

## 1. comunicazione fra processi tramite *pipe()*

---

Completare il programma presentato a lezione come *pipe1.c*. Come si invoca? come passare l'argomento se gli vogliamo far scrivere la stringa seguente?

```
ciao a tutti!
```

Inserire gli header file necessari alla compilazione e all'esecuzione basandosi sugli errori e warning rilevati dal compilatore.

Modificare il programma in modo che se *argc* vale 3 (cioè se il programma viene invocato con un secondo argomento) il processo figlio esegua un'ulteriore *fork()* e passi al proprio figlio un altro pipe da cui il figlio leggerà. Anche il secondo argomento deve essere stampato a video.

## 2. interfaccia a pipe

---

Scrivere due programmi cooperanti: il loro compito è prendere in input una stringa da tastiera e convertirla in caratteri maiuscoli (i caratteri eventualmente già maiuscoli devono restare inalterati).

Il primo programma (*filter.c*) implementa la seguente funzionalità: legge da *stdin* una stringa (un carattere alla volta) e restituisce il carattere maiuscolo corrispondente [NB: si ferma quando il carattere letto è EOF].

Il secondo programma (*main\_pipe.c*) apre un pipe in lettura (utilizzando come 'comando' il programma *filter*) e poi tenta di leggere dal pipe.

Verificare che fino a quando *filter* non produce un output il main resta in attesa.

## 3. *pipe* come strumento di sincronizzazione

---

Scrivere un programma seguendo le seguenti istruzioni:

- il genitore crea un pipe (utilizzando la system call *pipe()*) prima di creare 3 processi figli;
- ogni figlio eredita i descrittori del pipe, e chiude immediatamente il proprio descrittore per il *read end* del pipe. L'azione di ogni figlio è semplicemente mettersi in *sleep* per un paio di secondi, chiudere anche il descrittore per il *write end* e terminare;

- dopo che tutti i figli hanno chiuso i descrittori per il *read end* del pipe, il padre effettua un'operazione *read()* dal pipe ottenendo *end-of-file*. A tale fine, consultare il manuale per la funzione *read()*.

PERCHÉ si tratta di una forma di sincronizzazione: il padre *NON* effettua una *wait()* per attendere la terminazione dei figli, ma resta in attesa in lettura dal *read end* del pipe.

*Notare come la chiusura del write end del pipe da parte del padre è essenziale al funzionamento di questa tecnica (in caso contrario il padre si bloccherebbe nel tentativo di leggere dal pipe).*

Modificare il programma precedente in modo che prenda in input un numero arbitrario di interi letti da tastiera: fare attendere il primo figlio *argv[1]* secondi, il secondo *argv[2]* secondi, ..., e l'*n*-esimo figlio *argv[n]* secondi.