CSC433 Project 2: Producer-Consumer Synchronization

Team: Adam Morehouse, Ryan Fosco, Brandon Coulter

Overview

This document describes the testing and results for Project 2 in CSC433 (Operating Systems). The project implements a producer-consumer system in Python using semaphores and locks to synchronize threads. Producers read transactions from a file, append them to a FIFO queue, and consumers process them in order, respecting sleep times and FIFO depth constraints. The implementation ensures:

- Synchronization between producers to avoid duplicate or missed transactions.
- Synchronization between producers and consumers to respect FIFO depth (stopping producers when full, consumers when empty).
- End-of-file (EOF) handling to terminate all threads cleanly.
- Support for multiple producers and consumers with various FIFO depths and transaction files.

The tests below verify these requirements across different scenarios, including single and multiple producers/consumers, varying FIFO depths, and different transaction sleep times.

Implementation Notes

At first, our code had a bug where it would hang, especially when we had more consumers, like 1 producer and 2 consumers. Some consumers got stuck waiting for more transactions after the `9999` one was done.

We fixed it by:

- Keeping track of how many producers were still running and setting an EOF flag when they all finished.
- Making sure all consumers got a wake-up call (by releasing the `items` semaphore) when a producer was done or when `9999` was processed.
- Having consumers check if the FIFO was empty *and* EOF was set to know when to quit.

We used these to make it work:

- `items` semaphore to count transactions in the FIFO.
- `spaces` semaphore to track free spots in the FIFO.
- `mutex` lock to protect the FIFO.
- `eof_mutex` lock for the EOF flag.
- `producers_mutex` lock to count active producers. All our tests now finish properly, as you'll see below.

All our tests now finish properly, as you'll see below.

Test Cases

The following table summarizes the test cases, including commands, input files, thread counts, FIFO depth, and the purpose of each test.

Test #	Command	Input File	Producers	Consumers	FIFO Depth	Purpose
1	`python project2.py transactions.txt 1 1 1`	transactions.txt	1	1	1	Tests basic synchronization with minimal FIFO depth, ensuring strict producer-consumer alternation.
2	`python project2.py transactions.txt 1 2 5`	transactions.txt	1	2	5	Tests one producer with multiple consumers, verifying load balancing and EOF termination.
3	`python project2.py transactions.txt 2 1 5`	transactions.txt	2	1	5	Tests multiple producers with one consumer, checking for no race conditions in file reading.
4	`python project2.py transactionsMedium.txt 2 2 3`	transactionsMedium.txt	2	2	3	Tests balanced producers and consumers with moderate sleep times and smaller FIFO depth.
5	`python project2.py transactionsHeavy.txt 3 3 10`	transactionsHeavy.txt	3	3	10	Stress-tests with multiple threads, large FIFO depth, and long sleep times.
6	`python project2.py transactionsSingle.txt 1 1 1`	transactionsSingle.txt	1	1	1	Tests minimal case with one transaction plus termination.
7	`python project2.py transactionsEasy.txt 1 2 2`	transactionsEasy.txt	1	2	2	Tests provided easy file with multiple consumers and small FIFO depth.

Transaction Files

The following transaction files were used:

- transactions.txt - Purpose: Standard test file with multiple transactions and varying sleep times.

0001,100,200
0002,200,200
0003,200,200
0004,300,200
0005,300,200
0006,300,200

0007,300,200
0008,300,200
0009,300,200
0010,300,200
9999,000,000

- **transactionsEasy.txt** - Purpose: Provided file with minimal producer sleep and long consumer sleep.

0001,1,1000
0002,1,1000
9999,1,000

- **transactionsMedium.txt** - Purpose: Tests moderate sleep times and fewer transactions.

0001,100,200
0002,150,250
0003,200,300
0004,250,350
9999,000,000

- **transactionsHeavy.txt** - Purpose: Stress-tests with long sleep times and more transactions.

0001,500,500
0002,500,500
0003,500,500
0004,500,500
0005,500,500
0006,500,500
0007,500,500
0008,500,500
9999,000,000

- **transactionsSingle.txt** - Purpose: Minimal test with one transaction.

0001,50,50
9999,000,000

Test Script

To simplify testing and reproduce results across all configurations, we created a Python script, `run_tests.py`, which runs all our test cases and saves the results to `test_results.txt`.

Purpose

This script automates running the 7 documented test cases using `subprocess.run()` in Python. It verifies the presence of required files, executes each test command, and logs the full terminal output into a timestamped results file.

This ensures:

- Tests are repeatable and consistent.
- All output is preserved for review.

- Timeouts are enforced to prevent indefinite hangs

Usage:

```
python run_tests.py
```

The script:

- Runs each documented test case (Tests 1–7).
- Redirects all output into `test_results.txt` .
- Checks that all required `.txt` and `.py` files are present before starting.

Test Results

Test 1: Basic Synchronization

Purpose: Make sure one producer and one consumer take turns with a FIFO that only holds one transaction.

Command: `python project2.py transactions.txt 1 1 1`

Output:

```
Output: Transaction file: transactions.txt
Starting producers: 1
Starting consumers: 1
Producer: 0001 internalId: 1
Consumer: 0001 internalId: 1
Producer: 0002 internalId: 2
Consumer: 0002 internalId: 2
Producer: 0003 internalId: 3
Consumer: 0003 internalId: 3
Producer: 0004 internalId: 4
Consumer: 0004 internalId: 4
Producer: 0005 internalId: 5
Consumer: 0005 internalId: 5
Producer: 0006 internalId: 6
Consumer: 0006 internalId: 6
Producer: 0007 internalId: 7
Consumer: 0007 internalId: 7
Producer: 0008 internalId: 8
Consumer: 0008 internalId: 8
Producer: 0009 internalId: 9
Consumer: 0009 internalId: 9
Producer: 0010 internalId: 10
Consumer: 0010 internalId: 10
Producer: 9999 internalId: 11
Consumer: 9999 internalId: 11
Producer completed
Consumer completed
```

Test 2: One Producer, Multiple Consumers

Command: `python project2.py transactions.txt 1 2 5`

Purpose: Confirms two consumers share transactions in order with FIFO depth 5. Both terminate after `9999`

Output:

```
Output: Transaction file: transactions.txt
Starting producers: 1
Starting consumers: 2
Producer: 0001 internalId: 1
Consumer: 0001 internalId: 1
Producer: 0002 internalId: 2
Consumer: 0002 internalId: 2
Producer: 0003 internalId: 3
Consumer: 0003 internalId: 3
Producer: 0004 internalId: 4
Consumer: 0004 internalId: 4
Producer: 0005 internalId: 5
Consumer: 0005 internalId: 5
Producer: 0006 internalId: 6
Consumer: 0006 internalId: 6
Producer: 0007 internalId: 7
Consumer: 0007 internalId: 7
Producer: 0008 internalId: 8
Consumer: 0008 internalId: 8
Producer: 0009 internalId: 9
Consumer: 0009 internalId: 9
Producer: 0010 internalId: 10
Consumer: 0010 internalId: 10
Producer: 9999 internalId: 11
Consumer: 9999 internalId: 11
Producer completed
Consumer completed
Consumer completed
```

Test 3: Multiple Producers, One Consumer

Command: `python project2.py transactions.txt 2 1 5`

Purpose: Verifies two producers read without conflicts, feeding one consumer in order (internal IDs 1-11).

Output:

```
Output: Transaction file: transactions.txt
Starting producers: 2
Producer: 0001 internalId: 1
Producer: 0002 internalId: 2
Starting consumers: 1
Consumer: 0001 internalId: 1
Producer: 0003 internalId: 3
Producer: 0004 internalId: 4
```

```
Consumer: 0002 internalId: 2
Producer: 0005 internalId: 5
Consumer: 0003 internalId: 3
Producer: 0006 internalId: 6
Producer: 0007 internalId: 7
Consumer: 0004 internalId: 4
Producer: 0008 internalId: 8
Consumer: 0005 internalId: 5
Producer: 0009 internalId: 9
Consumer: 0006 internalId: 6
Producer: 0010 internalId: 10
Producer: 9999 internalId: 11
Producer completed
Consumer: 0007 internalId: 7
Producer completed
Consumer: 0008 internalId: 8
Consumer: 0009 internalId: 9
Consumer: 0010 internalId: 10
Consumer: 9999 internalId: 11
Consumer completed
```

Test 4: Balanced Producers and Consumers

Command: `python project2.py transactionsMedium.txt 2 2 3`

Purpose: Tests two producers and two consumers with FIFO depth 3. All threads terminate cleanly (internal IDs 1-5).

Output:

```
Output: Transaction file: transactionsMedium.txt
Starting producers: 2
Producer: 0001 internalId: 1
Producer: 0002 internalId: 2
Starting consumers: 2
Consumer: 0001 internalId: 1
Consumer: 0002 internalId: 2
Producer: 0003 internalId: 3
Producer: 0004 internalId: 4
Consumer: 0003 internalId: 3
Consumer: 0004 internalId: 4
Producer: 9999 internalId: 5
Producer completed
Producer completed
Consumer: 9999 internalId: 5
Consumer completed
Consumer completed
```

Test 5: Stress Test

Command: `python project2.py transactionsHeavy.txt 3 3 10`

Purpose: Stress-test: Push the program with three producers, three consumers, long sleeps, and a big FIFO (10).

Output:

```
Transaction file: transactionsHeavy.txt
Starting producers: 3
Producer: 0001 internalId: 1
Producer: 0002 internalId: 2
Starting consumers: 3
Producer: 0003 internalId: 3
Consumer: 0001 internalId: 1
Consumer: 0002 internalId: 2
Consumer: 0003 internalId: 3
Producer: 0004 internalId: 4
Producer: 0005 internalId: 5
Producer: 0006 internalId: 6
Consumer: 0004 internalId: 4
Consumer: 0005 internalId: 5
Consumer: 0006 internalId: 6
Producer: 0007 internalId: 7
Consumer: 0007 internalId: 7
Producer: 0008 internalId: 8
Producer: 9999 internalId: 9
Consumer: 0008 internalId: 8
Producer completed
Consumer: 9999 internalId: 9
Consumer completed
Producer completed
Consumer completed
Producer completed
Consumer completed
```

Test 6: Minimal Case

Command: `python project2.py transactionsSingle.txt 1 1 1`

Purpose: Try the smallest setup with one transaction and a FIFO of 1

Output:

```
Transaction file: transactionsSingle.txt
Starting producers: 1
Starting consumers: 1
Producer: 0001 internalId: 1
Consumer: 0001 internalId: 1
Producer: 9999 internalId: 2
Consumer: 9999 internalId: 2
Producer completed
Consumer completed
```

Test 7: Easy File with Multiple Consumers

Command: `python project2.py transactionsEasy.txt 1 2 2`

Purpose: Test the easy file with one producer and two consumers, small FIFO (2).

Output:

```
Transaction file: transactionsEasy.txt
Starting producers: 1
Starting consumers: 2
Producer: 0001 internalId: 1
Consumer: 0001 internalId: 1
Producer: 0002 internalId: 2
Consumer: 0002 internalId: 2
Producer: 9999 internalId: 3
Producer completed
Consumer: 9999 internalId: 3
Consumer completed
Consumer completed
```

How We Did

- **Producers Working Together (25 points)**: Tests 3, 4, and 5 show producers don't step on each other.
- **Producers and Consumers in Sync (25 points)**: All tests prove the FIFO size is respected (full stops producers, empty stops consumers).
- **End of File Handling (25 points)**: Every test shuts down after `9999` or file end.
- **Test Files and Setups (15 points)**: We used five files with different sleeps and sizes.
- **Documentation (10 points)**: We included all commands, outputs, and what they mean. I used Docmost (self-hosted markdown editor) to create the documentation.
- **Bonus: Consumers Stopping Right (10 points)**: Tests 2, 4, 5, and 7 show consumers all quit properly.