Relazione Progetto
Sistemi Operativi
Anno Accademico 2019/2020

Akinola Akinjolagbe Samuel Folabi - 5766689 - akinjolagbe.akinola@stud.unifi.it

1 gennaio 2021

Progettazione ed implementazione:

Il progetto si divide in 4 file:

- **pfcD.c** , che rappresenta PFC Disconnect Switch , i cui figli sono i tre processi PFC1, PFC2 e PFC3 e il progesso Generatore fallimenti.
- **transducers.c**, che rappresenta il processo transducers
- **wes.c**, che rappresenta il processo wes
- **principale**, che viene utilizzato per gestire l'esecuzione

Il file Readme.md contiene tutti i dati riguardanti la compilazione e l'esecuzione, Il Progetto è stato programmato su Ubuntu 18.04.5 LTS utilizzando VirtualBox

PfcD.c

Sinteticamente ilsuo funzionamento è il seguente: prende il file in input, estrae i valori importanti e genera i dati per poter calcolare la velocità. Successivamente i 3 PFC, figli del processo padre PFC Disconnect Switch, calcolano la velocità e la inviano a transducers utilizzando i 3 metodi richiesti (socket, pipe e file condiviso). L'ultimo figlio, Generatore fallimenti si occupa di mandare continuamente segnali per "disturbare" l'esecuzione dei 3 PFC. Ovviamente PFC Disconnect switch è anche in grado di terminare tutti i processi del programma.
Piu in dettaglio, all'inizio del codice notiamo i metodi :

- **dati**, estrae i dati importanti dal file in input e li immette nel file "useful.txt"
- **calcoloDati**, elabora i dati li immette nei due vettori "latit" e "longit"
- **converti**, converte da gradi,minuti,secondi a gradi decimali
- **distanza**, calcola la distanza in metri tra due dati(latitudine - longitudine), essendo i dati calcolati a distanza di un secondo, otteniamo anche la velocita in m/s
- **lunghezza**, calcola quanti sono i dati calcolati

I tre processi PFC calcolano la velocità e comunicano con transducers in 3 modi diversi, il tutto reiterato in un "while" fino a quando la totale lunghezza dei dati è stata elaborata(questa è calcolata dal metodo lungehzza). Possiamo notare che questi dormono per 1 secondo e, per fare cio, viene utilizzato il metodo msleep() che implementa nanosleep() cosi da evitare incongruenze nei tempi di processo inattivo. All'interno notiamo anche un "if"" che permette di applicare, quando mod* è settato a 1, la SIGUSR1. Dopo i figli abbiamo l'implementazione di Generatore fallimenti il quale manda i segnali SIGINT, SIGSTOP e SIGCONT i quali utilizzano l'implementazione di default e infine SIGUSR1 che altera il valore del prossimo calcolo della velocità, effettuando un left shift di 2 bits della velocità calcolata una volta arrotondata e, per fare ciò, utilizza i signalHandler* i quali cambiano il valore delle variabili globali mod* e scrivono il tutto nel file "failures.log". Dalla riga 233 riprende la parte di esecuzione del padre (PFC Disconnect Switch), il quale sovrascrive SIGUSR2, SIGINT e SIGCHLD con gli handler :

- **sigdisc**, prende il pid del processo errato(dal file err.txt) e ne controlla lo stato. Per fare ciò controlla il proc/PID/status.
- **sigend**, imposta la variabile globale sigcaugth a 0 (termina il programma).
- **sigend2**, decrementa la variabile globale sigcaugth di 1 (un figlio è terminato). Per fare cio utilizzo anche SA_NOCLDSTOP cosi che non venga notificato se il processo viene bloccato o ripreso.

In seguito scrive i PID dei figli e il suo pid nel file "pids.txt". Dopo ciò il processo padre rimane nel ciclo "while" e ne' uscira solo quando i 3 figli terminano oppure ha ricevuto un segnale di emergenza da wes ovvero SIGUSR2. Successivamente, all'uscita dal "while", termina tutti i processi e termina.

Transducers.c

Transducers permette di ricevere i dati dai vari PFC e di scriverli nei vari file di log. Per fare ciò si utilizza un "while" che reitera ogni 50ms fino a quando non riceve un SIGINT il quale viene sovrascritto con sighandler(cambia il valore della variabile globale sigcaugth). All'interno di questo "while" notiamo che prima di iniziare scrive il proprio pid nel file pids.txt che contiene i PID di tutti i processi, dopo di che scrive semplicemente i dati nei vari file di log, speedPFC*, utilizzando socket AF_UNIX(«socket») e una pipe con nome("aPipe") NON BLOCKING e un file condiviso(log3t.txt).

Wes.c

Wes permette di accedere ai valori di velocità registrati su speedPFC1.log, speedPFC2.log, speedPFC3.log. Se i tre valori sono concordi, invia sullo standard output un messaggio di OK.

Se due valori su tre sono concordi, ed un valore è discorde, invia un messaggio di ERRORE al PFC Disconnect Switch, indicando il processo che ha inviato il valore discorde. Se invece i tre valori sono discordi, invia un messaggio di EMERGENZA al PFC Disconnect Switch. Tutto cio accade dentro il while che reitera una volta ogni 500ms, all'interno di questo come prima cosa ottiene i 4 PID da PFC (tre figli e padre(Pfc Disconnect Switch)) poi aggiunge il suo pid al file "pids.txt" . Dopo aver fatto ciò mette nel vettore "match" l'ultimo valore restituito dai vari speedPFC* (nel caso i processi siano interrotti/sospesi restituiscono rispettivamente -1 -2 -3, questo per evitare che due processi interrotti risultino corretti), per riconoscere se il valore è nuovo si utilizzano gli indici ind* e oldind*. Dopo aver fatto ciò confronta i vari valori per controllare se tutto è corretto, se un processo non è congruo inserisce il PID in err.txt e stampa cio che è successo. Nel caso i 3 valori sono diversi tra loro manda un SIGUSR2 al processo Pfc Disconnect Switch. Il tutto è stampato sia sullo standard output che nel file "status.log".

Principale.c

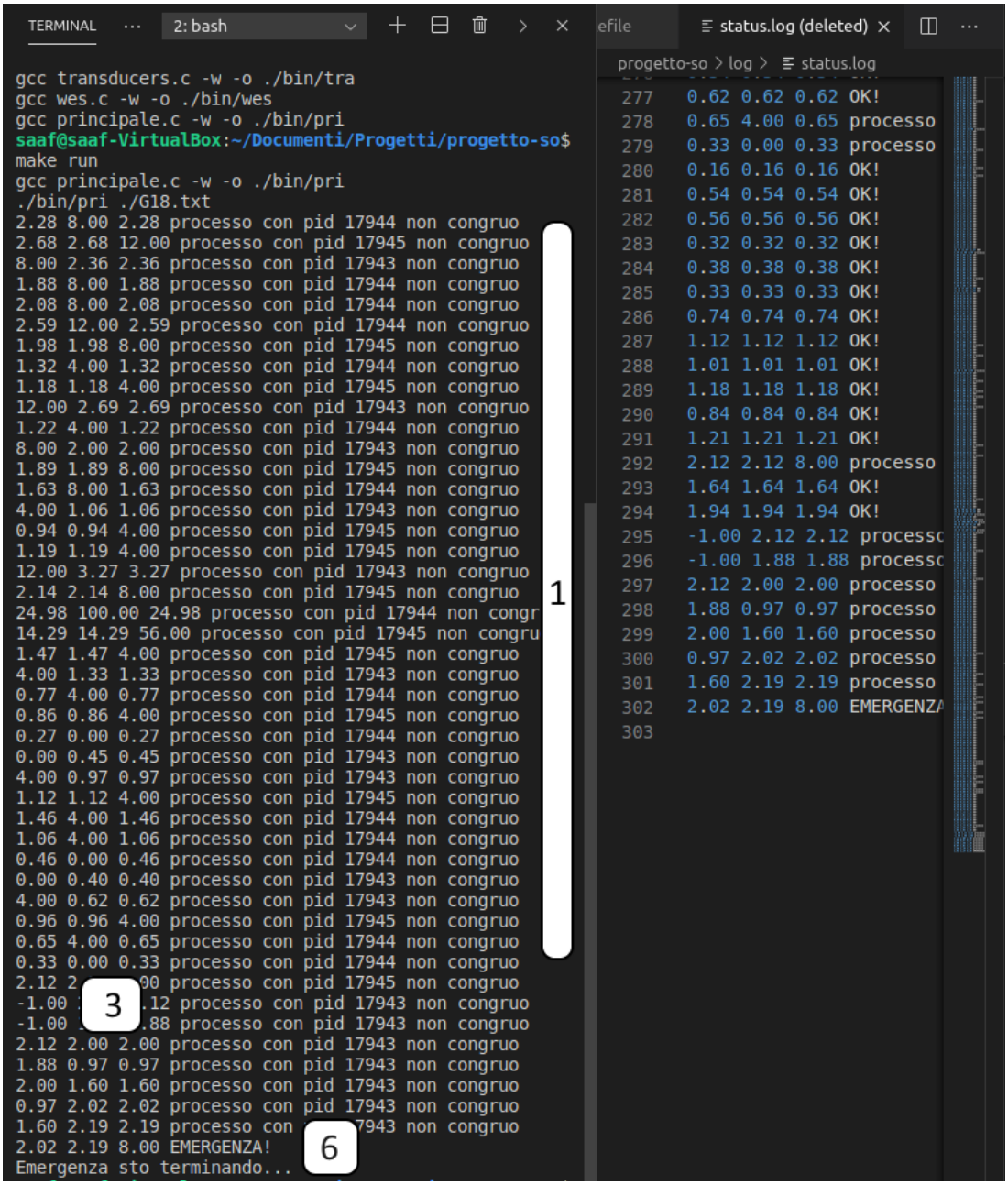
Si occupa dell'esecuzione dei vari programmi.

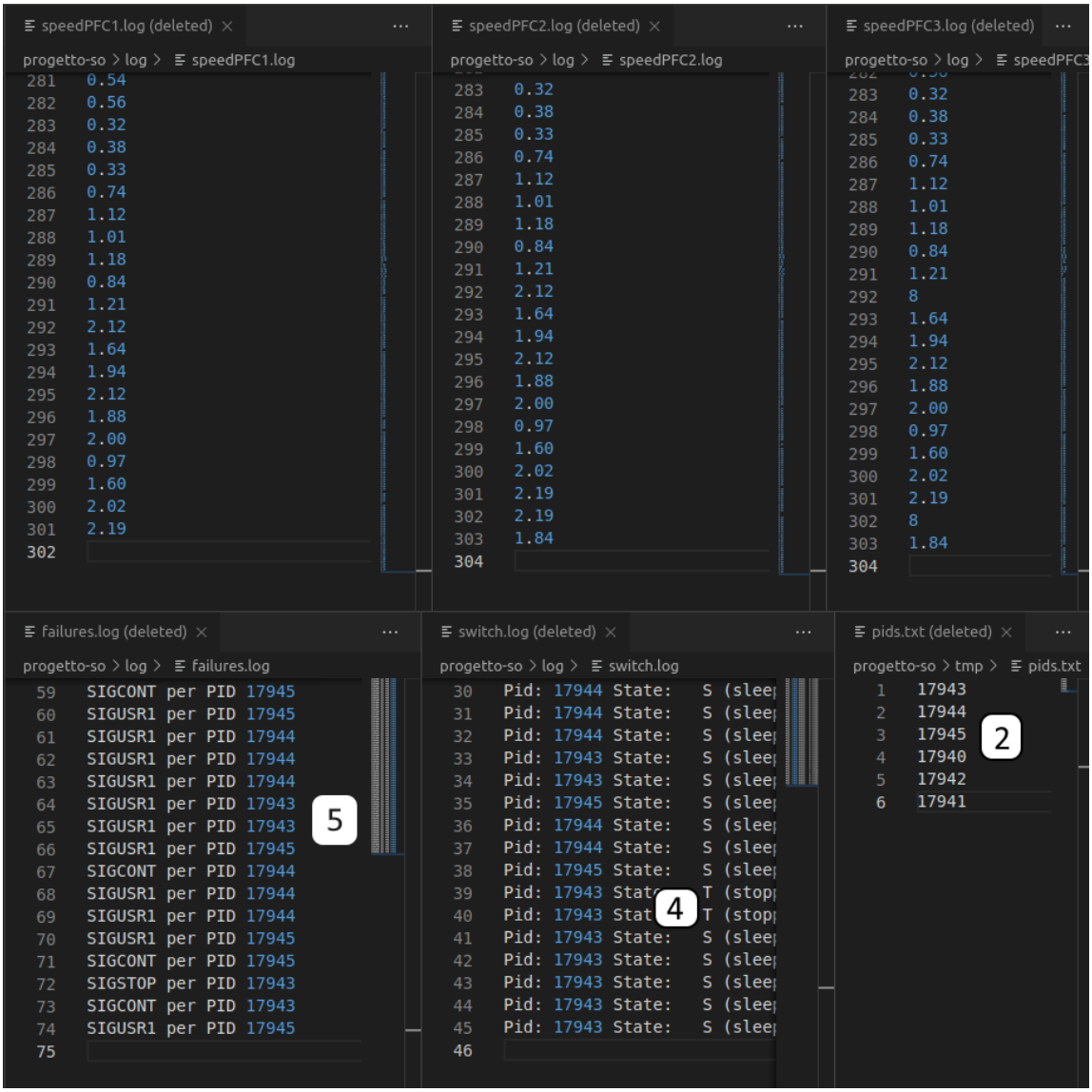
Esecuzione

I vari test sono stati fatti utilizzando il file G18.txt, il makefile permette sia la compilazione che la pulizia. Di default "make run" esegue il file impostato sul makefile ma è anche possibile utilizzare "make run ARGS= ..." sostutuendo ai puntini il percorso del file in input .

```
saaf@saaf-VirtualBox:~/Documenti/Progetti/progetto-so$
make all
mkdir -p bin log tmp
gcc pfcD.c -w -o ./bin/pfc -lm
gcc transducers.c -w -o ./bin/tra
gcc wes.c -w -o ./bin/wes
gcc principale.c -w -o ./bin/pri
saaf@saaf-VirtualBox:~/Documenti/Progetti/progetto-so$
make clean
rm ./bin/*
rmdir bin log tmp
saaf@saaf-VirtualBox:~/Documenti/Progetti/progetto-so$
make all
mkdir -p bin log tmp
gcc pfcD.c -w -o ./bin/pfc -lm
gcc transducers.c -w -o ./bin/tra
gcc wes.c -w -o ./bin/wes
gcc principale.c -w -o ./bin/pri
```

Questo è un esempio di esecuzione in cui possiamo notare i file utilizzati e anche l'output generato dal programma. sto mostrando un caso in cui si parte dal 700esimo dato (per evitare la lunga serie di zeri). Lo standard output per questo esempio è stato cambiato per far si che vengano mostrati solo i casi in cui ci siano dei problemi (nel file status.log vengono comunque mostrati tutti i dati). Vediamo:


TERMINAL
2: bash
gcc transducers.c -w -o ./bin/tra
gcc wes.c -w -o ./bin/wes
gcc principale.c -w -o ./bin/pri
saaf@saaf-VirtualBox:~/Documenti/Progetti/progetto-so$
make run
gcc principale.c -w -o ./bin/pri
./bin/pri ./G18.txt
2.28 8.00 2.28 processo con pid 17944 non congruo
2.68 2.68 12.00 processo con pid 17945 non congruo
8.00 2.36 2.36 processo con pid 17943 non congruo
1.88 8.00 1.88 processo con pid 17944 non congruo
2.08 8.00 2.08 processo con pid 17944 non congruo
2.59 12.00 2.59 processo con pid 17944 non congruo
1.98 1.98 8.00 processo con pid 17945 non congruo
1.32 4.00 1.32 processo con pid 17944 non congruo
1.18 1.18 4.00 processo con pid 17945 non congruo
12.00 2.69 2.69 processo con pid 17943 non congruo
1.22 4.00 1.22 processo con pid 17944 non congruo
8.00 2.00 2.00 processo con pid 17943 non congruo
1.89 1.89 8.00 processo con pid 17945 non congruo
1.63 8.00 1.63 processo con pid 17944 non congruo
4.00 1.06 1.06 processo con pid 17943 non congruo
0.94 0.94 4.00 processo con pid 17945 non congruo
1.19 1.19 4.00 processo con pid 17945 non congruo
12.00 3.27 3.27 processo con pid 17943 non congruo
2.14 2.14 8.00 processo con pid 17945 non congruo
24.98 100.00 24.98 processo con pid 17944 non congr
14.29 14.29 56.00 processo con pid 17945 non congru
1.47 1.47 4.00 processo con pid 17945 non congruo
4.00 1.33 1.33 processo con pid 17943 non congruo
0.77 4.00 0.77 processo con pid 17944 non congruo
0.86 0.86 4.00 processo con pid 17945 non congruo
0.27 0.00 0.27 processo con pid 17944 non congruo
0.00 0.45 0.45 processo con pid 17943 non congruo
4.00 0.97 0.97 processo con pid 17943 non congruo
1.12 1.12 4.00 processo con pid 17945 non congruo
1.46 4.00 1.46 processo con pid 17944 non congruo
1.06 4.00 1.06 processo con pid 17944 non congruo
0.46 0.00 0.46 processo con pid 17944 non congruo
0.00 0.40 0.40 processo con pid 17943 non congruo
4.00 0.62 0.62 processo con pid 17943 non congruo
0.96 0.96 4.00 processo con pid 17945 non congruo
0.65 4.00 0.65 processo con pid 17944 non congruo
0.33 0.00 0.33 processo con pid 17944 non congruo
2.12 2 00 processo con pid 17945 non congruo
-1.00 .12 processo con pid 17943 non congruo
-1.00 .88 processo con pid 17943 non congruo
2.12 2.00 2.00 processo con pid 17943 non congruo
1.88 0.97 0.97 processo con pid 17943 non congruo
2.00 1.60 1.60 processo con pid 17943 non congruo
0.97 2.02 2.02 processo con pid 17943 non congruo
1.60 2.19 2.19 processo con 7943 non congruo
2.02 2.19 8.00 EMERGENZA!
Emergenza sto terminando...
1
3
6
efile
status.log (deleted)
progetto-so > log > status.log
277 0.62 0.62 0.62 OK!
278 0.65 4.00 0.65 processo
279 0.33 0.00 0.33 processo
280 0.16 0.16 0.16 OK!
281 0.54 0.54 0.54 OK!
282 0.56 0.56 0.56 OK!
283 0.32 0.32 0.32 OK!
284 0.38 0.38 0.38 OK!
285 0.33 0.33 0.33 OK!
286 0.74 0.74 0.74 OK!
287 1.12 1.12 1.12 OK!
288 1.01 1.01 1.01 OK!
289 1.18 1.18 1.18 OK!
290 0.84 0.84 0.84 OK!
291 1.21 1.21 1.21 OK!
292 2.12 2.12 8.00 processo
293 1.64 1.64 1.64 OK!
294 1.94 1.94 1.94 OK!
295 -1.00 2.12 2.12 processo
296 -1.00 1.88 1.88 processo
297 2.12 2.00 2.00 processo
298 1.88 0.97 0.97 processo
299 2.00 1.60 1.60 processo
300 0.97 2.02 2.02 processo
301 1.60 2.19 2.19 processo
302 2.02 2.19 8.00 EMERGENZA
303



1. Quello che viene stampato quando un processo non è congruo (nell'insieme vediamo i segnali SIGUSR1).

2. I PID di tutti i processi, rispettivamente primo, secondo, e terzo figlio, padre transducers e wes.

3. Quello che viene restituito quando un processo viene fermato.

4. Quello che viene letto da proc/PID/status.

5. Quello che viene stampato su failures.log ovvero i segnali mandati da generatore fallimenti.

6. Quello che succede quando i 3 dati sono diversi e wes manda un segnale a PFC Disconnect Switch per terminare il tutto.

Elemento Facoltativo	Realizzato (SI/NO)	Metodo o file principale
Utilizzo Makefile per compilazione	SI	makefile
Organizzazione in folder, e creazione dei folder al momento della compilazione	SI	makefile
Riavvio di PFC1, PFC2, PFC3 alla ricezione di EMERGENZA	NO	
Utilizzo macro nel Generatore Fallimenti per indicare le probabilità.	SI	pfcD.c