SC 19/5/2026

19/05/26 10:16

SEGNALI

Gli interrupt sono un modo con cui l'HW comunica al SO la necessità di gestire una condizione particolare.

I segnali sono una astrazione con cui, in modo simile agli interrupt, il SO può notificare ad un programma una condizione particolare che deve essere gestita

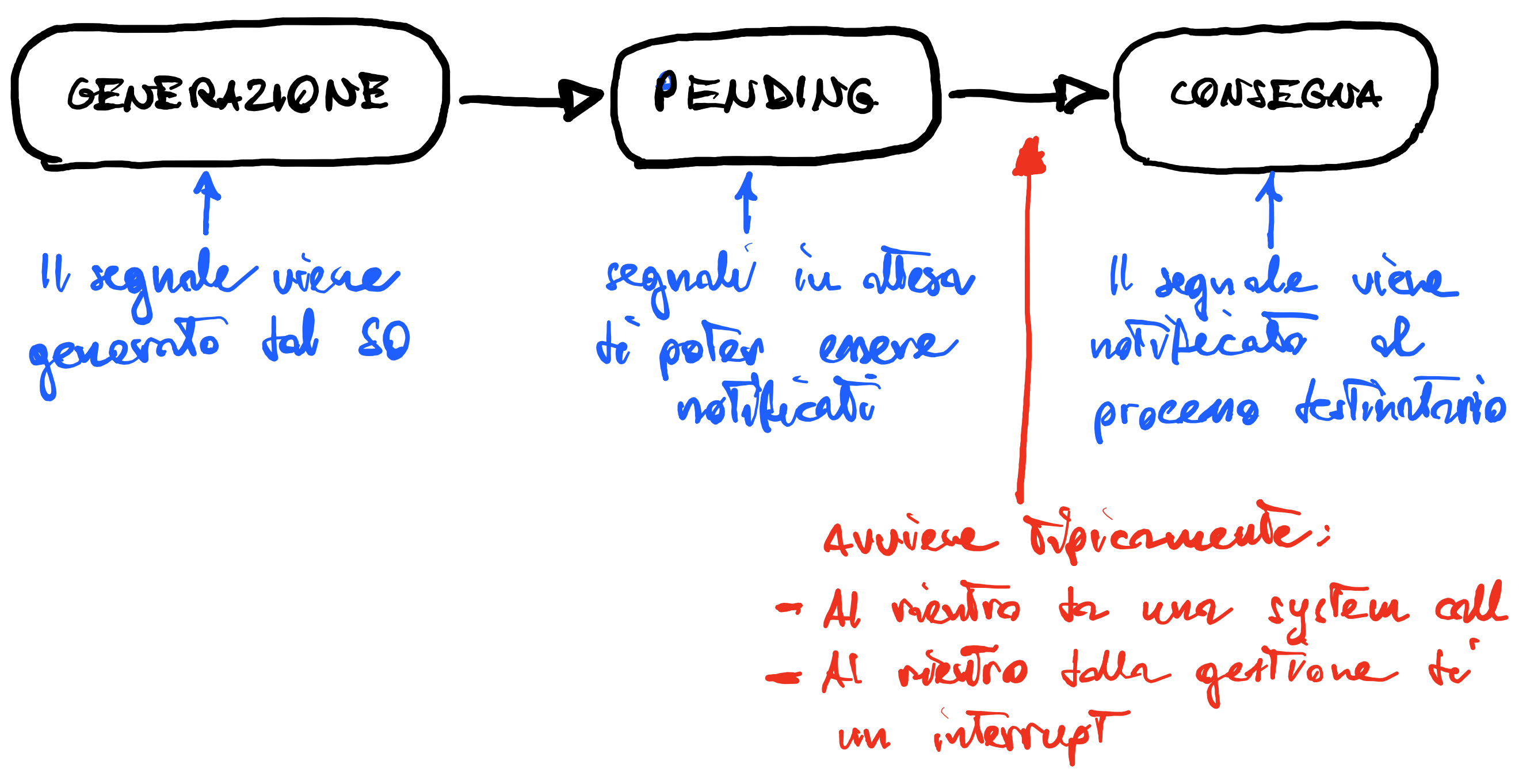
	DESTINATARIO	MITTENTE	GESTORE
INTERRUPT	KERNEL	periferiche, CPU, programma	KERNEL (int. handler)
SEGNALI	PROCESSO	kernel, programma	PROCESSO (signal handler)

Quindi, se un interrupt è sempre indirizzato all'unico kernel in funzione nella macchina, il SO può indirizzare segnali diversi a processi diversi.

Ogni segnale è caratterizzato da:

- **NUMERO IDENTIFICATIVO** : rappresenta il tipo di segnale generato. In Linux sono previsti 30 tipi di segnali diversi.
- **PID DESTINATARIO** : il PID che identifica il processo destinatario

Ogni segnale attraversa un ciclo di vita:



Quando un processo viene notificato dal kernel dell'arrivo di un segnale abbiamo tre possibili comportamenti:

TERM / CORE : il processo viene terminato, ed in alcuni casi viene generato un "core dump"

IGN : il processo "ignora" il segnale

CATCH : il processo "cattura" il segnale e passa ad eseguire una apposita routine di gestione (signal handler). È possibile solo se il signal handler è stato precedentemente configurato. Altrimenti si usa la default action

DEFAULT ACTIONS

Usate se il processo non gestisce il segnale

Copia dell'immagine di memoria del processo con lo stato della CPU salvato su disco. Utile ai fini di debug.

I principali segnali sono:

CAT	SEGNALE	ID	DESCRIZIONE	IGN?	CATCH	DEFAULT
TERMINAZIONE	SIGTERM	15	soft kill	✓	✓	TERM
	SIGINT	2	generato da CTRL+C	✓	✓	TERM
	SIGKILL	9	hard kill	✗	✗	TERM
	SIGCHLD	17	terminazione proc. figlio	✓	✓	IGN
MEM	SIGSEGV	11	accesso errato in mem	✓	✓	TERM
TIME	SIGALRM	14	timer di sistema	✓	✓	TERM

ATTENZIONE : per ogni tipo di segnale, dato il processo destinatario, ci può essere un solo segnale in stato PENDING.

Spesso la generazione di un interrupt porta alla notifica di un segnale. Ad esempio:

Utente preme CTRL+C → interrupt tastiera → int. handler → SIGINT verso processo attivo → gestione del processo
CTRL+Z → // → // → SIGSTP → //
Timer HW → interrupt timer → // → SIGALRM → //

GESTIONE DEI SEGNALI

Per inviare un segnale ad un processo:

```
# include <signal.h>

int kill ( pid_t  pid,   → PID del processo destinatario
                  int   sig);  → ID del segnale da generare
```

È possibile inviare segnali solo ai processi avviati dallo stesso utente.

Anche da terminale:

```
kill -9   12345
```

ID segnale · PID processo

Per gestire un segnale:

```
int  sigaction ( int sig,  ——→ ID del segnale da gestire
                 const struct sigaction * act, → indica come gestire il segnale
                 struct sigaction * oact); → ottiene info su come è gestito attualmente il segnale
```

Ø se tutto OK

```
struct sigaction {
    void (* sa_handler)(int);  → Definisce come gestire il segnale:
                                   • puntatore a funzione (signal handler)
                                   • SIG_IGN per ignorare il segnale
                                   • SIG_DFL per il comportamento di default
    void (* sa_sigaction)(int, siginfo_t*, void*); → serve per configurazioni più complesse
    sigset_t  sa_mask;  → set di segnali che devono essere ignorati durante l'esecuzione del signal handler
    int  sa_flags;  → flag che definiscono il comportamento del segnale
    void (* sa_restorer)(void);  → non utilizzato
}
```