## Operating Systems Project 2

Name:

Institution:

Course:

Instructor’s name:

Submission date:

Table of contents

1. Introduction………………………………………………………………………………..3
2. System documentation…………………………………………………………………….4
3. Flow diagram of the system……………………………………………………….4
4. List of routines…………………………………………………………………….5
5. Implementation……………………………………………………………………6
6. Test documentation……………………………………………………………………….8
7. User documentation……………………………………………………………………….8

# INTRODUCTION

The Internet Protocol (IP) is a worldwide network layer developed for delivering packets to specific destinations, allowing for communication between processes. This laid the groundwork for the introduction of the socket, which connects a running application to a communications backbone and stores information about the transmission of data between two separate programs (any necessary data may travel back and forth from a program running on one computer to a program running on another; in many cases, this is to and from a client and server).

The Internet has evolved from a network connecting servers to one connecting information objects with varying levels of interpretation. This change has necessitated the implementation of a data dissemination protocol at the Internet's global network layer that is capable of interacting with information objects rather than merely communication endpoints.

In this research, we show the architecture of a new application programming interface (API) and the accompanying protocol suite that may carry out tasks similar to those performed by sockets in an NDN network. Briefly, our contributions are as follows:

• A user- and producer-friendly programming strategy tailored to the delivery of data.

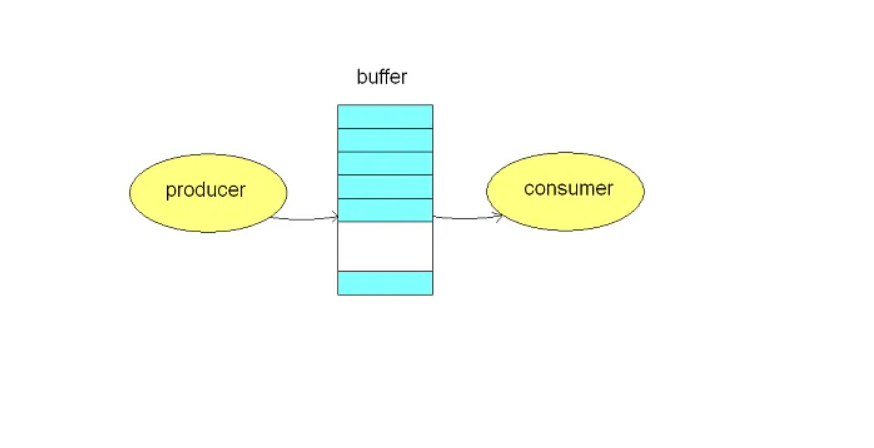
Data retrieval and content categorization protocols.

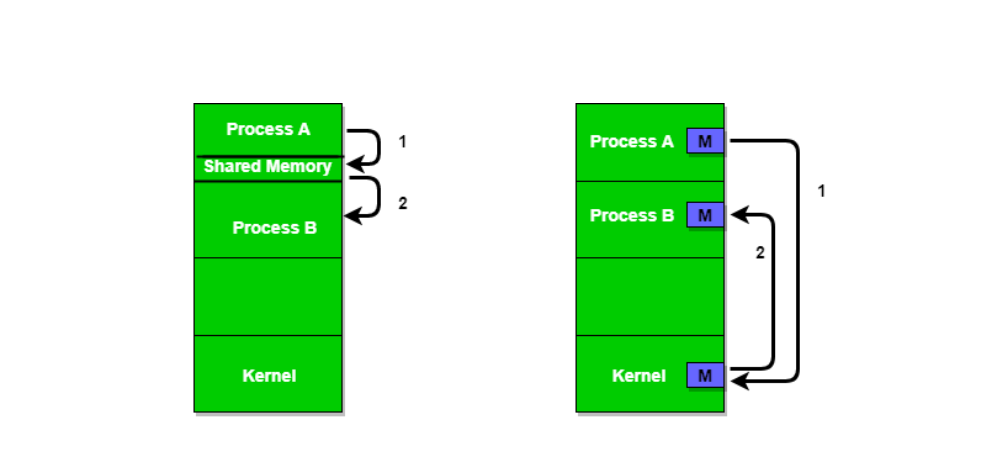
• A number of supplementary instruments, such as a manifest and negative acknowledgement, used by the protocols underlying the API to enhance application functionality.

In addition to developing the API and protocols, we have built and tested several working prototypes of consumer and producer apps. The success of our evaluation depends on the existence of fully functional, real-world applications, and on the ability to objectively measure the computational overhead suffered by a single producer who publishes ADUs for multiple consumers.

1. **System documentation**

**i). flow diagram for the whole system**





1. **A list of routines**

Global variables

-int \* buffer (pointer to an array that will serve as a buffer),

-int buff\_size (the size of the buffer set by the user),

-int num\_producer (number of producers set by the user),

-int num\_items (number of products set by the user),

-int count, in, out (variables used to handle the buffer, the number of products in the buffer, indicator i the first and last element of the buffer),

-semafory mutex, full, empty (they are used to secure the buffer so that only one thread can use it),

Functions

1. Int insert\_item (int item)

Function for putting products into the buffer. It only allows one thread to access the buffer using semaphores. Returns 0 on success and -1 on failure.

1. Int remove\_item (int \* item)

Function for removing products from the buffer. It only allows one thread to access the buffer using semaphores. Returns 0 on success and -1 on failure.

1. Void \* producer (void)

Function to create a producer thread. It performs the function of adding to the buffer a specified number of times.

1. Void \* consumer (void)

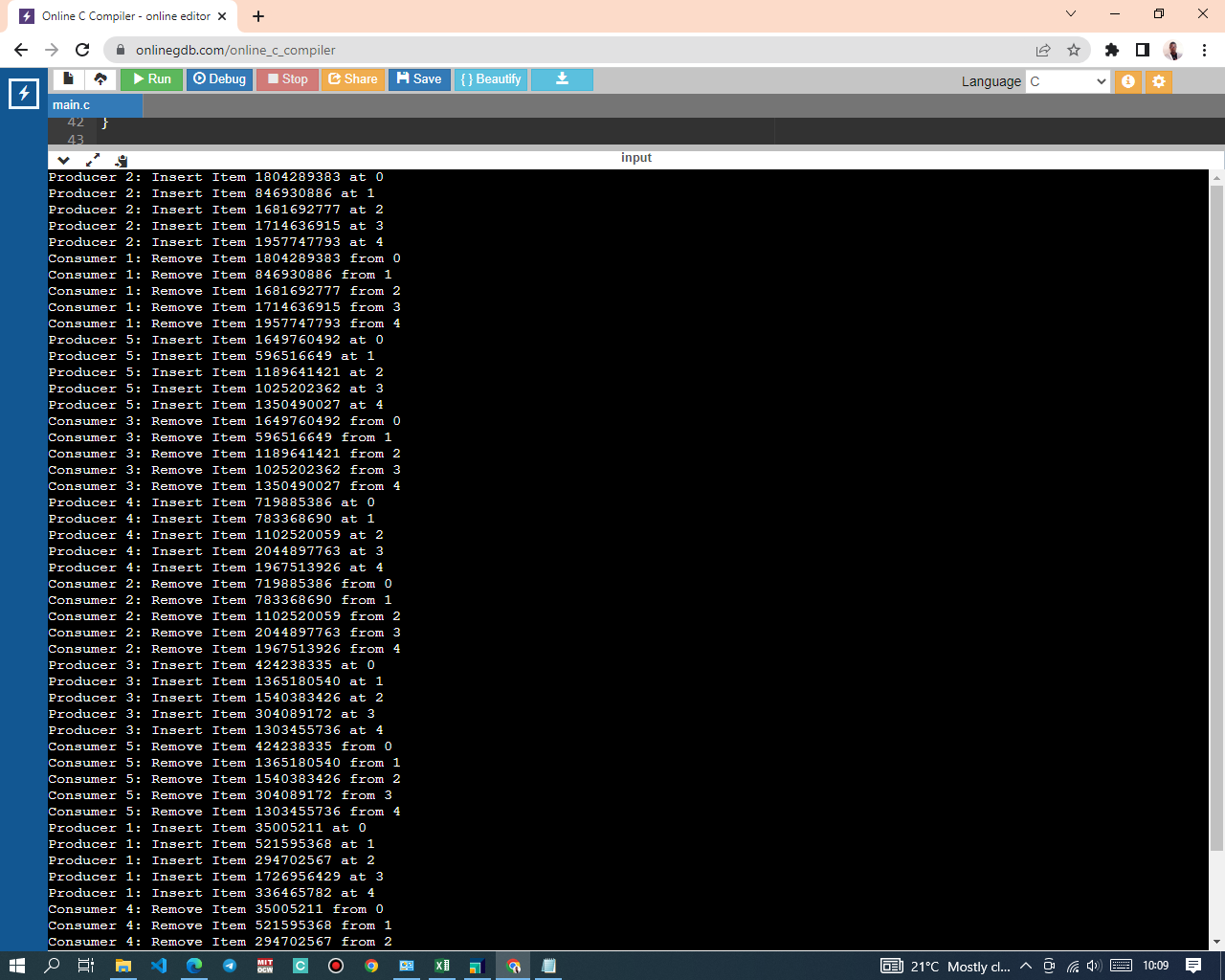
Function to create a consumer thread. It performs the unbuffer function as many times as the number of producers multiplied by the number of products.

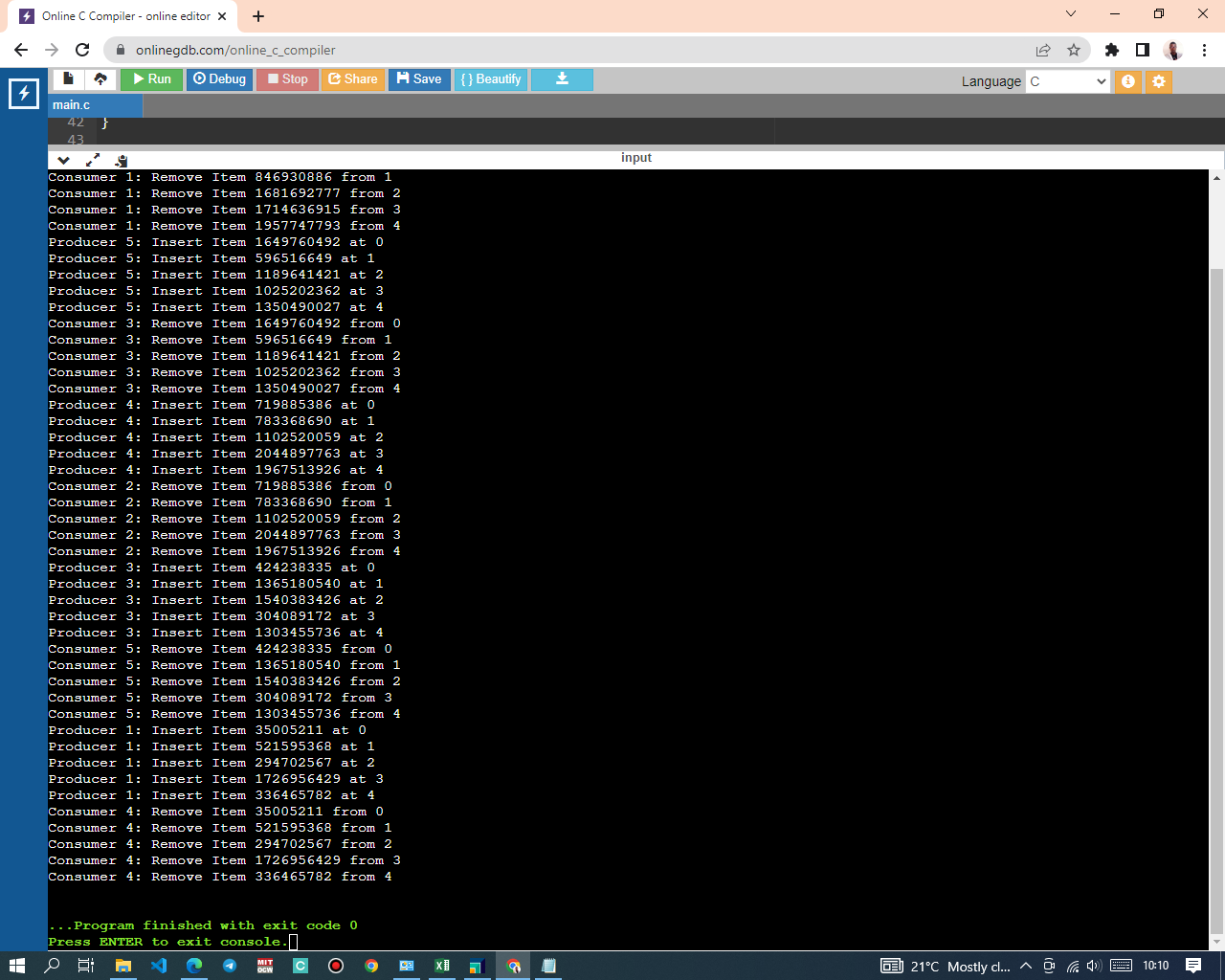
1. Int setId (void)

Function for assigning id numbers to producer processes.

1. **Implementation**

The program successfully implements all the pipes and produces an output shown below from our test.

****



# Test documentation

The problem of matching production with demand is a prototypical example of synchronization. This program does a C-based simulation of the Multi Producer-Consumer Problem. Our program was put to the test with these. There are no missing files just extract the package from the zip file.

# User documentation

Producing data, placing it in the buffer, and then beginning again is the producer's job.

The data is being consumed, or removed from the buffer, simultaneously by the consumer. If you unpack the zip file and run the code within, you should get the whole story.