Concurrent Job Processing Server

This Assignment implements a **concurrent server-client system** in C++. The server can handle multiple clients (**multi-threaded, queue-based server**) simultaneously, using worker threads and a request queue to manage incoming jobs efficiently.

How it works:

1. **Server**
    - Listens on a TCP port for client connections.
    - Accepts multiple clients and pushes their requests into a queue.
    - Worker threads process jobs from the queue and send results back to clients.
    - Supports configurable options: queue type (static/dynamic/circular etc), queue capacity, number of workers, and port.

2. **Client**
    - Connects to the server.
    - Sends a job request with the following details:

        - Job type (count-words, extract-emails, csv-to-tsv, compress, decompress, base64-encode, base64-decode)
        - Filename
        - Payload/message
    - Receives the processed result from the server in JSON format.
    - Can run multiple clients concurrently to test the server's multi-threaded handling.

Supported Jobs:

- count-words — Counts words in text.
- extract-emails — Extracts emails from text.
- csv-to-tsv — Converts CSV data to TSV format.
- compress / decompress — Simple file compression/decompression.
- base64-encode / base64-decode — Base64 encoding and decoding.

Description of all the queue classes:

1. `IQueue<T>`

- Type: Interface / Abstract class
- Purpose: Defines the standard queue operations for all queue types.
- Key methods:
  - `enqueue()`, `try_enqueue()` – add elements
  - `dequeue()`, `try_dequeue()` – remove elements
  - `peek()` – view front element
  - `isEmpty()`, `getSize()`, `getCapacity()`

---

2. `BoundedBlockingQueue<T>`

- Type: Thread-safe, bounded queue
- Purpose: Allows multiple threads to safely enqueue/dequeue with blocking.
- Features:
  - Fixed capacity
  - Blocks on `enqueue()` if full, blocks on `dequeue()` if empty

- Uses `mutex` and `condition_variable` for thread synchronization

---

3. `CircularQueue<T>`

- Type: Fixed-size circular buffer
- Purpose: Implements a classic circular queue with wrap-around indexing.
- Features:
  - Fixed capacity
  - Thread-safe using `mutex`
  - Can `peek`, `enqueue`, `dequeue`, `try_enqueue`, `try_dequeue`
  - Wrap-around ensures efficient memory usage

---

4. `DynamicArrayQueue<T>`

- Type: Resizable array queue
- Purpose: Queue that grows dynamically when full.
- Features:
  - Doubles capacity automatically when needed
  - Thread-safe with `mutex`
  - Maintains circular buffer logic internally

- Ideal for queues where size is unpredictable

---

5. `StaticArrayQueue<T>`

- Type: Fixed-size static array queue
- Purpose: Simple queue with pre-allocated memory.
- Features:
  - Fixed capacity
  - Thread-safe using `mutex`
  - Simple wrap-around logic for front/rear
  - Fast, low-overhead queue for known maximum size

Summary:

- `IQueue<T>` defines the interface.
- `BoundedBlockingQueue<T>` is a thread-safe blocking queue.
- `CircularQueue<T>` is a fixed-size circular queue.
- `DynamicArrayQueue<T>` is a resizable array-based queue.
- `StaticArrayQueue<T>` is a fixed-size array queue with no dynamic resizing.
