

Componenti gruppo: Francesco Nocentini, 5288788, francesco.nocentini2@stud.unifi.it

Data consegna: 07-02-2021

Compilazione ed esecuzione del codice

Per compilare il codice è previsto l'utilizzo dell'utility make, quindi è necessario posizionarsi nella cartella Progetto_Nocentini ed eseguire il comando **make**, il makefile compilerà i sorgenti. A schermo verrà stampato il comando della compilazione, la compilazione del file PFC1.c produrrà un warning ("... implicit declaration of function 'accept4'..."). Utilizzare il comando **./PFC_Disconnect_Switch** ed il percorso del file G18.txt per eseguire. Tra una esecuzione e l'altra utilizzare il comando **make clean** in modo da eliminare socket e pipe, eliminare tutti i file di log e txt ed anche i moduli oggetto, successivamente ricompilare ed eseguire come descritto sopra.

Sistema obiettivo

E' stata utilizzata una distribuzione Linux Ubuntu 20.10 a 64-bit installata su Oracle VM VirtualBox Versione 6.1.16 r140961 (Qt5.6.2). L'HW assegnato alla macchina virtuale è composto da 8192 MB di RAM e due CPU. Al sistema operativo sono stati assegnati 86 GB di spazio su HDD.

Elementi facoltativi

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	NO	
Riavvio di PFC1, PFC2, PFC3 alla ricezione di EMERGENZA	NO	
Utilizzo macro nel Generatore Fallimenti per indicare le probabilità.	SI	generatoreFallimenti.c
PFC Disconnect Switch sblocca il processo se bloccato, e lo riavvia altrimenti.	NO	
(continua da sopra) In entrambi i casi, il processo in questione deve riprendere a leggere dal punto giusto del file G18.txt.		

Progettazione ed implementazione

La soluzione da me proposta ricalca quanto visto nella lezione di presentazione del progetto e prevede quindi l'utilizzo di sette processi, PFC_Disconnect_Switch, generatoreFallimenti, PFC1, PFC2, PFC3, transducers ed infine WES. PFC_Disconnect_Switch è il "main" del progetto, questo processo ottiene l'ora corrente e ci aggiunge 2 secondi, l'ora così ottenuta viene inviata ai figli in modo che possono sincronizzarsi ed iniziare la loro esecuzione nello stesso istante, inoltre PFC_Disconnect_Switch ottiene da terminale il percorso del file G18.txt, anche questo viene inviato ai figli e verrà utilizzato da chi ne ha bisogno. A questo punto, dopo l'apertura dei file di cui necessita (il suo file di log, un file in cui salvarci i pid dei tre PFC ed un file in cui leggere i messaggi inviatogli dal WES), tramite un ciclo for vengono eseguite le fork per generare i figli, il

padre appena generato un figlio, tramite la funzione `strtok` ottiene il nome del prossimo figlio da generare e se il figlio è uno dei tre processi PFC ne salva il pid sul file `pid.txt`, sono solo questi tre pid quelli necessari per il funzionamento dell'applicazione. Successivamente il padre ogni 5ms controlla se ha ricevuto un segnale dal processo WES tramite il controllo di una variabile globale, se quest'ultima è true allora deve leggere il messaggio sul file `msg.log`. Se deve leggere il messaggio allora legge i primi sei caratteri per capire se è un messaggio di ERRORE o di EMERGENZA, se il messaggio letto è un messaggio di ERRORE deve leggere anche il nome del processo che ha generato l'errore leggendo i successivi quattro caratteri (prima deve fare una lettura di un carattere per leggere lo spazio tra il nome del messaggio ed il nome del processo), recuperare il suo pid e controllarne lo stato salvandolo in `switch.log` (scrive una stringa comprendente "State:", il suo stato ed il pid del processo) ed infine rimette la variabile globale a false. Il controllo dello stato viene effettuato aprendo come un file `proc/pid/status` e leggendo le righe fino a che non trovo "State:" tramite la funzione `fscanf`. Se invece il messaggio letto è un messaggio di EMERGENZA allora quello che fa è salvare una stringa in `switch.log` in cui dichiara di aver ricevuto il messaggio di emergenza, a questo punto ciclando sui vari pid termina i figli tramite l'invio del segnale SIGINT.

Il processo generatore Fallimenti dopo essersi sincronizzato sulla base del tempo inviato dal padre crea il file `failures.log` in cui ci salverà le azioni che esegue ed apre il file `pid.txt` memorizzando in un array i pid dei tre PFC, successivamente in un ciclo `do while` infinito genera un numero intero casuale compreso tra 0 e 2, se ha generato 0 significa che ha selezionato PFC1, se genera 1 significa che ha selezionato PFC2 altrimenti PFC3, la generazione del numero random è realizzata tramite la funzione `rand()`. Una volta selezionato il processo, sempre utilizzando la funzione `rand` (definita come macro tramite l'utilizzo di `define`) genero la prima probabilità (numero compreso tra 0 e 99 per probabilità di 10^{-2}), se il numero generato è pari ad 1 allora invia il segnale SIGSTOP al processo identificato dal pid scelto in precedenza, una volta inviato il segnale salva una stringa sul file `failures.log` per tenere traccia dell'azione effettuata. Tramite lo stesso procedimento genero le altre probabilità (numero compreso tra 0 e 9999 per probabilità di 10^{-4} , tra 0 e 9 per 10^{-1}) ed invio il segnale salvando l'azione su `failures.log` se il numero generato casualmente è pari ad 1.

I tre processi PFC sono strutturati nello stesso modo, dopo il ciclo utilizzato per la sincronizzazione viene aperto il file `G18.txt`, prima di iniziare la lettura di questo file nel caso di PFC1 viene creata la socket con nome `socketPFC1`, la socket non è bloccante (`accept4`), la connect verrà effettuata dal processo `transducers`. Nel caso di PFC2 viene aperta la pipe (lato scrittura), se l'apertura fallisce riprova dopo 5 millisecondi fino a che non riesce ad aprirla. Nel caso di PFC3 invece viene aperto ed eventualmente creato il file condiviso con `transducers` (`speed.txt`). A questo punto i PFC sono pronti per iniziare a leggere il file `G18.txt`, l'algoritmo è medesimo per questi processi e consiste dopo aver dormito per 1 secondo nella lettura dei primi 7 caratteri, se i caratteri letti corrispondono a "\$GPGLL," allora sono nella riga di mio interesse altrimenti scorro tutta la riga fino a che non trovo il carattere `newline` e rileggo i primi 7 caratteri della nuova riga. Una volta arrivato alla riga giusta devo estrarre le coordinate (funzione `computeSpeed` che utilizza le funzioni `getLatitude` e `getLongitude`), immediatamente dopo i primi 7 caratteri letti ho la latitudine separata da una virgola dagli altri valori della riga, quindi leggo un carattere alla volta convertendolo in intero e salvandolo in un array, se leggo un punto non effettuo la conversione, se leggo una virgola significa che ho letto tutti i caratteri corrispondenti alla latitudine e smetto di leggere. A questo punto ho le cifre che compongono la latitudine, per ottenere il valore della latitudine non devo far altro che sommare queste cifre tra di loro precedentemente moltiplicate per la potenza di 10 corrispondente alla loro posizione, infine la converto nella sintassi necessaria per il calcolo della velocità andando a sistemare il punto che separa le unità dai decimali (dividendo per sei la latitudine). Prima di leggere la longitudine devo leggere un carattere che indica nord o sud e la virgola. La lettura della longitudine e la conversione nella sintassi corretta avviene nello stesso modo della lettura della latitudine. A questo punto posso calcolare la distanza percorsa rispetto alle coordinate precedenti utilizzando la funzione `distanceInMetres`, se però è la prima volta che calcolo la velocità (lo capisco perché in questo caso ho valore 0 assegnato alle variabili che tengono conto dell'ultima latitudine e

longitudine calcolate) allora la distanza è pari a 0, infine se il generatoreFallimenti ha inviato il segnale SIGUSR1 al processo, viene effettuato lo shifting a sinistra di due bit della distanza. Ogni PFC quando riceve il segnale SIGUSR1 dal generatoreFallimenti lo gestisce con un handler personalizzato che imposta una variabile booleana globale (shifting) a true, il controllo del valore assunto da questa variabile mi consente di capire se deve essere effettuato lo shifting (in tal caso viene anche rimessa la variabile a false). Da notare che prendendo il tempo unitario, la velocità corrisponde alla distanza. La velocità viene inviata (convertita in stringa) dai PFC1, PFC2, PFC3 rispettivamente con una write su socket, una write su pipe ed una write su file condiviso. A questo punto non resta altro che leggere i caratteri fino a fine riga e ricominciare il ciclo.

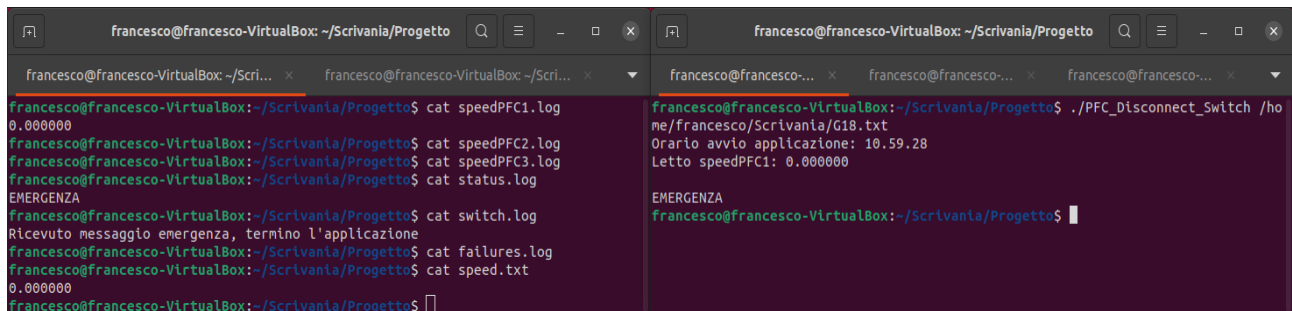
Il processo transducers dopo il classico ciclo per la sincronizzazione, crea i tre file di log in cui verranno salvate via via le velocità inviate dai diversi PFC, inoltre crea anche la socket provando a fare la connect (dormendo 5ms nel caso in cui fallisce la connessione) ed apre il lato lettura della pipe (non bloccante per evitare problemi nel caso in cui PFC2 si interrompe). A questo punto all'interno di un ciclo infinito dopo aver dormito 5ms legge prima il valore inviatogli da PFC1 tramite socket andando a scrivere sul file speedPFC1.log il valore ricevuto, poi legge il valore inviatogli da PFC2 tramite pipe andando a scrivere sul file speedPFC2.log il valore ricevuto ed infine apre il file condiviso (utilizzata in questo caso la funzione di libreria invece della system call per semplicità, l'inserimento di un carattere separatore tra i vari valori generava errori, errori risolti inserendo un valore per riga e letto grazie alla funzione di libreria fgets) effettuando la lettura della riga salvandosi ogni volta il numero dell'ultima riga letta, in questo modo al ciclo successivo posso fare tante letture fino a che non arrivo a leggere una riga nuova. Una volta raggiunta la riga nuova, leggo il valore e lo salvo nel file speedPFC3.log, infine chiudo il file e ricomincio il ciclo di letture.

Il processo WES dopo il ciclo di sincronizzazione crea il file status.log in cui salverà i risultati dei confronti tra i valori che prende dai vari file di log riguardanti le velocità ed inizia un ciclo infinito nel quale dopo aver dormito per 1 secondo legge una nuova riga dal file speedPFC1.log, poi una nuova riga dal file speedPFC2.log ed infine una nuova riga dal file speedPFC3.log. La lettura di questi valori è molto simile alla lettura del file condiviso effettuata dal processo transducers, ho scelto di utilizzare la funzione di libreria per leggere direttamente una riga e quindi un valore tramite la funzione fgets, attraverso la variabile numeroLineaLetto tengo conto dell'ultima riga letta così posso fare tante fgets fino a che non arrivo a leggere una riga nuova, a quel punto ho il valore in una variabile double e devo convertirla in stringa prima di poter passare a leggere il file successivo. Da notare che nel caso in cui quando il WES prova a leggere il file e non è stato aggiunto un valore nuovo (leggo quindi end of file, nessuna nuova riga) allora il valore letto è una stringa scelta da me e tramite il cambiamento di una variabile booleana tengo conto che è stato letto end of file, questo mi permette nel caso in cui tutte e tre le letture non leggono niente di nuovo di andare al prossimo ciclo di letture senza confrontare niente (nessun file ha un valore nuovo da leggere e confrontare), nel caso in cui invece un processo per un qualche motivo non ha inviato un valore nuovo mentre i rimanenti lo hanno inviato, posso generare un errore ed andare quindi come descritto nel processo PFC_Disconnect_Switch a prelevare lo stato del processo in questione (ad esempio può essersi bloccato perché ha ricevuto un segnale SIGSTOP dal generatoreFallimenti). A questo punto posso effettuare il confronto tra i tre valori letti (oppure tra i valori letti e la stringa scelta da me nel caso di end of file) se almeno in uno dei file ho letto un valore nuovo, tramite alcuni else if capisco se due valori su tre sono concordi e quale è quello discorde oppure se sono tutti discordi. In base al risultato dei confronti stampo a schermo una stringa di errore insieme al nome del processo che ha generato l'errore oppure scrivo EMERGENZA oppure scrivo OK, in qualsiasi caso vado a salvare la stringa scritta sullo standard output nel file status.log, nel caso di errore o emergenza salvo la stringa anche sul file msg.log (aperto e chiuso sul momento) in modo che possa essere letto dal processo PFC_Disconnect_Switch ed intraprendere le azioni necessarie. La scelta di utilizzare un ulteriore file deriva dal fatto che su status.log ho anche i messaggi di "OK", grazie all'utilizzo di msg.log ho solo i messaggi di "ERRORE" o "EMERGENZA", quindi posso leggere il file come descritto quando parlo di PFC_Disconnect_Switch senza preoccuparmi di leggere anche le righe di "OK". Dopo aver scritto la stringa sul file msg.log invio un segnale a

PFC_Disconnect_Switch (SIGUSR2) ottenendo il pid di PFC_Disconnect_Switch grazie alla funzione getpid(). La scelta di utilizzare un solo segnale invece che due (uno per errore ed uno per emergenza) è stata fatta per semplicità personale, in quel caso avrei evitato un ciclo ed un controllo in PFC_Disconnect_Switch per capire se il messaggio letto è di ERRORE oppure di EMERGENZA, avrei anche potuto utilizzare due file differenti, uno per le stringhe di ERRORE ed uno per le stringhe di EMERGENZA.

Esecuzioni

Per presentare le esecuzioni mi aiuto con alcuni screenshot al terminale per visualizzare il contenuto dei file di log e di stampe a schermo dei valori letti ogni secondo da parte del processo WES.

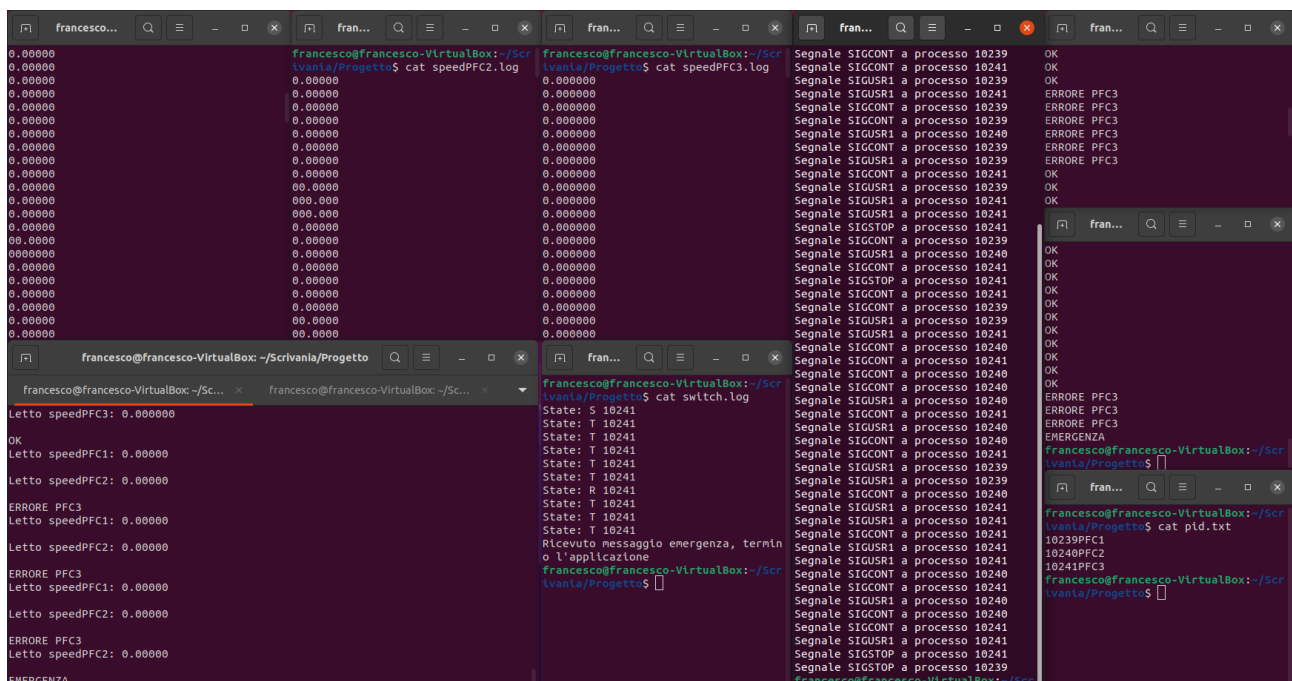


```
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speedPFC1.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speedPFC2.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speedPFC3.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat status.log
EMERGENZA
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat switch.log
Ricevuto messaggio emergenza, termino l'applicazione
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat failures.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speed.txt
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$

francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ ./PFC_Disconnect_Switch /home/francesco/Scrivia/G18.txt
Orario avvio applicazione: 10.59.28
Letto speedPFC1: 0.000000

EMERGENZA
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$
```

In questa esecuzione è stata generata immediatamente una emergenza, transducers è riuscito a leggere il valore inviatogli dal socket, non ha letto nulla dalla pipe e nonostante PFC3 abbia calcolato il valore e scritto nel file condiviso (speed.txt) non è riuscito a leggere il valore; per come è stato sviluppato il WES in questa circostanza manda un messaggio di emergenza al PFC_Disconnect_Switch (due processi su tre sono rimasti indietro quindi tutti e tre discordi). Questa situazione è dovuta all'esecuzione concorrente dei vari processi.



```
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speedPFC2.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speedPFC3.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat switch.log
Ricevuto messaggio emergenza, termino l'applicazione
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat failures.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speed.txt
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$

francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speedPFC3.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat switch.log
Ricevuto messaggio emergenza, termino l'applicazione
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat failures.log
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat speed.txt
0.000000
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$

francesco@francesco-VirtualBox: ~/Scrivia/Progetto$ cat pid.txt
10239PFC1
10240PFC2
10241PFC3
francesco@francesco-VirtualBox: ~/Scrivia/Progetto$
```

In questa esecuzione notiamo dai tre terminali in alto a sinistra che socket e pipe leggono dei valori 0 ma con un differente numero di cifre decimali ed unità, il controllo va comunque a buon fine a causa del fatto che le funzioni definite nel WES leggono il valore come una stringa e poi lo convertono in double per ritornarlo nel main, a questo punto viene nuovamente convertito in stringa per il confronto. Come vediamo dal terminale che visualizza il contenuto del file switch.log abbiamo un primo controllo sul processo con pid 10241 (PFC3 come si vede dal terminale in basso a destra) perché è stato generato un ERRORE PFC3 a causa della lettura dei valori sui file di log di


```
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...

francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...

francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
```

In questa esecuzione invece dopo aver letto il primo valore, il processo 28878 (PFC1) riceve un segnale SIGSTOP e nonostante la socket sia non bloccante (accept4 con flags SOCK_NONBLOCK) non vengono letti nuovi valori inviati da PFC2 e PFC3 come si vede dal terminale in basso a sinistra, il file failures.log ha registrato l'invio di tre segnali dopo il SIGSTOP quindi sono passati almeno tre secondi e dovrei vedere almeno tre ERRORE PFC1 sul file status.log che invece contiene solo un OK. Anche il file switch.log non contiene lo stato di PFC1.

```
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...

francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...

francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
francesco@francesco-VirtualBox: ~/Scrivan...
```

In questa esecuzione ottenuta eliminando le prime 5788 righe del file G18.txt abbiamo che i tre PFC calcolano la velocità e la inviano a transducers, essendo la prima velocità calcolata allora è pari a 0 ed il confronto va a buon fine scrivendo OK su standard output e sul file status.log. Successivamente PFC2 non riesce a causa della concorrenza a calcolare il valore, PFC3 lo calcola e lo invia mentre PFC1 ha ricevuto un SIGUSR1 e quindi invia un valore diverso da quello di PFC3 generando così una EMERGENZA e terminando l'esecuzione.