Chatroom Server (Message Queue + Threaded Router)

สมมติมี **บริการแชทเล็ก ๆ ในมหาวิทยาลัย** ทุกคนเปิด client แบบ command line ที่สามารถเข้า room อย่างเช่น `#os-lab` พูดคุย ส่งข้อความตรง (DM) หาเพื่อน หรือดูว่าใครออนไลน์อยู่ได้ ทั้งหมดนี้จะส่งข้อความไปยัง **เครื่องแม่ข่ายกลาง** ซึ่งทำงานเหมือนบรรณารักษ์คอยจัดระเบียบ ข้อความคำสั่งจาก client แต่ละตัวจะถูกส่งเข้าไปใน **ควบคุมกลาง (control queue)** และเครื่องแม่ข่ายจะอ่านข้อความเหล่านั้นทีละรายการ ปรับปรุงข้อมูลในระบบ แล้วกระจายข้อความกลับออกไปยัง client ที่เกี่ยวข้อง โดยใช้ **คิวตอบกลับส่วนตัว (reply queue)** ของแต่ละ client อีกที

ภายในเครื่องแม่ข่ายมีบทบาทหลักอยู่สองส่วน คือ

1. **Router** – ทำงานเป็น thread ที่ฟังคำสั่งจาก control queue ตลอดเวลา เมื่อมีคำสั่ง `JOIN` room ก็จะเพิ่ม client เข้าไปในรายการสมาชิกของ room นั้น ถ้าเป็น `SAY` room text... ก็จะสร้าง “งานกระจายข้อความ (broadcast task)” เพื่อบอกว่าต้องส่งข้อความนี้ไปให้สมาชิกทั้งหมดในห้อง
2. **Broadcaster Pool** – เป็นกลุ่ม thread คนงาน (workers) ที่รับงานกระจายข้อความเหล่านั้น แล้วทำการส่งต่อจริง ๆ ไปยัง client แต่ละราย การแยกระหว่าง router กับ broadcaster ช่วยให้ระบบสามารถรับคำสั่งใหม่ ๆ ได้ต่อเนื่อง แม้ขณะข้อความกำลังถูกส่งออก

เพื่อจัดการทุกอย่าง เครื่องแม่ข่ายต้องเก็บสมุดบันทึกสองเล่มในหน่วยความจำ

- **Room Registry** – บันทึกว่าห้องไหนมี client ID อะไรอยู่บ้าง
- **Client Registry** – บันทึกว่า client ID ไหนมี reply queue อะไรเพื่อใช้ส่งข้อความกลับ

เพราะมีหลาย thread ทำงานพร้อมกัน ข้อมูลพวกนี้ต้องป้องกันด้วย ตัวล็อกแบบอ่าน–เขียน (reader–writer lock) ส่วนใหญ่จะเป็นการอ่าน (ถามว่าใครอยู่ในห้อง) มีบางครั้งเท่านั้นที่ต้องเขียน (มีคน join หรือ leave)

ฝั่ง client มีงานที่เรียบง่ายกว่า ตอนเริ่มทำงานมันจะสร้าง reply queue ของตัวเองแล้วบอกเครื่องแม่ข่ายว่า “นี่คือฉัน และนี่คือที่อยู่ของฉัน” จากนั้น client จะสร้างสอง thread ขึ้นมา ดังนี้

- **Thread** แรกคอยอ่านคำสั่งจากแป้นพิมพ์แล้วส่งไปยัง control queue (`JOIN, SAY, DM, WHO, LEAVE, QUIT`)
- **Thread** ที่สองคอยบล็อก (block) รอข้อความใน reply queue แล้วพิมพ์ผลลัพธ์ออกมา (ข้อความในห้อง, system notice, รายชื่อสมาชิก, error message ฯลฯ)

  Error Handling: สิ่งที่น่าสนใจคือการจัดการ สถานการณ์ผิดปกติ

- ถ้า reply queue ของ client เต็ม จะทำยังไง? เครื่องแม่ข่ายควรรอ (block) จนกว่าจะส่งได้, ทิ้งข้อความไปเลย, หรือส่ง system notice แจ้งว่าคุณตามไม่ทัน?

- ถ้า client หายไปโดยไม่ส่ง `QUIT` ล่ะ? อาจต้องมี heartbeat เล็ก ๆ ระหว่าง client–server เพื่อตรวจสอบว่าใครยังมีชีวิตอยู่

ในด้านประสิทธิภาพ เราสามารถปรับจำนวน broadcaster threads ได้. ถ้ามีแค่ thread เดียวระบบจะเรียบง่ายแต่ช้า ถ้าเพิ่มเป็น 2, 4, หรือ 8 throughput จะดีขึ้น แต่ก็อาจเจอปัญหาการแย่งล็อกหรือ queue limit กลายเป็นคอขวด นักศึกษาสามารถทดลองวัด **latency** (เวลาตั้งแต่ enqueue ⟶ ส่งถึง) และ **throughput** (จำนวนข้อความต่อวินาที) ภายใต้การตั้งค่าที่ต่างกัน

- **โปรโตคอลถูกออกแบบให้ง่ายแต่ครบวงจร:** `JOIN` เพื่อเข้าห้อง, `SAY` ส่งข้อความ, `DM` ส่งตรง, `WHO` ขอรายชื่อ, `LEAVE` และ `QUIT` ออกจากห้อง เครื่องแม่ข่ายยังสามารถส่ง system event เล็ก ๆ เช่น “Alice joined”, “ห้องว่างแล้ว” ซึ่งทำให้ห้องแชทรู้สึกมีชีวิตชีวา

เมื่อโปรเจ็กต์นี้เสร็จ นักศึกษาจะได้ลงมือกับแนวคิด OS หลัก ๆ: **การสื่อสารระหว่างโปรเซส (IPC)** ผ่าน message queue จริง ๆ, **การทำงานพร้อมกัน (concurrency)** และการซิงโครไนซ์ และ **การออกแบบให้เริ่ม–หยุดอย่างเป็นระเบียบ**ในระบบที่มีหลาย thread/process ถ้าอยากต่อยอด อาจเพิ่มระบบชื่อผู้ใช้, การบันทึกประวัติการสนทนา, หรือเปลี่ยน layer ล่างจาก message queue ไปเป็น TCP socket โดยยังคงสถาปัตยกรรม router–broadcaster เดิม—บทเรียนก็ยังคงเหมือนเดิม

เกณฑ์ประเมินหลัก (100 คะแนน)

1. ความถูกต้องของการทำงาน (30 คะแนน)

- **โปรโตคอลและคำสั่ง** JOIN, SAY, DM, WHO, LEAVE, QUIT ทำงานถูกต้อง
    - การจัดการ system events (Alice joined, ห้องว่าง)
- **การจัดการข้อมูล** Room Registry และ Client Registry ทำงานถูกต้อง
    - การเพิ่ม/ลบ client จาก room
- **การส่งข้อความ**
    - Broadcast ในห้องทำงานถูกต้อง และ Direct message ส่งถูกคน

2. การใช้ IPC และ Concurrency (30 คะแนน)

- **Message Queue Implementation**
    - Control queue และ reply queue ทำงานถูกต้อง
- **Thread Management**
    - Router thread และ Broadcaster pool
- **Synchronization**
    - Thread-safe operations และ Reader-writer locks สำหรับ registries

3. Non-Functional Requirements (20 คะแนน)

- Queue Management และ Client Disconnection
- Throughput Testing วัดจำนวนข้อความต่อวินาที และทดสอบกับ client หลายตัว

4. Code Quality และ Documentation (20 คะแนน)

- **Code Structure**
- **Documentation and Presentation**
    - README อธิบายการ compile/run
    - Comment ใน code สำคัญ
    - รายงานผล performance testing

เกณฑ์การให้คะแนน

- **A (90-100)**: ทำงานถูกต้องครบ + performance ดี + error handling ครบถ้วน
- **B (80-89)**: ทำงานถูกต้องส่วนใหญ่ + มี basic error handling
- **C (70-79)**: ฟังก์ชันหลักทำงาน + มีปัญหา concurrency บางจุด
- **D (60-69)**: ทำงานได้บางส่วน + มีปัญหา synchronization
- **F (<60)**: ไม่สามารถทำงานได้ หรือมีปัญหา race condition ร้ายแรง

ช่วยให้นักศึกษาเข้าใจแนวคิด IPC, concurrency และ system design อย่างลึกซึ้ง
พร้อมทั้งได้ประสบการณ์จริงในการพัฒนา Distributed system