Relazione finale progetto Sistemi Operativi

Argomento Progetto: Semafori disastrOS e realizzazione Message Queue

Membri Gruppo: Vincenzo Claudio Casanova - 1662174
Giorgio De Magistris - 1645068
Marco Di Lisa - 1692433
Flavia Monti – 1632488

Git Repo: https://gitlab.com/Clarks95/disastrOS_semaphores

Implementazione Progetto:

1. **disastrOS_semopen.c** – Marco Di Lisa
    a. Parametri Input:
        - int id -> Identificativo del semaforo, unico in tutto il sistema
        - int count -> contatore a cui inizializzare il semaforo (serve solo per la prima allocazione, successivi parametri verranno ignorati)
    b. Valore di Ritorno:
        Descrittore del file oppure codice di errore
    c. Semantica:
        Viene cercato il semaforo nella lista dei semafori del SO, se il semaforo non è stato trovato significa che non è stato allocato. Nel caso in cui il semaforo non ci fosse viene allocato e si controlla che count sia maggiore o uguale a zero, in caso contrario il SO ritorna errore.
        Successivamente viene allocato il descrittore del semaforo e lo si aggiunge alla lista dei descrittori del processo running. Viene allocato e aggiunto il puntatore al descrittore del semaforo nella lista dei descrittori del semaforo.
2. **disastrOS_semclose.c** – Vincenzo Claudio Casanova
    a. Parametri Input:
        - int fd -> Descrittore del semaforo, unico all'interno del processo
    b. Valore di Ritorno:
        In caso di successo 0, altrimenti codice di errore
    c. Semantica:
        Viene cercato il descrittore del semaforo nella lista dei semafori del processo, se non ci fosse si ritorna errore al SO. Il descrittore del semaforo viene rimosso dalla lista dei descrittori del processo running. Dal descrittore del semaforo viene ricavato il semaforo e il puntatore al descrittore del semaforo, successivamente si rimuove dalla lista dei descrittori del semaforo il puntatore al descrittore e viene verificato che il semaforo non sia in uso analizzando le size delle due liste del semaforo, se fosse in uso

ritorna errore. A questo punto si può rimuovere il semaforo dalla lista dei semafori del SO.

Infine viene deallocata la memoria per il semaforo, il descrittore e il puntatore al descrittore del semaforo.

3. **disastrOS_sempost.c** – Flavia Monti
    a. Parametri Input:
        - int fd -> Descrittore del semaforo, unico all'interno del processo
    b. Valore di Ritorno:

        In caso di successo 0, altrimenti codice di errore
    c. Semantica:

        Viene salvato il descrittore del semaforo dal processo running tramite il fd del semaforo, viene ritornato errore in caso non lo si trovasse. Si ricava il semaforo e viene incrementato il contatore del semaforo. Se il contatore è minore o uguale a 0, si rimuove il primo puntatore al descrittore del semaforo dalla waiting list del semaforo e da questo si ricava il pcb del processo attualmente in waiting che dovrà essere messo nella lista dei processi ready del SO. Viene quindi rimosso il pcb del processo dalla waiting list del SO e inserito nella ready list del SO.
4. **disastrOS_semwait.c** – Giorgio De Magistris
    a. Parametri Input:
        - int fd -> Descrittore del semaforo, unico all'interno del processo
    b. Valore di Ritorno:

        In caso di successo 0, altrimenti codice di errore
    c. Semantica:

        Viene salvato il descrittore del semaforo dal processo running tramite il fd del semaforo, in caso non lo si trovasse viene ritornato errore. Si ricava il semaforo dal descrittore e viene decrementato il contatore. Se il contatore diventa minore di zero si alloca il puntatore al descrittore del semaforo e lo si aggiunge alla waiting list del semaforo. Il processo running viene spostato nella waiting list del SO e viene messo in running il primo processo che si trova nella ready list del SO.
5. **fixed_size_message_queue.h**, **fixed_size_message_queue.c** – Vincenzo Claudio Casanova, Giorgio De Magistris, Marco Di Lisa, Flavia Monti
    a. struct FixedSizeMessageQueue
        - char* message -> per memorizzare il messaggio
        - int size -> dimensione messaggio
        - int size_max -> dimensione massima del messaggio
        - int front_idx -> indice del primo carattere che si può leggere
        - int sem_full -> unico id del semaforo full. Questo semaforo ha lo scopo di bloccare i thread che vogliono fare la popFront su una coda vuota.

- int sem_empty -> unico id del semaforo empty. Questo semaforo ha lo scopo di bloccare i thread che vogliono fare la pushBack su una coda piena.
- int thread_sem -> unico id del semaforo per i thread. Questo semaforo si occupa della sezione critica della popFront e della pushBack.

b. FixedSizeMessageQueue_init

- Semantica:

  Vengono inizializzati i rispettivi pool allocator per le code e per i messaggi.

c. FixedSizeMessageQueue_alloc

- Parametri Input:
  - FixedSizeMessageQueue* q -> puntatore alla coda da inizializzare
  - int size_max -> dimensione massima del messaggio
  - int* sem_empty_fd -> fd del semaforo empty
  - int* sem_full_fd -> fd del semaforo full
  - int* thread_sem_fd -> fd del semaforo del thread
- Semantica:

  Vengono allocate e inizializzate tutte le variabili interne della struttura coda q.

d. FixedSizeMessageQueue_destroy

- Parametri Input:
  - FixedSizeMessageQueue* q -> puntatore alla coda da eliminare
  - int sem_empty_fd -> fd del semaforo empty
  - int sem_full_fd -> fd del semaforo full
  - int thread_sem_fd -> fd del semaforo del thread
- Semantica:

  Viene liberata la memoria occupata dal messaggio, vengono chiamate le funzioni per chiudere i semafori.

e. FixedSizeMessageQueue_pushBack

- Parametri Input:
  - FixedSizeMessageQueue* q -> puntatore alla coda da eliminare
  - char* message -> messaggio da inserire nella coda
  - int sem_empty_fd -> fd del semaforo empty
  - int sem_full_fd -> fd del semaforo full
  - int thread_sem_fd -> fd del semaforo del thread
- Semantica:

  Viene chiamata la semWait su sem_empty_fd e su thread_sem_fd. Viene calcolato l'indice di coda dove poter

inserire il messaggio e viene incrementata la size. Si chiama la funzone semPost su thread_sem_fd e su sem_full_fd.

f. FixedSizeMessageQueue_popFront

- Parametri Input:
  - FixedSizeMessageQueue* q -> puntatore alla coda da eliminare
  - int sem_empty_fd -> fd del semaforo empty
  - int sem_full_fd -> fd del semaforo full
  - int thread_sem_fd -> fd del semaforo del thread
- Valore di ritorno:

  Messaggio dalla coda.
- Semantica:

  Si chiama la funzione semWait su sem_full_fd e su thread_sem_fd. Viene memorizzato il messaggio che si trova in testa e viene aggiornato l'indice di testa e la dimensione della coda. Viene chiamata la funzione semPost su thread_sem_fd e su sem_empty_fd.

Per la realizzazione delle code abbiamo utilizzato il PoolAllocator che si occupa di allocare le code e i messaggi, sono quindi state aggiunte variabili per far si che tutto funzionasse correttamente.

In tutti i casi, abbiamo dovuto modificare alcuni file del sistema DisastrOS per far sì che tutte le funzioni da noi implementate si integrassero con il resto dell'ambiente (e.g. sono stati aggiunti gli handlers delle system call nei file disastrOS.c con i relativi prototipi nel file disastrOS.h).

Dopo l'implementazione dei semafori e delle code di messaggi sono stati realizzati due file di test per verificare il corretto funzionamento dei semafori stessi sia per la protezione della sezione critica sia per l'utilizzo delle code di messaggi.

Implementazione Test

1. simulazione di una sezione critica acceduta da più processi.
2. implementazione delle Message Queue proposte a lezione utilizzando i semafori di DisastrOS.

Compilazione ed Esecuzione:

Per compilare è presente un Makefile che genera i due eseguibili dei test, è quindi sufficiente usare il comando *make* nella CLI.

Per eseguire: *./disastrOS_queue_test* oppure *./disastrOS_cs_test* .