Università di Ferrara
Laurea Triennale in Informatica
A.A. 2021-2022
Sistemi Operativi e Laboratorio

Lab-05. Gestione dei Segnali

Prof. Carlo Giannelli

http://www.unife.it/scienze/informatica/insegnamenti/sistemi-operativi-laboratorio
http://docente.unife.it/carlo.giannelli
https://ds.unife.it/people/carlo.giannelli

Installare un gestore per un segnale 1/3

Esistono due system call che permettono di installare un gestore per un segnale (signal handler).

- **signal (man signal)**
  - `sighandler_t signal(int signum, sighandler_t handler);`
  - **signal** ritorna il precedente handler in caso di successo o SIG_ERR in caso di errore.
  - Esempio: **signal(SIGUSR1, sighandlerUSR1);**

- **sigaction (man 2 sigaction)**
  - `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)`
  - **sigaction** ritorna 0 in caso di successo e -1 in caso di errore.

Installare un gestore per un segnale 2/3

- Consigli per la gestione dei segnali:
  - system call **sigaction()** preferibile a **signal()** per migliorare portabilità del programma tra sistemi UNIX/Linux differenti: ai fini del corso nessuna differenza
  - bisogna prestare particolare attenzione a dove installare i gestori per i segnali
  - I processi figli ereditano la gestione dei segnali impostati nel processo padre

Installare un gestore per un segnale 3/3

L’utilizzo di **sigaction()** richiede una serie di operazioni preliminari.

Dichiarare una variabile di tipo sigaction:

```
struct sigaction sa;
```

Inizializzare la maschera:

```
sigemptyset(&sa.sa_mask);
```

Impostare i flag a 0:

```
sa.sa_flags = 0;
```

Impostare l’handler per il segnale:

```
sa.sa_handler = sighandlerUSR1;
```

Installare il gestore e controllare eventuali errori:

```
if (sigaction(SIGUSR1, &sa, NULL) < 0) {
   perror("sigaction");
   exit(1);}
```

Inviare un segnale 1/2

Per inviare un segnale bisogna utilizzare la system call **kill**.

kill (man 2 kill) ha la seguente definizione:

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

dove:

- `pid` è l'identificatore del processo a cui si vuole inviare il segnale
- `sig` è il segnale che si vuole inviare

Esempio: `kill(pid, SIGUSR1);`

Inviare un segnale 2/2

In generale, i processi figli ereditano dal processo padre il **process group ID**

Il process group ID può essere modificato con la system call setpgid
https://man7.org/linux/man-pages/man2/setpgid.2.html

```
int kill(pid_t pid, int sig);
```

- **pid > 0**, sig to **the only process** with the ID “pid”
- **pid = 0**, sig to **every** process in the **process group** of the calling process
- **pid < -1**, sig sent to **every process** in the process group with ID -pid
- **pid = -1**, then… see the man

https://man7.org/linux/man-pages/man2/kill.2.html

Esercizio 1 - Consegna

Si scriva un programma che preveda la seguente interfaccia:

itercounter Nf Nsec

In questo programma, un processo padre deve generare un numero **Nf** di processi figli. Ognuno di essi esegue un ciclo infinito, all'interno del quale il processo dorme per un secondo.

Il padre a sua volta dorme per **Nsec** secondi, dopodiché lancia un **SIGUSR1** ai figli.

Al ricevimento del segnale **SIGUSR1**, ogni figlio deve visualizzare le iterazioni compiute, prima di terminare le operazioni.

Esercizio 1 - Traccia

- **#include** necessarie
- gestore del segnale, con stampa del contatore globale
- controllo argomenti
- generazione dei figli
- se sono nel figlio:
  - imposto la gestione del segnale SIGUSR1
  - while(true):
    - dormo per 1 secondo
    - aumento il contatore
- se sono nel padre:
  - dormo per Nsec secondi
  - invio SIGUSR1 ai figli
  - attendo la morte dei figli

NB: due possibilità per inviare SIGUSR1 a tutti i figli. Quali? Cosa cambia?

Esercizio 2 - Consegna 1/2

Si scriva un'applicazione in C (con le system call del sistema operativo Unix) che presenti l'interfaccia:

cercaCarFile C1 [C2 ... Cn] NomeFile NumSec

dove i parametri **C1 [C2 ... Cn]** sono singoli caratteri alfabetici, **NumSec** è un numero intero e **NomeFile** è un nome di file assoluto.

L'applicazione è una semplice gara tra processi. Il processo iniziale deve generare un processo figlio per ciascun carattere **Ci** passato come parametro. Ciascuno di essi avrà il compito di **cercare uno specifico carattere all'interno del file passato come parametro**. In particolare, l'i-esimo processo figlio deve contare quante volte il carattere **Ci** è presente in **NomeFile**. Al termine della gara, ciascun figlio deve stampare a video il numero di caratteri trovati. Vince la gara il processo che finisce per primo.

Esercizio 2 - Consegna 2/2

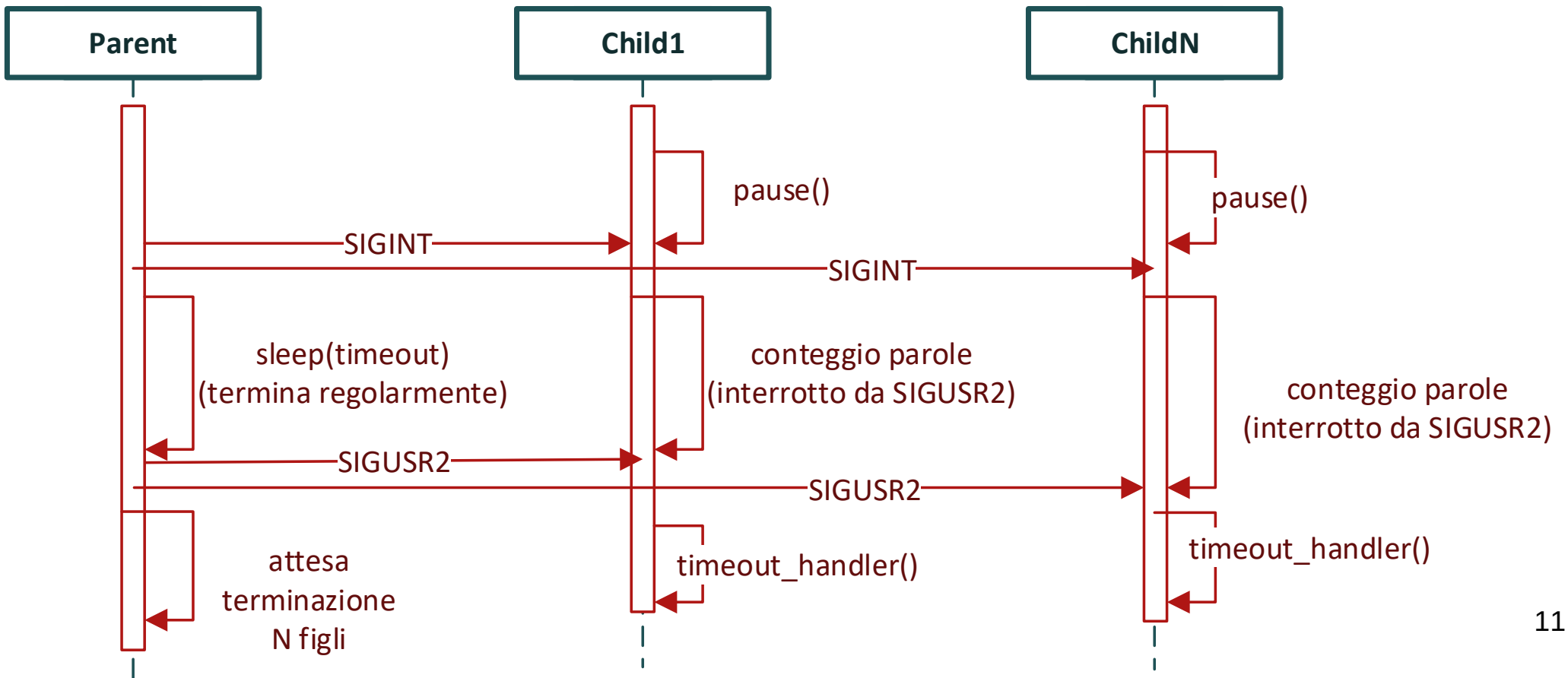
Si progetti l'applicazione cercando, per quanto possibile, di rendere la competizione tra i processi giusta, nel senso che **i processi dovrebbero avere tutti lo stesso tempo NumSec** per eseguire il conteggio. A questo proposito, il processo padre deve svolgere il ruolo di "arbitro" e utilizzare i segnali come strumento IPC per la notifica ai processi figli degli eventi di inizio e fine gara.

- **Se un processo figlio termina il conteggio prima che il tempo concessogli sia scaduto, vince la gara**. Il processo vincitore notifica la sua vittoria agli altri processi, che interrompono il conteggio, stampano a video il numero di caratteri trovati e infine terminano la loro esecuzione.
- **Se il tempo scade e nessun figlio ha già terminato**, il padre notifica ai figli che la gara è finita (stampano a video il numero di caratteri trovati e terminano).

Eseguire la ricerca dei caratteri in un file di testo di grandi dimensioni.

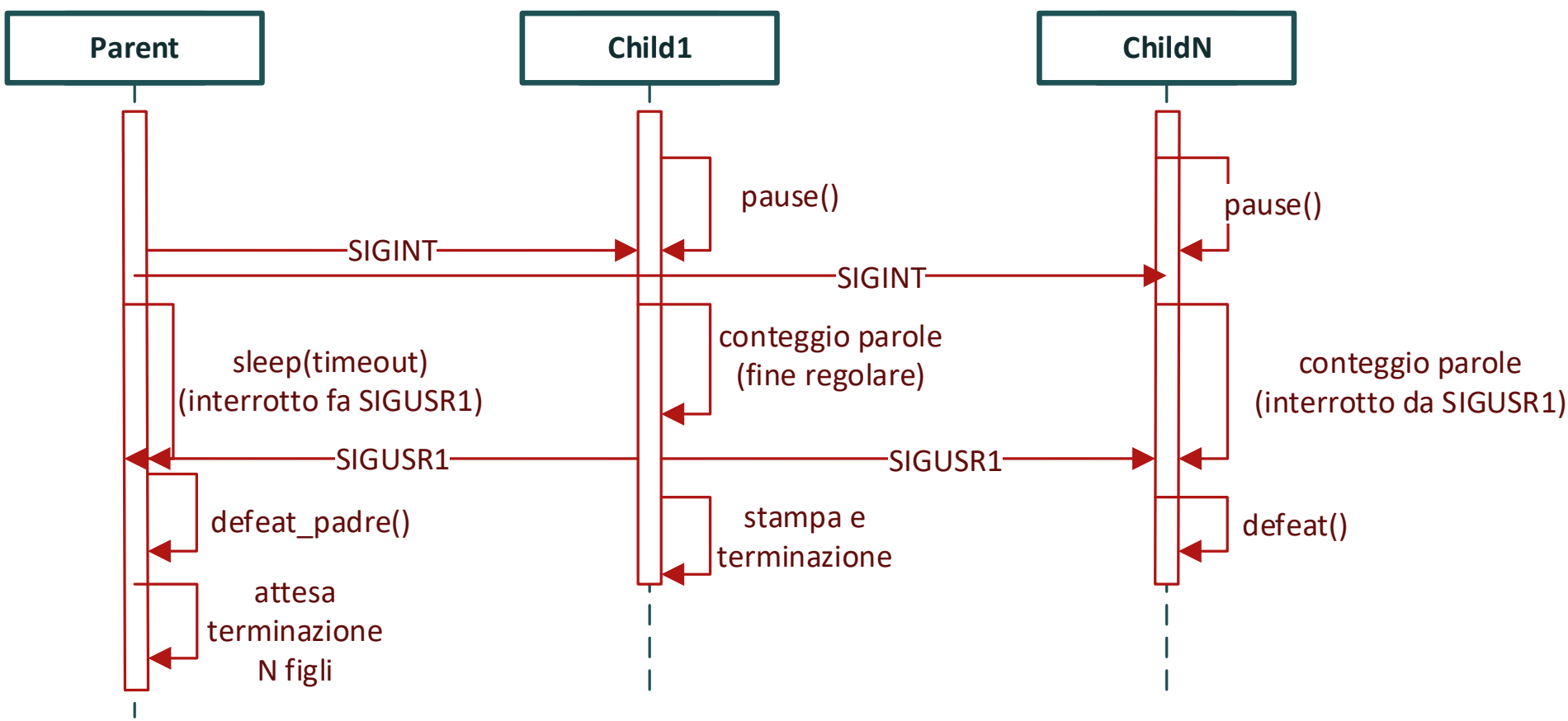
Esercizio 2 - Traccia 1/4

- **Nessun figlio termina prima del timeout: gara termina senza vincitore**
  - figli attendono inizio gara con pause()
  - padre avvisa inizio gara con SIGINT
  - timeout padre e nessun figlio ha ancora terminato
  - padre avvisa fine gara con SIGUSR2
  - figli avviano timeout_handler() per finire senza un vincitore



Esercizio 2 - Traccia 2/4

- **Un figlio termina per primo (e prima del timeout del padre): vincitore**
  - figli attendono inizio gara con pause()
  - padre avvisa inizio gara con SIGINT
  - uno dei figli termina: 1) invia SIGUSR1 al padre e agli altri figli,
    2) dichiara la propria vittoria, 3) termina
  - altri figli avviano defeat() per dichiarare la propria sconfitta
  - il padre avvia defeat_padre() per terminare con un vincitore



Esercizio 2 - Traccia 3/4

- **#include** delle librerie

- segnali e loro gestori:
  - SIGINT da padre a tutti i figli per **segnalare inizio gara** → gestore start_race sui figli
  - **se nessun figlio termina prima del timeout**, allora SIGUSR2 a tutti i figli per indicare che la gara è finita senza un vincitore → gestore timeout_handler su figli
  - **se un figlio termina prima del timeout**, allora SIGUSR1 da figlio che ha vinto a tutti gli altri figli (quindi escludendo se stesso) e al padre per segnalare che ha vinto e la gara è finita → gestore defeat sui figli e gestore defeat_padre sul padre

Esercizio 2 - Traccia 4/4

- controllo degli argomenti e dell'esistenza di **NomeFile**
- generazione dei figli

- se codice del padre:
  - ignoro segnali usati per gestire la gara
  - invio segnale di start ai figli: SIGINT
  - aspetto NumSec secondi
  - tempo scaduto, invio segnale per la stampa risultati e attendo terminazione dei figli

- se codice di un figlio:
  - installazione gestione dei segnali
  - aspetto il segnale di start: system call pause()
  - eseguo il conteggio sul file
  - se sono arrivato qui ho vinto, faccio terminare la gara e ignoro il segnale di terminazione
  - stampo il risultato
  - invio segnale di terminazione agli altri processi