PROGETTO SO 2020/21

Autori:

Filippo Zaccari 7030220 filippo.zaccari@stud.unifi.it
Elia Matteini 7033383 elia.matteini@stud.unifi.it
Ilaria Catone 7020875 ilaria.catone@stud.unifi.it

Università degli Studi di Firenze
Corso di laurea in Informatica

Data consegna: 23/08/2021

Indice

1 Struttura progetto

Il progetto si presenta con una cartella `progettoSO` che al suo interno contiene:

- Una cartella `SRC`
- Due file: `Avvia.c` e `makefile`

Oltre a questi è presente anche un file `readme` contenente le istruzioni necessarie per l'esecuzione del programma.
Per eseguire il programma è necessario seguire questi passi:

1. Compilare il file `Avvia.c`
2. Eseguire il file generato passando i due parametri richiesti:
    a. La modalità di avvio `NORMALE` o `FALLIMENTO`
    b. Il percorso del `dataset.csv`

Il programma a quel punto attraverso il `makefile` compilerà tutti i file sorgente che sono in SRC, creerà altre quattro cartelle: BIN, LIB, LOG e OBJ, dopodiché sposterà i vari file generati dalla compilazione nelle varie cartelle appena create, di cui i file .o nella cartella OBJ, il file `ConstHeader.h` in LIB e tutti gli eseguibili in BIN.
La cartella log conterrà tutti i file richiesti, quindi `system_log` e `voted_output`, e quelli necessari per l'esecuzione, tra cui il file condiviso `fileP3` usato per scambiare messaggi tra `InputManager` e `P3`, un `filePid` dove vengono salvati i PID dei vari processi, un file `pipeP1` per la pipe tra `InputManager` e `P1`, un file `socketP2` per il socket tra `InputManager` e `P2` ed infine `socketDF` che è il socket usato per scambiare messaggi tra i processi `P1`, `P2` e `P3` ed il `DecisionFunction`.
Per riportare il progetto nelle condizioni iniziali è sufficiente eseguire `make clean`, per rieseguire il programma è necessario rifare i passi indicati sopra.

2 Scopo del progetto

Lo scopo del progetto è costruire un meccanismo di N-version programming per realizzare una procedura di conteggio su file di log.
Il lavoro principale è svolto dai processi `P1`, `P2` e `P3` che svolgono lo stesso compito ma in maniera diversa.
L'`inputManager` legge il file dataset passato all'avvio del programma, scarta la prima riga ed invia le successive ai processi `P1`, `P2` e `P3` in tre modi differenti, rispettivamente con una pipe, un socket e una file condiviso.

Il processo `P1` una volta ricevuta la stringa utilizza la funzione `strtok()` per dividerla in blocchi separati dal carattere "virgola", poi esegue la somma dei valori interi di ogni carattere del blocco e il risultato lo somma al risultato degli altri blocchi.

`P2` invece esegue la somma della stringa partendo dalla fine verso l'inizio scartando il carattere "virgola", cioè scartando il valore intero 44 che è il corrispondente ASCII; `P3` è molto simile a `P2` con la differenza che esegue la somma dall'inizio verso la fine.

Il controllo dei risultati viene svolto dal programma `DecisionFunction` che attraverso un socket riceve i valori delle varie somme, scrive i tre risultati nel file `voted_output`, effettua un voto di maggioranza e in base al risultato scrive SUCCESSO o FALLIMENTO nel file `system_log`.

Nel caso di un successo il programma continuerà la sua routine altrimenti, se si verifica un fallimento, verrà mandato un segnale SIGUSR1 al processo `FailureManager`.
Si verifica un fallimento quando non si ha una maggioranza sui valori delle tre somme, se viene attivata la modalità random failure, questa fa sì che con probabilità 10% il risultato delle somme venga influenzato con una maggiorazione di 10, 20 e 30 (rispettivamente in `P1`, `P2` e `P3`).
Inoltre, prima di determinare SUCCESSO o FALLIMENTO invia un segnale al `Watchdog` per avvisarlo che l'esecuzione sta procedendo.

Il `Watchdog` attende un segnale I_AM_ALIVE con periodo pari a due secondi, che se non viene ricevuto in tempo invia un segnale SIGUSR1 al `FailureManager` che procede a terminare tutti i processi.

`FailureManager` è dotato di un handler per gestire l'arrivo di un segnale SIGUSR1 da parte del `DecisionFunction` che fa terminare tutti i processi.
Inoltre ha un altro handler per il segnale SIGUSR2 che se ricevuto dal `DecisionFunction` vuol dire che il processo `inputManager` ha scandito tutto il dataset e quindi può procedere alla terminazione di `FailureManager` e `Watchdog`.

Il programma `Avvia.c` crea e scrive l'header `ConstHeader.h` che contiene principalmente le varie costanti che indicano i nomi dei vari file, pipe e socket. `Avvia.c` prende come argomenti i due parametri elencati nel punto 1 e con questi valori definisce le costanti PATHDATASET e MODEXEC, che rappresentano il percorso del dataset e la modalità di esecuzione, e li va a scrivere nel file header.
Sempre attraverso `Avvia.c` vengono scritti nell'header i prototipi delle funzioni `savePidOnFile` e `findPid`.
Successivamente procede con la creazione del `filePid` che andrà a contenere i PID di tutti i processi e che viene consultato da: `DecisionFunction` e `Watchdog` per reperire i PID dei processi a cui mandare i vari segnali.
Infine esegue i comandi make clean, make e make install e poi prosegue ad eseguire 7 `fork()` che avviano tutti i vari processi.

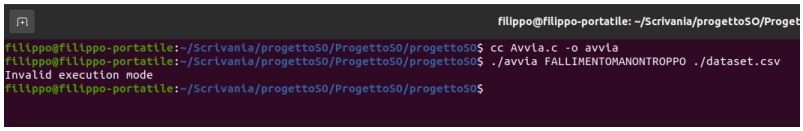
`ConstHeader.c` implementa le funzioni dichiarate nel file `ConstHeader.h`.

3 Specifiche del sistema

Il sistema su cui il programma è stato testato riporta le seguenti caratteristiche:
SO: Ubuntu 20.04.3 LTS
CPU: AMD Ryzen 7 3750H
RAM: 16GB DDR4

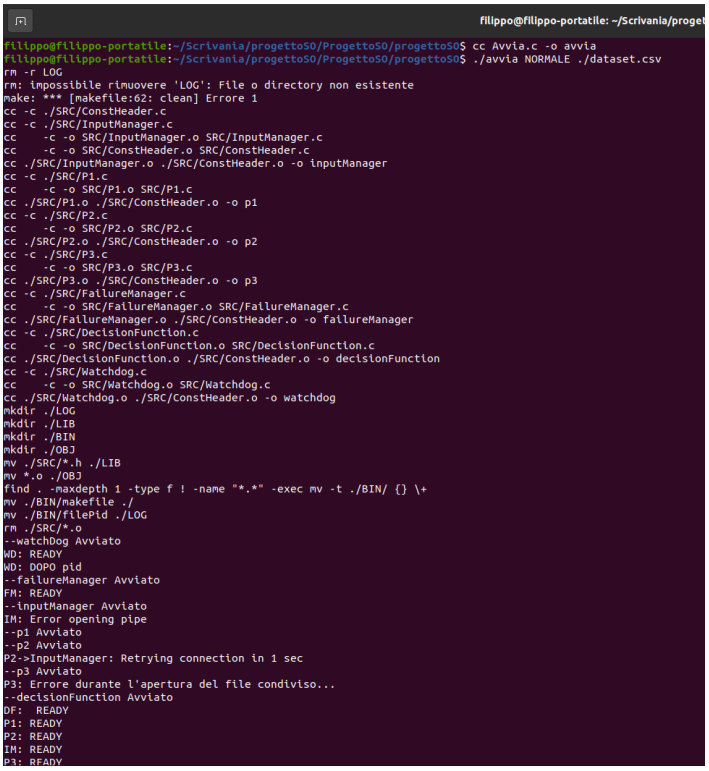
4 Esecuzione

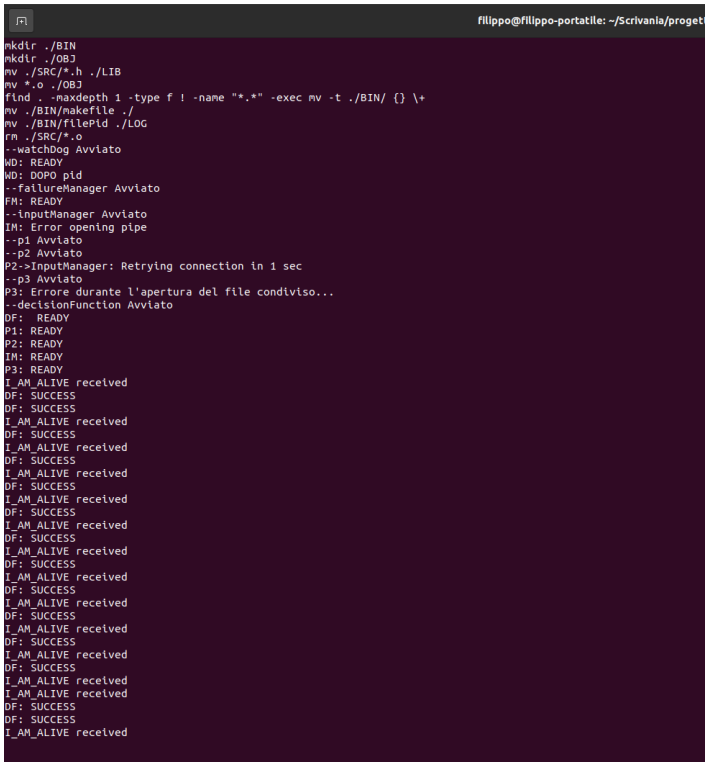
Esecuzione con modalità di esecuzione errata:



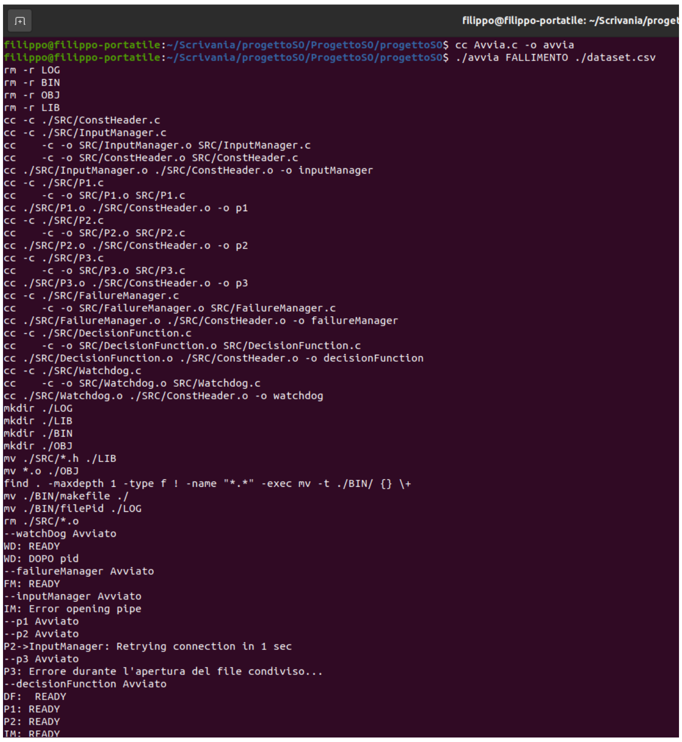
Esecuzione con percorso file .csv errato:

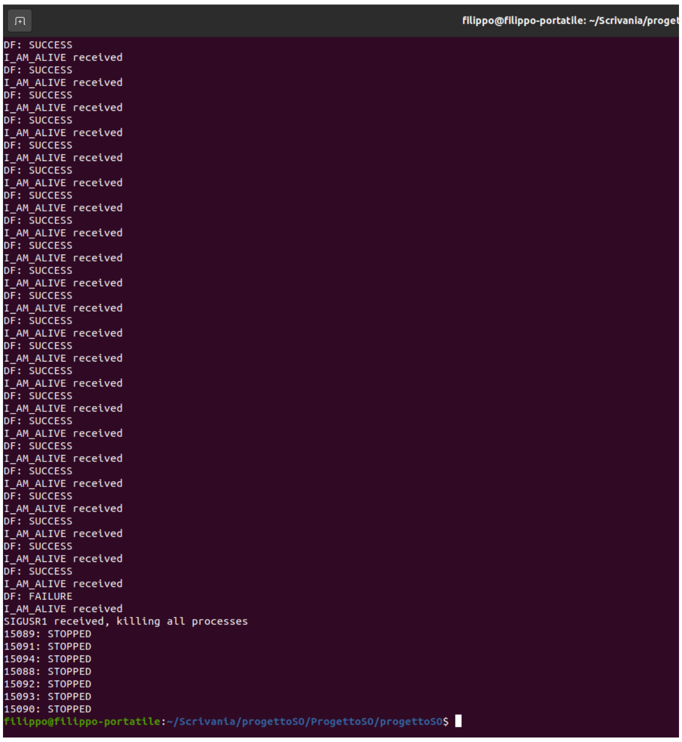
Esecuzione in modalità NORMALE:



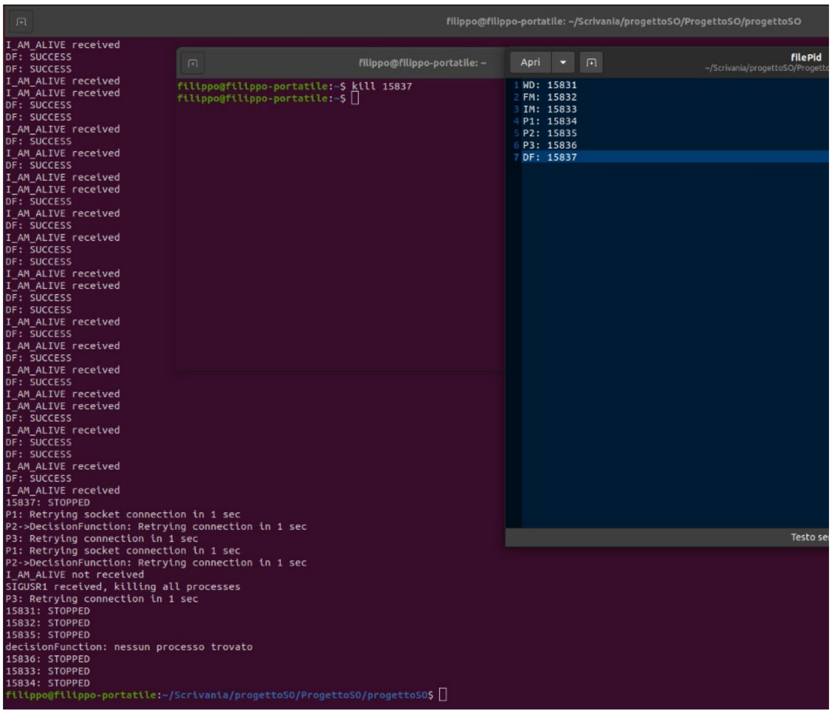


Esecuzione in modalità FALLIMENTO:





Esecuzione con terminazione di `DecisionFunction`: (quindi intervento del Watchdog)



Data consegna: 23/08/2021

5 Elementi facoltativi

Come mostrato nella tabella sottostante, nel programma è stato realizzato uno dei due elementi facoltativi cioè l'invio del segnale I_AM_ALIVE e di conseguenza la realizzazione del Watchdog.

Elemento Facoltativo	**Realizzato (SI/NO)**	**Metodo o file principale**
Invio di I_AM_ALIVE e realizzazione watchdog	SI	Watchdog
Failure Manager comanda il riavvio di P1, P2, P3	NO	

Il `Watchdog` inizializza un allarme di due secondi attraverso la funzione `alarm()` e gestisce due segnali I_AM_ALIVE e SIG_ALARM.
Alla ricezione del primo stampa a video un messaggio e poi resetta il timer; invece, se viene rilevato il secondo segnale vuol dire che il tempo di due secondi è scaduto senza essere stato resettato, quindi c'è qualcosa che non va perché il `DecisionFunction` non sta mandando il segnale I_AM_ALIVE; a questo punto l'handler invia un segnale SIGUSR1 al `FailureManager` che a sua volta terminerà tutti i processi.

6 Scelte progettuali

Costruzione dell'header nel programma:
La costruzione dell'header in `Avvia.c` è stato fatto per evitare di dover passare come parametri nei vari main del programma il percorso del dataset e della modalità di avvio. Costruendo l'header all'interno di avvia scriviamo questi valori come costanti e poi tutti i processi possono attingere a quei valori.

Definizione di `savePidOnFile` e `findPid` nell'header:
Queste due funzioni sono usate molto frequentemente da quasi tutti i processi in maniera identica, di conseguenza, per evitare il più possibile la duplicazione di codice le abbiamo inserite nell'header.

Implementazione del makefile:
L'utilizzo del makefile è dovuto a due cause:

1. Compilazione complessa da eseguire da terminale in quanto ogni sorgente richiede il collegamento con l'header.
2. Strutturare il programma nelle varie cartelle in modo da avere un'organizzazione più pulita e spostare i vari file che seguono la compilazione nelle giuste posizioni.