Design Document

Operating Systems - Assignment 1, Question 12

Eric Tsang, A00841554, 4O

|  |
| --- |
| Table of Contents  [State Diagrams 2](#_Toc411157583)  [Producer 2](#_Toc411157584)  [Consumer 3](#_Toc411157585)  [Circular Buffer 4](#_Toc411157586)  [Pseudocode 5](#_Toc411157587)  [Producer Thread 5](#_Toc411157588)  [Threaded Function 5](#_Toc411157589)  [Interrupt or Signal Handler 5](#_Toc411157590)  [Consumer Thread 5](#_Toc411157591)  [Threaded Function 5](#_Toc411157592)  [Interrupt or Signal Handler 5](#_Toc411157593)  [Circular Buffer 6](#_Toc411157594)  [Add Data or Object Method 6](#_Toc411157595)  [Remove Data or Object Method 6](#_Toc411157596) |

# State Diagrams

This section contains a finite state machine for each of the main classes in the program. These classes include:

* **Producer**; creates objects, and places them into the shared queue object.
* **Consumer**; takes objects from the shared queue object.
* **Queue**; the queue object that is shared among the producers and consumers.

## Producer



This is the finite state machine for the producer thread:

* **Produce State**; the starting state for this thread. Here, it instantiates a product, and then moves on to the enqueue state.
* **Enqueue State**; the thread calls the enqueue method if the Queue monitor. If the queue is full, this thread gets blocked, and enters the wait state. When the enqueue operation succeeds, then the thread logs this to standard out, and moves on to the sleep state.
* **Wait State**; here, the thread is waiting to be notified by the queue before it attempts to enqueue again. The thread can be interrupted in this state.
* **Sleep State**; the thread sleeps for a random amount of time to make to simulate asynchronous and unpredictable production. The thread can be interrupted in this state.

During the sleep or wait states, the thread can be interrupted. Once interrupted, the thread terminates, and is no longer executed.

## Consumer



The finite state machine for the Consumer thread:

* **Dequeue State**; starting state of the thread. In this state, it attempts to take an object from the shared queue. If the queue is empty, this thread becomes blocked, and moves to the wait state. When the dequeue operation completes, then it moves on to the consume state.
* **Consume State**; parsing of the dequeued object is done here. Once parsed, the thread logs it to standard out, and moves on to the sleep state.
* **Wait State**; here, the thread is waiting to be notified by the queue before it attempts to dequeue again. The thread can be interrupted in this state.
* **Sleep State**; the thread sleeps for a random amount of time to make to simulate asynchronous and unpredictable production. The thread can be interrupted in this state.

During the sleep or wait states, the thread can be interrupted. Once interrupted, the thread terminates, and is no longer executed.

## Circular Buffer

The queue class is the heart of the program; it is a monitor. This is a finite state machine of the thread-safe queue class. This class has 3 main states where it would spend most of its time in:

* **Empty State**; the queue is empty, and has no elements. Any attempts to dequeue will cause a wait to be issued, blocking the dequeueing thread. When enqueueing, a waiting thread will be notified, and the enqueue operation completed.
* **Normal State**; the queue is neither empty, or full. Both enqueue and dequeue operations can be done without blocking. Even so, each enqueue and dequeue in this state will issue a notification to a waiting thread, but in this state, there should be no waiting threads anyway.
* **Full State**; the queue is full. Any attempts to enqueue will cause a wait to be issued, blocking the enqueueing thread. When dequeueing, a waiting thread will be notified, and the dequeue operation completed.

# Pseudocode

Pseudocode for the main classes of the Consumer Producer program.

## Producer Thread

Here is the pseudocode from the producer thread.

### Threaded Function

The following pseudocode should be put on a thread, and executed concurrently:

1. forever for loop

generate data to put into the shared circular buffer monitor object

1. using a shared circular buffer method, put the generated data into the circular buffer

print a message to the screen about the data, and position the data was put into the circular buffer

1. sleep for a random amount of time

### Interrupt or Signal Handler

The following pseudocode is for handling any interrupts or signals received while the thread is running:

1. print to the screen informing the user that this thread has been interrupted or signaled

terminate the thread normally

## Consumer Thread

Here is the pseudocode from the consumer thread.

### Threaded Function

The following pseudocode should be put on a thread, and executed concurrently:

1. forever for loop

using a shared circular buffer method, remove data from the circular buffer

1. parse the data obtained from the circular buffer

print a message to the screen about the data, and position the data was taken from in the circular buffer

1. sleep for a random amount of time

### Interrupt or Signal Handler

The following pseudocode is for handling any interrupts or signals received while the thread is running:

1. print to the screen informing the user that this thread has been interrupted or signaled

terminate the thread normally

## Circular Buffer

The Circular Buffer is a thread-safe class; it is a monitor. Elements are added and removed in the FIFO order, like a queue. Instances contain an array that is encapsulated within them. The array can only be accessed only by their methods. This section contains the pseudocode for its most important methods.

### Add Data or Object Method

This method is used to put some data or an object into an element of the buffer, so it can be removed by the remove method later.

1. if the buffer is full, wait on the **notfull** condition

signal any thread that's waiting on the **notempty** condition

1. add the passed data or object to the queue in the next available spot

### Remove Data or Object Method

This method is used to take some data or an object out of an element in the buffer, creating more space in the buffer for more data or another object to be added into it.

1. if the buffer is empty, wait on the **notempty** condition

signal any thread that's waiting on the **notfull** condition

1. remove and return the data or object that has been in the buffer the longest