NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (FAST-NUCES)

OPERATING SYSTEM PROJECT PROPOSAL (2021)

Group members: Hasnain Somani (19k-0204)

Ashmal Anis (19k-0305)

Project name: System Call for Producer Consumer Problem

Introduction:

The producer Consumer Problem is very classical problem in computer science that is used for multi-process synchronization (synchronization between more than one things). In this particular problem, we have 2 ends, one for producer that is producing something and another for consumer who is consuming something. In the bounded buffer problem, the producer and consumer share the same memory space from a fixed buffer. The communication is fine until certain problems arises:

1) The producer should only be allowed to put some data if the buffer is not full. If the buffer is full, then producer should wait until there's some space available in the buffer or the consumer consumes it.
2) The consumer should only consume data when the buffer is not empty. If the buffer is empty, then the consumer shouldn't be allowed to access the buffer.
3) Producer and consumer shouldn't access the buffer at the same time.

Methodology:

Processes usually communicate through an external buffers. A producer writes to one end of the buffer, and the consumer reads from the other end. This approach was an efficient approach, but it had multiple problems. An issue faced could be a consumer trying to read from an empty buffer, or a producer trying to write inside a full buffer. This results in unexpected results- which is why bounded buffers have been introduced to make the producer consumer approach more efficient. In a producer consumer problem using bounded buffer, we have a fixed size buffer, having a producer write to one end of the buffer, and the consumer read from the other end. We will use 2 counting semaphores – one to count the number of empty spaces, and one to count the number of spaces filled with data. This counting would be done through signal() and wait() function calls. The use of semaphores will also help us implement the bounded buffer property that a consumer doesn't attempt to read if the buffer is empty, and a producer doesn't attempt to write if the buffer is full.