Middle East Technical University

Department of Computer Engineering

CENG 334

Introduction to Operating Systems

Spring 2024–2025

HW - 2 | Store (Concurrent Programming)

Due date: 2025-05-04 23:59

1 Introduction

In this assignment, you will implement a market store that sells 3 types of items: AAA, BBB, and CCC. The store has a limited stock capacity for each item type. Contracted suppliers produce and add items to the stock to ensure store maintains availability of all item types.

Customers purchase items in batches, which may include zero or more of each item type. For example, one customer might buy a single AAA, while another might buy 1 AAA, 2 BBB, and 1 CCC. A customer's order is processed as a single transaction, delivering all requested items together.

Suppliers deliver items to the store in bulk packages. The store cannot purchase individual items from suppliers; instead, suppliers determine the number of items in each package. When the store's capacity for an item type falls to or below a supplier's package size, the supplier begins preparing the items. Once ready, the items are added to the store's stock.

In this assignment, customers are represented as threads placing batch orders, and suppliers are threads preparing and delivering item packages. You will implement the necessary synchronization mechanisms to manage these threads.

2 Store Details

The store is defined by the following parameters:

- `cap[AAA]`, `cap[BBB]`, `cap[CCC]`: The maximum storage capacity for each item type.
- `avail[AAA]`, `avail[BBB]`, `avail[CCC]`: The current number of items available for sale, where `avail[item]` $\leq$ `cap[item]`
- The maximum quantity of each item type a customer can order in a single transaction. This value is significantly smaller (at least half) than the capacity for each item type.

Customer threads issue the following request to the store:

```
buy(countAAA, countBBB, countCCC)
```

Where `countAAA, countBBB, countCCC` $\leq$ `maxOrder`.

Supplier threads issue the following requests to the store:

```
maysupply(itemtype, n)
supply(itemtype, n)
```

Where `itemtype` is one of AAA, BBB, or CCC and n is the number of items in the supplier's package. A supplier first calls `maysupply` to check if the store has enough capacity. After preparing the items, the supplier calls `supply(itemtype, n)` to add n items to `avail[itemtype]`.

3 Synchronization Constraints

The `buy()`, `maysupply()`, and `supply()` functions, must enforce the following synchronization constraints:

1. A Customer thread calling `buy()` blocks if: `count[item]` > `avail[item]` for any of the `item` types in the order.
2. If `count[item]` ≤ `avail[item]` for all `items`, the `avail[item]` is reduced by the `count[item]` for each item type in a single transaction.
3. The total number of items sold via `buy()` calls must equal the total number of items removed from `avail`.
4. No item can be sold to multiple customers, and customers cannot buy items not deducted from `avail`.
5. A supplier thread calling `maysupply(itemtype, n)` blocks if: `avail[item]` + `n` > `cap[item]`
6. When `avail[item]` + `n` ≤ `cap[item]`, the capacity for `n` items is reserved for the supplier. Other suppliers of the same item type may block if the remaining capacity is insufficient.
7. A supplier thread that called `maysupply(itemtype, n)` eventually calls `supply(itemtype, n)` adding `n` to `avail[itemtype]`.
8. Every `maysupply(itemtype, n)` call is followed by a matching `supply(itemtype, n)` call with identical arguments. Suppliers always call `maysupply()` before `supply()`.
9. A `supply()` call unblocks one or more customer threads if their orders can now be fulfilled.
10. A `buy()` call that reduces `avail[itemtype]` may unblock a supplier thread waiting in `maysupply()` if sufficient capacity becomes available.
11. A customer thread blocked on `buy()` must not prevent another customer thread from proceeding if the latter's order can be fulfilled. For example if thread 1 orders 1 AAA and 2 BBB, thread 2 orders 1 AAA and 1 AAA, and the store has 1 AAA and 1 BBB available, thread 2 must always receive its order. An implementation that thread 1 reserves the last AAA and blocks waiting for 2 BBB's, causing thread 2 to block, is incorrect.
12. Similarly, a supplier thread must not block another supplier thread if capacity exists. For example, if 3 slots are available for AAA, a `maysupply(AAA, 5)c` call must not prevent a `maysupply(AAA, 3)` call from proceeding.
13. Implementation must preserve the integrity of store variables. It must be deadlock free. And it must not busy wait.
14. The fairness and optimization of orders and supplies is not an objective in the assignment. For example, when orders `buy(4,1,1)`, `buy(2, 1, 0)`, `buy(2, 0, 1)` arrive at the same time with availability `(5, 1, 1)`, solutions accepting the first one or second and third together are both acceptable.

3.1 Threads

The system supports any number of customer and supplier threads. Below is pseudocode for typical thread behavior:

Customer Thread:

```
customer()
{
    while (1) { // in a loop
      aA = random(0, maxOrder)
      aB = random(0, maxOrder)
      aC = random(0, maxOrder)
      if ( aA + aB + aC == 0 ) continue;
      buy(aA, aB, aC);
      sleep(forawhile);
    }
}
```

Supplier Thread:

```
supplier(itemtype, n) {
     while(1) {
          maysupply(itemtype, n);
          sleep(preperation);
          supply(itemtype, n);
     }
}
```

4 Implementation

The threads will be implemented by the test programs similar to description above. You will implement the following functions as a library:

- `void initStore(int capAAA, int capBBB,int capCCC, int maxOrder)` will initialize the store with the given parameters.
- `buy(int countAAA, int countBBB, int countCCC)`, the call by the customer threads.
- `maysupply(int itemtype, int count)`, the call by the supplier threads.
- `supply(int itemtype, int count)`, the call by the supplier threads.
- `monitorStore(int cap[3], int avail[3])` puts the current store variables on parameter arrays.

`initStore()` will be called once by the test programs. The others will be called by arbitrary number of threads. Initially `avail[itemtype]` is set to `cap[itemtype]` for each item, the stocks are full.

4.1 Input Format

Your simulation accepts 3 command line arguments: `Store Spec`, `Number of Customers` and `Supplier Spec`.

Store Spec Capacities of items `AAA`, `BBB` and `CCC` with “:” character as the delimiter, in [1-1000]. Example: “20:30:40”.

Number of Customers In [1-1000];

Supplier Spec Capacities of items, in a cyclic period of `AAA`, `BBB` and `CCC` with ":" character as the delimiter. Number of suppliers is [1-100], capacity is [1-10000]. Examples: "30"; 1 supplier of `AAA`. "20:10:50:10"; 2 suppliers of `AAA` with capacities 20 and 10. One supplier of `BBB` with capacity 10. One supplier of `CCC` with capacity 50.

5 Homework Specifications

You are given a student pack with `hw2.cpp`, `hw2.h`, `monitor.h` and `testrun.cpp`. Use of the `monitor.h` is optional. You can use POSIX semaphores, `pthread` library mutexes, condition variables and `monitor.h` (which is based on pthread condition variables and mutexes), or combination of those. Use of any other library is not allowed. You can use additional files, as long as your `Makefile` compiles your homework.

- Implement your homework in C++.
- Your homework should compile & run on our lab computers ("ineks"). Test your Makefile on an inek before submission.
- Do not use any library flag other than `-lpthread`.
- Prefer asking your questions through the ODTUClass discussion forum. Questions asked through e-mail such as "can you go over my code and tell me my mistake" will be ignored.
- This is an individual assignment. Everything you submit should be your own own. The violators will get no grade from this assignment and will be punished according to the department regulations.

We will use different `testrun.cpp` files to test & grade your submission with different deadlock and integrity scenarios.

6 Submission

Submit your homework to the ODTUClass assignment. Create a flat `.tar.gz` archive named "<ID_Number>.tar.gz" (e.g. 2482057.tar.gz) that includes your implementation and `Makefile`. Your homework will be evaluated using;

```
$ tar -xf 2482057.tar.gz
$ # Copy hw2.h, monitor.h, testrun.cpp to current working directory
$ make all
$ ./testrun <store spec> <number of customers> <supplier spec>
```

If your homework does not compile & run with the steps listed above, a penalty of 25 points will be deducted.