This Producer-Consumer project implements synchronization problem using separate producer and consumer programs communicating through POSIX shared memory and semaphores. The buffer—a "table"—holds at most two items simultaneously.

Program Description

The system consists of a producer that generates items and a consumer that consumes them. The producer places items onto the shared buffer, waiting if the buffer is full (2 items), while the consumer picks up items, waiting if the buffer is empty. Both processes synchronize using semaphores to prevent race conditions and ensure mutual exclusion during buffer access.

Key Components Explanation

- Shared Memory Buffer

A circular buffer of fixed size 2 exists in POSIX shared memory. It stores the produced items. Two indices, in and out, track the positions where the producer inserts and the consumer removes items. Circular arithmetic ensures wrap-around, keeping the buffer continuous.

- Semaphores

Three named POSIX semaphores control access:

  - empty_sem initialized to 2, counting free slots in the buffer
  - full_sem initialized to 0, counting filled slots in the buffer
  - mutex_sem initialized to 1, ensuring mutual exclusion when accessing the buffer to avoid race conditions

- Producer Program:

Opens/creates shared memory and semaphores. In a loop, it waits on empty_sem and mutex_sem, inserts an item at in, updates in index circularly, then posts mutex_sem and full_sem to signal new data availability.

- Consumer Program:

Opens existing shared memory and semaphores. In a loop, it waits on full_sem and mutex_sem, consumes the item at out, updates out index circularly, then posts mutex_sem and empty_sem to signal a free slot.

- Synchronization Logic:

This approach ensures the producer waits if the buffer is full and the consumer waits if empty, while only one accesses buffer data at a time, preserving data integrity and avoiding deadlock or data loss.