Caveau

Ivan Simonini 119314

15 gennaio 2012

1 Scelte e interpretazioni

Com'è possibile intuire dallo scambio di mail avvenuto durante la realizzazione di questo progetto, ho avuto qualche difficoltà a comprendere quale sarebbe dovuto essere il comportamento del dispositivo. Il termine "cache" mi ha fuorviato, dato che il dispositivo non deve fare alcuno sforzo per mappare i blocchi al momento della lettura, né tantomeno andare alla ricerca di eventuali blocchi mancanti.

Vista questa discrepanza tra il nome del dispositivo e il suo comportamento, l'ho chiamato `Caveau`.

1.1 Lettura e scrittura

La consegna prevede che il dispositivo accetti input a blocchi di 1024byte, e il mantenga in memoria secondo una certa policy specificata al momento del caricamento. La scrittura è sempre possibile, poiché il dispositivo non mantiene le informazioni in modo permanente, ma le può sovrascrivere dal momento in cui tutti i blocchi sono stati occupati.

La lettura invece, non fa che restituire il contenuto del dispositivo.

1.2 Obiettivo opzionale: secondo livello di cache

Lo sviluppo di un dispositivo a due livelli come obiettivo secondario è stato uno dei motivi in favore di questo progetto, a scapito delle altre alternative. Considerato poi che l'implementazione di un numero arbitrario di livello è assolutamente analoga allo sviluppo di un singolo livello ulteriore, ho costruito il dispositivo in modo che possa funzionare con un numero arbitrario di livello, secondo un parametro di configurazione.

In questo senso, e - ricollegandomi a sopra - dato che la consegna richiede spiegazioni sul comportamento del dispositivo in caso i livelli siano più d'uno, ho agito come segue.

1.3 Cache a più livelli

Ciascun livello del dispositivo, in particolare, il primo, è uno spazio di memoria di dimensione fissata (il numero di blocchi è uno dei parametri da indicare al caricamento) che accetta input. Indipendentemente dalla policy scelta, dal momento del riempimento dell'ultimo blocco disponibile, ogni successiva scrittura va a ricoprire un blocco già esistente.

Questi blocchi vengono per così dire spinti via. Mi è sembrato naturale, quindi, non farli cadere nell'oblio, bensì applicare ricorsivamente lo stesso metodo, passandoli come input al livello successivo. È ovvio, poiché la scrittura e la lettura del dispositivo sono completamente slegate, che questo meccanismo funzioni in una sola direzione.

A ciascuna scrittura, quindi, il dispositivo scrive i dati sul primo livello di cache. Qualora questo primo livello sia pieno (abbia cioè scritto almeno una volta su ogni blocco), il blocco selezionato per la scrittura verserà il proprio contenuto nel livello successivo, il quale lo accetterà nello stesso modo. Si viene a creare, con il tempo, una cascata di blocchi che scende dal primo livello verso i successivi. Ogni livello è libero di scegliere il blocco da sovrascrivere in modo indipendente.

Per quanto riguarda la lettura, invece, la consegna specifica che deve fornire un dump dell'intero contenuto. In caso di più livelli presenti, il loro contenuto viene dumpato in ordine, dal primo all'ultimo.

2 Dettagli implementativi

Il codice del dispositivo è piuttosto semplice, il sorgente `caveau.c` non supera le 500 righe.

2.1 Strutture dati

Nell'intero sorgente vengono implementate tre strutture.

In ordine gerarcico, la prima è `caveau_dev`, la quale rappresenta l'intero dispositivo. Essa contiene i riferimenti ai vari livelli, al semaforo utilizzato per sincronizzare gli accessi e il `char device`. Questa struttura viene allocata una singola volta, al caricamento del modulo, e memorizzata in una variabile globale, `cv_device`. Essa persiste fino alla rimozione del modulo.

La seconda struttura è `caveau_lev`, la quale rapprensenta un livello di cache. È la struttura più complessa, che sovrintende quasi l'intera logica del modulo e del dispositivo. Essa contiene un vettore di blocchi, alcuni membri riservati alla gestione degli utilizzi e soprattutto il riferimento ad un blocco di memoria nel quale mantenere il contenuto del livello presente.

Infine, `caveau_blk` è la struttura che rapprensenta un singolo blocco di 1024byte. Essa mantiene due soli parametri, la dimensione del contenuto occupato e un puntatore alla memoria. Questa struttura è di fatto quasi superflua, ma il suo uso permette di diminuire considerevolmente il numero di parametri da mantenere durante la gestione delle operazioni di lettura.

2.2 Lettura del dispositivo

La lettura del dispositivo avviene in tre fasi, una per il dispositivo, una per il livello, una per il blocco.

Innanzitutto, viene effettuata una comparazione tra l'offset richiesto e la dimensione del dispositivo (numero di livelli * numero di blocchi * 1024 caratteri). Qualora l'offset sia effettivamente all'intero dello spazio disponibile, il controllo viene passato al livello interessato. Nello stesso modo, il livello prende in considerazione l'offset per invocare il blocco che lo contiene.

A quel punto, il blocco, restituisce il suo contenuto, fino a riempire il buffer di destinazione o fino al suo limite. Se il limite viene raggiunto, il livello presente continua la lettura sui blocchi successivi. Se il contentuo dell'intero livello non è sufficiente a soddisfare le richieste di lettura, il dispositivo seleziona il livello successivo. Una volta raggiunto l'ultimo byte disponibile nell'ultimo blocco dell'ultimo livello, la lettura termina.

La chiamata effettua la copia del dispositivo su un buffer interno, il quale viene poi spostato in user-space tramite la `copy_to_user`. Considerato però che la memoria del dispositivo è assolutamente sequenziale, e che non esistono discontinuità al suo intero, sarebbe possibile, come sviluppo futuro, modificare l'implementazione in modo che il contenuto (`caveau_lev_t->safe`) dei livelli interessati venisse copiato direttamente in user-space, senza passare per livelli e blocchi.

2.3 Scrittura sul dispositivo

La scrittura sul dispositivo non avviene in modo sequenziale, e parte sempre dal primo livello. Essa viene effettuata in modo ciclico: al livello viene passato l'intero buffer sorgente, e fintanto che esso non è stato completamente elaborato, la scrittura prosegue.

Il livello sceglie (in base alla policy specificata) quale blocco scrivere. Se il blocco possiede già del contenuto, la scrittura viene momentaneamente sospeva e questo viene travasato al livello successivo. Se il blocco è libero oppure è stato liberato, 1024byte del buffer sorgente (o meno, se non ne sono rimasti) vengono scritti sulla sua porzione di memoria. Se la scrittura non è ancora terminata, il livello seleziona un nuovo blocco (che può essere quello appena utilizzato) e prosegue.

A differenza della lettura, la scrittura consuma sempre tutto il buffer disponibile.

Anche in questo caso, come sviluppo futuro, è possibile modificare la procedura perché effettui la copia da user-space direttamente sul blocco interessato.

2.4 Policy

La consegna prevede l'implementazione di tre diverse policy per la scelta dei blocchi, ma non solo: esse devono entrare in azione soltanto alla piena occupazione del livello; prima, i blocchi liberi hanno la precedenza su quelli occupati.

Ciascun livello conta gli accessi in scrittura effettuati su se stesso (nella variabile `caveau_lev_t->served`. Finché questo valore non supera il numero di blocchi del livello, la policy selezionata verrà ignorata in favore dei blocchi liberi. Questo tramite un indice (`caveau_lev_t->last`) che si muove come su un buffer circolare. Il suo valore iniziale è tirato a caso al momento dell'inizializzazione del livello.

Dopodiché, il livello comincia a comportarsi come da selezione:

- la policy `caveau_random` seleziona un blocco a caso, basandosi su `random32`;
- la policy `caveau_most` utilizza invece il valore di `last`, che rimane invariato dall'entrata in vigore della policy ed indica l'ultimo blocco rimasto libero

- la policy `caveau_least` continua a comportarsi esattamente come se i blocchi fossero ancora liberi, utilizzando `last` come un indice che circola su tutti i blocchi, in ordine

3 Testing

Per la compilazione e il testing del modulo, assieme al sorgente, vengono formiti un `Makefile` ed alcuni script di supporto.

Il Makefile contiene quattro target: mod, doc, test e clean. In ordine, essi compilano il modulo, compilano questo documento, eseguono dei test e ripuliscono la cartella dai file temporanei. Il target di default è il documento pdf.

Sono inoltre presenti tre script.

- `mkcache` si occupa di caricare il modulo, di recuperare dinamicamente il major number, di creare il device e di settare i permessi. Esso prende tre parametri: il numero di livelli, il numero di blocchi per livello e l'indice della policy (0 per random, 1 per most e 2 per least)

- `mkdump` si occupa di scrivere contenuti nel device, per il numero di volte specificato; esso scrive del breve testo, poi del testo meno breve e infine, una volta per chiamata, un testo particolarmente lungo (che ha lo scopo di occupare due diversi blocchi)

3.1 Il test

Quello che `make test` effettua è un semplice test di funzionamento. Esso monta il modulo con vari parametri e vi scrive un certo numero di volte (abbastanza da manifestare il funzionamento corretto di ogni livello).

Poiché però controllare la disposizione dei blocchi al momento della lettura stanca gli occhi, e poiché è davvero difficile controllare come i blocchi vengano scelti, ho preferito basare i miei controlli su opportuni messaggi che il dispositivo stampa tramite `printk` e che sono disponibili tramite `dmesg` (questo permette, tra l'altro, di controllare l'esito anche dopo che il modulo è stato rimosso).

Ad ogni accesso ad un blocco, viene prodotto un messaggio: il livello specifica quale blocco è stato scelto e in che modo (con la policy oppure, prima del riempimento, dando precedenza ai blocchi liberi). È molto più semplice, in questo modo, verificare non solo il corretto funzionamento delle policy, ma anche l'effetto di scrittura a cascata.