RELAZIONE PROGETTO GESTIONE DI RETE

Spediacci Fabio
matricola: 411615
corso B

SCOPO

Realizzare una libreria per ricevere ed inviare rapidamente pacchetti udp.
L'interfaccia offerta dalla libreria dovrà essere la stessa usata dalle socket bsd, ovvero quella definita nell'header sys/socket.h. Questo per far sì che la si possa sostituire alla libreria di sistema in modo trasparente per le applicazioni nel momento in cui queste vengono eseguite senza necessità di cambiare il codice o ricompilare.

CARICAMENTO DELLA LIBRERIA

Per caricare la libreria è possibile istruire il sistema affinchè carichi i suoi simboli prima di quelli della libreria di sistema, così da "scavalcarli". Questo può essere fatto impostando la variabile d'ambiente LD_PRELOAD con il path della libreria.
Esempio:

```
LD_PRELOAD=/path/to/lib/dnastack.so some_app
```

In dnastack.h è possibile definire una macro DNASTACK_DEBUG che abilita alcune stampe utili per il controllo delle performance.

REALIZZAZIONE

La libreria fà uso della libreria pfring per poter ricevere ed inviare pacchetti più rapidamente da\verso la scheda di rete.
A causa del fatto che la libreria non è in grado di determinare l'interfaccia sulla quale inoltrare un pacchetto, attualmente essa consente l'uso di una sola interfaccia per processo.
Il nome dell'interfaccia verrà letto dal file /etc/dnastack.conf (che dovrà contenere semplicemente il nome dell'interfaccia).

SUPPORTO

La libreria supporta le funzioni:

- ➔ socket che per le socket udp non supporta valori diversi da 0 per protocol.
- ➔ sendto supporta solo la flag MSG_DONTWAIT per le socket udp
- ➔ recvfrom supporta solo la flag MSG_DONTWAIT per le socket udp
- ➔ close funziona come ci si aspetterebbe
- ➔ bind per le socket udp non vi è supporto al bind su di un particolare indirizzo

INIZIALIZZAZIONE E RILASCIO RISORSE

Quando viene aperta una socket udp e non ce ne sono altre aperte viene chiamata una funzione di inizializzazione e quando l'ultima socket viene chiusa ne viene chiamata un altra per rilasciare le risorse.

ARP

La libreria durante la fase di inizializzazione recupera le informazioni dalla

tabella arp del kernel (e, durante la fase di chiusura, scrive nel kernel le sue informazioni). Durante l'invio dei pacchetti viene recuperato, se necessario, il mac address del destinatario tramite arp. La libreria è anche in grado di rispondere alle richieste fatte da altri host.

SENDTO

Non vi è molto da dire sulla sendto: la funzione calcola sia la checksum udp che quella ip. L'invio dei pacchetti è sincronizzato mediante una spinlock al momento di inviare effettivamente il pacchetto.

RICEZIONE PACCHETTI

Vi è un thread (da ora in poi chiamato thread lettore) che viene avviato dalla funzione di inizializzazione il cui unico compito è leggere continuamente pacchetti dalla socket pfring, effettuarne il parsing e, se è un pacchetto arp o udp, scriverne rispettivamente il loro contenuto nella tabella arp o nel buffer della corrispondente socket.
Questo thread ha un meccanismo piuttosto semplice, per stabilire se è più opportuno effettuare le chiamate a pfring_recv con wait_for_packets a 1 o a 0.
Il meccanismo consiste nel controllare periodicamente, ogni tot chiamate della funzione, le statistiche della socket: se nota un "significativo" aumento di pacchetti scartati allora lo imposterà a 0 altrimenti a 1.

STRUTTURE DATI (INTERESSANTI) USATE

-> Buffer delle socket
Per i buffer delle socket ho usato un ring buffer, implementandolo in modo che i pacchetti possano essere inseriti ed estratti in parallelo; senza necessità di sincronizzazioni esplicite purchè non si effettui più di un inserimento e di un estrazione in "contemporanea".
Nella struttura ho anche inserito alcune variabili per memorizzare i parametri passati alla funzione recvfrom. Questo consente al thread che effettua la chiamata di farsi scrivere il contenuto del pacchetto direttamente nel buffer passato alla funzione, invece che nel ring buffer (questo ovviamente a patto che la chiamata a recvfrom avvenga prima che il thread lettore tenti di inserire il pacchetto).
L'unico tipo di sincronizzazione avviene quando non vi sono pacchetti da estrarre, in quel caso il thread si mette in attesa su un semaforo.
Il contatore del semaforo verrà incrementato dal thread lettore solo se ha scritto usando i parametri della funzione recvfrom, oppure se il ring buffer è vuoto e non ha già scritto in precedenza qualcosa che non è ancora stato letto tramite i parametri.

-> Tabella arp
Per la tabella arp globale, ovvero quella comune a tutte le socket, ho usato una hashtable implementata da me, non essendo riuscito a trovarne una già fatta che andasse bene.
La tabella ha una dimensione massima prefissata e quando viene superata vengono rimossi gli elementi "scaduti"; se non esistono elementi "scaduti" allora verrà rimosso l'elemento usato meno di recente.

Ogni socket ha inoltre una sua tabella arp personale (più piccola)

implementata come un ring buffer (non con l'implementazione usata nei buffer dei pacchetti delle socket). Ogni ricerca inizia dall'ultimo elemento letto e la rimozione è molto più semplice in quanto segue una logica FIFO.
Ho scelto questa implementazione più semplice, per via del fatto che la cache delle socket è molto piccola e quindi usare strutture dati più evolute mi sembrava eccessivo e forse controproducente.

DNASTACK

Contiene le definizioni delle strutture dati delle socket.
Ne ho create due:
- PfSocket contiene le informazioni relative ad una interfaccia di rete come ad esempio: il puntatore alla socket pfring, la tabella arp, l'indirizzo ip e il mac address dell'interfaccia.
- DnaSocket invece contiene le informazioni relative ad una specifica socket (aperta dalla chiamata socket) quali la tabella arp personale, la porta sulla quale è collegata e il ring buffer per memorizzare i pacchetti.

DnaSocket contiene al suo interno un puntatore ad una funzione che è in grado di creare l'header per il protocollo sulla quale è stata aperta e il numero identificativo del protocollo.
Grazie a queste due caratteristiche è possibile estendere con poca fatica la libreria per supportare protocolli aggiuntivi.

Oltre a questo contiene le funzioni necessarie per gestire le socket.

READ

Contiene le funzioni necessarie per fare il parsing dei pacchetti e per gestire il thread lettore.

SEND

Contiene la funzione necessaria per inviare pacchetti con sendto.

ARP

Contiene le funzioni necessarie per inviare pacchetti arp, per gestire la tabella arp e per scrivere\leggere nella tabella arp del kernel.

PKT_RING E HASHTABLE

Rispettivamente hanno l'implementazione di ring buffer e hashtable.