# Relazione Sistemi Operativi

Matassini Cosimo (7083831)

cosimo.matassini@edu.unifi.it

Becattini Tommaso (7082245)

tommaso.becattini1@edu.unifi.it

Custodi Alessandro (7084103)

alessandro.custodi@edu.unifi.it

Data di consegna: Dicembre 2023

Software utilizzato: Raspberry Pi OS bullseye 64bit (Debian GNU/Linux 11) con versione Kernel 6.1.19-v8+, Linux Ubuntu 22.10, Ubuntu WSL2 su Windows 11, Github Codespaces
Hardware utilizzato: Raspberry Pi 3B, Desktop PC e laptops

- ## Istruzioni dettagliate per compilazione ed esecuzione

  All'interno del progetto è presente il makefile, che permette di gestire la compilazione dei file all'interno di cartelle che vengono create al fine di rendere tutto più chiaro. In particolare, con il comando make install ne vengono create quattro: src, che contiene tutti i file sorgente .c; data, contenente tutti i file sorgente da cui i vari sensori e attuatuori leggono le varie informazioni; bin, che contiene tutti gli eseguibili dei file presenti in src; logs, che contiene tutti i file di log generati dai vari processi. Per ricompilare è possibile eseguire il comando make all, che compila tutti i file *.c e crea gli eseguibili con lo stesso nome dei corrispettivi file sorgente. Il comando make clean esegue l'operazione inversa di make install, cioè rimuove tutte le cartelle spostando i file importanti nella cartella principale del progetto. Per eseguire il file, come suggerito nella richiesta del testo del progetto, è necessario eseguire i programmi ECU e output in due terminali diversi. Eventuali informazioni aggiuntive sono specificate quando si eseguono i programmi (in particolare ECU, che richede l'argomento per scegliere quale file sorgente devono leggere forward facing radar, park assist, surround view cameras).

- ## Progettazione ed implementazione

  Da notare innanzitutto che, siccome non si è reso necessario creare file di importazione, non sono presenti dipendenze tra questi. Il progetto è composto da vari file .c, che contengono il codice sorgente dei vari sensori e attuatori (a parte HMI , la quale viene gestita direttamente come figlio di ECU per prendere i dati da input nel

terminale e surroundviewcameras figlia di parkassist, visto che viene eseguita solo quando quest ultimo è attivo). Il processo principale è ECU, che viene eseguito per primo nel terminale ed è responsabile della creazione di tutti gli altri processi tramite le operazioni di fork, e relative pipe per la comunicazione. L'altro processo che viene avviato dal terminale è output, utilizzato per stampare in tempo reale tutte le operazioni eseguite durante l'esecuzione. All'avvio viene creata una cartella che contiene tutte le pipe e i socket di cui viene fatto uso. Il motivo per cui sono state utilizzate le pipe piuttosto che le socket è il fatto che i vari sensori e attuatori utilizzano una comunicazione unidirezionale con la Central ECU. L'unico caso in cui è stata utilizzata la funzione socketpair() piuttosto che pipe anonime è quello dell'HMI, che deve scambiare dati con la Central ECU in entrambe le direzioni.

- ECU

  Oltre al main, ECU contiene varie funzioni. Alcune di esse sono state definite appositamente per gestire i segnali che sono stati loro inviati. In particolare, una serve per gestire il fatto che quando viene chiuso il programma da terminale con CTRL + C il programma termini correttamente facendo sì che tutti i processi e file aperti si chiudano. SIGUSR1 è il segnale che le può essere inviato da throttle control nel caso in cui questo fallisca (elemento facoltativo spiegato nel dettaglio successivamente): anche in questo caso ECU si deve prendere cura di terminare correttamente il programma. SIGUSR2 è invece il segnale che HMI può inviarle, segnalando la presenza di un comando inviato da input (nel terminale) in modo che la ECU lo gestisca in maniera opportuna. HMI è il processo che viene creato come figlio di ECU tramite fork. Esso sta in ascolto di stringhe inviate come input da terminale e le inoltra a ECU, che poi le gestisce. Il codice vero e proprio di ECU comincia con l'apertura del file di log e l'ingresso in un loop in cui legge i dati che vengono inviati da forward facing radar e front windshield camera. In base al dato letto da quest'ultimo, invia i dati all'attuatore corrispondente effettuando le operazioni richieste dalla traccia del progetto.

- Output

  Apre la pipe in lettura dalla ECU e stampa il contenuto letto nel terminale.

- Brake By Wire

  Apre la pipe in lettura dalla ECU e il suo file di log. Quando legge dalla ECU, vengono formattati in una stringa data e ora e "FRENO 5" per essere scritti sul file di log. È stato definito un gestore del segnale SIGUSR2, nel caso in cui venga inviato dalla ECU nel caso di PERICOLO.

- Steer By Wire

  Apre la pipe in lettura dalla ECU e il suo file di log. Ripetutamente, una volta al secondo, prova a leggere dalla pipe non bloccante. Se non è presente nessun messaggio, scrive "NO ACTION" nel file di log, altrimenti entra in un ciclo ripetuto 4 volte (un'iterazione al secondo) che stampa nel file di log dove sta girando, a destra o a sinistra, in base a quello che ha letto dalla pipe.

- ## Throttle Control

  Apre la pipe in lettura dalla ECU e il suo file di log. Ripetutamente legge dalla pipe e, se non ci sono fallimenti (elemento facoltativo spiegato in dettaglio nella tabella), scrive sul file di log la stringa contenente data e ora, e “AUMENTO 5”.

- ## Forward Facing Radar

  La sorgente da cui leggere (urandom) gli viene passata come argomento al momento della creazione del processo. Apre la pipe in scrittura alla ECU e il suo file di log. Ripetutamente, una volta al secondo, prova a leggere da urandom 8 byte. Se questo avviene, i dati letti vengono inoltrati alla ECU tramite la pipe di comunicazione e al file di log.

- ## Front Windshield Camera

  Apre il file da cui deve leggere (frontCamera.data), la pipe in scrittura alla ECU e il suo file di log. Ripetutamente, una volta al secondo, legge una riga dal file e la inoltra sia a ECU, tramite la pipe di comunicazione, che al file di log.

- ## Park Assist

  Gestisce anche il componente Surround View Cameras, eseguendolo come figlio: comunica con esso tramite pipe anonime. Surround View Cameras apre il suo file di log e prova a leggere da urandom (il cui percorso gli viene passato come argomento da urandom) ripetutamente una volta al secondo. Se i byte letti sono 8, li inoltra a park assist e li scrive nel suo file di log. Park Assist apre la pipe in lettura dalla ECU, e il suo file di log. Per 30 secondi, una volta al secondo, legge 8 byte da urandom e li scrive sul suo file di log. Alla stringa dei dati letti, concatena i dati inviati da surround view cameras e la inoltra alla ECU. Al termine invia un segnale di terminazione anche a surround view cameras.

## Note Aggiuntive

La creazione del figlio steerbywire viene eseguita, diversamente dagli altri, solamente dopo il comando di inizio da input per impedire che, siccome la sua read dalla pipe è non bloccante, scriva nel suo file di log continuamente "NO ACTION" inutilmente.

Siccome tutte le volte che la ECU invia un dato ad un attuatore deve anche scriverlo sul file di log ECU.log e inviarlo al processo output che lo stamperà a schermo, abbiamo deciso di creare una funzione invia() (send, sendto e sendmsg sono già state definite in socket.h) che prende come input il file descriptor al quale eseguire la write e la stringa da inviare.

L’esecuzione di parcheggio è la stessa, sia che venga richiesto dalla HMI, sia che venga richiesto dal file frontCAMERA.data. Ciò che cambiano sono i processi che deve ignorare la ECU a seconda di chi ha richiesto la procedura di parcheggio: quindi, per non duplicare codice abbiamo deciso di creare una funzione parcheggio() che verrà eseguita una volta che ci si è assicurati che i messaggi dai processi da ignorare siano effettivamente “scartati”.

Per quanto riguarda la gestione dell'input dall'HMI, abbiamo deciso di creare un figlio apposito che legge i dati inviati con una read bloccante; nel momento in cui arriva uno tra i due messaggi "ARRESTO" o "PARCHEGGIO", questo invia un segnale al padre, il cui signal handler gestirà la procedura in maniera appropriata. In questo modo, qualunque cosa stesse facendo il padre, il segnale lo interrompe.

Se la central ecu riceve i valori "speciali" da parkassist, la procedura di parcheggio deve essere riavviata. Notare che sarebbe possibile far continuare surround view cameras come se non fosse accaduto niente e far ripartire il ciclo for di parkassist impostando la variabile contatore c a 0. Nonostante ciò, nelle specifiche del progetto è richiesto esplicitamente (o così abbiamo capito noi partecipanti del gruppo) di far ri-avviare la procedura dalla central-ECU, e così abbiamo fatto.

- ## Esecuzione: presentare e commentare una esecuzione tipo del programma

  Dopo aver eseguito il comando make install sul terminale, ognuno dei partecipanti ha testato sul proprio dispositivo (come indicato all'inizio) il programma. In particolare, abbiamo definito insieme di istruzioni da seguire per provare tutte le casistiche possibili che potrebbero verificarsi durante l'esecuzione, in modo da essere sicuri che tutto funzioni senza nessun tipo di problema. Eccole qui:

  1) Testare il programma con argomento NATURALE e aspettare che finisca tutto senza fare niente. Dopo ciò, controllare che tutti i file di log siano popolati correttamente. Controllare la struttura delle cartelle del progetto (che siano quelle che devono essere e non siano state effettuate modifiche, visto che il programma crea ed elimina cartelle). Infine eseguire il comando ps -u nel terminale per controllare che non ci siano processi zombie e tutto sia stato chiuso correttamente.

  2) Testare come il primo punto ma con ARTIFICIALE come argomento.

  3) Testare il programma stoppandolo da input con CTRL+C dopo qualche secondo e controllare che i file di log siano stati correttamente modificati e non abbiano "residui" da esecuzioni precedenti. Allo stesso modo controllare ps -u e le cartelle del progetto.

  4) Testare il programma con ARRESTO da input (magari anche più volte, e inserendo da input stringhe che non sono riconosciute per assicurarsi che effettivamente le scarti) e poi inserire il comando PARCHEGGIO, sempre da input. Dopo ciò, controllare che tutti i file di log siano popolati correttamente, controllare la struttura delle cartelle del progetto, e infine eseguire il comando ps -u per controllare sempre che non ci siano processi zombie e tutto sia stato chiuso correttamente.

  Ulteriori minori verifiche sono state effettuate durante il debugging.

# Elementi facoltativi

| # | Elemento Facoltativo | Descrizione dell'implementazione con indicazione del metodo/i |
|---|---|---|
| 1 | Ad ogni accelerazione, c'è una probabilità di 10^-5 che l'acceleratore fallisca. In tal caso, il componente throttle control invia un segnale alla Central ECU per evidenziare tale evento, e la Central ECU avvia la procedura di ARRESTO | Importando la libreria time.h, abbiamo utilizzato la funzione rand che permette di generare un numero, in particolare compreso tra 0 e 99999 con l'operazione di modulo. Se il numero generato è 0 (ovvero la probabilitá di essere generato è quella specificata) allora invia un segnale SIGUSR1 alla ECU. |
| 2 | Componente "forward facing radar" | Le operazioni di tale componente sono svolte in un processo indipendente che comunica alla ECU le informazioni richieste tramite una pipe. Se i byte letti dalla sorgente (inserita come parametro da input) sono 8, questi vengono inviati come richiesto, altrimenti il processo passa alla lettura successiva. |
| 3 | Quando si attiva l'interazione con park assist, la Central ECU sospende (o rimuove) tutti I sensori e attuatori, tranne park assist e surround view cameras. | All'avvio della procedura di parcheggio, viene inviato un segnale di terminazione a tutti i processi (e chiusi i relativi file descriptor e pipe) tranne brakebywire, che inizialmente serve per fermare l'autovettura. Dopo aver chiuso anche questo, viene creato il processo park assist dalla ECU, che a sua volta crea il processo surround view cameras. |
| 4 | Il componente Park assist non è generato all'avvio del Sistema, ma creato dalla Central ECU al bisogno. | Il processo park assist non viene creato insieme a tutti gli altri, ma solamente all'avvio della procedura di parcheggio specificata sopra. Il procedimento per creare park assist è lo stesso di tutti gli altri. |
| 5 | Se il componente surround view cameras è implementato, park assist trasmette a Central ECU anche i byte ricevuti da surround view cameras. | Quando park assist ha scritto sul suo file di log i dati che ha ricevuto dalla sorgente, concatena nel buffer i dati che ha letto surround view cameras, che gli vengono inviati tramite le pipe anonime; infine, il contenuto del buffer viene inoltrato alla ECU come un messaggio unico di lunghezza massima 16. |
| 6 | Componente "surround view cameras" | Il componente viene creato da park assist tramite fork(). Viene aperto il file di log corrispondente (cameras.log), vengono letti dati dalla sorgente e se questi sono 8 byte, vengono inoltrati sia al file di log che a parkassist. |
| 7 | Il comando di PARCHEGGIO potrebbe arrivare mentre i vari attuatori stanno eseguendo ulteriori comandi (accelerare o sterzare). I vari attuatori interrompono le loro azioni, per avviare le procedure di parcheggio. | Il comando di parcheggio potrebbe arrivare in qualsiasi momento. Se viene inviato dalla HMI in input, viene mandato un segnale SIGUSR2 alla ECU e la funzione che lo gestisce chiama a sua volta la procedura di parcheggio se il messaggio inviato da input corrisponde a PARCHEGGIO. La funzione parcheggio(), come spiegato nei punti precedenti, interrompe i vari attuatori. |
| 8 | Se la Central ECU riceve il segnale di fallimento accelerazione da "throttle control", imposta la velocità a 0 e invia all'output della HMI un messaggio di totale terminazione dell'esecuzione | Quando la ECU riceve il segnale SIGUSR1 da throttle control, viene invocato sigusr1_handler() che imposta la velocità a 0, invia il messaggio della terminazione ad output, termina e chiude tutti i file descriptor e pipe aperti. |