STRUTTURA	ATTRIBUTI SIGNIFICATIVI	DESCRIZIONE	CONDIVISA CON
Chiusura	**int** *codice*	Assume 0 all'inizio, 1 dopo SIGHUP, 2 dopo SIGQUIT.	SH, Cliente, Cassiere, Direttore
TotClienti	**int** *n*	Assume 0 all'inizio. Incrementata dal direttore quando crea un cliente. Decrementata dal cliente quando sta uscendo.	Direttore, Cliente, Cassiere
permessi	**int** *len* **elem_permessi** **inizio* **elem_permessi** **fine*	*len* indica il totale di elementi in lista; *inizio* e *fine* sono puntatori agli elementi testa e coda di una lista di strutture *elem_permessi*.	Direttore, Cliente
elem_permessi	**int** *confermato* **int** *flag_ultimo*	*confermato* assume 0 all'inizio, 1 quando direttore dà il consenso di uscita al cliente; *flag_ultimo* assume 1 per terminare l'attesa di altri consensi quando estratto.	Direttore, Cliente
config	**(un intero per ogni valore nel file di configurazione)**	memorizza tutti i valori estratti dal file di configurazione.	\
Aggiornamento [array]	**int** *len_coda*	*len_coda* indica il totale di clienti in coda ad una singola cassa; il valore viene aggiornato dal cassiere ad ogni intervallo prestabilito e letto dal direttore per la gestione delle casse.	Cassiere, Direttore
Cassa [array]	**int** *len* **int** *flag_chiusa* **int** *tot_servizi* **int** *tot_chiusure* **int** *tot_prodotti* **elem_coda** **inizio* **elem_coda** **fine* **int** *flag_in_uso* **int** *tot_cassieri*	*len* indica il numero di clienti in coda; *flag_chiusa* assume 1 quando la cassa non è in funzione; *tot_servizi* indica il numero di clienti che sono stati serviti; *tot_chiusure* indica il numero di volte in cui la cassa è stata chiusa; *inizio* e *fine* sono puntatori agli elementi testa e coda di una lista di strutture *elem_coda*; *flag_in_uso* assume 1 se c'è un cassiere che sta utilizzando la cassa; *tot_cassieri* indica quanti cassieri vogliono utilizzare in contemporanea la cassa.	Direttore, Cliente, Cassiere, SH
elem_coda	**int** *my_p* **int** *flag_servito* **elem_coda** **prima* **elem_coda** **dopo*	*my_p* indica il totale di prodotti acquistati dal cliente; *flag_servito* assume 0 all'inizio, 1 quando il cassiere termina di servire il cliente, -1 se la cassa è stata chiusa; *prima* e *dopo* sono puntatori ai clienti che rispettivamente precededono e seguono il cliente identificato da questa struttura (lista doppiamente collegata).	Direttore, Cliente, Cassiere
Log	**FILE** **fp*	*fp* è un puntatore al file di log aperto in scrittura.	Cliente, Cassiere
join	**elem_join***inizio **elem_join** **fine*	*inizio* e *fine* sono puntatori agli elementi testa e coda di una lista di strutture *elem_join*.	Direttore
elem_join	**pthread_t** *id* **int** *flag_ultimo*	*id* memorizza un thread in esecuzione di cui aspettare la terminazione tramite pthread_join(); *flag_ultimo* assume 1 per terminare l'attesa di altri thread quando estratto.	Direttore

(*) Tutte le strutture elencate sono condivise con l'entità *Supermercato*, in quanto è questa ad allocarle.
(*) mutex_t e **cond_t** non sono stati riportati tra gli attributi di questa tabella.

LEGENDA: **grassetto** per nomi (o simili) di strutture dati, sottolineato per nomi (o simili) di attributi.

ENTITA'	THREAD	DESCRIZIONE
CLIENTE	*th_cliente()*	Calcola casualmente il totale di prodotti acquistati e attende un tempo casuale. Se il totale è nullo si accoda in fondo alla **lista permessi** e attende la conferma di uscire dal direttore. Altrimenti scansiona casualmente l'**array di Casse** alla ricerca di una aperta. Se durante la scansione **Chiusura** ha codice 2, allora esce. Altrimenti si accoda alla cassa trovata e attende di essere servito. Se la cassa viene chiusa mentre è in attesa, allora riparte dalla ricerca di una cassa aperta. In fase di uscita decrementa il **totale di clienti** nel supermercato.
CASSIERE	*th_cassiere()*	Incrementa il totale di cassieri che vogliono usare la cassa e attende finché questa non è più in uso. Crea il thread *th_cassiere_agg()*. Se la lunghezza della coda è nulla e la cassa non è chiusa verifica se c'è un altro cassiere destinato a questa cassa terminando o se il **totale di clienti** nel supermercato è nullo e se **Chiusura** ha un codice di terminazione così da uscire, altrimenti attende. Se la cassa viene chiusa, avverte tutti i clienti in coda impostando i flag_servito a -1, rimuovendoli dalla coda e terminando. In tutti gli altri casi estrae un cliente dalla coda, aggiorna il totale di prodotti scansionati, attende un tempo costante ed uno lineare al numero di prodotti e avverte il cliente di aver finito di servirlo. Il ciclo si ripete dal controllo della lunghezza della coda e della chiusura della cassa. Al termine decrementa il totale di cassieri e la cassa non è più in uso.
	th_cassiere_agg()	Attende l'intervallo di tempo stabilito, legge il totale di clienti in coda alla **cassa** e lo memorizza nell'**array di aggiornamento**. Se la cassa è chiusa, termina. Se c'è un cassiere che deve sostituirlo, termina. Se il **totale di clienti** nel supermercato è nullo e **Chiusura** ha un codice di terminazione, termina. In tutti gli altri casi, il ciclo si ripete.
DIRETTORE	*th_direttore()*	Crea *th_direttore_c()* e *th_direttore_p()*. Crea i *th_cliente()* iniziali. Attende che escano almeno E clienti. Se **Chiusura** ha un codice di terminazione accoda ultimo elemento alla lista **join** clienti, attende che la lunghezza della lista **permessi** sia nulla e vi accoda l'ultimo elemento, terminando. Altrimenti crea un nuovo *th_cliente()*, lo accoda alla lista **join** clienti ed incrementa il **totale** di clienti. Il ciclo si ripete dal controllo del codice per E volte, poi riparte dall'attesa di uscita di E clienti.
	th_direttore_c()	Crea i *th_cassiere()* iniziali, impostando le rispettive casse nell'**array di casse** come aperte. Attende lo stesso intervallo di comunicazione dei cassieri, scorre le casse aperte e legge dall'**array di aggiornamento** le lunghezze delle loro code. Verifica i requisiti per l'apertura di una nuova cassa e la creazione di un nuovo *th_cassiere()*, altrimenti controlla i requisiti per la chiusura di una cassa. Se il **totale di clienti** nel supermercato è nullo e se **Chiusura** ha un codice di terminazione termina. Il ciclo si ripete dall'attesa intervallo.
	th_direttore_p()	Attende almeno un cliente in **lista permessi**. Estrae un cliente. Se il cliente è l'ultimo, termino, altrimenti avverto il cliente di aver confermato la sua uscita senza prodotti. Il ciclo si ripete dall'attesa di un cliente nella lista permessi.

SIGNAL HANDLER	*th_signal_handler()*	Imposta la maschera dei segnali per riceverli tutti. Attende l'arrivo di un segnale. Se riceve SIGHUP, imposta ad 1 il codice di **Chiusura** e termina. Se riceve SIGQUIT, imposta a 2 il codice di **Chiusura**, chiude ogni cassa nell'**array di casse** e termina. Per tutti gli altri segnali non avviene niente.
SUPERMERCATO	*main()*	Blocca tutti i segnali, quindi blocca tutti i segnali dei thread successivamente creati. Legge il file di configurazione estraendo i dati nella struttura **config**. Apre in scrittura il file di log. Alloca la struttura **Chiusura**, l'**array di casse**, l'**array di aggiornamenti**, i **dati signal handler**, la **lista join** per i clienti, la **lista permessi**, la struttura col **totale di clienti**, i **dati cassiere**, i **dati cliente**, i **dati direttore**. Crea *th_signal_handler* e *th_direttore*. Attende almeno un thread cliente in lista join, lo estrae per terminarlo e ripete finché non estrae l'ultimo. Sveglia le casse rimaste aperte per farle terminare. Attende il termine di *th_signal_handler* e *th_direttore*. Aggiorna il file di log. Dealloca tutte le strutture precedentemente create e termina.

RELAZIONI ENTITA'-THREAD

Per ogni *entità* sono state individuate delle *macro-operazioni* eseguibili in modo concorrente. A ciascuna di queste è stato associato un *thread*, come mostrato nella precedente tabella. Le relazioni tra *entità* e *thread* prediligono il parallelismo rispetto alla programmazione sequenziale, dove questo sia applicabile.
I benefici che si intende raggiungere col citato approccio riguardano il garantire la precisione degli intervalli di attesa nella comunicazione tra singolo **Cassiere** e **Direttore** ed evitare che i tempi di esecuzione di una *macro-operazione* costituiscano fonte di ritardo per un'altra nel caso di raggruppamenti sequenziali di codice. In tal modo, l'alternarsi delle *macro-operazioni* è gestita dallo Scheduler, il che evita una gestione dinamica degli intervalli di attesa e gli inevitabili ritardi che ciò comporterebbe, garantendo anche una maggiore leggibilità e manutenzione del codice sorgente.

CONDIVISIONE DATI TRA ENTITA'

Ogni *entità* dispone di tutti e soli i dati su cui può operare. L'entità **Supermercato** è l'unica ad avere accesso ad ogni struttura dati, in quanto è l'entità all'interno della quale operano tutte le altre. E' compito dell'entità **Supermercato** limitare e raggruppare i parametri estratti dal file di configurazione e le strutture dati condivise, da essa allocate, in strutture specifiche per le altre entità: *DatiCliente*, *DatiCassiereCondivisi*, *DatiDirettore*, *DatiSignalHandler*. Queste strutture saranno gli argomenti dei rispettivi thread al momento della creazione. L'unica eccezione riguarda il thread *th_direttore_c()* che ha accesso all'**array di casse** nel quale è memorizzata anche la lunghezza attuale di ciascuna coda. Tale dato non è comunque letto, poiché le lunghezze comunicate dai cassieri dalle quali il thread determina l'apertura e la chiusura delle casse sono memorizzate nell'**array aggiornamento**.

COMUNICAZIONE CASSIERE-DIRETTORE

Ogni Cassiere, dal momento in cui viene creato fino alla terminazione, memorizza ad intervalli regolari la lunghezza della coda alla sua cassa nell'**array aggiornamento**. Il Direttore utilizza lo stesso intervallo per attendere gli aggiornamenti dei cassieri e ciclicamente controllare l'**array aggiornamento** per stabilire eventuali aperture o chiusure di cassa.

POLITICA DI APERTURA E CHIUSURA CASSE

Supponendo che si verifichino le condizioni per l'apertura o la chiusura di una cassa come da specifica, il thread *th_direttore_c()* seleziona **casualmente** la nuova cassa da aprire tra quelle al momento chiuse e come cassa da chiudere una cassa aperta tra le casse che hanno una coda clienti lunga al più uno. La chiusura di una cassa ha la precedenza sull'apertura e quest'ultima non viene effettuata se valgono le condizioni per chiuderne una. La **casualità** è stata introdotta per distribuire uniformemente le aperture e le chiusure.

GESTIONE INDIPENDEZA CASSA-CASSIERE

Cassa è la struttura dati dinamica che compone l'**array di casse** ed esiste per tutta la durata del processo. **Cassiere** è un'entità composta di due thread creata quando *th_direttore_c()* apre una nuova cassa e che termina quando la cassa su cui sta operando viene chiusa. Apertura e chiusura di una **Cassa** consistono nel cambiamento di valore di un flag in essa memorizzato. Ogni **Cassiere** ha accesso ad una sola **Cassa** nell'**array di casse**.
Un particolare caso si verifica quando il thread *th_direttore_c()* chiude una **Cassa** e immediatamente dopo la riapre, con conseguente creazione di un nuovo **Cassiere** erede della cassa. E' infatti possibile che il **Cassiere** che vi operava precedentemente non si accorga dell'avvenuta chiusura della cassa, ad esempio perché stava servendo un cliente e al termine del servizio continui ad operare in quanto la cassa risulta aperta. Al fine di garantire l'uso di una specifica **Cassa** ad al più un **Cassiere** per volta, sono stati introdotti nella struttura dati della **Cassa** gli attributi *flag_in_uso* e *tot_cassieri*: *flag_in_uso* permette al nuovo **Cassiere** di attendere la fine del precedente e *tot_cassieri* è utilizzato come flag di terminazione per il **Cassiere** attuale che, verificando la presenza di altri **Cassieri** destinati alla sua cassa, riconosce l'altrimenti non rilevato stato di chiusura.

ATTESA DI EVENTI

Premettendo che tutte le strutture dati condivise, i cui valori vengano modificati in fase di esecuzione da più entità, sono accedute in mutua esclusione tramite meccanismi di *lock* e *unlock*, sono state utilizzate le istruzioni *wait* e *signal* per l'attesa passiva e la segnalazione di condizioni a tempo indefinito.
Ogni Cliente in **coda** alle casse o in **lista permessi** ha una propria variabile di condizione per attendere la fine del servizio e segnala rispettivamente al Cassiere o al Direttore di essersi messo in coda se questa era vuota. Inoltre, all'uscita, il Cliente sveglia il Direttore per fargli controllare il **totale di clienti** e nel caso farne entrare di nuovi.
Ogni Cassa dispone di due variabili di condizione: una per far attendere il nuovo Cassiere finché la Cassa è occupata dal precedente e l'altra per attendere l'arrivo di un Cliente in coda. Il Cassiere segnala quindi al Cliente la fine del servizio o la chiusura della Cassa e sveglia un eventuale sostituto prima di terminare.
Il Direttore attende che ci sia almeno un cliente in **lista permessi** per successivamente farlo uscire ed attende che il **totale di clienti** scenda di E per creare nuovi Clienti. Inoltre sveglia i Clienti senza prodotti in attesa del permesso di uscita.

TIPOLOGIA DEI THREAD E TERMINAZIONE

Tutti i thread utilizzati sono di tipo **non detached**. E' quindi prevista una chiamata *pthread_join()* per ogni thread in esecuzione al fine di rilasciare correttamente le risorse assegnategli. I thread per cui sono previste istanze multiple sono accodati all'atto della creazione in specifiche **liste join**. I thread *th_cliente()* vengono estratti dalla loro **lista join** e rilasciati parallelamente a tutti gli altri thread, mentre i thread *th_cassiere()* vengono estratti e rilasciati in fase di chiusura del Supermercato. Tutti gli altri thread, di cui è presente una sola istanza per esecuzione, sono memorizzati in variabili rilasciate dal thread che si occupa di crearli prima di terminare. Quando non vi sono più thread da creare, viene aggiunto in fondo ad ogni **lista join** un elemento contrassegnato come ultimo, per terminare il ciclo di estrazione dei thread.

GOTO

Etichette destinate al *goto* sono state inserite solo in casi dove la loro assenza avrebbe dato luogo al ripresentarsi di interi settori di codice e sono finalizzate unicamente alla gestione di fallimenti di chiamate di sistema: **[dealloca:]**, nel thread *main()*, seguita da tutte le deallocazioni delle variabili create dal Supermercato, raggiunta in seguito al fallimento di un'allocazione; **[termina:]**, nel thread *th_signal_handler()*, seguita dalla gestione del segnale SIGQUIT; **[fine:]**, nel thread *th_cliente()*, seguita dalle operazioni di terminazione.

ARGOMENTO DEL MAIN

Il processo accetta al più un argomento corrispondente al file di configurazione da utilizzare. Se il file non viene specificato o non esiste, allora viene utilizzato il file "*config.txt"* presente nella cartella "*test"* che corrisponde allo stesso file utilizzato dal comando *make test*.

GESTIONE FALLIMENTI

L'esito delle allocazioni dinamiche, delle funzioni introdotte e di libreria/sistema viene controllato e in caso di fallimento gli errori vengono stampati nello *standard error*. Se il processo non è in grado di continuare, si procede con la terminazione delle varie entità, in alcuni casi generando artificialmente il segnale SIGQUIT o saltando alle specifiche etichette tramite *goto*. Se invece è possibile continuare l'esecuzione, nello *standard error* sarà specificato il tipo di gestione attuato dell'errore.
(!) Non è stato invece gestito l'esito delle funzioni *pthread_mutex_lock, pthread_mutex_unlock, pthread_cond_wait, pthread_cond_signal* per non rendere il codice particolarmente scomodo da leggere (ma è ovviamente una gestione necessaria, comunque simile alle altre presenti nel codice).

LISTE E CODE

Liste e code implementano una gestione di tipo FIFO e non presentano una lunghezza massima.
Sono accompagnate nel codice da puntatori al loro primo ed ultimo elemento per garantire inserimenti ed estrazioni efficienti in O(1). I nuovi elementi sono inseriti in fondo, mentre le estrazioni avvengono in testa.
Le code alle Casse consistono in liste doppiamente collegate. Tale indirizzamento non è necessario in questa versione del progetto ma è stato mantenuto in quanto utilizzato nella versione precedente.

FILE DI LOG

Ogni riga del file di log è composta da un'etichetta seguita da un'insieme di valori intervallati dal punto e virgola. Le etichette sono:
"CLIENTE", seguita dall'ID del cliente, dal tempo trascorso nel supermercato, dal tempo trascorso in coda, dal totale di prodotti acquistati e dal totale di code visitate;
"CASSA", seguita dall'indice della cassa e dal tempo totale per un singolo servizio clienti;
"CASSA_CHIUSA", seguita dall'indice della cassa e dal tempo trascorso dall'ultima apertura;
"FINE_CASSA", seguita dall'indice della cassa, dal totale servizi cliente effettuati dalla cassa, dal totale di chiusure della cassa e dal totale di prodotti dei clienti serviti;
"SUPERMERCATO", seguita dal totale di servizi effettuati dai cassieri e dal totale di prodotti acquistati nel supermercato.
Le righe con etichetta *"FINE_CASSA"* e *"SUPERMERCATO"* vengono inserite quando tutti i clienti sono usciti dal supermercato in chiusura. I clienti senza prodotti hanno il totale di code visitate impostato a zero e non sono considerati nel totale dei clienti serviti.

MAKE FILE

Nel *Makefile* sono presenti le seguenti phony aggiuntive:
[run] esegue il processo supermercato con argomento il file *"config.txt"* presente **all'esterno delle cartelle**;
[debug] identica a **[run]** ma il processo viene controllato con *valgrind*.
Inoltre **[run]** e **[debug]** generano l'eseguibile.

SCRIPT DI ANALISI

Analizza i dati contenuti nel file *"log.txt"* generato al termine dell'esecuzione del processo e contenuto nella cartella *"bin"*. L'output finale dello script rispetta l'ordine crescente degli ID dei clienti attraverso l'uso del comando **sort** eseguito su un file di testo temporaneo (*"not_sorted.txt"*) contenente l'output non ordinato.

[i] Per la corretta visualizzazione del codice sorgente è necessario impostare a quattro spazi la lunghezza del carattere di tabulazione.

Niccolò Campitelli
Matricola 583283
Corso B