Course : Operating Systems **Instructor:** 黃能富

Student name: Tuguldur Tserenbaljir **Assignment:** Assignment 4

Student ID: 109006271

Problem Statement:

For the second problem in the Reader and Writers scenario, the main challenge is to ensure that once a writer is ready, they should be able to perform their write as soon as the currently active readers or writer have completed their operations. This setup must also prevent new readers from starting while a writer is waiting, addressing potential reader starvation.

Solution Approach:

The proposed algorithm uses semaphores to solve this issue by managing access to the shared resource and ensuring that writers can proceed with minimal wait time once they signal readiness. Here's how it works (Table 1- Pseudo Code):

```
Initialize:
    read_count = 0
    resource_access = Semaphore(1)
    read_try = Semaphore(1)
    service_queue = Semaphore(1)

Reader Process:
    P(service_queue)         // Wait on service queue to ensure order
    P(read_try)              // Check if entry is allowed (blocked if writer is waiting)
    P(resource_access)       // Lock access if they are the first reader
    if read_count == 0:
        P(resource_access)   // First reader locks the resource
    V(service_queue)         // Release service queue to allow the next in line
    read_count += 1
    V(read_try)              // Allow next readers to proceed
    V(resource_access)       // Release resource if first reader

    // Reading is performed here (Critical Section)

    P(resource_access)       // Lock to change read_count safely
    read_count -= 1
    if read_count == 0:
        V(resource_access)   // Last reader unlocks the resource
    V(resource_access)       // Release resource if last reader

Writer Process:
    P(service_queue)         // Ensure order
    V(service_queue)         // Release service queue to not block others
    P(read_try)              // Block new readers from starting
    P(resource_access)       // Lock the resource for exclusive access

    // Writing is performed here (Critical Section)

    V(resource_access)       // Release the resource for others
    V(read_try)              // Allow readers to proceed
```

Course : Operating Systems **Instructor:** 黃能富

Student name: Tuguldur Tserenbaljir **Assignment:** Assignment 4

Student ID: 109006271

1. **Initialize Semaphores and Variables:**

   1.1) **read_count:** Tracks the number of readers currently accessing the resource.

   1.2) **resource_access:** A semaphore that ensures mutual exclusion for writers or the first/last reader.

   1.3) **read_try:** A semaphore that helps in prioritizing writers over new readers when a writer is waiting.

   1.4) **service_queue:** A semaphore to ensure fairness and avoid potential starvation by managing the order of requests.

2. **Reader Process**

Entry Section:

- Wait on service_queue to ensure order and fairness.
- Wait on **read_try** to check if they can attempt to read (this is locked if a writer is waiting).
- Wait on resource_access only if they are the first reader.
- Increment **read_count**.
- If the first reader, release **resource_access**.
- Signal **service_queue** to allow the next process.

Critical Section: (Reading happens here)

Exit Section:

- Wait on **resource_access** if they are the last reader.
- Decrement **read_count**.
- If the last reader, signal resource_access to allow writers to proceed.
- Signal **read_try** to allow next readers if no writers are waiting.

3. Writer Process

   Entry Section:

   - Wait on **service_queue** to ensure order.

- Signal **service_queue** (to not hold other readers or writers).
- Wait on **read_try** to prevent new readers from starting.
- Wait on **resource_access** to ensure exclusive access.

Critical Section: (Writing happens here)

Exit Section:

- Signal **resource_access** to allow next writers or readers.
- Signal **read_try** to allow readers to start if no other writers are waiting.

4. Explanation:

   In solving the second Reader and Writer problem, our algorithm employs three key semaphores: **service_queue**, **read_try**, and **resource_access**, to manage access and ensure fairness. The **service_queue** semaphore is critical as it controls the entry sequence for both readers and writers, ensuring that requests are served in the order they arrive and preventing starvation. This orderly queuing is crucial for ensuring that each participant has an equal opportunity to access the resource without undue delay.

   The **read_try** semaphore plays an essential role in prioritizing writers. When a writer arrives, this semaphore is locked to prevent new readers from starting, allowing the writer to begin as soon as current sessions end. This setup is vital to avoid writer starvation, particularly in high reader activity scenarios.

   Additionally, the **resource_access** semaphore allows either multiple readers to access the resource simultaneously or grants exclusive access to a single writer. This flexibility ensures that the resource is used efficiently without conflicts.

   Overall, the algorithm effectively addresses the need for writer prioritization while maintaining fairness among all users through structured access control. This approach not only prevents potential starvation of writers but also supports a balanced resource sharing environment.