Relazione Tecnica
Sistemi Operativi Laboratorio

Reazione a catena

Gabriele Amara
1044619

Introduzione

Reazione a catena è il progetto di sistemi operativi dell’anno 2023/24, descritto brevemente consiste in una simulazione di gestione di un insiemi di atomi.

Finalità

La finalità del progetto è la comprensione profonda della programmazione concorrente mediata della realizzazione dello stesso. Per comprensione della programmazione concorrente si intende piena padronanza dei processi/thread, IPC, segnali e concetti come scheduling/interleaving (CPU), sezione critica e molto altro...

Ambiente / Strumenti

Ambiente di sviluppo è Unix/Linux e il linguaggio scelto per la realizzazione/corso è C. La scelta di C sembra essere uno standard per i sistemi operativi (linguaggio del kernel linux) inoltre offre una vicinanza di basso livello per poter interagire con hardware in modo ottimale. Come strumenti secondari derivati dell'ambiente e il linguaggio sono presenti man (Linux Programmer's Manual), bash, utility make, env e molto altro…

Architettura

Architettura del progetto giustifica/spiega la maggior parte delle scelte progettuali. Verrà suddivisa e trattata in due sotto sezioni di uguale importanza: comunicazione e software.

Architettura di comunicazione

Per architettura di comunicazione in questo documento si intendono i meccanismi/protocolli di comunicazione che gli enti presi in esame dovranno utilizzare per poter rispettare le specifiche della consegna.

Nel caso specifico Reazione a catena è una simulazione di gestione di atomi, una specie di centrale nucleare. La simulazione dovrà gestire diversi enti: master, attivatore, alimentazione, inibitore, atomi ognuno di essi è rappresentato tramite un processo perciò per interagire dovranno utilizzare strumenti di IPC.

I possibili approcci e soluzioni sono molti perciò verranno proposti sono quelli vagliati e infine ne verrà scelto uno giustificando le motivazioni della scelta.

Ognuno degli approcci dovrà coprire diversi requisiti/esigenze implementative ovvero: massimizzare la concorrenza, utilizzare semafori, memoria condivisa, code di messaggi.

Comunicazione scissione

La comunicazione della scissione da parte del attivatore agli atomi è il centro del progetto, rappresenta l'azione che dovrà essere ripetuta più volte, e se necessario deve essere comoda da inibire (Guardare meccanismi di inibizione).

I principali approcci che sono emersi riflettendo sulle specifiche sono: utilizzo di un semaforo, utilizzo della coda di msg.

Approfondendo l'utilizzo del semaforo contatore davanti azione di scissione dell'atomo ci permette di non dover utilizzare i pid dei singoli atomi per la comunicazione, avere una gestione adattiva del numero di atomi da scindere, una "randomicità" implementata con lo scheduler, una facile inibizione tramite la concessione di n scissioni da parte della comunicazione (attivatore, inibitore) (mediante il calcolo del caso peggiore). Però comporta degli evidenti svantaggi ovvero mancata flessibilità di scelta dell'attivazione su atomi, mancanza di dati di fine simulazioni sugli atomi (non richiesto ma utile) per analisi postere alla simulazione, mancata scalabilità di azioni degli atomi ovvero la mancanza di payload comporta che atomo possa fare solo una azione.

Approfondendo l’utilizzo della coda di messaggi come mezzo di comunicazione per la scissione risolve tutti i problemi che comporta l'utilizzo del semaforo, però comporta onere di dover salvare i pid degli atomi per poter comunicare la scissione. Da notare che la coda di messaggi potrebbe essere leggermente meno performante dei semafori ma in questo contesto non conta. Per quanto riguarda inibizione è ugualmente flessibile con la deviazione del messaggio di scissione.

In conclusione verrà utilizzata la coda di messaggi, per la comunicazione di scissione, nonostante entrambi gli approcci forniscono una attesa passiva come da specifica.

Meccanismi di inibizione

I meccanismi di inibizione sono strettamente legati al meccanismo di comunicazione della scissione come detto al paragrafo precedente.

Nel caso di utilizzo di semaforo si dovrà usare un IPC per poter far comunicare attivatore con inibitore per potersi accordare/concedere un numero di attivazioni “sicuro” per il sistema secondo le logiche del inibitore.

Invece con utilizzo della coda di messaggi la comunicazione tra inibitore e attivatore potrebbe avvenire in più modi con feedback o senza (es. protocolli di rete). Per questo progetto non sembra essere necessario un feedback perciò la comunicazione può avvenire secondo lo schema: se inibitore è attivo attivatore richiede il permessi per far scendere n atomi con un messaggio che se accettati a loro volta verranno inviati ai rispettivi atomi, invece se inibitore è disattivato allora attivatore invierà ordine di scissione direttamente.

Secondo quando detto in comunicazione di scissione verrà utilizzata la coda di messaggi senza feedback.

Inoltre inibitore implementare dei meccanismi di assorbimento per evitare che nonostante sia attiva la scissione di un atomo comporti esplosione (attivazione tardiva). L'assorbimento avviene tramite invio di un segnale SIGUSR1 da parte del primo atomo che dovrebbe comunicare l'esplosione che attiverà il meccanismo del inibitore secondo delle logiche di caso peggiore.

Utilizzo dati globali

Quasi per definizione alcuni dati della simulazione sono globali ovvero vengono/potrebbero essere utilizzati da tutti perciò ha senso inserirli anche in ottica espansiva del progetto della memoria condivisa. Questo è il caso dei dati che il master dovrà stampare ogni secondo le “statistiche”.

Come unico “difetto” la memoria condivisa ha l'obbligo della sincronizzazione che però con i semafori è molto comoda. Inoltre è molto veloce e intuitiva da utilizzare (un grosso buffer).

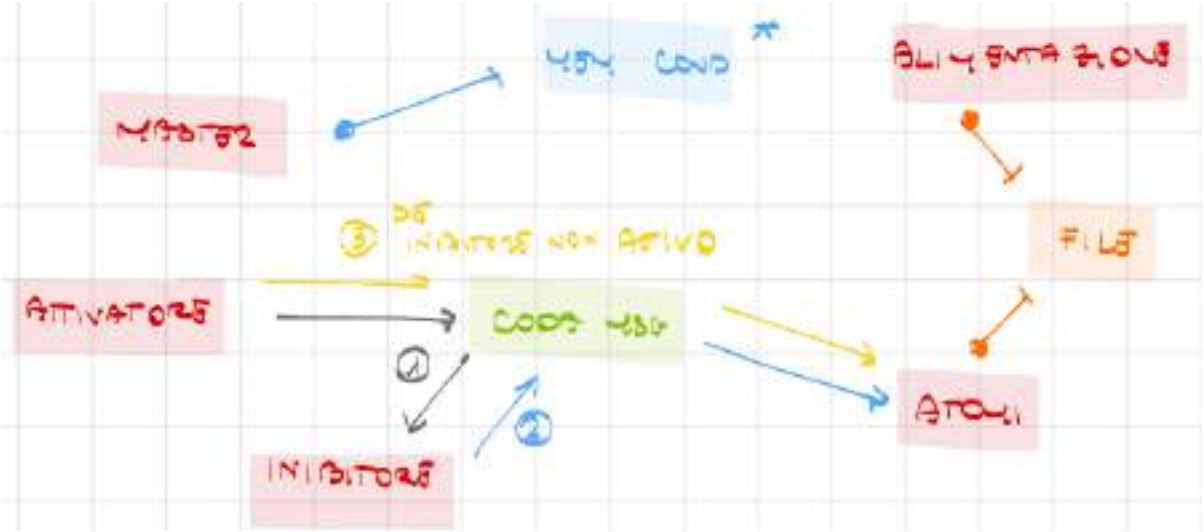
Utilizzo dati di dimensione variabile

Oltre ai dati della memoria condivisa di dimensione fissa utilizzando la scelta implementativa della coda di messaggi si dovranno salvare i pid degli atomi però essendo una quantità di dati variabile non dovrebbero essere inseriti della memoria condivisa poiché comporterebbe il suo ridimensionamento constate il che sarebbe scomodo e inefficiente. Perciò come memoria a dimensione variabile è stato scelto un file che conterrà informazioni sugli atomi compreso i pid. L'utilizzo del file potrebbe sembrare lento ma in realtà torna molto utile per diverse ragioni scalabilità della scelta, logica del inibitore, analisi postere dei atomi. Inoltre il file viene utilizzano in modo che le ricerche su di esso siano ridotte al minimo (per evitare una crescita di tempo di esecuzione al crescere del input).

Terminazione simulazione

La terminazione sono di un insieme di operazioni estremamente importanti però il sistema su cui verrà eseguita la simulazione poiché potrebbe "inquinare"/danneggiare lo stesso se non fatta correttamente (troppi zombie, IPC non eliminate ...).

Il meccanismo di terminazione più sensato e meno complicato da implementare si basa sull'utilizzo dei segnali (SIGTERM, SIGKILL...) che per comodità verrà implementato tramite killpg.



Architettura del software

Per architettura software in questo documento si intende la gestione del codice in ogni sua forma: gestione dei moduli, design del codice, utilizzo di determinate funzioni ecc.

Come principio cardine in questo progetto è stata utilizzata la chiarezza e il design del codice e non le prestazioni. Questo perché per il mio parere è molto più importante che il progetto sia chiaro e facilmente manutenibile (di conseguenza correggibile) che performante in maniera estrema.

Struttura progetto

La struttura del progetto è stata decisa sempre per massimizzare la chiarezza e per scelte riguardanti la compilazione trattare in Makefile gestione compilazione. Per ragioni di formattazione la struttura non è stata riportata (usare il comando tree).

Makefile gestione compilazione

La gestione della compilazione è stata fatta tramite utility make, brevemente è un makefile che gestisce tutti i file in src in modo ricorsivo tramite utilizzo dei vpath e pattern generici, si avvale anche di strumenti come gcc -MM per generare dipendenze.

La gestione della compilazione è stata la cosa più difficile e tediosa del progetto, alla fine si ottiene il risultato che si desidera ma non è soddisfacente.

In particolare i vpath dei target falliscono la prima volta che cercano di essere risolti però la seconda volta vengono risolti (nel makefile viene specificato il problema nello specifico).

Gestione dei moduli

Idea generale per la gestione dei moduli era di poter ripetere il meno possibile codice ed renderlo semanticamente coerente ed testabile, ne consegue che ogni IPC e risorsa avrà il proprio modulo (.c/.h), con struttura più omogenea possibile per rendere il codice più leggibile (condizioni interne). Ogni eseguibile necessariamente avrà il proprio modulo .c. Inoltre sono presenti moduli di utilità per operazioni utilizzate che non hanno un vero è proprio posto semantico.

Scrittura del codice

La scrittura del codice è stata fatta utilizzando le pratiche di design del codice come la scrittura di funzioni/macro che possono incapsulare dati/operazioni tramite scope, per citarne

una mutex_critical_section è stata implementata per ridurre al minimo la possibilità di errore utilizzando un mutex (ispirata dalle callback come stile, anche se non crea un record di attivazione vero, poi è sincrona sul processo stesso se il codice è sincrono).

Inoltre per specifica non è stato possibile utilizzare strumenti non standard di c che però il compilatore di gcc supporta come le closure (con cui avrei probabilmente implementato delle classi).

Infine è stato fatto un grande uso di comment, enum e strutture sintattiche che potessero ulteriormente migliorare la leggibilità del codice in generale.