CS 342: Computer Networks Lab

(July-November 2023)

Assignment – 4

This assignment is a programming assignment where you need to implement an application using socket programming in C/C++/java programming language. This is a Group **assignment; each group needs to create and** submit the assignment. The applications' description is given in this document.

Instructions:

- The application should be implemented in C/C++/java programming language only. No other programming language other than C/C++/java will be accepted.
- Submit the set of source code files of the application and report as a zipped file on MSTeams (maximum file size is 1 MB) by the deadline of 11:**55 pm on < 9.11.2023 >** (hard deadline). The ZIP file's name should be the **group number**.
- Only one student per group needs to upload the assignment on Teams with the group name.
- The assignment will be evaluated offline/through viva-voce during your lab session. where you will need to explain your source codes and execute them before the evaluator.
- Write your own source code and do not copy from any source. Plagiarism detection tool will be used, and any detection of unfair means will be penalized by awarding NEGATIVE marks (equal to the maximum marks for the assignment).

Weighted Fair Queuing (WFQ): WFQ is a flow-based queuing algorithm used in Quality of Service (QoS) that does two things simultaneously: It schedules interactive traffic to the front of the queue to reduce response time, and it fairly shares the remaining bandwidth between high bandwidth flows. A stream of packets within a single session of a single application is known as flow or conversation. WFQ is a flow-based method that sends packets over the network and ensures packet transmission efficiency which is critical to interactive traffic.

1. Implement a load balancer for a web server cluster using WFQ algorithm.

Scenario:

Consider a cloud-based web server cluster that serves multiple websites. Each website owner has paid for a specific amount of bandwidth and processing power, which they should get proportionally.

Requirements:

- **Website Class:**
  Create a **website** class that represents a website hosted on the server cluster. Each website should have a unique ID, owner, and queue for incoming requests.
- **HttpRequest Class:**
  Implement an **HttpRequest** class that represents an incoming HTTP request. Each request should have a unique ID, website ID, and processing time.
- **LoadBalancer Class:**
  Create a **LoadBalancer** class that manages the incoming requests and implements WFQ. The load balancer should support the following operations:
- **add_website(website_id, owner_id, bandwidth, processing_power)**:
  Add a new website with specified ID, owner ID, allocated bandwidth, and processing power to the system.
- **enqueue_request(http_request)**:
  Enqueue an HTTP request into the corresponding website's queue.
- **dequeue_request()**:
  Dequeue the next HTTP request based on the WFQ scheduling algorithm.
- **WFQ Scheduling:**
  Implement the WFQ scheduling algorithm. Websites with higher allocated resources get served more frequently.

Test Case 1: Basic Load Balancing

- **Scenario:** Two websites with equal allocated resources.

Test Case 2: Differential Bandwidth Allocation

- **Scenario:** Websites have different allocated bandwidths.

Test Case 3: Differential Processing Power Allocation

- **Scenario:** Websites have different allocated processing powers.

Test Case 4: Equal Allocations

- **Scenario:** Websites have equal allocated resources.

Test Case 5: Large Number of Requests

- **Scenario:** Stress testing with a large number of HTTP requests.

Test Case 6: Empty Queues

- **Scenario:** No requests in any website queues.

Test Case 7: Unequal Bandwidth and Processing Power

- **Scenario:** Websites have both unequal bandwidth and processing power allocations.

Test Case 8: Edge Case - Single Website

- **Scenario:** Only one website present.

2. Analyzing Airport Security Lines

Your assignment is to create a simulation that replicates and improves security screening processes in a busy airport environment. As in real-life airports, passengers arrive at irregular intervals and undergo varying durations of security checks. This simulation scenario mirrors the actual challenges of queuing in airport scenarios, emphasizing the need to optimize security lines for heightened passenger satisfaction and overall airport efficiency.

Instructions:

- Simulation Setup:
  Your C++ program simulates the airport security screening process, which can be modeled as a queuing system. Initially, consider a single security line with one security scanner.

- User-Defined Parameters:
  Allow the airport authorities to specify two critical parameters:
  - Arrival Rate (λ): The rate at which passengers arrive at the security checkpoint.
  - Service Rate (μ): The rate at which passengers are processed by the security scanner.

- Data Collection and Analysis: Collect essential statistics, such as:
  - Average Waiting Time: The average time passengers spend waiting in line before their security checks.
  - Average Queue Length: The average number of passengers in the queue at any given time.
  - System Utilization: The percentage of time the security scanner is actively processing passengers.

- Queue Length Optimization:
  As airports often experience variations in passenger flow, expand your program to incorporate a finite buffer ('K') to accommodate waiting passengers. Allow the authorities to set the buffer size and assess how it impacts the overall efficiency of the security screening process.

- Multi-Server Expansion:
  In larger airports, consider scenarios with multiple security lines having multiple security scanners. The airport authorities should be able to specify the number of security scanners and observe how it affects the system's performance.

- Buffered Multi-Server Model:

Finally, Extend the multi-server model by adding a buffer ('m') in front of 'm' security scanners. Allow authorities to set the buffer size and the number of security scanners ('m') to investigate the behavior of this more complex system.

- Simulation Results:
  Run the simulations with user-defined parameters and configurations, analyze the results, and discuss how different models and optimizations can enhance the efficiency of the airport security screening process, thereby improving passenger experience.

- Report:
  Prepare a comprehensive report summarizing the analysis, optimization strategies, and the potential impact on airport security line management.