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 that writers are given precedence over readers to avoid starvation. When a writer indicates an intention to write, subsequent reader entries are blocked until all waiting writers have completed their operations.

Implementation Details

Additional synchronization variables are introduced:

- waiting_writers: An integer counter that tracks the number of writers waiting to write.
- read_sem: A semaphore that allows or blocks readers depending on the presence of waiting writers.
- mutex: A semaphore to ensure atomic updates to the shared counters.

Writer Entry Section

```
sem_wait(&data.mutex);
waiting_writers++;
if (waiting_writers == 1)
   sem_wait(&data.read_sem); // First waiting writer blocks readers
sem_post(&data.mutex);
```

Writer Exit Section

```
sem_wait(&data.mutex);
waiting_writers--;
if (waiting_writers == 0)
   sem_post(&data.read_sem); // Last writer allows readers again
sem_post(&data.mutex);
```

Expected Behavior

- Multiple readers may access the shared memory simultaneously when no writers are waiting.
- As soon as a writer begins waiting, new reader entries are blocked until all writers complete.
- Writers are given access in sequence, preventing writer starvation.

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:
- Readers initially access the shared memory.
- Once Writer 1 and Writer 2 arrive, they are given priority.
- Readers resume only after all waiting writers complete.

Comparison Summary

Conclusion

Both synchronization strategies effectively manage concurrent access to shared memory among readers and writers. The reader-priority approach favors concurrent reading but may lead to writer starvation. In contrast, the writer-priority strategy ensures fair access for writers at the cost of reader delays. The appropriate choice between the two depends on the system requirements and desired fairness constraints.

Aspect	Reader-Priority	Writer-Priority
Reader Concurrency	Multiple readers can access concurrently	Multiple readers can access concurrently
Writer Exclusivity	Writers have exclusive access	Writers have exclusive access
Reader Precedence	Readers admitted even if writers are waiting	Readers blocked if writers are waiting
Writer Precedence	Writers may starve due to frequent readers	Writers prioritized over readers
Risk of Starvation	Writers may starve	Readers may face delays