Sistemi Operativi Avanzati

A.Y. 2020-2021

TAG-based data exchange service

Tiziana Mannucci 0285727
ACM Member
Master Degree Student
Università degli Studi di Roma Tor Vergata
Via del Politecnico 1
Roma, Lazio, Italia
Email:
tiziana.mannucci@alumni.uniroma2.eu

***Abstract*— In questo articolo viene argomentato lo sviluppo di un servizio per lo scambio dei dati basato su TAG e di tutti i moduli necessari per il suo funzionamento.**

1 Introduzione

Nella presente relazione viene esposta una possibile implementazione per il servizio di scambio di messaggi basato su TAG
Molte scelte implementative sono state dettate da aspetti relativi alle performance, nello specifico si è cercato di proporre una soluzione che fosse efficiente e che non impattasse sulle altre parti del kernel.
Nella trattazione che segue si consideri che in tutte le operazioni si fa un check sui permessi e che dove ci sono sezioni critiche protette si è comunque interrompibili. Questo significa che quando viene eseguita un'operazione bloccante questa potrà essere interrotta e lo stato del modulo sarà comunque preservato.

2 System Call Table Hacking Modulo

Questo modulo implementa una system call table discovery a runtime nello scenario peggiore in cui non si hanno informazioni preliminari e la randomizzazione è attiva.
L'unica ipotesi che viene fatta e che rappresenta la chiave per identificare l'indirizzo della system call table è la conoscenza degli indici di 7 entry libere che corrispondono alla "ni_syscall".
Viene fatta una ricerca lineare scandendo la memoria virtuale, tra gli indirizzi effettivamente mappati in memoria fisica è stato cercato quello le cui entry corrispondenti alle 7 posizioni note avessero il medesimo valore poiché l'unica entry che si ripete nella system call table è la ni_syscall.
Una volta trovato l'indirizzo della tabella viene fatta un'ulteriore scansione lineare per cercare tutte le posizioni libere della tabella.
Tra i parametri che il modulo espone vi sono:

- un array che mantiene le posizioni trovate
- il numero di entry "libere" trovate

Le funzioni systbl_hack e systbl_entry_restore vengono usate rispettivamente per inserire e rimuovere una system call. Tutte le modifiche alla system call table vengono effettuate in mutua esclusione e viene *temporaneamente* disabilitata la *Write-Protection*.
Viene mantenuta una mappa di stato per ciascuna delle entry che sono state modificate per sapere quali non sono più utilizzabili e per bloccare qualsiasi tentativo di eliminare una syscall illecitamente. Nello specifico solo le system call inserite grazie alla systbl_hack potranno essere rimosse con la systbl_restore.

3 TAG Service data exchange modulo

Questo modulo sfrutta il modulo descritto in sezione 2 per inserire le API *tag_get, tag_send, tag_receive e tag_ctl*.
Il modulo mantiene traccia delle associazioni key-tag con la key_list di dimensione pari al numero massimo di chiavi consentite (parametro del modulo) e che è indicizzata dal numero di chiave stesso.
Una struttura analoga viene usata per tenere traccia dei tag creati: *tag_list*.

3.1 tag_get(int key, int command, int permissions)

Questa funzione si comporta in modo differente in base ai parametri specificati.

3.1.1 Key : IPC_PRIVATE

In questo caso il parametro command viene ignorato in quanto si tenta di creare un tag ex-novo. In caso di successo viene restituito un descrittore per il tag che però non è associato ad alcuna chiave né registrato nella

key_list. In questo modo il tag resta privato e solo il processo che lo ha creato può usufruirne insieme ai processi figli.

3.1.2 Command: IPC_CREAT

Se non esiste una corrispondenza nella lista delle chiavi il tag viene creato ex-novo, in caso contrario viene immediatamente restituito il descrittore del tag preesistente (caso open).
In caso si debba fare la creazione del tag, vengono prima istanziate e correttamente inizializzate tutte le risorse necessarie al suo funzionamento e, una volta a conoscenza del descrittore del tag utilizzato, viene fatto il collegamento con la chiave inserendolo nella key_list.
Per la creazione di un nuovo tag si deve ricercare il primo descrittore di tag disponibile. Tale ricerca viene fatta in modo lineare provando ad acquisire un write lock sull'i-esimo elemento della lista dei tag. Si ***prova*** ad ottenere il lock in quanto se non lo si acquisisce immediatamente significa che la risorsa è occupata, ovvero il tag è già esistente o qualcuno sta già creando il tag corrispondente a quel descrittore. In questo modo si fa una ricerca non bloccante che nella peggiore delle ipotesi, saturazione di tutti i descrittori per i tag, restituirà un errore.

3.1.3 Command: IPC_CREAT | IPC_EXCL

Il comportamento è analogo al caso della sola IPC_CREAT con la differenza che se la risorsa è già esistente viene restituito un errore EEXIST.

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

Un Sender prende un **read** lock sul tag per evitare che venga eliminato mentre sta consegnando il messaggio.
La consegna del messaggio avviene secondo lo schema **RCU**.

1. mutex_lock solo sul livello specificato. In questo modo solo i Sender che vogliono consegnare un messaggio sullo stesso tag e sullo stesso livello verranno bloccati.
2. Consegna del messaggio tramite copy_from_user per evitare problemi legati al cross ring data move.
3. L'attesa dei reader (ed il risveglio) viene gestita tramite una wait event queue. Nello specifico i Reader si risvegliano solo in base a due condizioni: AWAKE e MESSAGE. Un Sender segnala la condizione MESSAGE per l'epoca corrente (grace_epoch).
4. Cambia epoca sempre sotto write lock. In questo modo tutti i Reader ritardatari si sottoscriveranno alla nuova epoca e non verranno risvegliati dal messaggio corrente.
5. Sveglia di tutti i Reader in attesa sulla grace_epoch.
6. Aspetta che tutti consumino il messaggio e poi rilascia tutte le risorse e lock acquisiti in precedenza.

L'invio di messaggi di dimensione nulla non è considerata una condizione di errore per il Sender ma viene mappata in una nop. Tale scelta è motivata dal fatto che non si vuole permettere ad un Sender di risvegliare Readers con messaggi nulli. Tale compito spetta all'Awaker che dal punto di vista logico si comporta come un Sender che però consegna una notifica di risveglio al posto di un messaggio.
Contrariamente si creerebbe una situazione ambigua in cui un Reader si risveglierebbe senza un vero messaggio né una notifica di AWAKE.

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

Analogamente al Sender prende un **read** lock sul tag per evitare che venga eliminato mentre sta aspettando un messaggio.
Successivamente si aggiunge agli standing readers per il tag-level specificato e per l'epoca a cui appartiene in modo atomico con una:

- `__sync_fetch_and_add(&my_tag->msg_rcu_util_list[level]->standings[my_epoch_msg], 1)`

Dopo aver effettuato la sottoscrizione aspetta fino a quando non viene consegnato un messaggio, una notifica di AWAKE o un segnale.
Come descritto in sezione 3.2 l'attesa viene implementata tramite una wait event queue e le condizioni per il risveglio sono quelle sopra citate.
Se avviene l'arrivo di un messaggio questo viene consumato e viene ritornata la lunghezza del messaggio stesso. Qualunque sia la condizione del risveglio il Reader cancella la sua sottoscrizione:

- `__sync_fetch_and_add(&my_tag->msg_rcu_util_list[level]->standings[my_epoch_msg], -1)`

e rilascia tutte le risorse.

3.4 tag_ctl(int tag, int command)

Questa funzione si comporta in modo differente in base ai parametri specificati.

3.4.1 Command: AWAKE_ALL

Un Awaker prende un **read** lock sul tag per evitare che venga eliminato mentre sta svolgendo il suo lavoro.
Dal punto di vista logico si comporta come un Sender con la differenza che al posto di consegnare un messaggio viene notificata la condizione di AWAKE. Tale operazione viene fatta per tutti i livelli di un tag iterando la seguente procedura:

1. mutex_**trylock** solo sul livello specificato.
2. Consegna della notifica di AWAKE.
3. Cambia epoca sempre sotto write lock. In questo modo tutti i Reader ritardatari si sottoscriveranno alla nuova epoca e non verranno risvegliati dalla notifica corrente.
4. Sveglia di tutti i Reader in attesa sulla grace_epoch.
5. Aspetta che tutti consumino e cancellino la sottoscrizione, poi rilascia lock acquisito e va avanti.

Come si può notare la differenza critica con un Sender risiede nel modo in cui si acquisisce il lock. Nello specifico l'uso di trylock è legato al fatto che questo servizio vuole favorire in un qualche modo la consegna di un messaggio. Se non si acquisisce immediatamente il lock significa che per l'epoca corrente di quel livello un Sender sta già inviando il messaggio o c'è un altro Awaker che sta risvegliando quel livello. Se si fosse scelto un comportamento bloccante, finita l'attesa, si risveglierebbero i Reader sottoscritti alla nuova epoca. La scelta implementativa è stata quella di preferire la consegna di un messaggio alla cancellazione dell'operazione di lettura.

3.4.2 Command: IPC_RMID

Un Remover **prova** ad ottenere un write lock sul tag e se non viene acquisito immediatamente viene ritornato un errore EBUSY. La scelta di adottare un **trylock** è legata al fatto che non si può rimuovere un tag fino a che ci sono standing Readers (acquisiscono read lock) ma soprattutto perché altrimenti si creerebbe un deadlock.
Nello specifico si usano N rw_semaphore per gestire l'accesso ai tag. Questi hanno una politica fair e cercano di evitare la starvation degli scrittori. Tale comportamento provoca un blocco per tutti i lettori che tentano di acquisire un read lock quando uno scrittore è ancora in attesa per prendere il write lock.
Un Remover resterebbe bloccato poiché Readers prima di lui hanno preso il read lock. Un Reader può essere risvegliato solo da Sender o Awaker (o segnale) che a loro volta, prendendo un read lock, resterebbero bloccati dal Remover.
Pertanto, un Remover che acquisisce il write lock immediatamente rimuove l'associazione tag-key dalla lista delle chiavi, rimuove il tag con tutte le risorse ad esso associate e rilascia tutti i lock acquisiti.
Il Remover può comunque avere un comportamento bloccante poiché deve anche acquisire il lock sulla lista delle chiavi per potervi scrivere e cancellare l'associazione con il tag, se esistente.
Si può prevenire questo comportamento bloccante andando ad usare IPC_NOWAIT | IPC_RMID.

4 DEVICE DRIVER

Il device driver implementato è un modo per vedere lo stato del sistema ad un certo istante, nello specifico per sapere per ciascun tag quanti standing readers sono attivi.
Poiché l'idea fondamentale è quella di prendere un'istantanea del sistema, l'accesso al device file non può avvenire in time sharing: istanza singola. Tale scelta è motivata dall'unicità stessa dell'istantanea sul tag service.
Nello specifico open_tag_status si occupa di scandire la lista dei tag e di prendere le informazioni necessarie da ciascuno di essi. Con le informazioni ricavate si può costruire un testo che può essere letto usando read_tag_status.
Con release_tag_status si rilasciano le risorse ed il lock acquisito nella open_tag_status.