Laboratorio di Sistemi Operativi, A.A. 2022–2023

Simulazione di esercitazione

Lunedì 5 giugno 2023

Esercizio 1

Realizzare un programma `main.c` che, avendo a disposizione una pipe di nome `pipe` già creata nella directory corrente:

- scrive nella pipe il proprio process ID;
- stampa sullo standard error una successione di righe contenenti il proprio process ID, seguito da uno spazio, un numero progressivo che parte da 1 e incrementato di volta in volta, e un carattere di "a capo". Le righe devono essere stampate a intervalli di tempo regolari, inizialmente di un secondo.

In aggiunta, il processo deve rispondere ai seguenti segnali:

`SIGUSR1` il processo deve scrivere nella pipe il valore attuale del contatore;

`SIGUSR2` il processo deve leggere dalla pipe il nuovo intervallo fra le stampe, in secondi;

`SIGALRM` il processo deve sdoppiarsi: il figlio deve scrivere nella pipe il proprio process ID, poi deve proseguire nel conteggio.

Specifiche:

- I numeri interi devono essere scritti nella pipe come stringhe in rappresentazione decimale, terminate da un carattere di "a capo". In altre parole, i valori trasmessi attraverso la pipe devoo essere umanamente leggibili.
- L'intervallo di attesa fra una stampa e la successiva potrebbe essere sbloccato prematuramente alla ricezione di un segnale: non è un problema.
- Si può assumere che non ci sia mai contesa per l'uso della pipe: quando un processo riceve un segnale, la pipe è da considerarsi disponibile per l'utilizzo previsto.

Per collaudare il programma, possiamo aprire due terminali: nel primo ci poniamo in ascolto di `pipe` e nell'altro lanciamo il programma. Nel primo terminale dovremmo veder comparire il PID del processo appena avviato (per esempio `12345`), nel secondo il processo dovrebbe stampare la seguente sequenza di righe, una al secondo:

```
12345 1
12345 2
12345 3
⋮
```

Il collaudo delle funzionalità legate ai segnali può a questo punto aver luogo nel primo terminale, ogni volta inviando il segnale al processo e escrivendo nella pipe o mettendosi in ascolto.

Esercizio 2

Realizzare lo script bash `test.sh` che avvia il programma in un terminale separato, ne stampa il PID ricevuto via pipe, poi:

- dopo 5 secondi stampa il valore attuale del contatore;
- dopo altri 5 secondi fa sdoppiare il processo e stampa il PID comunicato dal figlio;
- dopo altri 5 secondi rallenta il processo figlio aumentando il ritardo di stampa a 2 secondi;
- dopo altri 30 secondi termina i due processi.

Segue un esempio di svolgimento dello script (i numeri possono variare):

Terminale principale	**Secondo terminale**
`./test.sh`	*Appare il secondo terminale*.
`Processo 12345 avviato.`	`12345 1`
	⋮ (*Una riga al secondo*)
	`12345 5`
Dopo 5 secondi:	`12345 6`
`Processo 12345: contatore=6`	`12345 7`
	⋮
Dopo 5 secondi:	`12345 11`
`Processo 12360 avviato.`	`12345 12`
	`12360 12`
	`12345 13`
	`12360 13`
	⋮
	`12345 18`
Dopo 5 secondi:	`12360 18`
`Processo 12360 rallentato.`	`12360 19`
	`12345 19`
	`12345 20`
	`12360 20`
	`12345 21`
	`12345 22`
	`12360 21`
Dopo 30 secondi:	⋮ (*Una riga di 12360 ogni due di 12345*)
`Termino il processo 12345`	
`Termino il processo 12360`	*Il terminale si chiude*.

Esercizio 3

Realizzare un Makefile con i seguenti obiettivi:

pipe	crea la pipe;
main	compila l'eseguibile di nome `main`;
all	(default) crea la pipe e compila l'eseguibile;
test	crea tutto il necessario ed esegue lo script `test.sh`;
clean	cancella tutti i file eseguibili e la pipe, lasciando solo i file sorgenti scritti dallo studente.

Indicazioni utili

Le seguenti informazioni servono solo a facilitare la stesura del codice, ma non è obbligatorio seguirle.

- Comandi bash utili (manuale accessibile con `man 1` *nomecomando*):
  `echo`, `cat`, `kill`, `sleep`, `mkfifo`.
  - Per l'avvio di un nuovo terminale e l'esecuzione del processo `./main` al suo interno:
    `gnome-terminal -- ./main`
- Altre funzionalità utili di bash:
  - Esecuzione di un comando in background (senza il blocco del terminale):
    comando `&`
  - Cattura dello standard output di un comando e sua assegnazione a una variabile d'ambiente:
    nomevariabile`=$(`*comando*`)`
    (senza spazi prima del carattere "=")
- Chiamate di sistema utili (manuale accessibile con `man 2` *nomefunzione*):
  `open`, `write`, `read`, `close`, `fork`, `signal`.
- Altre funzioni utili (manuale accessibile con `man 3` *nomefunzione*):
  `fprintf`, `sprintf`, `atoi`, `strlen`, `sleep`.
- I seguenti file header dovrebbero essere sufficienti per tutte le funzioni utilizzate nell'esercitazione:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
```