[illegible]

TESTO

INDICAZIONI

All'interno della cartella principale creare solo ed esattamente 3 file: un sorgente C "`main.c`" (che dovrà generare un eseguibile denominato "`program`"), un makefile "`Makefile`" e uno script bash "`run.sh`" (eliminare file temporanei, di servizio, generati, etc. prima della consegna!!!).

Realizzare un'applicazione in C che accetti in input esattamente due argomenti nell'ordine qui indicato: il nome di un file con eventuale percorso (relativo o assoluto, argomento "target", di seguito indicato con `<target>`) ed un intero da 1 a 10 (compresi gli estremi, argomento "n", di seguito indicato con `<n>`). Il codice di uscita dell'applicazione deve essere 0 se non ci sono errori, maggiore di 0 altrimenti.

Esempio di chiamata: `./program /tmp/info.txt 3`

L'applicazione deve realizzare le funzionalità indicate più avanti riportando eventuali messaggi d'errore (ad esempio numero argomenti errato o valori non accettabili, errori di I/O, altro) su *stderr* con un codice di uscita maggiore di 0.

1. **[4 punti]** Il Makefile deve funzionare in modo che eseguendo il comando "`make`" senza alcun parametro l'applicazione sia correttamente compilata generando l'eseguibile denominato "`program`", mentre con il comando "`make NAME=...`" la compilazione deve essere eseguita generando un eseguibile con il nome passato come argomento.

   Esempi:
   `make` -> genera eseguibile di nome `program`
   `make NAME=exam` -> genera eseguibile di nome `exam`

2. L'applicazione "`program`" generata compilando "`main.c`" implementa le seguenti caratteristiche:
   a. [**6 punti: OBBLIGATORIO**] scrive nel file `<target>` il proprio *pid* seguito da "`\n`", genera 1 processo figlio che chiamiamo "manager", scrive in `<target>` il *pid* del "manager" seguito da "`\n`". Il "manager" a sua volta genera `<n>` processi figli che chiamiamo "foglie" (nipoti del primo processo). In `<target>` va dunque scritta una riga per ciascuno con il corrispondente *pid* seguito da "`\n`". Il processo principale deve poi terminare, lasciando "manager" e "foglie" attivi (*).
      NB: NON si possono generare più processi di quelli indicati sopra.
      Se si ritiene necessario implementare un metodo di IPC (tra i processi coinvolti) si può utilizzarne uno qualunque a scelta.
   b. [**2 punti**] Il programma dovrà gestire eventuali errori, controllando (con questa precisa priorità):
      - che siano passati esattamente due argomenti, altrimenti il codice di uscita deve essere `3`
      - che `<n>` sia interpretabile come un valore intero nell'intervallo 1..10 (estremi compresi), altrimenti il codice di uscita deve essere `4`

- che <target> non esista già e che sia accessibile il percorso (potrebbe non esserlo, ad esempio, se il percorso è inesistente), altrimenti il codice di uscita deve essere 5

Qualunque altro tipo di errore deve restituire un codice di uscita ≥6.

ES: ./program NON_EXISTING_FILE → codice di uscita 3
ES: ./program NON_EXISTING_FILE 24 → codice di uscita 4
ES: ./program EXISTING_FILE 5 → codice di uscita 5

c. [**4 punti**] Se un processo esterno "X" invia un segnale SIGUSR1 al "manager", una delle "foglie" dovrà essere terminata. Dopo <n> azioni di questo genere (<n> SIGUSR1 inviati) non rimarranno più foglie attive e, a questo punto (con l'n-esimo SIGUSR1), anche il "manager" deve terminare.
d. [**9 punti**] se un processo esterno "X" invia un segnale SIGUSR1 al "manager", tale processo "X" deve a sua volta ricevere un segnale SIGUSR2 da una "foglia" qualsiasi (NON dal "manager").
NB: se il punto 2.c è svolto, la terminazione della "foglia" deve avvenire dopo le sue eventuali operazioni, quali l'invio del segnale SIGUSR2.
e. [**3 punti**] se il "manager" riceve un segnale SIGTERM tutte le "foglie" e il "manager" stesso devono terminare.
f. [**2 punti**] Il "manager" deve bloccare (solo) i segnali SIGALARM, mentre i figli devono bloccare (solo) i segnali SIGCHLD e SIGCONT.

3. [**5 punti**] realizzare uno script bash "run.sh", impostando la prima riga come hash-bang e flag di esecuzione "x", che elimini il file <target> se esistente, e che compili ed esegua l'applicazione passando gli argomenti che riceve a sua volta, stampando successivamente su *stdout* il contenuto di <target> (che conterrà i pid come da punto 2.a) in caso di successo.

   Ad esempio eseguendo ./ run.sh /tmp/info.txt 3 l'applicazione deve essere compilata se necessario, poi essere eseguita come se si fosse digitato ./program /tmp/info.txt 3 (rimuovendo prima /tmp/info.txt se esiste) e poi si deve visualizzare su *stdout* il contenuto del file /tmp/info.txt. Se lo script è eseguito redirezionando l'output su un file questo deve essere correttamente valorizzato (ad esempio con ./ run.sh /tmp/info.txt 3 >/tmp/log.txt l'effetto deve essere che /tmp/log.txt ha lo stesso contenuto di /tmp/info.txt).

(*) l'utente, una volta lanciata l'applicazione dal prompt bash, deve quindi avere a disposizione nuovamente il prompt: il processo principale deve essere terminato e il codice di uscita deve essere valorizzato opportunamente (valore 0 in particolare in caso non vi siano errori) mentre i processi discendenti generati (il figlio "manager" ed i nipoti "foglie") devono rimanere attivi. In caso di successo il file <target> deve quindi contenere più righe: nella prima ci deve essere il *pid* del processo principale (terminato), nella seconda il *pid* del processo figlio "manager" (attivo) e nelle seguenti i *pid* dei nipoti generati (attivi) (le prime due righe DEVONO essere nell'ordine indicato: le seguenti possono essere in qualunque ordine).