Lab 3

Distributed Software Systems
UNIBO - 2024/25

Anouk Wuyts - 1900124167

Davide De Rosa - 1186536

Producer-Consumer System

A **Producer-Consumer system** is a concurrency model where two types of entities—*producers* and *consumers*—interact through a *shared buffer*. Producers generate data or tasks and place them into the buffer, while consumers retrieve and consume this data.

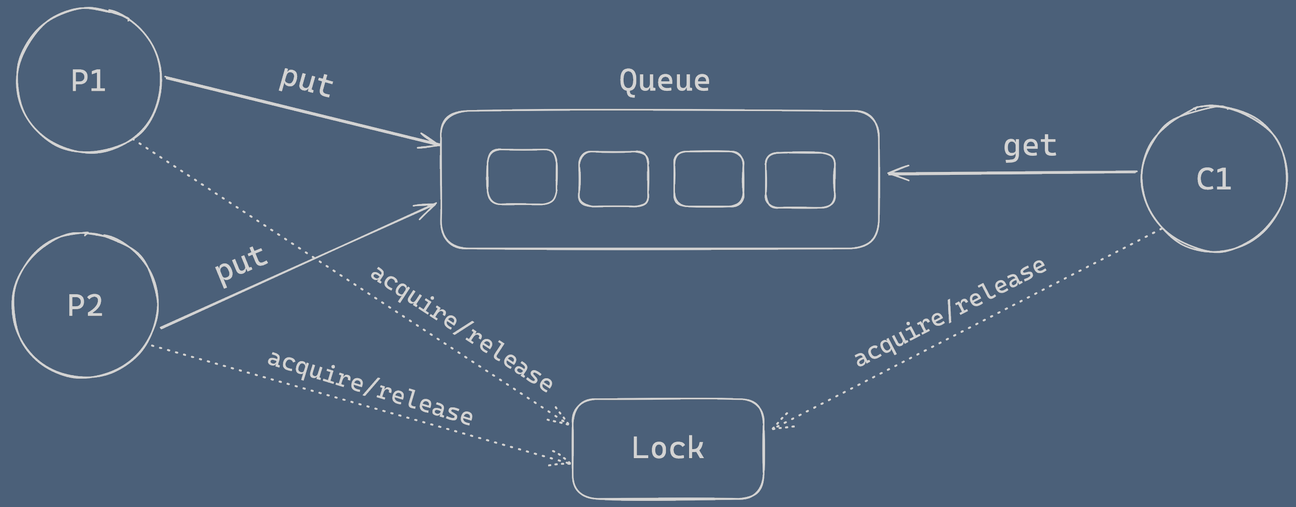
The system ensures **synchronization** between producers and consumers to prevent issues like overfilling the buffer or consuming from an empty one, often using mechanisms like **locks** or **semaphores**. This model is commonly used in multithreading and parallel processing to efficiently manage tasks and resources.

Producer-Consumer System

A simple **diagram** can make the Producer-Consumer architecture more clear.

The **Shared Queue** acts as a **buffer** between producers, which add tasks or data to the queue, and consumers, which retrieve and consume them.

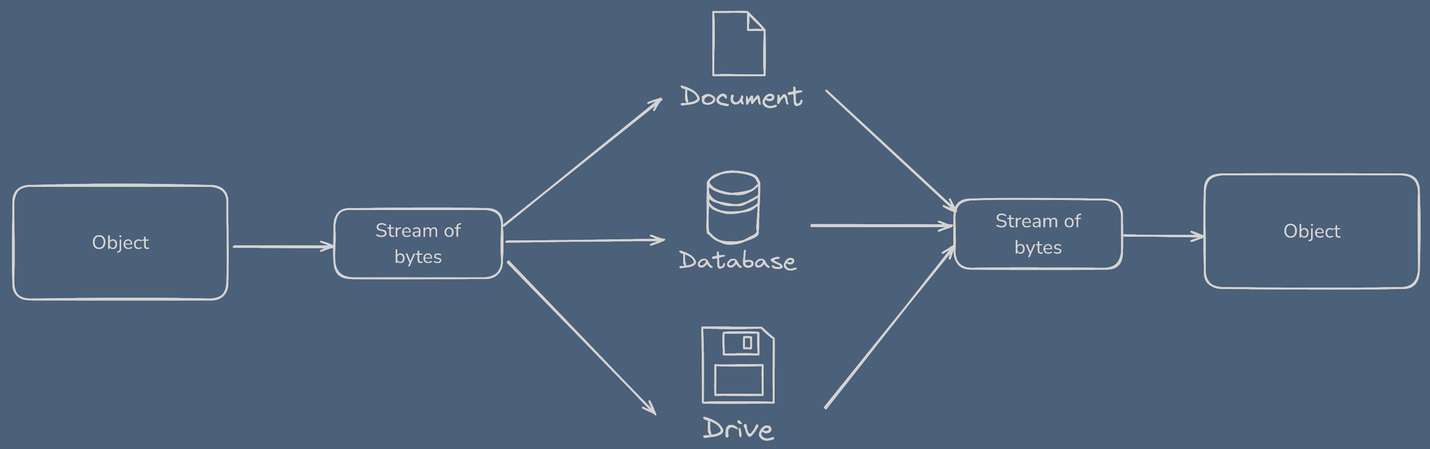
Since multiple entities access the shared queue, **synchronization mechanisms** (like locks or semaphores) are often needed to ensure safe and orderly access, preventing issues like data corruption or race conditions. The queue helps manage task flow and balance workloads in multithreaded environments.



JSON Serialization-Deserialization

Serialization (also known as encoding) refers to the process of converting an object into a JSON string, we did this by using json.dumps() in Python.

Meanwhile, **Deserialization** (or decoding) is the reverse: converting a JSON string back into an object and for this we used json.loads() in Python.



Producer-Consumer System with Threads

For our first implementation, we only used **Threads** and simulated **concurrency** on the same machine.

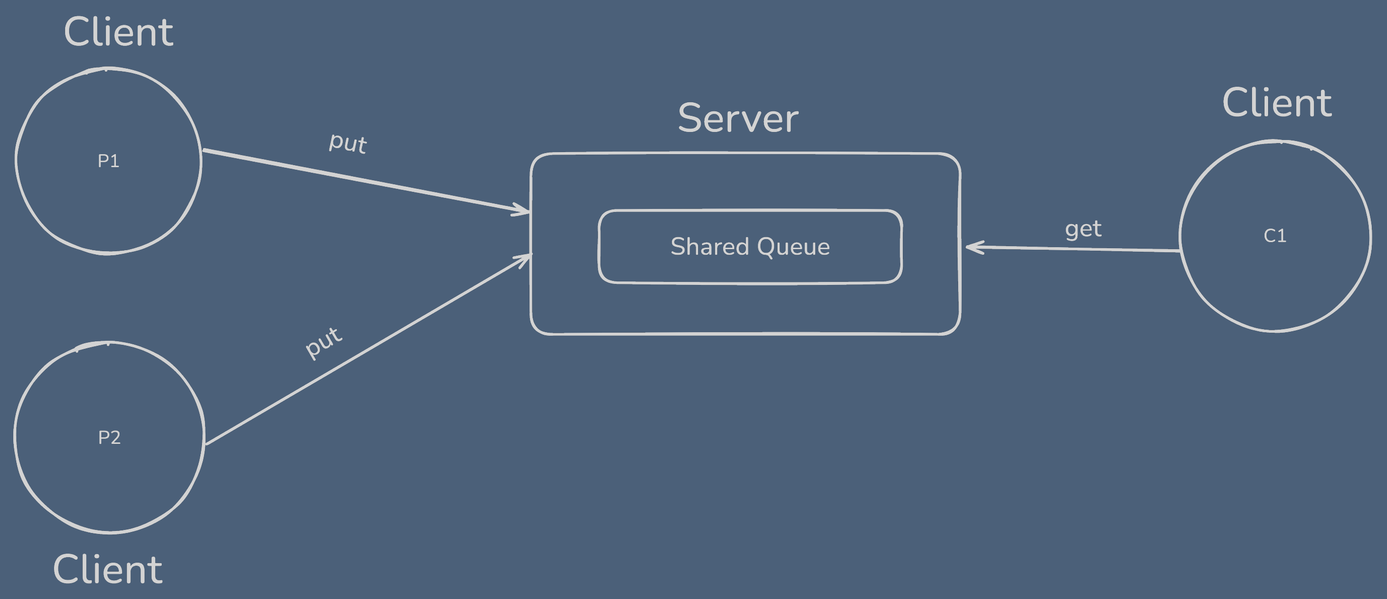
That solution is still reasonable, but we wanted to take it to the next level by also using **sockets** to establish the connection between different machines.

Our first implementation can be found in the *threads* folder inside the delivery package.

Producer-Consumer System with Sockets

Using **sockets**, we can establish a connection between the *clients* and the *server*.

The server manages the *shared queue*, and the clients can create and *put* objects inside the queue (**Producer**) or *get* and consume objects from the queue (**Consumer**).



Server implementation

Function that handles the connection with clients.

It recognizes which type of client is connecting (**Producer**, **Consumer**) and it handles that case.

When the connection is *lost*, or there is a *JSON Error*, an exception is being handled.

```python
def handle_client(self, client_socket, client_address) -> None:
        print(f"New connection from {client_address}")
        while True:
            try:
                message = client_socket.recv(1024).decode('utf-8')
                if not message:
                    break
                data = json.loads(message)
                client_type = data.get('type')

                if client_type == 'producer':
                    self.handle_producer(data)
                elif client_type == 'consumer':
                    self.handle_consumer(client_socket)

            except (ConnectionResetError, json.JSONDecodeError) as e:
                print(f"Connection lost from {client_address}: {e}")
                break

        client_socket.close()
```

Server implementation

Function that handles the case if a client is a *Producer*. The generated data is *put* inside the shared queue. If the queue is full, it prints it.

```python
def handle_producer(self, data) -> None:
        try:
            queue.put(data, timeout=1)  # Block for 1 second if full
            print(f"Produced: {data}")
        except Full:
            print(f"Queue is full. Producer {data['producer_id']} is waiting.")
```

Function that handles the case if a client is a *Consumer*. It gets the data from the queue and it returns it to the client using JSON dumps. If the queue is empty, it prints it.

```python
def handle_consumer(self, client_socket) -> None:
        try:
            item = queue.get(timeout=1)  # Block for 1 second if empty
            client_socket.send(json.dumps(item).encode('utf-8'))
            print(f"Consumed: {item}")
        except Empty:
            client_socket.send(b'queue_empty')
            print("Queue is empty. Consumer is waiting.")
```

Producer implementation

The *Producer* client generates some random JSON data before sending it to the *Server*.

After it sends the data, we put the thread to sleep to simulate a pause before producing new data.

```
def produce(self):
        while True:
            data = {
                "type": "producer",
                "producer_id": self.id,
                "value": random.randint(1, 100),
                "timestamp": time.time()
            }
            json_data = json.dumps(data)

            self.log(f"Sending data: {json_data}")
            self.socket.send(json_data.encode('utf-8'))

            time.sleep(random.uniform(0.5, 2))
```

Consumer implementation

The *Consumer* client asks the server to consume some data.

When the queue is empty, the client waits before trying again.

When the data is returned by the *Server*, it consumes it by printing it on the console.

```
def consume(self):
        while True:
            data = {"type": "consumer"}
            self.socket.send(json.dumps(data).encode('utf-8'))

            response = self.socket.recv(1024).decode('utf-8')

            if response == 'queue_empty':
                self.log("Queue is empty. Waiting for new items...")
            else:
                item = json.loads(response)
                self.log(f"Consumed: {item}")

            time.sleep(random.uniform(1, 3))
```

Usage

```
~/De/U/MAGISTRALE/Dis/L/3/c/sockets main !1 ?11 python server.py
Server started and listening on port 12345
New connection from ('127.0.0.1', 65435)
Produced: {'type': 'producer', 'producer_id': 1, 'value': 36, 'timestamp': 1728033747.8338869}
New connection from ('127.0.0.1', 65436)
Produced: {'type': 'producer', 'producer_id': 2, 'value': 62, 'timestamp': 1728033748.224261}
New connection from ('127.0.0.1', 65437)
Consumed: {'type': 'producer', 'producer_id': 1, 'value': 36, 'timestamp': 1728033747.8338869}
Produced: {'type': 'producer', 'producer_id': 2, 'value': 42, 'timestamp': 1728033749.091201}
Produced: {'type': 'producer', 'producer_id': 1, 'value': 100, 'timestamp': 1728033749.4633899}
Produced: {'type': 'producer', 'producer_id': 2, 'value': 39, 'timestamp': 1728033749.597884}
Produced: {'type': 'producer', 'producer_id': 1, 'value': 17, 'timestamp': 1728033750.137204}
Produced: {'type': 'producer', 'producer_id': 1, 'value': 60, 'timestamp': 1728033751.1613371}
Consumed: {'type': 'producer', 'producer_id': 2, 'value': 62, 'timestamp': 1728033748.224261}
Queue is full. Producer 2 is waiting.
Produced: {'type': 'producer', 'producer_id': 1, 'value': 5, 'timestamp': 1728033752.399925}
Consumed: {'type': 'producer', 'producer_id': 2, 'value': 42, 'timestamp': 1728033749.091201}
Queue is full. Producer 2 is waiting.
Queue is full. Producer 1 is waiting.
Queue is full. Producer 1 is waiting.
Queue is full. Producer 2 is waiting.
Produced: {'type': 'producer', 'producer_id': 1, 'value': 56, 'timestamp': 1728033755.418102}
Consumed: {'type': 'producer', 'producer_id': 1, 'value': 100, 'timestamp': 1728033749.4633899}
Produced: {'type': 'producer', 'producer_id': 2, 'value': 25, 'timestamp': 1728033756.335927}
Consumed: {'type': 'producer', 'producer_id': 2, 'value': 39, 'timestamp': 1728033749.597884}
Queue is full. Producer 1 is waiting.
Queue is full. Producer 2 is waiting.
Produced: {'type': 'producer', 'producer_id': 1, 'value': 20, 'timestamp': 1728033758.4178848}
Consumed: {'type': 'producer', 'producer_id': 1, 'value': 17, 'timestamp': 1728033750.137204}
Queue is full. Producer 2 is waiting.
Produced: {'type': 'producer', 'producer_id': 2, 'value': 93, 'timestamp': 1728033759.746306}
Consumed: {'type': 'producer', 'producer_id': 1, 'value': 60, 'timestamp': 1728033751.1613371}
Queue is full. Producer 1 is waiting.
Queue is full. Producer 2 is waiting.
Consumed: {'type': 'producer', 'producer_id': 1, 'value': 5, 'timestamp': 1728033752.399925}
```

```
~/Desktop/UNIBO/MAGISTRALE/Distributed software systems/Labs/3/code/new_sockets main !1 ?11 python consumer_client.py
Enter Consumer ID: 5
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 74, 'timestamp': 1728033830.603394}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 48, 'timestamp': 1728033830.978511}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 22, 'timestamp': 1728033831.816652}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 64, 'timestamp': 1728033832.584658}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 71, 'timestamp': 1728033833.1927922}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 70, 'timestamp': 1728033833.571337}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 11, 'timestamp': 1728033834.70993}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 11, 'timestamp': 1728033835.108917}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 36, 'timestamp': 1728033836.050004}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 81, 'timestamp': 1728033838.46359}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 68, 'timestamp': 1728033839.94849}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 14, 'timestamp': 1728033842.968396}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 28, 'timestamp': 1728033844.949576}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 2, 'timestamp': 1728033846.621964}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 76, 'timestamp': 1728033848.396756}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 91, 'timestamp': 1728033850.4099252}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 1, 'value': 11, 'timestamp': 1728033851.9928908}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 28, 'timestamp': 1728033854.567287}
[Consumer 5] Consumed: {'type': 'producer', 'producer_id': 2, 'value': 8, 'timestamp': 1728033857.089202}
```

```
~/Desktop/UNIBO/MAGISTRALE/Distributed software systems/Labs/3/code/new_sockets main !1 ?11 python producer_client.py
Enter Producer ID: 2
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 48, "timestamp": 1728033830.978511}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 22, "timestamp": 1728033831.816652}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 70, "timestamp": 1728033833.571337}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 11, "timestamp": 1728033834.70993}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 29, "timestamp": 1728033835.422067}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 71, "timestamp": 1728033836.6451719}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 81, "timestamp": 1728033838.46359}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 52, "timestamp": 1728033839.363647}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 68, "timestamp": 1728033839.94849}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 59, "timestamp": 1728033841.948748}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 38, "timestamp": 1728033843.1664379}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 10, "timestamp": 1728033844.849136}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 2, "timestamp": 1728033846.621964}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 38, "timestamp": 1728033847.648113}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 6, "timestamp": 1728033849.377312}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 91, "timestamp": 1728033850.4099252}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 75, "timestamp": 1728033851.185728}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 56, "timestamp": 1728033852.917672}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 28, "timestamp": 1728033854.567287}
[Producer 2] Sending data: {"type": "producer", "producer_id": 2, "value": 40, "timestamp": 1728033856.326606}
```

Usage with more Clients

In the slide before we tried using 2 *Producer Clients* and 1 *Consumer Client*.

When scaling it to 3 *Producer Clients* and 2 *Consumer Clients* you can see how the concurrency on the shared queue gets messier.

For every client, the wait time to do something is way higher the more you scale it. This is because the locking mechanism is not optimal or fair to manage this type of concurrency.

Another mechanism like **monitors** would be more fair, considering the fact that locks do not follow a *FCFS (First Come First Serve)* or *Round-Robin* logic.

Thank you!