labso2022-1--esame–info+testo

All'interno della cartella di lavoro realizzare un'applicazione scrivendo un singolo file "`main.c`" il cui eseguibile compilato dovrà chiamarsi "`coda`" creando un "makefile" rispettando le indicazioni di seguito.

VINCOLI:

- la cartella finale deve contenere solo due file: "`main.c`" e "`Makefile`"
- digitando "`make`" all'interno della cartella il file deve generare un binario denominato "`coda`" che è una sorta di "gestore di QUEUE" che accetta 3 o 4 parametri: un "nome", una "azione", un eventuale "valore" aggiuntivo (il cui significato dipende dall' "azione") e un "PID". Per tutti i parametri si può considerare un massimo di 32 caratteri ciascuno.

ATTIVITÀ:

- [4/40] realizzare quanto sopra indicato gestendo correttamente i parametri

- [2/40] realizzare correttamente il "Makefile" e far sì che digitando "`make DEST=<path>`" il binario sopra citato deve essere generato dentro la cartella *<path>*. Se la cartella, quindi il percorso, non esiste o non è accessibile si deve stampare a video su stderr il testo "`?ERROR`"

- con `./coda` *<name> <action>* [*<value>*] *<pid>* si deve accedere alla QUEUE *<name>* (*) per poi eseguire l'azione identificata con *<action>*. Dopo aver eseguito l'azione l'applicazione DEVE poi inviare al processo il cui pid è l'ultimo argomento un segnale SIGUSR1 in caso di successo e un SIGUSR2 in caso di errore di qualunque genere.

  <action> può assumere uno dei seguenti valori:

  - [8/40] `'new', 'put', 'get'`:

    - `'new'` : crea una QUEUE con riferimento di nome *<name>*(*) se già non esiste. Se esiste ne recupera il riferimento stampando su *stdout* l'identificativo della coda stessa,

      es.:
      ```
      ./coda /tmp/coda1 new 100 # event. crea QUEUE e stampa il suo id
      ```

    - `'put'` : scrive *<value>* dentro la QUEUE *<name>* (la crea se non esiste), es.:
      ```
      ./coda /tmp/coda1 put ciao # salva la stringa "ciao" in QUEUE
      ```

    - `'get'` : legge un dato dalla QUEUE *<name>* (la crea se non esiste) e restituisce il valore (nulla se non ci sono valori) in *stdout* seguito da un carattere a capo ('\n'), es.:
      ```
      ./coda /tmp/coda1 get # stampa "ciao" dopo il comando prec.
      ```

  - [6/40] `'del', 'emp'`:

    - `'del'` : elimina la QUEUE *<name>* (se esiste, altrimenti non fa nulla), es.:
      ```
      ./coda /tmp/coda1 del # event. elimina la QUEUE
      ```

    - `'emp'` : legge la QUEUE *<name>* (la crea se non esiste) restituendo in *stdout* i dati uno per riga, es.:
      ```
      ./coda /tmp/coda1 emp # stampa tutti i dati della QUEUE
      ```

  - [5/40] `'mov'` : sposta l'intero contenuto della QUEUE *<name>* su un'altra denominata *<value>*. Se la prima coda non esiste restituisce errore. Se la seconda coda esiste già usa quella che c'è, altrimenti la crea. La prima coda deve essere poi eliminata mostrando su *stdout* i valori durante lo spostamento (analogamente al comando `'emp'`) e alla fine il numero di messaggi spostati (seguito sempre da '\n'), es.:
    ```
    ./coda /tmp/coda1 mov /tmp/coda2 # da /tmp/coda1 a /tmp/coda2
    ```

- deve essere restituito il codice 0 in uscita o un valore maggiore nel caso di errori (ad esempio se fallisce una chiamata a una syscall).

(*) Ogni QUEUE deve essere creata partendo dalla funzione `ftok` con primo parametro il valore dell'argomento passato dall'utente (creando eventualmente un file con lo stesso nome se non esiste) e secondo parametro pari a 1. Ad esempio con `./coda /tmp/xxx new` si deve utilizzare qualcosa come `ftok("/tmp/xxx", 1)` per generare la "chiave" univoca per poi ottenere l'identificativo della coda stessa attraverso la funzione `msgget`.