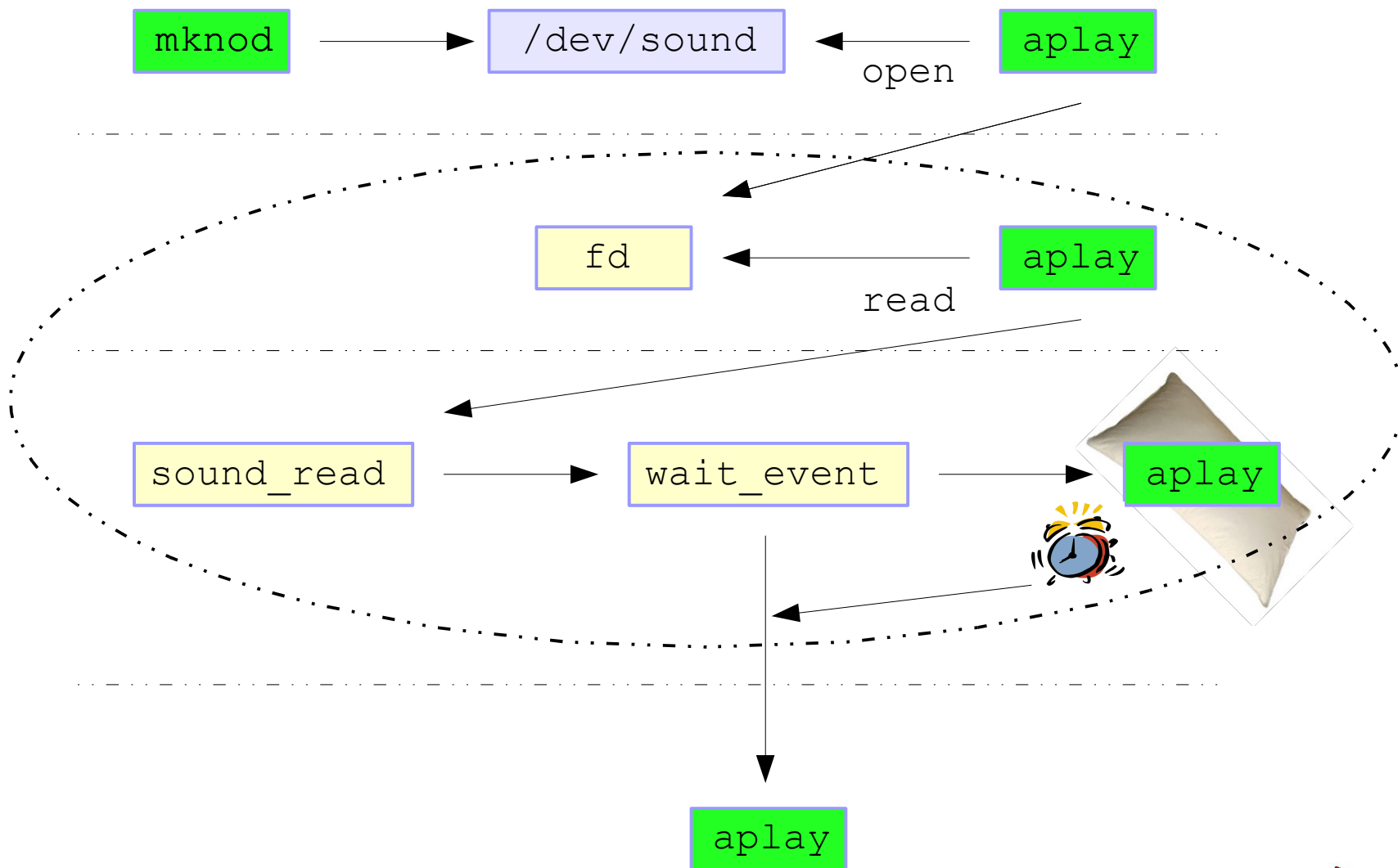
Parlando dei processi si parla di *spazio di esecuzione*, mentre parlando della CPU si parla di *contesto di esecuzione*.



I contesti di esecuzione sono principalmente 2: *process* (o *user context*) e *interrupt context*, e sopravvivono tutti nel nucleo.



Ricapitolando...



Dove trovare aiuto!!!

Use the source, Luke!

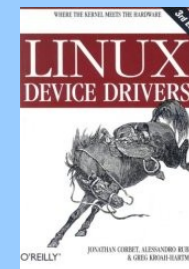
Thanks to LucasArts



Le fonti di informazione possono essere molte a partire dalla rete.



Ci sono poi i libri specifici per la programmazione in user/kernel space.



E in fine ci sono anche i «poveri» consulenti...

