Relazione progetto Sistemi Operativi

Informazioni

Autori:
Innocenti Diego, 7047836, diego.innocenti1@stud.unifi.it
Mannucci Riccardo, 7049471, riccardo.mannucci@stud.unifi.it

Hardware utilizzato

OS: Arch Linux x86_64
Host: MS-7788 1.0
Kernel: 5.17.1-arch1-1
CPU: Intel i5-2400 (4) @ 3.100GHz
GPU: Intel 2nd Generation Core Processor Family
GPU: NVIDIA GeForce GTX 1050 Ti
Memory: 11852MiB

Software utilizzato

Shell: bash 5.1.16
Text editor: VSCodium
Compilatore: Using built-in specs. COLLECT_GCC=gcc COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-pc-linux-gnu/11.2.0/lto-wrapper Target: x86_64-pc-linux-gnu Configured with: /build/gcc/src/gcc/configure --enable-languages=c,c++,ada,fortran,go,lto,objc,obj-c++,d --enable-bootstrap --prefix=/usr --libdir=/usr/lib --libexecdir=/usr/lib --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=https://bugs.archlinux.org/ --with-linker-hash-style=gnu --with-system-zlib --enable-__cxa_atexit --enable-cet=auto --enable-checking=release --enable-clocale=gnu --enable-default-pie --enable-default-ssp --enable-gnu-indirect-function --enable-gnu-unique-object --enable-linker-build-id --enable-lto --enable-multilib --enable-plugin --enable-shared --enable-threads=posix --disable-libssp --disable-libstdcxx-pch --disable-werror --with-build-config=bootstrap-lto --enable-link-serialization=1 gdc_include_dir=/usr/include/dlang/gdc Thread model: posix Supported LTO compression algorithms: zlib zstd gcc version 11.2.0 (GCC)

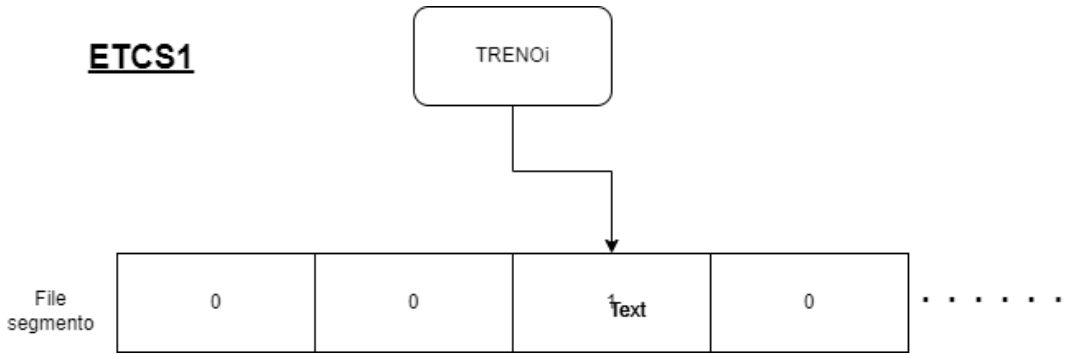
Progettazione ed implementazione

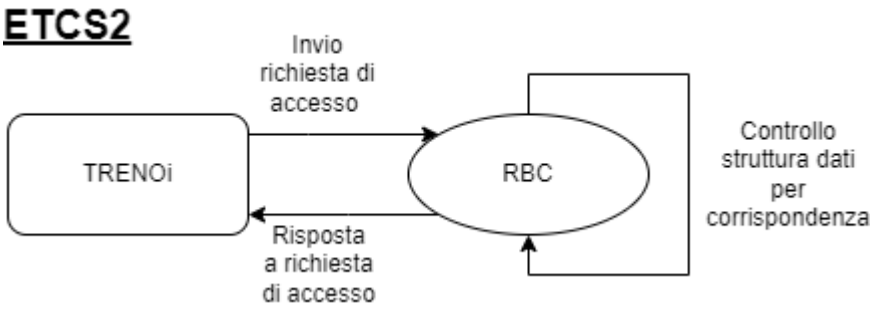
I 4 elementi principali del programma sono:

- PADRE_TRENI è il processo adibito alla creazione dei 16 file (ovvero il numero dei segmenti MAx). Questi file sono impostati per avere accesso in scrittura ed in lettura. All'interno di ogni file vi è un 1 od uno 0, che indicano rispettivamente "segmento occupato" o "segmento libero". In fase di inizializzazione PADRE_TRENI accede in scrittura ad ogni file e vi scrive il valore 0.

- PROCESSI_TRENI sono i 5 treni che seguiranno l'itinerario fornitogli dal processo REGISTRO. Inoltre questi 5 processi sono creati da PROCESSO PADRE.
- REGISTRO gestisce gli itinerari dei treni e li comunica ad ogni PROCESSO_TRENO, inoltre se l'avvio del programma è di tipo ETC2, allora invierà la mappa anche a RBC.
- RBC è un server socket AF_UNIX che amministra le occupazioni di ogni segmento da parte dei treni, i quali fanno richiesta per accedervi.

L'elemento RBC entra in gioco solo se vi è stata un'esecuzione del tipo "ETCS2 RBC". La differenza tra l'avvio con ETCS1 ed ETCS2 è la gestione dei segmenti MAx. In ETCS1 è il PROCESSO_TRENO che controlla lo stato del segmento successivo, andando quindi a leggere il valore all'interno del file corrispondente.



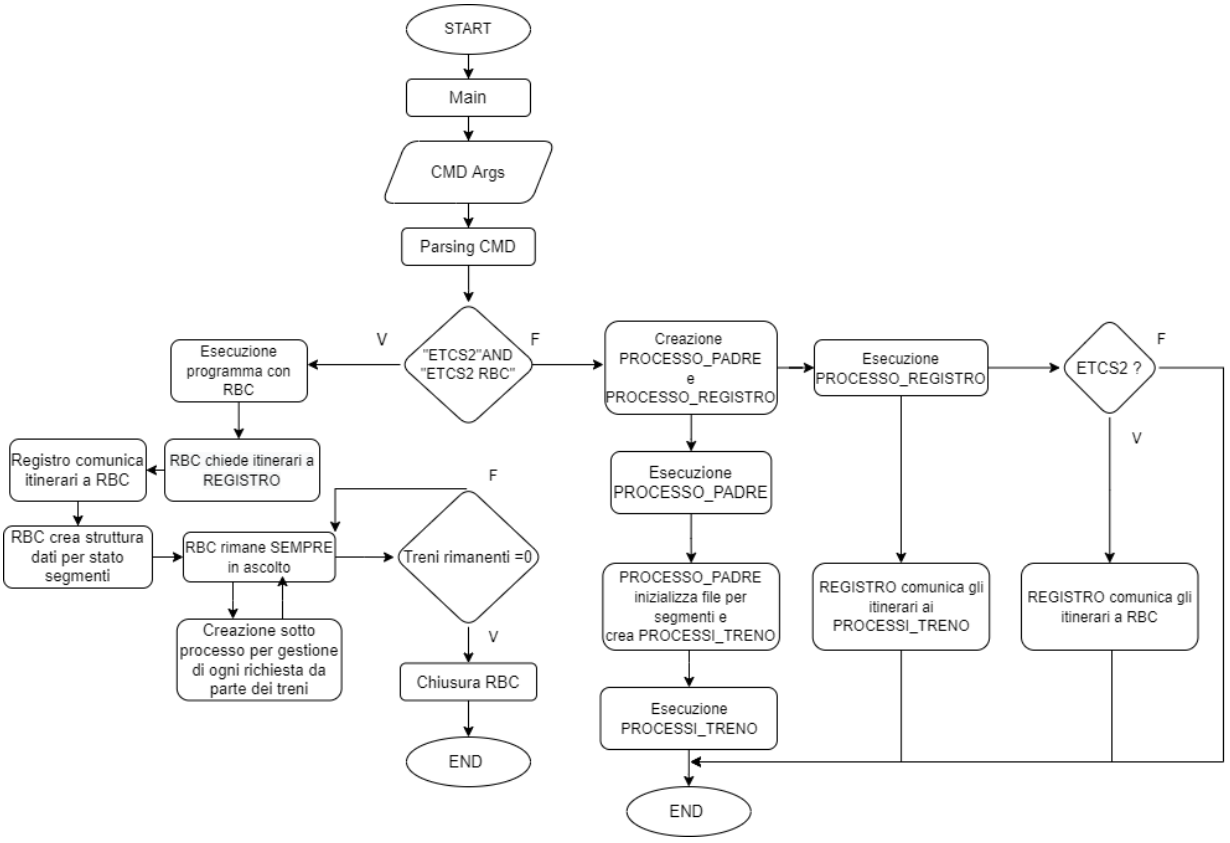


il Main visiona gli argomenti all'interno della CMD, questi argomenti sono:

- l'itinerario (MAPPA1 o MAPPA2);
- il tipo di esecuzione (ETCS1 o ETCS2);
- presenza di RBC.

Se il tipo di esecuzione è ETCS2 o ETCS2 RBC, avviene l'esecuzione del programma con il server socket RBC. Quest'ultimo dopo aver richiesto al REGISTRO l'itinerario di ciascun treno e creato una struttura dati dedicata allo stato dei segmenti, rimane in ascolto in attesa di una richiesta di accesso da parte dei PROCESSI_TRENO. Quando l'RBC riceve una richiesta, crea un sottoprocesso incaricato di gestirla gestirla. La gestione della richiesta e il rimanere sempre in ascolto avviene in maniera parallela.

Se il tipo di avvio, invece, è ETCS1 allora il programma viene avviato solo con l'esecuzione parallela di PROCESSO_PADRE, il quale creerà i PROCESSI_TRENO (t1..t5), e di PROCESSO_REGISTRO, che provvedrà a comunicare gli itinerari a ciascun treno.



<u>Istruzioni dettagliate per compilazione ed esecuzione</u>

Il compilatore adoperato è GCC.
Esso viene utilizzato in concomitanza con i seguenti flag:

```
# flag compilatore
INCL_FLAG = $(addprefix -I,$(INCL_DIR))
CFLAGS = $(INCL_FLAG) -MMD -MP -g
```

La variabile ambientale INCL_FLAG rappresenta le librerie statiche da includere durante la compilazione, esso viene fatto tramite il flag -I seguito dalle directory contenenti gli header file.
La variabile ambientale CFLAGS include tutti i flag utilizzati dal compilatore. Questi sono -MMD e -MP, i quali sopprimono le dipendenze agli header file da parte dei file sorgente nel Makefile; -g, questo flag in fase di compilazione permette di poter vedere il codice sorgente di un eseguibile all'interno di un debugger (es. GDB).

Per la compilazione viene utilizzato un Makefile.

Durante la fase di compilazione viene eseguito questo script: per ogni file sorgente viene creato un suo corrispettivo file oggetto. Il percorso della directory dei file sorgente è SRC_DIR; i file oggetto verranno creati in OBJ_DIR. Lo script in questione, prima di tutto crea la directory per i file oggetto in caso non esistesse, a questo punto compila il file sorgente senza linking prendendo come file di input le dipendenze del Makefile e come output il nome del task del Makefile.

```
# compilazione file sorgente
$(OBJ_DIR)/%.o: $(SRC_DIR)/%.c
	mkdir -p $(dir $@)
	$(CC) $(CFLAGS) -c $< -o $@
```

Durante la fase di linking per ogni file eseguibile c'è un suo corrispettivo task. Il nome del task corrisponde al pathname dell'eseguibile e le dipendenze sono i file oggetto necessari a creare l'eseguibile. Gli eseguibili vengono creati in BIN_DIR, in caso non dovesse essere presente la directory essa viene creata.

```
# creazione eseguibile progetto-so
$(BIN_DIR)/$(MAIN_BIN): $(MAIN_OBJS)
	mkdir -p $(dir $@)
	$(CC) $(MAIN_OBJS) -o $@
```

Esempio di creazione eseguibile

Il comando make (make all) si occuperà di creare tutti gli eseguibili necessari.

```
# usage: make
# creazione tutti file eseguibili
all: $(BIN_DIR)/$(MAIN_BIN) \
$(BIN_DIR)/$(PTRENI_BIN) \
$(BIN_DIR)/$(REG_BIN) \
$(BIN_DIR)/$(TRENO_BIN) \
$(BIN_DIR)/$(RBC_BIN)
```

Il comando make clean si occupa di eliminare file binari ed altri file temporanei.

```
# usage: make clean
# eliminazione file: oggetto, eseguibili, dipendenze, log e segmento
clean:
	rm -rf bin obj log /tmp/MA*.txt /tmp/rbc_server /tmp/reg_pipe*
```

Per l'esecuzione, una volta presi gli argomenti in cmd e creato l'eseguibile, se il tipo di avvio è ETCS1, viene eseguita una sola istanza del programma. Se è ETCS2 vengono invece eseguite due istanze del programma, una dispone dell'argomento RBC, che esegue RBC, e l'altra sprovvista di RBC, la quale esegue PADRE_TRENI e REGISTRO.

Elementi facoltativi

Elemento Facoltativo	Realizzato (SI/NO)	Descrizione e metodo o file principale
implementare soluzioni per gestire letture/scritture concorrenti.	Si	File segmento mappati in memoria
in caso di informazione discordante tra RBC e boe, il TRENO rimane fermo	Si	Contenuto file segm corrente = stato segm corrente rbc_data e contenuto file segm successivo = stato segm successivo rbc_data

Esempi di Output

Esecuzione con ETCS2 (omessi output processi treno escluso TRENO 1) con le due istanze del programma eseguite in modo asincrono. Questo è un esempio di output dello stdout del programma:

```
MAIN     | ETCS2 MAPPA2 RBC=1
RBC      | Inizio esecuzione.
RBC      | Creata SHM rbc_data (fd=3).
RBC      | Tentivo di connessione a data/reg_pipe0.
MAIN     | ETCS2 MAPPA2 RBC=0
MAIN     | Creato processo REGISTRO
MAIN     | Creato processo PADRE_TRENI
REGISTRO         | Inizio esecuzione.
PADRE_TRENI      | Inizio esecuzione.
REGISTRO         | Creato data/reg_pipe0.
PADRE_TRENI      | Inizializzati file segmento
PADRE_TRENI      | Creato processo TRENO 1
TRENO 1 | Inizio esecuzione.
```

Processi RBC PADRE_TRENI e REGISTRO sono creati e cominciano la loro esecuzione. processi treno sono creati da PADRE_TRENI ed entrano in esecuzione.

```
TRENO 1 | Tentivo di connessione a data/reg_pipe1.
REGISTRO         | Creato data/reg_pipe1.
RBC      | Connessione a data/reg_pipe0 (fd=4) stabilita.
REGISTRO         | Aperto data/reg_pipe0 (fd=3).
```

Tentativi di comunicazione tra REGISTRO e gli altri processi.

```
REGISTRO      | Inviata mappa S2-MA5-MA6-MA7-MA3-MA8-S6~S3-
      MA9-MA10-MA11-MA12-S8~S4-MA14-MA15-MA16-MA12-S8~S6-
            MA8-MA3-MA2-MA1-S1~S5-MA4-MA3-MA2-MA1-S1 ad RBC.
REGISTRO         | Chiuso data/reg_pipe0 (fd=3).
RBC      | Ricevuta mappa S2-MA5-MA6-MA7-MA3-MA8-S6~S3-MA9-
      MA10-MA11-MA12-S8~S4-MA14-MA15-MA16-MA12-S8~S6-MA8-
      MA3-MA2-MA1-S1~S5-MA4-MA3-MA2-MA1-S1 da REGISTRO.
RBC      | Connessione a registro pipe (fd=4) interrotta.
```

Invio e ricezione mappa.

Invio e ricezione itinerari. RBC crea server e attende richieste.

```
RBC      | Server creato (fd=4).
RBC      | Server in ascolto. In attesa di richieste TRENO.
TRENO 1  | Connessione a data/reg_pipe1 (fd=3) stabilita.
REGISTRO         | Aperto data/reg_pipe1 (fd=3).
TRENO 1  | Ricevuto itinerario S2-MA5-MA6-MA7-MA3-MA8-S6 da REGISTRO.
REGISTRO         | Inviato itinerario S2-MA5-MA6-MA7-MA3-MA8-S6 di TRENO 1.
REGISTRO         | Chiuso data/reg_pipe1 (fd=3).
```

```
RBC      | Richiesta TRENO accolta (client_fd=5).
TRENO 1  | Locazione corrente: S2, richiesta di procedere alla
             locazione successiva: MA5.
TRENO 1  | Tentivo di connessione ad RBC (fd=3).
RBC      | Server in ascolto. In attesa di richieste TRENO.
TRENO 1  | Connessione ad RBC stabilita (fd=3).
TRENO 1  | Inviato messaggio 1~S2~MA5 identificativo ad RBC.
RBC      | Ricevuto messaggio 1~S2~MA5 da TRENO.
RBC      | Richiesta TRENO accolta (client_fd=5).
RBC      | Creata SHM rbc_data (fd=6).
RBC      | Inviata autorizzazione 1 a TRENO 1.
RBC      | Server in ascolto. In attesa di richieste TRENO.
TRENO 1  | Ricevuta autorizzazione 1 da RBC.
TRENO 1  | Richiesta accolta. Nuova locazione corrente: MA5.
TRENO 1  | Locazione corrente: MA5, richiesta di procedere alla
             locazione successiva: MA6.
TRENO 1  | Tentivo di connessione ad RBC (fd=3).
TRENO 1  | Connessione ad RBC stabilita (fd=3).
TRENO 1  | Inviato messaggio 1~MA5~MA6 identificativo ad RBC.
RBC      | Richiesta TRENO accolta (client_fd=5).
```

Una serie di richieste da parte di TRENO 1 vengono accolte da RBC.

```
TRENO 1  | Terminazione esecuzione.
RBC      | Treni rimanenti: 0.
PADRE_TRENI      | Processi TRENO terminati.
MAIN     | REGISTRO e PADRE_TRENI: esecuzione terminata
RBC      | Eliminata SHM rbc_data.
RBC      | Chiuso server data/rbc_server.
RBC      | Terminazione esecuzione.
```

*Tutti i treni sono giunti a destinazione, di conseguenza RBC termina la sua esecuzi*one.