Laboratorio SO1 - UINITN - Progetti 2015-2016

INDICAZIONI FONDAMENTALI

(*ulteriori approfondimenti saranno forniti in aula: consultare il messaggio inviato via mail per maggiori dettagli*)

- si svolge in gruppo (particolari eccezioni sono state segnalate singolarmente agli interessati) di due o tre studenti
- deve funzionare almeno su un sistema Ubuntu 14.x di base senza installazioni extra solo da shell (senza interfaccia grafica)
- l'identificazione del gruppo (detto "id") è data dai numeri di matricola separati da un trattino (segno "meno"): ad esempio le matricole 12345 e 67890 formano il gruppo 12345-67890
- è costituito da una cartella, con dei files, denominata "labso1-2'15_2016_"+id-gruppo (es. labso1-2015_2016_ 12345-67890) in cui ci si posiziona con la bash shell per eseguire tutte le operazioni e contenente esattamente:
    - un "Makefile"
    - una sottocartella "src" con i sorgenti in linguaggio C
    - una sottocartella "assets" (eventualmente vuota) con file di supporto (se necessari): ad esempio dei file di "input" già pronti per fare dei test
    - una sottocartella "build" (inizialmente vuota) che conterrà il progetto eseguibile
    - un file README con la descrizione del progetto sintetica (circa una schermata media) con indicate le funzionalità ed evidenziate eventuali scelte particolari
- il Makefile deve contenere almeno le seguenti regole (se ne possono fare altre per comodità o per migliorie varie):
    - help (deve essere il "default"): elenca a video le varie regole disponibili e la loro funzionalità
    - compile: deve compilare i sorgenti (dentro "src") in eseguibili (dentro "build") in modo che il file di ingresso (o l'unico) da lanciare sia denominato come indicato nel progetto specifico
    - clean: deve ripulire eventuali file temporanei o di supporto che possono essere generati durante la compilazione o l'esecuzione del progetto
    - build: deve richiamare "clean" e "compile"
    - test: deve richiamare "build" e poi eseguire il progetto su almeno un caso di test (meglio se su una serie)
- deve essere compilabile con il comando `gcc` senza impostazioni particolari (si può usare solo l'opzione "-o" per scegliere il nome dell'output)
- deve utilizzare solo le librerie standard di base discusse e in particolare:
    - per l'accesso agli argomenti può usare solo `argc` e `argv`
    - può richiamare comandi esterni non per elaborare dati
    - come istruzioni si possono usare solamente:
        - codice c: parole chiavi standard ANSI + librerie standard (stdio.h, string.h, …)
        - bash script: clear, alias, ls, mkdir, rm, cp, echo, cat e altri comandi standard + redirezionamenti e piping
- il codice deve essere ben commentato, in particolare segnalando ogni funzione che azione svolge e che tipo di dati tratta in input e output
- occorre rispettare i requisiti indicati per il progetto specifico che sono minimali: soddisfare i requisiti è necessario per avere la valutazione minima, mentre migliorie aggiuntive e la qualità della soluzione sono elementi considerati per una valutazione superiore
- l'ordine generale (pulizia dei files, commenti, presentazione delle informazioni anche nel file README) può essere elemento di sottrazione o aggiunta di punti nella valutazione finale
- se si implementa una qualche miglioria, in particolare tra quelle suggerite, segnalarlo nel file README in coda al documento in un paragrafo apposito "Migliore": elencare quelle proposte e indicare per ciascuna se è stata implementata o meno, più eventuali altre personali
- ogni progetto proposto ha un "range di valutazione" (da una valutazione minima ad una massima) stimato in base al livello di difficoltà
- si prevede normalmente una "discussione" del progetto stesso (presentazione in aula) salvo la valutazione riesca ad essere effettuata direttamente
- in linea di massima NON è possibile ripresentare lo stesso progetto negli A.A. successivi
- **la consegna avviene creando uno ".zip" dell'intera cartella denominandolo come la cartella stessa con in più il suffisso ".zip" e inviandolo per mail con oggetto ESATTAMENTE "LABSO1-2015_2016: PROGETTO - <nome_cartella>" (es. "LABSO1-2015_2016: PROGETTO - labso1-2015_2016_ 12345-67890") da un solo account del gruppo (NON INVIARE più copie dello stesso progetto), riportando nel corpo della mail l'elenco dei membri del gruppo (nome, cognome, mail diretta di ciascuno e matricola)**
- la scadenza (termine ultimo invio mail) è per il giorno 08/05/2016 ore 23.59)
- entro una settimana circa dalla scadenza sarà inviata una mail di conferma e sarà poi fornita indicazione sulla presentazione del progetto

Progetto 1: merge sort distribuito

range di valutazione: 18-24

Realizzare un'applicazione denominata "`mergesort`" che effettui un "merge-sort" (standard) utilizzando come input un file di testo passato come argomento, contenente un dato per linea almeno di tipo numerico intero, e come output (un dato per linea, ordinati) lo stdout o un possibile file di uscita passato come argomento. Quando un processo ordina un insieme, i due sottoinsiemi in cui si divide devono essere gestiti da due processi figli (e così a cascata fino al limite di uno o due dati) che poi dovranno "passare" i risultati parziale al padre che provvederà al "merge".

Esempio di chiamata, ipotizzando che nell'ordine si passino il file di input e di output:

```
mergesort input.txt output.txt
```

se l'input contiene (uno per riga) i dati 7,3,5,4, in output si dovrà ottenere 3,4,5,7 (uno per riga): al primo step si creano due processi figli in cui il primo lavora sul sottoinsieme 7,3 e il secondo su 5,4, essendo questo già il caso limite in cui non occorre più generare ulteriori processi.

Migliorie suggerite:

- gestire gli argomenti con dei marcatori (ad esempio "-i" per il file di input e "-o" per quello di output) evitando di usare la posizione come filtro
- gestire dati numerici non interi
- gestire dati di tipo stringa
- mostrare il progresso delle azioni (ad esempio la gerarchia dei processi, la quantità di dati elaborati, etc.)
- gestire l'impossibilità di generare nuovi processi (ad esempio perchè troppi) attraverso un'esecuzione diretta

Progetto 1: search distribuito

range di valutazione: 24-30

Realizzare un'applicazione denominata "`splitsearch`" che effettua una ricerca in un insieme di dati almeno di tipo numerico. L'input è un file di testo passato come argomento (un dato per linea), mentre l'output (la/e posizione/i del valore o 0 - zero - se non presente) può avvenire su stdout o su un file di output passato ulteriormente come argomento. La ricerca deve avvenire dividendo in due parti l'insieme dei dati e iterando la chiamata in modo che ognuna sia verificata da un processo figlio (e così a cascata fino al limite di un unico dato da confrontare con il valore cercato).

Esempio di chiamata, ipotizzando che nell'ordine si passino valore, file di input e di output:

```
splitsearch 3 input.txt output.txt
```

se l'input contiene (uno per riga) i dati 7,3,5,4, in output si dovrà ottenere 2 (la posizione del valore cercato, cioè il "3" tra i dati di input): se il valore fosse presente più volte si avrà una lista di risultati.

Migliorie suggerite:

- gestire gli argomenti con dei marcatori (ad esempio "-v" per il valore da cercare, "-i" per il file di input e "-o" per quello di output) evitando di usare la posizione come filtro
- gestire dati numerici non interi
- gestire dati di tipo stringa
- mostrare il progresso delle azioni (ad esempio la gerarchia dei processi, la quantità di dati elaborati, etc.)
- gestire un'opzione aggiuntiva per decidere il numero massimo di risultati da restituire e far terminare immediatamente l'applicazione. In questo caso se si decidesse di impostare come argomento "1" (massimo 1 risultato) e il valore fosse presente anche più volte tra i dati, l'applicazione dovrebbe terminare senza attendere ulteriormente
- gestire l'impossibilità di generare nuovi processi (ad esempio perchè troppi) attraverso un'esecuzione diretta