PROGETTI LabSO1 - AA 2019-2020

INDICAZIONI

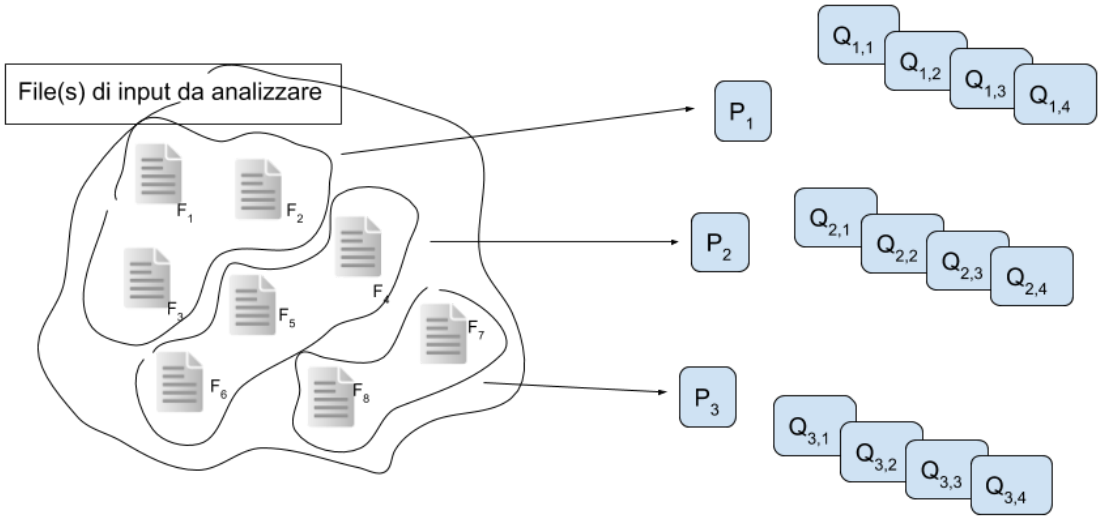
- i progetti devono funzionare correttamente sul sistema operativo Ubuntu 18.04 (modello Docker) o su Raspberry (PI 4b) a seconda della variante di progetto scelta
- quando si parla di "processo" si intende che DEVE esistere un processo "autonomo" (con un suo pid distinto dagli altri) che però può essere affiancato da altri processi di "servizio" aggiuntivi se ritenuto necessario/utile costituendo di fatto un gruppo.
- la generazione dei processi (forking) deve parallelizzare le attività il più possibile limitando al massimo l'eventualità di creare processi "in sequenza"
- si devono evitare processi "zombie"
- le comunicazioni tra processi devono avvenire nel modo più "diretto" possibile favorendo quindi primariamente le "pipe anonime" (quando vi possa essere una gerarchia diretta) e poi nell'ordine - quindi cercando di usare un metodo di questo elenco solo se si ritiene non applicabile nessuno dei precedenti (perché impossibile per le specifiche) - segnali, fifo (named pipe), code, memoria condivisa e file-system.
- gli errori di sistema devono essere tutti gestiti correttamente e non devono verificarsi né avvisi/errori, né crash: in particolare si deve puntare a gestire l'impossibilità di creare nuovi processi (forking) o usare risorse in generale con un feedback all'utente senza interrompere però l'esecuzione (se possibile mantenendo lo stato fino all'eventualità di nuova disponibilità di risorse)
- il progetto deve essere completamente contenuto in un'unica macrocartella con modello di denominazione `LabSO1-AA_2019_2020--xxxxxx_xxxxxx_xxxxxx_xxxxxx` dove le parti `xxxxxx` rappresentano i numeri di matricola dei componenti del gruppo (tanti quanti necessari) che deve contenere almeno:
  - la sottocartella `src` con tutti i "sorgenti"
  - un file `README` con il riepilogo delle informazioni (mettendo nella prima riga il nome della macrocartella stessa, poi un'eventuale mail "di gruppo" collettiva se esistente e poi la lista dei componenti del gruppo con nome+cognome completo, numero di matricola e indirizzo mail di ciascuno. A seguire una brevissima descrizione delle scelte implementative e/o delle peculiarità/difficoltà/etc. del progetto, segnalando cosa eventualmente non si è riusciti a completare del tutto o in parte). Infine le indicazioni (rapide) su come utilizzare/testare il progetto.
  - un "`Makefile`" con almeno le regole (più altre che si ritengono utili/necessarie):
    - `help` (default): mostra un veloce "help" (ad esempio il README)
    - `build`: effettua la compilazione completa
    - `clean`: ripulisce eventuali file temporanei creati in fase di building
- il "make" deve funzionare nell'ambiente di riferimento (modello docker e/o hardware Raspberry a seconda della versione) e l'esecuzione deve avvenire senza avvisi o errori. Il prodotto finito deve trovarsi in una sottocartella "bin" (che può essere creata "al volo")
- la compilazione deve avvenire con il tool `gcc -std=gnu90` a cui eventualmente aggiungere i flag: `-o`, `-c`, `-S`, `-E`, `-I`, `-D`, `-pthread`. Si possono usare le librerie: `assert.h`, `ctype.h`, `errno.h`, `fcntl.h`, `float.h`, `limits.h`, `math.h`, `pthread.h`, `signal.h`, `stddef.h`, `stdio.h`, `stdlib.h`, `string.h`, `sys/ipc.h`, `sys/msg.h`, `sys/shm.h`, `sys/stat.h`, `sys/types.h`, `sys/wait.h`, `time.h`, `unistd.h` (oltre ad eventuali necessarie per l'interfacciamento hardware nel caso del Raspberry)

STATISTICA FREQUENZE CARATTERI

Si vuole realizzare un sistema che gestisce le statistiche della frequenza dei caratteri all'interno di file(s) di testo distinguendo almeno lettere, cifre, spazi, punteggiatura e altri caratteri, eventualmente realizzando anche dei "cluster" (ad esempio computo con distinzione di maiuscole e minuscole oppure aggregati).

Un processo `C` ("Counter") principale tiene traccia dei conteggi (ricevendo quindi i dati) i quali devono essere eseguiti da processi separati da esso generati "P_i", $i \in \{k|1 \le k \le n\}\ k,n \in \mathbb{N}$ che si occupano di gestire ciascuno un sottoinsieme degli input (partizionato in "`n`" sottogruppi) creando a sua volta "`m`" figli "$Q_{i,j}$", $j \in \{k|1 \le k \le m\}\ k,m \in \mathbb{N}$ (che sono gli unici ad accedere effettivamente ai file da analizzare): ogni processo $Q_{i,j}$ analizza uno spezzone di ogni file di input a cui accede. "`n`" ed "`m`" hanno come valore di default rispettivamente 3 e 4 ma possono essere passati come argomenti.

Caso esemplificativo



Ponendo $n=3$ ed $m=4$, si analizzano 8 files che sono suddivisi in 3 gruppi dati in gestione ai processi P_1, P_2 e P_3. Ad esempio P_1 si occupa di analizzare i files F_1,F_2 e F_3 creando 4 sottoprocessi. Ogni file di testo è "spezzato" quindi in 4 parti (dall'inizio del file a un quarto della sua lunghezza, dal carattere successivo fino a metà, dal carattere successivo fino a tre quarti della sua lunghezza e la parte restante): per esempio il file F_1 è suddiviso in $F_{1,1}$, $F_{1,2}$, $F_{1,3}$, e $F_{1,4}$ che sono rispettivamente analizzati dai sottoprocessi $Q_{1,1}$, $Q_{1,2}$, $Q_{1,3}$, e $Q_{1,4}$. Ogni sottoprocesso $Q_{i,j}$ analizza dunque singole parti di files e i suoi conteggi devono essere passati a un gestore (può essere direttamente P_i) che ricompone questi risultati parziali che a loro volta sono passati a un gestore "globale" (che può essere il processo principale).
Ovviamente uno dei gruppi del partizionamento dell'input e una delle parti in cui sono suddivisi gli input potrebbe avere dimensione diversa dagli altri, oltre al fatto che alcuni di questi sottoinsiemi possono essere anche vuoti (se ad esempio si hanno solo due files di input ed n è pari a 3).

I contenuti da analizzare possono essere forniti dando in input direttamente uno o più file oppure una cartella (per cui se ne deve analizzare il contenuto) sia al momento dell'avvio ma anche successivamente comunicando dati aggiornati.

Occorre poter:

- passare dati di input al primo avvio (almeno contenuti da analizzare e parametri `n` ed `m`)
- ricevere un feedback ordinato (ad esempio una "stampa a video" a mo' di tabella) con le statistiche (frequenze dei caratteri e percentuali) e un riepilogo degli input (cartelle/files e conteggi) prevedendo qualche forma di aggregazione in modo da rendere comprensibile il report
- passare nuovi input aggiuntivi, rimuovere o forzare un riconteggio di quelli presenti selettivamente e modificare i valori di `n` ed `m` dinamicamente

Le azioni di analisi (statistica effettiva implementata con il sistema di processi già descritto) e report (richiesta feedback) si dovrebbero svolgere invocando eseguibili distinti che poi interagiscono tra loro.

L'intera applicazione dovrebbe essere avviata da un ulteriore processo generale `M` che funge anche da shell interattiva in cui l'utente può inserire dei comandi testuali per:

- Presentare la situazione al primo avvio
- Visualizzare dei report a video in modalità differenti (statistiche generali, per cluster, altro…)
- Modificare i parametri "on-the-fly"

L'applicazione ideale comprende quindi almeno:

- Un processo/sottosistema "`A`" ("Analyzer") per calcolare le statistiche (che coincide o comprende l'albero descritto con `C`, `P`… e `Q`…)
- Un processo/sottosistema "`R`" ("Report") per recuperare le informazioni
- Un processo/sottosistema "`M`" ("Main") principale per la gestione generale: ha solo funzione di "gateway" e interfaccia-utente. DEVE richiamare gli altri due processi/sottosistemi di seguito indicati (che devono essere utilizzabili anche autonomamente direttamente da cli)

Variante "U": UBUNTU
Realizzare quanto indicato senza variazioni.

Variante "R": RASPBERRY
Rispetto a quanto indicato realizzare un'interfaccia hardware di qualunque genere al posto di "`M`" (o in aggiunta: in tal caso si interfaccia con esso) per poter effettuare le azioni da parte dell'utente (per ogni caso è riportato un ESEMPIO, che vale appunto solo come tale per comprendere meglio la richiesta dove con "trigger" si intende un qualche sistema di input) come da ESEMPI seguenti:

- Richiesta feedback (es.: con un trigger si ha un report dello stato attuale e/o con altri trigger si può scegliere la modalità di feedback)
- Variazione `n` ed `m` (ad esempio con un trigger si sceglie la variabile da modificare e con altri due si incrementa/decrementa)
- Variazione dati input (in questo caso dato che si devono selezionare file/cartelle ci vorrebbe un metodo con feedback visivo a mo' di file-explorer, altrimenti si può valutare una semplificazione - anche se meno valida - per cui tutti i contenuti sono dati in input all'avvio e con dei trigger si possono solo selezionare/deselezionare per conteggiarli o meno)