Task 5: Comparative Performance Analysis of Intra-process and Inter-process Communication Overheads

Student ID: 11314389
Name: Zheng Xu

November 26, 2025

Abstract

This report presents a quantitative analysis of communication overheads in concurrent systems, comparing shared memory (Threads) and datagram sockets (UDP). Based on a dedicated benchmark simulation performing 10,000 iterations, the results demonstrate that Thread-based communication ($0.0139\mu s$) is approximately **195.5 times faster** than UDP-based Inter-Process Communication ($2.7178\mu s$). The report also theoretically analyzes WebSockets, concluding they would incur the highest overhead due to TCP connection management and protocol framing.

1 Introduction

In the development of the relay race simulation, two distinct concurrency architectures were employed:

- **Intra-process (Threads):** Used in Tasks 1-3, utilizing `std::thread`, `std::mutex`, and shared memory for high-speed synchronization between athletes within a single process's virtual address space.
- **Inter-process (UDP IPC):** Used in Task 4, utilizing Windows Sockets (Winsock) to transmit race status updates to an external receiver process via the loopback network interface.

This document aims to quantify the performance cost (overhead) of decoupling the simulation and display logic using IPC compared to the shared memory approach.

2 Methodology

To accurately measure the communication overhead without the interference of the simulation's logic (printing, random delays), a specific benchmarking tool was developed.

2.1 Test Environment

The benchmark was executed on a Windows system using the C++11 standard library and Winsock2. Timing data was captured using `std::chrono::high_resolution_clock` to ensure microsecond-level precision.

2.2 Benchmark Design

The test consisted of **10,000 iterations** of message passing for each method:

1. **Thread Latency Test:** A thread acquires a mutex, writes a 21-byte string to a global variable, and releases the mutex. This simulates the overhead of the baton passing mechanism.
2. **UDP Latency Test:** The program creates a socket and uses `sendto()` to transmit the same 21-byte payload to `localhost` (127.0.0.1). This measures the cost of the OS network stack processing.

3 Quantitative Results

The experimental data collected from the benchmark execution is summarized in Table 1.

Table 1: Communication Overhead Comparison (10,000 Iterations)

Communication Method	**Total Time (μs)**	**Avg Latency / Msg (μs)**	**Performance Factor**
Threads (Shared Memory)	139	**0.0139**	1x (Baseline)
UDP (Inter-Process)	27,178	**2.7178**	~195.5x Slower

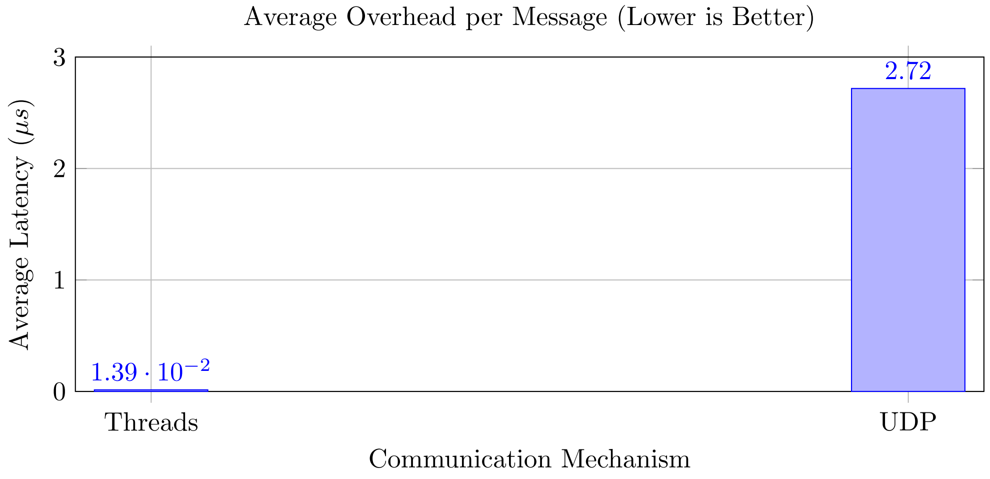

Figure 1: Visual comparison of latency. The disparity highlights the significant cost of kernel-mode operations in UDP compared to user-mode thread operations.

4 Analysis and Discussion

4.1 Threads vs. UDP

The results reveal a massive performance gap, with UDP being **195.5 times slower** than threads.

- **Threads (User Space):** Thread synchronization using `std::mutex` primarily occurs in user space. If there is no contention, the lock operation is compiled down to atomic CPU instructions (like Compare-and-Swap). Crucially, data is not copied; threads simply access the same memory address. This results in negligible latency ($0.0139\mu s$).
- **UDP (Kernel Space):** Despite running on the local loopback interface, UDP requires system calls. When `sendto()` is called, a context switch occurs from user mode to kernel mode. The data is copied from the application buffer to the kernel buffer, processed by the network stack (adding IP/UDP headers), and routed back. This OS intervention accounts for the $2.7178\mu s$ latency.

4.2 WebSockets Comparison (Theoretical)

Although not benchmarked directly, WebSockets would introduce significantly higher overhead than UDP.

- **Protocol Overhead:** WebSockets run over TCP, requiring a 3-way handshake for connection and acknowledgment packets (ACKs) for reliability, whereas UDP is connectionless and "fire-and-forget."
- **Data Framing:** WebSockets require HTTP headers for the initial handshake and specific framing for data packets, increasing the payload size compared to raw UDP.

5 Conclusion

For the relay race simulation, **Threads** are the optimal choice for internal logic (baton passing) due to their nanosecond-level latency. However, for the external display (Task 4), **UDP** is the correct architectural choice. Despite being 195x slower than threads, a latency of $2.7\mu s$ is imperceptible to the human eye and provides the necessary decoupling between the simulation engine and the user interface.