News Agency Publishing and Subscribing System: Algorithm Explanation and Output Tracing

**Introduction:**

This report details the implementation of a multi-threaded News Agency Publishing and Subscribing System, developed in the C programming language using POSIX threads **(pthreads)**. The system manages the publication and subscription of news items, effectively addressing the producer-consumer and reader-writer synchronization problems. It employs a circular buffer to store up to MAX\_NEWS (20) news items, each containing an ID, category, title, content, and timestamp. Writers (producers) add or edit news, readers (consumers) view news by category or all at once, and subscribers manage news, including removing old items to free space. Synchronization is achieved through semaphores and mutexes, ensuring safe concurrent access to the shared news database. This report explains the system’s architecture, synchronization mechanisms, algorithms, and provides hypothetical output tracing to demonstrate its behavior, including how it handles edge cases. The code snippets are formatted to resemble a dark-mode terminal with white text for visibility.

**Program Working:**

The system comprises the following components:

* **News Database (Shared Resource):** A circular buffer storing up to 20 news items, defined by MAX\_NEWS. Each item includes an ID, category (e.g., BREAKING, POLITICS), title, content, and timestamp. The buffer is persistent, with news saved to and loaded from a file (news\_database.txt).
* **Writers (Producers):** Threads that add or edit news items, ensuring space is available in the buffer before writing.
* **Readers (Consumers):** Threads that read news by category or view all news, ensuring news is available before reading.
* **Subscribers:** Similar to readers but with the ability to remove the oldest news to free space for new items.
* **News Agency:** A thread providing a manual interface for adding and editing news items.

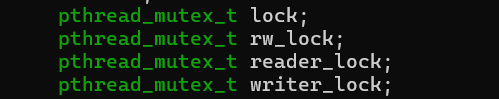
The system uses synchronization primitives to manage concurrent access to the shared news database, ensuring data integrity and preventing race conditions.

**Synchronization Mechanisms:**The system employs the following synchronization primitives to manage access to the news database:

1. Semaphores:
   1. **free\_slots**: Initialized to MAX\_NEWS (20). Writers wait on this semaphore to ensure there is space in the buffer before adding news.
   2. **used\_slots**: Initialized to 0. Readers wait on this semaphore to ensure there is at least one news item available to read.

`

1. Mutexes:
   1. **lock**: A general-purpose mutex for protecting critical sections of the news database structure, such as updating start, end, or num\_news.
   2. **rw\_lock**: A reader-writer lock that allows multiple readers to access the database simultaneously or grants exclusive access to a single writer.
   3. **reader\_lock**: Manages the count of active readers (num\_readers), ensuring proper acquisition and release of rw\_lock.
   4. **writer\_lock**: Serializes writers, ensuring only one writer can modify the database at a time.



These primitives work together to address the producer-consumer and reader-writer synchronization problems, as detailed in the following sections.

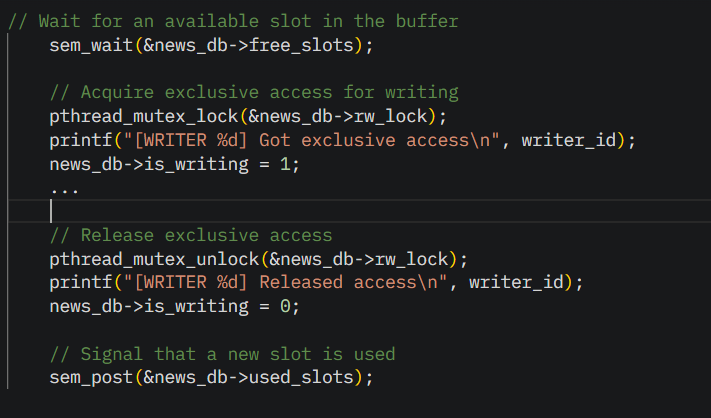
**Producer-Consumer Problem Handling:**  
The producer-consumer problem involves producers adding items to a shared buffer and consumers accessing those items, with synchronization to prevent buffer overflow or underflow. In this system, writers act as producers, adding news items to a circular buffer, while readers act as consumers, reading news without removing it. The buffer has a fixed size of MAX\_NEWS (20), and when it reaches WARN\_THRESHOLD (18), the system automatically removes the oldest news item to make space for new additions.

Code Snippets**:**

The following code snippets illustrate how the system implements the producer-consumer mechanism, formatted to resemble a terminal in dark mode with white text for visibility:

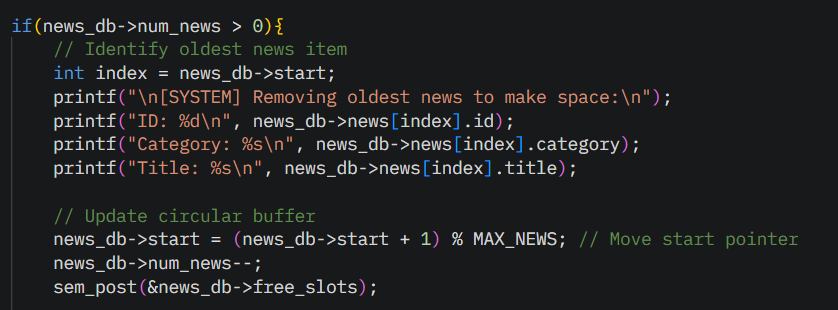
* **Adding News (Producer):** This code ensures writers only add news when there is space in the buffer (wait for free slot).

In add\_news function:

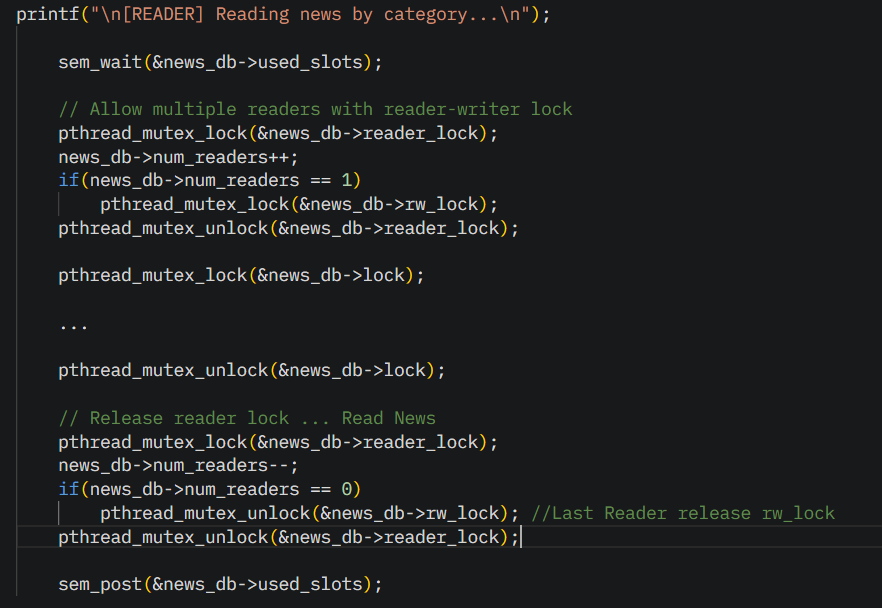


* **Removing Oldest News:** This code handles the case when the buffer is full or nearing capacity.

In remove\_oldest\_news function:



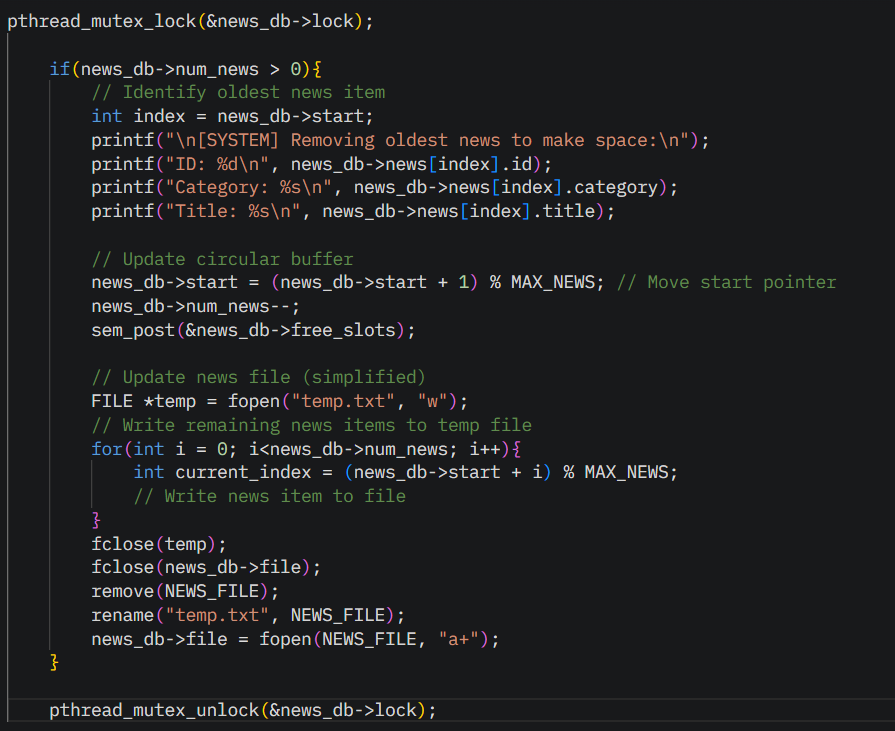
1. **Reading News (Consumer):** This code ensures readers only read when news is available.



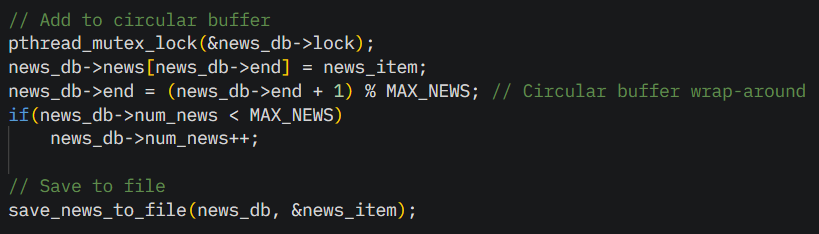
**Circular Buffer Operation:**  
The circular buffer is implemented as an array of size MAX\_NEWS (20), with start and end pointers tracking the oldest and newest news items, respectively. The operation is as follows:

* + - **Adding News:**
* Check if num\_news >= WARN\_THRESHOLD (18). If true, call remove\_oldest\_news to increment start (modulo 20) and decrease num\_news, freeing a slot.
* Wait on free\_slots to ensure there is space in the buffer.
* Add the news item at index end, increment end (modulo 20), and increase
* num\_news if less than 20.
* Save the news item to the file (news\_database.txt) for persistence.

In remove\_oldest\_news function:

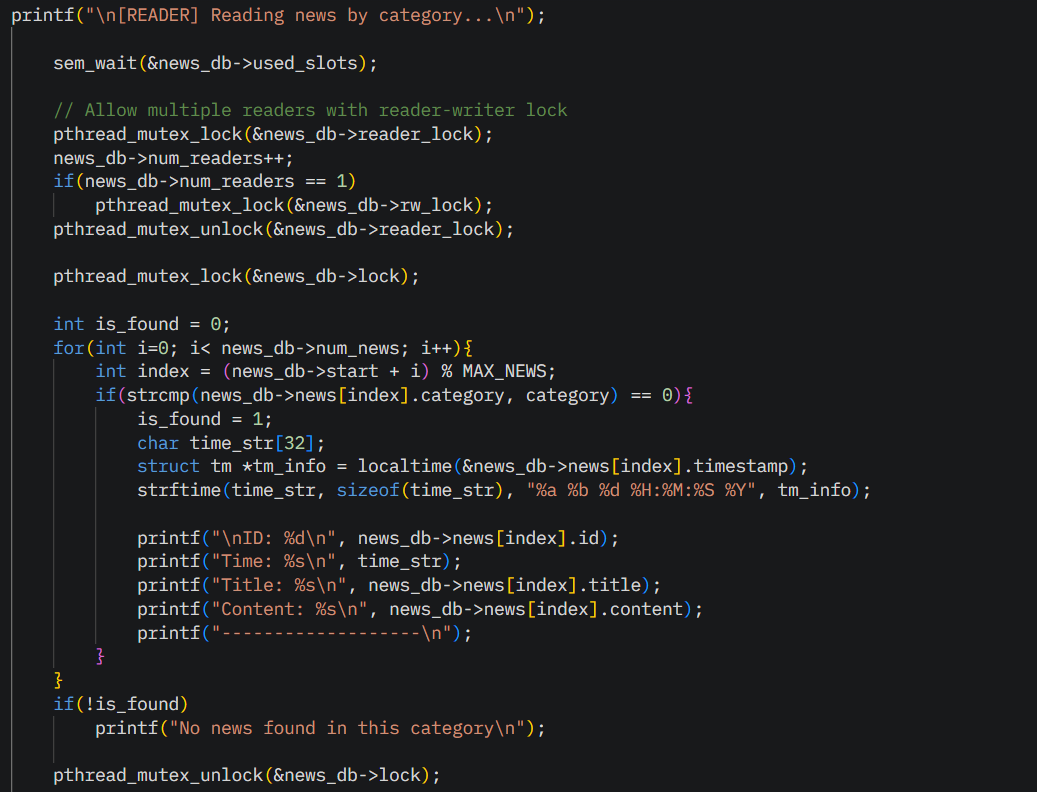


In add\_news function:



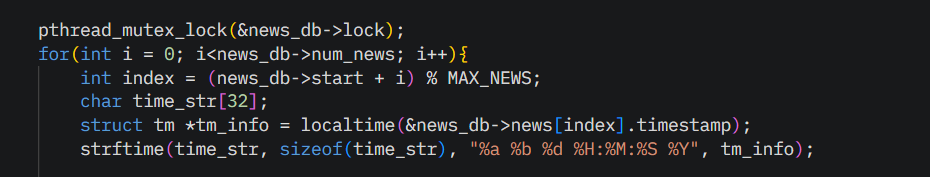
* + - **Reading News:**
      * Readers wait on used\_slots to ensure at least one news item is available.

In show\_news\_by\_category function:



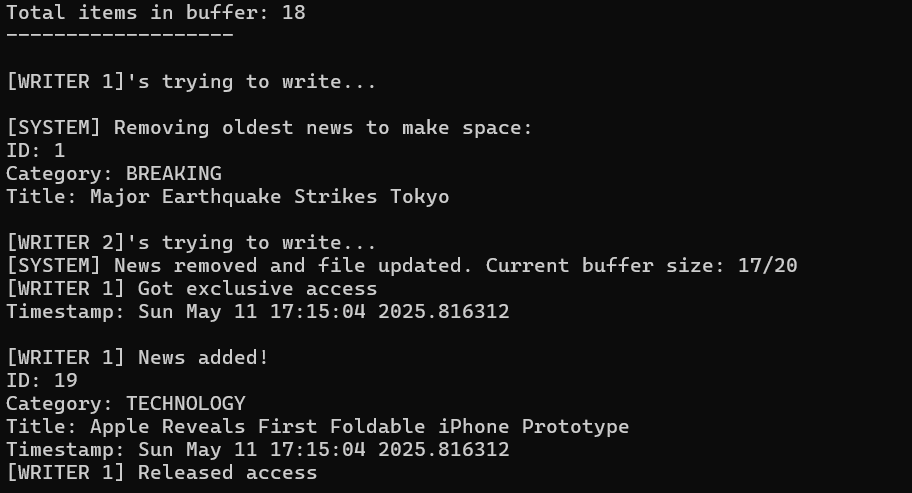
* + - * Access existing news items without modifying the buffer, allowing multiple readers to read concurrently.

In show\_all\_news function:



The buffer’s persistence is maintained by saving news items to a file and loading them during initialization, ensuring no data is lost between program executions.

Output:

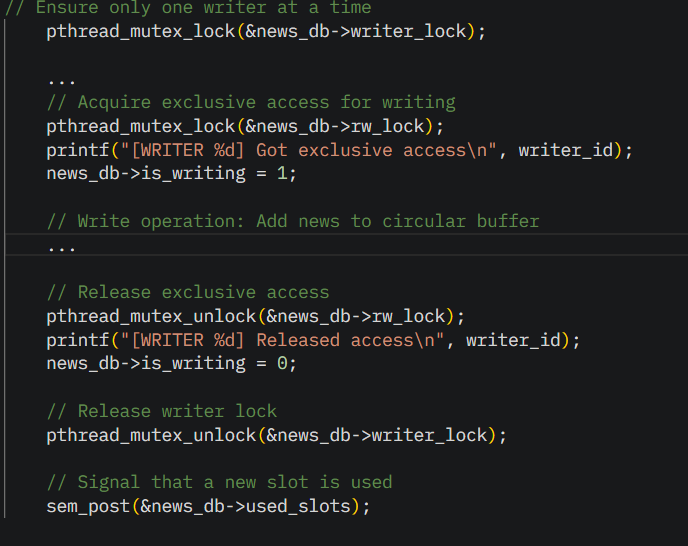


**Reader-Writer Problem Handling:**The reader-writer problem allows multiple readers to access a shared resource simultaneously without modifying it, while writers require exclusive access to prevent data corruption. In this system, multiple readers can read news items concurrently, but writers need exclusive access to add or edit news.

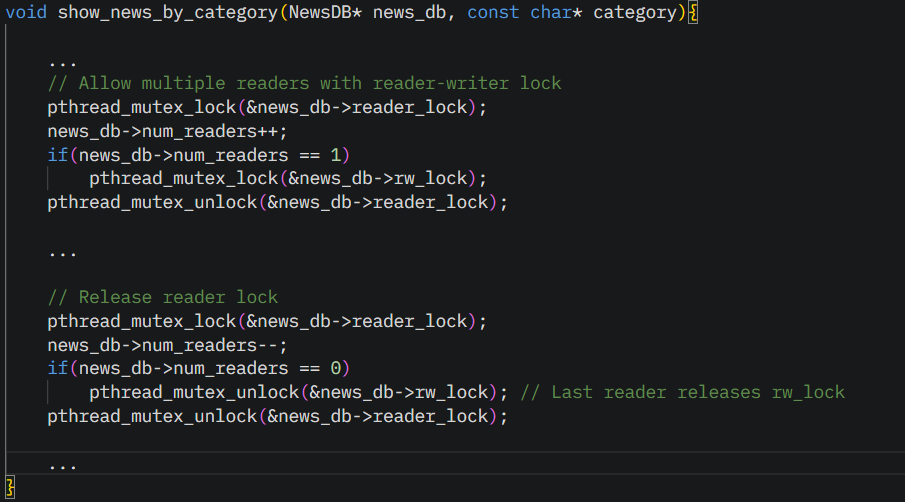
Code Snippets:

The following code snippets demonstrate how the system implements the reader-writer mechanism, formatted with white text for visibility:

* + - **Writer Acquiring Access**: This code ensures writers have exclusive access to the database.  
      In add\_news function:



* + - **Reader Acquiring Access:** This code allows multiple readers to access the database simultaneously.In show\_news\_by\_category function:



**Role of Reader-Writer in the problem:**

* + - **Readers:** Multiple readers can access the news database simultaneously. The reader\_lock is used to increment num\_readers. The first reader acquires rw\_lock, preventing writers from accessing the database. Subsequent readers increment num\_readers without acquiring rw\_lock. The last reader decrements num\_readers to 0 and releases rw\_lock, allowing writers to proceed.
    - **Writers:** Writers acquire writer\_lock to ensure only one writer is active, then acquire rw\_lock for exclusive access, setting is\_writing to 1. After complet- ing the write operation, they release rw\_lock, set is\_writing to 0, and release writer\_lock.

This implementation follows a reader-preference approach, allowing multiple readers to proceed unless a writer is active, while ensuring writers have exclusive access when needed.

**Edge Cases:**The system is designed to handle various edge cases to ensure robust operation.The following table summarizes these cases and their handling:

|  |  |
| --- | --- |
| **Edge Case** | **Handling** |
| Buffer Full | When **num\_news >= WARN\_THRESHOLD (18)**, the remove\_oldest\_news function is called to increment start, decrease num\_news, and post to free\_slots, making space for new news |
| Multiple Readers | Multiple readers can read simultaneously. The reader\_lock manages num\_readers, and the first reader acquires rw\_lock, allowing subsequent readers to proceed without additional locking. |
| Writer While Readers Active | Writers wait for all readers to release rw\_lock (**when num\_readers reaches 0**) before acquiring it exclusively, ensuring no data corruption. |
| No News Available | Readers wait on used\_slots until a writer posts to it, indicating that news is available. |
| Concurrent Writers | The writer\_lock ensures only one writer can acquire rw\_lock and modify the database at a time, preventing race conditions. |

**Output Tracing:**

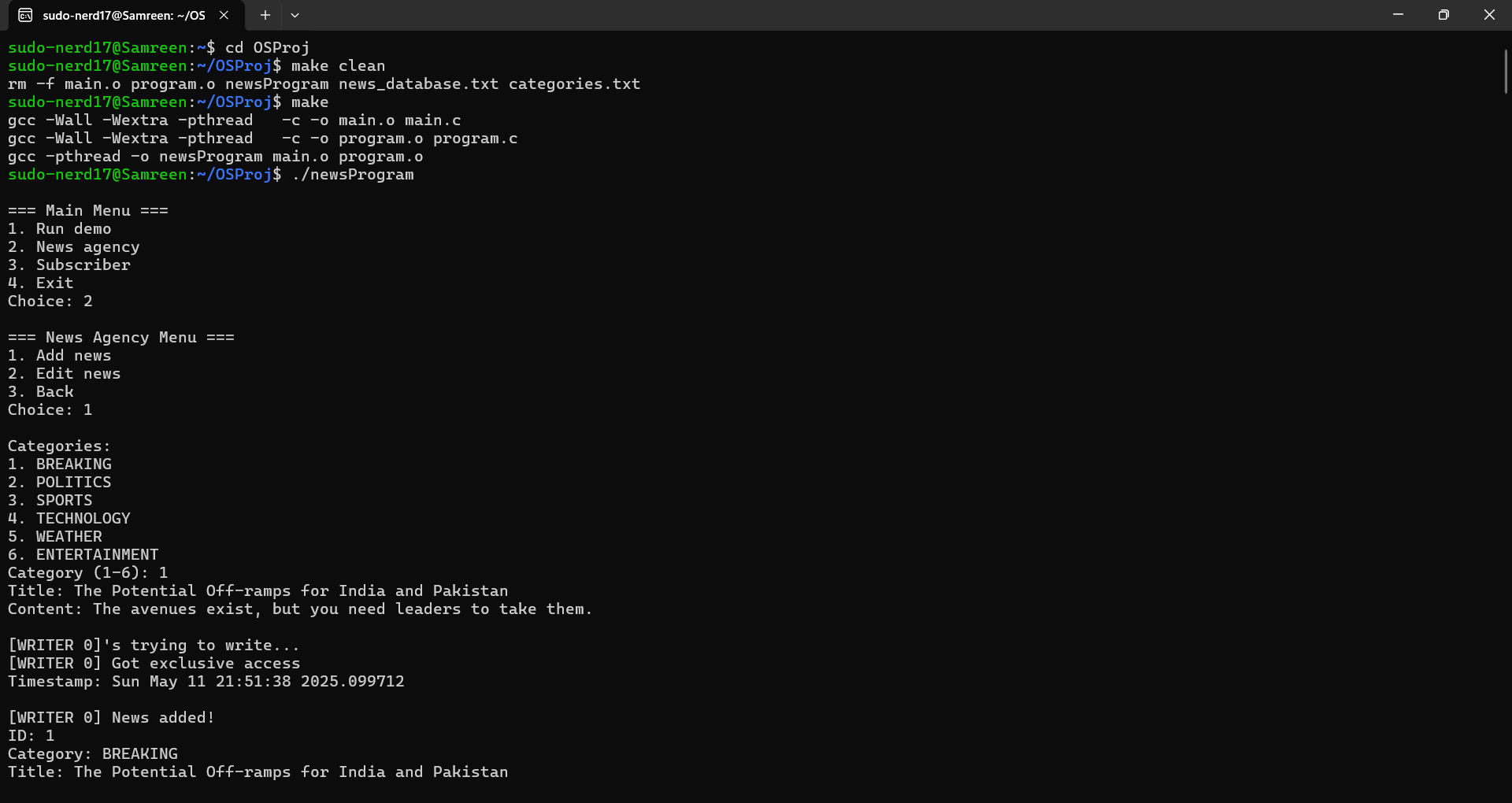
To illustrate the system’s behavior, the following hypothetical sequence traces the execution of various operations, showing how synchronization primitives and the circular buffer operate:

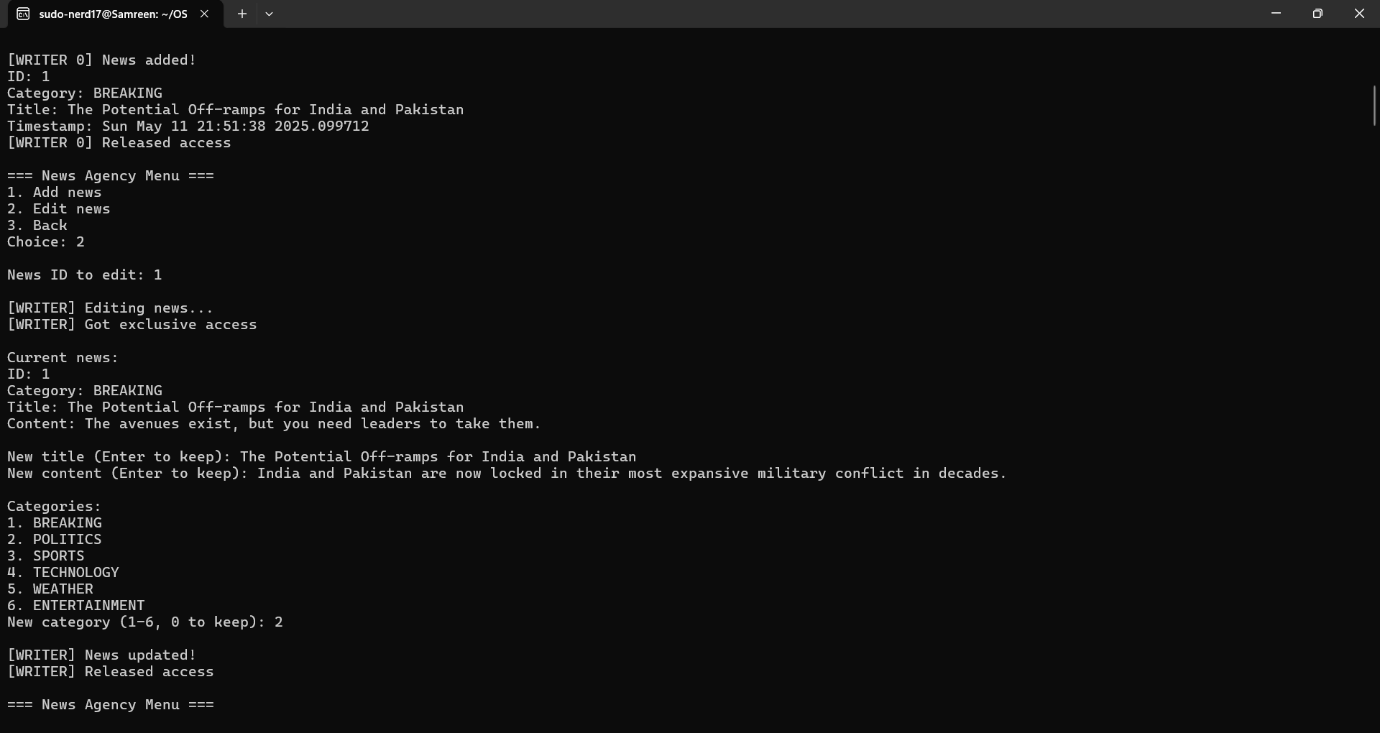
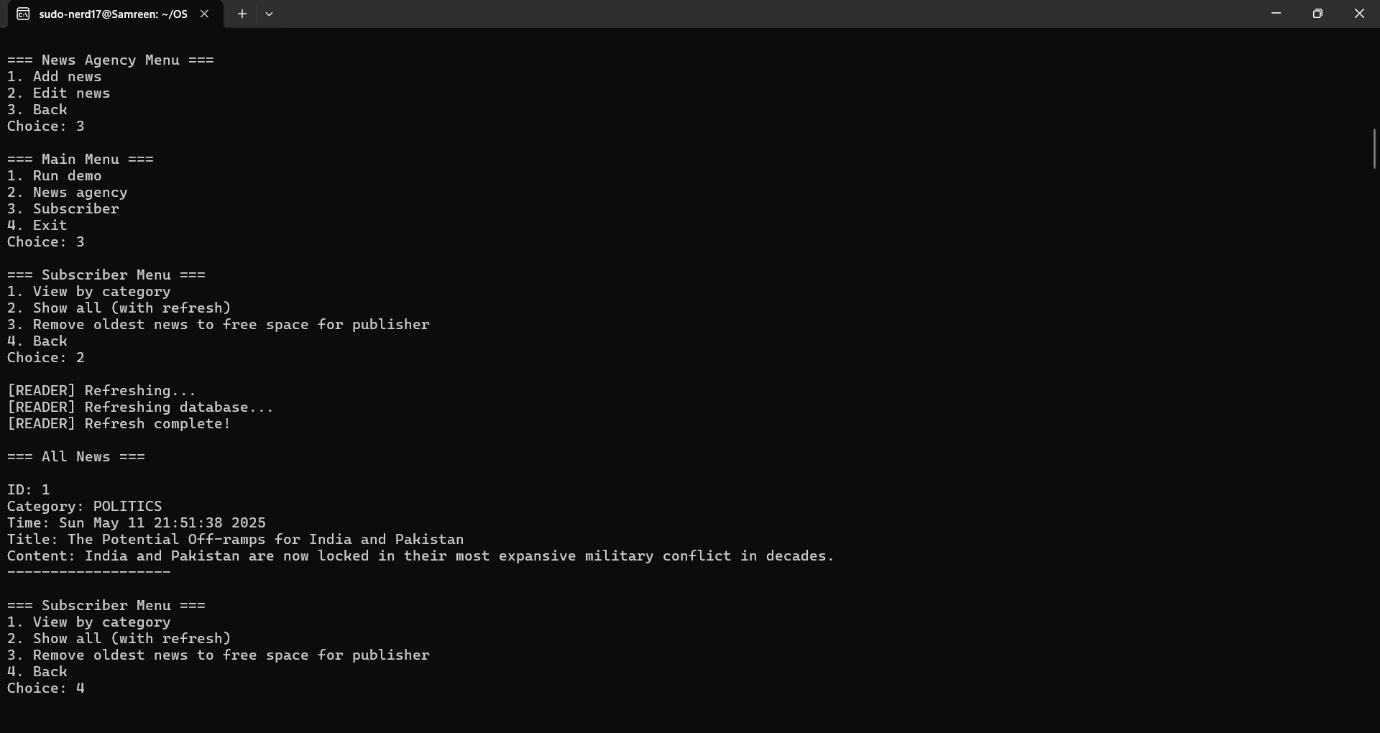
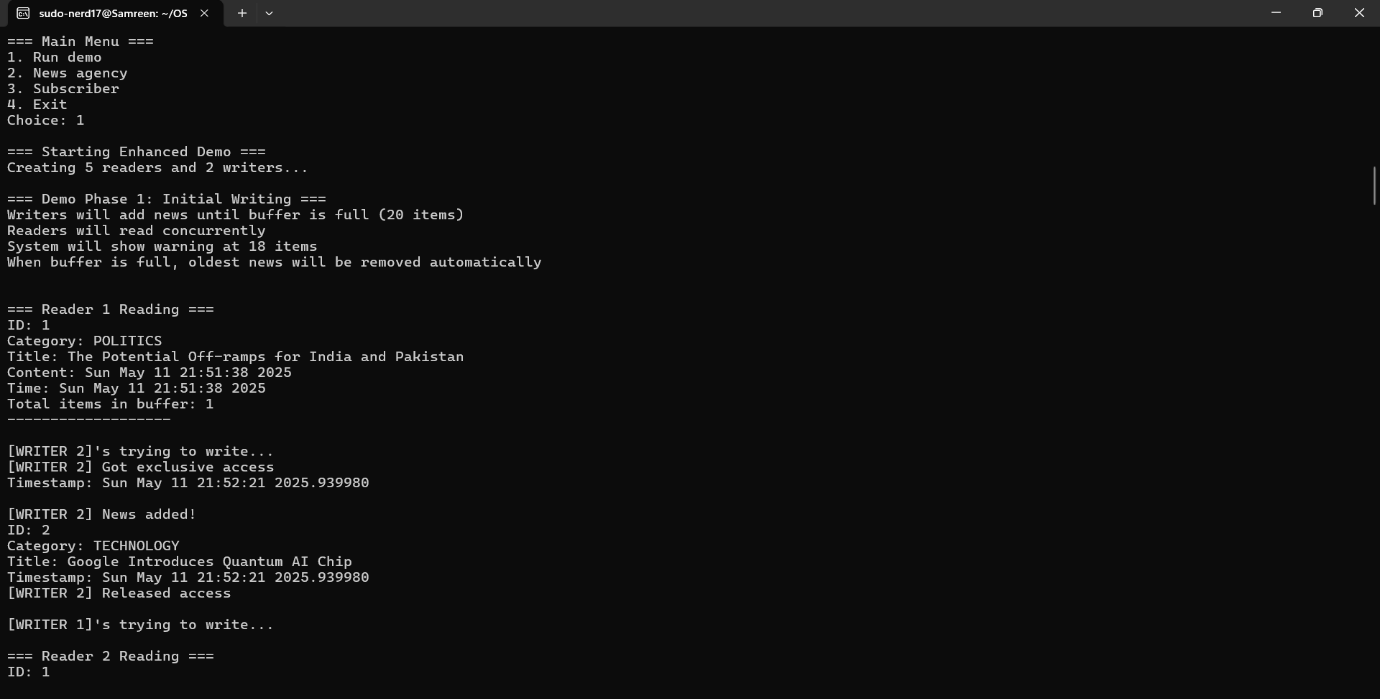
1. Initialization:
   * The news database is initialized with an empty buffer: num\_news = 0, start= 0, end = 0.
   * Semaphores are set: free\_slots = 20, used\_slots = 0.
   * All mutexes (lock, rw\_lock, reader\_lock, writer\_lock) are initialized and unlocked.
   * The news file (news\_database.txt) is opened in append mode, and categories are written to categories.txt.
2. Writer Adds News:
   * Writer 1 attempts to add a news item (e.g., category: BREAKING, title: "Major Earthquake").
   * Waits on free\_slots, decrementing it from 20 to 19.
   * Acquires writer\_lock to ensure no other writers are active.
   * Acquires rw\_lock for exclusive access, setting is\_writing = 1.
   * Acquires lock, adds the news item at news[0], sets num\_news = 1, and in crements end to 1.
   * Saves the news to news\_database.txt and releases lock.
   * Releases rw\_lock, sets is\_writing = 0, and releases writer\_lock.
   * Posts to used\_slots, incrementing it from 0 to 1.

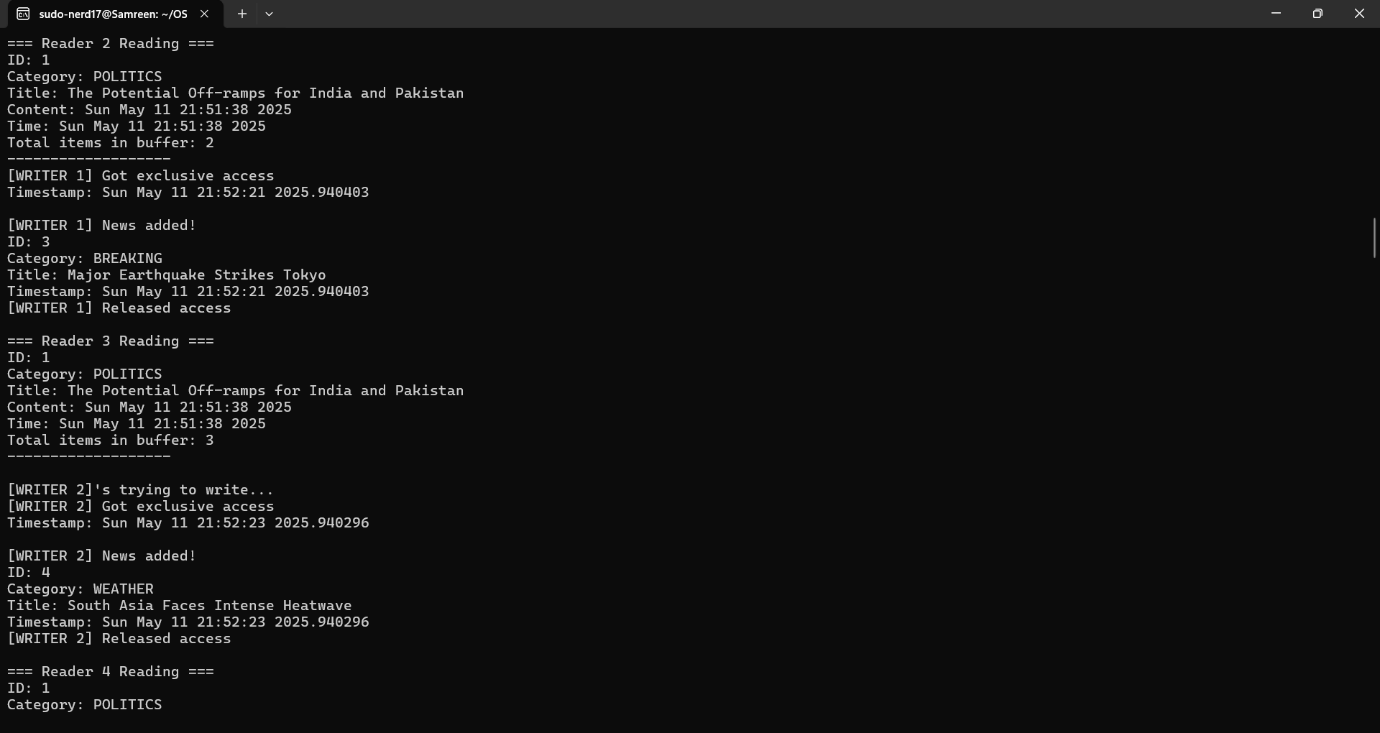
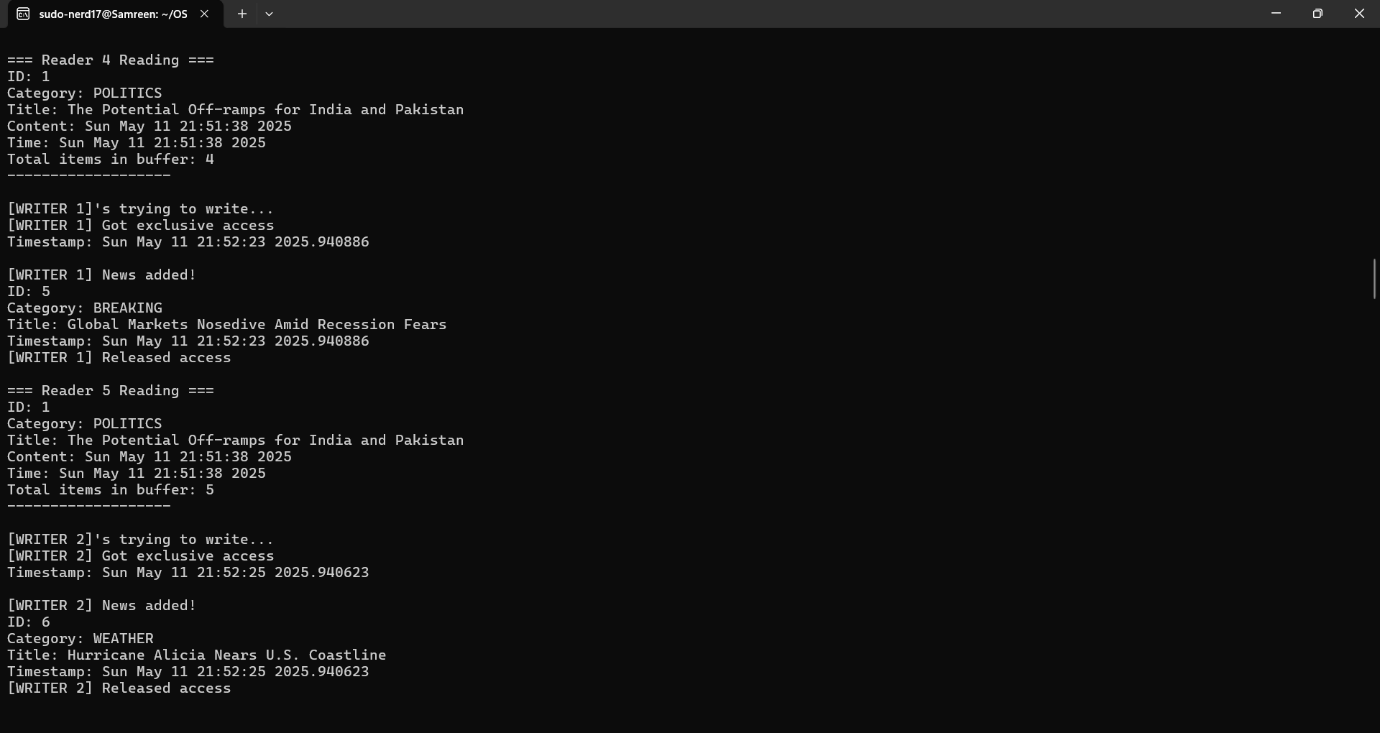
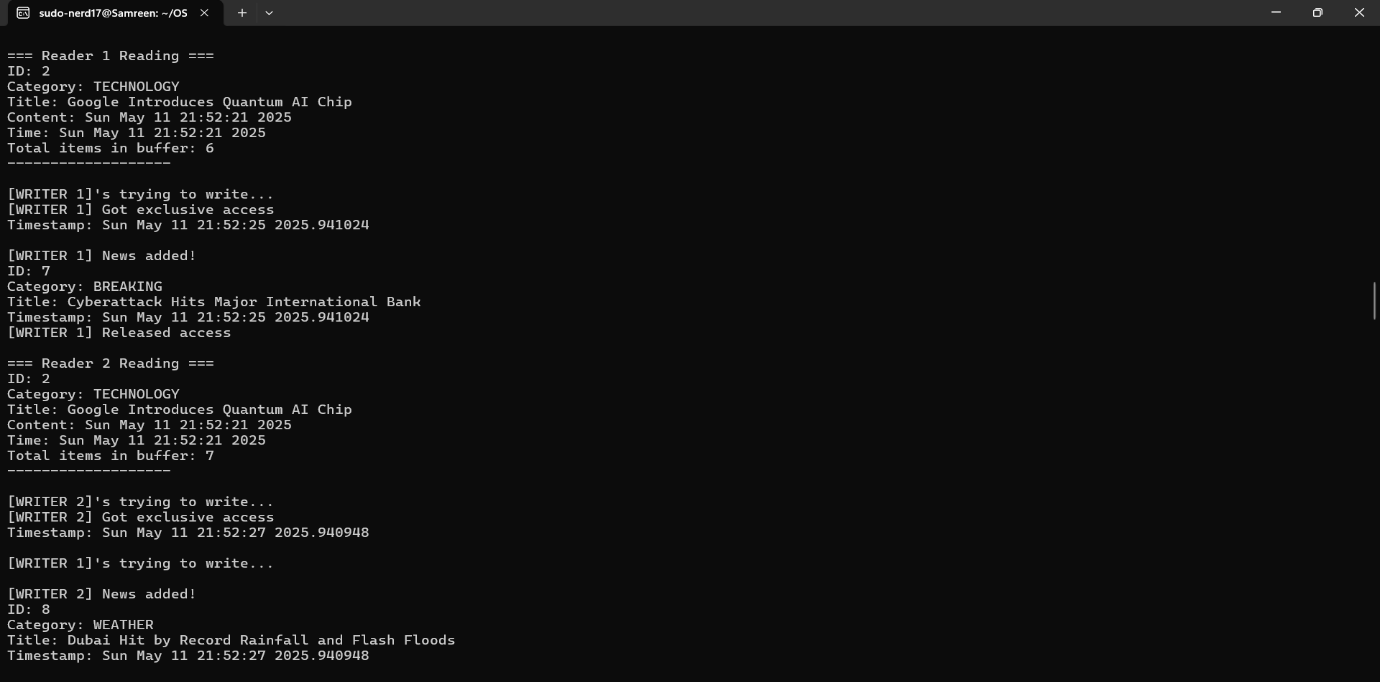
### Reader Reads News:

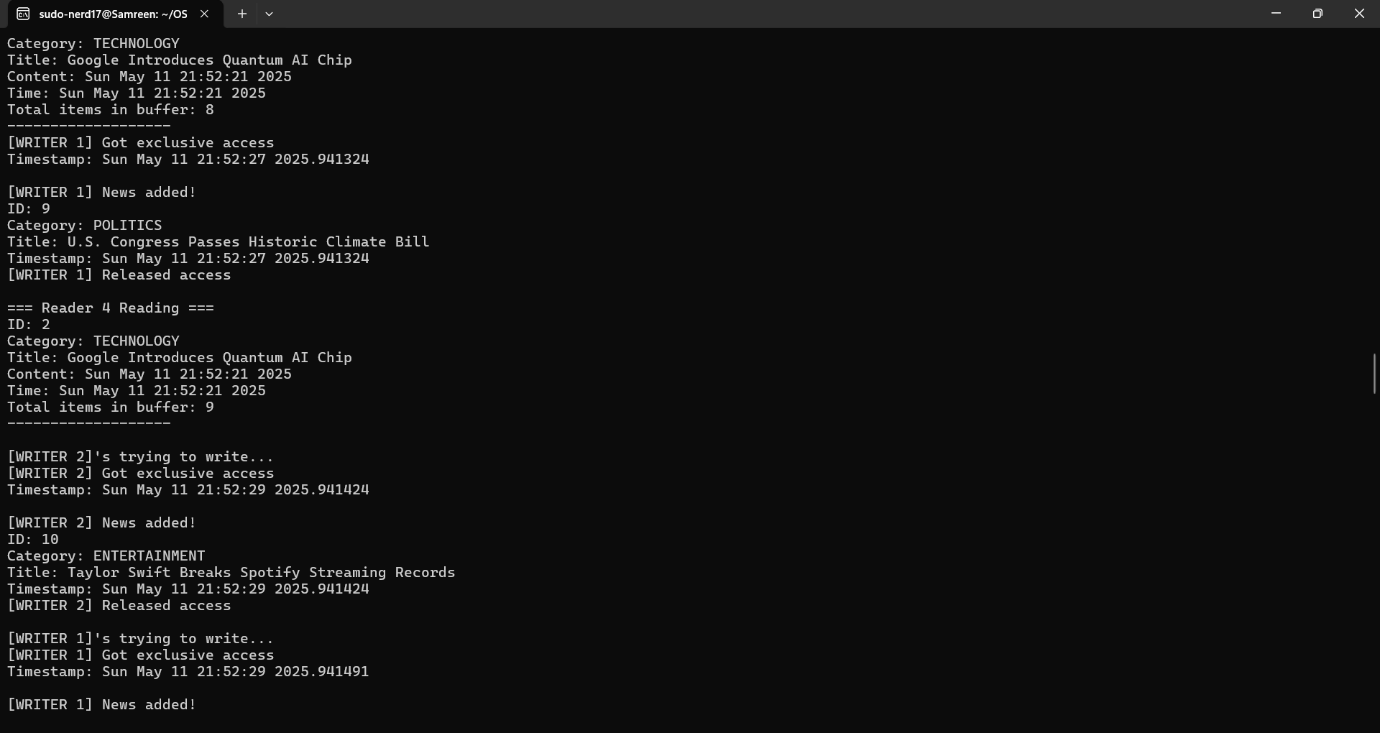
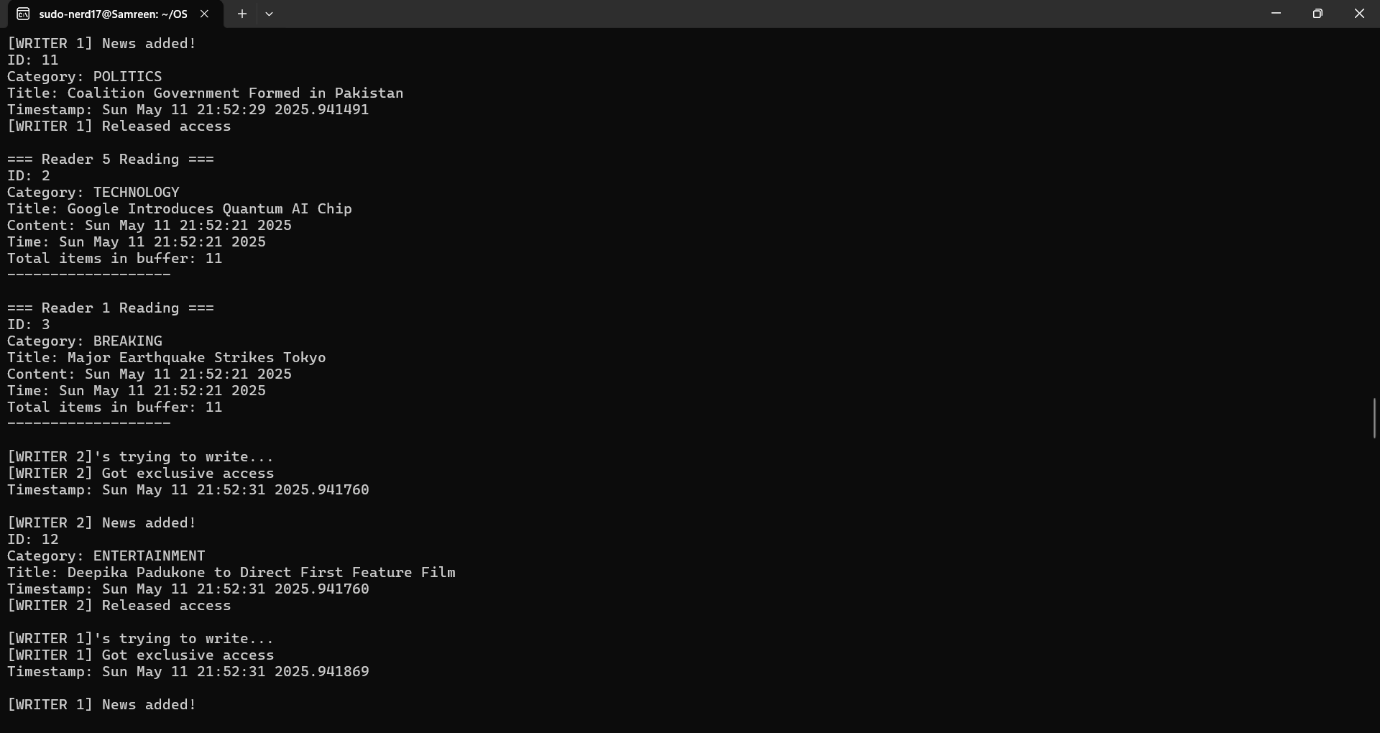
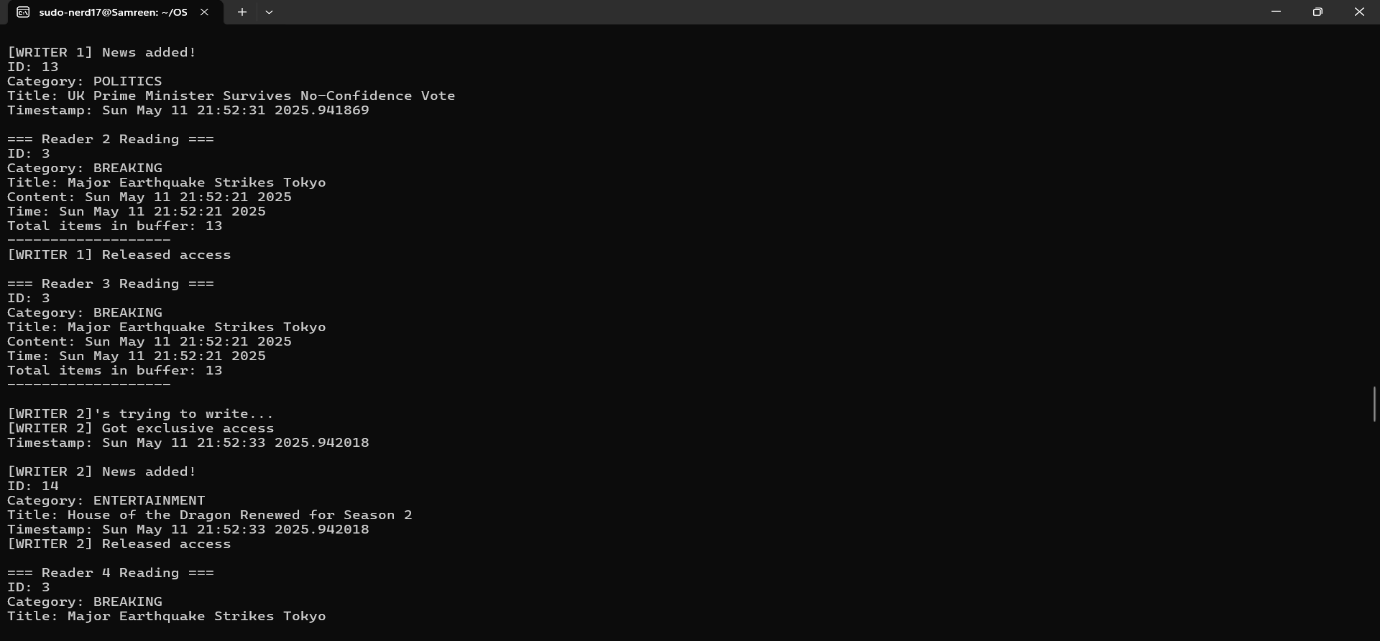
* + Reader 1 attempts to read news by category (e.g., BREAKING).
  + Waits on used\_slots, decrementing it from 1 to 0.
  + Acquires reader\_lock, increments num\_readers to 1, and acquires rw\_lock (first reader).

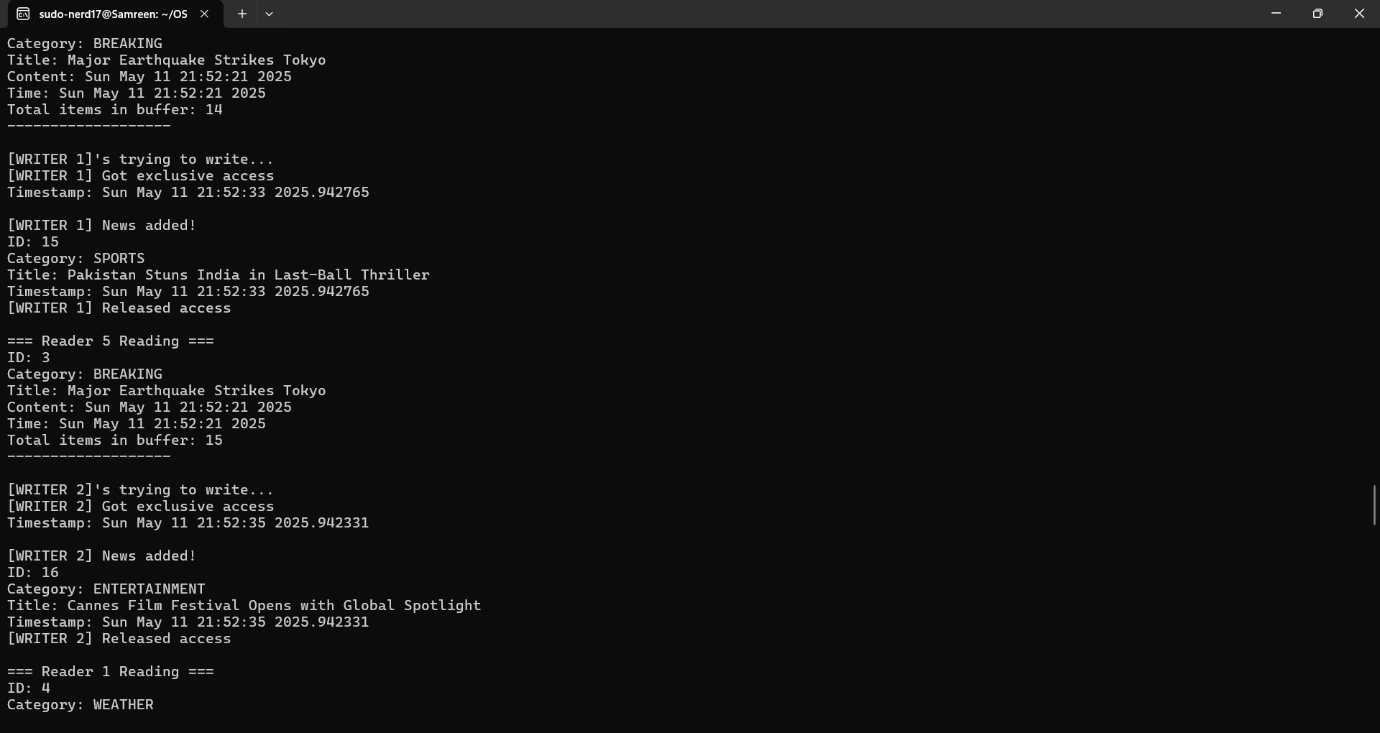
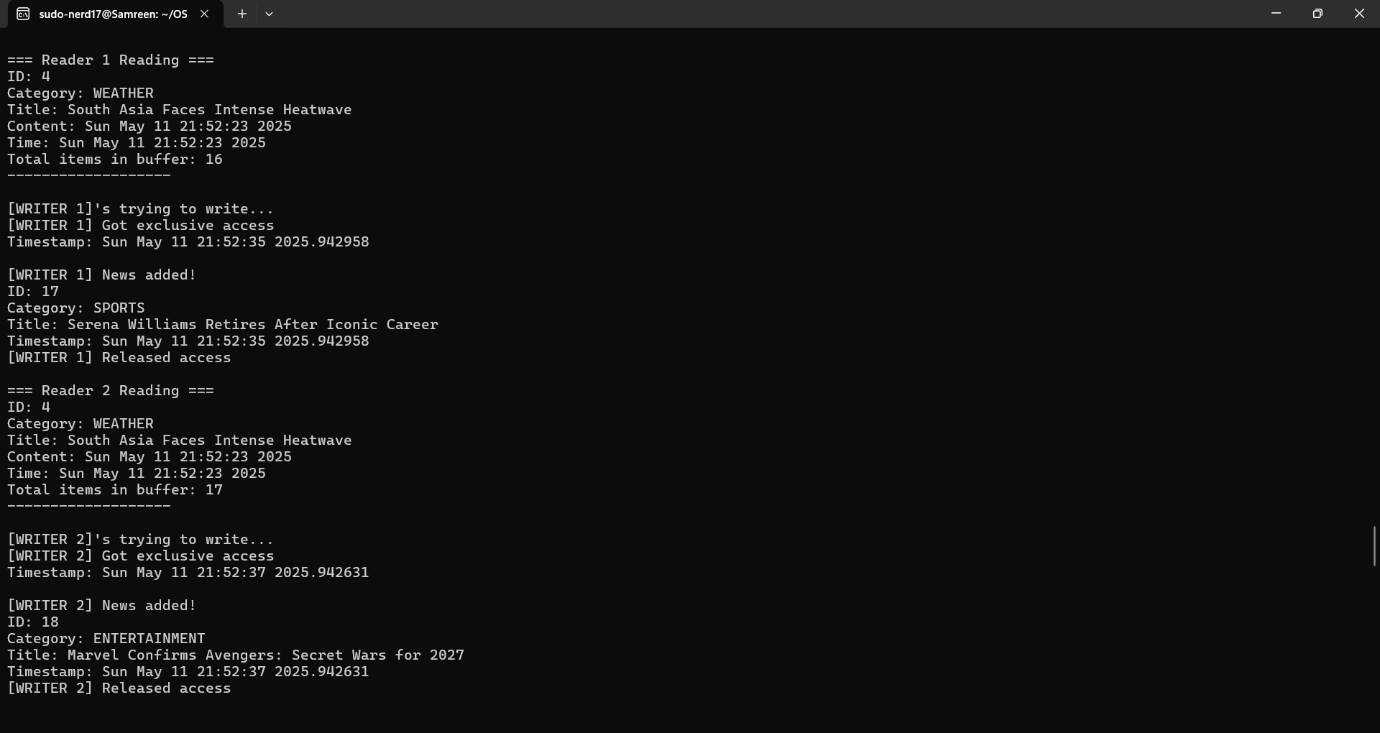
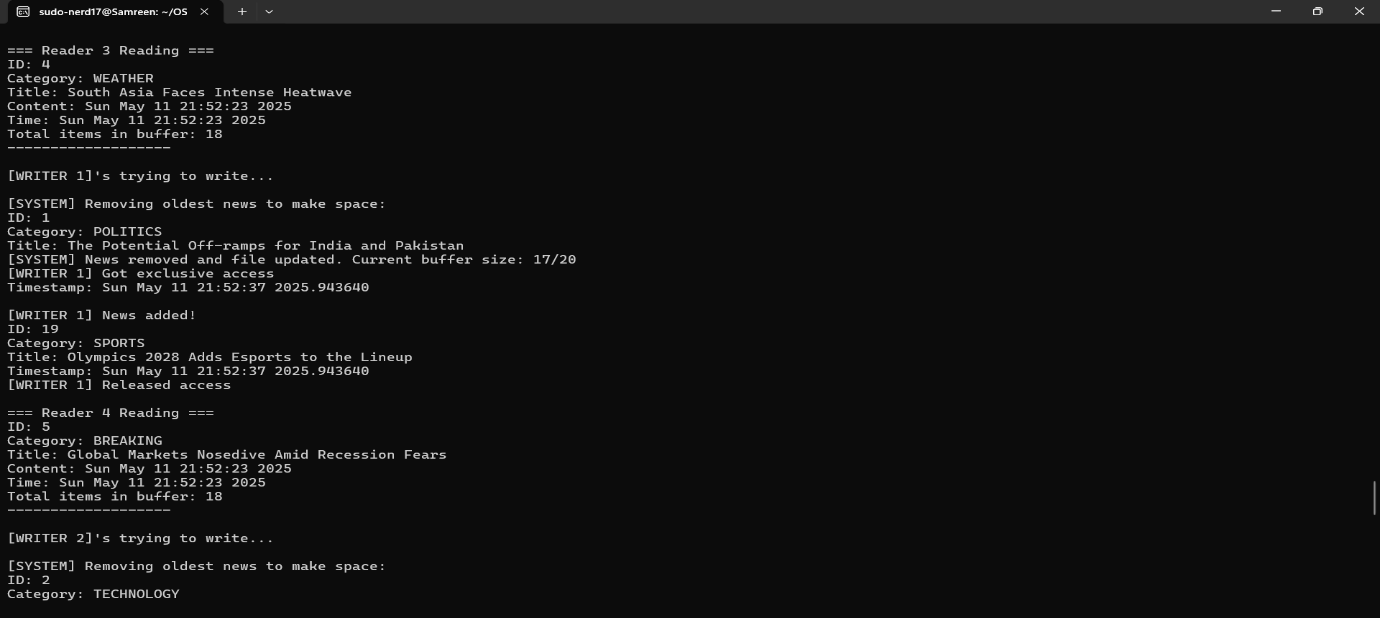
**Complete Run:**

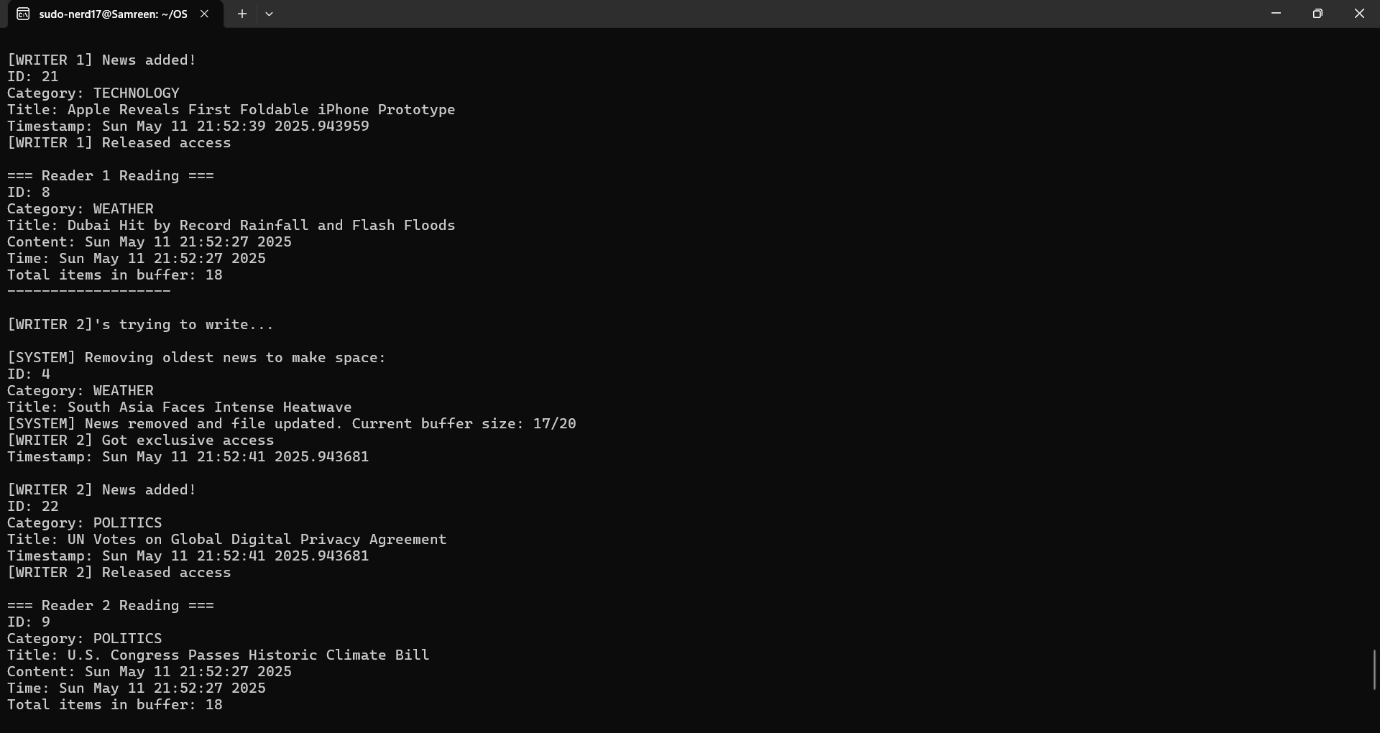
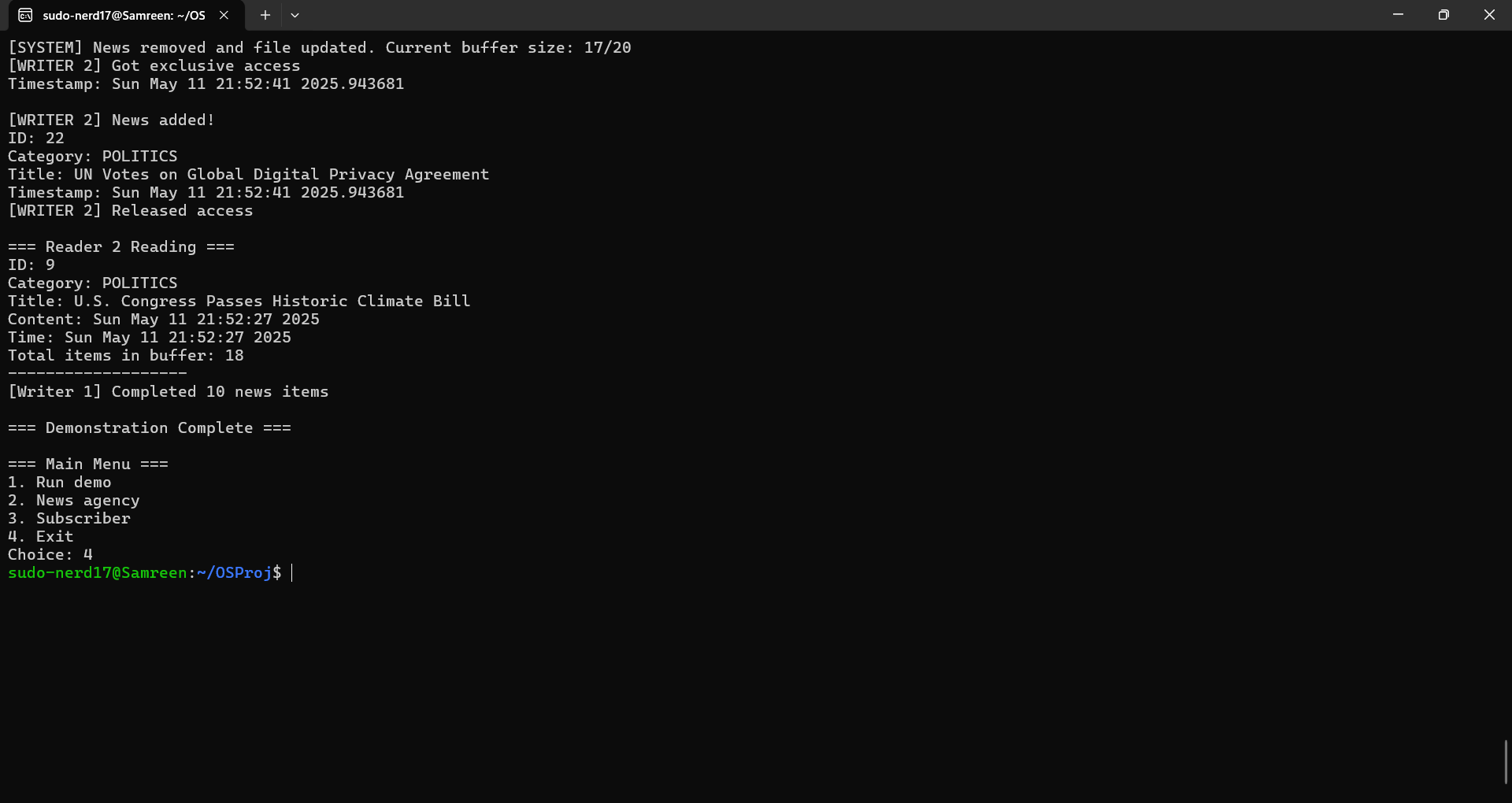
****

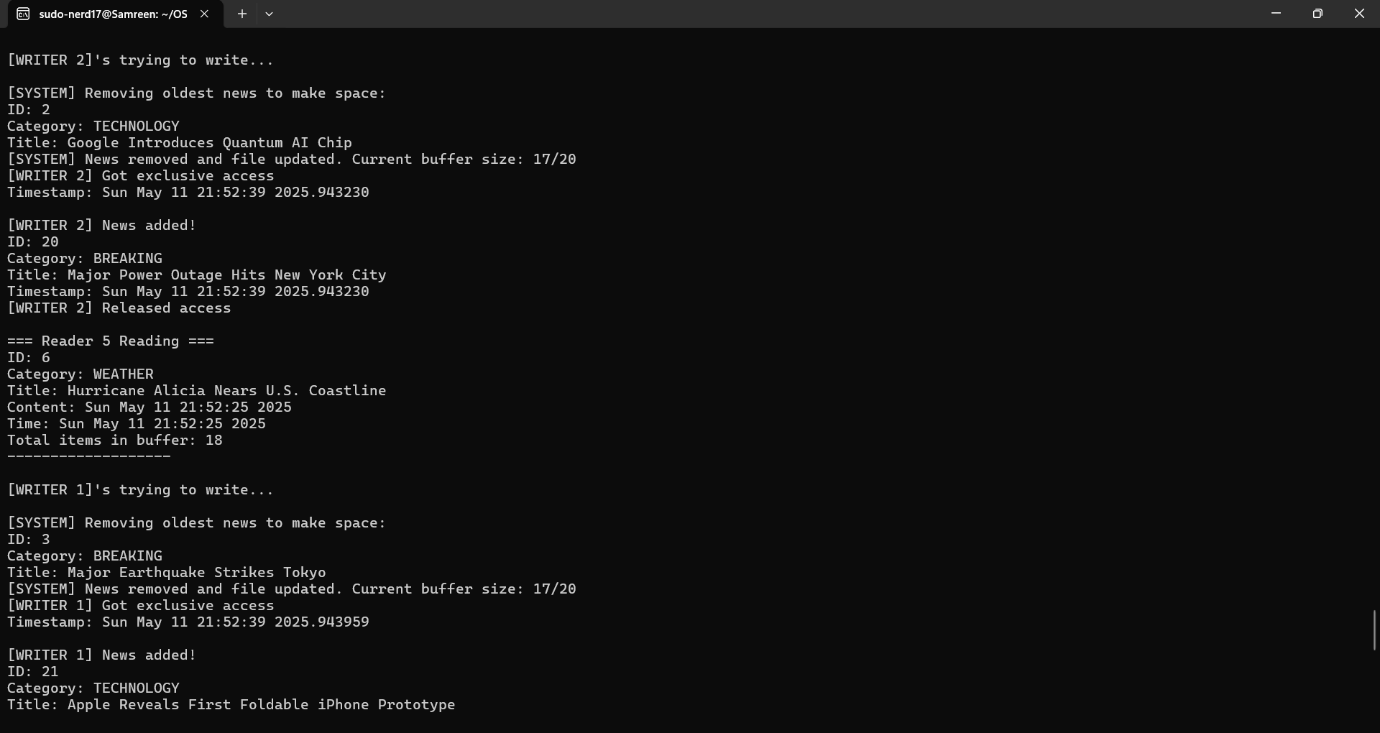
****

****

****

****

****

****