TAG-BASED DATA EXCHANGE

ADVANCED OPERATING SYSTEMS AND SYSTEM SECURITY

Lisa Trombetti 0290810

2020/2021

1. Introduzione

Questa relazione ha come scopo quello di andare ad esporre le caratteristiche fondamentali e le scelte implementative intraprese nella realizzazione del "*Tag-based Data Exchange*", un **sottosistema del kernel Linux** in grado di permettere lo **scambio di messaggi fra threads**.

In particolare, questo servizio si compone di quattro system calls, *tag_get*, *tag_send*, *tag_receive* e *tag_ctl*, di cui si parlerà nel dettaglio nelle sezioni più avanti, e di un device driver implementato per controllare lo stato corrente dei tag.

Per informazioni più dettagliate sulle specifiche del progetto fare riferimento al seguente link:

https://francescoquaglia.github.io/TEACHING/AOS/PROJECTS/project-specification-2020-2021.html

2. Architettura

L'architettura del sistema si compone di cinque moduli fondamentali:

- **usctm**: si occupa dell'inserimento delle nuove system calls
- **service**: implementa le system calls tag_get, tag_send, tag_receive e tag_ctl
- **tag**: gestisce la creazione, rimozione e l'invio dei messaggi sui tag
- **level**: implementa una lista rcu di livelli
- **driver**: implementa un semplice device driver che permette di controllare lo stato dei tag e dei loro livelli

Il codice relativo ad ognuno dei moduli sopracitati è possibile trovarlo nei rispettivi file *.c contenuti nella directory *lib*.

È inoltre possibile configurare il numero di tag, il numero di livelli e la dimensione massima dei messaggi tramite le costanti definite nel file "*config.h*" presente nella root del progetto. Di default è previsto che il software gestisca 256 tags, di cui ognuno è caratterizzato da 32 livelli e può inviare e ricevere messaggi fino a 4 KB.

2.1. Usctm

Questo modulo **implementa una discovery a runtime della posizione della syscall table** e delle sue voci libere, ovvero quelle che puntano a sys_ni_syscall. Tale codice è stato ispirato da quello

presente nella repository Github riportata nel link di seguito ed opportunamente modificato per l'inserimento di quattro nuove system calls.

https://github.com/FrancescoQuaglia/Linux-sys_call_table-discoverer

2.2. Service

Il modulo service va ad implementare il corpo delle system calls che gestiscono il servizio di data exchange. Esso si basa sulle funzioni presenti nel modulo tag, che a sua volta è basato sul modulo level per la gestione dei livelli.

int tag_get(int key, int command, int permission)

Tramite la funzione tag_get è possibile creare o aprire un tag specificandone la chiave ed i permessi. Per ottenere un servizio privato, ovvero in modo che non sia possibile aprirlo nuovamente una volta creato, basta specificare come key IPC_PRIVATE. Se invece si intende rendere possibile a qualsiasi utente di operare sul tale tag basta inserire come valore di permission -1.

Passando 1 come valore di command verrà creato un nuovo tag service solo se la chiave ha un valore non negativo e non è già presente nel sistema un altro tag con la stessa key.

Passando invece 2 come valore di command avverrà un tentativo di apertura del tag service con la chiave specificata.

In entrambi i casi se si avrà successo verrà ritornato il descrittore del tag, necessario per ogni altra operazione che si intenda effettuare sul servizio.

Int tag_send(int tag, int level, char *buffer, size_t size)

La funzione tag_send permette di inviare ad uno specifico tag su uno specifico livello il messaggio di taglia size contenuto in buffer (è consentito anche l'invio di messaggi vuoti).

Tale messaggio viene copiato dallo user space tramite la funzione *copy_from_user* ed inviato a tutti i threads in attesa di riceverlo, se presenti, o altrimenti viene semplicemente scartato.

int tag_receive(int tag, int level, char *buffer, size_t size)

La funzione tag_receive pone il thread che l'ha invocata in attesa finché non viene inviato il messaggio per cui sta aspettando oppure viene ricevuto un segnale.

Se la ricezione del messaggio è andata buon fine, questo verrà passato all'utente tramite la funzione *copy_to_user*.

int tag_ctl(int tag, int command)

Tag_ctl permette sia di rimuovere un tag che di svegliare tutti i threads ancora in attesa di un messaggio per ogni livello presente in esso.

È da tenere in considerazione che per l'eliminazione di un tag è necessario che nessun thread sia in attesa su quel servizio.

Per svegliare tutti i threads in attesa utilizzare il valore 3 per il parametro command, mentre per la rimozione del tag usare il valore 4.

2.3. Tag

Il modulo tag fornisce un insieme di funzioni per creare, aprire, eliminare, svegliare, inviare e ricevere messaggi su un tag service. **Le caratteristiche di ogni tag vengono incapsulate all'interno della struttura *tag_t*** definita in *"/include/struct.h*". Essa è caratterizzata dai seguenti campi:

- **key**: la chiave associata al tag
- **private**: se il servizio è privato ha valore 1
- **perm**: uid dell'utente che ha creato il tag
- **used**: numero di thread che stanno usando il servizio
- **removing**: pari ad 1 se il servizio sta per essere rimosso
- **lv_head**: head della lista dei livelli del tag
- **lv_lock**: lock utilizzato per scrivere sulla lista dei livelli

I primi due campi vengono utilizzati nelle funzioni *open_tag* ed *insert_tag* che permettono rispettivamente di aprire un tag service e di crearne uno nuovo.

Quando un nuovo tag viene creato, viene allocata una nuova *struct tag_t*, che va ad essere inserita in uno slot vuoto dell'array *tags*. Quest'ultimo è semplicemente una lista con numero di elementi pari a *MAX_TAGS*, come specificato nel file di configurazione. Se un tag con la stessa chiave già esiste oppure tutti gli slot di *tags* sono stati occupati, la funzione *insert_tag* ritornerà -1, mentre se avrà successo verrà ritornato il descrittore del nuovo tag, ovvero la posizione i di *tags* in cui è stato inserito.

Per quanto riguarda invece l'apertura del tag, viene controllata non solo la presenza o meno di un servizio con la chiave ricercata, ma anche il campo *private* ed i permessi dell'utente che sta cercando di aprirlo. Nel caso in cui l'utente non sia idoneo oppure il servizio sia stato definito come privato, la funzione *open_tag* ritornerà -1, altrimenti sarà restituito il descrittore del tag.

I **controlli sul tag service** non vengono svolti solo in caso di apertura ma anche **per ogni** altra **operazione si intenda compiere su di esso**. A tale scopo sono state implementate due funzioni: *check_tag* e *uncheck_tag*. La prima ha il compito di verificare se il servizio passato in input è presente o meno in *tags* ed in caso positivo controllerà anche se si dispone dei permessi necessari. Se il campo *removing* del tag è impostato ad 1 allora il check ritornerà -1, in quanto si vuole evitare che un nuovo thread effettui una qualsiasi operazione su un servizio che sta per essere rimosso, in caso contrario invece il numero di utilizzatori del tag rappresentato dal campo *used* verrà incrementato. La seconda funzione ha invece come scopo quello di decrementare il numero di utilizzatori del servizio e verrà quindi chiamata alla fine di ogni operazione sul tag.

Solo la funzione *delete_tag* ha la possibilità si impostare il valore di *removing* ad 1 quando tenta di eliminare il servizio. Se la rimozione dovesse fallire per qualche motivo, per esempio perché ci sono ancora dei thread in attesa, *removing* verrà nuovamente posto uguale a 0.

Infine, gli ultimi due campi definiti nella struttura *tag_t* sono utilizzati nelle operazioni che riguardano la lista dei livelli associata a ciascun tag. Per la ricezione dei messaggi è stata definita la funzione *wait_tag_message* che va a cercare il livello richiesto nella lista definita da *lv_head*, eventualmente inserendolo se non presente, e si mette in attesa. Come è stato già specificato il thread può essere svegliato sia dall'invio di un messaggio che dal comando AWAKE_ALL di tag_ctl. Perciò, è stata definita un'unica funzione per entrambe le operazioni: *wakeup_tag_level*. Passando

al parametro *level* il valore -1 verranno svegliati tutti i thread in attesa su uno qualsiasi dei livelli gestiti da quel tag service.

2.4. Level

Il modulo level **utilizzando la RCU API del kernel Linux va ad implementare una lista** i cui elementi sono definiti tramite la struttura *level*_t in *"/include/struct.h*". Ogni livello è caratterizzato da:

- **num**: numero del livello
- **message**: puntatore al messaggio da inviare
- **threads**: numero dei threads correntemente in attesa
- **wq**: wait queue

Durante la creazione di un nuovo livello, tramite la funzione *insert_level*, viene inizializzato il valore di *threads* a 0 e quello di *message* a NULL. Inoltre, viene allocato lo spazio per una nuova wait queue, utilizzata dai threads per attendere l'invio del nuovo messaggio (la condizione di attesa è: *message != NULL)*.

Quando un nuovo thread si mette in attesa su un livello con la funzione *wait_for_message*, il corrisponde valore di *threads* viene aumentato atomicamente di 1 segnalandone la presenza. Per far si che il processo possa essere risvegliato anche da un eventuale invio di un segnale, si è scelto di usare la funzione *wait_event_interruptible*. In ogni caso, sia per l'avvenuta ricezione del messaggio che per la presenza di un segnale, il valore di threads verrà decrementato di 1 prima di ritornare dalla funzione.

Le funzioni *wakeup_all e wakeup_level* hanno un funzionamento simile, in quanto vanno entrambe a risvegliare tutti i threads in attesa su una wait queue, ma il primo riguarda tutti i livelli di un tag ed il secondo è specifico solo del livello passato in input. Una volta che tutti i thread hanno ricevuto il messaggio o sono stati svegliati, viene utilizzata la funzione *replace_level*, che crea un nuovo livello vuoto sostituendo quello già presente nella lista.

Sono state inoltre create due funzioni per la rimozione dei livelli, *cleanup_levels* e *force_cleanup*, con l'unica differenza che la prima ritorna un errore nel caso in cui ci siano threads in attesa e la seconda no.

2.5. Driver

Questo modulo si occupa di implementare un semplice device driver che permette di controllare lo stato del servizio in termini di tag, livelli e threads in attesa. Il device file corrispondente viene aggiornato sfruttando una funzione del modulo tag, *tag_info*, che scorre tutti i servizi presenti in *tags* ed aggiunge una nuova riga di informazioni per ogni livello.

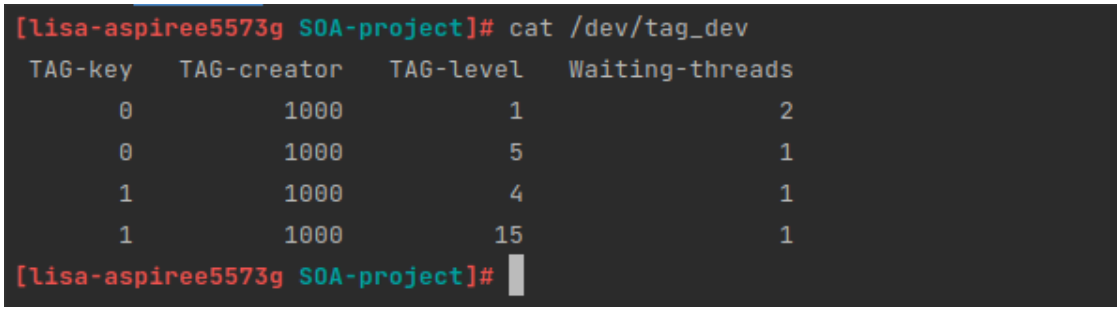

FIGURA 1 - ESEMPIO DI FUNZIONAMENTO DEL DRIVER TAG_DEV

3. Demo

Per permettere all'utente di **provare in modo interattivo le funzionalità** descritte nelle sezioni precedenti, è stato creata una semplice demo il cui codice è presente nella cartella *demo*. Di seguito vengono riportate una serie di immagini di esempio del suo funzionamento:

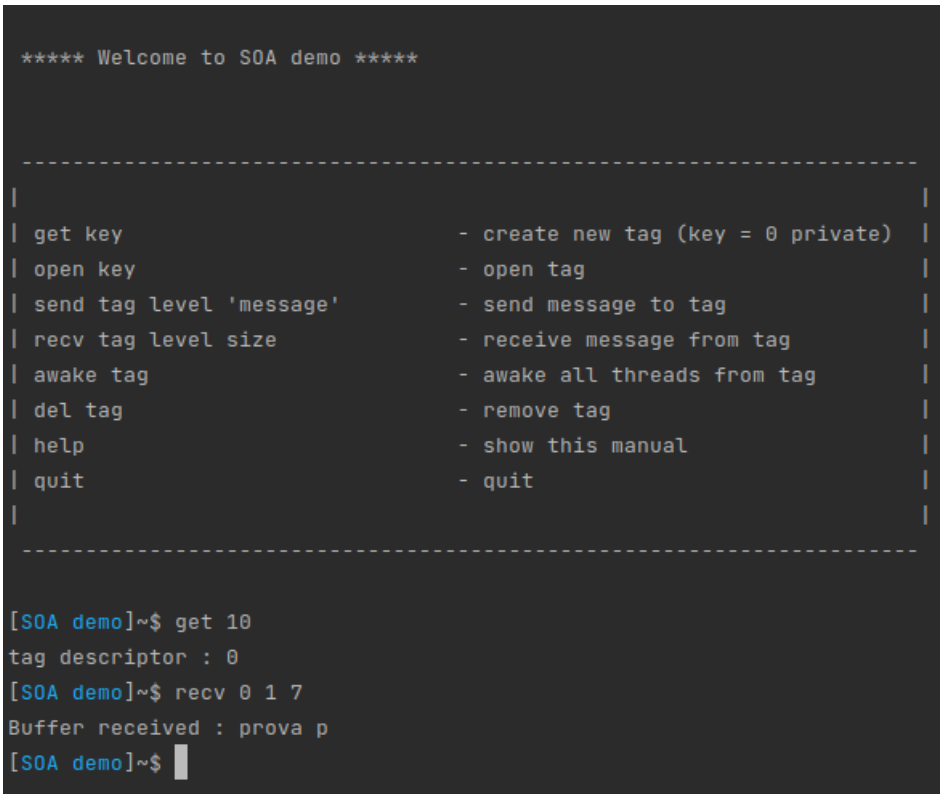

FIGURA 2 - DEMO DI RICEZIONE DI UN MESSAGGIO

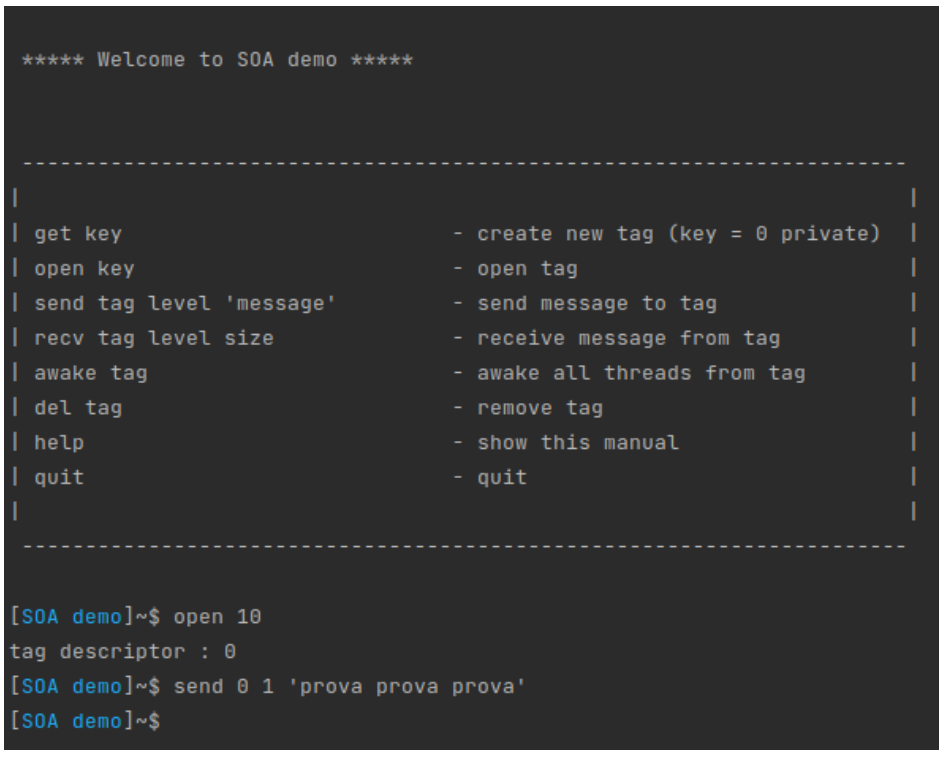

FIGURA 3 - DEMO DI INVIO DI UN MESSAGGIO

```
[  343.951097] SERVICE: tag_get called with params 10 - 1 - 1000
[  343.951102] SERVICE: New tag service 0 created
[  360.204493] SERVICE: tag_receive called with params 0 - 1 - 7
[  360.204504] TAG SERVICE: Process 2077 waiting for message...
[  366.006736] SERVICE: tag_get called with params 10 - 2 - 1000
[  366.006740] SERVICE: Tag service 0 opened by process 2069
[  378.797021] SERVICE: tag_send called with params 0 - 1 - prova prova prova - 17
[  378.797049] SERVICE: New message successfully sent to process 2077
[  378.820260] SERVICE: Message successfully sent to tag service 0 level 1
```

FIGURA 4 - TERMINALE DELLA FIGURA 2 E 3

4. Test

Oltre alla demo sono stati implementati anche una serie di test per accertare il corretto funzionamento delle system calls. Tutto il codice dei test si trova nella cartella *test*, mentre per la loro esecuzione è possibile usare lo script *test.sh*.

```
*** testing tag_get ***


Testing tag creation with slots available...                  128/256 created with user 1000 --- 128/256 created open to all

Testing tag creation with already existing tags...            0/256 tags created

Testing tag creation with max number of services reached...   0/256 tags created

Testing opening tags with permission...                       255/256 tags opened

Testing opening tags without permission...                    128/256 tags opened
```

FIGURA 5 - ESECUZIONE DI TEST.SH PARTE 1

```
*** testing tag_ctl ***


Testing tag deletion ...                                      256/256 tags removed

Testing tag deletion with waiting threads...                  0/1 tags removed

Testing awakening tag...                                      5/5 tags awakened


*** testing tag_send and tag_receive ***


Testing sending and empty message                       ...   5/5 tags successfully received the message

Testing sending message                                 ...   5/5 tags successfully received the message
```

FIGURA 6 - ESECUZIONE DI TEST.SH PARTE 2

5. Installazione ed esecuzione

Tutti i dettagli su come installare ed eseguire il progetto è possibile trovarli nel file *"README.md"* presente nella root directory di questa repository.

https://github.com/Lisa9601/SOA/blob/master/README.md