UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Relazione di accompagnamento per il progetto di Sistemi Operativi

Autore:
Marco Pampaloni

Anno Accademico 2019/2020

1 Introduzione

Il progetto descritto in questo elaborato è da intendersi come soluzione proposta per il progetto finale semplificato relativo al corso di Sistemi Operativi. La struttura del documento ricalca, per semplificare la trattazione, quella del codice sorgente sviluppato. Il codice è stato sviluppato e testato su una macchina Ubuntu 20.04 a 64 bit ed è conforme POSIX nella sua interezza, non includendo altre dipendenze se non la libreria *pthread*. Tutto il codice è versionato e ad ogni iterazione è stato testato con valgrind in modo da garantire l'assenza di memory leak. Inoltre sono stati implementati vari test di unità eseguibili automaticamente tramite il comando *make test*. Ciò è descritto più in dettaglio nella sezione 2. Infine, il codice è dettagliatamente commentato, sia nella specifica che nell'implementazione. I commenti al codice sono complementare a questo elaborato e descrivono precisamente il funzionamento delle singole funzioni, oltre alla loro specifica formale.

2 Makefile

Il Makefile è stato scritto con in mente l'utilizzo attivo di tecniche di testing automatico. Per questo, diverse funzionalità del programma sono state testate creando dei file che testassero i requisiti dettati dall'implementazione. Il Makefile gestisce la loro compilazione ed esecuzione, oltre a controllare automaticamente l'output dei test, confrontandolo con quello atteso. I test di unità sono poi ricchi di asserzioni interne volte a verificare la correttezza dei post e prerequisiti.

Il Makefile presenta tre target di tipo phony: *test, test2* e *clean*. Il target test esegue tutti i singoli test di unità e, solo se terminano con successo, procede all'esecuzione del test di simulazione test2 prodotto secondo i termini della specifica di progetto. Infine il target clean ripulisce la directory del progetto da tutti i file temporanei, file oggetto ed eseguibili.

3 Script di analisi

Lo script Bash *analisi.sh* parsa il file di log prodotto dalla simulazione e fornisce in output (su di un file) e in bella forma le informazioni raccolte durante l'esecuzione del programma. Lo script utilizza prevalentemente i comandi *cut, grep, sed* e *awk* per formattare il file di log e verificare alcune condizioni sui valori prodotti. Poichè la specifica non era molto rigorosa, i test implementati sono vari e i valori soglia sono arbitrari.

4 Strutture dati e funzioni di utilità

Per coadiuvare lo sviluppo e la modularità del codice, sono state implementate varie strutture dati e funzioni di utilità, tra le quali una coda FIFO thread safe (*queue.h, queue.c*), un parser per il file di configurazione della simulazione (*parser.h, parser.c*), una threadpool per gestire la creazione e l'esecuzione concorrente della clientela (*threadpool.h, threadpool.c*), un cronometro utilizzato per misurare gli intervalli di tempo impiegati dal programma per eseguire sotto-task o per fini statistici (*stopwatch.h, stopwatch.c*) e delle funzioni di logging thread safe per scrivere su disco statistiche di utilizzo.

Ognuna delle strutture dati implementate fornisce anche le funzionalità necessarie a liberare la memoria occupata dal loro utilizzo. Nel seguito saranno descritte le strutture più importanti, lasciando la documentazione di quelle minori all'implementazione stessa.

4.1 Queue

Il file queue.c implementa una coda FIFO unidirezionale e fornisce le funzionalità di push, pop, top, size, empty e l'operazione funzionale map. La coda è thread-safe nelle sue operazioni di base, le quali garantiscono l'esecuzione atomica delle istruzioni contenute. Per questo motivo la coda è sempre passata alle funzioni come puntatore (a dati non costanti) poichè il lock è contenuto all'interno di essa ed è unico per ogni coda.

4.2 Threapool

Il file threadpool.c implementa una threadpool di dimensione fissa. Ad una threadpool si possono sottomerre dei task, i quali saranno eseguiti da un singolo thread non appena ve ne sarà uno disponibile. Inizialmente la threadpool alloca il massimo numero di thread specificati al momento della creazione e ognuno si mette in attesa di task su una coda condivisa e thread safe.

La threadpool è utilizzata all'interno del progetto per gestire l'esecuzione dei thread clienti e la loro liberazione dalla memoria. Un thread dedicato all'interno dell'entry point del programma (*simulazione.c*) si occupa invece di gestire la creazione della threadpool e quella scaglionata di tutti i clienti.

4.3 Logger

Il file logger.c implementa le funzionalità di logging utilizzate dai vari thread della simulazione. La chiamata a *log_write()* comporta la scrittura su disco del contenuto della stringa formattata passata come parametro alla funzione. Ogni chiamata è thread safe e la scrittura concorrente su file è garantita che sia completata prima di passare il controllo a un nuovo thread. Il logger è inizializzato durante la creazione del supermercato e deallocato solo dopo la sua chiusura.

5 Supermercato

Il file supermercato.c fornisce l'implementazione delle funzionalità richieste per la simulazione di un generico supermercato. Il supermercato non è un attore della simulazione, poichè non gli è fornito un thread indipendente su cui svolgere il lavoro, ma è bensì una struttura dati utilizzabile tramite l'interfaccia definita dal file supermercato.h. Tale interfaccia permette di creare un supermercato, allocandolo dinamicamente sullo heap, aprire o chiudere una cassa (la cassa è scelta internamente dall'implementazione), posizionare un cliente in coda ad una cassa aperta (arbitrariamente scelta dall'implementazione) e chiudere il supermercato, aspettando la terminazione di tutte le casse ancora attive. Alla creazione del supermercato, la quale richiede un record contenente i parametri di configurazione, vengono inizializzati tutti i cassieri e creati i thread dei singoli cassieri attivi, il cui numero è pari a quanto richiesto dalla simulazione e fornito tramite il record di configurazione. Alla chiusura (*close_supermercato()*) vengono processati i dati statistici raccolti dai vari cassieri, aggregati e scritti sul file di log. I log relativi ai singoli clienti sono invece prodotti durante la simulazione dai thread clienti.

6 Cassiere

Il Cassiere è un attore della simulazione e gli è concesso un proprio thread di esecuzione. Anche esso è implementato come una struttura dati conforme a una interfaccia in stile API POSIX, definita nel file cassiere.h. Le funzionalità proposte consentono di ottenere le informazioni di attività del cassiere (id, stato di esecuzione, ecc.) e di causare l'apertura/chiusura delle casse.

Il thread del cassiere è implementato come un loop che termina quando viene ricevuto un segnale di chiusura dall'esterno (viene settato lo stato interno del cassiere). All'interno del ciclo di vita, il cassiere processa la sua coda clienti, la quale è personale e protetta nell'accesso da una lock. Quando la coda è vuota il cassiere si mette in attesa di nuovi clienti, per un periodo lungo al più S secondi. Quest'ultimo è un parametro di configurazione fornito dall'utente e indica l'intervallo di tempo passato il quale, il cliente deve comunicare al direttore il numero di clienti correntemente in coda alla cassa. Poichè la specifica non era esaustiva sotto questo punto, è stato scelto di rispettare questo intervallo cercando sempre di servire prima il cliente e, qualora durante il tempo di servizio fosse stato necessario di comunicare con il direttore, effettuare la comunicazione e tornare a servire il cliente. Ciò permette di sfruttare al massimo le risorse del sistema e di non danneggiare il tempo di servizio del cassiere. Il processamento del singolo cliente è simulato con una chiamata alla funzione *nanosleep()*.

Quando un cassiere termina, esso segnale la sua terminazione a tutti i clienti attualmente in coda, i quali verranno ricollocati in altre casse aperte.

7 Cliente

Il Cliente è un altro attore della simulazione e la sua interfaccia è specificata dal file cliente.h. Poichè i thread clienti sono numerosi e il loro ciclo di vita teoricamente minimo, questi vengono gestiti tramite una threadpool, la quale ricicla i thread degli utenti ormai terminati. La dimensione della threadpool è pari al massimo numero di clienti che possono essere presenti all'interno del supermercato.

Il cliente, dopo aver trascorso un tempo variabile in attesa per simulare la scelta degli acquisti, si mette in coda ad una cassa si mette e attende di essere servito dal cassiere sulla sua variabile di condizione. Quando il cassiere termina di processare il cliente, questo glielo segnala e il cliente si risveglia, terminando la sua esecuzione e rilasciando le risorse alla threadpool, la quale potrà fornirle ad altri task.

8 Direttore

Il thread direttore è l'ultimo attore della simulazione e la sua interfaccia è specificata dal file direttore.h. Esso svolge sostanzialmente la verifica del carico di lavoro sulle casse all'interno del supermercato. Durante il suo ciclo di vita, il direttore controlla periodicamente i dati ricevuti dai cassieri relativi ai clienti in coda e decide se aprire o meno le casse, in base a dei valori soglia $S1$ e $S2$, forniti dal file di configurazione. La principale aggiunta dell'implementazione riguarda un dettaglio non specificato dalla richiesta del progetto, ovvero dopo quanto e con che frequenza il direttore deve aprire o chiudere una cassa al verificarsi delle condizioni necessarie. Un problema immediato lo si può verificare quando la condizione per l'apertura di una cassa è verificata: il direttore apre conseguentemente una nuova cassa, se disponibile e si rimette in attesa di nuovi eventi. Tuttavia le condizioni che erano precedentemente verificate, rimangono tali per un intervallo di tempo anche molto lungo. Ciò comporterebbe l'apertura quasi istantanea di tutte le casse e lo stesso si verificherebbe per la loro chiusura, risultando in un comportamento sicuramente non corretto.

Per ovviare al problema descritto, si è deciso di implementare una condizione di "pazienza" per il direttore, il quale prima di prendere una decisione riguardo l'apertura/chiusura di una cassa, aspetta che gli siano arrivate un certo numero di comunicazioni da parte dei clienti. In tal modo dopo l'apertura o la chiusura di una cassa, il supermercato ha il tempo di stabilizzarsi e di dividersi il carico di lavoro tra le nuove casse aperte.

9 Simulazione

Il file simulazione.c è l'entry point del programma. Al suo avvio avviene la registrazione dell'handler dei segnali POSIX, il quale gestisce i segnali SIGHUP e SIGQUIT, come da specifica. L'handler non fa altro che settare atomicamente lo stato interno della simulazione alla ricezione di un segnale, causandone la terminazione dipendentemente dal valore dello stato.

Il passo successivo è il parsing del file di configurazione, cercato di default nella directory locale, ma specificabile tramite argomento da linea di commando tramite la sintassi *./simulazione [-c configpath]*.

Viene poi creato il thread cha ha il compito di allocare i nuovi clienti e sottomettere il loro thread alla threadpool. Il thread principale si mette poi in attesa della ricezione di segnali e infine termina liberando tutte le risorse allocate. Il thread di creazione dei clienti utilizza dei cleanup handler per gestire la deallocazione delle risorse e attendere tutti i clienti in caso di terminazione tramite segnale SIGHUP.

10 Interdipendenze e analisi della concorrenza

Il seguente grafico illustra le interazioni tra i vari attori della simulazione, a cui vi è incluso il supermercato poichè durante l'esecuzione del programma si comporta da tramite per le varie funzionalità.

Le linee tratteggiate descrivono chiamate di funzioni in cui il lock del chiamante non è acquisito. Sono quindi chiamate safe che non possono conseguire delle deadlock. Le linee spesse sono invece invocazioni con lock

acquisito. Il senso delle frecce stabilisce infine l'ordine tra chiamante e chiamato, mentre l'assenza di cicli di linee piene mostra che non si possono verificare deadlock (a patto che i lock siano gestiti correttamente).

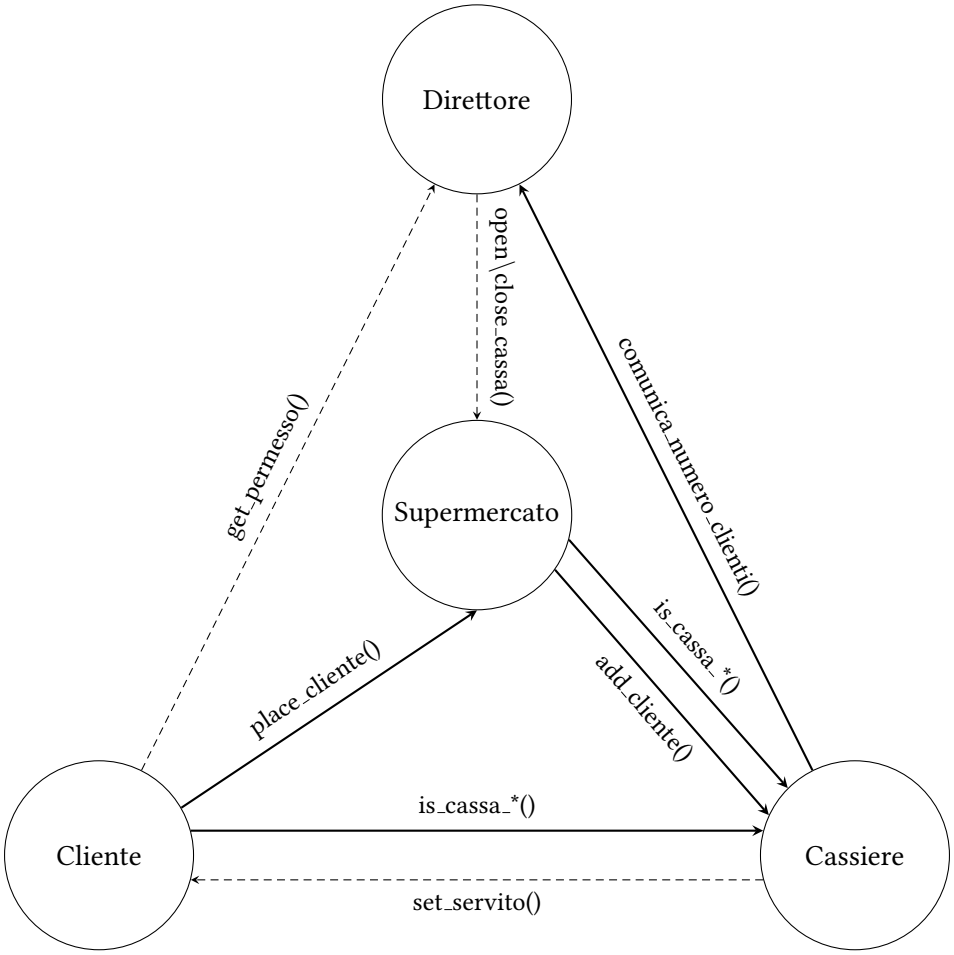

Figura 1: Analisi grafica delle interdipendenze tra i vari attori della simulazione