Università degli studi di Roma Tor Vergata

Corso di Advanced Operating System

Relazione Progetto A.A 2021/2022

Tummolo Gabriele – Matricola: 0283629

Introduzione

Il progetto consiste nell'implementazione di un device driver per la gestione di due flussi di priorità: alta e bassa. Il device driver deve supportare 128 device differenti, utilizzando una politica per la gestione delle richieste di lettura e scrittura di tipo FIFO (First In First Out). Le operazioni di lettura e scrittura possono essere sia bloccanti che non bloccanti.

La specifica del progetto richiedeva anche di implementare il supporto per il servizio Ioctl() che permette di gestire la sessione I/O nel quale è possibile modificare i parametri della sessione stessa, in particolare:

- Cambiare impostazione del livello di priorità (Alta o bassa) per le operazioni
- Cambiare il tipo di operazione della sessione (Bloccante o non bloccante)
- Cambiare il timeout per la regola di attivazione delle operazioni di blocco

Per le operazioni di lettura sia ad alta che a bassa priorità i dati letti vengono eliminati dal flusso, quindi se un certo numero di caratteri viene letto nella lettura successiva questi non saranno più presenti.

Le operazioni di scrittura a bassa priorità devono offrire un'esecuzione asincrona basata sul lavoro ritardato (deferred work).

Inoltre, si richiedeva anche di definire alcuni parametri e funzioni del modulo Linux per abilitare/disabilitare il file del dispositivo (indicando il minor number), la disabilitazione deve far sì che l'apertura di una sessione verso quel file non può essere effettuata.

I parametri permetto di mantenere informazioni sullo stato del dispositivo, in particolare permettono di mantenere le seguenti informazioni:

- Numero di byte presenti nei due flussi (Alta priorità e Bassa priorità)
- Numero di thread in attesa sui due flussi

Strutture dati

Per poter mantenere delle informazioni importanti al fine di soddisfare le richieste del progetto sono state utilizzate 3 strutture:

- *Info_sessione*

- *Info_device*
- *Data_work*

La struttura dati *info_sessione* ha permesso di mantenere le informazioni della sessione I/O, nello specifico:

- Int priorita, indica il tipo di priorità della sessione I/O (Alta o Bassa)
- Int tipo_operaz, indica se l'operazione è bloccante/non bloccante
- Unsigned long timeout, indica il timeout in millisecond

La struttura *info_device* mantiene le informazioni del device:

- Struct mutex mutex_op[2], mutex utilizzati per coordinare le operazioni di lettura e scrittura sui due flussi
- Wait_queue_head_t coda_attesa[2], struttura utilizzata per poter gestire le operazioni di lettura e scrittura bloccanti
- Int byte_validi[2],indica il numero di byte presenti all'interno dei due flussi
- Char *streams[2], rappresenta il flusso verso lo specifico device

L'ultima struttura *data_work* è stata utilizzata per poter implementare il deferred work per la scrittura a bassa priorità, le informazioni che vengono salvate sono:

- Int minor, indica il minor number
- Struct work_struct work, utilizzata per la gestione della scrittura a bassa priorità
- Char *buffer, utilizzato per recuperare la stringa che si vuole scrivere nel flusso
- Int len, indica il numero di byte che si vogliono scrivere nel flusso

Funzioni

Per la scrittura del modulo.ko sono state sviluppate le 5 funzioni che sono state registrate nel file_operations:

- Lettura: *lettura_device*
- Scrittura: *scrittura_device*
- Apertura device: *apertura_device*
- Chiusura device: *rilascio_device*
- Ioctl : *operazione_ioctl*

Oltre a queste funzioni ci sono le funzioni *inizializzazione_modulo()* e *rilascio_modulo()* che servono rispettivamente ad allocare le risorse quando il modulo viene installato e deallocare le risorse al rilascio del modulo.

Per gli stream che rappresentano i flussi per i due livelli di priorità è stata utilizzata la funzione *krealloc()* che ha permesso di ridimensionare i buffer rappresentare i flussi, in questo modo dinamicamente i buffer aumentano la loro dimensione (in caso di scrittura) e la diminuiscono (in caso di lettura).

Apertura device

Questa funzione serve per poter aprire una sessione di I/O verso un device, identificato tramite il suo minor number, in questa funzione viene solo creata la sessione assegnando le impostazioni di default (Alta priorità, operazione non bloccante e con un timeout di 3000 ms); viene effettuato un controllo sul minor number che non deve essere superiore a 128 e un controllo sul device che deve essere abilitato, in caso contrario la sessione verso quel device non viene creata e aperta.

Scrittura

Dopo l'apertura di una sessione I/O verso un device è possibile effettuare una scrittura, per effettuare la scrittura si utilizza la funzione *copy_from_user()* che permette di recuperare i dati dallo spazio utente per utilizzarli nello spazio kernel. Quindi dopo aver recuperato i byte che si vogliono scrivere sul flusso, si procede con la scrittura sul buffer che rappresenta il flusso.

Prima della scrittura si acquisisce il lock (rappresentato da un mutex) che permette di effettuare la scrittura in modo atomico, per l'acquisizione del lock è stata definita una funzione apposita *prendi_lock()*.

Se il lock viene preso si procede con la scrittura del buffer rappresentativo del flusso, per effettuare la scrittura tramite la funzione *strncat()* il buffer viene ri-dimensionato con la funzione *krealloc()*, successivamente vengono aggiornate i parametri del modulo con la funzione *aggiorna_variabili()* e poi viene rilasciato il lock. Poi viene utilizzata la funzione *wake_up()* che sveglia i thread che sono in attesa e sono all'interno della wait_queue_head_t del flusso corrispondente.

Quello descritto sopra rappresenta la logica che viene eseguita nel caso in cui la scrittura viene fatta sul flusso a priorità alta, nel caso di priorità bassa è stata implementata la deferred work.

Il deferred work è stato implementato con le funzioni *INIT_WORK()* e *queue_work()* queste funzioni hanno permesso di inizializzare ed inserire all'interno di una work queue il lavoro da eseguire (la work queue viene dichiarata in precedenza). Prima di queste funzioni sono stati popolati i dati della struttura work_struct che successivamente vengono acceduti tramite la funzione *container_of()* utilizzata all'interno della funzione *deferred_work()* che permette di eseguire la scrittura ritardata.

Lettura

La logica utilizzata per la lettura è simile a quella utilizzata per la scrittura, ma non c'è differenziazione su quello che deve essere fatto in base a sé ci si trova a bassa o ad alta priorità. Dopo l'acquisizione del lock tramite la funzione *prendi_lock()* si procede a salvare su un buffer temporaneo i byte che si vogliono leggere dal flusso per poi eliminare i dati letti, per eliminare i dati letti è stata utilizzata la funzione *memmove()* e *krealloc()*.

Dopo vengono aggiornati i parametri del modulo e rilasciato il lock.

Per poter portare i dati a livello utente è stata utilizzata la funzione *copy_to_user()*.

Funzione Ioctl

Questa funzione permette di modificare le impostazioni definite all'interno della sessione I/O, in particolare permette di:

- Impostare la priorità alta o bassa
- Definire le operazioni su quella sessione bloccanti o non bloccanti
- Modificare il timeout
- Reimpostare le impostazioni di default
- Abilitare o disabilitare un device

Funzioni di supporto

Le funzioni sono:

- *prendi_lock()*: questa funzione è utilizzata per prendere il lock, nel caso di operazione non bloccante viene eseguita una semplice *mutex_trylock()*, mentre per le operazioni bloccanti viene eseguita la funzione *wait_event_timeout()*. Quest'ultima funzione sospende il processo fino a quando la condizione indicata non diventa vera, la condizione in questo caso è quando il thread riesce a prendere il lock.
- *aggiorna_variabili()*: questa funzione permette di aggiornare i parametri del modulo in modo atomico, l'atomicità viene garantita grazie all'utilizzo delle funzioni *__sync_fatch()*.
- *deferred_work()*: questa funzione è molto importante in quanto definisce il lavoro che deve essere eseguito quando viene eseguita la scrittura ritardata. Dopo aver eseguito la funzione *container_of()* e recuperato le informazioni di work_struct, si esegue la *mutex_lock()* per poi effettuare la scrittura sullo stream a bassa priorità e poi rilasciare il lock con la funzione *mutex_unlock()*.

Per testare il modulo è stato scritto il file *read_and_write.c* nel quale è possibile effettuare letture e scritture per uno specifico device.

Link Utili

Link Progetto: https://github.com/brielino/SOA_Project (il file Funzionamento spiega nello specifico come fare i test sul programma)

Link Deferred Work: https://linux-kernel-labs.github.io/refs/heads/master/labs/deferred_work.html

Link Module Param: http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch16lev1sec6.html