Operating Systems Project 2

Name:

Institution:

Course:

Instructor’s name:

Submission date:

Table of contents

1. 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.

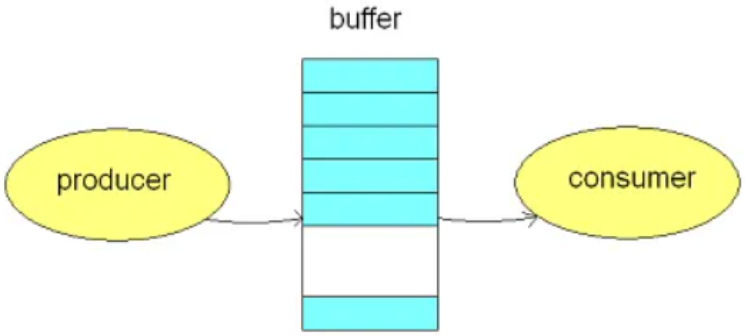
Data retrieval and content categorization protocols.

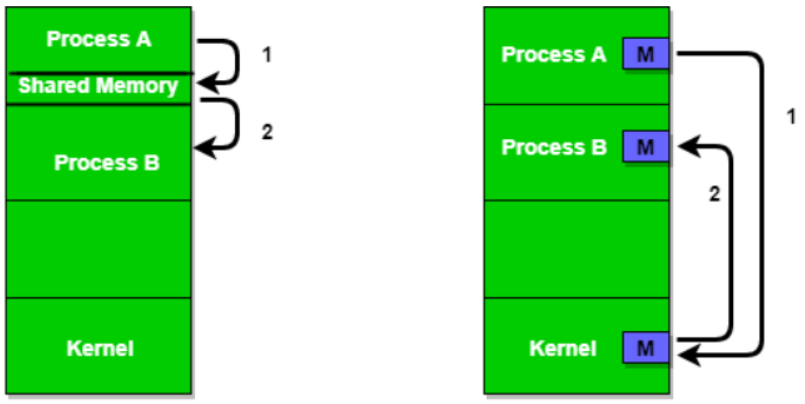
• A number of supplementary instruments, such as a manifest and negative acknowledgement, used by the protocols underlying the API to enhance application functionality.

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.

2. **System documentation**

i). flow diagram for the whole system





i. 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.

2. 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.

3. Void * producer (void)

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

4. Void * consumer (void)

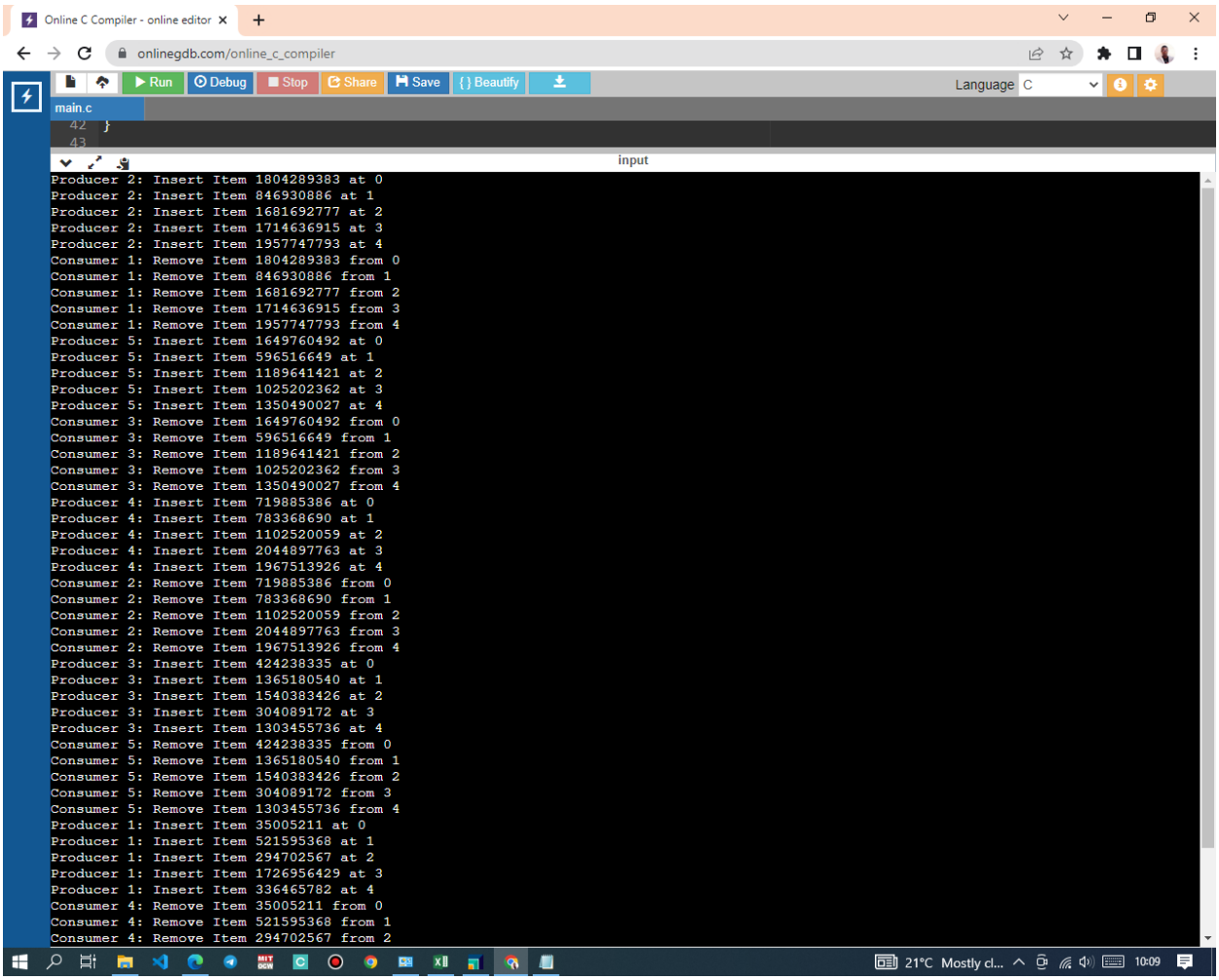
   Function to create a consumer thread. It performs the unbuffer function as many times as the number of producers multiplied by the number of products.

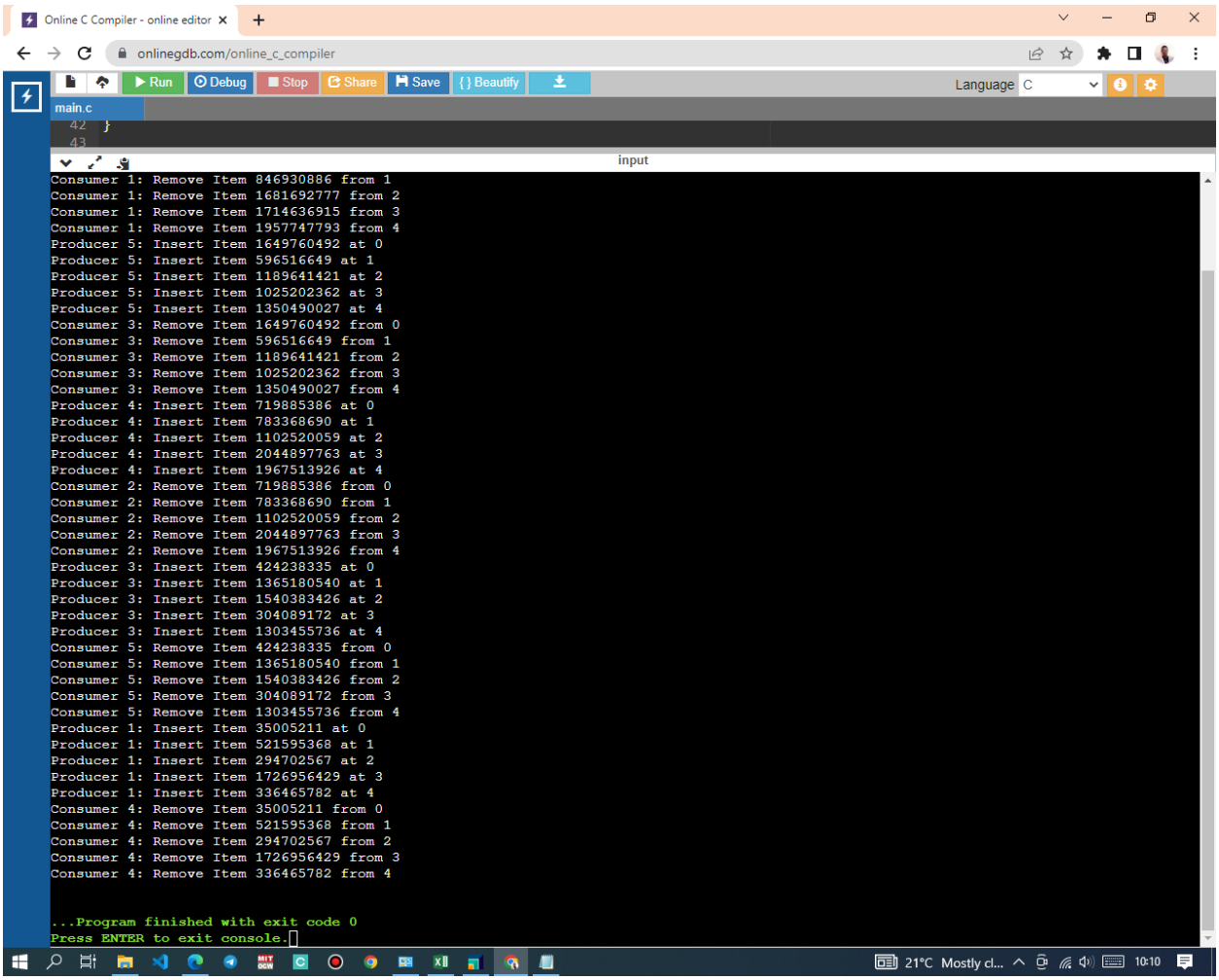
5. Int setId (void)

   Function for assigning id numbers to producer processes.

ii. Implementation

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


Online C Compiler - online editor
onlinegdb.com/online_c_compiler
Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
42 }
43
input
Producer 2: Insert Item 1804289383 at 0
Producer 2: Insert Item 846930886 at 1
Producer 2: Insert Item 1681692777 at 2
Producer 2: Insert Item 1714636915 at 3
Producer 2: Insert Item 1957747793 at 4
Consumer 1: Remove Item 1804289383 from 0
Consumer 1: Remove Item 846930886 from 1
Consumer 1: Remove Item 1681692777 from 2
Consumer 1: Remove Item 1714636915 from 3
Consumer 1: Remove Item 1957747793 from 4
Producer 5: Insert Item 1649760492 at 0
Producer 5: Insert Item 596516649 at 1
Producer 5: Insert Item 1189641421 at 2
Producer 5: Insert Item 1025202362 at 3
Producer 5: Insert Item 1350490027 at 4
Consumer 3: Remove Item 1649760492 from 0
Consumer 3: Remove Item 596516649 from 1
Consumer 3: Remove Item 1189641421 from 2
Consumer 3: Remove Item 1025202362 from 3
Consumer 3: Remove Item 1350490027 from 4
Producer 4: Insert Item 719885386 at 0
Producer 4: Insert Item 783368690 at 1
Producer 4: Insert Item 1102520059 at 2
Producer 4: Insert Item 2044897763 at 3
Producer 4: Insert Item 1967513926 at 4
Consumer 2: Remove Item 719885386 from 0
Consumer 2: Remove Item 783368690 from 1
Consumer 2: Remove Item 1102520059 from 2
Consumer 2: Remove Item 2044897763 from 3
Consumer 2: Remove Item 1967513926 from 4
Producer 3: Insert Item 424238335 at 0
Producer 3: Insert Item 1365180540 at 1
Producer 3: Insert Item 1540383426 at 2
Producer 3: Insert Item 304089172 at 3
Producer 3: Insert Item 1303455736 at 4
Consumer 5: Remove Item 424238335 from 0
Consumer 5: Remove Item 1365180540 from 1
Consumer 5: Remove Item 1540383426 from 2
Consumer 5: Remove Item 304089172 from 3
Consumer 5: Remove Item 1303455736 from 4
Producer 1: Insert Item 35005211 at 0
Producer 1: Insert Item 521595368 at 1
Producer 1: Insert Item 294702567 at 2
Producer 1: Insert Item 1726956429 at 3
Producer 1: Insert Item 336465782 at 4
Consumer 4: Remove Item 35005211 from 0
Consumer 4: Remove Item 521595368 from 1
Consumer 4: Remove Item 294702567 from 2
21°C Mostly cl...
10:09



3. 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.

4. 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.