BookingService – Technical Brief & Demo Validation Guide

1. System Overview

Goal:
A fully in-memory, thread-safe backend that enables a CLI/UI to view:

- View only active movies i.e., actually playing (those with at least one show)
- Select a movie → list its theaters (via active shows)
- Select a show → list available seats
- Book one or more seats with **no overbooking** under concurrent requests

Shape:
A single service object (BookingService) exposing a small, documented API; a CLI that drives it; and unit tests that prove correctness and concurrency safety. The CLI demonstrates graceful error handling and signal handling for Ctrl-C (SIGINT).

2. Core Design & Data Structures

Entities:

- Movie { id:int, title:string }_
- Theater { id:int, name:string }
- Show { movieId:int, theaterId:int, seats:vector<bool>, availableCount:int, maximite }

Data Structures/ In-memory storage (hash maps):
Uses unordered_map for O(1) lookup and insert operations.

- unordered_map<int, Movie> movies_
- unordered_map<int, Theater> theaters_
- unordered_map<long long, shared_ptr<Show>> shows_

Values are small aggregates or shared pointers; move construction is cheap; try_emplace avoids redundant moves and constructs in place

IDs & uniqueness:

- ID generation uses atomic counters (std::atomic<int/long long>) for race-free increments without taking the global lock.
- No duplicate movie titles/theater names: case-insensitive check before insert.
- No duplicate shows for the same (movieId, theaterId) pair

Optimizations:
• movieNameToId_ and theaterNameToId_ for O(1) duplicate check.
• showLookup_ for O(1) (movieId, theaterId) mapping.
• activeMovies_ and movieToTheaters_ for fast movie and theater listings.

3. Concurrency Model

Locking Strategy:
Global shared_mutex for read-write control (shared_lock for reads, unique_lock for writes).

- **Global RW lock** over the catalogs: mutable std::shared_mutex mtx_
    - **Readers** (std::shared_lock) for pure lookups (getAvailableSeats, listing functions).
    - **Writers** (std::unique_lock) for mutations (add movie/theater, create show).

- **Per-show exclusive lock**: std::mutex inside each Show
    - Seat booking is isolated to the **selected show,** so unrelated shows don't block.
    - Lock scope is **minimal**: we take the show pointer under the global shared lock, **release global lock**, then take the **per-show** lock for seat verification and commit.

This isolates contention and prevents overbooking even with many threads.

Demonstration: The tests launch multiple threads booking the same seat; only one succeeds. There's also a stress test with N threads booking random seats, and we assert booked + available == total. So, it guarantees parallel seat bookings across different shows, minimal contention, and no double-booking.

4. Seat Model & Helpers

Seat storage uses vector<bool> with cached availableCount. It's compact and sufficient; with only 20 seats, the differences between vector<bool> vs std::bitset<20> are negligible while vector<bool> keeps the "variable count" option open (for flexibility/scaling) if we ever change capacity without recompiling.

Labeling: A1..A20. Helpers convert labels ⇄ indices and generate seat labels.
Centralized helpers avoid ad-hoc parsing; any validation change is in one place.

5. API Surface & Modern C++ Features

- **addMovie, addTheater, createShow**
    - Take unique-lock (writers), validate uniqueness (case-insensitive), then try_emplace.
    - Atomics generate IDs (no blocking for the increment itself).

- **getAvailableSeats(showId)**
    - Shared-lock to find the show; then **per-show lock** to read seats reliably and build the list.
- **bookSeats(showId, vector<string> seatLabels)**
    - Shared-lock to find the show → unlock → per-show lock

  - Validate all seats (valid label, not already booked, no duplicates in request) → **commit all** or **fail all** (transactional semantics). This avoids partial booking surprises for clients under concurrency.

- **[[nodiscard]]** is applied where callers must not ignore failures (e.g., booking, creation returning IDs). It helps compilers warn about ignored outcomes in critical paths.

Modern C++ features used:

- RAII locks (std::unique_lock, std::shared_lock, std::lock_guard)
- std::atomic counters for IDs
- std::unordered_map + try_emplace (in-place construction, avoids redundant moves)
- std::shared_ptr for Show lifetime (safe references across threads while not holding global locks)
- constexpr domain constants (seat row, seat count)
- Range-based for, structured bindings in listings
- Exception-free "false/true" for booking failures to keep the hot path fast and predictable
- [[nodiscard]] - Enforces caller checks on critical results.
- Structured bindings - Cleaner iteration over maps (for (auto& [id, obj])).

6. Error Handling & UX

- **Service API**: returns bool/IDs; validation failures become clear false/-1 results and (optionally) log messages.

- **CLI**: robust I/O parsing with std::getline + std::stoi/stoll wrapped in safe helpers; re-prompts on invalid input; never crashes on non-numeric entries; prints friendly messages. Ctrl-C triggers a graceful shutdown via a signal handler.

7. Performance Consideration

All major operations are O(1) average after optimization:

- O(1) duplicate checks for movies/theaters.
- O(1) show creation using composite key.
- O(M_active) movie listing via active cache.
- O(K) theater listing per movie.
- Cached available seat count avoids rescans.

Contention reduction: per-show locking instead of a single global mutex means different shows can be booked in parallel.

Lock scope minimization: we only hold the global lock to *find* the show, then release it before seat validation/commit.

Data locality: 20-seat vector fits in a few cache lines; ops are branch-light after validation.

Branching: using operator[] = std::move(ptr) when the key is guaranteed unique (from atomic counter) is as fast and clearer than try_emplace there (we still use try_emplace where uniqueness must be *checked*—e.g., titles, names, (movieId, theaterId))

8. Complexity Analysis (with Locking Overhead)

This section summarizes both algorithmic and synchronization costs for each key function. Locking adds minor constant overheads per call, mainly under contention.

Function	Purpose	Time Complexity	Space Complexity	Locking Overhead / Notes
toLower()	Convert string to lowercase.	O(n)	O(n)	No lock. Pure string transform.
seatIndexFromLabel()	Parse label (A5).	O(1)	O(1)	No lock.
seatLabelFromIndex()	Format label.	O(1)	O(1)	No lock.
addMovie()	Add movie (duplicate-checked).	O(1)	O(1)	shared_lock→unique_lock transition (~50–150ns).
addTheater()	Add theater (duplicate-checked).	O(1)	O(1)	Single write lock, low contention.
createShow()	Create new show.	O(1)	O(1)	unique_lock; short critical section.
getAvailableSeats()	List unbooked seats.	O(1)	O(1)	shared_lock + per-show mutex; parallel safe.
bookSeats()	Book multiple seats atomically.	O(k)	O(1)	shared_lock + mutex; isolated per show.
listMovies()	List active movies.	O(M_active)	O(1)	Single shared_lock; non-blocking.

listTheatersForMovie()	List theaters for movie.	O(K)	O(1)	Single shared_lock.
getMovieTitle()	Get movie title by ID.	O(1)	O(1)	shared_lock.
getTheaterName()	Get theater name by ID.	O(1)	O(1)	shared_lock.
getAllShows()	List all shows.	O(S)	O(S)	Single shared_lock.
getAllMovies()	List all movies.	O(M)	O(M)	Single shared_lock.
getAllTheaters()	List all theaters.	O(T)	O(T)	Single shared_lock.

Locking Characteristics:
• shared_lock: multiple readers allowed, near O(1) acquisition under no contention.
• unique_lock: exclusive; linear scaling under heavy writers but short-lived.
• mutex: per-show lock (~50–100ns acquisition) ensures isolation.

Summary:
• Most operations are O(1) average with small synchronization cost.
• Read-heavy workloads scale efficiently with shared locks.
• Writes remain fast due to narrow lock scopes.

9. Testing Strategy

- **Mini Catch-style test harness** (MINI_CATCH_MAIN): self-contained, prints PASSED/FAILED lines and a summary.

- **Tests included: [Minimal tests]**

    1. **Basic flow**: add movie/theater, create show, check initial seats, book a couple, assert remaining count.
    2. **Invalid seat**: booking returns false (no exceptions on the hot path).
    3. **Concurrent single-seat booking**: 10 threads attempt A1; only one succeeds; counts reconcile.
    4. **Stress test**: configurable thread count (--threads via env/global) booking shuffled seats; assert integrity and print result.

How this proves concurrency correctness:
Tests cover both **mutual exclusion** (single seat) and **liveness** (many threads, many seats). The accounting check (remaining == total - successes) ensures no “ghost” bookings or double-bookings occurred.

10. CLI Design (User/Admin)

- **Menu** with input validation & re-prompting (no silent crashes on letters/garbage).
- **Signal handling**: SIGINT → clean exit message and shutdown.

11. Portability and Tooling

- **CMake**-based build (cmake -S . -B build && cmake --build build) with a separate test target.
- **No external DB** (per assignment).
- **Dockerfile** and **Conan** are optional helpers; not required to run but demonstrate packaging and dependency hygiene if desired.

12. Security and Robustness Notes

- Input validated at the CLI; the service itself validates IDs and label formats.
- No dynamic memory exposure: shows are owned by the service via shared_ptr; clients cannot mutate internals without locks.
- Booking is **atomic at the request level**—either all requested seats get booked, or none.

13. Extensibility and “What-ifs”

- **Multiple rows**: generalize label helpers from fixed “A” row to row*seat scheme (e.g., A1…D10). The seat container can be swapped to bitset<N> if N becomes compile-time or vector<bool> remains fine if N is config-driven.
- **Pricing/holds/timeouts**: per-show structure already has a seat mutex; we can add metadata and a small state machine (AVAILABLE/HOLD/BOOKED) with timestamps.
- **Persistence**: a persistence adapter can snapshot/restore maps at startup/shutdown without changing public APIs.
- **Horizontal scaling**: move to per-show sharding (or per theater) with an actor model if multiple processes are needed later.

Project Demo & Validation Guide

1. Build & Run

- **Compile**

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build --config Release
```

- **Run CLI**

```
./build/bin/booking_cli
```

2. Run Tests

```
# Display test help
./build/booking_tests –help

# Run all tests
./build/booking_tests

# Run only basic test
./build/booking_tests --filter="Basic"

# Run only concurrency test
./build/booking_tests --filter="Concurrent"

# Run stress test with 15 threads
./build/booking_tests --filter="stress" --threads=15
```

3. Test Scenarios (with Purpose)

Test Case	**Goal**	**Concurrency feature demonstrated**
Basic booking flow	Add → Create Show → Book few seats → Verify available count.	Confirms base logic.
Invalid seat booking	Book seat “Z9” → expect false.	Input validation path (no exceptions).
Concurrent booking	10 threads try to book same seat A1	Only one succeeds → proves mutual exclusion on seat mutex.
Stress Test	50 threads randomly book seats.	Checks lock contention & atomic integrity.

4. Concurrency Proof (Using Valgrind / Helgrind)

Build with debug info:

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Debug
cmake --build build -j
```

A. Run Valgrind Memcheck (detect leaks, invalid reads)

```
valgrind --leak-check=full ./build/booking_tests --filter="Basic"
```

Expected output snippet

```
==8316== Memcheck, a memory error detector
==8316== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==8316== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==8316== Command: ./build/booking_tests --filter=Basic
Basic booking flow works PASSED

Executed: 1 | Failed: 0
==8316== HEAP SUMMARY:
==8316==     in use at exit: 0 bytes in 0 blocks
==8316==   total heap usage: 87 allocs, 87 frees, 80,105 bytes allocated
==8316== All heap blocks were freed -- no leaks are possible
==8316== For lists of detected and suppressed errors, rerun with: -s
==8316== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

B. Helgrind (data-race detector)

```
valgrind --tool=helgrind ./build/booking_tests --filter="Concurrency"
```

Expected output snippet:

```
==8428== Helgrind, a thread error detector
==8428== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==8428== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==8428== Command: ./build/booking_tests --filter=Concurrency
==8428==
Seat already booked: A11
Seat already booked: A9
Seat already booked: A20
Seat already booked: A6
Seat already booked: A5
Seat already booked: A17
Seat already booked: A16
Seat already booked: A8
Seat already booked: A5
Seat already booked: A17
Seat already booked: A19
Seat already booked: A18
Seat already booked: A15
Seat already booked: A1
Seat already booked: A4
Seat already booked: A10
Seat already booked: A14
Seat already booked: A3
Seat already booked: A7
Seat already booked: A13
Seat already booked: A11
Seat already booked: A9
Seat already booked: A20
Seat already booked: A6
Seat already booked: A2
Seat already booked: A12
Seat already booked: A16
Seat already booked: A8
Seat already booked: A2
Seat already booked: A12
Threads: 50 | Successful: 20 | Remaining: 0
Concurrency stress test: configurable thread count PASSED

Executed: 1 | Failed: 0
==8428== Use --history-level=approx or =none to gain increased speed, at
==8428== the cost of reduced accuracy of conflicting-access information
==8428== For lists of detected and suppressed errors, rerun with: -s
==8428== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 343 from7
```

If any data race is detected, Helgrind will point to a specific variable or line (e.g., misuse of std::mutex or missing lock on shows_).

C. Callgrind (performance profiling)

```
valgrind --tool=callgrind ./build/booking_tests --filter="Concurrency"
callgrind_annotate callgrind.out.<pid> | less
```

You can see time spent in:

std::mutex::lock
BookingService::bookSeats
std::unordered_map::find

This helps you show that:

Global lock (shared_mutex) is held minimally.
Most time is in show->mtx, confirming good lock granularity.

D. Cachegrind (CPU efficiency):

To check that your helper functions (like seat parsing) are branch-predictable:

```
valgrind --tool=cachegrind ./build/booking_tests --filter="Concurrency"
cg_annotate cachegrind.out.<pid> | less
```

Look for high L1 hit rate and low branch misprediction.

5. CLI Demonstration Steps (Interactive)

```
./booking_cli
```

Walkthrough

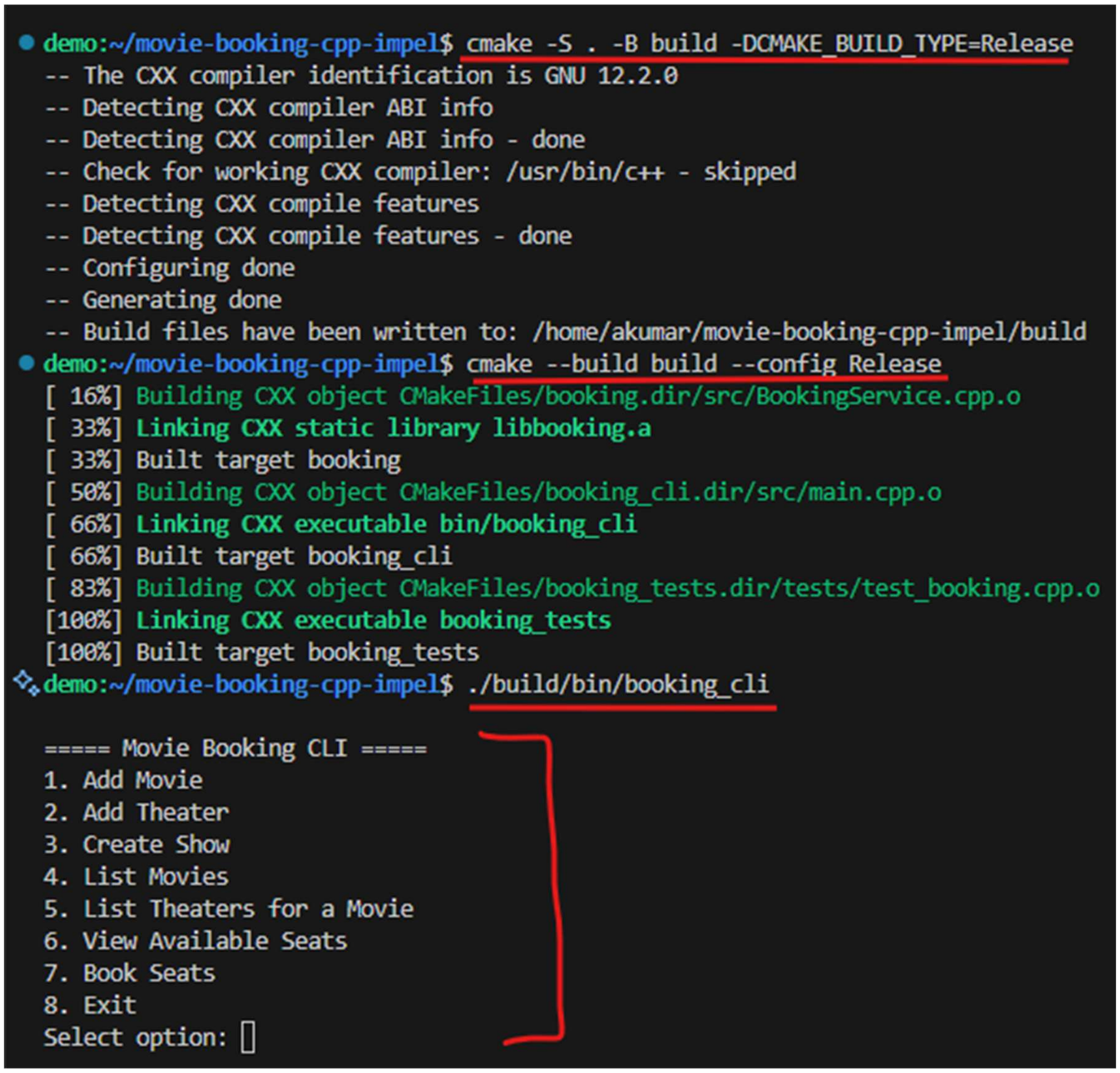
Choose 1. Add Movie → Enter *Rockstar*
Choose 1. Add Movie → Enter *Inception*
Choose 2. Add Theater → Enter *PVR Downtown*
Choose 3. Create Show → Pick movie [1] Rockstar, theater [1] PVR Downtown
Choose 4. List active movies (associated with shows) → [1] Rockstar
Choose 5. List theaters for a selected active movie. → (displays) [1] Rockstar → Enter [1] → [1] PVR Downtown
Choose 6. View Available Seats (for all the shows) → Shows 20 free seats
Choose 7. Book Seats (for the selected show)→ Enter [1] -> Enter A1,A2
Choose 6. View Available Seats again → 18 seats left + booked list
Repeat 7 with other seat combinations to verify blocking logic.

Press Ctrl+C anytime → the custom SIGINT handler prints:

```
Received Ctrl+C — shutting down gracefully...
Program terminated cleanly
```

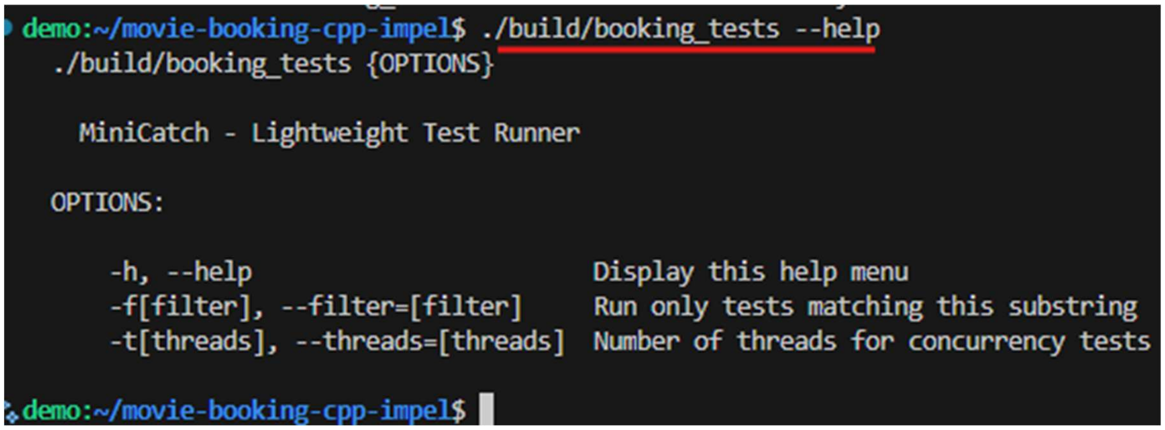
SCREENSHOTS OF THE EXECUTIONS

BUILD & RUN

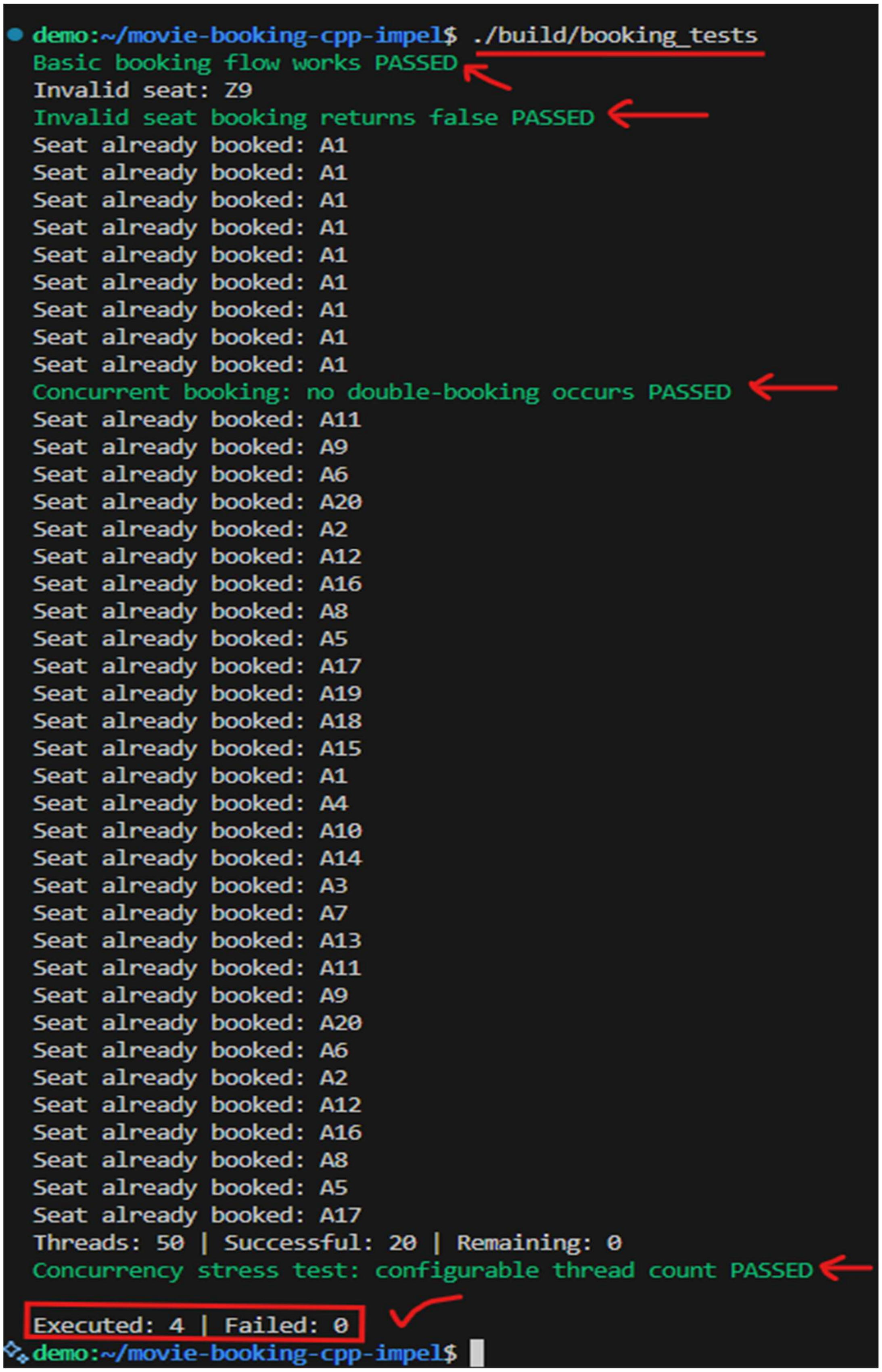


```
● demo:~/movie-booking-cpp-impel$ cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
  -- The CXX compiler identification is GNU 12.2.0
  -- Detecting CXX compiler ABI info
  -- Detecting CXX compiler ABI info - done
  -- Check for working CXX compiler: /usr/bin/c++ - skipped
  -- Detecting CXX compile features
  -- Detecting CXX compile features - done
  -- Configuring done
  -- Generating done
  -- Build files have been written to: /home/akumar/movie-booking-cpp-impel/build
● demo:~/movie-booking-cpp-impel$ cmake --build build --config Release
  [ 16%] Building CXX object CMakeFiles/booking.dir/src/BookingService.cpp.o
  [ 33%] Linking CXX static library libbooking.a
  [ 33%] Built target booking
  [ 50%] Building CXX object CMakeFiles/booking_cli.dir/src/main.cpp.o
  [ 66%] Linking CXX executable bin/booking_cli
  [ 66%] Built target booking_cli
  [ 83%] Building CXX object CMakeFiles/booking_tests.dir/tests/test_booking.cpp.o
  [100%] Linking CXX executable booking_tests
  [100%] Built target booking_tests
  demo:~/movie-booking-cpp-impel$ ./build/bin/booking_cli

  ===== Movie Booking CLI =====
  1. Add Movie
  2. Add Theater
  3. Create Show
  4. List Movies
  5. List Theaters for a Movie
  6. View Available Seats
  7. Book Seats
  8. Exit
  Select option:
```

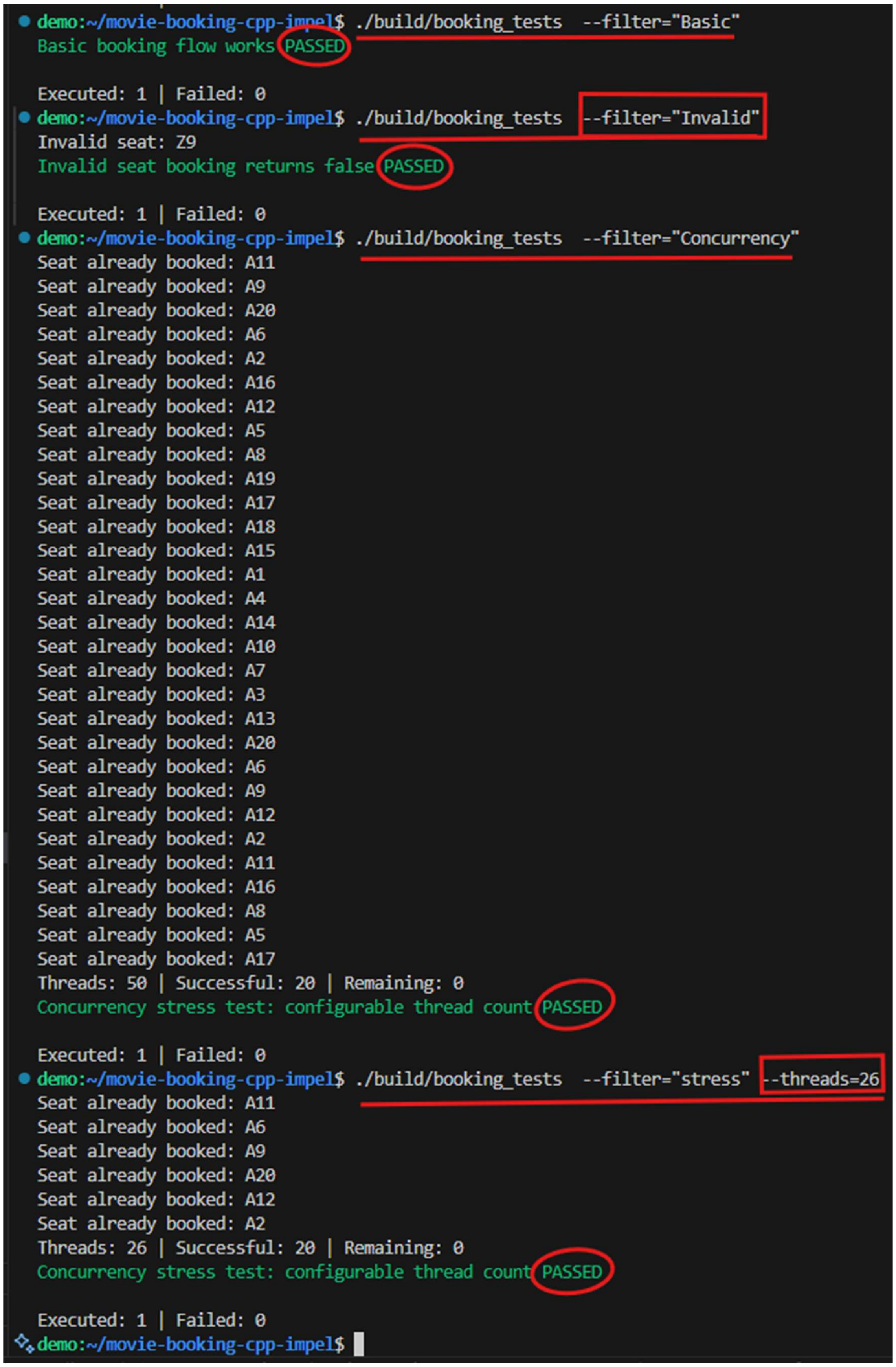
RUN UNIT TESTS*:*



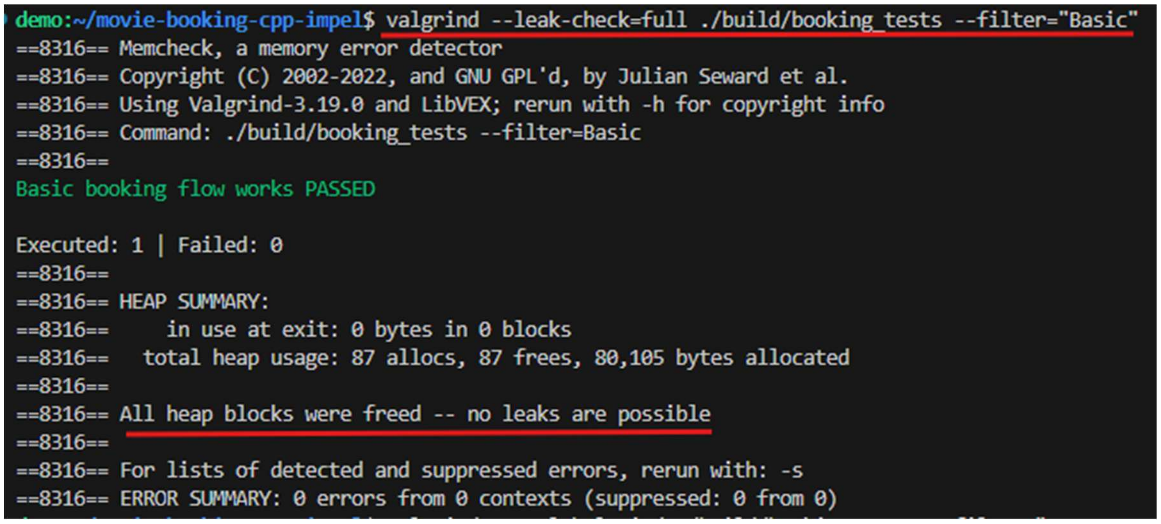
```
demo:~/movie-booking-cpp-impel$ ./build/booking_tests --help
  ./build/booking_tests {OPTIONS}

    MiniCatch - Lightweight Test Runner

  OPTIONS:

      -h, --help                         Display this help menu
      -f[filter], --filter=[filter]      Run only tests matching this substring
      -t[threads], --threads=[threads]   Number of threads for concurrency tests

demo:~/movie-booking-cpp-impel$
```



```
demo:~/movie-booking-cpp-impel$ ./build/booking_tests
Basic booking flow works PASSED
Invalid seat: Z9
Invalid seat booking returns false PASSED
Seat already booked: A1
Seat already booked: A1
Seat already booked: A1
Seat already booked: A1
Seat already booked: A1
Seat already booked: A1
Seat already booked: A1
Seat already booked: A1
Seat already booked: A1
Concurrent booking: no double-booking occurs PASSED
Seat already booked: A11
Seat already booked: A9
Seat already booked: A6
Seat already booked: A20
Seat already booked: A2
Seat already booked: A12
Seat already booked: A16
Seat already booked: A8
Seat already booked: A5
Seat already booked: A17
Seat already booked: A19
Seat already booked: A18
Seat already booked: A15
Seat already booked: A1
Seat already booked: A4
Seat already booked: A10
Seat already booked: A14
Seat already booked: A3
Seat already booked: A7
Seat already booked: A13
Seat already booked: A11
Seat already booked: A9
Seat already booked: A20
Seat already booked: A6
Seat already booked: A2
Seat already booked: A12
Seat already booked: A16
Seat already booked: A8
Seat already booked: A5
Seat already booked: A17
Threads: 50 | Successful: 20 | Remaining: 0
Concurrency stress test: configurable thread count PASSED

Executed: 4 | Failed: 0
demo:~/movie-booking-cpp-impel$
```

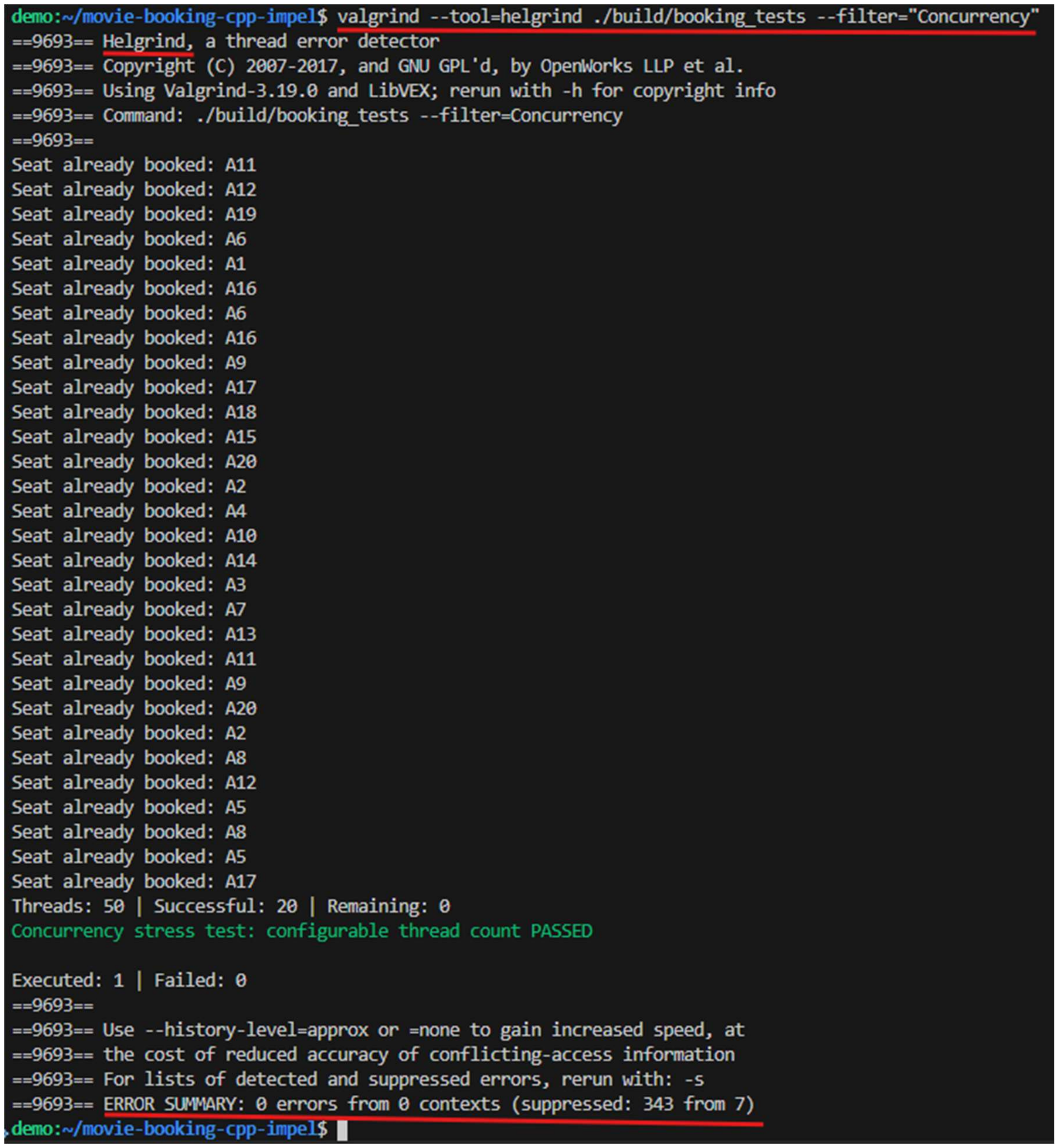

```
● demo:~/movie-booking-cpp-impel$ ./build/booking_tests  --filter="Basic"
  Basic booking flow works PASSED

  Executed: 1 | Failed: 0
● demo:~/movie-booking-cpp-impel$ ./build/booking_tests  --filter="Invalid"
  Invalid seat: Z9
  Invalid seat booking returns false PASSED

  Executed: 1 | Failed: 0
● demo:~/movie-booking-cpp-impel$ ./build/booking_tests  --filter="Concurrency"
  Seat already booked: A11
  Seat already booked: A9
  Seat already booked: A20
  Seat already booked: A6
  Seat already booked: A2
  Seat already booked: A16
  Seat already booked: A12
  Seat already booked: A5
  Seat already booked: A8
  Seat already booked: A19
  Seat already booked: A17
  Seat already booked: A18
  Seat already booked: A15
  Seat already booked: A1
  Seat already booked: A4
  Seat already booked: A14
  Seat already booked: A10
  Seat already booked: A7
  Seat already booked: A3
  Seat already booked: A13
  Seat already booked: A20
  Seat already booked: A6
  Seat already booked: A9
  Seat already booked: A12
  Seat already booked: A2
  Seat already booked: A11
  Seat already booked: A16
  Seat already booked: A8
  Seat already booked: A5
  Seat already booked: A17
  Threads: 50 | Successful: 20 | Remaining: 0
  Concurrency stress test: configurable thread count PASSED

  Executed: 1 | Failed: 0
● demo:~/movie-booking-cpp-impel$ ./build/booking_tests  --filter="stress" --threads=26
  Seat already booked: A11
  Seat already booked: A6
  Seat already booked: A9
  Seat already booked: A20
  Seat already booked: A12
  Seat already booked: A2
  Threads: 26 | Successful: 20 | Remaining: 0
  Concurrency stress test: configurable thread count PASSED

  Executed: 1 | Failed: 0
demo:~/movie-booking-cpp-impel$
```

RUN Valgrind –leak-check



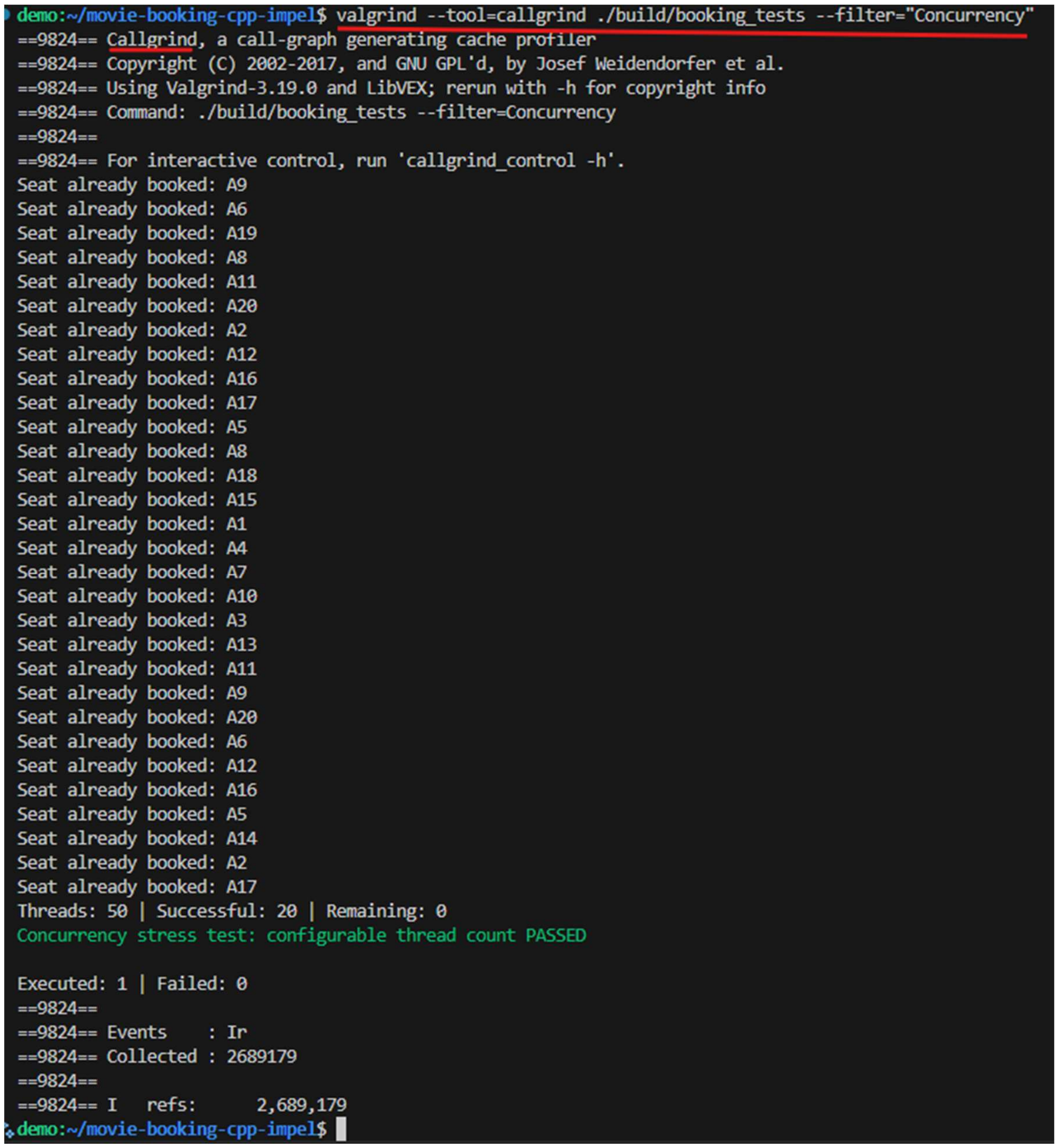
```
demo:~/movie-booking-cpp-impel$ valgrind --leak-check=full ./build/booking_tests --filter="Basic"
==8316== Memcheck, a memory error detector
==8316== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==8316== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==8316== Command: ./build/booking_tests --filter=Basic
==8316==
Basic booking flow works PASSED

Executed: 1 | Failed: 0
==8316==
==8316== HEAP SUMMARY:
==8316==     in use at exit: 0 bytes in 0 blocks
==8316==   total heap usage: 87 allocs, 87 frees, 80,105 bytes allocated
==8316==
==8316== All heap blocks were freed -- no leaks are possible
==8316==
==8316== For lists of detected and suppressed errors, rerun with: -s
==8316== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

RUN Valgrind –Helgrind (a thread error detector)



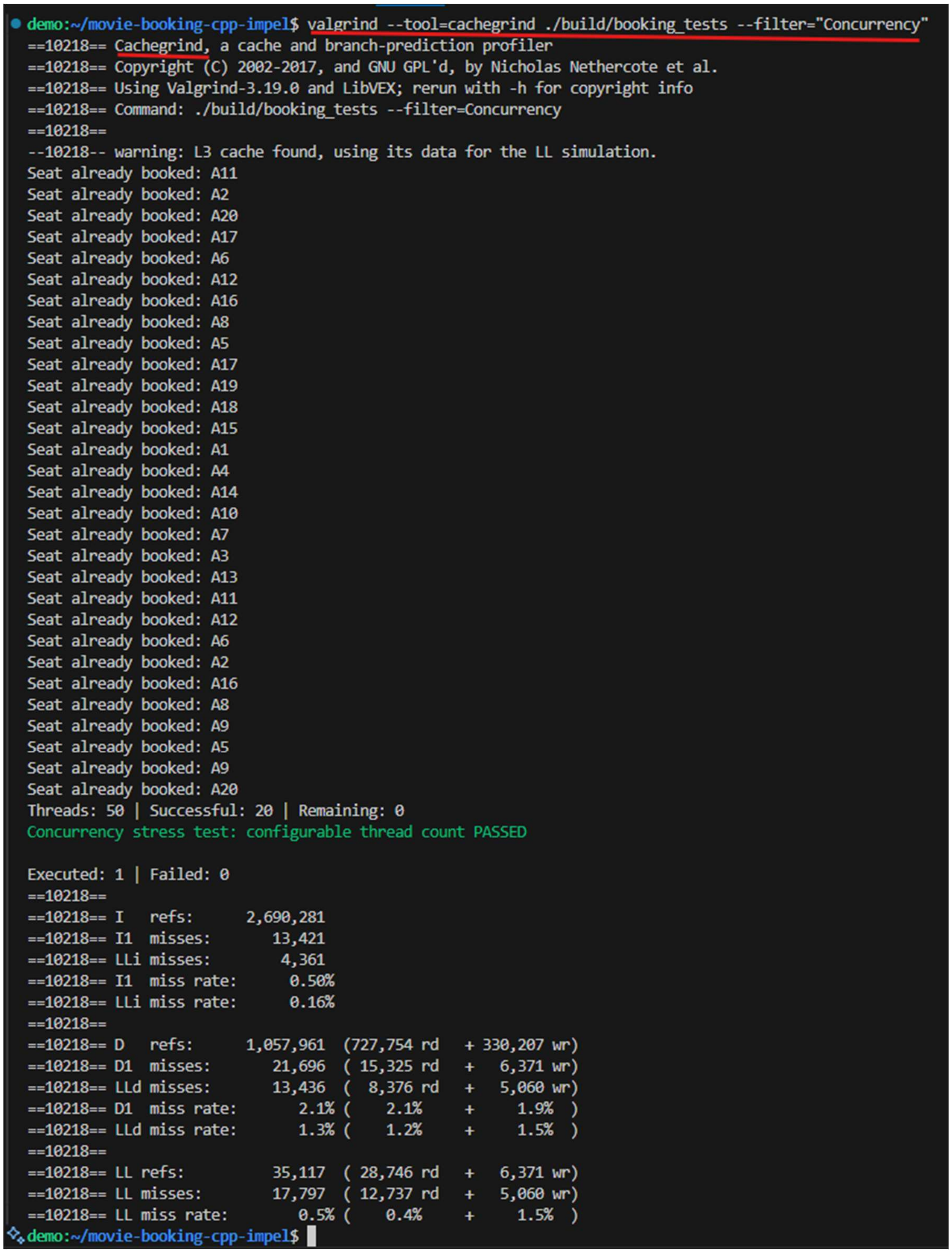
```
demo:~/movie-booking-cpp-impel$ valgrind --tool=helgrind ./build/booking_tests --filter="Concurrency"
==9693== Helgrind, a thread error detector
==9693== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==9693== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==9693== Command: ./build/booking_tests --filter=Concurrency
==9693==
Seat already booked: A11
Seat already booked: A12
Seat already booked: A19
Seat already booked: A6
Seat already booked: A1
Seat already booked: A16
Seat already booked: A6
Seat already booked: A16
Seat already booked: A9
Seat already booked: A17
Seat already booked: A18
Seat already booked: A15
Seat already booked: A20
Seat already booked: A2
Seat already booked: A4
Seat already booked: A10
Seat already booked: A14
Seat already booked: A3
Seat already booked: A7
Seat already booked: A13
Seat already booked: A11
Seat already booked: A9
Seat already booked: A20
Seat already booked: A2
Seat already booked: A8
Seat already booked: A12
Seat already booked: A5
Seat already booked: A8
Seat already booked: A5
Seat already booked: A17
Threads: 50 | Successful: 20 | Remaining: 0
Concurrency stress test: configurable thread count PASSED

Executed: 1 | Failed: 0
==9693==
==9693== Use --history-level=approx or =none to gain increased speed, at
==9693== the cost of reduced accuracy of conflicting-access information
==9693== For lists of detected and suppressed errors, rerun with: -s
==9693== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 343 from 7)
demo:~/movie-booking-cpp-impel$
```

RUN Valgrind –Callgrind (Call graph generating cache profiler)



```
demo:~/movie-booking-cpp-impel$ valgrind --tool=callgrind ./build/booking_tests --filter="Concurrency"
==9824== Callgrind, a call-graph generating cache profiler
==9824== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==9824== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==9824== Command: ./build/booking_tests --filter=Concurrency
==9824==
==9824== For interactive control, run 'callgrind_control -h'.
Seat already booked: A9
Seat already booked: A6
Seat already booked: A19
Seat already booked: A8
Seat already booked: A11
Seat already booked: A20
Seat already booked: A2
Seat already booked: A12
Seat already booked: A16
Seat already booked: A17
Seat already booked: A5
Seat already booked: A8
Seat already booked: A18
Seat already booked: A15
Seat already booked: A1
Seat already booked: A4
Seat already booked: A7
Seat already booked: A10
Seat already booked: A3
Seat already booked: A13
Seat already booked: A11
Seat already booked: A9
Seat already booked: A20
Seat already booked: A6
Seat already booked: A12
Seat already booked: A16
Seat already booked: A5
Seat already booked: A14
Seat already booked: A2
Seat already booked: A17
Threads: 50 | Successful: 20 | Remaining: 0
Concurrency stress test: configurable thread count PASSED

Executed: 1 | Failed: 0
==9824==
==9824== Events    : Ir
==9824== Collected : 2689179
==9824==
==9824== I   refs:      2,689,179
demo:~/movie-booking-cpp-impel$
```

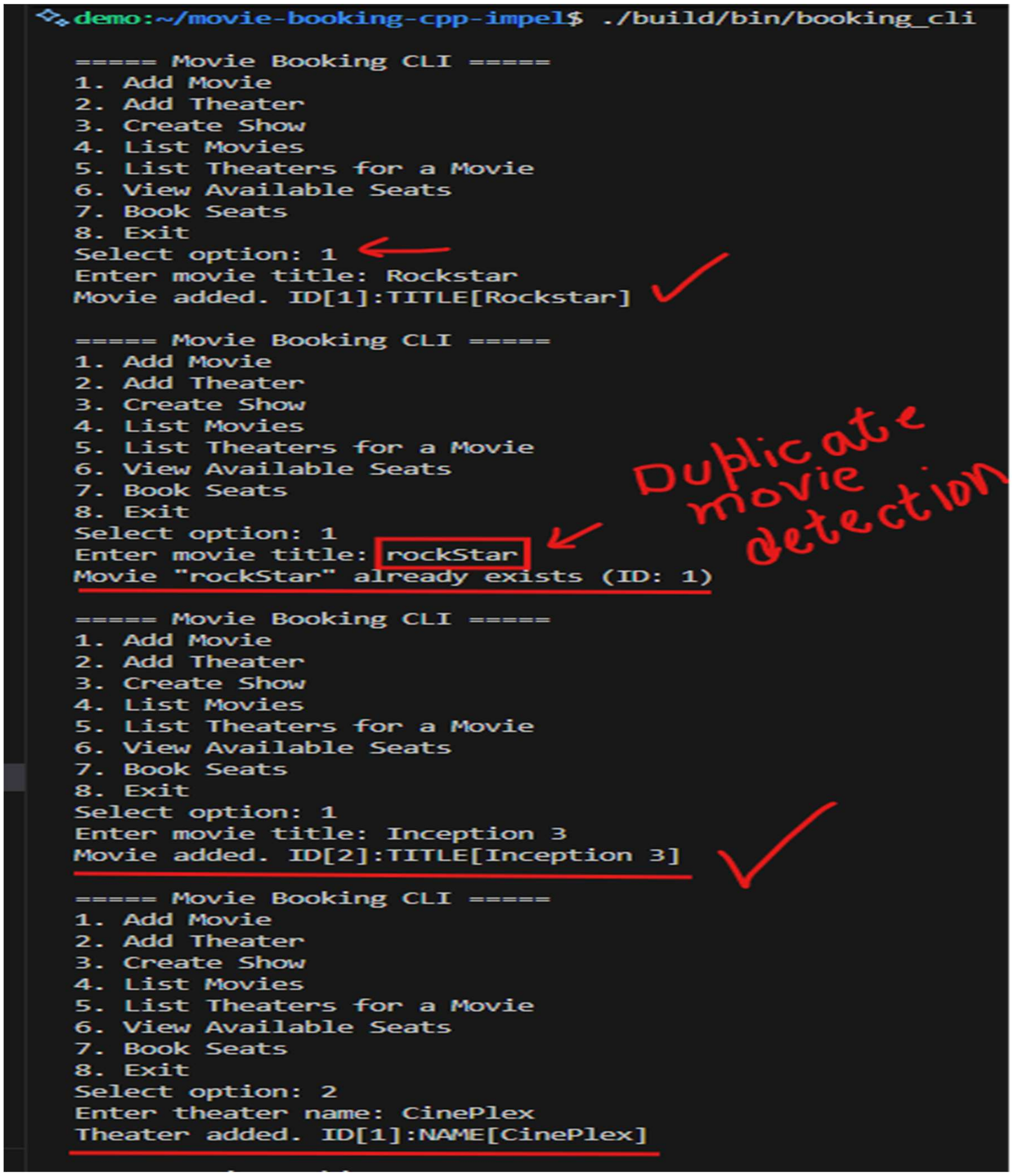
RUN Valgrind –Cachegrind (A cache and branch prediction profiler)



```
demo:~/movie-booking-cpp-impel$ valgrind --tool=cachegrind ./build/booking_tests --filter="Concurrency"
==10218== Cachegrind, a cache and branch-prediction profiler
==10218== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10218== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==10218== Command: ./build/booking_tests --filter=Concurrency
==10218==
--10218-- warning: L3 cache found, using its data for the LL simulation.
Seat already booked: A11
Seat already booked: A2
Seat already booked: A20
Seat already booked: A17
Seat already booked: A6
Seat already booked: A12
Seat already booked: A16
Seat already booked: A8
Seat already booked: A5
Seat already booked: A17
Seat already booked: A19
Seat already booked: A18
Seat already booked: A15
Seat already booked: A1
Seat already booked: A4
Seat already booked: A14
Seat already booked: A10
Seat already booked: A7
Seat already booked: A3
Seat already booked: A13
Seat already booked: A11
Seat already booked: A12
Seat already booked: A6
Seat already booked: A2
Seat already booked: A16
Seat already booked: A8
Seat already booked: A9
Seat already booked: A5
Seat already booked: A9
Seat already booked: A20
Threads: 50 | Successful: 20 | Remaining: 0
Concurrency stress test: configurable thread count PASSED

Executed: 1 | Failed: 0
==10218==
==10218== I   refs:      2,690,281
==10218== I1  misses:       13,421
==10218== LLi misses:        4,361
==10218== I1  miss rate:      0.50%
==10218== LLi miss rate:      0.16%
==10218==
==10218== D   refs:      1,057,961  (727,754 rd   + 330,207 wr)
==10218== D1  misses:       21,696  ( 15,325 rd   +   6,371 wr)
==10218== LLd misses:       13,436  (  8,376 rd   +   5,060 wr)
==10218== D1  miss rate:       2.1% (    2.1%     +     1.9%  )
==10218== LLd miss rate:       1.3% (    1.2%     +     1.5%  )
==10218==
==10218== LL refs:          35,117  ( 28,746 rd   +   6,371 wr)
==10218== LL misses:        17,797  ( 12,737 rd   +   5,060 wr)
==10218== LL miss rate:        0.5% (    0.4%     +     1.5%  )
demo:~/movie-booking-cpp-impel$
```

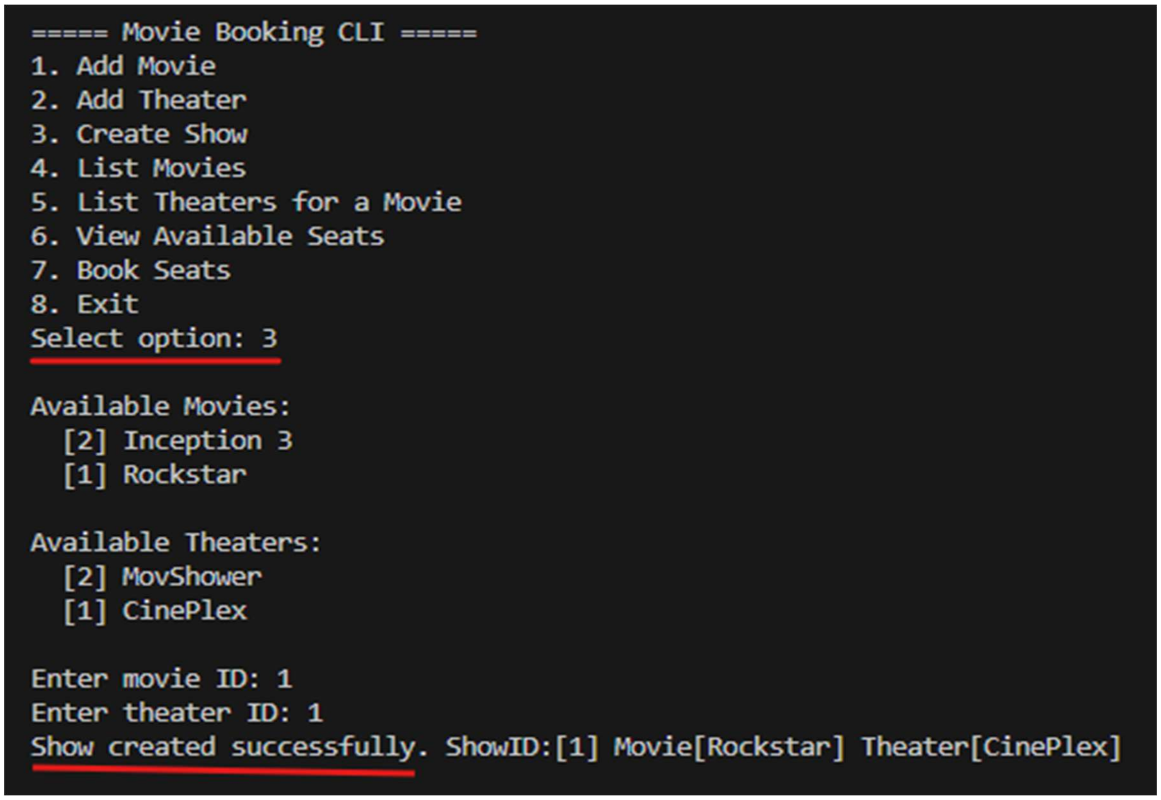
RUN CLI OPTIONS:

MOVIE AND THEATER ADDITION



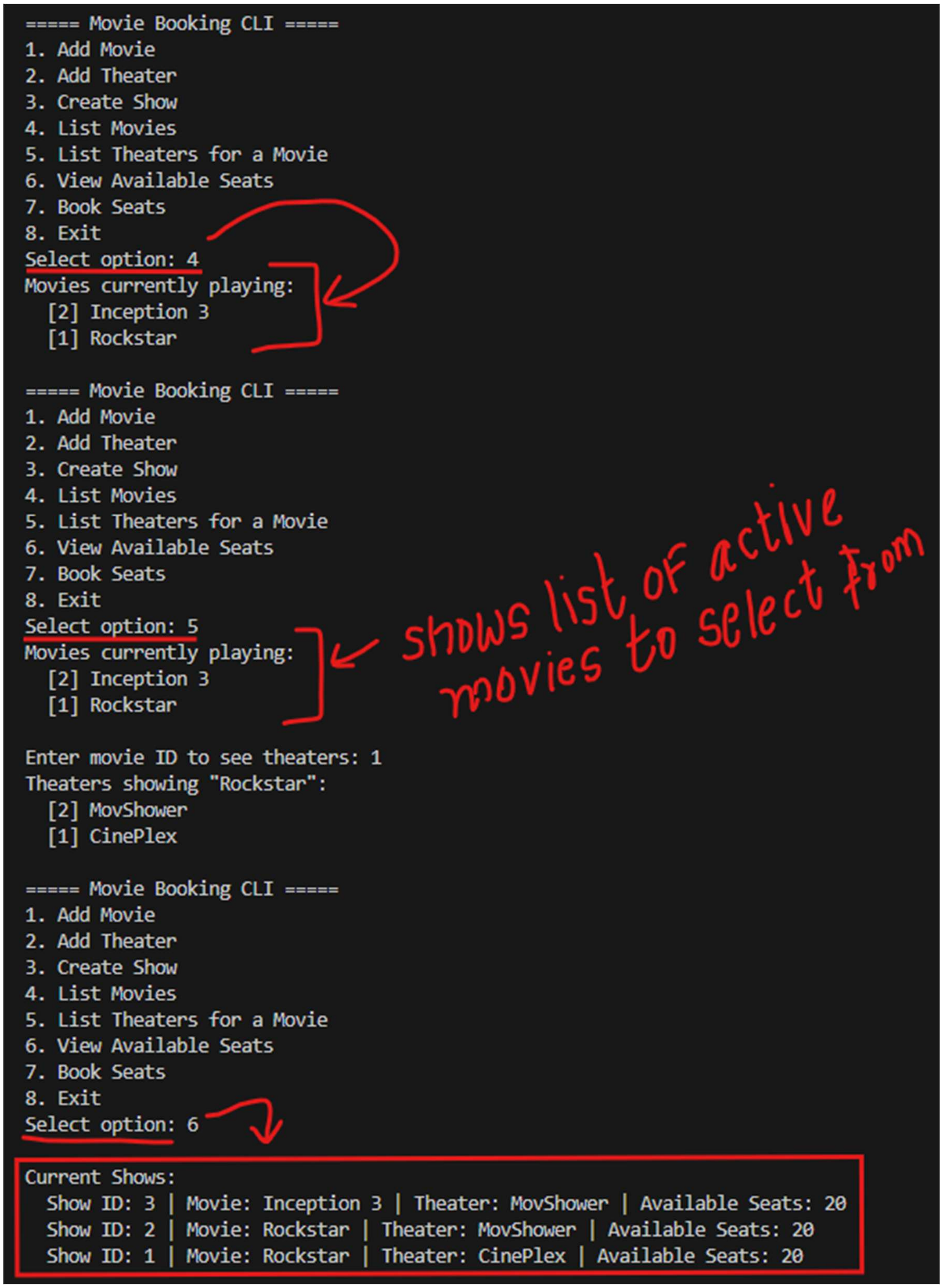
```
demo:~/movie-booking-cpp-impel$ ./build/bin/booking_cli

    ===== Movie Booking CLI =====
    1. Add Movie
    2. Add Theater
    3. Create Show
    4. List Movies
    5. List Theaters for a Movie
    6. View Available Seats
    7. Book Seats
    8. Exit
    Select option: 1
    Enter movie title: Rockstar
    Movie added. ID[1]:TITLE[Rockstar]

    ===== Movie Booking CLI =====
    1. Add Movie
    2. Add Theater
    3. Create Show
    4. List Movies
    5. List Theaters for a Movie
    6. View Available Seats
    7. Book Seats
    8. Exit
    Select option: 1
    Enter movie title: rockStar
    Movie "rockStar" already exists (ID: 1)

    ===== Movie Booking CLI =====
    1. Add Movie
    2. Add Theater
    3. Create Show
    4. List Movies
    5. List Theaters for a Movie
    6. View Available Seats
    7. Book Seats
    8. Exit
    Select option: 1
    Enter movie title: Inception 3
    Movie added. ID[2]:TITLE[Inception 3]

    ===== Movie Booking CLI =====
    1. Add Movie
    2. Add Theater
    3. Create Show
    4. List Movies
    5. List Theaters for a Movie
    6. View Available Seats
    7. Book Seats
    8. Exit
    Select option: 2
    Enter theater name: CinePlex
    Theater added. ID[1]:NAME[CinePlex]
```

SHOW CREATION (MOVIE ←→ THEATER)



```
===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 3

Available Movies:
  [2] Inception 3
  [1] Rockstar

Available Theaters:
  [2] MovShower
  [1] CinePlex

Enter movie ID: 1
Enter theater ID: 1
Show created successfully. ShowID:[1] Movie[Rockstar] Theater[CinePlex]
```

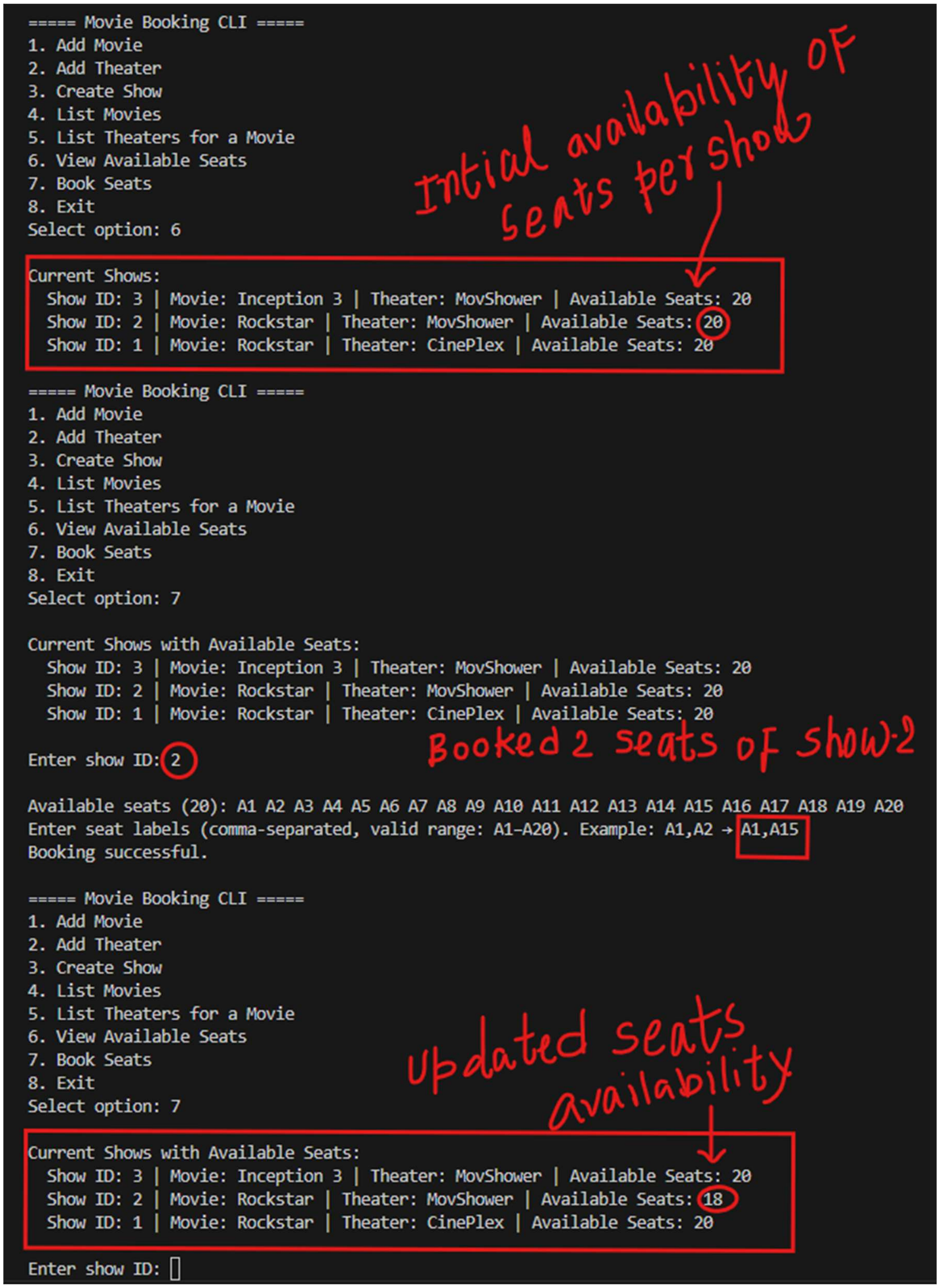
SEATS AVAILABILITY IS DISPLAYED PER SHOW FOR ALL THE SHOWS



```
===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 4
Movies currently playing:
  [2] Inception 3
  [1] Rockstar

===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 5
Movies currently playing:
  [2] Inception 3
  [1] Rockstar

Enter movie ID to see theaters: 1
Theaters showing "Rockstar":
  [2] MovShower
  [1] CinePlex

===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 6

Current Shows:
  Show ID: 3 | Movie: Inception 3 | Theater: MovShower | Available Seats: 20
  Show ID: 2 | Movie: Rockstar | Theater: MovShower | Available Seats: 20
  Show ID: 1 | Movie: Rockstar | Theater: CinePlex | Available Seats: 20
```

shows list of active movies to select from

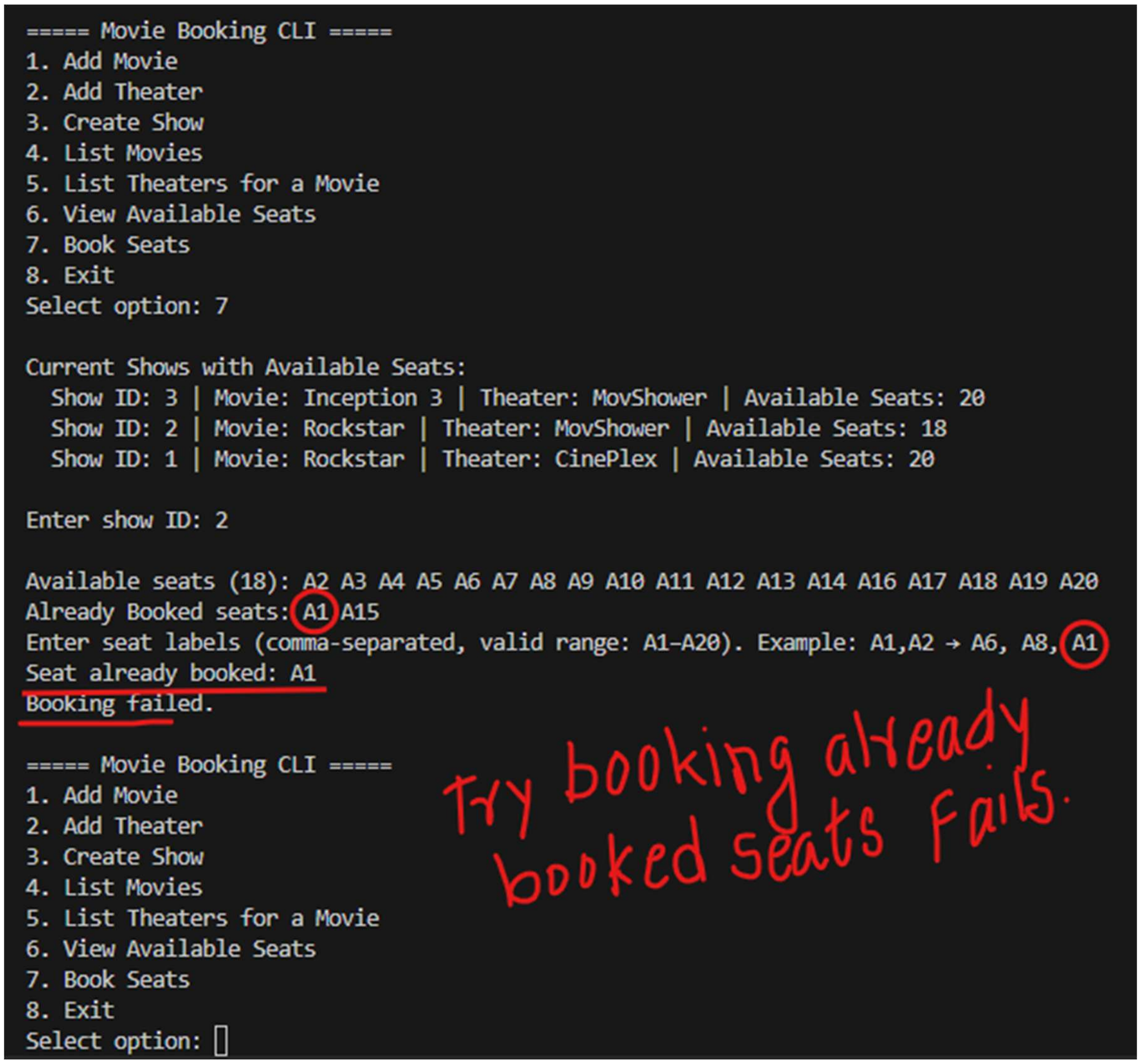
SEATS AVAILABILITY GETS UPDATED AFTER SUCCESSFUL BOOKING



```
===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 6

Current Shows:
  Show ID: 3 | Movie: Inception 3 | Theater: MovShower | Available Seats: 20
  Show ID: 2 | Movie: Rockstar | Theater: MovShower | Available Seats: 20
  Show ID: 1 | Movie: Rockstar | Theater: CinePlex | Available Seats: 20

===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 7

Current Shows with Available Seats:
  Show ID: 3 | Movie: Inception 3 | Theater: MovShower | Available Seats: 20
  Show ID: 2 | Movie: Rockstar | Theater: MovShower | Available Seats: 20
  Show ID: 1 | Movie: Rockstar | Theater: CinePlex | Available Seats: 20

Enter show ID: 2

Available seats (20): A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 A16 A17 A18 A19 A20
Enter seat labels (comma-separated, valid range: A1-A20). Example: A1,A2 → A1,A15
Booking successful.

===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 7

Current Shows with Available Seats:
  Show ID: 3 | Movie: Inception 3 | Theater: MovShower | Available Seats: 20
  Show ID: 2 | Movie: Rockstar | Theater: MovShower | Available Seats: 18
  Show ID: 1 | Movie: Rockstar | Theater: CinePlex | Available Seats: 20

Enter show ID: 
```

Intial availability of seats per show
Booked 2 seats of show-2
updated seats availability

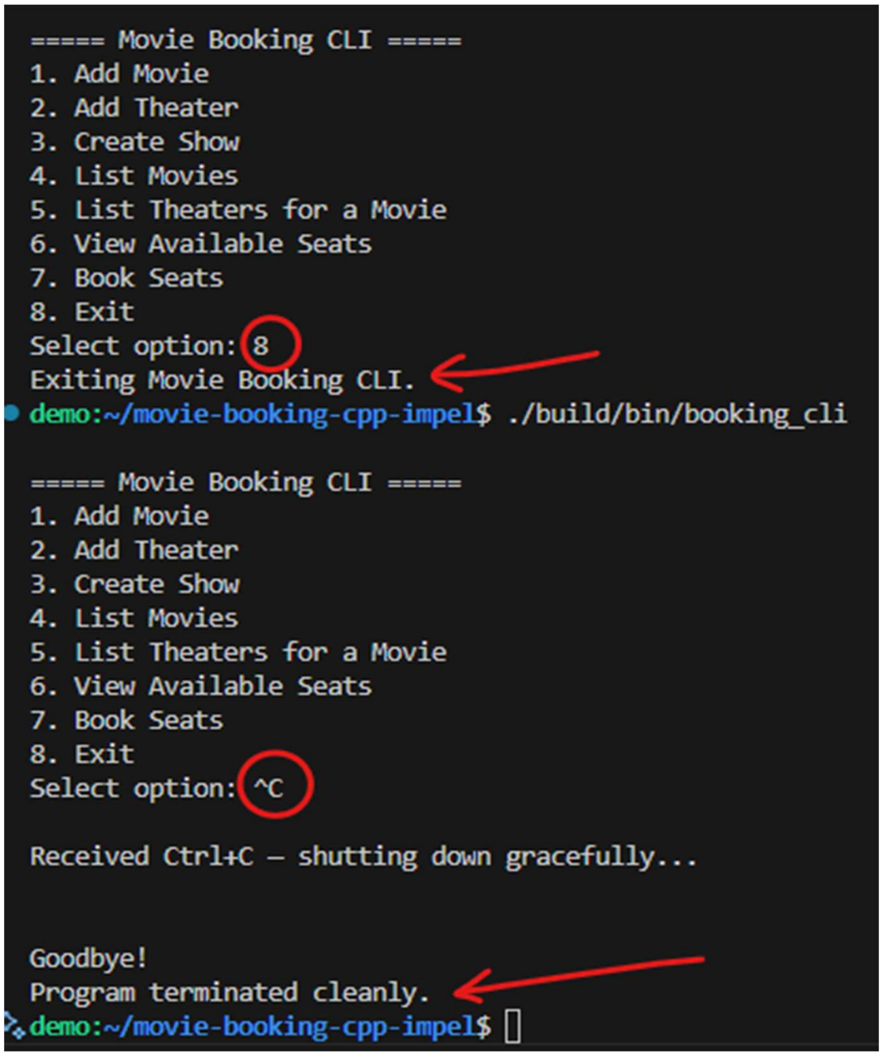
REQUEST TO BOOK ALREADY BOOKED SEAT(S) GETS DENIED.



```
===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 7

Current Shows with Available Seats:
  Show ID: 3 | Movie: Inception 3 | Theater: MovShower | Available Seats: 20
  Show ID: 2 | Movie: Rockstar | Theater: MovShower | Available Seats: 18
  Show ID: 1 | Movie: Rockstar | Theater: CinePlex | Available Seats: 20

Enter show ID: 2

Available seats (18): A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A16 A17 A18 A19 A20
Already Booked seats: A1 A15
Enter seat labels (comma-separated, valid range: A1-A20). Example: A1,A2 → A6, A8, A1
Seat already booked: A1
Booking failed.

===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 
```

Try booking already booked seats Fails.

EXITTING (MENU OPTION-8 OR CRTL-C)



```
===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: 8
Exiting Movie Booking CLI.
demo:~/movie-booking-cpp-impel$ ./build/bin/booking_cli

===== Movie Booking CLI =====
1. Add Movie
2. Add Theater
3. Create Show
4. List Movies
5. List Theaters for a Movie
6. View Available Seats
7. Book Seats
8. Exit
Select option: ^C

Received Ctrl+C — shutting down gracefully...


Goodbye!
Program terminated cleanly.
demo:~/movie-booking-cpp-impel$
```