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

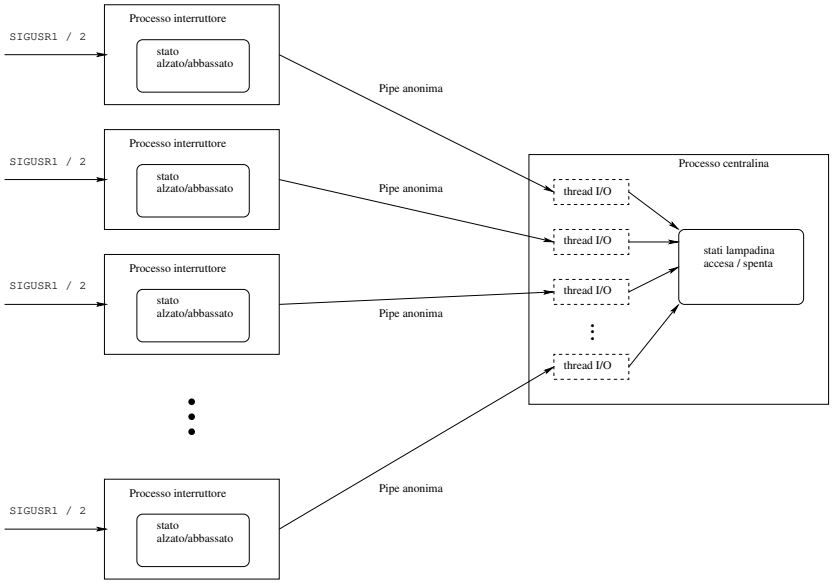
Simulazione di esercitazione

Lunedì 29 maggio 2023

Nota bene — questa esercitazione può richiedere più tempo delle due ore previste per il test di laboratorio. Si suggerisce di provare a svolgerla comunque per verificare le proprie competenze.

Esercizio 1

In questo esercizio simuleremo un sistema domotico in cui una schiera di interruttori comanda l'accensione di una singola lampadina attraverso una centralina. Se il numero di interruttori con la levetta alzata è pari, allora la centralina accende la lampadina, altrimenti la spegne.
Il programma C deve creare una serie di processi figlio, ciascuno dei quali mantiene lo stato di un interruttore il cui stato può essere modificato inviando al relativo processo i segnali `SIGUSR1` (levetta alzata) e `SIGUSR2` (levetta abbassata). Ogni processo figlio è collegato al processo genitore, che agisce da centralina, attraverso una pipe anonima attraverso la quale comunica ogni cambio di stato. Sulla base dello stato degli interruttori, la centralina decide se accendere o meno la lampadina:



Il programma è invocato passando a parametro il numero di interruttori / processi figlio. Il processo genitore deve, per ogni interruttore (processo figlio), creare una nuova pipe anonima, creare il processo figlio, tenere per sé il lato in lettura della pipe, mettere un thread in ascolto della pipe. Il numero di interruttori viene passato come parametro di avvio.
All'avvio, ogni interruttore (processo figlio) deve stampare il proprio PID, inizializzare lo stato della propria levetta a "basso", e inviarlo alla centralina attraverso la pipe. Quindi si mette perpetuamente in attesa di segnali.
A ogni segnale, se lo stato dell'interruttore varia, il processo che lo gestisce deve stampare una notifica sul terminale e inviare il nuovo stato alla centralina attraverso la pipe anonima; anche la centralina deve segnalare l'accensione / spegnimento della lampadina attraverso messaggi sul terminale.
Si propone l'uso di thread per l'ascolto delle pipe da parte del processo genitore in modo da poter eseguire più `read()` bloccanti in contemporanea (una valida alternativa sarebbe l'uso di `select()`, che però non è stata vista a lezione).
Punteggio pieno se il codice dell'interruttore (processo figlio) e della centralina (processo genitore, dopo l'avvio dei figli) si trovano in moduli separati, ciascuno con i propri file header e sorgente. È consentito l'uso di variabili globali.

Esercizio 2

Realizzare lo script bash `test.sh` che riceve come parametri i PID degli interruttori e invia in sequenza un segnale di levetta alzata a ciascuno di essi, seguito da una sequenza di segnali di levetta abbassata. Il ciclo va ripetuto all'infinito.
I segnali devono essere inviati a distanza di un secondo uno dall'altro.
Per avere punteggio pieno, lo script dovrà essere invocato con un comando simile a

```
./test.sh 12345 12346 12347 12348
```

senza menzionare esplicitamente la shell nella riga di comando.

Esercizio 3

Realizzare un Makefile con i seguenti tre obiettivi:

- `main` (obiettivo di default) crea l'eseguibile dell'esercizio 1;
- `clean` cancella tutti i file eseguibili, 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`, `kill`, `sleep`.
- Comandi bash built-in utili (manuale accessibile con `help` *nomecomando*):
  `for`, `while`.
- Chiamate di sistema utili (manuale accessibile con `man 2` *nomefunzione*):
  `pipe`, `write`, `read`, `close`,
  `pthread_mutex_init`, `pthread_mutex_lock`,`pthread_mutex_unlock`,
  `pthread_start`, `pthread_join`,
  `fork`, `signal`, `pause`.
- Altre funzioni utili (manuale accessibile con `man 3` *nomefunzione*):
  `printf`, `puts`, `malloc`.
- I seguenti file header dovrebbero essere sufficienti per tutte le funzioni utilizzate nell'esercitazione:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
```