Reader-Writer Synchronization Strategies Using Semaphores

Objective

The goal of this project is to design and implement two synchronization strategies to manage concurrent access to a shared memory region by multiple reader and writer threads. The two strategies are:

1. Reader-Priority Synchronization: Enables multiple readers to access the shared resource concurrently while preventing access to writers during reading operations.
2. Writer-Priority Synchronization: Gives priority to writer threads to prevent writer starvation by blocking incoming readers when writers are waiting to write.

1. Reader-Priority Synchronization

Strategy Overview

In this strategy, multiple reader threads are allowed to access the shared memory simultaneously. However, writer threads are granted access only when no readers are active. The implementation ensures:

- Reader concurrency when no writer is active.
- Writers have exclusive access to the shared memory.
- Readers are given precedence, which may lead to potential writer starvation.

Implementation Details

Two synchronization primitives are used:

- mutex: A binary semaphore to protect updates to the reader_count variable, which tracks the number of active readers.
- write_sem: A semaphore that ensures exclusive access to writers.

Reader Entry Section

```
sem_wait(&data.mutex);
reader_count++;
if (reader_count == 1)
```

```
   sem_wait(&data.write_sem); // First reader blocks writers
sem_post(&data.mutex);
```

Reader Exit Section

```
sem_wait(&data.mutex);
reader_count--;
if (reader_count == 0)
   sem_post(&data.write_sem); // Last reader releases writers
sem_post(&data.mutex);
```

Expected Behavior

- Multiple readers can access the shared memory simultaneously.
- Writers access the memory only when no reader is active.
- New readers arriving during writer wait times are granted access, giving readers priority and potentially causing writer starvation.

Sample Output

Reader 1 reads: 0
Reader 2 reads: 0
Writer 1 writes: 1
Reader 1 reads: 1
Reader 2 reads: 1
Writer 2 writes: 2

Explanation:
- Reader 1 and Reader 2 concurrently access the shared data.
- Writer 1 waits for both readers to finish, then writes.
- The cycle repeats, prioritizing readers over writers.

2. Writer-Priority Synchronization

Strategy Overview

This strategy ensures writers are never starved. When a writer is active or waiting, incoming readers are blocked until all pending writers complete. Key properties:

- **Exclusive writer access**: Only one writer modifies data at a time.
- **Reader blocking**: New readers defer if writers are queued or active.
- **No writer starvation**: Writers always gain access after bounded waiting.

Implementation Details

Synchronization Variables

- mutex: Binary semaphore protecting shared counters (readers, writers, waiting_writers).
- write_sem: Ensures only one writer writes at a time.
- **Counters**:
    - waiting_writers: Tracks queued writers.
    - writers: Tracks active writers.
    - readers: Tracks active readers.

Reader Workflow

1. **Check for pending/active writers. If present, retry.**
2. **Increment readers. If first reader, block writers via write_sem.**
3. **Read data.**
4. **Decrement readers. If last reader, release write_sem.**

Writer Workflow

1. **Increment waiting_writers and queue for write_sem.**
2. **On acquiring write_sem, decrement waiting_writers and increment writers.**
3. **Write data.**
4. **Decrement writers and release write_sem.**

Reader Entry/Exit:

```
// Entry
sem_wait(&data.mutex);
if (data.waiting_writers > 0 || data.writers > 0) {
   sem_post(&data.mutex);
   continue; // Retry
}
data.readers++;
if (data.readers == 1) sem_wait(&data.write_sem);
sem_post(&data.mutex);

// Exit
sem_wait(&data.mutex);
data.readers--;
if (data.readers == 0) sem_post(&data.write_sem);
sem_post(&data.mutex);
```

Writer Entry/Exit:

```
// Entry
sem_wait(&data.mutex);
data.waiting_writers++;
sem_post(&data.mutex);

sem_wait(&data.write_sem);

sem_wait(&data.mutex);
data.waiting_writers--;
data.writers++;
sem_post(&data.mutex);

// Exit
sem_wait(&data.mutex);
data.writers--;
sem_post(&data.mutex);
sem_post(&data.write_sem);
```

Expected Behavior

- **Readers**:
    - Concurrent access only when no writers are active/waiting.
    - Immediate blocking if writers queue up.
- **Writers**:
    - Exclusive, sequential access.
    - Priority over readers.

Sample Output

Reader 1 reads: 0
Reader 2 reads: 0
Writer 1 writes: 1
Writer 2 writes: 2
Reader 1 reads: 2
Reader 2 reads: 2

Explanation:

1. Initial readers proceed with no contention.
2. Writers preempt readers once they arrive.
3. Readers resume only after all writers finish.

Comparison Summary

Aspect	Reader-Priority	Writer-Priority
Reader Concurrency	Allowed concurrently	Blocked if writers pending
Writer Starvation	Possible	Impossible
Throughput	High for reads	Balanced

Conclusion

The writer-priority strategy eliminates writer starvation by enforcing strict precedence. Choose:

- **Reader-priority** for read-dominant workloads (e.g., caches).
- **Writer-priority** for write-critical systems (e.g., databases).