Electrical and Computer Engineering Department

ENCS339 Operating Systems

The first assignment: Shared Memory

Prepared by:

- Tala Flaifel __ 1201107
- Lana Batnij __ 1200308

Instructor: Dr. Bashar Tahayna

Section: 1

Date: 9th/May/2023

Abstract:

In the C programming language, we were asked to build a solution to the constrained buffer producer-consumer dilemma utilizing shared memory and fork/exec routines. So we created The producer file, which forks a consumer process and replaces the child process with the appropriate consumer code using exec (), Following that, we write test cases to run the code in various scenarios to check if there is an issue.

Table of Contents:

Theory:

Shared memory:

In computer science, shared memory is a memory that may be simultaneously accessed by multiple programs with the intent to provide communication among them or avoid redundant copies. Shared memory is an efficient means of passing data between programs. Depending on the context, programs may run on a single processor or on multiple separate processors. Using memory for communication inside a single program, e.g. among its multiple threads, is also referred to as shared memory [1].

Bounded Buffer problem:

The bounded-buffer problem (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer. Producers must block if the buffer is full. While Consumers must block if the buffer is empty [2].

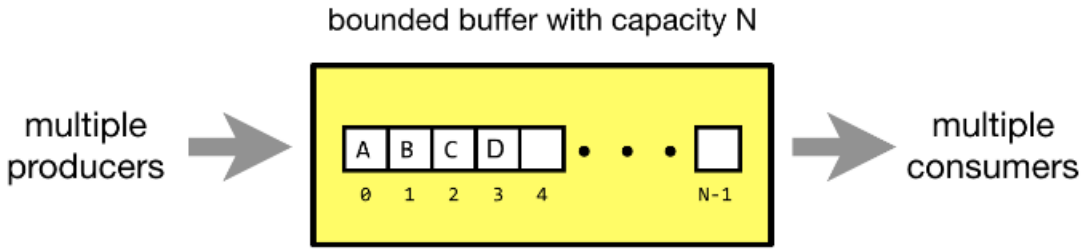

Figure 2.1: the bounded buffer [2]

Synchronization:

A bounded buffer with capacity N has can store N data items. The places used to store the data items inside the bounded buffer are called slots. Without proper synchronization, the following errors may occur [2].

- The producers don't block when the buffer is full.
- A Consumer consumes an empty slot in the buffer.
- A consumer attempts to consume a slot that is only half-filled by a producer.
- Two producers write into the same slot.
- Two consumers read the same slot.
- And possibly more …

Our problem and the requirements:

The bounded buffer producer-consumer dilemma includes a shared circular buffer in which producers contribute items and consumers delete things. The buffer has a set capacity, and producers and consumers must work together to ensure that it does not overflow (i.e., producers must wait when the buffer is full) or underflow (i.e., consumers must wait when the buffer is empty). The circular buffer and synchronization objects are stored in shared memory in this version of the problem. However, using a semaphore or mutex to fix the synchronization problem is not recommended.

The requirements:

1. Using shared memory, create a circular buffer of fixed size N (e.g., N=10).

2. Create a producer function that inserts things into the buffer.

3. Create a main function that forks a consumer process and starts it with the appropriate consumer code using exec ().

4. Ensure that when the buffer is full, the producers wait, and when the buffer is empty, the consumers wait.

5. Make certain that neither producers nor consumers use the buffer at the same time (i.e., mutual exclusion).

6. Check that the shared memory has been correctly initialized and cleaned up.

7. Verify that your solution works properly by testing it with numerous producers and customers.

8. Create a consumer code (separate C program) that removes things from the buffer (to be transferred to the child you produced in step 3).

9. Have your application pause/resume the producer when you press the "p" key and the consumer when you press the "c" key.

Discussion of our proposed solution:

We accomplish the approach by creating a circular buffer with a defined size in shared memory. Producers and consumers use shared memory to add and delete objects from the buffer. To maintain appropriate synchronization, the producers wait until the buffer is filled and the consumers wait until the buffer is empty. Mutual exclusion is achieved by creating a separate process for the consumer using the fork () and exec () methods. To prevent memory leaks, shared memory is correctly established and cleaned away. But keep in mind that the producer and shared memory are in the same file, whereas the is in another.

It is tested with multiple producers and consumers to ensure that our solution is right and this by randomizing the producers. We also allowed the user to stop and restart the producer and consumer processes by pressing the "p" and "c" keys. Overall, the method presents a solid and fast solution to the bounded buffer producer-consumer problem utilizing shared memory and fork/exec functions in the C programming language, without the need for semaphores or mutexes for synchronization.

Note: the code itself, and the video that shows the outputs will be sent beside this report.

References:

[1]https://en.wikipedia.org/wiki/Shared_memory#:~:text=In%20computer%20science%2C%20shared%20memory,of%20passing%20data%20between%20programs. [Accessed at 5:40 pm,7th/May/2023]

[2] https://it.uu.se/education/course/homepage/os/vt18/module-4/bounded-buffer/#:~:text=The%20bounded-buffer%20problems%20(aka,read%20data%20from%20the%20buffer. [Accessed at 5:40 pm, 7th/May/2023]