1.Case Study: Efficient File Synchronization in a Multi-Process Linux System

Problem Statement:

A company maintains a large number of log files that are **continuously updated** by multiple processes. To ensure data consistency, a synchronization mechanism is required that allows multiple processes to write to these log files without data corruption while maintaining high performance.

The challenge is to design and analyze an efficient file synchronization mechanism that balances data integrity, concurrency, and performance.

Requirements:

1. **Concurrent Writes:** Multiple processes should be able to **write to the same file** without **corrupting data.**
2. **Data Integrity:** Ensuring that log entries are not interleaved or lost.
3. **Performance:** The solution should minimize I/O bottlenecks and context switching overhead.
4. **Scalability:** The system should scale efficiently as the number of processes increases.

•5) **Considerations:**
Synchronization Mechanisms:

•File locking (fcntl, flock)

•Memory-mapped files (mmap)

•POSIX message queues for logging

•Named pipes (FIFOs)

•**Buffering vs. Direct Writes:** Should logs be buffered before writing or written directly to disk?

•**Handling Failures:** What happens if a process crashes while holding a lock?

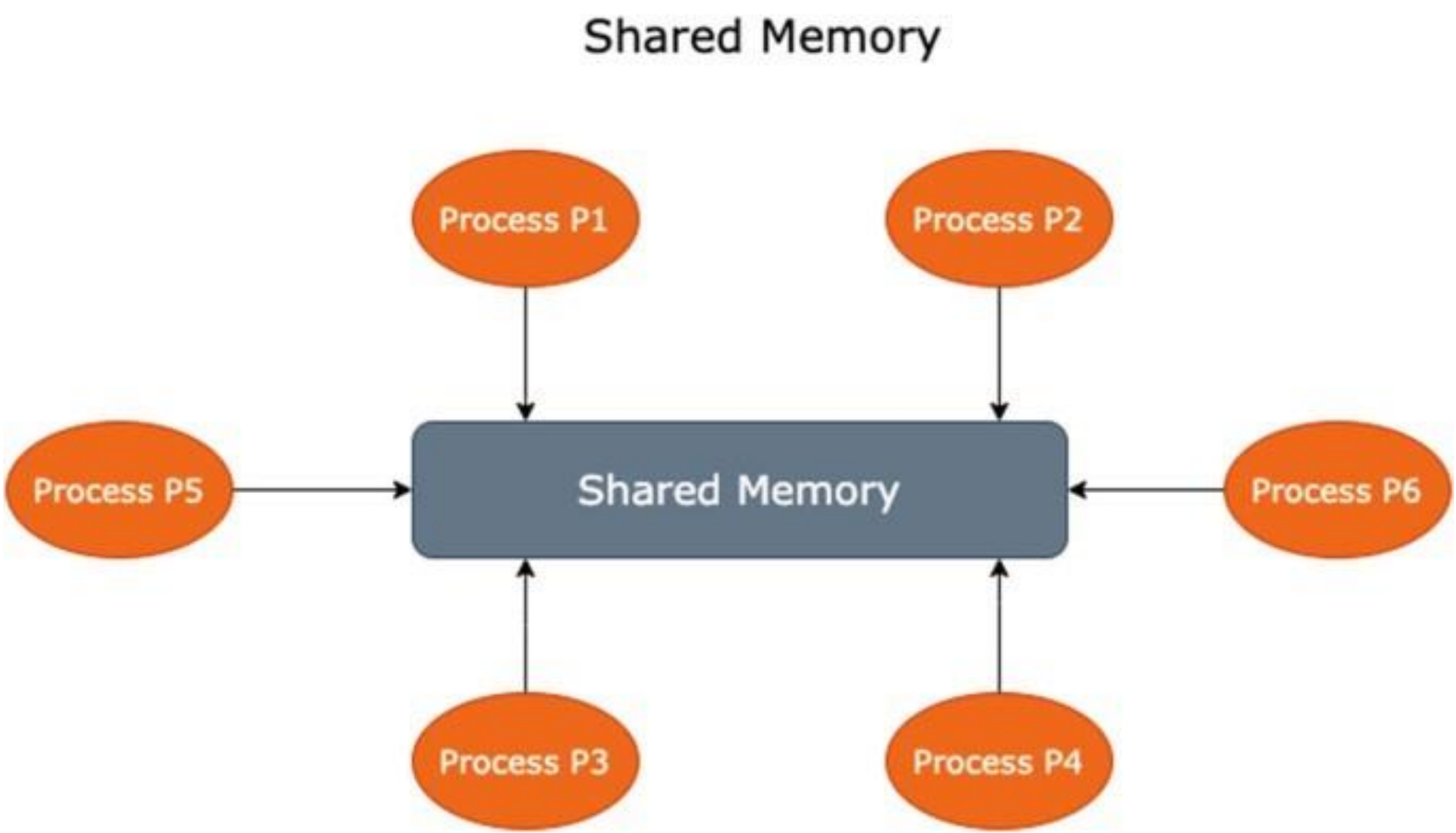
•**Performance Analysis:** Compare different approaches using system tools (strace, perf, iotop).

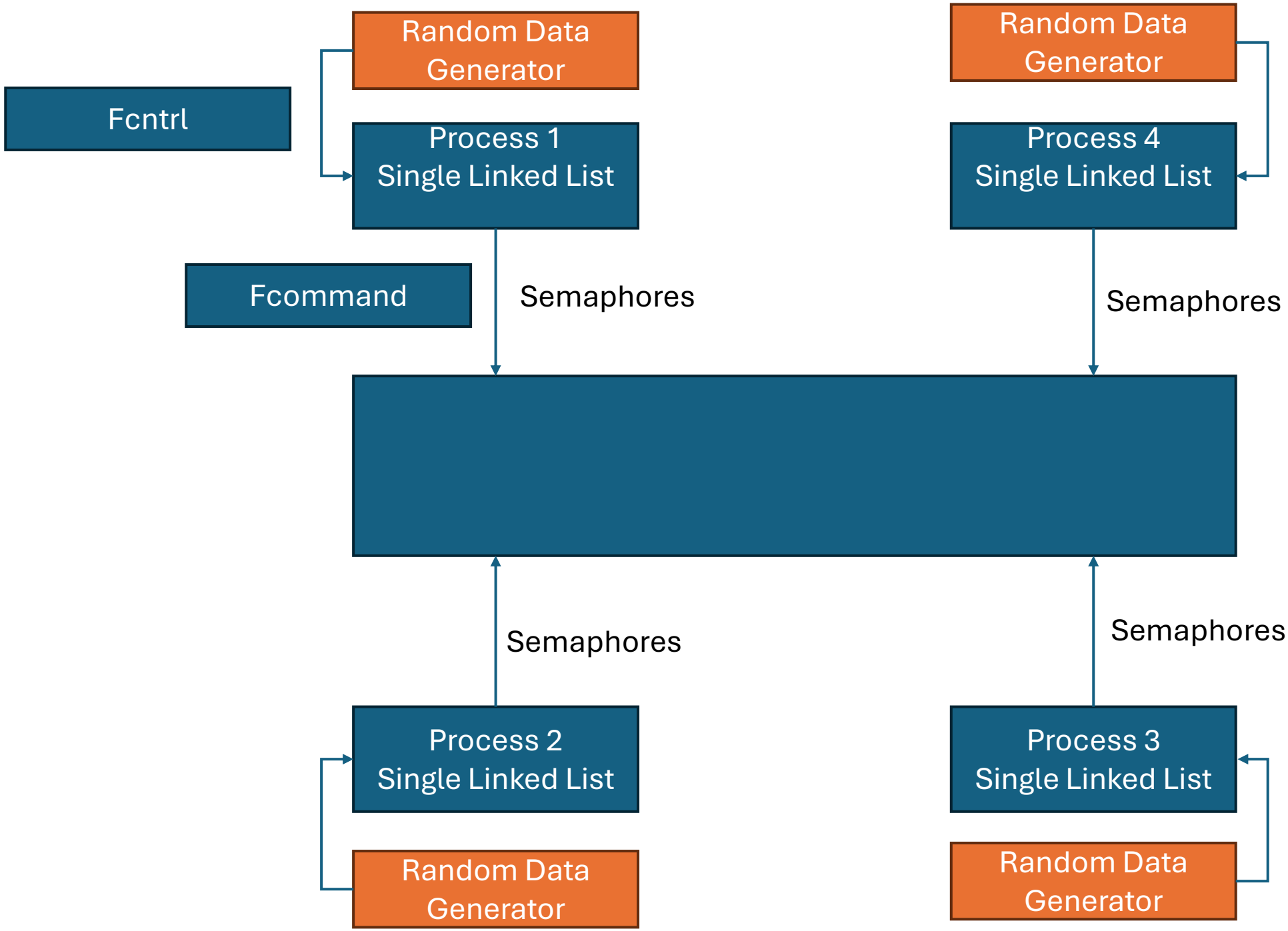
1.Deliverables for Analysis:
Implementation Approach: Describe different synchronization strategies and justify the best choice.

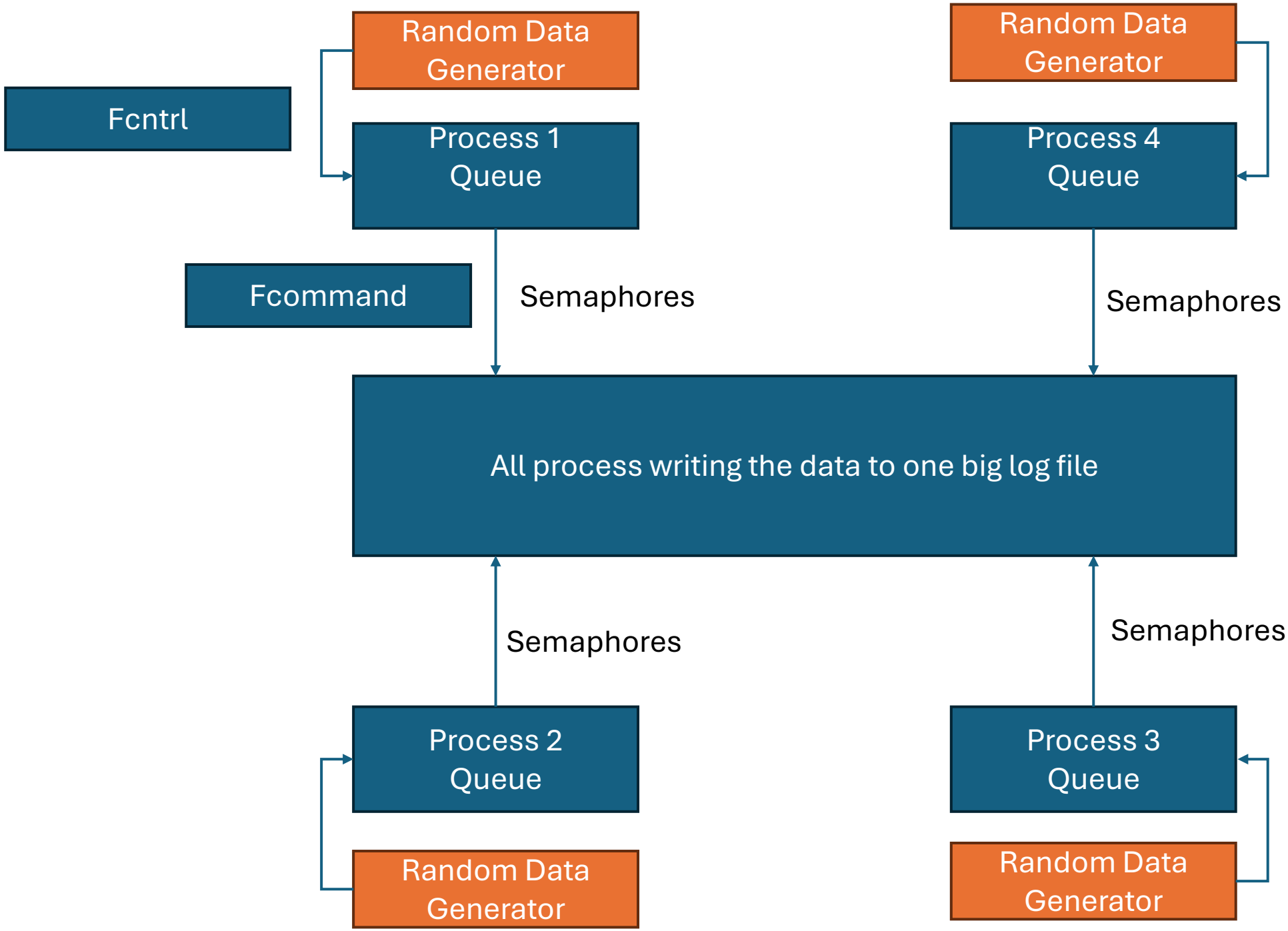
2.Performance Evaluation: Analyze the trade-offs using benchmarks.

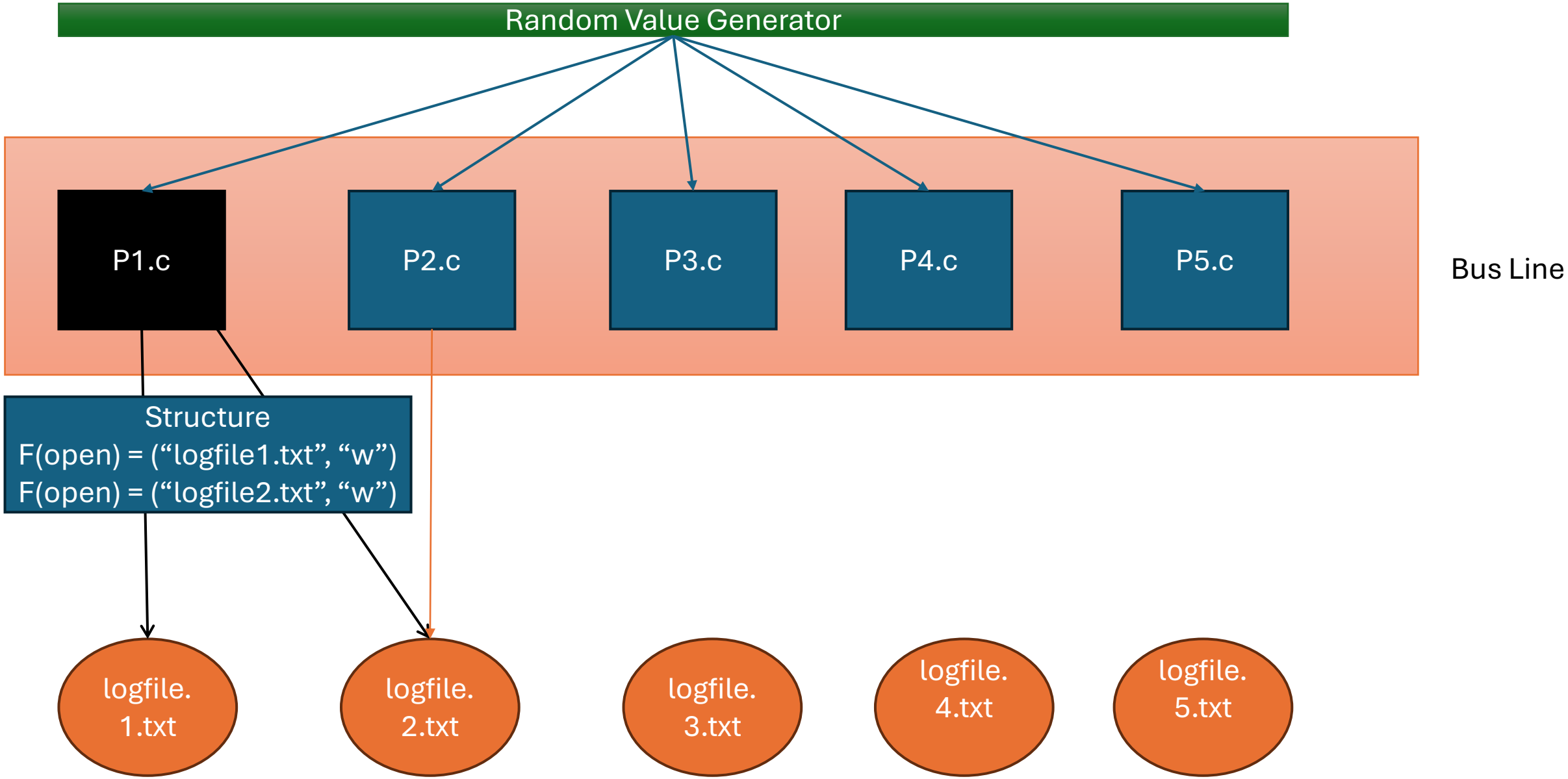
3.Failure Handling: Discuss how to ensure robustness against crashes.

4.Code Sample: Implement a prototype using C with proper error handling and logging mechanisms.


Shared Memory
Process P1
Process P2
Process P5
Shared Memory
Process P6
Process P3
Process P4


Random Data Generator
Random Data Generator
Fcntrl
Process 1 Single Linked List
Process 4 Single Linked List
Fcommand
Semaphores
Semaphores
Semaphores
Semaphores
Process 2 Single Linked List
Process 3 Single Linked List
Random Data Generator
Random Data Generator


Random Data Generator
Random Data Generator
Fcntrl
Process 1 Queue
Process 4 Queue
Fcommand
Semaphores
Semaphores
All process writing the data to one big log file
Semaphores
Semaphores
Process 2 Queue
Process 3 Queue
Random Data Generator
Random Data Generator


Random Value Generator
P1.c
P2.c
P3.c
P4.c
P5.c
Bus Line
Structure
F(open) = (“logfile1.txt”, “w”)
F(open) = (“logfile2.txt”, “w”)
logfile.
1.txt
logfile.
2.txt
logfile.
3.txt
logfile.
4.txt
logfile.
5.txt

What don’t we know?

> how many code files we need required?
> I mean each process have each code file?
>How different multiple process are communication each other?
> How many processes need to take for instance?
>If one more process is added, how it is scalable?

>How to identify which process came first?