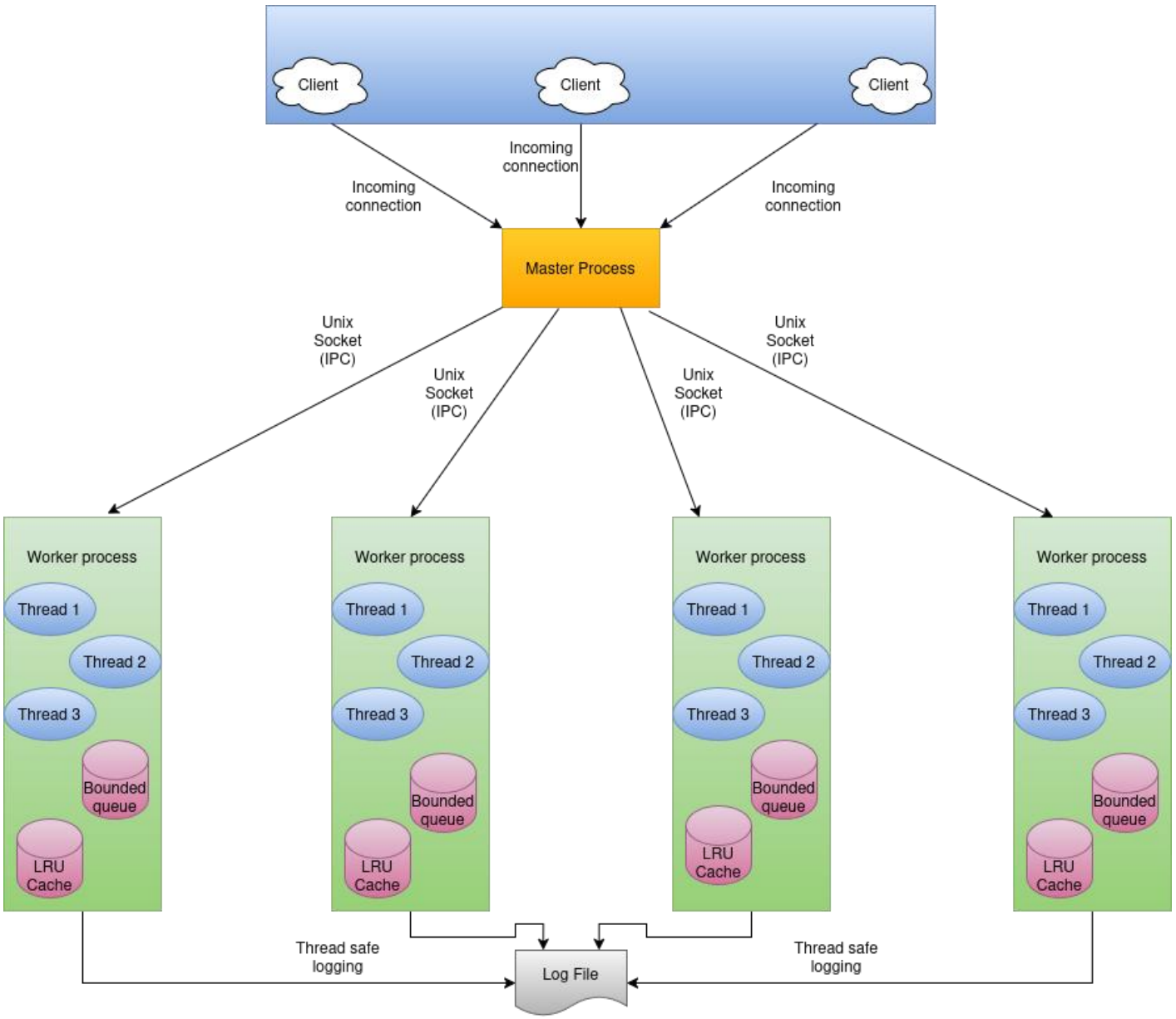
Multi-Threaded Web Server with IPC and Semaphores

Design

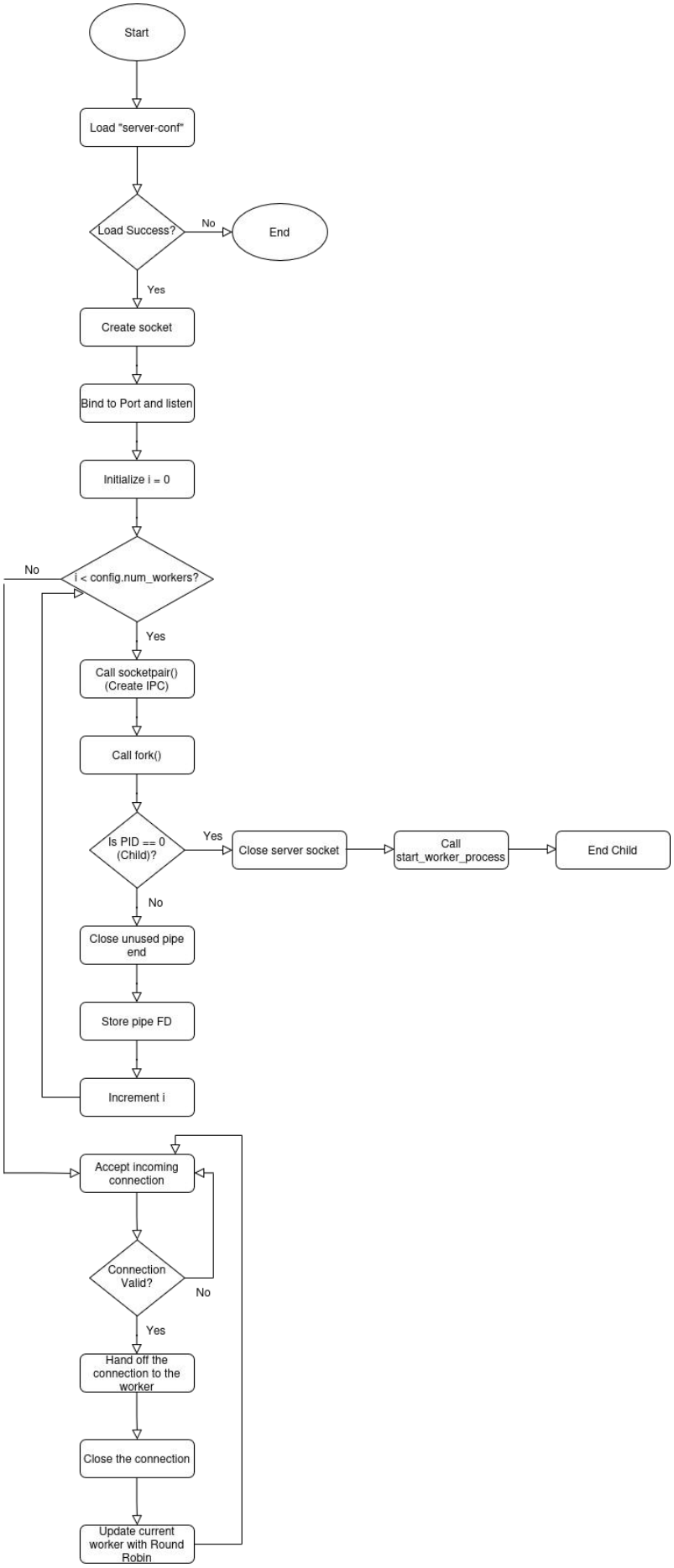
Authors:
Alan Marques (125046)
Miguel Sousa (125624)

Sistemas Operativos

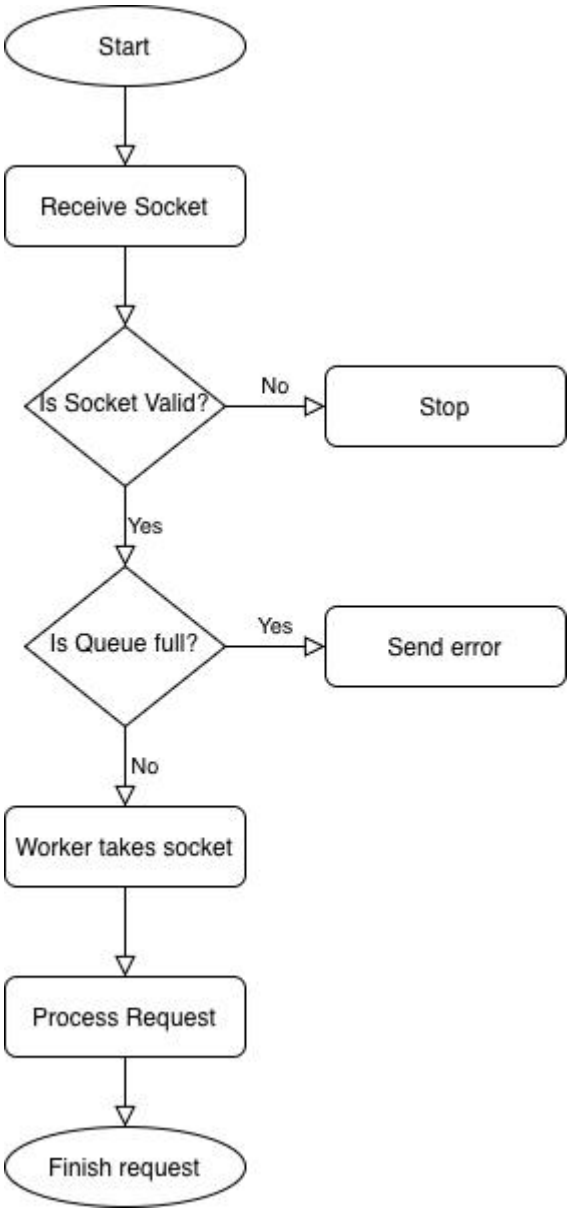
Architecture Diagram



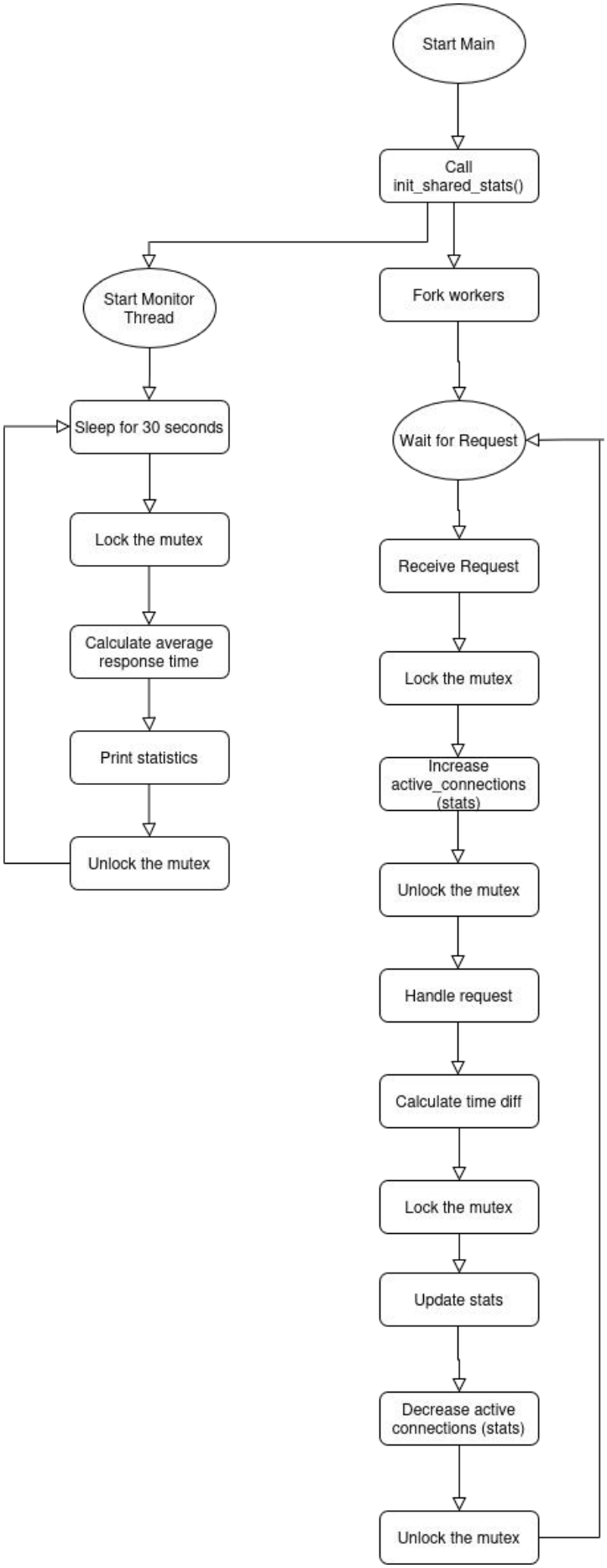
Feature 1 (Master Process) Flowchart



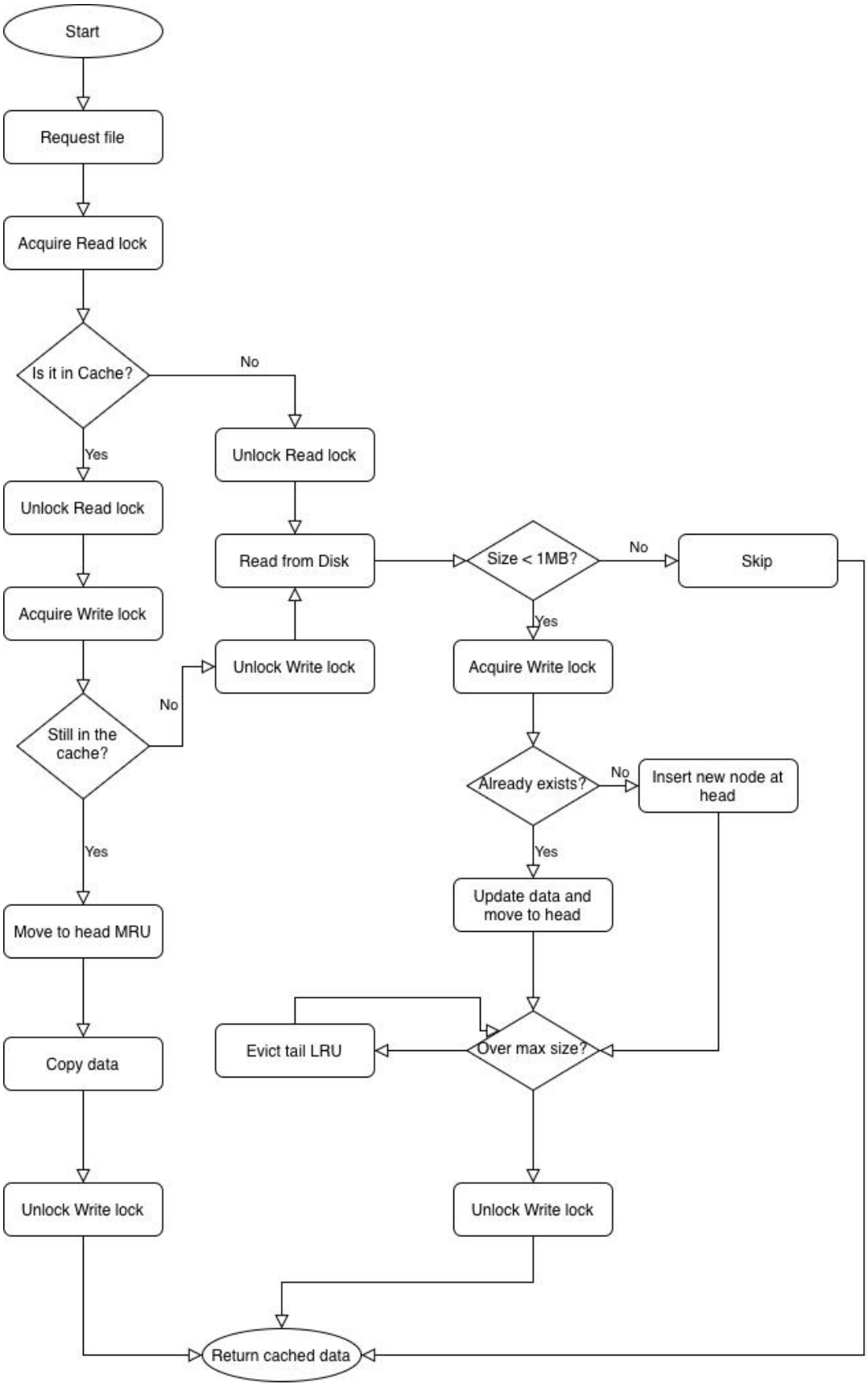
Feature 2 (Thread Pool Management) Flowchart



Feature 3 (Shared Statistics) Flowchart



Feature 4 (Thread-Safe File Cache) Flowchart



Synchronization Analysis

The server uses a Hybrid Model to handle concurrent requests:

- Master Process: Accepts connections.
- Worker Processes: Independent processes that handle the actual work.
- Worker Threads: Threads within each worker that process requests in parallel.

The synchronization mechanisms are Shared Resources (Inter-Process)

These resources are shared across all Worker processes. We use Named Semaphores to ensure safety.

Resource	Protection Mechanism	Description
Statistics	Named Semaphore	Ensures only one process updates the global counters (requests, bytes) at a time.
Log File	Named Semaphore	Ensures lines written to server.log do not get mixed up between processes.
Request Queue	Named Semaphore	Coordinates the handoff of connections from Master to Workers.

Local Resources (Intra-Process)

These resources are private to a single Worker process but shared among its threads.

Resource	Protection Mechanism	Description
Thread Pool	Mutex + Condition Variable	Threads sleep until a new request arrives, saving CPU.
LRU Cache	Read-Write Lock	Allows multiple threads to read cached files simultaneously, but locks exclusively when adding new files.

Key Analysis

- Safety: The use of Named Semaphores (sem_open) guarantees that all processes refer to the exact same lock in the operating system kernel. This prevents race conditions where two processes might try to write to the log file at the exact same time.
- Efficiency:
    - Statistics: Updates are fast and atomic.
    - Caching: The Read-Write lock ensures that the cache is not a bottleneck; most requests are reads, so they don't block each other.

Conclusion

The synchronization logic is thread-safe.

- Global data is protected by kernel-level semaphores.
- Local data is protected by fast thread mutexes.
- No deadlocks or race conditions were found in our implementation.