PROGETTO SULLA MOVEMENT AUTHORITY

06/09/18

Sistemi Operativi 2017-2018

AUTORI:

Iacopo Bonechi 3506491 iacopo.bonechi@stud.unifi.it

Niko Dalla Noce 6134764 niko.dalla@stud.unifi.it

PROGETTO SULLA MOVEMENT AUTHORITY

DESCRIZIONE DELL' ARCHITETTURA

Abbiamo deciso di suddividere il programma su cinque files sorgenti: main.c, etcs1.c, etcs2.c, rbc.c e trainsUtil.c

- **Main**: oltre a permettere all'utente di scegliere tra tre modalità di avvio del programma (etcs1, etcs2, rbc), rappresenta il processo "PADRE_TRENI" a cui viene delegato il compito di creare cinque processi figlio (PROCESSO_TRENO) tramite la system call fork(), che rappresenteranno i nostri treni, e la creazione dei file relativi alle boe (MAx) tutti inizializzati a zero in quanto i treni inizialmente risiederanno nelle relative stazioni.

- **Etcs1**: rappresenta la modalità di avvio del programma in cui i treni si autoregolano grazie alle boe presenti sul percorso, ogni treno prima di tutto salva il proprio itinerario in una struttura dati e ad ogni passo (uno ogni tre secondi) prova ad accedere al segmento successivo relativo al suo percorso leggendo dal file relativo alla boa.
  Se legge uno zero occupa il binario scrivendo 1 sul file e mettendo uno zero su quello precedente altrimenti attende fino a che il binario non viene liberato.

- **Etcs2**: rappresenta la modalità di avvio del programma in cui i treni sono regolati da una stazione radio esterna (RBC), oltre a mettere in esecuzione il PADRE_TRENO e i PROCESSI_TRENO in questo file viene implementata la "Socket" lato client.
  Il "botta e risposta" con il server avviene seguendo i seguenti passi:
  I treni comunicano al server la loro identità, la loro posizione e che binario vorrebbero occupare; il client controlla la boa se non ci sono discordanze con quanto comunicato dal server comunica che sta per occupare il binario e dopo il via libera del server aggiorna le relative boe.

- **RBC**: la stazione radio rappresenta il lato server della Socket è dotato di tre strutture dati, una conserva gli itinerari di tutti i treni, una per monitorare lo stato dei binari e l'ultima per tenere traccia di quanti treni ci sono in ogni stazione.
  Una volta avviato rimane in ascolto per poter soddisfare eventuali richieste, quando il client comunica, controlla se è possibile accedere a quel binario e in caso positivo garantisce l'accesso aspettando una conferma, quando la riceve modifica le relative strutture dati ed è pronto per soddisfare nuove richieste.

- **TrainUtils**: fornisce supporto a tutti gli altri file sorgente contenendo tutti i metodi in comune tra di loro per poter effettuare le operazioni di cui hanno bisogno, tra le quali:

  - Leggere e scrivere sulle boe
  - Creare i file di log
  - Leggere da i file CSV i percorsi e salvarli in un'apposita struttura dati
  - Lock e unlock dei file

SCELTE IMPLEMENTATIVE E ALGORITMI UTILIZZATI

Algoritmo di conversione da CSV a struttura dati

Abbiamo deciso di salvare i tragitti relativi ai processi treno in una matrice di char (array di stringhe), quando nel file si trova la virgola come carattere separatore si salva tutto ciò che la precede in una posizione della matrice, così da avere in ogni posizione gli step del percorso relativo ad un certo treno.

Sincronizzazione temporale dei processi treno

Per permettere la sincronizzazione dei processi abbiamo fissato un tempo limite arbitrario espresso in microsecondi (per avere una maggiore precisione), poi il padre treno calcola il tempo trascorso dall'avvio del programma e lo sottrae al tempo limite ogni volta che genera un figlio ed invia questo risultato al figlio stesso che così saprà quanto deve dormire (usleep) prima di continuare l'esecuzione.

Accesso esclusivo ai file

Ai fini di consentire una corretta concorrenza tra i processi treno abbiamo utilizzato la funzione flock(), che ci ha permesso di garantire l'accesso esclusivo ai file relativi alle boe così quando un treno accede in lettura o scrittura al file gli altri processi che cercano di accedere allo stesso file dovranno attendere fino al loro rilascio.
Questo ci ha permesso di evitare errori di scrittura/lettura dei file in condizione di concorrenza da parte di più processi.

Gestione discordanza informazioni client server

Per garantire una maggiore sicurezza dei treni abbiamo implementato un doppio controllo prima dell'accesso ad un binario.
Il primo avviene da parte di RBC dove viene confrontata la richiesta di etcs2 con la struttura dati relativa ad i binari, susseguentemente da parte del treno dopo un primo ok viene controllato il file MAx relativo a quello stesso binario.
Se non vengono riscontrate discrepanze il treno comunica che sta per occupare il binario e tutte le strutture dati e i file vengono aggiornati.

Creazione socket e comunicazione client server

Abbiamo implementato una socket per permettere la comunicazione tra i singoli treni e la stazione radio (RBC), il dominio utilizzato è AF_UNIX perché risiedono sulla stessa macchina , il tipo di comunicazione è SOCK_STREAM e per protocollo abbiamo scelto DEFAULT_PROTOCOL che seleziona il più opportuno tra quelli disponibili.
In entrambi i lati , server e client, abbiamo inizializzato le strutture dati per permettere la creazione della socket.
Il lato server con la funzione BIND() assegna un indirizzo alla socket, tramite la funzione LISTEN() dichiara quante connessioni è disposto ad accettare (ACCEPT) e in seguito opera sulla socket tramite READ() e WRITE().
Il lato client crea la socket e la collega con il server tramite CONNECT() e anche lui opera tramite le funzioni READ() e WRITE().
Il server rimane sempre in ascolto mentre il client una volta terminata l'esecuzione chiude la socket tramite la funzione CLOSE().

EVIDENZA DEL CORRETTO FUNZIONAMENTO DEL PROGRAMMA

I processi treno in competizione per lo stesso binario sono rispettivamente:

- T1, T4, T5 per il binario MA3
- T2, T3 per il binario MA12

Non possiamo sapere a priori chi occuperà per primo, tra i treni, i binari condivisi in quanto essendo eseguiti in concorrenza ogni esecuzione sarà diversa dalla precedente.

MODALITA ETCS1

In questo esempio possiamo vedere come ogni treno non occupa mai lo stesso binario nello stesso momento e quando ci prova rimane in attesa che venga liberato. Sono evidenziati i tratti salienti del percorso.

T1 LOG:

```
[Attuale: S2][Next: MA5] Fri Jul 27 12:59:38 2018
[Attuale: MA5][Next: MA6] Fri Jul 27 12:59:41 2018
[Attuale: MA6][Next: MA7] Fri Jul 27 12:59:44 2018
[Attuale: waiting][Next: MA3] Fri Jul 27 12:59:47 2018
[Attuale: MA7][Next: MA3] Fri Jul 27 12:59:50 2018
[Attuale: MA3][Next: MA8] Fri Jul 27 12:59:53 2018
[Attuale: MA8][Next: S6] Fri Jul 27 12:59:56 2018
```

T2 LOG:

```
[Attuale: S3][Next: MA9] Fri Jul 27 12:59:38 2018
[Attuale: MA9][Next: MA10] Fri Jul 27 12:59:41 2018
[Attuale: MA10][Next: MA11] Fri Jul 27 12:59:44 2018
[Attuale: MA11][Next: MA12] Fri Jul 27 12:59:47 2018
[Attuale: MA12][Next: S8] Fri Jul 27 12:59:50 2018
```

T3 LOG:

```
[Attuale: S4][Next: MA14] Fri Jul 27 12:59:38 2018
[Attuale: MA14][Next: MA15] Fri Jul 27 12:59:41 2018
[Attuale: MA15][Next: MA16] Fri Jul 27 12:59:44 2018
[Attuale: waiting][Next: MA12] Fri Jul 27 12:59:47 2018
[Attuale: MA16][Next: MA12] Fri Jul 27 12:59:50 2018
[Attuale: MA12][Next: S8] Fri Jul 27 12:59:53 2018
```

T4 LOG:

```
[Attuale: S6][Next: MA8] Fri Jul 27 12:59:38 2018
[Attuale: MA8][Next: MA3] Fri Jul 27 12:59:41 2018
[Attuale: MA3][Next: MA2] Fri Jul 27 12:59:44 2018
[Attuale: MA2][Next: MA1] Fri Jul 27 12:59:47 2018
[Attuale: MA1][Next: S1] Fri Jul 27 12:59:50 2018
```

T5 LOG:

```
[Attuale: S5][Next: MA4] Fri Jul 27 12:59:38 2018
[Attuale: waiting][Next: MA3] Fri Jul 27 12:59:41 2018
[Attuale: MA4][Next: MA3] Fri Jul 27 12:59:44 2018
[Attuale: MA3][Next: MA2] Fri Jul 27 12:59:47 2018
[Attuale: MA2][Next: MA1] Fri Jul 27 12:59:50 2018
[Attuale: MA1][Next: S1] Fri Jul 27 12:59:53 2018
```

MODALITA ETCS2 RBC

Anche dal log del server RBC possiamo notare come ai treni che tentano di passare sopra un segmento già occupato (verde) gli viene negato l'accesso(giallo).

```
[Treno T3][Attuale: S4][Next: MA14][Autorizzato: SI] [Thu Jul 26 17:36:33 2018]
[Treno T1][Attuale: S2][Next: MA5][Autorizzato: SI] [Thu Jul 26 17:36:33 2018]
[Treno T2][Attuale: S3][Next: MA9][Autorizzato: SI] [Thu Jul 26 17:36:33 2018]
[Treno T4][Attuale: S6][Next: MA8][Autorizzato: SI] [Thu Jul 26 17:36:33 2018]
[Treno T5][Attuale: S5][Next: MA4][Autorizzato: SI] [Thu Jul 26 17:36:33 2018]
[Treno T3][Attuale: MA14][Next: MA15][Autorizzato: SI] [Thu Jul 26 17:36:36 2018]
[Treno T1][Attuale: MA5][Next: MA6][Autorizzato: SI] [Thu Jul 26 17:36:36 2018]
[Treno T2][Attuale: MA9][Next: MA10][Autorizzato: SI] [Thu Jul 26 17:36:36 2018]
[Treno T4][Attuale: MA8][Next: MA3][Autorizzato: SI] [Thu Jul 26 17:36:36 2018]
[Treno T5][Attuale: MA4][Next: MA3][Autorizzato: NO] [Thu Jul 26 17:36:36 2018]
[Treno T3][Attuale: MA15][Next: MA16][Autorizzato: SI] [Thu Jul 26 17:36:39 2018]
[Treno T1][Attuale: MA6][Next: MA7][Autorizzato: SI] [Thu Jul 26 17:36:39 2018]
[Treno T2][Attuale: MA10][Next: MA11][Autorizzato: SI] [Thu Jul 26 17:36:39 2018]
[Treno T4][Attuale: MA3][Next: MA2][Autorizzato: SI] [Thu Jul 26 17:36:39 2018]
[Treno T5][Attuale: MA4][Next: MA3][Autorizzato: SI] [Thu Jul 26 17:36:39 2018]
[Treno T3][Attuale: MA16][Next: MA12][Autorizzato: SI] [Thu Jul 26 17:36:42 2018]
[Treno T1][Attuale: MA7][Next: MA3][Autorizzato: NO] [Thu Jul 26 17:36:42 2018]
[Treno T2][Attuale: MA11][Next: MA12][Autorizzato: NO] [Thu Jul 26 17:36:42 2018]
[Treno T4][Attuale: MA2][Next: MA1][Autorizzato: SI] [Thu Jul 26 17:36:42 2018]
[Treno T5][Attuale: MA3][Next: MA2][Autorizzato: SI] [Thu Jul 26 17:36:42 2018]
[Treno T3][Attuale: MA12][Next: S8][Autorizzato: SI] [Thu Jul 26 17:36:45 2018]
[Treno T1][Attuale: MA7][Next: MA3][Autorizzato: SI] [Thu Jul 26 17:36:45 2018]
[Treno T2][Attuale: MA11][Next: MA12][Autorizzato: SI] [Thu Jul 26 17:36:45 2018]
[Treno T4][Attuale: MA1][Next: S1][Autorizzato: SI] [Thu Jul 26 17:36:45 2018]
[Treno T5][Attuale: MA2][Next: MA1][Autorizzato: SI] [Thu Jul 26 17:36:45 2018]
[Treno T1][Attuale: MA3][Next: MA8][Autorizzato: SI] [Thu Jul 26 17:36:48 2018]
[Treno T2][Attuale: MA12][Next: S8][Autorizzato: SI] [Thu Jul 26 17:36:48 2018]
[Treno T5][Attuale: MA1][Next: S1][Autorizzato: SI] [Thu Jul 26 17:36:48 2018]
[Treno T1][Attuale: MA8][Next: S6][Autorizzato: SI] [Thu Jul 26 17:36:51 2018]
```

ISTRUZIONI PER COMPILARE I FILE SORGENTI E LANCIARE IN ESECUZIONE IL PROGRAMMA

Abbiamo creato un file **create.make** per poter compilare tutti i file sorgente, per la compilazione e l' esecuzione è sufficiente digitare da terminale:

Per la compilazione: posizionarsi dentro la cartella compilazione e digitare ***$ make –f create.make***

Per avviare la prima modalità: ***$ ./main etcs1***

Per avviare la seconda modalità:

1. ***$ ./main etcs2 rbc*** in una finestra per avviare il server
2. ***$ ./main etcs2*** nella seconda finestra per avviare il client