CSCI 4220 Lab 4 — Multi-Client Multi-Threaded Fibonacci Service

Overview

In this lab, you will implement a **multi-client, multi-threaded TCP server** that performs **Fibonacci computations** concurrently.
Each connected client requests a positive integer `N`. The server divides the range `[1..N]` among several worker threads, each computing the **Fibonacci numbers** within its assigned subrange.
After all threads finish, the server aggregates the computed Fibonacci values to calculate the **sum of the first N Fibonacci numbers**, and sends a formatted report back to the client. You can check some example code in unpbook code folder threads/tcpserv01.c and tcpserv02.c and our lecture 7 slides to see how to create threads and join results.

This lab deepens your understanding of: - TCP socket programming - Two-level threading (client-handler + worker threads) - Per-thread computation state and result aggregation - Parallel decomposition of computational tasks

---

Learning Objectives

By the end of this lab, you will: - Combine networking and threading to serve multiple clients simultaneously. - Use per-thread data structures to compute and store partial results. - Divide a computational sequence into balanced ranges for parallel execution. - Aggregate per-thread results to produce a final total.

---

1. Server Overview

Run as:

```
./lab4 <port> [num_threads]
```

- Default `num_threads = 4`
- The server:
  1. Listens for incoming TCP connections.
  2. For each client, spawns a **client-handler thread**.
  3. Each handler requests an integer `N` and creates `num_threads` **worker threads**.
  4. Each worker computes the **Fibonacci numbers** for its assigned subrange `[start, end]`.
  5. The handler collects all results, sums them, and sends the report to the client.

Why subranges?

The Fibonacci sequence up to `N` can be computed in parallel by dividing the range of indices evenly across threads.
For example, if `N = 12` and there are 4 threads:

Thread	Range	Computed Values
1	[1-3]	1, 1, 2
2	[4-6]	3, 5, 8
3	[7-9]	13, 21, 34
4	[10-12]	55, 89, 144

Each thread computes its own subset of Fibonacci numbers independently (for example, thread 1 computes Fib(1), Fib(2) and Fib(3)), and the main thread aggregates their sums:

[Total Sum = Fib(1) + Fib(2) + … + Fib(N)]

This design demonstrates parallel task decomposition — a key idea in concurrent server computation.

---

2. Client Interaction

Example using netcat:

```
$ nc 127.0.0.1 9000
Connected from 52000. Please enter an integer N:
12
T1: [1-3] -> 1 1 2
T2: [4-6] -> 3 5 8
T3: [7-9] -> 13 21 34
T4: [10-12] -> 55 89 144
Total computed = 12 Fibonacci numbers
Sum = 375
```

Each client can connect concurrently. Each request runs independently in its own handler thread.

---

3. Thread Data Structures

Each worker thread uses:

```c
typedef struct {
    int tid;        // Thread index (1..T)
    int start;      // Start of Fibonacci range
    int end;        // End of Fibonacci range
    long sum;       // Sum of computed Fibonacci values
    char result[256]; // Formatted Fibonacci values
} fib_data_t;
```

The result field is a character array (string buffer) used to store all Fibonacci numbers computed by a thread, formatted as a space-separated string (e.g., “13 21 34”). This makes it easy for the server to print or send human-readable output for each thread:

```
T3: [7-9] -> 13 21 34 (sum=68)
```

- Each worker thread:
  1. Iterates from `start` to `end`
  2. Computes `Fib(k)` iteratively
  3. Stores each value in `result`
  4. Accumulates the sum for its subrange
  5. Returns the struct pointer via `pthread_exit()`

---

4. Fibonacci Computation

Use an **iterative** (not recursive) implementation to avoid exponential time:

```
long fib(int n) {
    if (n <= 2) return 1;
    long a = 1, b = 1, c;
    for (int i = 3; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}
```

---

5. Example Server Output

```
[INFO] Server listening on port 9000
[INFO] New client from port 52000
[Client 1] Received N = 12
[Client 1][Thread 1] Range [1-3]: sum=4
[Client 1][Thread 2] Range [4-6]: sum=16
[Client 1][Thread 3] Range [7-9]: sum=68
[Client 1][Thread 4] Range [10-12]: sum=287
[Client 1] Total sum = 375
```

Deliverables

Submit:

`lab4.c`

Your submission must:
- Implement a multi-client TCP server.
- Spawn per-client handler threads.
- Within each client handler, spawn worker threads to compute Fibonacci subranges.
- Aggregate partial sums and send the formatted result to the client.

Compile with:

```
gcc -o lab4 -lm lab4.c libunp.a
```

Academic Integrity and AI Code Warning

You must write your own implementation of the TCP server and threading logic.
All submissions will be automatically analyzed for **code similarity and AI-generated content**.
Plagiarism or unauthorized use of code generation tools (e.g., copying from another student or online source) will result in a violation report under the RPI academic integrity policy.

If you use AI tools for debugging or explanation purposes, you must clearly document this in your README and ensure that all submitted code reflects **your own understanding and implementation**.
