COMP2017 9017 Assignment 2

Due: 23:59 20 May 2025

This assignment is worth 10% of your final assessment

Contents

1 Assignment 2 - [ZOIT Docs] - 10%

We strongly recommend reading this entire document at least twice. You are encouraged to ask questions on Ed after you have first **searched**, and checked for updates of this document. If the question has not been asked before, make sure your question post is of type **"Question"** and is under **"Assignment" category** → **"P2"** . Please follow the staff directions for using the question template.

Note, **your Git commit messages matter**. Make sure you enter informative information into your commit messages as per the accompanying Edstem slide. Also, use `.gitignore` to avoid committing garbage files. There is a guide to both on the accompanying Ed slide titled Git Usage Guideline, with a required formatting guide you **must** follow.

It is important that you continually back up your assignment files onto your own machine, flash drives, external hard drives and cloud storage providers (as private). You are encouraged to submit your assignment regularly while you are in the process of completing it.

Full reproduction steps (seed, description of what you tried) **MUST** be given if you are enquiring about a test failure or if you believe there is a bug in the marking script.

Academic Integrity and Compliance Statement

This assignment will be submitted in Ed Lessons, under **Assignments - P2**.

In addition to the Academic Integrity and Compliance statements below, an Academic Declaration appears in your assignment in Ed. All statements apply.

Academic Integrity

While the University is aware that the vast majority of students and staff act ethically and honestly, it is opposed to and will not tolerate academic integrity breaches and will treat all allegations seriously.

Further information on academic integrity, and the resources available to all students can be found on the academic integrity pages on the current students website:
`https://sydney.edu.au/students/academic-integrity.html`

You may only use generative artificial intelligence (AI) and automated writing tools in assessment tasks if you are permitted to by your unit coordinator. If you do use these tools, you must acknowledge this in your work, either in a footnote or an acknowledgement section. For information on acknowledging AI please refer to the guidance in the AI in Education Canvas site.

We use Turnitin, which includes AI detection, to detect potential instances of plagiarism or other forms of academic integrity breach. If such matches indicate evidence of plagiarism or other forms of academic integrity breaches, your teacher is required to report your work for further investigation.

Further information on research integrity and ethics for postgraduate research students and stu-

dents undertaking research-focused coursework such as Honours and capstone research projects can also be found on the current students website:
https://sydney.edu.au/students/research-integrity-ethics.html

Compliance Statement

In submitting this work, I acknowledge I have understood the following:

1. I have read and understood the University of Sydney's Academic Integrity Policy.
2. The work is substantially my own and, where any parts of this work are not my own, I have indicated this by acknowledging the source of those parts of the work and clearly indicated any quoted text by quotation marks or indentation according to accepted style standards.
3. I have acknowledged any assistance provided in preparing the work including the use of copy-editing, proof-reading, and automated writing and drawing tools (including artificial intelligence (AI), reference generators, translation software, grammar checkers, but not spell checkers).
4. The work has not previously been submitted in part or in full for assessment in another unit unless I have been given permission by my unit of study coordinator to do so.
5. The work will be submitted to similarity detection software (Turnitin) and a copy of the work will be retained in Turnitin's paper repository for future similarity checking.
6. Engaging in plagiarism or academic dishonesty in coursework will, if detected, lead to the University commencing proceedings under the Academic Integrity Policy and the Academic Integrity Procedures.
7. Engaging in plagiarism or academic dishonesty in research-focused work will lead to the University commencing proceedings under the Research Code of Conduct and the Academic Integrity Procedures.
8. Engaging another person to complete part or all of the submitted work will, if detected, lead to the University commencing proceedings against me for potential student misconduct under the University of Sydney (Student Discipline) Rule.
9. That the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

You are advised to keep copies of your assignment submission, drafts and any other research materials as evidence of your research and writing process. If you have used AI in the completion of your assignment, you should keep copies of the AI outputs.

2 Introduction

We are developing a new service aimed at revolutionising document editing, with the ambitious goal of outperforming competitors such as Word, Notion and Google Docs. To achieve this, we must implement a server-client architecture where a central server hosts the document and coordinates all client interactions. Multiple authorised clients will be able to send commands to the server to edit the document concurrently.

The document format will be a modified version of Markdown. Content will be stored as plain text using only ASCII characters, simplifying compatibility and transmission.

The server will always maintain the most up-to-date and accurate version of the document. Note that if your data structure is a simple character array, inserting characters into the middle of the document would require shifting all subsequent characters, resulting in inefficiency. Therefore, you are advised to use a more suitable data structure, such as a linked list, to allow for efficient editing operations.

The structures and basic operations can be found in the code scaffold You **must** adhere to the function signatures for the marking script to correctly assess your code.

3 Task

Implement a simulated collaborative document editing system in C, consisting of two programs:

1. **`server`** — the authoritative document host
   - Listens for client connections over named pipes.
   - Maintains the document in an efficient in-memory structure.
   - Applies editing commands (`INSERT`, `DEL`, `BOLD`...) atomically and in the order received.
   - Broadcasts updates to all connected clients.
   - Handles client registration, graceful disconnects, and error reporting.
2. **`client`** — a console-based editor
   - Connects to the server's named pipe endpoint.
   - Sends user commands to read or modify the document.
   - Receives update notifications and applies them to its local view.

Deliverables & Constraints

- Source files: `server.c`, `client.c`, plus any helpers.
- Must compile on Edstem Linux environment using with no warnings (`-Wall -Wextra`).

4 Set-up

Client–Server Connection Protocol

1. Start the server by running it with the following command. `<TIME INTERVAL>` is the time interval in milliseconds at which the versions of the document updates.

   ```
   ./server <TIME INTERVAL>
   ```

2. The server starts and immediately prints its Process ID (PID) to stdout:

   ```
   Server PID: <pid>
   ```

3. A client joins by invoking:

   ```
   ./client <server_pid> <username>
   ```

4. Upon starting, the client immediately sends `SIGRTMIN` to the server's PID and then blocks, waiting for `SIGRTMIN+1`. The staff client will timeout after 1 second if no signal is received[1][2]. Student will not need to handle these conditions.

5. On receiving `SIGRTMIN`, the server:

   5.1. Spawns a new POSIX thread to handle that client.

   5.2. Within that thread, creates two named FIFOs to establish bi-directional communication:

   - `FIFO_C2S_<client_pid>` (Client-to-Server)
   - `FIFO_S2C_<client_pid>` (Server-to-Client)
   - If either already exist, clean them up first and then create them.

   5.3. Sends `SIGRTMIN+1` back to the client.

6. The client, upon receiving `SIGRTMIN+1`, opens:

   - `FIFO_C2S_<client_pid>` for writing, and
   - `FIFO_S2C_<client_pid>` for reading,

   and writes its username into `FIFO_C2S_<client_pid>`.

[1]The timeout should not block the server thread, or the server process
[2]Timeout is approximate

Authorisation and Role Communication

7. After receiving the client's username (as a newline-terminated ASCII string), the server must:

   (1) Check the provided username against the `roles.txt` file to determine if the user is authorised and what permission level they possess.

   (2) If the username is found, the server will send the following through `FIFO_S2C_<client_pid>`:

   - The server must send the user's role (permission) as a single, lowercase ASCII string, terminated by exactly one newline character (`\n`). The role string must be one of the following exact values: `write`, or `read`.
   - After sending the role, the server must send the full document contents, followed immediately by entering the edit-command loop.

   (3) If the username is not found:

   - The server must send a rejection message: `Reject UNAUTHORISED`.
   - Wait for 1 second on this thread alone, not blocking the whole process[3].
   - Then close both FIFOs, unlink them (i.e., delete them from the filesystem), and terminate the server thread handling that client.

`roles.txt` Format

The `roles.txt` file must reside in the same directory as the running server executable. Each line of the file contains a username followed by a permission level, separated by one or more spaces or tab characters. The valid permission levels are:

- `write` — permission to edit the document.
- `read` — permission to view the document but not modify it.

Example format:

```
bob   write
eve   read
ryan  write
```

Leading and trailing whitespace around usernames and roles must be ignored. Comparisons must be case-sensitive.

Document Transmission Protocol

After sending the role string, the server must transmit the full document contents to the client over the named pipe `FIFO_S2C_<client_pid>`, following these rules:

[3]Wait time is approximate

- The document must be serialised as plain ASCII text, preserving all internal newlines (‘`\n`‘) exactly as they exist in the server’s current document state.
- The document’s current version will be sent as a newline terminated number, this is guaranteed to be smaller than uint64.
- The document length in bytes will be first send as an integer followed by a newline delimiter. The size of the document in bytes is guaranteed to be smaller than uint64.
- The entire document must be sent as a length prefixed byte stream.
- The document content must be immediately readable by the client.
- Transmission must continue after the entire document content is written to the pipe.

Overall example payload:

```
write\n // role
0\n // version number
34\n // document length
This is the document.\n
Hello world.
```

Signal Handling Hints (Optional)

Whilst the following functions are not allocated marks, correct behaviour is marked.

- Use `sigaction()` to register a handler for `SIGRTMIN` and use `sigqueue()` for sending signals if you want to include additional data (e.g., client PID).
- Use `sigemptyset()`, `sigaddset()`, and `sigprocmask()` to block `SIGRTMIN` and `SIGRTMIN+1` before creating the thread. This prevents signal loss during handler setup.
- Use `sigwait()` in the client to synchronously wait for `SIGRTMIN+1`—this avoids race conditions and allows clean blocking until the server responds.
- Realtime signals (like `SIGRTMIN`) are queued by the kernel and not merged, making them safer than standard signals (like `SIGUSR1`) in high-concurrency environments.

5 Structure

Both the server and the client need to store a copy of the document. The server’s copy will always be the most up-to-date and the "true" state of the document, whilst the client’s copy will be updated as changes are broadcasted from the server.

This is necessary to prevent constantly transmitting the entire document over the network, which slows down the app.

The following table contains the formatting that is supported by the document.

Element	Markdown Syntax
Heading 1	`# H1`
Heading 2	`## H2`
Heading 3	`### H3`
Bold	`**bold text**`
Italic	`*italicised text*`
Blockquote	`> blockquote`
Ordered List	`1.  First item`
	`2.  Second item`
	`3.  Third item`
Unordered List	`- First item`
	`- Second item`
	`- Third item`
Code	`` `code` ``
Horizontal Rule	`---`
Link	`[title](https://www.example.com)`

Table 1: Markdown Syntax Reference

Formatting Notes: Note although the following rules must be adhere to when adding formatting, markdown is just plain text and the added formatting will not be treated differently to plain text.

- **Space Requirement for Certain Tags:** The following Markdown tags must be followed by a space character (‘ ‘) immediately after the tag to be correctly parsed:
  - Heading markers: `#`, `##`, `###`
  - Blockquote marker: `>`
  - List item markers: `1.`, `2.`, `3.`, `-`

  If a space is omitted after these tags, the element will not be recognised as correct formatting and any of our (hypothetical) markdown renderer would not be able to correctly render it.
- **Block-level Elements:** The following elements are classified as block-level:
  - Headings (Heading 1, Heading 2, Heading 3)
  - Blockquotes
  - Ordered Lists
  - Unordered Lists

  - Horizontal Rules

- **Newline Requirements for Block-level Elements:** When inserting a block-level element, the editor must:
  - Ensure there is a newline character (‘`\n`‘) immediately preceding the inserted block element, unless it is inserted at the start of the document.
  - If the insertion point is not already at the beginning of a line (i.e., the previous character is not a newline), automatically insert a newline before the block element.
  - The horizontal rule must have newline after as well. If one does not exist, automatically insert one.
- **Inline Elements:** Inline elements — specifically Bold, Italic, Code, and Links — do not require preceding or trailing newlines. They may appear within an existing line of text without requiring structural separation.

6 Client-Server Synchronisation

Since both the server and clients maintain a copy of the document state, it is critical to ensure consistent synchronisation of all changes across the system.

All modifications to the document must originate as requests from clients, sent to the server in the form of commands.

At intervals, the server will issue a new version of the document. It will broadcast all changes made since the previous version to all connected clients, ensuring synchronisation.

To maintain consistency, the server will only accept commands that target the current version of the document. Any command referencing an outdated version will be rejected.

It is guaranteed that no two commands will be received at the EXACT same time and operate at overlapping cursor positions. Targeting overlapping cursor positions of the same version of the document is allowed.

7 Functionality

7.1 Part 1: Single Client

Commands are issued by the client and sent to the server for processing. Each command affects the document and must specify a cursor position, which is defined as the number of characters from the beginning of the document to the intended location.

To simulate user interaction with a GUI, users may type commands directly into the standard input of the client or server.

All positions are interpreted on the version they reference, and if valid, changes are reflected in the next broadcasted version. Once edits are made, future edits will need to have their cursor positions adjusted if necessary.

Command Format Constraints

All commands transmitted between the client and server must satisfy the following properties:

- The command must be composed exclusively of printable ASCII characters (byte values 32–126), with the exception of the final newline character.
- Each command must be terminated by exactly one newline character ‘`\n`‘.
- The total size of the command, including the terminating newline character, must not exceed 256 bytes.
- Commands that do not satisfy these constraints (e.g., missing newline, exceeding 256 bytes, or containing non-ASCII characters) are considered invalid and must be rejected by the receiving side.
- All command arguments are separated by a single space character.
- For commands with a cursor range, end position should not equal to or precede the start position.
- The following commands are all also valid terminal commands.
- INSERT and DEL commands that causes formatting errors or broken formatting can happen but do not need to be handled in any way, you can just accept them.

Example:

```
0   1   2   3   4   5   6   7   8
|   |   |   |   |   |   |   |   |
  D   o   c   u   m   e   n   t
```

Position 0 is **before** the first character `D`.
Position 8 is **after** the last character `t`.

Editing Commands

- `INSERT <pos> <content>`; Inserts textual content at the specified cursor position and document version. Newlines are NOT permitted in this command.
- `DEL <pos> <no_char>`; Deletes `<no_char>` characters starting from the specified cursor position. If the deletion flows beyond the end of the document, truncate at the end of the document.

Formatting Commands

Make sure you adhere to the formatting notes 5!

- `NEWLINE <pos>`; Inserts newline formatting at the position.
- `HEADING <level> <pos>`; Inserts a heading of the given level (1 to 3) at the specified position.
- `BOLD <pos_start> <pos_end>`; Applies bold formatting from the start to end cursor position.
- `ITALIC <pos_start> <pos_end>`; Applies italic formatting from the start to end cursor position.
- `BLOCKQUOTE <pos>`; Adds blockquote formatting at the specified position.
- `ORDERED_LIST <pos>`; Adds ordered list formatting at the position. Any surrounding list items will be renumbered appropriately (e.g., 1, 2, 3...). For example, if an ordered list is added to the end, it'll find the previous number and set the new item to previous number +1. If an ordered list is added in between existing ordered list blocks, it'll be the previous item + 1 and all list items will also be + 1. You must handle newlines which can break ordered lists into two. Delete that breaks formatting does NOT need to be handled.
- `UNORDERED_LIST <pos>`; Applies unordered list formatting at the specified location.
- `CODE <pos_start> <pos_end>`; Applies inline code formatting to the given range.
- `HORIZONTAL_RULE <pos>`; Inserts a horizontal rule and newline at the cursor position.
- `LINK <pos_start> <pos_end> <link>`; Wraps the text between positions in `[]` and appends the link in `()` to create a Markdown-style hyperlink.

7.1.1 Ordered List

Consider a list:

```
1. tomato
2. cheese burger
3. asparagus
```

If I send a command ORDERED_LIST targeting the cursor position between cheese and burger, the list becomes:

```
1. tomato
2. cheese
3. burger
4. asparagus
```

Also if I send a NEWLINE command to the same cursor position to the original list, it becomes:

```
1. tomato
2. cheese
burger
1. asparagus
```

If I delete the entire second list item from the original list, the list becomes:

```
1. tomato
2. asparagus
```

Sending a NEWLINE command to the cursor position between 2 and . to the first list is NOT permitted.

List formatting is guaranteed to not be sent if:

- The target cursor position is at or next to a list formatting `2. ` or `- `.
- The target is a list item formatted as a different type. For example sending ORDERED_LIST between love and gaming `- I love gaming.`

Nested lists are not allowed and guaranteed to not happen. DELETE that causes formatting problems do NOT need to be considered.

Also note, ordered list numbering is limited 1 to 9.

INSERT will not insert anything that looks like ORDERED_LIST formatting.

Metadata and Other Commands

- `DISCONNECT`; Disconnects the client from the server. When the user types DISCONNECT to the client stdin, it'll send DISCONNECT to the server. Then the server will close and clean up the pipes. The client will detect this and exit gracefully, freeing all resources.

Client Debugging Commands

- `DOC?`; Prints the entire document to the client's terminal.
- `PERM?`; Requests and displays the client's current role/permissions from the server.
- `LOG?`; Outputs a full log of all executed commands in order.

Server Responses

- `SUCCESS`; Indicates that the client's command was successfully executed.
- `Reject <reason>`; Indicates the command was rejected, with a message detailing the reason. See Section 7.4.

These responses are NOT sent immediately, but only sent as part of the broadcast message.

Server Debugging Commands

- `DOC?`; Prints the current document state to the server terminal.
- `LOG?`; Outputs a full log of all executed commands in order.

Output Formatting

If the command required output onto the terminal, print the output after the newline, for example:

```
DOC?
This is the document
```

This is an example of the `LOG?` command and its output:

```
LOG?
VERSION 1
EDIT <user> <command> SUCCESS  // first command
EDIT <user> <command> Reject <reason>  // second command
EDIT <user> <command> SUCCESS // third command
END
VERSION 2
EDIT <user> <command> SUCCESS  // first command
EDIT <user> <command> SUCCESS  // second command
...
EDIT <user> <command> SUCCESS // nth command
...
END
VERSION <N>
...
END
```

Not that the message for each version is the same as the message broadcasted to the clients at that version's update.

7.2 Part 2: Multiple Client

If your collaborative document contains multiple users, multiple edits may occur simultaneously. To handle this, the server spawns a separate thread for each connected client. Commands are queued within each thread. At the `TIME INTERVAL` update interval, the main server will collect all the commands from the thread queues based on timestamp order and process the changes in order. The server adds the timestamp when a command arrives.

Handling Deleted Text and Cursor Position Adjustments All the edits target the broadcasted document, i.e. what the client was looking at when sending the command. Generally, no matter the execution order of the commands queued for a version, the final document will look the same. However for commands with overlapping cursors, this is not true and executing the commands in the order of timestamp arrival will be critical.

If a section of text is deleted and a future command (from any client) operating on the same version is to run at a cursor position within that deleted region, apply the following rules:

- For commands with a **single cursor position**, replace the cursor position with the location where the deletion began.
  Example: If positions 3 to 9 were deleted in an earlier command on the same version, an insertion at position 5 will now occur at position 3.
- For commands requiring **starting and ending cursor positions**:
  - If **both** positions fall within the deleted region, it's a DELETED_POSITION error.
  - If **only one** position is within the deleted region, the cursor position within the deleted region will be adjusted to the closer of the deleted edges to the valid cursor position.

If a section of text is formatted or inserted and a cursor region which overlaps this change is deleted by a future command targeting the same version, apply the following rules:

- The deleted will deleted the original intended deletion and the new insertion or formatting is untouched.

  Consider a sentence: `"Hello world"`

  - Insert at cursor position 5: `" beautiful"`
  - Then delete starting at position 0, deleting 100 characters

  The final result is:

  ```
  " beautiful"
  ```

Example: Basic Insert

This example demonstrates a soft insert operation followed by a commit.

1. A new document is initialized.
2. Two soft insertions are made at version 0:
   - Insert `"World"` at position 0.
   - Insert `"Hello "` also at position 0, effectively pushing `"World"` to the right.
3. The document is flattened without committing the changes, and it correctly returns an empty string since no version has been committed yet.
4. After incrementing the version (committing version 0), the document is flattened again and the output is expected to be:

```
Hello World
```

Example: Basic Delete

This example shows how deletions behave before and after a commit.

1. A new document is initialized.
2. The string `"Hello World"` is inserted at version 0 and committed.
3. A delete operation is issued at version 1 to remove 6 characters from index 5 (i.e., deleting the substring `" World"`).
4. Flattening the document at version 1 (before committing the deletion) returns:

```
Hello World
```

5. After committing the deletion (incrementing the version), flattening the document again returns:

```
Hello
```

Example: Multiple Insertions Within a Sentence

This example illustrates the behavior of inserting text both within and at the end of an existing sentence.

1. A new document is initialized.
2. The string `"I love COMP2017"` is inserted at version 0 and committed.

3. Two soft insertions are made at version 1:
   - Insert `"really "` between `"I "` and `"love"`, resulting in `"I really love COMP2017"`.
   - Insert `" so much"`, directly after `"COMP2017"`.
4. Before committing version 1, flattening the document will still return:

```
I love COMP2017
```

5. After committing the insertions (by incrementing the version), flattening the document will return the final expected sentence:

```
I really love COMP2017 so much
```

Example: Deleting Items from a Numbered List

This example shows how consecutive deletions modify a structured list and how renumbering is reflected in the final output.

1. A new document is initialized.
2. The following string is inserted at version 0 and committed:

```
Things to buy
1. yoyo
2. switch
3. ps5
4. COMP2017 textbook
5. gaming chair
```

3. At version 1, two delete operations are performed:
   - Delete the line corresponding to `2. switch`.
   - Delete the line corresponding to `3. ps5`.
4. Before committing, flattening the document still yields the original list:

```
Things to buy
1. yoyo
2. switch
3. ps5
4. COMP2017 textbook
5. gaming chair
```

5. After committing the deletions (by incrementing the version), the flattened document reflects the updated list with renumbering:

```
Things to buy
1. yoyo
2. COMP2017 textbook
3. gaming chair
```

Versioning System To address the issue of commands arriving in different orders, a versioning system is used. This system keeps track of the user's originally intended edit position and adjusts cursor positions accordingly.

- A new broadcast is issued every `TIME INTERVAL`.
- Version numbers start from `0` and increment by `1`.
- Version number only increments if at least one edit was made.
- Even if no changes were made, it'll broadcast just the version number.
- All of the following messages are newline terminated.
- The payload must be closed with an `END` message.
- The following message is broadcast to all users when a new document version is issued:

```
VERSION <version_number>
EDIT <user> <command> SUCCESS  // first command
EDIT <user> <command> Reject <reason>  // second command
...
EDIT <user> <command> SUCCESS // nth command
END
```

This ensures every client can sync their local copy of the document to the most up-to-date version.

Edit Validation The server will only accept edits on the **current** document version. Any attempt to edit an older version will be rejected.

7.3 COMP9017

Implement the server such that both edits on version N and N-1 are accepted.

You must document and justify your program's behaviour in a file named `README.md`. In this, you must justify, AND demonstrate with testcases with sufficient complexity (incl. edge cases), why your implementation is well-defined and makes intuitive sense.

COMP9017 students have their marks scaled by 0.9. This component counts for 10%.

7.4 Rejections

The server may reject a client's command for several reasons. In such cases, it will respond with a rejection message in the format:

```
Reject <reason>
```

Below is a list of possible rejection reasons:

UNAUTHORISED The client does not have sufficient permissions (i.e., `write`) to execute the specified command.

INVALID_POSITION One or more provided cursor positions are outside the bounds of the current document. For cursor ranges, the end position is equal to or lower than the start position.

DELETED_POSITION The specified cursor position points to text that has already been deleted.

OUTDATED_VERSION The version targeted by the command is no longer accepted by the server. Commands must reference either the current (or previous for COMP9017) document version.

INVALID_POSITION takes precedence over DELETED_POSITION, so if both are true for an edit, return INVALID_POSITION.

In your `markdown.h` functions, use the following return codes.

```
    SUCCESS = 0,
    INVALID_CURSOR_POS = -1,
    DELETED_POSITION = -2,
    OUTDATED_VERSION = -3,
```

8 Tear Down

If there no clients connected to the server, you can send QUIT via the terminal to the server to shut it down.

In this case it'll gracefully close the pipes, clean up all the memory, and free up all related resources. It'll also save the document in a `doc.md` in the same directory. If such a file already exists, overwrite it.

If any clients are still connected, it'll print

```
    QUIT rejected, <number> clients still connected.
```

. Example:

```
QUIT
QUIT rejected, 3 clients still connected.
```

9 Marking

Automated marking will be conducted through the following two categories:

9.1 Markdown Unit Tests

- A suite of tests will be run to verify the correctness of all Markdown functionality specified in Section 7.
- These tests include insertion, deletion, formatting commands (e.g., headings, bold, italic, blockquotes, lists), and version handling.
- The tests will check for correct document state updates and proper command responses.
- Submissions will also be checked for memory safety to ensure there are no memory leaks.

9.2 Staff Client-Server Interoperability Tests

- A reference `staff client` and `staff server` have been implemented to verify protocol compliance.
- Your **server** must correctly interoperate with the **staff client**.
- Your **client** must correctly interoperate with the **staff server**.
- Interoperability will be verified using input-output based automated tests.
- Strict adherence to the communication protocol (signals, FIFO creation, document transmission format, command structure) is essential for passing these tests.

10 Safe Assumptions

Below is a list of safe assumptions that you are guaranteed to not be tested on.

- Malformed commands which do not follow the given format
- INSERT containing ordered list like formatting
- DELETE or NEWLINE breaking ordered list formatting (i.e. removing just the "." from "`\n1. `")
- Nested lists
- Two commands sent at the exact same time with the same timestamps
- 10 or more ordered list items

11 Short Answer Questions

1. How do you ensure that you do not have to constantly copy and shift large chunks of memory in your code?

2. If multiple edits are made to the same version of the document, how do you make sure the cursor positions where any edits are made are what the client originally intended.

3. How did you (or would you) test multiple clients interacting with the server at the same time? Does your test case(s) target race conditions or concurrency-related edge cases?

4. How did you efficiently ensure that all the formatting of ordered lists were correct? If not, how did you repair them?

5. How can select or epoll change our implementation? Are threads still necessary?

6. Is our program more efficient or runs better than a single process single thread model? Does non-blocking help? If so in what ways?

12 Restrictions

To successfully complete this assignment you must follow the restrictions below. Submissions breaking these restrictions will receive a deduction of up to 1 mark per breach.

- The implementation of the server must make use of one POSIX thread per client.
- The code must entirely be written in the C programming language.
- Must use dynamic memory for document.
- Free all dynamic memory that is used.
- POSIX library is needed for POSIX threads, however other libraries such as `regex.h` is NOT allowed.
- NOT use any external libraries other than those in `libc`.
- NOT use VLAs.
- NOT use shared memory.
- NOT use regex
- NOT have unclean repositories. This means no object, executable, or temporary files **for any commit** in the latest repository. A clean final commit is NOT good enough. You SHOULD use `.gitignore` to avoid committing garbage files.
- Only include header files that contain declarations of functions and extern variables. Do not define functions within header files.
- Any and all comments, including git commit messages, must be written only in the English language.
- Must use meaningful commits and meaningful comments on commits. [4]
- Other restricted functions may come at a later date.
- NOT manually use return code 42, reserved by ASAN.

[4]"forcing the seed of a test case" does not count as valid commit. Must cite reason and identified failure.