

UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA



PROGETTO SISTEMI OPERATIVI

Corso di Laurea in Ingegneria Informatica

Anno Accademico 2023/2024

A cura di

Lorenzo Franceschelli 0327688

Gabriele Monti 0294561

Sistema di Prenotazione Posti



Realizzazione di un sistema di prenotazione posti per una sala cinematografica.

Un **processo** su una **macchina server** gestisce una **mappa di posti per una sala cinematografica**. Ciascun posto è caratterizzato da un **numero di fila**, un **numero di poltrona** ed un **FLAG** indicante se il posto è già stato prenotato o meno.

Il **server accetta** e **processa** le richieste di prenotazione di posti **da uno o più client** (residenti, in generale, **su macchine diverse**). Un client deve fornire ad un utente le seguenti funzioni:

Visualizzare la mappa dei posti in modo da individuare quelli ancora disponibili.

Inviare al server l'elenco dei posti che si intende prenotare (ciascun posto da prenotare viene ancora identificato tramite numero di fila e numero di poltrona).

Attendere dal server la conferma di effettuata prenotazione ed un codice di prenotazione.

Disdire una prenotazione per cui si possiede un codice.

Si precisa che lo studente è tenuto a realizzare sia il client che il server.

Il server deve poter gestire le richieste dei client in modo **concorrente**.

Manuale di utilizzo	6
Scaricare il programma	6
Compilare il programma	6
Avviare i programmi.....	6
Esecuzione simultanea di client e server	7
Installazione di tmux.....	7
Makefile	7
Introduzione	8
Funzionalità del Programma	8
1. Visualizzazione della Programmazione delle Sale.....	9
2. Prenotazione di Biglietti	9
3. Cancellazione di Prenotazioni	10
4. Uscita dal Sistema	10
Componenti Principali	10
Server HTTP (server.c).....	10
Funzione GETrootHandler.....	10
Funzione GETFilmsHandler	10
Funzione GETFilmsListHandler	11
Funzione GETBookShowtimesListHandler	11
Funzione GETFilmHallMapHandler.....	11
Funzione POSTBookSeat	11
Funzione POSTUnbookSeat.....	12
Funzione Main Server	12
Client HTTP (client.c)	13
Funzione printClearedResponse	13
Funzione countLinesOfResponse	13
Funzione parseSeat	13
Funzione bookSeatPages	14
Funzione unBookSeatPage	14
Funzione Main Client	15
Gestione Cinema (cinema.h e cinema.c).....	15
File cinema.c	16
Funzione initializeSeats	16
Funzione createHallsForShowtimes	17
Funzione initFilmsList.....	17

Funzione generateHallMapResponse	18
Funzione bookSeats	18
Funzione saveBookingsToFile	19
Funzione removeBookingFromFile	20
Funzione loadBookingsFromFile	20
Funzione printTicket	21
Il ruolo del prefisso e del suffisso nel biglietto	21
Esempio di un biglietto	22
La funzione generateRandomString	23
File cinema.h	23
Generazione Mappa Cinema (cinemaMap.h e cinemaMap.c)	24
cinemaMap.c	24
Funzione centerMapText	24
Funzione drawSeatNumbers	24
Funzione drawSeparatorLine	25
Funzione generateHallMap	25
cinemaMap.h	25
Parser CSV (filmsCSVparser.h e filmsCSVparser.c)	26
File filmsCSVparser.c	26
Funzioni di Utilità	26
Funzioni di Parsing	26
Funzioni di Calcolo	26
Funzioni di Lettura e Scrittura	27
Funzioni di Stampa	27
Funzioni di Conto	27
Utilità (utils.h e utils.c)	27
Funzione fdeleteBytes	27
Gestione Input Utente (userInput.h e userInput.c)	28
Server HTTP (httpServer.h e httpServer.c)	28
Rotte HTTP	29
Risposte HTTP	29
Richieste HTTP	30
Socket	30
Client e handleClient	30
Thread di Lavoro	31
Informazioni sull'Host	31

Funzione handleSigStop.....	31
Funzione handleCriticalError.....	32
Funzione httpServerServe.....	32
File httpServer.h.....	33
Definizioni di Costanti.....	33
Strutture Dati.....	33
Dichiarazioni di Funzioni.....	34
Client HTTP (httpClient.h e httpClient.c)	35
Funzione connectToHost.....	35
Funzione removeHttpHeaders.....	36
Funzione getHttpStatusCode	36
Funzione sendHttpRequest	36
Esempio di Utilizzo	36
httpClient.h.....	37
httpLib.h.....	38
Enumerazione HttpMethod	38
Enumerazione HttpStatusCode	39
Libreria HTTP (httpLib.h).....	39
Libreria CSV (csvlib.h e csvlib.c)	39
Funzioni di Lettura	39
Funzioni di Inizializzazione e Deallocazione	40
Funzioni di Stampa	40
Funzioni di Accesso ai Campi	40
Funzioni di Aggiunta	40
Csvlib.h.....	40
Funzioni di Lettura	41
Funzioni di Aggiunta	41
Funzioni di Stampa	41
Funzioni di Inizializzazione e Deallocazione	41
Funzioni di Accesso ai Campi	41
Relazioni tra i Componenti	41
Conclusione.....	42

Manuale di utilizzo

Scaricare il programma

Recarsi sul seguente link **GitHub**: <https://github.com/Lorenx03/Sistema-di-prenotazione-posti.git>

Una volta nella pagina del repository, cliccare sul pulsante **Code** e selezionare Download ZIP.

Scaricato il file **.zip**, procedere con l'estrazione dei suoi contenuti in una cartella a piacere. Utilizzare la vostra **shell** preferita e spostarsi nella directory in cui è stato estratto il file. All'interno della cartella si trova un file **Makefile**.

Compilare il programma

Per compilare ed eseguire il programma, digitare il comando **make** e premere Invio. Dopo aver eseguito il comando, Make compilerà il progetto e restituirà i file eseguibili del client e del server:

- **server**: che rappresenta la macchina server.
- **client**: che rappresenta la macchina client.

Avviare i programmi

Eeguire il **server** con il comando: `./server -p <port> -t <numThreads>`

- **<port>**: Porta su cui il server deve ascoltare.
- **<numThreads>**: Numero di thread da utilizzare per gestire le richieste.
- Esempio: `./server -p 8090 -t 10`

Per **impostazione predefinita**, il server si avvia sulla porta **8090** e utilizza **10** thread di lavoro, in questo caso è sufficiente eseguire il server con il comando: `./server`

Eeguire il **client** con il comando: `./client -a <ip> -p <port>`

- **<ip>**: Indirizzo IP del server a cui connettersi.
- **<port>**: Porta su cui il server sta ascoltando.
- Esempio: `./client -a 192.168.x.x -p 8090`

Per **impostazione predefinita**, il programma client si connette all'indirizzo IP **127.0.0.1** e alla porta **8090**, in questo caso è sufficiente eseguire il client con il comando: `./client`

Per individuare l'indirizzo IP su **macOS** esegui i seguenti comandi:

- `ipconfig getifaddr en0`
- `ipconfig getifaddr en1`

Interfaccia `en0` per Wi-Fi e interfaccia `en1` per Ethernet.

Su **Linux**, il comando `ip` è lo strumento preferito per configurare e visualizzare la rete. Per trovare l'indirizzo IP, esegui:

- `ip addr show`

Cerca l'interfaccia attiva `wlan0` per Wi-Fi e `eth0` per Ethernet.

Esecuzione simultanea di client e server

Se si desidera gestire l'esecuzione di client e server in maniera compatta e simultanea ai fini dello sviluppo è possibile utilizzare il comando `./rundev.sh` a condizione che il software `tmux` sia installato nel sistema, esso consente di visualizzare e gestire più sessioni all'interno di una singola finestra della `shell`.

Assicurarsi che tutte le dipendenze necessarie siano installate come i compilatori richiesti ad esempio `gcc` o `g++`. Se si verificano problemi durante l'esecuzione dei comandi verificare i permessi dei file eseguibili e rendere i file eseguibili utilizzando `chmod +x SERVER CLIENT rundev.sh`.

Il file `rundev.sh` è uno script `Bash` che automatizza la compilazione e l'esecuzione di un progetto di prenotazione di cinema, utilizzando `tmux` per gestire le sessioni del server e del client.

Installazione di tmux

Su Linux

Per installare `tmux` su Linux, puoi installare `tmux` utilizzando il gestore di pacchetti della tua distribuzione:

- **Ubuntu/Debian:** `sudo apt install tmux`
- **Arch Linux:** `sudo pacman -S tmux`

Su macOS

Su macOS, puoi utilizzare **Homebrew**: `brew install tmux`

Makefile

Il file `Makefile` fornito definisce un processo di build automatizzato per un progetto `C` che include un client e un server. Inizialmente, viene specificato il compilatore da utilizzare (`gcc`). Vengono definiti i flag di compilazione (`CFLAGS`), che includono avvisi aggiuntivi e ottimizzazioni, e i flag di collegamento (`LDFLAGS`), che includono la libreria `pthread` per il supporto al multithreading.

Le directory di origine (`SRCDIR`), libreria (`LIBDIR`) e oggetti (`OBJDIR`) sono definite per organizzare i file del progetto. Viene utilizzata la funzione wildcard per trovare tutti i `file.c` nella directory della libreria, e `patsubst` per generare i nomi dei file oggetto corrispondenti nella directory degli oggetti.

Le regole specifiche per la creazione degli eseguibili del `client` e del `server` collegano i file oggetto necessari utilizzando il compilatore e i flag definiti. Le regole per compilare i file sorgente (`client.c` e `server.c`) in file oggetto specificano come trasformare i file `.c` in file `.o`.

Introduzione

Il programma è stato sviluppato con un'architettura modulare, che utilizza **librerie** per organizzare e separare logicamente definizioni e implementazioni, migliorando la leggibilità e la manutenibilità del codice. L'interfaccia utente è interattiva e accessibile tramite terminale, simulando un sistema di biglietteria per un cinema.

Funzionalità del Programma

[illegible]

La grafica è migliorata tramite `escape sequences ANSI (\033[1J)` per cancellare lo schermo e rendere l'interazione più fluida.

1. Visualizzazione della Programmazione delle Sale

2. Prenotazione di Biglietti
3. Cancellazione di Prenotazioni
4. Uscita dal Sistema

1. Visualizzazione della Programmazione delle Sale

Gli utenti possono consultare la lista dei film attualmente disponibili con le relative informazioni, come titolo, durata, genere e orari di proiezione.

I dati vengono letti da un file CSV, in modo che l'operatore (anche se non è uno sviluppatore) possa configurare la programmazione tramite un semplice editor di testo.

Il file `films.csv` contiene informazioni dettagliate su una serie di film. Ogni riga del file rappresenta un film e include vari campi separati da virgole. Ecco una spiegazione dei campi presenti:

- **Titolo del film:** Il nome del film.
- **Genere:** Il genere del film, che può essere un singolo genere o una combinazione di generi.
- **Lingua:** La lingua principale in cui è girato il film.
- **Durata:** La durata del film in minuti.
- **Cast:** Gli attori principali del film, elencati tra virgolette e separati da virgole.
- **Descrizione:** Una breve descrizione della trama del film, racchiusa tra virgolette.
- **Orari di proiezione:** Gli orari in cui il film viene proiettato, separati da virgole.
- **Numero di file:** Il numero di file di posti disponibili nella sala per questo film.
- **Numero di posti per fila:** Il numero di posti disponibili per ogni fila.

2. Prenotazione di Biglietti

Il sistema permette agli utenti di prenotare uno o più biglietti (fino a un massimo di quattro per transazione). Durante la prenotazione, gli utenti possono scegliere manualmente i posti disponibili in base alla mappa della sala.

Alla conferma dell'acquisto, il sistema genera un codice univoco di prenotazione che serve per la gestione futura.

Fast and Furious - 21:00

Sala 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
A	[A1]	[A2]	[A3]	[A4]	[A5]	[A6]	[A7]	[A8]	[A9]	[A10]	[A11]	[A12]	[A13]	[A14]	[A15]	A
B	[B1]	[B2]	[B3]	[B4]	[B5]	[B6]	[B7]	[B8]	[B9]	[B10]	[B11]	[B12]	[B13]	[B14]	[B15]	B
C	[C1]	[C2]	[C3]	[C4]	[C5]	[C6]	[C7]	[C8]	[C9]	[C10]	[C11]	[C12]	[C13]	[C14]	[C15]	C
D	[D1]	[D2]	[D3]	[D4]	[D5]	[D6]	[D7]	[D8]	[D9]	[D10]	[D11]	[D12]	[D13]	[D14]	[D15]	D
E	[E1]	[E2]	[E3]	[E4]	[E5]	[E6]	[E7]	[E8]	[E9]	[E10]	[E11]	[E12]	[E13]	[E14]	[E15]	E
F	[F1]	[F2]	[F3]	[F4]	[F5]	[F6]	[F7]	[F8]	[F9]	[F10]	[F11]	[F12]	[F13]	[F14]	[F15]	F
G	[G1]	[G2]	[G3]	[G4]	[G5]	[G6]	[G7]	[G8]	[G9]	[G10]	[G11]	[G12]	[G13]	[G14]	[G15]	G
H	[H1]	[H2]	[H3]	[H4]	[H5]	[H6]	[H7]	[H8]	[H9]	[H10]	[H11]	[H12]	[H13]	[H14]	[H15]	H
I	[I1]	[I2]	[I3]	[I4]	[I5]	[I6]	[I7]	[I8]	[I9]	[I10]	[I11]	[I12]	[I13]	[I14]	[I15]	I
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

Legenda:
[A1] Disponibile
[A1] Prenotato
[A1] Disabili

Quanti posti vuoi prenotare (1-4)(0=indietro)?

3. Cancellazione di Prenotazioni

Gli utenti possono annullare una prenotazione esistente inserendo il codice di prenotazione generato in precedenza. Questa funzionalità garantisce flessibilità e controllo nella gestione delle prenotazioni.

4. Uscita dal Sistema

Una semplice opzione per chiudere il programma.

Componenti Principali

Server HTTP (server.c)

Descrizione: Il file `server.c` contiene l'implementazione del server per un sistema di prenotazione di posti in un cinema. Questo server gestisce diverse rotte HTTP per visualizzare informazioni sui film, prenotare posti, cancellare prenotazioni e altro ancora.

Funzioni Principali:

- `GETRootHandler`: Gestisce la richiesta alla radice del server, mostrando il menu principale.
- `GETFilmsHandler`: Restituisce la lista dei film disponibili.
- `GETFilmsListHandler`: Restituisce solo i nomi dei film disponibili.
- `GETBookShowtimesListHandler`: Restituisce gli orari di proiezione di un film selezionato.
- `GETFilmHallMapHandler`: Restituisce la mappa della sala per un determinato film e orario.
- `POSTBookSeat`: Gestisce la prenotazione di uno o più posti per un determinato film e orario.

Relazioni: Utilizza funzioni definite in `cinema.h`, `filmsCSVparser.h`, `utils.h` e `httpServer.h` per elaborare le richieste.

La variabile globale `cinemaFilms` viene utilizzata per memorizzare i dati dei film disponibili nel cinema.

Funzione `GETRootHandler`

La funzione `GETRootHandler` gestisce la richiesta HTTP per la radice del server (`/`). Questa funzione costruisce una risposta HTML che include un'intestazione decorativa e un menu con quattro opzioni:

- Programmazione sale
- Acquista biglietto
- Disdire una prenotazione
- Esci

Funzione `GETFilmsHandler`

La funzione `GETFilmsHandler` gestisce la richiesta HTTP per l'endpoint `/films`. Costruisce una risposta che elenca tutti i film disponibili, utilizzando la funzione `print_films` per formattare l'elenco.

Funzione GETFilmsListHandler

La funzione `GETFilmsListHandler` gestisce la richiesta HTTP per l'endpoint `/films/list`. Costruisce una risposta che elenca solo i nomi dei film disponibili, utilizzando la funzione `print_films_name`.

Funzione GETBookShowtimesListHandler

La funzione `GETBookShowtimesListHandler` gestisce la richiesta HTTP per l'endpoint `/films/showtimes`. Analizza l'ID del film dalla richiesta e, se l'ID è valido, costruisce una risposta che elenca gli orari disponibili per il film selezionato. Se l'ID del film non è valido, restituisce un messaggio di errore.

Funzione GETFilmHallMapHandler

La funzione `GETFilmHallMapHandler` gestisce la richiesta HTTP per l'endpoint `/films/map`. Analizza l'ID del film e l'indice della sala dalla richiesta e, se entrambi sono validi, costruisce una risposta che include la mappa della sala per l'orario selezionato. Se uno dei due valori non è valido, restituisce un messaggio di errore.

Funzione POSTBookSeat

La funzione `POSTBookSeat` gestisce una richiesta HTTP POST per prenotare posti in una sala cinematografica. Riceve come parametri la richiesta HTTP (`request`) e un buffer per costruire la risposta HTTP (`response`). La funzione segue un flusso ben definito per analizzare la richiesta, validare i dati, prenotare i posti e generare una risposta appropriata.

All'inizio, viene inizializzato un buffer `response_body` per costruire il corpo della risposta. Se la richiesta è vuota (`strlen(request) == 0`), la funzione risponde immediatamente con uno stato HTTP `400 Bad Request`, indicando che la richiesta non è valida. Successivamente, la funzione utilizza `strtok_r` per suddividere la stringa della richiesta in token separati da punti (.). I primi due token rappresentano l'indice del film (`selected_film`) e l'indice della sala (`hall_index`), che vengono convertiti in interi e validati. Se uno di questi valori è fuori dai limiti, viene restituita una risposta `400 Bad Request`.

Una volta validati il film e la sala, la funzione seleziona il film corrispondente dalla lista globale `cinemaFilms` e recupera l'orario dello spettacolo utilizzando `getNthToken`. Inoltre, viene generato un codice di prenotazione unico (`prenotationCode`) che sarà utilizzato per identificare la prenotazione.

La funzione procede quindi a elaborare i token successivi, che rappresentano i posti da prenotare. Ogni posto è specificato come una combinazione di riga (una lettera) e colonna (un numero). La riga viene convertita in un indice numerico sottraendo 'A', mentre la colonna viene convertita utilizzando `safeStrToInt`. Se un posto è fuori dai limiti della sala, viene restituita una risposta `400 Bad Request`. Per ogni posto valido, viene generato un codice di prenotazione unico (`bookingCode`) combinando il codice di prenotazione generale con un identificatore specifico per il posto.

Dopo aver raccolto i dettagli dei posti, la funzione tenta di prenotarli chiamando `bookSeats`. Se la prenotazione ha successo, per ogni posto viene salvata la prenotazione in un file CSV (`bookings.csv`).

utilizzando `saveBookingsToFile`. Se il salvataggio fallisce, viene restituita una risposta `500 Internal Server Error`. Inoltre, per ogni posto prenotato, viene generato un biglietto utilizzando `printTicketToBuff`, che aggiunge i dettagli del biglietto al buffer della risposta.

Infine, se tutte le operazioni hanno successo, la funzione restituisce una risposta HTTP con stato `201 Created`, indicando che la prenotazione è stata completata con successo. In caso di errore durante la prenotazione, viene restituita una risposta `500 Internal Server Error`. Questo approccio garantisce una gestione robusta delle richieste, con validazione dei dati, gestione degli errori e generazione di risposte.

Funzione `POSTUnbookSeat`

La funzione `POSTUnbookSeat` gestisce la rotta HTTP `/unbook` per cancellare una prenotazione di un posto in una sala cinematografica. La funzione accetta due parametri: `request`, che contiene il codice della prenotazione da cancellare, e `response`, che verrà utilizzato per costruire la risposta HTTP.

All'inizio, la funzione verifica se la richiesta è vuota. Se lo è, costruisce una risposta HTTP con stato `"Bad Request"` e termina l'esecuzione.

La funzione utilizza una serie di cicli annidati per iterare attraverso tutti i film, gli orari, le file e i posti delle sale cinematografiche. Durante l'iterazione, verifica se lo stato del posto è `"BOOKED"` e se il codice di prenotazione corrisponde al codice fornito nella richiesta. La corrispondenza può essere per i primi 8 caratteri (per cancellare un'intera prenotazione) o per tutti i 17 caratteri (per cancellare un singolo posto).

Se trova un posto corrispondente, tenta di acquisire un mutex per quel posto. Se riesce ad acquisire il mutex, lo stato del posto viene aggiornato a `FREE`, a meno che il posto non si trovi nell'ultima riga e in una delle prime tre colonne, nel qual caso viene impostato a `DISABLED`.

La funzione chiama `removeBookingFromFile` per rimuovere la prenotazione dal file `"bookings.csv"`.

Il codice di prenotazione del posto viene azzerato utilizzando `memset`.

Il mutex del posto viene rilasciato con `pthread_mutex_unlock`.

Dopo aver completato la ricerca, la funzione verifica se è stata trovata e annullata una prenotazione. Se sì, costruisce una risposta HTTP con stato `200 OK` e un messaggio di conferma. In caso contrario, restituisce una risposta `404 Not Found`, indicando che la prenotazione non è stata trovata.

Funzione `Main Server`

La funzione `main` del file `server.c` è il punto di ingresso del server per la gestione delle prenotazioni di una sala cinematografica. Questa funzione accetta argomenti da riga di comando per configurare il server, come la porta su cui ascoltare e il numero di thread di lavoro da utilizzare.

All'inizio, vengono impostati i valori predefiniti per la porta (`8090`) e il numero di thread (`10`).

Successivamente, un ciclo `for` analizza gli argomenti della riga di comando per sovrascrivere questi valori

predefiniti. Se viene passato l'argomento `-h`, il programma stampa un messaggio di aiuto e termina. Se vengono passati gli argomenti `-p` e `-t`, il programma aggiorna rispettivamente la porta e il numero di thread, verificando che i valori siano validi.

Viene quindi inizializzato `srand(time(NULL))` per la generazione casuale dei codici di prenotazione.

La funzione definisce diverse rotte HTTP (`HttpRoute`) per gestire le richieste del client organizzandole in una struttura ad albero. Ogni rotta è associata a un `handler` specifico per gestire le richieste `GET` o `POST`. Le rotte vengono poi aggiunte alla rotta principale (`rootRoute`) utilizzando la funzione `addHttpSubroute`.

La struttura `HttpServer` viene inizializzata con la porta, il numero di thread e la rotta principale (radice dell'albero). Vengono poi caricate le liste dei film e delle prenotazioni dai file `films.csv` e `bookings.csv`.

Infine, il server viene avviato con `httpServerServe`. Se il server non riesce ad avviarsi, viene stampato un messaggio di errore. Alla fine, viene eseguita la pulizia delle risorse liberando la lista dei film.

La funzione termina restituendo 0 in caso di successo o 1 in caso di errore.

Client HTTP (client.c)

Descrizione: Consente agli utenti di interagire con il server del cinema. Fornisce un'interfaccia utente per la selezione dei film, la prenotazione dei posti e la gestione delle prenotazioni.

Funzioni Principali:

- `printClearedResponse`: Rimuove gli header HTTP dalla risposta e la stampa.
- `countLinesOfResponse`: Conta il numero di righe nella risposta.
- `parseSeat`: Analizza un posto specificato dall'utente.
- `bookSeatPages`: Gestisce il processo di prenotazione dei posti.

Relazioni: Utilizza funzioni definite in `httpClient.h`, `userInput.h`, `utils.h` e `cinemaMap.h`.

Funzione `printClearedResponse`

La funzione `printClearedResponse` rimuove gli header HTTP dalla risposta e stampa il contenuto rimanente, utilizzando una sequenza di escape per cancellare lo schermo prima di stampare la risposta.

Funzione `countLinesOfResponse`

La funzione `countLinesOfResponse` conta il numero di righe in una risposta HTTP, iterando attraverso ogni carattere della risposta e incrementando un contatore ogni volta che trova un carattere di nuova linea (`\n`).

Funzione `parseSeat`

La funzione `parseSeat` analizza una stringa che rappresenta un posto a sedere, estraendo la riga e la colonna. Se uno dei parametri è `NULL` o se il formato del posto è invalido, la funzione stampa un messaggio di errore e imposta valori di default.

Funzione bookSeatPages

La funzione `bookSeatPages` gestisce il processo di prenotazione di un posto. Inizia inviando una richiesta HTTP per ottenere gli orari disponibili per un film specifico e stampa la risposta. L'utente viene quindi invitato a scegliere un orario. Se la scelta è valida, il programma invia un'altra richiesta HTTP per ottenere la mappa della sala per l'orario scelto. La mappa viene analizzata e stampata, e l'utente può scegliere quanti posti prenotare e specificare la riga e la colonna di ciascun posto.

Durante la selezione dei posti, il programma verifica se il posto è già prenotato o selezionato e aggiorna la mappa di conseguenza. Una volta che tutti i posti sono stati selezionati, l'utente può confermare la prenotazione. Se confermata, il programma invia una richiesta HTTP per prenotare i posti e stampa la risposta del server. L'utente ha quindi la possibilità di salvare la prenotazione su un file.

Funzione unBookSeatPage

La funzione `unBookSeatPage` gestisce la pagina di cancellazione delle prenotazioni per un client. La funzione accetta un puntatore a una struttura `TargetHost`, che rappresenta il server di destinazione per le richieste HTTP.

All'inizio, vengono dichiarati tre variabili: `response`, un array di caratteri per memorizzare la risposta del server; `currentPage`, un intero che tiene traccia della pagina corrente del menu; e `seatCode`, un array di caratteri per memorizzare il codice del posto.

La funzione utilizza un ciclo `do-while` per gestire le diverse pagine del menu. All'interno del ciclo, uno `switch` controlla il valore di `currentPage` per determinare quale pagina visualizzare.

- **Caso -1:** Visualizza il menu principale per la cancellazione delle prenotazioni, con opzioni per tornare indietro, cancellare un singolo posto o cancellare un'intera prenotazione. L'utente inserisce la sua scelta, che viene letta e memorizzata in `currentPage`.
- **Caso 1:** Gestisce la cancellazione di un singolo posto. L'utente inserisce il codice del sedile. Se il codice è valido, viene inviata una richiesta HTTP al server per cancellare la prenotazione. Se la richiesta ha successo, viene visualizzata la risposta del server; altrimenti, viene mostrato un messaggio di errore.
- **Caso 2:** Gestisce la cancellazione di un'intera prenotazione. L'utente inserisce il codice della prenotazione. Se il codice è valido, viene inviata una richiesta HTTP al server per cancellare la prenotazione. Se la richiesta ha successo, viene visualizzata la risposta del server; altrimenti, viene mostrato un messaggio di errore.
- **Default:** Gestisce le scelte non valide, mostrando un messaggio di errore e riportando l'utente al menu principale.

Il ciclo continua fino a quando `currentPage` non è uguale a 0, che indica che l'utente ha scelto di tornare al menu principale.

Funzione Main Client

La funzione `main` nel file `client.c` è il punto di ingresso per il client del sistema di prenotazione dei posti. Questa funzione accetta argomenti da riga di comando per configurare l'indirizzo IP del server e la porta su cui connettersi.

All'inizio, vengono impostati i valori predefiniti per l'indirizzo IP (`127.0.0.1`) e la porta (`8090`). Successivamente, un ciclo `for` analizza gli argomenti della riga di comando per sovrascrivere questi valori predefiniti. Se viene passato l'argomento `-h`, il programma stampa un messaggio di aiuto e termina. Se vengono passati gli argomenti `-p` e `-a`, il programma aggiorna rispettivamente la porta e l'indirizzo IP, verificando che i valori siano validi.

La variabile `currentPage` viene inizializzata a 0, indicando che il client inizia dal menu principale. Viene anche dichiarato un buffer `responseBuffer` per memorizzare le risposte del server e una struttura `TargetHost` per memorizzare l'indirizzo IP e la porta del server.

Il client entra in un ciclo `do-while` che gestisce le diverse pagine del menu. All'interno del ciclo, uno `switch` controlla il valore di `currentPage` per determinare quale pagina visualizzare:

- **MAIN_MENU**: Invia una richiesta `GET` alla radice del server e visualizza la risposta. L'utente inserisce una scelta, che viene letta e memorizzata in `choice`. Se la scelta è valida, `currentPage` viene aggiornato di conseguenza.
- **FILMS_LIST**: Invia una richiesta `GET` alla rotta `/films` del server e visualizza la lista dei film. L'utente preme invio per tornare al menu principale.
- **BOOK_SEAT**: Invia una richiesta `GET` alla rotta `/films/list` del server e visualizza la lista dei film disponibili per la prenotazione. L'utente inserisce il numero del film per cui vuole acquistare il biglietto. Se la scelta è valida, viene chiamata la funzione `bookSeatPages` per gestire la prenotazione.
- **CANCEL_BOOKING**: Chiama la funzione `unBookSeatPage` per gestire la cancellazione di una prenotazione. Dopo la cancellazione, torna al menu principale.
- **EXIT**: Esce dal ciclo e termina il programma.

Il ciclo continua fino a quando `currentPage` non è uguale a 4, che indica che l'utente ha scelto di uscire. La funzione termina restituendo 0 in caso di successo o 1 in caso di errore.

Gestione Cinema (`cinema.h` e `cinema.c`)

Descrizione: Definisce le strutture dati per rappresentare i film, le sale e i posti. Fornisce funzioni per inizializzare le sale, prenotare posti, annullare prenotazioni e generare mappe delle sale.

Strutture Dati:

- **SeatState**: Enum per rappresentare lo stato di un posto (libero, prenotato, disabilitato, selezionato).
- **Seat**: Struttura per rappresentare un posto.

- `Hall`: Struttura per rappresentare una sala.
- `Film`: Struttura per rappresentare un film.
- `Films`: Struttura per rappresentare una lista di film.

Funzioni Principali:

- `initFilmsList`: Inizializza la lista dei film.
- `initializeSeats`: Inizializza i posti di una sala.
- `createHallsForShowtimes`: Crea le sale per gli orari di proiezione di un film.
- `generateHallMapResponse`: Genera la risposta della mappa della sala.
- `printTicket`: Stampa un biglietto.
- `saveBookingsToFile`: Salva le prenotazioni su un file.
- `loadBookingsFromFile`: Carica le prenotazioni da un file.
- `removeBookingFromFile`: Rimuove le prenotazioni da un file.

Relazioni: Utilizza funzioni definite in `filmsCSVparser.h` e `utils.h`.

File `cinema.c`

Il file `cinema.c` contiene diverse funzioni per la gestione di un sistema di prenotazione dei posti in un cinema.

Funzione `initializeSeats`

La funzione `initializeSeats` è progettata per inizializzare i posti di una sala cinematografica rappresentata dalla struttura `Hall`. Prende tre parametri: un puntatore a una struttura `Hall`, il numero di righe e il numero di colonne della sala.

All'interno della funzione, vengono utilizzati due cicli annidati per iterare su ogni posto della sala. Il ciclo esterno scorre attraverso le righe, mentre il ciclo interno scorre attraverso le colonne. Per ogni posto, vengono eseguite le seguenti operazioni:

- Il campo `row` del posto viene impostato su una lettera che rappresenta la riga, partendo da 'A' e incrementando per ogni riga successiva.
- Il campo `seat_number` viene impostato sul numero del posto all'interno della riga, partendo da 1.
- Il campo `state` viene impostato su `FREE` (libero) o `DISABLED` (riservato a persone con disabilità).
- Viene inizializzato un mutex per ogni posto utilizzando `pthread_mutex_init`.
- Il campo `booking_code` viene azzerato utilizzando `memset` per garantire che non ci siano codici di prenotazione residui.

Questa funzione garantisce che tutti i posti nella sala siano correttamente inizializzati con i valori appropriati per riga, numero del posto, stato, mutex e codice di prenotazione.

Funzione createHallsForShowtimes

La funzione `createHallsForShowtimes` è progettata per creare e inizializzare le sale cinematografiche per ogni spettacolo di un film. Prende un puntatore a una struttura `Film` come parametro.

All'inizio, la funzione alloca memoria per un array di strutture `Hall`, una per ogni spettacolo del film, utilizzando `malloc`. Se l'allocazione fallisce, stampa un messaggio di errore e termina il programma.

Successivamente, la funzione itera su ogni spettacolo del film utilizzando un ciclo `for`. Per ogni spettacolo, esegue le seguenti operazioni:

- Imposta il numero di righe e colonne della sala cinematografica utilizzando i valori presenti nella struttura `Film`.
- Alloca memoria per un array di puntatori a strutture `Seat`, uno per ogni riga della sala. Se l'allocazione fallisce, stampa un messaggio di errore e termina il programma.
- Utilizza un secondo ciclo `for` per iterare su ogni riga della sala. Per ogni riga, alloca memoria per un array di strutture `Seat`, una per ogni posto nella riga. Se l'allocazione fallisce, stampa un messaggio di errore e termina il programma.
- Chiama la funzione `initializeSeats` per inizializzare i posti della sala con i valori appropriati.

Questa funzione garantisce che tutte le sale cinematografiche per gli spettacoli di un film siano correttamente create e inizializzate, con memoria allocata per ogni posto e con i campi dei posti impostati correttamente.

Funzione initFilmsList

La funzione `initFilmsList` è progettata per inizializzare una lista di film leggendo i dati da un file CSV e configurando le sale cinematografiche per ogni film. Prende due parametri: il nome del file CSV (`filename`) e un puntatore a una struttura `Films` (`filmsStruct`).

All'inizio, la funzione chiama `readFilmsCsv` per leggere i dati dal file CSV specificato e memorizzare i film nella lista all'interno della struttura `Films`. Questa funzione aggiorna anche il conteggio dei film letti.

Dopo aver letto i dati, la funzione itera su ogni film nella lista utilizzando un ciclo `for`. Per ogni film, esegue le seguenti operazioni:

- Chiama `countShowtimes` per determinare il numero di spettacoli per il film corrente e aggiorna il campo `numbersShowtimes` della struttura `Film`.
- Chiama `createHallsForShowtimes` per creare e inizializzare le sale cinematografiche per gli spettacoli del film corrente.

Questa funzione garantisce che tutti i film siano correttamente letti dal file CSV e che le sale cinematografiche per ogni spettacolo siano create e inizializzate, preparando così i dati per ulteriori elaborazioni o visualizzazioni.

Funzione generateHallMapResponse

La funzione `generateHallMapResponse` è progettata per generare una rappresentazione testuale della mappa dei posti di una sala cinematografica e memorizzarla in un buffer. Prende tre parametri: un puntatore a una struttura `Hall` (`hall`), un puntatore a un buffer di caratteri (`buffer`) e la dimensione rimanente del buffer (`remaining_size`).

All'inizio, la funzione chiama `appendToBuffer` per aggiungere al buffer le dimensioni della sala (numero di righe e colonne) nel formato `"%d.%d."`. Questa funzione aggiorna il puntatore del buffer e la dimensione rimanente.

Successivamente, la funzione utilizza due cicli annidati per iterare su ogni posto della sala. Il ciclo esterno scorre attraverso le righe, mentre il ciclo interno scorre attraverso le colonne. Per ogni posto, se c'è ancora spazio nel buffer, la funzione converte lo stato del posto (`state`) in un carattere ('0' per `FREE`, '1' per `DISABLED`) e lo aggiunge al buffer. Dopo aver aggiunto il carattere, la funzione aggiorna il puntatore del buffer e decrementa la dimensione rimanente.

Infine, la funzione aggiunge il carattere di terminazione della stringa ('\0') alla fine del buffer per assicurarsi che la stringa generata sia correttamente terminata.

Questa funzione garantisce che la mappa dei posti della sala sia correttamente convertita in una stringa e memorizzata nel buffer, pronta per essere utilizzata per ulteriori elaborazioni o inviata come risposta.

Funzione bookSeats

La funzione `bookSeats` è progettata per prenotare fino a un massimo di 4 posti in una sala cinematografica. Accetta come parametri un puntatore alla sala (`Hall *hall`), il numero di posti da prenotare (`numSeats`), una matrice di coordinate dei posti (`seats[numSeats][2]`) e una matrice di codici di prenotazione (`bookingCodes[numSeats][18]`). La funzione garantisce la sicurezza nei contesti multi-thread utilizzando mutex per sincronizzare l'accesso ai posti.

Validazione dei parametri

La funzione inizia verificando che i parametri forniti siano validi. Controlla che la sala, i posti e i codici di prenotazione non siano nulli, e che il numero di posti sia compreso tra 1 e 4. Se uno di questi controlli fallisce, viene stampato un messaggio di errore su `stderr` e la funzione restituisce 1, indicando un fallimento.

Blocco dei posti

Per ogni posto da prenotare, la funzione tenta di acquisire il mutex associato al posto utilizzando `pthread_mutex_trylock`. Questo approccio evita il blocco del thread nel caso in cui un altro thread stia già prenotando lo stesso posto. Se il mutex non può essere acquisito o se il posto è già prenotato (`BOOKED`), la funzione rilascia i mutex acquisiti fino a quel momento e restituisce un errore. Questo garantisce che i posti siano prenotati in modo atomico.

Prenotazione dei posti

Dopo aver bloccato con successo tutti i posti, la funzione verifica che le coordinate di ciascun posto siano valide, ovvero che rientrino nei limiti della sala. Se una coordinata è invalida, la funzione ripristina lo stato originale dei posti già prenotati, rilascia i mutex e restituisce un errore. Questo evita modifiche parziali in caso di errore.

Se le coordinate sono valide, la funzione aggiorna lo stato del posto a **BOOKED**, assegna il codice di prenotazione al posto e rilascia il mutex. Per garantire la consistenza, lo stato originale dei posti viene salvato in un array temporaneo (**temp**) prima di essere modificato.

Ritorno del risultato

Se tutti i posti vengono prenotati con successo, la funzione restituisce 0, indicando che l'operazione è stata completata correttamente. Questo approccio garantisce che la prenotazione sia sicura, atomica e coerente, anche in presenza di accessi concorrenti.

Funzione `saveBookingsToFile`

Questa funzione è utilizzata per registrare i dettagli di una prenotazione (film, orario, posto e codice di prenotazione) in un file, garantendo che l'operazione sia sicura in ambienti multi-thread. È utile per mantenere un registro persistente delle prenotazioni effettuate.

Accetta diversi parametri: **filmIndex** e **showtimeIndex** per identificare il film e l'orario dello spettacolo, **row** e **col** per specificare il posto a sedere, **bookingCode** come codice di prenotazione, e **filename** come nome del file in cui salvare i dati.

La funzione inizia con una verifica dei parametri. Se **filmIndex** e **showtimeIndex** sono validi (≥ 0), ma **bookingCode** o **filename** sono nulli, viene stampato un messaggio di errore su **stderr** e la funzione restituisce 1, indicando un fallimento. Successivamente, controlla che il posto a sedere sia valido: la riga (**row**) deve essere una lettera tra 'A' e 'Z', e la colonna (**col**) deve essere un numero tra 1 e 99. Se questi valori non rientrano nei limiti, viene stampato un altro messaggio di errore e la funzione termina con un valore di ritorno pari a 1.

Se i parametri sono validi, la funzione tenta di aprire il file specificato in modalità di aggiunta ("a"). Se l'apertura fallisce, viene stampato un messaggio di errore tramite **perror**, che include una descrizione dell'errore fornita dal sistema, e la funzione restituisce 1. Una volta aperto il file, la funzione utilizza un mutex (**pthread_mutex_lock**) per garantire che l'accesso al file sia **thread-safe**, evitando conflitti in ambienti multi-thread. I dettagli della prenotazione vengono quindi scritti nel file in un formato specifico: **filmIndex.showtimeIndex.rowcol.bookingCode**. Dopo aver scritto i dati, il file viene svuotato con **fflush** per assicurarsi che i dati siano effettivamente salvati.

Infine, il mutex viene sbloccato con **pthread_mutex_unlock**, il file viene chiuso con **fclose**, e la funzione restituisce 0 per indicare che l'operazione è stata completata con successo. Questo approccio garantisce che i dati siano salvati in modo sicuro e che eventuali errori vengano gestiti in modo appropriato.

Funzione removeBookingFromFile

La funzione `removeBookingFromFile` è progettata per rimuovere una prenotazione specifica da un file, identificata tramite un codice di prenotazione (`bookingCode`). Accetta due parametri: il codice di prenotazione (`bookingCode`) e il nome del file (`filename`) da cui rimuovere la prenotazione. La funzione utilizza un approccio basato su file per cercare e modificare il contenuto del file in modo sicuro.

All'inizio, la funzione verifica che i parametri forniti non siano nulli. Se uno dei due è nullo, viene stampato un messaggio di errore su `stderr` e la funzione restituisce 1, indicando un fallimento.

Successivamente, il file specificato viene aperto in modalità lettura e scrittura ("`r+`"). Se l'apertura fallisce, viene stampato un messaggio di errore tramite `perror`, e la funzione termina con un valore di ritorno pari a 1.

La funzione utilizza un buffer line per leggere il file riga per riga utilizzando `fgets`. Prima di accedere al file, viene acquisito un mutex (`pthread_mutex_lock`) per garantire che l'operazione sia `thread-safe`, evitando conflitti in ambienti multi-thread. Durante la lettura, ogni riga viene confrontata con il codice di prenotazione utilizzando `strstr`. Se il codice di prenotazione viene trovato all'interno della riga, la funzione utilizza `fseek` per spostare il puntatore del file all'inizio della riga corrispondente.

Successivamente, chiama la funzione `fdeleteBytes` per eliminare i byte corrispondenti alla lunghezza della riga dal file. Dopo l'eliminazione, il file viene svuotato con `fflush` per assicurarsi che le modifiche siano salvate, e il ciclo si interrompe.

Dopo aver completato l'operazione, il mutex viene rilasciato con `pthread_mutex_unlock`, e il file viene chiuso con `fclose`. La funzione restituisce 0 per indicare che l'operazione è stata completata con successo. Questo approccio garantisce che la rimozione della prenotazione sia sicura e che il file rimanga coerente, anche in presenza di accessi concorrenti.

Funzione loadBookingsFromFile

La funzione `loadBookingsFromFile` è progettata per caricare le prenotazioni dei posti da un file e aggiornare la struttura `Films` con queste informazioni. Prende due parametri: un puntatore a una struttura `Films` (`filmsStruct`) e una stringa che rappresenta il nome del file (`filename`).

All'inizio, la funzione apre il file specificato in modalità lettura ("`r`") utilizzando `fopen`. Se il file non può essere aperto, stampa un messaggio di errore utilizzando `perror` e restituisce -1.

La funzione dichiara variabili per memorizzare le righe lette dal file (`line`), i token estratti dalla riga (`token` e `saveptr`), gli identificatori del film e dello spettacolo (`film_id` e `showtime_id`), e le stringhe per il posto (`seat`) e il codice di prenotazione (`bookingCode`).

La funzione utilizza un ciclo `while` per leggere ogni riga del file utilizzando `fgets`. Per ogni riga letta, esegue le seguenti operazioni:

- Utilizza `strtok_r` per estrarre il primo token dalla riga, che rappresenta l'identificatore del film, e lo converte in un intero utilizzando `safeStrToInt`.
- Estrae il secondo token, che rappresenta l'identificatore dello spettacolo, e lo converte in un intero.
- Estrae il terzo token, che rappresenta il posto, e lo copia nella variabile `seat`.
- Estrae il quarto token, che rappresenta il codice di prenotazione, e lo copia nella variabile `bookingCode`.
- Chiama la funzione `bookSeat` per prenotare il posto specificato nella sala corrispondente utilizzando le informazioni estratte.
- Dopo aver elaborato tutte le righe del file, la funzione chiude il file utilizzando `fclose` e restituisce 0 per indicare che l'operazione di caricamento delle prenotazioni è stata completata con successo.

Funzione `printTicket`

La funzione `printTicket` è responsabile della generazione di un biglietto di prenotazione per un film. Riceve diversi parametri: un puntatore al buffer `buff` dove verrà scritto il biglietto, il `bookingCode` che rappresenta il codice di prenotazione, il `filmTitle` che è il titolo del film, il `filmShowtime` che indica l'orario dello spettacolo, il `seat` che rappresenta il posto prenotato, e un puntatore a `remaining_size` che tiene traccia dello spazio rimanente nel buffer.

All'interno della funzione, vengono effettuate chiamate successive alla funzione `appendToBuffer` per aggiungere diverse linee di testo al buffer. Queste linee includono un'intestazione del biglietto, il codice di prenotazione, il titolo del film, l'orario dello spettacolo e il posto prenotato. Ogni chiamata a `appendToBuffer` aggiorna il puntatore del buffer e riduce la dimensione rimanente, assicurandosi che il testo venga aggiunto correttamente senza superare i limiti del buffer.

Il risultato finale è una stringa formattata che rappresenta un biglietto di prenotazione, pronta per essere inviata al client o salvata per riferimento futuro. La funzione garantisce che tutte le informazioni rilevanti siano incluse nel biglietto in un formato leggibile e strutturato.

Il ruolo del prefisso e del suffisso nel biglietto

Nella gestione dei biglietti per eventi è importante poter distinguere un'intera prenotazione da ogni singolo biglietto al suo interno. Per questo si utilizza un sistema di codici di prenotazione composto da due parti: il **prefisso** e il **suffisso**.

Il prefisso identifica un gruppo di biglietti legati alla stessa prenotazione, permettendo di gestire l'intera prenotazione come un unico blocco.

Il suffisso, invece, è unico per ogni biglietto e serve a distinguere i singoli biglietti all'interno dello stesso gruppo, consentendo operazioni mirate su ciascun biglietto.

Questo sistema rende la gestione delle prenotazioni più organizzata, tracciabile e flessibile.

Esempio di un biglietto

Il sistema di gestione genera biglietti in formato testuale strutturato, riportando informazioni essenziali come il codice di prenotazione, il film, l'orario e il posto assegnato. Ogni biglietto include un codice univoco che si compone di un prefisso e un suffisso. Ecco un esempio di due biglietti appartenenti alla stessa prenotazione:

===== BIGLIETTO =====

Codice prenotazione: PB9GL2NB-8TKVRSJ5

Film: Fast and Furious

Orario: 23:30

Posto: A10

=====

===== BIGLIETTO =====

Codice prenotazione: PB9GL2NB-OHMB3P8A

Film: Fast and Furious

Orario: 23:30

Posto: C7

=====

Vuoi salvare la prenotazione? (s/n): s

Prenotazione salvata con nome: PB9GL2NB.txt

In questo caso, entrambi i biglietti condividono il prefisso **PB9GL2NB**, che identifica l'intera prenotazione, mentre il suffisso (rispettivamente **8TKVRSJ5** e **OHMB3P8A**) distingue i singoli biglietti.

Dopo aver generato i biglietti, il sistema consente di salvare la prenotazione in un file.

Il file viene nominato utilizzando il prefisso del codice di prenotazione, seguito dall'estensione **.txt**.

Ad esempio, per i biglietti riportati sopra, la prenotazione viene salvata come **PB9GL2NB.txt**, contenente l'intera lista dei biglietti associati al prefisso.

La funzione `generateRandomString`

La funzione `generateRandomString` è utilizzata per generare una stringa casuale di una lunghezza specificata. Riceve due parametri: un puntatore a `char str`, dove verrà memorizzata la stringa generata, e una variabile `length` che indica la lunghezza desiderata della stringa.

All'interno della funzione, viene definito un array `charset` che contiene i caratteri possibili che possono essere inclusi nella stringa casuale. Questo array include lettere maiuscole dell'alfabeto inglese e cifre numeriche. La dimensione del set di caratteri viene calcolata sottraendo uno dalla dimensione dell'array, poiché l'array include anche il carattere di terminazione nullo `\0`.

La funzione utilizza un ciclo `for` per iterare attraverso ogni posizione della stringa da generare. In ogni iterazione, viene selezionato un carattere casuale dal `charset` utilizzando la funzione `rand()` per generare un indice casuale. Questo indice viene utilizzato per accedere a un carattere nel `charset`, che viene poi assegnato alla posizione corrente nella stringa `str`.

Infine, dopo aver riempito tutte le posizioni della stringa con caratteri casuali, viene aggiunto un carattere di terminazione nullo `\0` alla fine della stringa per assicurarsi che sia correttamente terminata. Questo rende la stringa utilizzabile come una stringa C standard.

File `cinema.h`

Il file `cinema.h` è un file di intestazione per un sistema di prenotazione di posti in una sala cinematografica. Inizia con le direttive di precompilazione `#ifndef`, `#define` e `#endif` per evitare inclusioni multiple del file.

Include diverse librerie standard di C, come `stdio.h`, `stdlib.h`, `string.h`, `stdbool.h`, `stdarg.h` e `pthread.h`, necessarie per le operazioni di input/output, gestione della memoria, manipolazione delle stringhe, utilizzo di tipi booleani, gestione degli argomenti variabili e threading.

Definisce un'enumerazione `SeatState` che rappresenta lo stato di un posto, con i possibili valori `FREE` (libero), `BOOKED` (prenotato), `DISABLED` (riservato a persone con disabilità) e `SELECTED` (selezionato).

La struttura `Seat` rappresenta un singolo posto in una sala cinematografica. Include campi per la riga (`row`), il numero del posto (`seat_number`), lo stato del posto (`state`), un mutex per la sincronizzazione (`lock`) e un codice di prenotazione (`booking_code`).

La struttura `Hall` rappresenta una sala cinematografica. Include campi per il numero di righe (`rows`), il numero di colonne (`columns`) e un array bidimensionale di posti (`seats`).

La struttura `Film` rappresenta un film. Include campi per il nome (`name`), il genere (`genre`), la lingua (`language`), la durata (`duration`), gli attori (`actors`), la trama (`plot`), gli orari degli spettacoli (`showtimes`), il numero di righe e colonne nella sala (`rows` e `columns`), il numero di spettacoli (`numbers_showtimes`) e un array di sale (`halls`).

La struttura `Films` rappresenta una lista di film. Include un array di film (`list`) e un conteggio dei film (`count`).

La funzione `generateHallMapResponse` genera una rappresentazione della mappa della sala in formato stringa, dove i numeri rappresentano lo stato dei posti. Per quanto riguarda le prenotazioni, la funzione `bookSeats` consente di prenotare fino a quattro posti in una sala, mentre `printTicketToBuff` stampa i dettagli di un biglietto in un buffer.

La gestione dei file è supportata da funzioni come `saveBookingsToFile`, che salva le prenotazioni in un file CSV, `removeBookingFromFile`, che rimuove una prenotazione dato il codice di prenotazione, e `loadBookingsFromFile`, che carica le prenotazioni da un file CSV e aggiorna la struttura dei film. La funzione `freeFilmsList` si occupa della pulizia, liberando la memoria allocata per la lista dei film.

Generazione Mappa Cinema (`cinemaMap.h` e `cinemaMap.c`)

Descrizione: Fornisce funzioni per generare e disegnare la mappa della sala del cinema. Utilizza le informazioni sui posti per creare una rappresentazione visiva della sala.

Funzioni Principali:

- `centerMapText`: Centra un testo nella mappa.
- `drawSeatNumbers`: Disegna i numeri dei posti.
- `drawSeparatorLine`: Disegna una linea di separazione.
- `generateHallMap`: Genera la mappa della sala.

Relazioni: Utilizza funzioni definite in `utils.h` e `cinema.h`.

`cinemaMap.c`

Il file `cinemaMap.c` contiene funzioni per generare e visualizzare la mappa dei posti di una sala cinematografica. Include i file di intestazione `cinemaMap.h`, `utils.h` e `cinema.h`.

Funzione `centerMapText`

La funzione `centerMapText` prende il numero di colonne e una stringa formattata come parametri. Utilizza `va_list` per gestire un numero variabile di argomenti, formatta la stringa e la centra rispetto al numero di colonne specificato, aggiungendo spazi di padding prima di stamparla.

Funzione `drawSeatNumbers`

La funzione `drawSeatNumbers` aggiunge i numeri dei posti alla mappa. Prende un puntatore a un buffer, la dimensione rimanente del buffer e il numero di colonne. Aggiunge i numeri dei posti al buffer, con un formato diverso a seconda che il numero del posto sia inferiore o superiore a 9.

Funzione `drawSeparatorLine`

La funzione `drawSeparatorLine` disegna una linea di separazione tra le righe dei posti. Prende un puntatore a un buffer, la dimensione rimanente del buffer e il numero di colonne. Aggiunge una linea di separazione al buffer, con un formato diverso a seconda che il numero del posto sia inferiore o superiore a 9.

Funzione `generateHallMap`

La funzione `generateHallMap` genera la mappa completa della sala cinematografica. Prende una mappa dei posti, un buffer, la dimensione rimanente del buffer, il numero di righe e il numero di colonne. Chiama `drawSeatNumbers` e `drawSeparatorLine` per aggiungere i numeri dei posti e le linee di separazione. Itera su ogni posto della sala, aggiungendo al buffer una rappresentazione colorata del posto a seconda del suo stato (`FREE`, `BOOKED`, `DISABLED`, `SELECTED`). Aggiunge anche una linea vuota dopo un terzo delle righe per separare visivamente la mappa. Infine, chiama nuovamente `drawSeparatorLine` e `drawSeatNumbers` per completare la mappa.

Queste funzioni lavorano insieme per creare una rappresentazione visiva della disposizione dei posti in una sala cinematografica, utilizzando colori per indicare lo stato di ciascun posto e centrando il testo per una migliore leggibilità.

`cinemaMap.h`

Il file `cinemaMap.h` è un file di intestazione per il modulo che gestisce la generazione e la visualizzazione della mappa dei posti di una sala cinematografica. Inizia con le direttive di precompilazione `#ifndef`, `#define` e `#endif` per evitare inclusioni multiple del file.

Include diverse librerie standard di C, come `stdio.h`, `stdlib.h`, `string.h`, `stdbool.h` e `stdarg.h`, necessarie per le operazioni di input/output, gestione della memoria, manipolazione delle stringhe, utilizzo di tipi booleani e gestione degli argomenti variabili.

Dichiara quattro funzioni che saranno implementate nel file sorgente corrispondente (`cinemaMap.c`):

- `centerMapText`: Questa funzione prende il numero di colonne e una stringa formattata come parametri. Utilizza un numero variabile di argomenti per formattare la stringa e la centra rispetto al numero di colonne specificato.
- `drawSeatNumbers`: Questa funzione prende un puntatore a un buffer, la dimensione rimanente del buffer e il numero di colonne. Aggiunge i numeri dei posti al buffer, formattandoli in modo appropriato.
- `drawSeparatorLine`: Questa funzione prende un puntatore a un buffer, la dimensione rimanente del buffer e il numero di colonne. Aggiunge una linea di separazione al buffer per distinguere visivamente le righe dei posti.
- `generateHallMap`: Questa funzione prende una mappa dei posti, un buffer, la dimensione rimanente del buffer, il numero di righe e il numero di colonne. Genera la mappa completa della sala cinematografica, utilizzando le altre funzioni dichiarate per aggiungere numeri dei posti, linee di separazione e rappresentazioni colorate dei posti a seconda del loro stato.

Queste funzioni lavorano insieme per creare una rappresentazione visiva della disposizione dei posti in una sala cinematografica, migliorando la leggibilità e la comprensione della mappa.

Parser CSV (filmsCSVparser.h e filmsCSVparser.c)

Descrizione: Legge i dati dei film da un file CSV e li memorizza nelle strutture dati appropriate. Fornisce funzioni per stampare i dettagli dei film e contare gli orari di proiezione.

Funzioni Principali:

- `trimWhitespace`: Rimuove gli spazi bianchi all'inizio e alla fine di una stringa.
- `parseFilmsCsvLine`: Analizza una riga del CSV e la memorizza in una struttura `Film`.
- `calculateTotalSeats`: Calcola il numero totale di posti in una sala.
- `countShowtimes`: Conta il numero di orari di proiezione di un film.
- `readFilmsCsv`: Legge il file CSV e memorizza i dati nei film.
- `freeFilms`: Libera la memoria allocata per i film.
- `printFilms`: Stampa i dettagli dei film.
- `printFilmsName`: Stampa i nomi dei film.
- `countCsvLines`: Conta il numero di righe nel file CSV.

Relazioni: Utilizza funzioni definite in `cinema.h`.

File filmsCSVparser.c

Il file `filmsCSVparser.c` contiene funzioni per leggere, analizzare e gestire i dati dei film da un file CSV.

Include i file di intestazione `cinema.h` e `filmsCSVparser.h`.

Funzioni di Utilità

`trimWhitespace`: Questa funzione rimuove gli spazi bianchi iniziali e finali da una stringa. Utilizza un ciclo per avanzare il puntatore della stringa fino al primo carattere non spazio e un altro ciclo per trovare l'ultimo carattere non spazio. Infine, aggiunge un terminatore nullo alla fine della stringa.

Funzioni di Parsing

`parseFilmsCsvLine`: Questa funzione analizza una riga del file CSV e popola una struttura `Film`. Gestisce i campi tra virgolette e le virgole all'interno dei campi. Utilizza un buffer per memorizzare i caratteri del campo corrente e un flag `inside_quotes` per tenere traccia se il parser si trova all'interno di un campo tra virgolette. Assegna i campi analizzati ai campi corrispondenti della struttura `Film`.

Funzioni di Calcolo

`calculateTotalSeats`: Questa funzione calcola il numero totale di posti in una sala cinematografica, moltiplicando il numero di righe per il numero di colonne.

`countShowtimes`: Questa funzione conta il numero di spettacoli di un film, dividendo la stringa degli orari degli spettacoli e contando i token separati da virgole.

Funzioni di Lettura e Scrittura

readFilmsCsv: Questa funzione legge un file CSV e memorizza i dati dei film in un array dinamico di strutture `Film`. Apre il file, legge ogni riga, raddoppia la capacità dell'array se necessario e utilizza `parseFilmsCsvLine` per analizzare ogni riga. Alla fine, chiude il file e aggiorna il conteggio dei film.

free_films: Questa funzione libera la memoria allocata per un array di strutture `Film`. Libera la memoria per ogni campo della struttura `Film` e poi libera l'array stesso.

Funzioni di Stampa

printFilms: Questa funzione stampa i dettagli dei film in un buffer per scopi di test. Utilizza `snprintf` per formattare i dettagli di ogni film e aggiungerli al buffer.

printFilmsName: Questa funzione stampa solo i nomi dei film in un buffer. Utilizza `snprintf` per formattare i nomi dei film e aggiungerli al buffer.

Funzioni di Conto

countCsvLines: Questa funzione conta il numero di righe in un file CSV. Apre il file, legge ogni riga e incrementa un contatore. Alla fine, chiude il file e restituisce il conteggio delle righe.

Queste funzioni lavorano insieme per fornire un'implementazione completa per la gestione dei dati dei film da un file CSV, inclusa la lettura, l'analisi, il calcolo, la stampa e la gestione della memoria.

Utilità (utils.h e utils.c)

Descrizione: Contiene funzioni di utilità generali utilizzate in tutto il sistema, come la conversione di stringhe in interi, la generazione di stringhe casuali e la gestione dei buffer.

Funzioni Principali:

- **safeStrToInt:** Converte una stringa in un intero in modo sicuro.
- **getLine:** Ottiene una linea da una stringa.
- **appendToBuffer:** Aggiunge una stringa a un buffer.
- **generateRandomString:** Genera una stringa casuale.
- **getNthToken:** Ottiene il n-esimo token da una stringa.
- **convertToUppercase:** Converte una stringa in maiuscolo.
- **fdeleteBytes:** Rimuove un certo numero specifico di byte in un file.

Funzione fdeleteBytes

La funzione `fdeleteBytes` è progettata per eliminare un numero specifico di byte da un file aperto, spostando i dati rimanenti per riempire lo spazio vuoto e ridimensionando il file di conseguenza. Accetta due parametri: un puntatore al file (`FILE* fp`) e il numero di byte da eliminare (`int bytes`). La funzione utilizza una combinazione di operazioni di lettura, scrittura e manipolazione della posizione del file per ottenere il risultato desiderato.

All'inizio, la funzione calcola due posizioni chiave nel file: `readPos`, che rappresenta la posizione da cui iniziare a leggere i dati dopo i byte da eliminare, e `writePos`, che rappresenta la posizione in cui i dati letti verranno riscritti. La posizione iniziale del file viene salvata in `startingPos` per ripristinarla alla fine dell'operazione.

La funzione utilizza un ciclo `while` per leggere i byte dal file, uno alla volta, a partire da `readPos`. Ogni byte letto viene immediatamente scritto nella posizione `writePos`, sovrascrivendo i dati che si trovavano precedentemente in quella posizione. Dopo ogni operazione di lettura e scrittura, le posizioni `readPos` e `writePos` vengono aggiornate per continuare il processo fino alla fine del file.

Una volta che tutti i dati rimanenti sono stati spostati, la funzione utilizza `ftruncate` per ridurre la dimensione del file, eliminando i byte in eccesso che non sono più necessari. Infine, la posizione del file viene riportata a `startingPos` per garantire che il puntatore del file rimanga coerente con la posizione iniziale.

Se si verifica un errore durante la scrittura, la funzione restituisce il valore di errore (`errno`). In caso di successo, restituisce 0. Questo approccio garantisce che i byte specificati vengano eliminati in modo sicuro, senza lasciare spazi vuoti o dati non validi nel file.

Gestione Input Utente (`userInput.h` e `userInput.c`)

Descrizione: Fornisce funzioni per leggere input dall'utente, come numeri interi e stringhe. Gestisce l'attesa di input da parte dell'utente.

Funzioni Principali:

- `waitForKey`: Attende un input dall'utente.
- `readInt`: Legge un numero intero dall'utente.
- `readStr`: Legge una stringa dall'utente.

Relazioni: Utilizzato dal client per leggere input dall'utente.

Server HTTP (`httpServer.h` e `httpServer.c`)

Descrizione: Implementa un server HTTP che gestisce le connessioni dei client e instrada le richieste alle rotte appropriate. Fornisce funzioni per costruire risposte HTTP e gestire le richieste dei client. Di seguito sono descritte le principali strutture dati e funzioni presenti nel codice.

Strutture Dati:

- `HttpRoute`: Struttura per rappresentare una rotta HTTP.
- `ParsedHttpRequest`: Struttura per rappresentare una richiesta HTTP analizzata.
- `HttpServer`: Struttura per rappresentare il server HTTP.
- `WorkerThreadParams`: Parametri per i thread lavoratori.

Funzioni Principali:

- `addHttpSubroute`: Aggiunge una nuova rotta figlia a una rotta esistente.
- `findHttpRoute`: Trova una rotta HTTP.
- `httpResponseBuilder`: Costruisce una risposta HTTP.
- `errorResponse`: Costruisce una risposta di errore HTTP.
- `parseHttpRequest`: Analizza una richiesta HTTP.
- `sendall`: Invia tutti i dati su un socket.
- `handleClient`: Gestisce un client.
- `printHostInfo`: Raccoglie e stampa le informazioni sull'host, inclusi il nome dell'host e gli indirizzi IP risolti
- `workerRoutine`: Routine del thread dei lavoratori.
- `handleSig`: Gestisce i segnali.
- `httpServerServe`: Avvia il server HTTP.

Relazioni: Utilizza funzioni definite in `userInput.h` e `httpLib.h`.

Il file `httpServer.c` contiene l'implementazione di un server HTTP in C. La selezione attiva include diverse funzioni chiave per la gestione delle rotte HTTP, la costruzione delle risposte, la gestione delle richieste, la gestione dei client e l'avvio del server.

La variabile globale `running` di tipo `atomic_int` è la inizializza inizialmente con il valore 1 essa funge da flag per controllare l'arresto del server.

Il tipo `atomic_int` è una variabile atomica fornita dalla libreria standard C (a partire da C11) e garantisce che le operazioni su di essa siano eseguite in modo atomico. Questo significa che le operazioni di lettura, scrittura o modifica della variabile non possono essere interrotte da altri thread o processi, rendendola sicura per l'accesso concorrente in ambienti multithread.

Rotte HTTP

La funzione `addHttpSubroute` aggiunge una nuova rotta figlia a una rotta esistente. Se la rotta padre non ha figli, la nuova rotta viene aggiunta come figlio. Altrimenti, la funzione scorre i fratelli della rotta padre fino a trovare l'ultimo e aggiunge la nuova rotta come suo fratello.

La funzione `findHttpRoute` cerca una rotta specifica a partire da un nodo radice. La funzione divide il percorso in token e scorre i figli del nodo corrente per trovare una corrispondenza. Se trova una corrispondenza, aggiorna il nodo corrente e continua con il token successivo. Se non trova una corrispondenza, restituisce `NULL`.

Risposte HTTP

La funzione `httpResponseBuilder` costruisce una risposta HTTP formattata utilizzando un modello di risposta e i parametri forniti (codice di stato, messaggio di stato e corpo della risposta).

```
#define HTTP_RESPONSE_TEMPLATE "HTTP/1.1 %d %s\r\n" \

"Content-Type: text/plain\r\n" \

"Content-Length: %zu\r\n" \

"\r\n" \

"%s"
```

La funzione `errorResponse` costruisce una risposta di errore HTTP basata sul codice di errore fornito. Utilizza `httpResponseBuilder` per creare la risposta con il messaggio di stato e il corpo della risposta appropriati.

Richieste HTTP

La funzione `parseHttpRequest` analizza una richiesta HTTP grezza e popola una struttura `ParsedHttpRequest` con il metodo HTTP, il percorso e il corpo della richiesta. La funzione identifica il metodo HTTP confrontando l'inizio della richiesta con le stringhe dei metodi noti. Trova il percorso cercando gli spazi nella richiesta e trova il corpo della richiesta cercando la sequenza `\r\n\r\n`.

Socket

La funzione `sendall` è progettata per inviare tutti i dati contenuti in un buffer a un socket specificato, gestendo eventuali invii parziali che possono verificarsi durante la comunicazione.

La funzione utilizza un ciclo `while` per assicurarsi che tutti i dati vengano inviati. All'inizio, le variabili `total` e `bytesleft` vengono inizializzate rispettivamente a 0 e alla lunghezza totale dei dati da inviare (`*len`).

All'interno del ciclo:

- La funzione `send` viene chiamata per inviare i dati rimanenti, partendo dal punto corretto del buffer (`buf + total`) e per una lunghezza pari a `bytesleft`.
- Se `send` restituisce `-1`, significa che si è verificato un errore. In particolare, se l'errore è `EPIPE`, viene stampato un messaggio che indica che la connessione con il client è stata chiusa inaspettatamente. In questo caso, il ciclo si interrompe.
- Se l'invio ha successo, il numero di byte inviati (`n`) viene aggiunto a `total`, e `bytesleft` viene decrementato dello stesso valore, aggiornando così il progresso dell'invio.
- Al termine del ciclo, il valore di `*len` viene aggiornato con il numero totale di byte inviati. La funzione restituisce `0` se tutti i dati sono stati inviati correttamente, oppure `-1` in caso di errore.

Client e handleClient

La funzione `handleClient` è progettata per gestire una connessione HTTP da parte di un client. Riceve come parametri il file descriptor del socket della connessione (`connSocketFd`) e un puntatore alla radice delle rotte HTTP (`HttpRoute *root`). La funzione si occupa di leggere la richiesta del client, elaborarla e inviare una risposta appropriata.

All'inizio, vengono dichiarati due buffer: `rawRequest` per memorizzare la richiesta HTTP grezza e `response` per costruire la risposta HTTP. Viene anche configurato un `timeout` di 10 secondi per le operazioni di lettura dal socket utilizzando `setsockopt`. Se questa configurazione fallisce, viene stampato un messaggio di errore e la funzione termina con un valore di ritorno pari a -1.

La fase di lettura utilizza un ciclo `while` per ricevere i dati dal socket in blocchi, accumulandoli in `rawRequest`. La lettura continua finché la richiesta non è completa o finché non viene raggiunta la dimensione massima del buffer (`MAX_REQUEST_SIZE`). Se si verifica un errore durante la lettura, come un timeout o una disconnessione del client, la funzione gestisce l'errore e termina. La completezza della richiesta viene verificata cercando la sequenza `\r\n\r\n`, che separa gli header HTTP dal corpo. Se è presente un header `Content-Length`, la funzione calcola la lunghezza del corpo e verifica se è stato ricevuto completamente.

Se la richiesta è valida e completa, viene analizzata utilizzando `parseHttpRequest`, che popola una struttura `ParsedHttpRequest`. La funzione cerca quindi una rotta corrispondente al percorso richiesto utilizzando `findHttpRoute`. Se la rotta esiste e il metodo HTTP è supportato, viene chiamato il gestore associato alla rotta per elaborare la richiesta e costruire la risposta. In caso contrario, viene generata una risposta di errore appropriata, come "Metodo non consentito" o "Risorsa non trovata".

Infine, la risposta viene inviata al client utilizzando `sendall`. Se si verifica un errore durante l'invio, viene stampato un messaggio di errore e la funzione termina con -1. Se tutto va a buon fine, la funzione restituisce 0, indicando che la richiesta è stata gestita correttamente. Questo approccio garantisce una gestione robusta delle richieste HTTP, con un'attenzione particolare alla validazione dei dati e alla gestione degli errori.

Thread di Lavoro

La funzione `workerRoutine` è la routine eseguita dai thread di lavoro. Accetta le connessioni in arrivo, gestisce i client chiamando `handleClient` e chiude le connessioni.

Informazioni sull'Host

La funzione `printHostInfo` raccoglie e stampa le informazioni sull'host, inclusi il nome dell'host e gli indirizzi IP risolti.

Funzione `handleSigStop`

Questa funzione viene chiamata quando il server riceve un segnale che richiede l'arresto (ad esempio, un segnale personalizzato o un segnale di terminazione).

La funzione stampa un messaggio per indicare che il server sta per essere arrestato e utilizza `atomic_fetch_sub` per decrementare in modo atomico il valore della variabile globale `running`. Questo decremento consente di tracciare il numero di richieste di arresto ricevute. Successivamente, viene controllato il valore di `running` con `atomic_load`. Se il valore scende a -2 o inferiore, ciò indica che il

server è stato forzato a fermarsi (ad esempio, dopo più tentativi di arresto). In questo caso, viene stampato un messaggio di errore e il programma termina immediatamente con `exit(EXIT_FAILURE)`.

Funzione `handleCriticalError`

Questa funzione gestisce errori critici che si verificano quando il server riceve segnali come `SIGSEGV` (errore di segmentazione), `SIGBUS` (errore di bus), `SIGFPE` (eccezione di punto flottante), `SIGILL` (istruzione illegale) o `SIGABRT` (abort). Il parametro `sig` identifica il segnale ricevuto e viene utilizzato per determinare il tipo di errore.

La funzione inizia con uno `switch` per associare il numero del segnale a una descrizione leggibile (`signame`). Successivamente, stampa un messaggio dettagliato che include il tipo di segnale ricevuto e il numero associato, segnalando che il server tenterà di eseguire una chiusura sicura. La variabile globale `running` viene impostata a `-10` utilizzando `atomic_store`, indicando che il server è in uno stato critico e deve essere terminato.

Per garantire che il segnale venga gestito correttamente dal sistema, la funzione reimposta il gestore del segnale al comportamento predefinito (`SIG_DFL`) utilizzando `sigaction`. Infine, il segnale originale viene rilanciato con `raise(sig)`, consentendo al sistema di eseguire l'azione predefinita per quel segnale (ad esempio, generare un core dump).

Funzione `httpServerServe`

La funzione `httpServerServe` rappresenta il cuore del server HTTP, gestendo l'inizializzazione, la configurazione, l'esecuzione e la terminazione del server.

La funzione configura tre gruppi di gestori di segnali:

- **Gestione dell'arresto** (`handleSigStop`): Viene associata ai segnali come `SIGINT`, `SIGQUIT`, `SIGTERM` e `SIGHUP`, che indicano richieste di terminazione del server. Questo consente di gestire l'arresto in modo controllato.
- **Ignorare segnali non critici**: Alcuni segnali, come `SIGPIPE`, `SIGALRM`, `SIGUSR1` e `SIGUSR2`, vengono ignorati per evitare che interferiscano con il funzionamento del server.
- **Gestione degli errori critici** (`handleCriticalError`): Segnali come `SIGSEGV`, `SIGBUS`, `SIGFPE`, `SIGILL` e `SIGABRT` vengono gestiti per tentare una chiusura sicura in caso di errori gravi.

Il server crea un `socket TCP` utilizzando `socket` e lo configura per riutilizzare l'indirizzo (`SO_REUSEADDR`). Successivamente, il socket viene associato a un indirizzo e una porta specifici tramite `bind`. Se una di queste operazioni fallisce, il server stampa un messaggio di errore e termina.

Dopo il binding, il server entra in modalità di ascolto utilizzando `listen`, consentendo di accettare connessioni in entrata. La funzione `printHostInfo` stampa informazioni sull'host, e un messaggio conferma che il server è in ascolto sulla porta specificata.

Il server utilizza un **pool di thread** per gestire le connessioni in entrata. Ogni thread esegue la funzione `workerRoutine`, che probabilmente gestisce le richieste dei client. I parametri per ciascun thread sono configurati in un array di strutture `WorkerThreadParams`. I thread vengono creati in modalità **"detached"** per evitare la necessità di unirli manualmente.

Routine principale e cleanup

Il primo thread worker (`workerParams[0]`) viene eseguito direttamente nel thread principale. Al termine, il server chiude il socket, distrugge gli attributi dei thread e stampa un messaggio di conferma dell'arresto. Infine, attende un input dell'utente con `waitForKey` prima di terminare.

File `httpServer.h`

Il file `httpServer.h` è un header file per un server HTTP scritto in C. Questo file definisce le strutture dati e le funzioni necessarie per gestire le rotte HTTP, le richieste e le risposte e il funzionamento del server.

Il file include diverse librerie standard di C necessarie per la gestione dei socket, dei thread, della memoria condivisa e della sincronizzazione. Inoltre, include due header file personalizzati: `userInput.h` e `httpLib.h`.

Definizioni di Costanti

Definisce diverse costanti utilizzate nel server:

- `MAX_ROUTE_NAME`: Lunghezza massima del nome di una rotta (100 caratteri).
- `MAX_REQUEST_SIZE`: Dimensione massima di una richiesta HTTP (4096 byte).
- `MAX_RESPONSE_SIZE`: Dimensione massima di una risposta HTTP (4096 byte).
- `BUFFER_SIZE`: Dimensione del buffer utilizzato per le operazioni di rete (1024 byte).
- `HTTP_RESPONSE_TEMPLATE`: Modello di risposta HTTP utilizzato per formattare le risposte.

Strutture Dati

HttpRoute: Rappresenta una rotta HTTP nel server. Contiene:

- `name`: Nome della rotta.
- `handlers`: Array di puntatori a funzioni per gestire i vari metodi HTTP.
- `children`, `parent`, `sibling`, `child`: Puntatori per costruire una struttura ad albero delle rotte.

ParsedHttpRequest: Rappresenta una richiesta HTTP analizzata. Contiene:

- `method`: Metodo HTTP (GET, POST, ecc.).
- `path`: Percorso della richiesta.
- `body`: Corpo della richiesta.

HttpServer: Rappresenta il server HTTP. Contiene:

- `port`: Porta su cui il server ascolta le connessioni.

- `numThreads`: Numero di thread dei lavoratori.
- `root`: Puntatore alla rotta radice.

`WorkerThreadParams`: Parametri per i thread dei lavoratori. Contiene:

- `id`: ID del thread.
- `serverSocket`: Socket del server.
- `root`: Puntatore alla rotta radice.

Dichiarazioni di Funzioni

- `addHttpSubroute`: Aggiunge una nuova sotto-rotta a una rotta esistente nell'albero delle rotte HTTP.
- `findHttpRoute`: Trova una rotta nell'albero delle rotte HTTP in base al percorso fornito.
- `httpResponseBuilder`: Costruisce una risposta HTTP con il codice di stato, il messaggio e il corpo forniti.
- `errorResponse`: Genera una risposta di errore HTTP con il codice di errore specificato.
- `parseHttpRequest`: Analizza una stringa di richiesta HTTP in una struttura `ParsedHttpRequest`.
- `setNonBlocking`: Imposta un socket in modalità non bloccante.
- `nonBlockWriteSocket`: Scrive dati su un socket non bloccante.
- `nonBlockReadSocket`: Legge dati da un socket non bloccante.
- `handleClient`: Gestisce una connessione client individuale.
- `workerRoutine`: Routine principale per i thread dei lavoratori che gestiscono le connessioni dei client.
- `httpServerServe`: Inizializza e avvia il server HTTP.

Queste funzioni e strutture lavorano insieme per fornire un'implementazione completa di un server web HTTP, gestendo le richieste, le risposte, le connessioni dei client e la gestione dei thread lavoratori.

Client HTTP (httpClient.h e httpClient.c)

Descrizione: Implementa un client HTTP che invia richieste al server e riceve risposte. Fornisce funzioni per connettersi al server e inviare richieste HTTP.

Strutture Dati:

- `TargetHost`: Struttura per rappresentare un host di destinazione.

Funzioni Principali:

- `connectToHost`: Connette il client al server.
- `sendHttpRequest`: Invia una richiesta HTTP.
- `removeHttpHeaders`: Rimuove gli header HTTP dalla risposta.

Relazioni: Utilizza funzioni definite in `utils.h` e `httpLib.h`.

Il file `httpClient.c` contiene funzioni per gestire le connessioni HTTP tramite socket. Include il file di intestazione `httpClient.h`.

Funzione `connectToHost`

La funzione `connectToHost` è progettata per stabilire una connessione TCP con un host remoto specificato tramite una struttura `TargetHost`. Analizziamo il codice passo per passo.

All'inizio, viene creato un socket utilizzando la funzione `socket`. Il socket è configurato per utilizzare il protocollo IPv4 (`AF_INET`) e il protocollo TCP (`SOCK_STREAM`). Se la creazione del socket fallisce (ritorna un valore negativo), viene stampato un messaggio di errore e il programma termina con `exit(1)`.

Successivamente, vengono configurati i parametri dell'indirizzo dell'host remoto. Il campo `sin_family` della struttura `sockaddr_in` è impostato su `AF_INET` per indicare che si sta utilizzando IPv4. Il campo `sin_port` è configurato con il numero di porta dell'host remoto, convertito in formato di rete tramite la funzione `htons`. La funzione `inet_pton` viene utilizzata per convertire l'indirizzo IP in formato stringa (ad esempio, "192.168.1.1") in un formato binario compatibile con `sin_addr`. Se questa conversione fallisce, viene stampato un messaggio di errore e il programma termina.

Dopo aver configurato l'indirizzo, la funzione tenta di connettersi all'host remoto utilizzando `connect`. Se la connessione fallisce, viene stampato un messaggio di errore e il programma termina. La funzione `connect` utilizza il socket creato e l'indirizzo configurato per stabilire la connessione.

Infine, vengono impostati i timeout per le operazioni di ricezione e invio sul socket. Questo viene fatto utilizzando la funzione `setsockopt` con le opzioni `SO_RCVTIMEO` e `SO_SNDTIMEO`. I timeout sono configurati per 10 secondi (`timeout.tv_sec = 10`), con 0 microsecondi (`timeout.tv_usec = 0`). Se l'impostazione di uno dei timeout fallisce, viene stampato un messaggio di errore e il programma termina.

Funzione removeHttpHeaders

Questa funzione rimuove gli header HTTP dalla risposta del server. Prende una stringa response come parametro. Cerca la sequenza `\r\n\r\n` che separa gli header dal corpo della risposta. Se trova questa sequenza, sposta il corpo della risposta all'inizio della stringa.

Funzione getHttpStatusCode

Questa funzione estrae il codice di stato HTTP dalla risposta del server. Prende una stringa response come parametro e restituisce il codice di stato come intero. Verifica se la risposta inizia con "HTTP/" e se la prima riga contiene un codice di stato valido. Se la risposta non è valida, restituisce -1.

Funzione sendHttpRequest

Questa funzione invia una richiesta HTTP al server e riceve la risposta. Prende come parametri un puntatore a una struttura `TargetHost`, un metodo HTTP (`HttpMethod`), un percorso (`path`), un corpo della richiesta (`body`) e un buffer per la risposta (`response`). La funzione esegue i seguenti passaggi:

Chiama `connectToSockServer` per stabilire una connessione con il server.

Costruisce la richiesta HTTP in base al metodo specificato e al corpo della richiesta, se presente.

Invia la richiesta al server utilizzando la funzione `send`.

Legge la risposta del server utilizzando la funzione `recv` e la memorizza nel buffer response.

Chiama `getHttpStatusCode` per estrarre il codice di stato dalla risposta.

Chiude il socket e restituisce il codice di stato.

Esempio di Utilizzo

Le richieste HTTP vengono utilizzate per comunicare con un server remoto e ottenere o inviare dati relativi alla prenotazione dei posti per il cinema. Ecco una spiegazione dettagliata delle richieste HTTP utilizzate:

Richiesta per ottenere gli orari dei film:

```
sendHttpRequest(targetHost, GET, "/films/showtimes", requestBody, response);
```

Questa richiesta viene inviata per ottenere gli orari disponibili per un determinato film. Utilizza il metodo `GET` e l'endpoint `/films/showtimes`. Il `requestBody` contiene l'ID del film, e la risposta viene memorizzata in `response`.

Richiesta per ottenere la mappa della sala:

```
sendHttpRequest(targetHost, GET, "/films/map", requestBody, response);
```

Questa richiesta viene inviata per ottenere la mappa della sala per un determinato orario di un film. Utilizza il metodo `GET` e l'endpoint `/films/map`. Il `requestBody` contiene l'ID del film e la scelta dell'orario, e la risposta viene memorizzata in `response`.

Richiesta per prenotare i posti:

```
sendHttpRequest(targetHost, POST, "/book", requestBody, response);
```

Questa richiesta viene inviata per prenotare i posti selezionati. Utilizza il metodo `POST` e l'endpoint `/book`. Il `requestBody` contiene i dettagli della prenotazione, inclusi i posti selezionati, e la risposta viene memorizzata in `response`.

Richiesta per ottenere la lista dei film:

```
sendHttpRequest(&targetHost, GET, "/films", NULL, buffer);
```

Questa richiesta viene inviata per ottenere la lista dei film disponibili. Utilizza il metodo `GET` e l'endpoint `/films`. Non richiede un corpo della richiesta (`NULL`), e la risposta viene memorizzata in `buffer`.

Richiesta per ottenere la lista dei film per la prenotazione:

```
sendHttpRequest(&targetHost, GET, "/films/list", NULL, buffer);
```

Questa richiesta viene inviata per ottenere la lista dei film disponibili per la prenotazione. Utilizza il metodo `GET` e l'endpoint `/films/list`. Non richiede un corpo della richiesta (`NULL`), e la risposta viene memorizzata in `buffer`.

Richiesta per ottenere la pagina principale:

```
sendHttpRequest(&targetHost, GET, "/", NULL, buffer);
```

Questa richiesta viene inviata per ottenere la pagina principale del sistema. Utilizza il metodo `GET` e l'endpoint `/`. Non richiede un corpo della richiesta (`NULL`), e la risposta viene memorizzata in `buffer`.

In generale, le richieste `GET` vengono utilizzate per ottenere dati dal server, mentre le richieste `POST` vengono utilizzate per inviare dati al server. Le risposte ricevute vengono poi elaborate e visualizzate all'utente.

httpClient.h

Il file `httpClient.h` è un file di intestazione per un client HTTP che utilizza socket per comunicare con un server. Inizia con le direttive di precompilazione `#ifndef`, `#define` e `#endif` per evitare inclusioni multiple del file.

Include diverse librerie standard di C necessarie per le operazioni di input/output (`stdio.h`), gestione della memoria (`stdlib.h`), manipolazione delle stringhe (`string.h`), funzioni di sistema (`unistd.h`), operazioni di rete (`arpa/inet.h` e `netdb.h`). Include anche i file di intestazione `utils.h` e `httpLib.h` per funzioni e definizioni aggiuntive.

Definisce una costante `BUFFER_SIZE` che rappresenta la dimensione del buffer utilizzato per le operazioni di rete, impostata a 4096 byte.

La struttura `TargetHost` rappresenta un host di destinazione per la connessione HTTP e contiene i seguenti campi:

- `ip_addr`: un puntatore a una stringa che memorizza l'indirizzo IP dell'host.
- `portno`: un intero che rappresenta il numero di porta dell'host.
- `server_addr`: una struttura `sockaddr_in` che memorizza l'indirizzo del server.
- `sockfd`: un intero che rappresenta il file descriptor del socket.

Il file dichiara tre funzioni principali per gestire le operazioni HTTP:

- `connectToSockServer`: Questa funzione stabilisce una connessione socket con l'host di destinazione specificato nella struttura `TargetHost`.
- `sendHttpRequest`: Questa funzione invia una richiesta HTTP all'host di destinazione e riceve la risposta. Prende come parametri un puntatore a una struttura `TargetHost`, un metodo HTTP (`HttpMethod`), un percorso (`path`), un corpo della richiesta (`body`) e un buffer per la risposta (`response`). Restituisce il codice di stato HTTP della risposta.
- `removeHttpHeaders`: Questa funzione rimuove gli header HTTP dalla risposta del server. Prende una stringa `response` come parametro e modifica la stringa per contenere solo il corpo della risposta.

Queste funzioni lavorano insieme per fornire un'implementazione completa per la gestione delle connessioni HTTP tramite socket, inclusa la creazione del socket, l'invio delle richieste, la ricezione delle risposte e la gestione degli header HTTP.

httpLib.h

Il file `httpLib.h` è un file di intestazione che definisce enumerazioni per i metodi HTTP e i codici di stato HTTP. Inizia con le direttive di precompilazione `#ifndef`, `#define` e `#endif` per evitare inclusioni multiple del file.

Enumerazione HttpMethod

L'enumerazione `HttpMethod` definisce i vari metodi HTTP che possono essere utilizzati nelle richieste HTTP. I metodi inclusi sono:

- `GET`
- `POST`
- `PUT`
- `DELETE`
- `HEAD`
- `OPTIONS`
- `TRACE`
- `CONNECT`

- PATCH
- UNKNOWN

Questi metodi rappresentano le diverse operazioni che possono essere eseguite su una risorsa HTTP.

Enumerazione HttpStatusCode

L'enumerazione `HttpStatusCode` definisce i vari codici di stato HTTP che possono essere restituiti in risposta a una richiesta HTTP.

Libreria HTTP (`httpLib.h`)

Descrizione: Definisce enumerazioni e costanti per i metodi HTTP e i codici di stato. Utilizzato sia dal server che dal client per gestire le richieste e le risposte HTTP.

Strutture Dati:

- `HttpMethod`: Enum per rappresentare i metodi HTTP.

Relazioni: Utilizzato sia dal server che dal client per gestire le richieste e le risposte HTTP.

Libreria CSV (`csvlib.h` e `csvlib.c`)

Descrizione: Fornisce funzioni per leggere, scrivere e gestire file CSV. Utilizzato per gestire i dati dei film e delle prenotazioni.

Strutture Dati:

- `CSVFile`: Struttura per rappresentare un file CSV.

Funzioni Principali:

- `readCsvHeadings`: Legge le intestazioni del CSV.
- `readCsvRows`: Legge le righe del CSV.
- `appendCsvRow`: Aggiunge una riga al CSV.
- `csvPrint`: Stampa il contenuto del CSV.
- `csvInit`: Inizializza un file CSV.
- `csvFree`: Libera la memoria allocata per il CSV.
- `getStringField`: Ottiene un campo stringa dal CSV.
- `getIntField`: Ottiene un campo intero dal CSV.

Relazioni: Utilizzato per gestire i dati dei film e delle prenotazioni.

Il file `csvlib.c` contiene funzioni per leggere, scrivere e gestire file CSV. Include il file di intestazione `csvlib.h`.

Funzioni di Lettura

`readCsvHeadings`: Questa funzione legge le intestazioni di un file CSV. Verifica se il file è aperto correttamente, quindi legge la prima riga del file. Divide la riga in token separati da virgole e memorizza

ogni intestazione in un array dinamico. Se ci sono errori durante l'allocazione della memoria o la duplicazione dei token, la funzione stampa un messaggio di errore e termina il programma.

readCsvRows: Questa funzione legge le righe di dati di un file CSV. Verifica se il file è aperto correttamente, quindi legge ogni riga del file. Ignora le righe vuote e divide ogni riga in campi separati da virgole, tenendo conto delle virgolette per i campi che contengono virgole. Memorizza ogni campo in un array bidimensionale dinamico. Se ci sono errori durante l'allocazione della memoria o la duplicazione dei campi, la funzione stampa un messaggio di errore e termina il programma.

Funzioni di Inizializzazione e Deallocazione

csvInit: Questa funzione inizializza una struttura **CSVFile**. Alloca memoria per la struttura, apre il file CSV in modalità append e lettura, e imposta i campi iniziali della struttura. Se ci sono errori durante l'allocazione della memoria o l'apertura del file, la funzione stampa un messaggio di errore e termina il programma.

csvFree: Questa funzione dealloca la memoria utilizzata da una struttura **CSVFile**. Libera la memoria per le intestazioni e le righe, chiude il file e libera la memoria per la struttura stessa.

Funzioni di Stampa

csvPrint: Questa funzione stampa le intestazioni e le righe di un file CSV. Itera attraverso le intestazioni e le righe, stampando ogni campo.

Funzioni di Accesso ai Campi

getStringField: Questa funzione restituisce il valore di un campo specifico in una riga specifica, dato il nome dell'intestazione. Cerca l'intestazione e restituisce il valore corrispondente nella riga specificata.

getIntField: Questa funzione restituisce il valore intero di un campo specifico in una riga specifica, dato il nome dell'intestazione. Converte il valore del campo in un intero e gestisce eventuali errori di conversione.

Funzioni di Aggiunta

appendCsvRow: Questa funzione aggiunge una nuova riga a un file CSV. Verifica che il numero di campi nella nuova riga corrisponda al numero di intestazioni; quindi, aggiunge la riga al file e rilegge tutte le righe per aggiornare la struttura **CSVFile**.

Csvlib.h

Il file **csvlib.h** è un file di intestazione per una libreria che gestisce operazioni su file CSV. Inizia con le direttive di precompilazione **#ifndef**, **#define** e **#endif** per evitare inclusioni multiple del file.

Include diverse librerie standard di C, come **stdio.h**, **stdbool.h**, **stdlib.h**, **string.h**, **limits.h**, **stdarg.h** e **unistd.h**, necessarie per le operazioni di input/output, gestione della memoria, manipolazione delle stringhe, gestione dei limiti dei tipi di dati, gestione degli argomenti variabili e funzioni di sistema.

Definisce una costante **MAX_LINE_LENGTH** che rappresenta la lunghezza massima di una riga nel file CSV, impostata a 4096 caratteri.

La struttura `CSVFile` rappresenta un file CSV e contiene i seguenti campi:

- `filename`: un array di caratteri che memorizza il nome del file.
- `headings`: un array di puntatori a caratteri che memorizza le intestazioni delle colonne.
- `headings_count`: un contatore che memorizza il numero di intestazioni.
- `rows`: un array bidimensionale di puntatori a caratteri che memorizza le righe di dati.
- `rows_count`: un contatore che memorizza il numero di righe.
- `file`: un puntatore a `FILE` che rappresenta il file CSV aperto.

Il file dichiara diverse funzioni per gestire le operazioni sui file CSV:

Funzioni di Lettura

`void readCsvHeadings(CSVFile *csv)`: legge le intestazioni del file CSV.

`void readCsvRows(CSVFile *csv)`: legge le righe di dati del file CSV.

Funzioni di Aggiunta

`void appendCsvRow(CSVFile *csv, const char *newRow)`: aggiunge una nuova riga al file CSV.

Funzioni di Stampa

`void csvPrint(CSVFile *csv)`: stampa le intestazioni e le righe del file CSV.

Funzioni di Inizializzazione e Deallocazione

`CSVFile *csvInit(char *filename)`: inizializza una struttura `CSVFile`.

`void csvFree(CSVFile *csv)`: dealloca la memoria utilizzata da una struttura `CSVFile`.

Funzioni di Accesso ai Campi

`char *getStringField(CSVFile *csv, char *heading, size_t row)`: restituisce il valore di un campo specifico in una riga specifica, dato il nome dell'intestazione.

`int getIntField(CSVFile *csv, char *heading, size_t row)`: restituisce il valore intero di un campo specifico in una riga specifica, dato il nome dell'intestazione.

Queste funzioni lavorano insieme per fornire un'implementazione completa per la gestione di file CSV, inclusa la lettura, la scrittura, l'inizializzazione, la deallocazione e l'accesso ai campi.

Relazioni tra i Componenti

Server e Client: Il client invia richieste HTTP al server, che le elabora e restituisce le risposte appropriate.

Gestione Cinema e Generazione Mappa: Le funzioni di gestione cinema utilizzano le funzioni di generazione mappa per creare rappresentazioni visive delle sale.

Parser CSV e Gestione Cinema: Il parser CSV legge i dati dei film e li memorizza nelle strutture dati utilizzate dalle funzioni di gestione cinema.

Utilità e Altri Moduli: Le funzioni di utilità sono utilizzate in tutto il sistema per operazioni comuni come la gestione dei buffer e la conversione di stringhe.

Gestione Input Utente e Client: Le funzioni di gestione input utente sono utilizzate dal client per leggere input dall'utente.

Server HTTP e Libreria HTTP: Il server HTTP utilizza la libreria HTTP per gestire le richieste e le risposte HTTP.

Client HTTP e Libreria HTTP: Il client HTTP utilizza la libreria HTTP per inviare richieste e ricevere risposte dal server.

Interfaccia Grafica e Client: Le funzioni di interfaccia grafica sono utilizzate dal client per migliorare l'esperienza utente.

Conclusione

Il sistema di prenotazione posti per cinema è un software che utilizza diversi moduli per fornire le funzionalità richieste. Ogni modulo ha un ruolo specifico e collabora con gli altri moduli per garantire il corretto funzionamento del sistema. La modularità del software consente una facile manutenzione e l'estensione delle funzionalità in futuro.