

Chapter 1

“Automotive Test Devices” di LUCA BELLUCCINI

Gli *Automotive Devices* sono due periferici virtuali del sistema.

Lo sviluppo del supporto a basso livello dell'*Automotive API* su Google®Android è avvenuto in due step distinti:

- ottenere l'accesso ad un device a caratteri virtuale, per poter simulare scritture, letture di dati e operazioni specifiche
- modificare tale device con il fine di avere uno strumento di emulazione di un insieme di sensori

Android è basato su *Kernel Linux*, opportunamente modificato dal team di sviluppo. Ad esso sono state apportate ottimizzazioni alla gestione della memoria e risparmio energetico. Gli *Automotive Devices* sono quindi dei moduli caricabili dal Kernel che si comportano non solo da device drivers, ma simulano anche il comportamento dei periferici.

1.1 Premessa sull'HAL di Google®Android

L'*Hardware Abstraction Layer* ha una struttura a più strati. A seconda del tipo, dell'interazione e della complessità del device, possiamo avere:

- un servizio runtime che accede direttamente ad una libreria a basso livello
- un servizio runtime che comunica ad un servizio nativo
- un servizio runtime che comunica ad un demone nativo

La parte nativa comunica infine con i driver, caricati come moduli o built-in nel Kernel.

Una spiegazione approfondita del funzionamento è presente nella sezione Google®I/O Sessions all'indirizzo <http://www.youtube.com/watch?v=G-36noTCaiA#t=49m16s>.

1.2 Simple Automotive Device

Il *Simple Automotive Device* si comporta come un buffer di dimensione fissa. L'accesso è mediato da un semaforo read/write. Su di esso è possibile:

- eseguire le primitive `open`, `close`, `read` e `write`
- svuotare il buffer
- shiftare il contenuto del buffer di un carattere
- riempire il buffer con un carattere
- ritardare la lettura di un valore espresso in ms

Le ultime 4 operazioni sono eseguibili mediante delle chiamate `ioctl`.

Inoltre è possibile cambiare il comportamento del periferico mediante `ioctl` specifica. In tale modalità, chiamata *progressive*, il buffer restituisce un numero long pari al numero di letture effettuate dal device.

La struttura logica del device è la seguente:

```
1 struct automotive_simple_struct {
2     /* Buffer */
3     char data[DEVICE_DATASIZE];
4     /* Semaforo read/write */
5     struct rw_semaphore sem;
6     /* Struttura interna del Kernel per un Char Device */
7     struct cdev cdev;
8     /* Struttura interna base del Kernel per un Device */
9     struct device *pdev;
10    /* Ritardo di lettura */
11    unsigned long delay;
12    /* Switch di modalità */
13    int buffermode;
14    /* Numero di read effettuate */
15    unsigned long progressive;
16 };
```

In fase di `insmod`, il modulo si occupa di inizializzare tale struttura dati. Vengono ottenuti i `major` e `minor number` del device. Infine viene dichiarato l'*owner* e specificate le funzioni per le operazioni su file.

In particolare, è necessario che il device sia presente nel `sysfs` come classe (a se stante o facente parte del tree delle classi). Questo perchè l'*init* di Android prevede una fase di scansione di tutti i device presenti nel tree del `sysfs`:

- se vi sono corrispondenze nella struttura `devperms` del file `/system/core/init/devices.c` dei sorgenti, verrà creato un nodo nel `devfs` con i permessi specificati
- se il periferico non è hardcoded nei sorgenti, verrà creato un nodo sotto `devfs`, ma con i permessi di `root`; è possibile specificare i permessi nel

file di *init*, mediante l'istruzione `device <path> <permessi> <gruppo> <utente>` che può essere utilizzata nella sezione `on early-init`

Il modulo *Simple Automotive Device* è stato ideato per la fase di testing della *API* sull'emulatore. L'architettura simulata dall'emulatore è ARM.

Per mantenere la coerenza dei tipi di dato tra le componenti ad alto livello *JNI* e il modulo del Kernel su tutte le architetture hardware, nell'*Automotive Device* vengono utilizzati tipi di dato a dimensione esplicita (le famiglie `uint` e `ulong`).

1.3 Automotive Device per emulatore

L'*Automotive Device* è uno strumento di emulazione, che rende dinamico il buffer del modulo del Kernel. Tale modulo simula due periferici:

- `automotive-onreq` simula un insieme di valori che, una volta richiesti mediante lettura, vengono ritornati al chiamante istantaneamente (a meno di ritardi)
- `automotive-onchange` simula un insieme di valori che vengono ritornati al chiamante se e solo se sono cambiati dall'ultima lettura effettuata

Dal punto di vista logico, il modulo è un client *TCP/IP* che si connette ad una applicazione server remota scritta in linguaggio *Java*, il *Trace Emitter*. Ogni device virtuale utilizza una porta diversa.

Nella fase di `insmod`, il modulo crea due periferici aventi la seguente struttura:

```
1 struct automotive_struct {
2     /* Buffer */
3     uint8_t *data;
4     /* Lunghezza del buffer */
5     uint16_t datalength;
6     /* Semaforo read/write */
7     struct rw_semaphore sem;
8     /* Struttura interna del Kernel per un Char Device */
9     struct cdev cdev;
10    /* Struttura interna base del Kernel per un Device */
11    struct device *pdev;
12    /* Ritardo di lettura */
13    unsigned long delay;
14    /* Struttura indirizzo remoto */
15    struct sockaddr_in sockaddr;
16    /* Socket di connessione */
17    struct socket *sock;
18    /* Porta del servizio */
19    uint16_t port;
20    /* Indirizzo remoto */
21    uint32_t address;
```

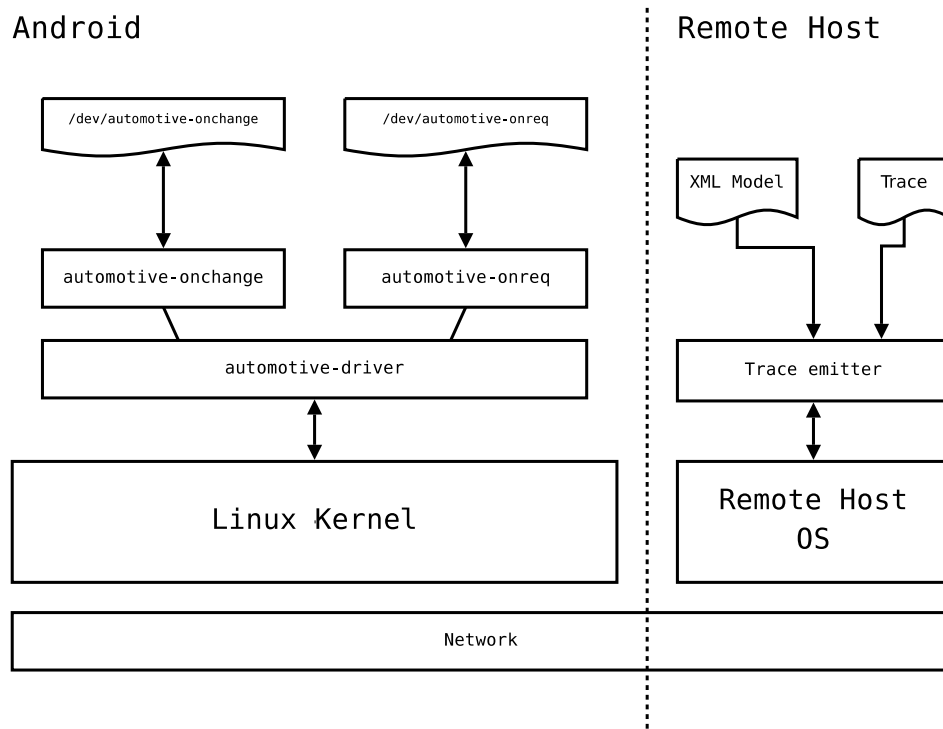


Figure 1.1: Interazione modulo automotive e trace emitter

```

22  /* MD5 del modello XML remoto */
23  uint8_t md5[16];
24  };

```

Mediante un set di `ioctl` è possibile impostare le porte *TCP* per ciascun device e l'indirizzo remoto. Due `ioctl` dedicate si occupano di instaurare o chiudere la connessione. Durante la chiamata `IOCTL_CONNECT`, viene richiesta al server la lunghezza del buffer. Una volta che la connessione è avvenuta:

- ad ogni `read`, una richiesta di lettura viene routata al server mediante un protocollo *fixed length*; a seconda del device connesso (distinguibile dal server dalla porta *TCP* utilizzata), il server restituirà il nuovo buffer immediatamente o solo quando un suo elemento cambia
- ad ogni `write`, una richiesta di scrittura e il buffer vengono inviati al server, che può ignorarla o meno

Il buffer è quindi utilizzato come *status word* dell'insieme di tutti i valori memorizzati nel server. La figura ?? rappresenta una vista dell'interazione tra il

modulo del kernel e il server.

Notare che le connessioni instaurate sono due (una per periferico).

1.4 Commenti e miglioramenti del device

Allo stato attuale, nell'*Automotive Device per emulatore*:

- la trasmissione dati via socket è critica, dato che si sta comunicando senza alcuna system call per l'invio o ricezione dei dati (**send** o **recv**)
- la disconnessione del socket deve essere gestita catturando correttamente dei segnali
- un *kernel thread* o un *daemon user space* potrebbe portare ad una maggiore stabilità