Programmazione di sistema in Linux: System Call per i Segnali

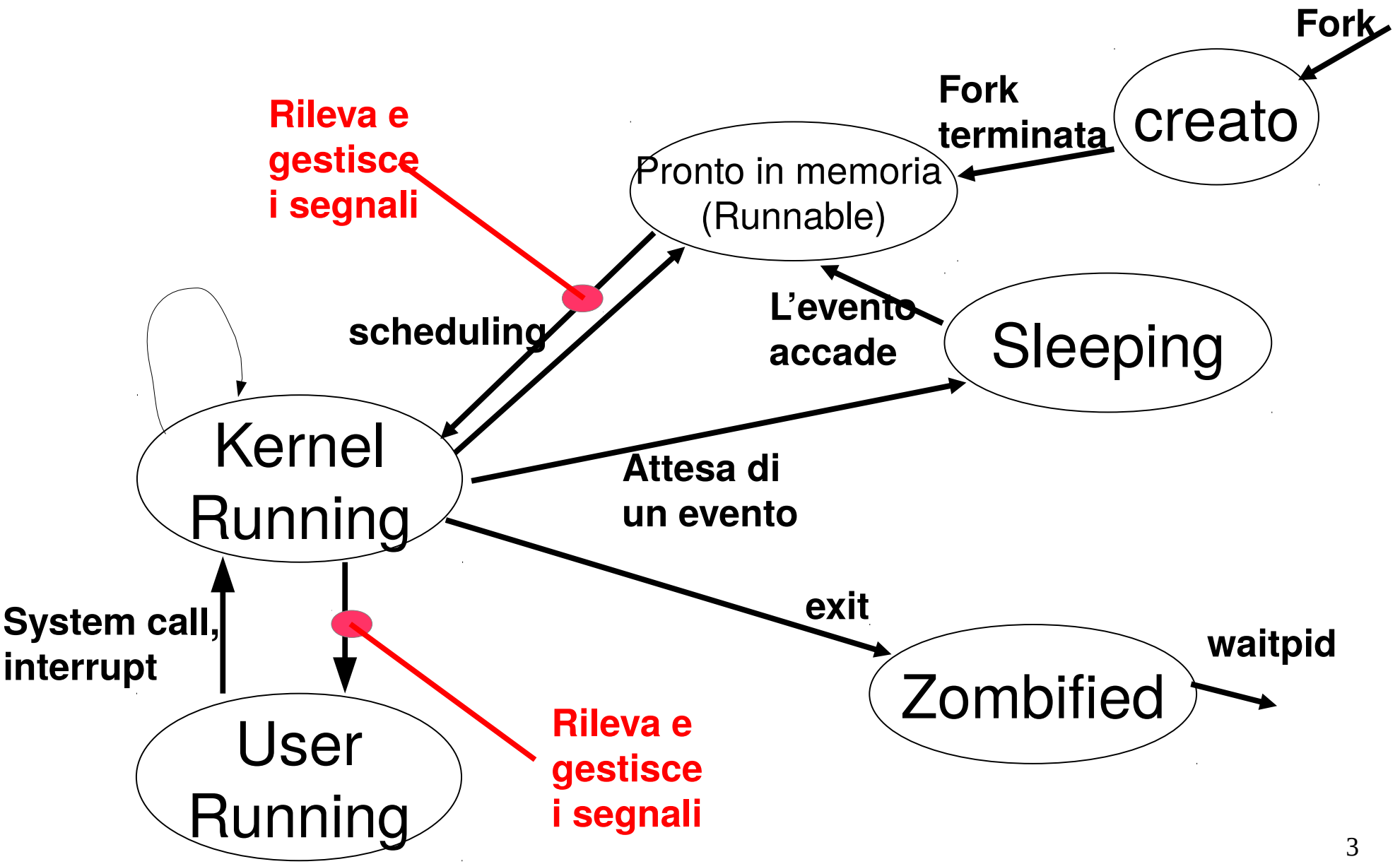
E. Mumolo

Eventi asincroni: segnali

- **I segnali** permettono la gestione di eventi asincroni che interrompono il normale funzionamento di un processo
- **Caratteristiche dei segnali**
  - Ogni segnale ha un identificatore mnemonico
  - Identificatori di segnali iniziano con i tre caratteri SIG
    - Es. **SIGABRT** è il segnale di abort
  - Numero segnali: 15-40, a seconda della versione di UNIX:
    - POSIX: 18, Linux: 38
  - I nomi simbolici corrispondono ad un intero positivo (**signal.h)**

- **I segnali sono eventi asincroni**
  - La gestione avviene tramite signal handler:
    - "Se e quando avviene un segnale, esegui questa funzione"

Stati dei processi in UNIX



Come vengono generati i segnali

- **Tramite pressione di tasti speciali sul terminale**
  - Es: Premere il tasto **Ctrl-C** genera il segnale **SIGINT**
- **Da eccezioni hardware**
  - Divisione per 0 (**SIGFPE**)
  - Riferimento non valido a memoria (**SIGSEGV**)
- **Tramite la system call kill**
  - Permette di spedire un segnale ad un altro processo
  - Vincolo se l'utente non è root, i segnali sono inviati ebtro gli stessi procerr uid
- **Tramite condizioni software**
  - Eventi asincroni generati dal software del sistema operativo, non dall'hardware della macchina. Esempi:
  - terminazione di un processo figlio (**SIGCHLD**)
  - generazione di un alarm (**SIGALRM**)

Risposta dei processi

- **Quando un segnale viene ricevuto, si può:**
  - **Ignorare il segnale**
    - Alcuni segnali non possono essere ignorati: **SIGKILL** e **SIGSTOP** per permettere al superuser di terminare processi
  - **Eseguire l'azione di default**
    - Per molti segnali "critici", l'azione di default consiste nel terminare il processo
    - Può essere generato un file di core
  - **Catturare ("catch") il segnale:**
    - Il kernel informa il processo chiamando una funzione specificata dal processo stesso (signal handler)
    - Il signal handler gestisce il problema nel modo più opportuno. **Esempio:**
      - nel caso del segnale **SIGTERM** (terminazione standard) → possibili azioni: rimuovere file temporanei, salvare file

Alcuni Segnali importanti

- **SIGABRT(Terminazione, core):** da system call **abort()**;
- **SIGALRM (Terminazione):** Generato da un timer system call **alarm** o la funzione **setitimer**
- **SIGCHLD (Default: ignore):** Quando un processo termina, **SIGCHLD** viene spedito al processo padre. Il processo padre deve definire un signal handler che chiami **wait** o **waitpid**
- **SIGFPE (Terminazione, core):** Eccezione aritmetica
- **SIGHUP (Terminazione)** Inviato ad un processo se il terminale viene disconnesso
- **SIGILL (Terminazione, core):** istruzione illegale
- **SIGINT (Terminazione):**se un processo riceve **Ctrl-C** dal terminale
- **SIGKILL (Terminazione)** Maniera sicura per uccidere un processo
- **SIGPIPE (Terminazione):** Scrittura su pipe/socket che il lettore ha terminato/chiuso
- **SIGSEGV (Terminazione, core) :** Accesso di memoria non valido
- **SIGTERM (Terminazione):** Segnale di terminazione normalmente generato dal comando kill

Segnali in ubuntu 10.04 (kill -l)

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2
13) SIGPIPE	14) SIGALRM	15) SIGTERM	16) SIGSTKFLT
17) SIGCHLD	18) SIGCONT	19) SIGSTOP	20) SIGTSTP
21) SIGTTIN	22) SIGTTOU	23) SIGURG	24) SIGXCPU
25) SIGXFSZ	26) SIGVTALRM		27) SIGPROF
28) SIGWINCH		29) SIGIO	30) SIGPWR
31) SIGSYS	34) SIGRTMIN	64) SIGRTMAX	

Invio di segnali

- **`int kill(pid_t pid, int signo);`**

kill spedisce un segnale ad un processo oppure a un gruppo di processi

Argomento **pid**:
pid > 0 spedito al processo identificato da pid

pid == 0 spedito a tutti i processi appartenenti allo stesso gruppo del processo che invoca kill

pid < -1 spedito al gruppo di processi identificati da –pid

pid == -1 non definito

Argomento **signo**:
Numero di segnale spedito

Invio di segnali

- **`int alarm(unsigned int count);`**
  - implementa un timeout
  - invia un segnale **SIGALRM** al processo che l'ha invocata dopo **count** secondi
  - restituisce (**`0`**) se non c'erano allarmi settati oppure (**`x>0`**) se mancavano **`x`** secondi allo scadere dell'ultimo allarme settato
- Esempio:

```
#include <stdio.h>

int main (void) {
   alarm(3);          /* SIGALRM fra 3 secondi */
   int i=0;
   while (1) ;        /* ciclo infinito */
   printf("Pippo") ;    /* mai eseguita */
   return 0 ;
}
```

Cattura di segnali

```
#include <signal.h>

int sigaction(int signum, const struct sigaction *act,

                  struct sigaction *oldact);
```

- serve a definire un nuovo handler
- **signum :** segnale da trattare
- **&act :** struttura che definisce il nuovo trattamento del segnale signum;
- **&oldact :** ritorna il contenuto precedente del signal handler array (può servire per ristabilire il comportamento precedente)
- ritorna (-1) se c'è stato errore

Cattura di segnali

- System call:

```
int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
```

Nuova reazione al segnale

Qui viene memorizzata la vecchia reazione al segnale

- Struttura sigaction:

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};
```

come gestire il segnale (SIG_IGN/SIG_DFL)

funzione da invocare all'arrivo del segnale

I segnali da bloccare

Non usato

- Normalmente sa_mask, flags, restorer sono posti a zero

Cattura di segnali-esempio

```
#include    <signal.h>
#include    <sys/types.h>
#include    <sys/wait.h>
#include    <stdio.h>
#include    <stdlib.h>
void func(int signum)
{
  printf("ricevo %d\n", signum);
}
int main (void)
{
  struct sigaction new_action, old_action;

  new_action.sa_handler = func;
  sigemptyset (&new_action.sa_mask);
  new_action.sa_flags = 0;

  sigaction (SIGINT, NULL, &old_action);
  if (old_action.sa_handler != SIG_IGN)    sigaction (SIGINT, &new_action, NULL);

  sigaction (SIGHUP, NULL, &old_action);
  if (old_action.sa_handler != SIG_IGN)    sigaction (SIGHUP, &new_action, NULL);

  sigaction (SIGTERM, NULL, &old_action);
  if (old_action.sa_handler != SIG_IGN)    sigaction (SIGTERM, &new_action, NULL);

   while(1) ;
}
```

Attesa di segnali

- **`int pause(void);`**
  - sospende il processo fino all’arrivo di un segnale
  - serve a implementare l’attesa passiva di un segnale
  - ritorna dopo che il segnale è stato catturato ed il gestore è stato eseguito, restituisce sempre (-1)

segnali-esempio.c: conta i segnali SIGHUP e SIGINT ricevuti

```
int main() {
//segnali-esempio.c
     pid_t pid = fork();
     if( pid == 0 ) {
         receiver();
     } else {
         sleep(1);
         generator(pid);
         waitpid(pid, NULL, 0);
     }
     return 0;
}
```

```
void contaSegnali(int signal){
    if(signal==SIGHUP) {
        contaPari++;
    } else if (signal==SIGINT){
        contaDispari++;
    }
}
```

```
void stopReceiver(int signal) {
    if(signal==SIGTERM){
        printf("Ricevuti %i pari
        ed %i dispari\n",
        contaPari, contaDispari);
        isDone = 1;
    }
}
```

```
void generator(pid_t receiver) {
    double randomUniform;
    int i, buffer[BUFFSIZE];

    /* controllo di parita' e invio del segnale  */
    for(i=0; i<BUFFSIZE; i++) {
        r=floor(((float)rand() )/RAND_MAX)*BUFFSIZE;
        if(r % 2 == 0) {
            kill(receiver, SIGHUP);
        } else {
            kill(receiver, SIGINT);
        }
        usleep(1e5);
    }
    kill(receiver, SIGTERM); /* fine */

}
```

```
void receiver() {
     /* inizializzazione delle sigaction */
     struct sigaction count_action, stop_action;

     count_action.sa_flags=0;
     stop_action.sa_flags=0;
     sigemptyset(&count_action.sa_mask);
     sigemptyset(&stop_action.sa_mask);

     count_action.sa_handler=contaSegnali;
     stop_action.sa_handler=stopReceiver;

    /* registrazione dei signal handler */
     sigaction(SIGHUP, &count_action, NULL);
     sigaction(SIGINT, &count_action, NULL);
     sigaction(SIGTERM, &stop_action, NULL);

    while(!isDone);/* busy wait */
}
```