Project Explanation

The **Priority-Based Task Execution System with Script Dispatcher** is a multithreaded client-server Java application designed for the Spring 2025 Operating Systems Lab at Qatar University. The system allows clients to submit Linux shell scripts for prioritized execution on a server, with features like task queuing, synchronization, rate limiting, and task history logging. It builds on Phase 1 work and involves three Ubuntu Server 22.04 LTS virtual machines (VMs):

1. **VM1 (Server)**: Hosts the server application, user accounts, shared directories, SSH, and MySQL services.
2. **VM2 (Client - Development Team)**: Runs client-side scripts and sends task requests to the server.
3. **VM3 (Client - Operations Team)**: Runs different client-side scripts and sends task requests to the server.

The project objectives include:

- Developing a server (Server.java) that listens on port 2500, validates client connections, manages a priority-based task queue, and executes scripts with synchronization to prevent concurrent execution of the same script.
- Implementing two client applications (Client1.java and Client2.java) that run local scripts and interact with the server to submit tasks, query the queue, cancel tasks, or retrieve task history.
- Ensuring thread safety, rate limiting (one task per client every 5 minutes), and proper logging of task statuses (QUEUED, EXECUTING, COMPLETED, REJECTED, ERROR).
- Writing error-free shell scripts with inline comments and adhering to anti-plagiarism measures through live demos and code explanations.

Key features include:

- **Task Prioritization**: Tasks are queued based on priority (1=High, 2=Medium, 3=Low) and timestamp.
- **Communication Protocol**: Clients send commands (e.g., REQUEST_TASK;2003;Client1;1) and receive responses in the format STATUS;TIMESTAMP;MESSAGE.
- **Client Commands**: QUEUE_STATUS, CANCEL_TASK;TaskID, TASK_HISTORY for managing tasks.

- **Scripts**: Server executes scripts like user_setup.sh, dir_perms.sh, etc., while clients run local scripts like ssh_config.sh or resource_report.sh.

How to Start Implementing the Project

1. **Environment Setup**:
    - **Provision VMs**: Set up three Ubuntu Server 22.04 LTS VMs with the specified configurations (VM1: 4GB RAM, 4 CPUs; VM2/VM3: 2GB RAM, 2 CPUs). Use tools like VirtualBox or VMware. If VMs cannot run on one machine, use separate physical machines.
    - **Install Dependencies**: Install Java (e.g., OpenJDK 17), MySQL, and SSH on VM1. Ensure Java is installed on VM2 and VM3 for client applications.
    - **Network Configuration**: Ensure VMs can communicate over the network. Assign static IP addresses and test connectivity using ping or netcat.
    - **Directory Setup**: Create shared directories on VM1 for script storage and execution. Set up user accounts and permissions as needed.
2. **Project Structure**:
    - Create a project directory with subfolders for Java classes (src), shell scripts (scripts), and documentation (docs).
    - Write the following Java files: Server.java, Client1.java, Client2.java.
    - Develop the required shell scripts: Network.sh, user_setup.sh, dir_perms.sh, system_monitor.sh, file_audit.sh, MySQL_login_<USER_NAME>.sh for the server, and client-specific scripts (ssh_config.sh, fix_perms.sh, login_audit.sh for Client1; resource_report.sh, quota_check.sh for Client2).
3. **Server Implementation (Server.java)**:
    - **Socket Setup**: Create a ServerSocket listening on port 2500. Accept client connections in a loop, spawning a new thread for each client.
    - **Connection Validation**: On client connection, execute Network.sh to verify connectivity with both clients (e.g., using ping or nc).
    - **Task Queue**: Implement a PriorityQueue to store tasks, sorted by priority and timestamp. Each task should include ServiceNumber, ClientName, Priority, TaskID, and Timestamp.

- **Thread Synchronization**: Use synchronized blocks or ReentrantLock to ensure only one instance of a script runs at a time.
- **Rate Limiting**: Track client task submissions using a HashMap<ClientName, Timestamp>. Reject tasks if the client submitted one within the last 5 minutes.
- **Task Execution**: Use ProcessBuilder to execute shell scripts. Capture output and errors for logging.
- **Command Handling**: Parse client commands (REQUEST_TASK, QUEUE_STATUS, CANCEL_TASK, TASK_HISTORY) and respond in the format STATUS;TIMESTAMP;MESSAGE.
- **Logging**: Store task history (completed, cancelled, failed) in a file or MySQL database on VM1.

4. **Client Implementation (Client1.java and Client2.java)**:
   - **Socket Connection**: Connect to the server on port 2500 using Socket.
   - **Local Scripts**: Use ProcessBuilder to execute local scripts (ssh_config.sh, etc., for Client1; resource_report.sh, etc., for Client2).
   - **Task Submission**: Send REQUEST_TASK commands every 5 minutes using a timer or scheduler (e.g., ScheduledExecutorService).
   - **Command Interface**: Allow sending QUEUE_STATUS, CANCEL_TASK, or TASK_HISTORY commands, possibly via a simple console interface.
   - **Response Handling**: Parse server responses and display them to the user.
5. **Shell Scripts**:
   - Write each script with error handling (e.g., set -e for immediate exit on errors) and inline comments explaining the logic.
   - Example for Network.sh: Use ping or nc to check connectivity with client VMs and return a success/failure status.
   - Example for user_setup.sh: Create user accounts, set passwords, and assign permissions.
   - Test scripts independently to ensure they are error-free.
6. **Testing**:

- Test server-client communication using netcat or a simple socket program before full implementation.
- Simulate multiple clients submitting tasks to verify priority queuing, synchronization, and rate limiting.
- Test all client commands and server responses.
- Validate shell script execution and error handling.

7. **Documentation and Submission**:
   - Create a Word document with a cover page, task distribution table, and screenshots of task execution.
   - Comment all Java classes and scripts thoroughly.
   - Package all files into a zip file named Student1_Student2_Student3_Student4.zip.
   - Prepare for the demo by practicing script explanations and live task execution.

Task Distribution for 4 Students

To distribute the project tasks equally among four students, divide responsibilities based on the major components: server implementation, client implementations, shell scripts, and documentation/testing. Each student should contribute to coding, testing, and documentation, with clear roles to avoid overlap. Below is a suggested task distribution, assuming equal contribution (25% per student):

Student	Tasks	Contribution (%)
Student 1	**Server Core Implementation** - Implement Server.java: socket setup, client connection handling, and Network.sh execution. - Develop task queue using PriorityQueue and synchronization logic. - Write and test Network.sh and user_setup.sh. - Document server code and script logic in the Word document. - Take screenshots of server task execution.	25%
Student 2	**Server Task Management and Logging** - Implement task execution, rate limiting, and command handling (REQUEST_TASK, QUEUE_STATUS, CANCEL_TASK, TASK_HISTORY) in	25%

Student	Tasks	Contribution (%)
	Server.java. - Set up task history logging (file-based or MySQL). - Write and test dir_perms.sh and system_monitor.sh. - Document task management code and script logic. - Take screenshots of task status responses.	
Student 3	**Client 1 Implementation and Scripts** - Implement Client1.java: socket connection, local script execution (ssh_config.sh, fix_perms.sh, login_audit.sh), and server task submission. - Implement command interface for QUEUE_STATUS, CANCEL_TASK, TASK_HISTORY. - Write and test ssh_config.sh, fix_perms.sh, and login_audit.sh. - Document client code and script logic. - Take screenshots of Client1 interactions.	25%
Student 4	**Client 2 Implementation and Testing** - Implement Client2.java: socket connection, local script execution (resource_report.sh, quota_check.sh), and server task submission. - Implement command interface for QUEUE_STATUS, CANCEL_TASK, TASK_HISTORY. - Write and test resource_report.sh, quota_check.sh, and file_audit.sh. - Perform system-wide testing (server-client communication, task prioritization, rate limiting). - Compile the final Word document, create the zip file, and take screenshots of Client2 interactions.	25%

Additional Notes for Task Distribution

- **Collaboration**: All students should meet regularly to discuss progress, integrate components, and test the system as a whole. Use version control (e.g., Git) to manage code and scripts.
- **Shell Scripts**: Each student writes 2-3 scripts, ensuring they are error-free and well-commented. The MySQL_login_<USER_NAME>.sh script can be a joint effort or assigned to Student 2 (since it relates to server logging).
- **Documentation**: Each student contributes to the Word document by documenting their code/scripts and providing screenshots. Student 4 finalizes the document and zip file.

- **Demo Preparation**: All students must understand the entire system to explain their code and scripts during the demo. Practice running tasks live and explaining logic.
- **VM Setup**: Assign one student (e.g., Student 1) to set up the VMs initially, but all students should have access to test their components.

Tips for Success

- **Start Early**: Begin with VM setup and basic socket programming to ensure the environment is ready.
- **Modular Development**: Implement and test small components (e.g., socket connection, task queue) before integrating.
- **Error Handling**: Ensure robust error handling in both Java code and shell scripts to avoid penalties.
- **Practice for Demo**: Rehearse explaining every line of code and script to avoid plagiarism penalties.
- **Use Comments**: Add clear inline comments in Java and scripts to demonstrate understanding.

This distribution ensures each student has a balanced workload, covering coding, scripting, testing, and documentation, while fostering collaboration to meet the project's requirements.