School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg
ELEN4017A – Network Fundamentals

Author: Natan Grayman *Student Number: 2344104*
Date: 9 May 2023

Abstract

This report illustrates the design and implementation of a minimal yet robust synchronization protocol that meets all the minimal requirements as well as an advanced login feature. Synchronization techniques, multithreading and socket programming were investigated and implemented. Successful latency checking, delay propagation and response time capturing were executed. This report consists of a literature review, a detailed description of the implementation, a description of the results and experimentation and a critical analysis.

1) Introduction

This report illustrates the design and implementation of a custom-made synchronization protocol through socket programming. The system designed consists of one multi-threaded server that allows for multiple clients to connect. A simple mathematics game has been designed to demonstrate the underlying synchronization. This report consists of a literature review, a detailed description of the implementation, a description of the results and experimentation and a critical analysis.

2) Literature Review

Synchronization is a fundamental aspect of distributed systems and networks, playing a vital role in ensuring coordination, consistency, and reliability. Likewise, synchronization is vital across all networking infrastructure, which permeates into every aspect of our lives, playing a key role in sectors such as telecom, utility, financial systems, and industrial networks [1].

Firstly, many clock synchronization algorithms have been designed, one of which is the Network Time Protocol (NTP). NTP is a widely installed clock synchronization protocol [2]. It employs a hierarchical architecture consisting of primary time servers, secondary servers, and clients defined as strata [2]. Furthermore, NTP utilizes timestamp exchange and error estimation algorithms to synchronize clocks with high accuracy [2].

In addition to NTP, a simplified version of the protocol called Simple Network Time Protocol (SNTP) also exists [2]. SNTP is designed for systems or devices that require basic time synchronization without the advanced features of NTP. While SNTP eliminates some of the complexities of NTP, it still follows the fundamental request-response mechanism and leverages the same timestamp exchange and error estimation algorithms to achieve time synchronization [2]. Therefore, SNTP serves as a lightweight alternative to NTP, offering a simplified solution for applications with less stringent time synchronization requirements [2].

Finally, the Precision Time Protocol (PTP) is a more advanced protocol designed to achieve sub-microsecond clock synchronization in local area networks (LANs) [3] . It uses a master-slave architecture, where the master clock distributes synchronization information to the slave clocks [3]. A best master clock selection algorithm is used to determine which system has the best clock

to use for synchronization [3]. Thereafter, a delay message structure consisting of a timestamp value and a correction field is used to measure the delay between the master and slave clocks, allowing for precise time synchronization by compensating for network latency and device processing delays [3].

3) Description of implementation.

i. System Overview

The system designed includes a server and client which correspondingly produce a synchronized message structure. The server's host and port are customizable, where multiple clients can respectively join. In order to access the game, each client needs to send the correct username and password. Once a minimum of two players have successfully entered the correct login details, the game process begins. The game process includes finding each client's respective latency, which in turn is utilized to calculate the required delay. Thereafter, the math question is sent to each player with the corresponding delay initiated. The cycle repeats for each new round of questions being sent to the players, effectively producing synchronization.

ii. Description of the synchronization algorithm

Initially, each client needs to send the server the correct username and password pair. However, in order to ensure synchronization, each player needs to start the game process at the same time instance.

Therefore, the Barrier class within the threading module was implemented, with specific usage of the wait() method[4]. Simply put, the wait() method causes a player's thread to block until the other player has also reached that point and called the wait() method [4]. This allows for a pseudo-lobby, whereby only once both players have successfully logged in, does the game start executing.
This ensures that the player's latencies are calculated at approximately the same time instance and therefore the corresponding delay is accurate and provides effective synchronization.

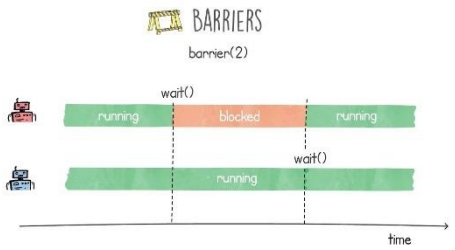

Figure 1: Visual representation of wait() method functionality [4].

The following flowchart represents the high-level logic of the synchronization protocol:

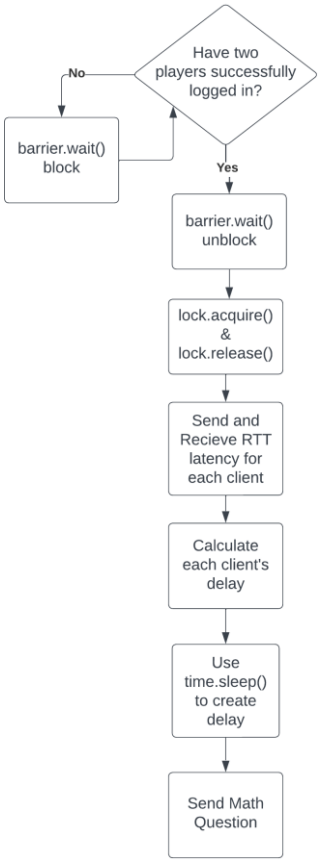

Figure 2: Flow chart of Synchronization logic of a round

Furthermore, when checking latencies, delays, and sending math questions, it is crucial to ensure that the actions of multiple threads are synchronized properly. Therefore, the locking functionality was implemented to ensure only one thread can access shared resources at a time ,avoiding conflicting modifications and race conditions [5]. Likewise, this serialization access to shared resources ensures data integrity, prevents data corruption, and maintains synchronization within the code [5].

Throughout the game, the latencies of each player can fluctuate. Thus, it was decided to implement the latency check before sending each math question in a round, ensuring that in each round the appropriate delay can be utilized. The round-trip time latency was dynamically computed and applied to determine the corresponding delay, reflecting the actual delay encountered by players in transmitting and receiving data. Additionally, the latencies are calculated by the server, which ensures that all players are subject to the same latency checks and that there are no discrepancies or inaccuracies in the latency measurements.

Once the latencies are captured by the server, each player's delay is dynamically calculated by finding the largest latency and using that to allocate delays to the other players. The sleep() functionality within the time class was implemented to create the resultant delay before sending the math question.

iii. Implemented Features

- A log in feature- username and password pair. The passwords are sent and stored as hashes.
- Lobby for players- the game waits for two players to join before starting the synchronization.
- Latency check- the client and server have corresponding latency check methods, used to effectively calculate round time latencies.
- Delay calculations- delays for each client's latency are dynamically calculated on the server side.
- Error checking on server side- Timeout functionality and error handling for keyboard malfunctions.
- Error Checking on client side– Attempt and timeout functionality and catching exceptions in keyboard input process.
- Each player's response time is dynamically calculated on the server side.
- Score – the server takes into account if the player has answered the question correctly and which player answered quickest and subsequently allocates a point.

iv. Brief overview of code structure

The code consists of a server, client, and math quiz class. The server file handles all aspects of the synchronization such as lobby creation, shared resource handling ,latency checking , delay propagation and response time calculations. The main function within the server code where the game and synchronization is implemented is called "client_thread_game". The client code consists of the corresponding message pair to that of the server and contains a comparable loop for rounds. Finally, the math_quiz class randomly generates the math questions and subsequent valid answers for the server to instantiate. Comments and numbering of each command and message pair has been placed in both the server and client code.

4) Results and experimentation

To check the latency resulting from the synchronized protocol, the internal results obtained were compared to that of Ngrok's

value given within the connection terminal.

Table 1: Showing Comparative latency values.

Host and Port	Internal Value (ms)	Ngrok Value (ms)
“0.tcp.in.ngrok.io:13192”	595	292
“0.tcp.in.ngrok.io:18312”	586	294
“0.tcp.eu.ngrok.io:15216”	358	180
“5.tcp.eu.ngrok.io:13696”	360	177
“2.tcp.ngrok.io:15305”	504	253
“8.tcp.ngrok.io:13063”	502	245
“0.tcp.au.ngrok.io:13889”	908	446
“0.tcp.au.ngrok.io:11168”	892	429
“0.tcp.sa.ngrok.io:14544”	321	159

The Ngrok terminal displays the one-way latency. Therefore, doubling the Ngrok values shown in the table produces the approximate Ngrok round-time trip latency, which as shown in the table is accurately captured by the synchronization protocol. An example of the Ngrok terminal, showing the latency metric is shown in logs.

Additionally, to experiment if each client was receiving the math question at the same time (with delay taken into account), the time was printed on the client’s side. The time instances were compared and successfully demonstrated, which can also be seen in the “Snapshot of game” image.

Likewise, rigorous experimentation was conducted throughout the project's lifecycle. This included Wireshark packet capturing (shown in logs) as well as simulating scenarios involving different players producing valid and invalid answers and comparisons of response times to a stopwatch.

5) Critical analysis

i. Discussion of features, success of each and future improvements

- The successful implementation of the login feature prevents unauthorized access to the game by restricting access to stored usernames and passwords only. Furthermore, the passwords are stored as hash value, providing security as a hack of the server would not disclose the raw passwords. However, improvements to security could have been made to only allow the password/username pair to be used once for each user and for the passwords sent to the server to be encrypted.
- The lobby waiting for the players was successfully implemented and produces the desired synchronization. However, this approach ultimately has the downside of not being able to accept players throughout the game and has to be predefined before the game begins. The amount of players joining needs to be predefined in order to set the barrier method. Therefore, a future improvement would include the first user also defining the game player mode, effectively setting the barrier/lobby.
- The latency check was successful throughout the project. Nevertheless, if a bad intentioned user manipulated the client/server code, they could essentially hack the system and rig the game in their favor. Therefore, a future improvement could include a NTP server which ensures that the time taken/received for the latency falls into a predefined threshold.
- The locking functionality effectively ensures data integrity,

prevents data corruption, and maintains synchronization within the code. However, the disadvantage of this approach is that the main shared resources had to be placed between the acquire() and release() lock methods, creating a large loop , which is not best coding practices. A future improvement thus would be researching how to effectively separate the code into smaller functions whilst maintaining the advantages of the lock functionality.

- The Score functionality was successfully implemented, however, the score is only sent at the end of each round and the users do not have access to a high score metric. Therefore, an improvement could include a server/ player message pair to check scoring metrics.
- Finally, the main challenge of the project was in the synchronization aspect. However, after exploring and implementing the synchronization, an improved and more generalized protocol could be developed. This would include advancing the command to response pairs and allow for any client to connect to the server.

ii. Unimplemented features

- The implementation of simultaneous connections of three or more users was not completed. Additionally, joining an existing game was not implemented.
- The feature where the first user connecting decides when to start the game was not implemented. However, an additional message/answer pair could be added which changes the 5 seconds delay functionality already initiated.

6) Conclusion

In conclusion, synchronization techniques, multithreading and socket programming were investigated and implemented. Successful latency checking, delay propagation and response time capturing were executed. Ultimately, a minimal yet robust synchronization protocol has been effectively designed and implemented, meeting all the minimal requirements as well as an advanced login feature.

References

[1] Nokia, “Network synchronization,” 2020. [Online]. Available: https://www.nokia.com/networks/network-synchronization/. [Accessed 1 May 2023].

[2] P. Kirvan and J. Scarpati, “Network Time Protocol (NTP),” TechTarget, February 2022. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/Network-Time-Protocol. [Accessed 1 May 2023].

[3] P. Krzyzanowski, “Precision Time Protocol,” 15 February 2021. [Online]. Available: https://people.cs.rutgers.edu/~pxk/417/notes/ptp.html. [Accessed 1 May 2023].

[4] J. Cutajar, “Python's Thread Barriers in 8 mins,” Youtube, 11 February 2021. [Online]. Available https://www.youtube.com/watch?v=3Y6l76AS4l4. [Accessed May 2023].

[5] Educative Answers Team, “What are locks in Python?,” Educative, 2023. [Online]. Available: https://www.educative.io/answers/what-are-locks-in-python. [Accessed May 2023].

Appendix

Number	Server Command	Message Format	Client Response
1.	Welcome Message and Username Request	“Welcome to the Synchronized Math Multiplayer game! Enter Username: “	Natan or Benjy
2.	Receive and check respective client Username	-	-
3.	Request Password	“Enter Password: “	START or ENTER
4.	Password Receive and Check	-	-
5.	Send Correct / Incorrect Message	"Correct! Game will begin in 5 seconds once 2 players join" Or “Incorrect Username or Password!”	-
6.	Send Latency Check message	“Ping”	Ping (Automated)
7.	Send Math Question:	“Num1 +/* Num2”	Any Number/Word
8.	Receive answer and check	-	-
9,	Updated Score message	“ Player {number} score: {score value} “	-