**Samonorom Chum**

**Wednesday May 7, 2025**

**EECE 4811/5811 Operating System Spring 2025**

**Professor Tseng**

**HW8**

1. **Safe Logger: Exploring Synchronization and fsync in Go (6 pt)**

* **Overview**: In this assignment, you will build a simplified logging service in Go that writes log entries to a file. The logger will support concurrent access from multiple goroutines, simulating client requests. You will implement three different versions of the logger to explore synchronization and the importance of flushing data to disk with fsync.
* **Objectives/Concepts**: By the end of this assignment, you should be able to:
  + Understand the use and impact of fsync for ensuring data durability.
  + Explore synchronization in Go using mutexes vs. channels.
  + Observe the effect of race conditions and data corruption when synchronization is missing or incorrect.
  + Build basic I/O programs in Go that reflect real-world systems behavior.
* **Log entry format**
  + Each log entry should include:

1. Log level (INFO, WARN, ERROR)
2. Context (e.g., request ID or user ID)
3. Message

That is the format is [timestamp] [LEVEL] [context] message

* Example:

[2025-05-01 14:02:11] [INFO] [req-723] Starting processing

* Note: You can generate random or fixed strings for the log level, context, and message.

**Specification of your THREE loggers:**

■ **Naive Logger**:

1. No synchronization.
2. Observe what happens when multiple goroutines write to the file concurrently.
3. Use fsync after every write.

■ **Mutex Logger**:

1. Use a global sync.Mutex to protect file writes.
2. Use fsync after every write.
3. Add batching: only call fsync after every 10 log entries.

**■ Channel Logger:**

1. Use a chan string or chan LogEntry to send log messages to a dedicated logger goroutine.

(LogEntry is your self-defined data structure.)

1. The logger goroutine handles writing to the file.
2. Use fsync after every write.
3. Add batching: only call fsync after every 10 log entries.

■ Note: Batching means the logger can buffer 10 writes in memory and then flush them to disk with one fsync. You can still write to the file immediately

— just delay the call to fsync.

* **Benchmarking**:

■ Write a test program to simulate concurrent logging:

1. Create 5 to 10 goroutines, each writing 50 log entries to the logger.
2. Time how long it takes to complete all writes for each logger.
3. Collect and compare the results.

* **Evaluation**:

■ What problems did you observe in the naive logger?

1. Did you see corrupted or interleaved output?
2. Did any data go missing?
3. How can you detect a data race (e.g., using go run -race)?

■ How did the mutex and channel loggers solve these problems?

■ Compare performance across all three implementations.

1. How long did each take to complete?
2. How did batching affect performance?

■ When is fsync necessary? Why is it expensive?

**References:**

1. ChatGPT, “Assistance with HW8 Exploring Synchronization and fsync in Go,” OpenAI, personal communication, May. 7, 2025.