Question 1:

Objective:

Develop a basic client-server application in Python that demonstrates understanding of socket programming, inter-process communication, and file handling in a distributed network environment.

Problem Statement:

Design and implement a basic distributed client-server application using Python. The server should transmit a binary image file to connected clients, ensuring data integrity with checksum validation. The client should verify and save the received file, with both components demonstrating knowledge of socket programming, file handling, and multithreading.

Requirements:

1. Server:

Commence by configuring the server application to listen on a port number P derived from your student ID S; this involves calculating A as the sum of the digits in S, B as the product of all non-zero digits in S, and C as the result of reversing S and applying a modulo operation with 1000, subsequently computing P using the formula P = ((A * B) + C) % 49152 + 16384 to ensure it falls within the valid non-system port range (16384–65535). The server should accept incoming client connections, managing multiple clients sequentially without the need for restarting. It must read a binary file named '<ID>.bmp' located in the server's directory (which you should create), and transmit this file to the connected client in chunks of 1024 bytes. After completing the file transfer, implement a simple checksum (such as summing the byte values modulo 256) and send this checksum to the client. Additionally, employ multi-threading to handle each client connection in a separate thread.

2. Client:

Initiate the client application by establishing a connection to the server using the accurate IP address and the port number derived from the specified calculation. Upon successful connection, the client should receive the binary file in incremental segments, reconstructing it as the data arrives. Concurrently, it must compute a checksum on the received data, employing the same algorithm used by the server, and subsequently compare this checksum with the one provided by the server post-transfer. Following this verification process, the client should display a message that clearly indicates the success or failure of the file transfer based on the checksum comparison outcome. Finally, the reconstructed file should be saved in the client's directory under the name 'received-<ID>.bmp'.

Question 2:

Objective:

Develop a concurrent distributed mini banking system with secure authentication, basic banking operations, and persistent data storage. (Language: Python/Java)

Problem Statement:

Design and implement a distributed banking system using a client-server model that supports concurrent client handling. The server must provide secure authentication and basic banking operations (balance inquiry, deposit, withdrawal, and fund transfer) while maintaining persistent user data.

Requirements:

1. Server:

Incorporating multi-threading to ensure the server's capacity to handle numerous clients concurrently, the system should embed a fundamental authentication protocol by maintaining a repository of predefined user credentials (specifically, usernames paired with passwords) within a data structure akin to a dictionary; this facilitates the verification of client login attempts, thereby granting or denying access based on credential validity. Once a client is authenticated, the server ought to offer a range of services: it should disclose the client's current monetary balance upon inquiry; allow the client to augment this balance by accepting deposit amounts and updating the stored value accordingly; enable the client to decrement their balance through withdrawal requests, contingent upon the balance being sufficient to cover the requested amount; and keep a meticulous record of all transactional activities (including deposits and withdrawals) pertaining to the client session, effectively maintaining a transaction log that can be communicated to the client as needed.

2. Client:

Each client is expected to establish a connection with the server and provide their username and password to authenticate their identity; once authenticated, they should engage with the server to perform various operations such as viewing their current balance, depositing funds, withdrawing money (while ensuring there are sufficient funds available) and requesting a log of their transactions, ultimately disconnecting gracefully after all desired transactions are completed.

Question 3:

Objective:

Design and implement a Java RMI-based application to perform matrix operations. The application should include both server and client functionalities, with the server providing matrix-related services remotely.

Problem Statement:

You are required to design and implement a Matrix Operations Service using Java RMI that can remotely perform a set of mathematical operations on matrices. The service must support Matrix Addition, which involves element-wise addition of two matrices A and B of dimensions m × n, ensuring that each element $A_{ij} + B_{ij}$ is computed only if m1 = m2 and n1 = n2. For Matrix Multiplication, compute the product C = A × B, where matrix A is of dimensions m × n and matrix B is of dimensions n × p, ensuring compatibility where n1 = m2 to generate a resulting matrix C of size m × p. Implement Matrix Transposition, where a given matrix A of size m × n is transformed into A^T, such that each element $A_{ij} = A^T_{ji}$. Finally, develop logic for Determinant Calculation, which requires finding the determinant det(A) of a square matrix of size n × n using recursive Laplace expansion or cofactor methods, explicitly throwing an error if the matrix is not square. Ensure robust validation for all operations, rejecting addition when matrices are of different dimensions or multiplication when the inner dimensions do not match, while also throwing exceptions with meaningful error messages to guide the user.

Requirements:

1. Server:

The server must implement the MatrixOperations remote interface, defining methods for matrix operations such as addition, multiplication, transposition, and determinant calculation. A server-side class (MatrixOperationsImpl) should provide the actual logic for these operations while validating matrix dimensions and handling errors (e.g., incompatible matrices or non-square matrices for determinant calculation). The service should be registered with the RMI registry, making it accessible for remote invocation by clients. The server must also support handling multiple client requests simultaneously, ensuring reliability and proper error handling for invalid inputs.

2. Client:

The client application should connect to the RMI server and provide a user-friendly, menu-driven interface. The interface should allow users to input two matrices for operations like addition or multiplication, or a single matrix for transposition or determinant calculation. The application must display results of the operations or meaningful error messages when inputs are invalid (e.g., mismatched matrix dimensions). The client should ensure seamless communication with the server, enabling users to perform multiple operations in a single session.

Question 4:

Objective:

The objective is to design a high-performance matrix computation system capable of executing matrix multiplication in parallel using CPU (sequential and OpenMP) and GPU (CUDA). The goal is to explore multi-platform parallel programming techniques, optimize for performance, and dynamically choose the most efficient computation method based on input size. (Language: C/C++)

Problem Statement:

You are tasked with developing a module that performs two consecutive matrix multiplications: $D = A \times B$ and $E = D \times C$, where matrices A, B, and C have dimensions $m \times n$, $n \times k$, and $k \times p$ respectively. The implementation must include three approaches: sequential execution, parallel CPU computation with OpenMP, and parallel GPU computation with CUDA. Additionally, the system should dynamically select the most efficient method for execution based on the dimensions of the matrices, and validate the results for correctness.

Requirements:

Implement functions for sequential, OpenMP, and CUDA - based matrix multiplication. Dynamically decide whether to use sequential, OpenMP, or CUDA based on input size, using predefined thresholds. Measure execution time for each method and compute the speedup achieved by OpenMP and CUDA compared to the sequential implementation. Ensure correctness by validating all results against the sequential method. Provide output for execution times, speedup calculations, and dynamic selection decisions. (Optimize computations by skipping redundant steps when C is the identity matrix. - Optional)

Question 5:

Objective:

The objective of this is to design and implement a hybrid distributed-parallel sorting system using C/C++ that combines Normal QuickSort, Parallel QuickSort (using OpenMP), and Hyper QuickSort (simulated with OpenMP). The goal is to effectively utilize multi-threading and parallel processing techniques to improve sorting efficiency, while also analyzing the performance and scalability of these algorithms across varying dataset sizes.

Problem Statement:

Sorting large datasets efficiently is a critical challenge, especially with traditional algorithms like QuickSort that underutilize multi-core architectures. Parallel algorithms, such as Parallel QuickSort and Hyper QuickSort, offer performance improvements but face challenges like thread management and load balancing. This question requires implementing a hybrid sorting system that combines Parallel QuickSort and Hyper QuickSort, dynamically adjusts recursion depth based on available threads, ensures balanced workload distribution, and demonstrates superior performance and scalability compared to standalone methods.

Requirements:

To accomplish the task, students are required to implement a hybrid sorting system that combines the strengths of Normal QuickSort, Parallel QuickSort, and Hyper QuickSort. The system must partition the dataset into two halves, using Parallel QuickSort with OpenMP for the first half and Hyper QuickSort with dynamic recursion depth for the second half. Dynamic depth control should be implemented in both parallel algorithms to limit recursion levels based on the number of available threads, retrieved using omp_get_max_threads(). Additionally, load balancing is essential to optimize performance, and this must be achieved by introducing a thread pool in Hyper QuickSort to ensure an even distribution of data chunks among threads.

Beyond implementation, students must conduct a detailed performance analysis by measuring and comparing execution times for Normal QuickSort, Parallel QuickSort, Hyper QuickSort, and the hybrid system. These algorithms should be tested on datasets of sizes 5,000, 50,000, 500,000, and 5,000,000 elements. Key performance metrics, such as speedup—representing the time reduction relative to Normal QuickSort—and load imbalance factor—quantifying the efficiency of workload distribution—must be calculated and analyzed. By meeting these requirements, students will demonstrate their ability to effectively integrate parallel programming techniques with theoretical concepts and critically evaluate algorithm performance.

Question 6:

Objective:

The objective of this lab is to design and implement a complete machine learning pipeline using AWS services. Participants will leverage Amazon S3 and SageMaker to manage data, train a machine learning model, evaluate its performance, and deploy it for real-time predictions.

Problem Statement:

You are required to build a machine learning pipeline that starts with creating a uniquely named Amazon S3 bucket based on your student ID. The dataset will be uploaded to this bucket and pre-processed to prepare it for training. Following this, you will train a Random Forest Classifier on the data using SageMaker.

Requirements:

Participants must programmatically create a unique Amazon S3 bucket using the formula bucket_name = f"student-no-{student_id % 10000}" (e.g., for Student ID 202363001, the bucket name will be student-no-3001) and upload a dataset to this bucket using Python code executed in Jupiter Notebook or Google Colab. The dataset should be a CSV file with at least 2,000 rows and 21 columns, including a binary target column for classification (you can use the .csv file provided during the lab session). The uploaded dataset must be accessed programmatically for preprocessing, which includes handling missing values, encoding categorical features, and normalizing numerical data. The pre-processed data will then be used to train a Random Forest Classifier using Amazon SageMaker, and the trained model will be saved as a .joblib file in the SageMaker model directory. Participants must utilize Python libraries like boto3, pandas, and SageMaker for completing the tasks.

Dataset Link: https://shorturl.at/hvfE5

Question 7:

Objective:

The objective is to analyze and compare the performance of training a deep learning model on a CPU-only setup versus a GPU-enabled setup. Students will measure and evaluate the training time, validation time, and total runtime for both setups while gaining insights into the benefits of hardware acceleration in machine learning tasks. (Platforms: Google Colab or Jupyter Notebook)

Problem Statement:

You are tasked with designing and implementing a neural network to classify grayscale images of American Sign Language (ASL) letters. The dataset contains 28x28 pixel images corresponding to 24 different classes (letters a to i and k to z). Letters j and z are excluded due to their motion-based representation.

Your goal is to train this neural network using the PyTorch library on two separate setups: one using only the CPU and the other utilizing a GPU. You will measure and compare the time taken for training and validation in both setups, analyze the results, and draw conclusions on the scenarios where GPU acceleration provides significant benefits.

Requirements:

You are required to implement a Python-based solution using PyTorch to classify grayscale images of American Sign Language (ASL) letters into 24 classes (letters a to i and k to z). The dataset must be pre-processed by normalizing pixel values to the range [0, 1], and the neural network model should include a flatten layer to reshape inputs, two dense layers with 512 neurons each and ReLU activations, and a final output layer with 24 neurons corresponding to the number of classes. Use the cross-entropy loss function and Adam optimizer, and train the model for 5 epochs. The implementation should support both CPU and GPU configurations, explicitly managing device placement using torch.device. For each setup, you must record training time, validation time, and total runtime per epoch, comparing the performance between CPU and GPU environments. Additionally, provide a comprehensive analysis of runtime differences, explain the benefits of GPUs for machine learning, and discuss the impact of dataset size and model complexity on performance.

Dataset Link: https://shorturl.at/hvfE5

Question 8:

Objective:

Design and implement a simple Java RMI-based application to track lab attendance for IIITV students. The application should allow faculty to mark and retrieve attendance records for a lab session.

Problem Statement:

You are tasked with creating a distributed system to manage lab attendance for a specific course at IIITV. Faculty should be able to mark students present for a given lab session and retrieve attendance details for any student or session. The system should handle basic validations, such as ensuring the session date is valid.

Requirements:

1. Server:

The server must implement a remote interface, AttendanceTracker, with methods to manage attendance records: markAttendance(String studentId, String sessionDate), getAttendanceByStudent(String studentId), and getAttendanceBySession(String sessionDate). A server-side implementation (AttendanceTrackerImpl) will handle these methods by maintaining attendance records in memory using a HashMap, where each session date is the key, and the value is a list of student IDs marked present for that session. The server should be registered with the RMI registry to allow remote access by clients and ensure thread-safe operations for multiple concurrent requests.

2. Client:

The client application must provide a user-friendly, menu-driven interface for faculty to interact with the server. It should allow faculty to mark attendance for a specific session, retrieve attendance details for a student, or fetch the list of students present for a session. The interface should handle invalid inputs gracefully, displaying meaningful error messages (e.g., when a student ID is not found or a session date is incorrectly formatted). The client must establish seamless communication with the RMI server, ensuring multiple operations can be performed in a single session.

Question 9:

Objective:
Design a client-server system in Python that handles multiple types of operations. The server should process requests from a client and return the appropriate results based on the operation type. The focus is on implementing a protocol for communication and handling errors in requests.

Problem Statement:
The server must perform the following operations based on the client's request:

1. **Arithmetic operations:**
    - ADD <num1> <num2>
    - SUBTRACT <num1> <num2>
    - MULTIPLY <num1> <num2>
    - DIVIDE <num1> <num2> (handle division by zero)
2. **String manipulations:**
    - REVERSE <string>
    - UPPERCASE <string>
    - LOWERCASE <string>
3. **Time operation:**
    - TIME (returns current server time)

The client sends one command per request, and the server responds with the result or an error message for invalid commands.

Requirements:

1. **Server:**
    - Use Python's socket library.
    - Handle client requests sequentially (one at a time).
    - Validate input for valid operations and handle errors (e.g., invalid command, division by zero).
    - Respond with the result of the operation or an error message.
2. **Client:**
    - Connect to the server and send commands.
    - Display the server's response (result or error).

Testing:
Test the system with various valid and invalid operations and verify that the server handles errors (like division by zero) correctly.

Question 10

Objective:
The objective is to design and implement a multi-client, multi-server system in a distributed environment using Java. This lab demonstrates Java's concurrency capabilities and its support for managing complex networking scenarios.

Problem Statement:
You are required to create a distributed chat application where multiple clients can connect to multiple servers. The system should enable clients to send messages to a server, which will broadcast the messages to all connected clients in real time.

Requirements:

1. **Servers:**
    - Implement the servers using **Java's java.net package.**
    - Each server should handle multiple clients concurrently using threads.
    - The server must broadcast any message received from a client to all other connected clients, ensuring real-time communication.
2. **Clients:**
    - Clients should connect to a specified server and send messages using a user-friendly interface (a console-based interface is acceptable).
    - Clients must display messages broadcasted by the server in real time.
3. **Concurrency and Communication:**
    - The system should support concurrent communication, allowing multiple clients to interact with the server without blocking other operations.
    - Use Java's multithreading features to manage multiple clients and ensure seamless communication.
4. **Testing:**
    - Test the application with multiple servers and clients running simultaneously. Verify that messages sent by one client are broadcasted correctly to all other clients connected to the same server.
    - Simulate scenarios where a client disconnects and ensure that the server continues to function smoothly.