Presented by Group 1

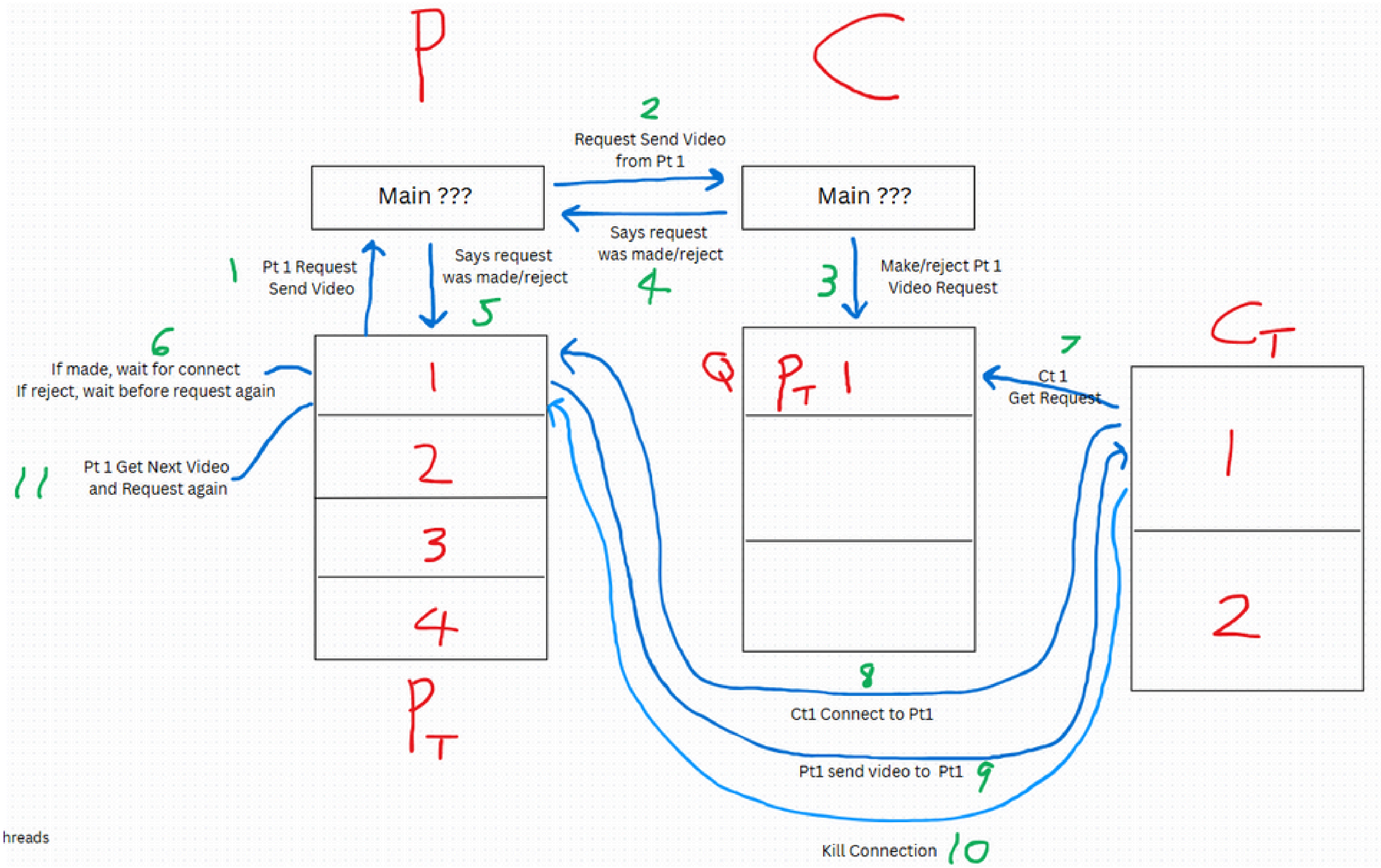
P3 - NETWORKED PRODUCER AND CONSUMER

Concio, Montenegro, Yu, Sia

Key Implementation Details

- Implemented **multi-threaded** Producer system where **each thread** handles **a folder of videos.**
- Used **TCP sockets** for communication between Producer and Consumer.
- Uses **VideoRequest** to represent requests containing: **port**, **video name**, and the **video's SHA256 hash.**
- Implemented **queue** with bounded capacity on Consumer.
- Introduced **hash-based deduplication** on the Consumer side to avoid duplicate downloads.
- Added support for **compression** before sending and **decompression** upon receiving. Uses GZip algorithm.
- Ensured **clean termination** by signaling when all producers are finished.

Initial Implementation Illustration


P
C
2
Request Send Video from Pt 1
Main ???
Main ???
Says request was made/reject
4
1
Pt 1 Request Send Video
Says request was made/reject
5
3
Make/reject Pt 1 Video Request
6
If made, wait for connect
If reject, wait before request again
1
2
3
4
Q
P_T 1
7
Ct 1 Get Request
C_T
1
2
11
Pt 1 Get Next Video and Request again
P_T
8
Ct1 Connect to Pt1
Pt1 send video to Pt1
9
Kill Connection
10
hreads

Producer and Consumer Concepts Applied

Producer

- Multi-threaded video sender
- Sends request, waits for acceptance, then transmits the video
- Retries on queue overflow, skips on deduplication

Consumer

- Handles requests, maintains queue
- Multiple threads pull from queue and connect back to receive videos
- Uses hash set to track downloaded videos

- Models bounded buffer problem using queue + deduplication
- Application of resource sharing, synchronization, and backpressure

Queueing details

- The consumer uses a **bounded queue** (**VideoRequestQueue**) with size **q** to hold and queue requests.
- The Producer Threads send video send requests to the consumer's queue and waits for its response.
  - If the video is a duplicate (based on SHA256 hash), Producer thread skips the video and goes to the next.
  - If the queue is full, the producer thread retires to send again after a delay.
  - Accepted requests are added to the queue and processed by the available Consumer threads.

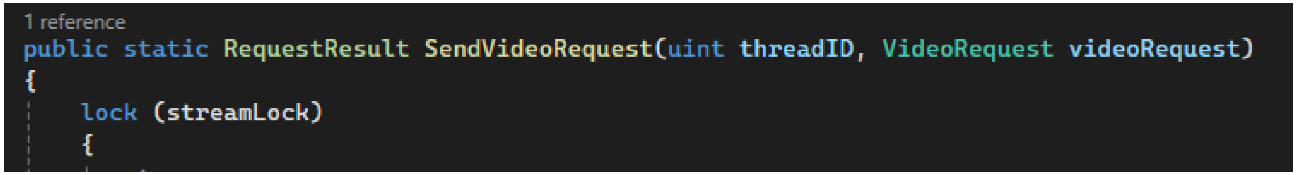
Connection details

- After the request is successfully put into the queue
    - The **requesting Producer Thread** will listen on their **port** and wait for a Consumer Thread's connection.
    - Then, an available **Consumer Thread** will **get the request** and connect to the **port** of the requesting **Producer Thread.**
- Once the threads connect, the Producer Thread will **compress** the video file and send it through the stream, while the Consumer Thread will accept the stream, **decompress** the file, and save it to the download folder
- Afterwards, the threads will disconnect and the request is removed
    - The **Producer Thread** will go to the **next video** and **request** again
    - The **Consumer Thread** will get the **next request** to process

Synchronization mechanisms

PRODUCER

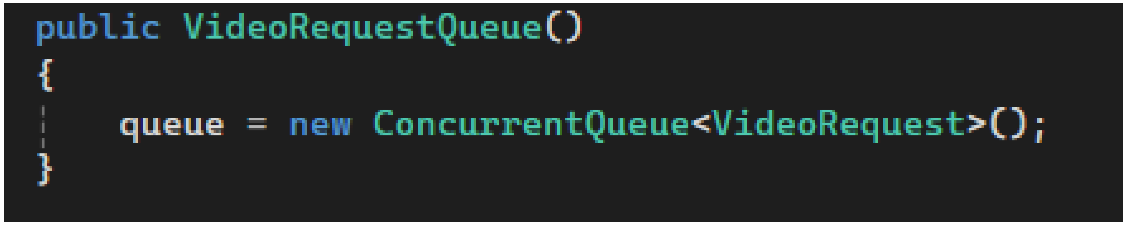
- Used lock **(lock(streamLock)**) to protect shared NetworkStream between producer threads



```
1 reference
public static RequestResult SendVideoRequest(uint threadID, VideoRequest videoRequest)
{
    lock (streamLock)
    {
```

CONSUMER

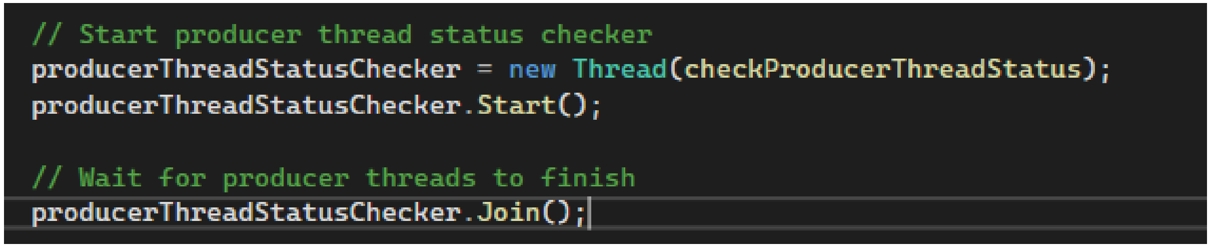
- Used **ConcurrentQueue<VideoRequest>** to manage shared access to the download queue between multiple Consumer threads.



```
public VideoRequestQueue()
{
    queue = new ConcurrentQueue<VideoRequest>();
}
```

Synchronization mechanisms

- Avoided simultaneous write/read to stream by:
  - Having only the **main thread** send **VideoRequests**
  - Ensuring each **ProducerThread** manages its own **listener/socket** for video data
- Ensured thread-safe termination by tracking all producer thread states in a **monitoring thread**



```
// Start producer thread status checker
producerThreadStatusChecker = new Thread(checkProducerThreadStatus);
producerThreadStatusChecker.Start();

// Wait for producer threads to finish
producerThreadStatusChecker.Join();
```

- No contention and deadlocks because thread responsibilities are isolated.

Bonus Features

NON-LEAKY BUCKET

The queue uses back pressure, so if it becomes full, it signals the requester to wait before sending another request.

DUPLICATION CHECK

When loading the application, it first saves the hash of all existing videos in the folder. When a new request comes in, it checks the video's hash with the rest of the hashes and rejects it if it's a duplicate. If it's unique, it saves the hash for future comparisons.

Bonus Features

VIDEO COMPRESSION

To save bandwidth, when the Producer Thread first compresses the video before sending it. When the Consumer Thread receives the data, it decompresses it and saves the video.

Thank You!

end of slides