labso2022-1--esame–info+testo
ISTRUZIONI PRATICHE
Esame del modulo di laboratorio di “Sistemi Operativi”

Durata: 120’ (2 ore)

● Creare una cartella principale denominata con il proprio numero di matricola e dentro tutti i contenuti richiesti dal compito oltre a rispondere alle domande previste nel modulo online.

● Questa cartella andrà consegnata “zippandola” (compressione formato “zip”) in modo da creare un file avente per nome il proprio numero di matricola più l’estensione “.zip”. Deve essere compressa l’intera cartella e non solo il suo contenuto.

Se il proprio numero di matricola fosse 123456 questo deve essere anche il nome della cartella e l’archivio compresso da consegnare deve chiamarsi 123456.zip.

● Consegna:

○ Dopo 50’ ed entro 60’ dall’inizio della prova si deve fare una prima consegna (lavoro parziale) con il lavoro compiuto complessivo fino a tale momento ANCHE SE NON FUNZIONANTE utilizzando il modulo online che sarà indicato.

○ Dopo 90’ ed entro 120’ dall’inizio della prova si deve fare una seconda consegna (lavoro finale) utilizzando il modulo online che sarà indicato (diverso dal precedente): questa consegna è l’unica considerata per la valutazione finale. Rispondere inoltre alle domande valide per la valutazione.

○ I moduli sono “moduli Google”: effettuare il login con il proprio account universitario e inserire tutti i dati richiesti (orientativamente: nome, cognome, numero di matricola e file “zip” da allegare)

○ I punteggi sono indicativi dato che la valutazione tiene conto anche di dettagli “trasversali” che non sono riferibili a singoli punti.

La parte di sviluppo vale fino a 25 punti, mentre la parte con le domande vale fino a 15 punti con 5 quesiti a risposta multipla (1 sola risposta esatta, 2 errate: 3 punti per ogni risposta esatta, 0 per ogni risposta non data e -1 per ogni risposta sbagliata). Il totale massimo è 40 punti.
Il voto equivale al punteggio finale se minore di 30, è pari a 30 per punteggi da 30 a 35 ed è pari a 30L per punteggi da 36 e 40.

NOTA: parte delle verifiche può avvenire con procedure completamente o parzialmente automatizzate per cui le denominazioni e gli output devono essere rigorosamente aderenti alle indicazioni. Dove sono indicate stringhe con sequenze di “escape” vanno rispettate, ad esempio generare un output tipo “7\n” significa che sono presenti esattamente 2 bytes: quello che rappresenta il carattere “7” e un “a capo” (sequenza “\n”. Il carattere “a capo” può essere rappresentato anche dal simbolo ⏎.

Eventuali irregolarità comportano l’esclusione dalla prova oltre a possibili sanzioni disciplinari.

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`.