CS205 Operating System Concepts with Android
AY2024/25 Term 2

CS205 Assignment 3: Concurrency

Revised: 20th March 2025

1. Learning Outcomes

a. To get familiar with multi-threaded programming.
b. To get familiar with OS concepts (process concurrency, synchronization, mutual exclusion) through practicing Java concurrency programming.

2. Introduction

Consider a factory with multiple machines making hot dogs and multiple machines packing hot dogs. The hot dogs will be first made by the making machines, then sent to a hot dog pool. The packing machines will take hot dogs from the pool for packing. The manager of the factory specifies the number of hot dogs needed. There is a log file containing records for logging when a hot dog is made and packed. When all the hot dogs are produced, the manager will append a summary to the log file.

3. Requirements

Design and implement a multi-threaded Java program that works as follows:

- Each **making machine** is modeled with a separate thread (created by the main thread)
  - Each making machine is identified by a making machine ID with a prefix "`m`" and a serial number of the machine (counting from `1` to `M`).
  - The making machine makes one hot dog at a time.
  - The making machine takes 4 units of time to make one hot dog.
  - After a hot dog is made, the making machine takes 1 unit of time to send the hot dog together with its making machine ID to the hot dog pool.
  - After sending the hot dog, the making machine adds a record to the text log file in the form of "`<making_machine_id> puts <hot_dog_id>`", where
    - `hot_dog_id` is a serial number of the hot dog (counting from `1` to `N`),
    - `making_machine_id` is the making machine ID.
  - Once the making machine finishes writing the record, it can start to make the next hot dog.
  - When the required number of hot dogs is made, the making machine stops.
- Each **packing machine** is modeled with a separate thread (created by the main thread).
  - Each packing machine is identified by a packing machine ID with a prefix "`p`" and a serial number of the machine (counting from `1` to `P`).
  - The packing machine starts by checking if there is any hot dog in the pool.
  - If there are hot dogs in the pool, the packing machine takes 1 unit of time to take a hot dog from the pool, one at a time. Otherwise, the packing machine waits for a hot dog to be made.
  - After taking the hot dog from the pool, the packing machine takes 2 units of time to pack the hot dog.
  - After packing the hot dog, the packing machine adds a record to the text log file in the form of "`<packing_machine_id> gets <hot_dog_id> from <making_machine_id>`", where
    - `packing_machine_id` is the packing machine ID.
    - `hot_dog_id` is a serial number of the hot dog,
    - `making_machine_id` is the making machine ID.

    - Once the packing machine finishes writing the record, it can start to pack the next hot dog.
    - When all the requested `N` hot dogs are packed, the packing machine stops.
- The **manager** is the main thread that manages the production of hot dogs.
    - The manager captures from the command-line arguments the requested number of hot dogs, `N`, to be prepared.
    - The manager captures from the command-line arguments the number of slots available in the hot dog pool, `S`, to be handled.
    - The manager captures from the command-line arguments the requested number of making machines, `M`, to be created.
    - The manager captures from the command-line arguments the requested number of packing machines, `P`, to be created.
    - The manager initiates `M` making machines and `P` packing machines, each with a machine ID, for the hot dog production; `1 <= M`, `P <= 30`.
    - The manager waits until all hot dogs are made and packed, as well as all the records are logged.
    - After all machines stop, the manager reads the log file and appends a summary to the log file. The summary provides statistics about the number of hot dogs made and packed by each machine.
- The **hot dog pool** represents a circular queue.
    - The queue has `S` slots, where `S < N`.
    - On arrival, a hot dog is inserted to the end of the queue.
    - The pool cannot accept hot dogs from a making machine when there are no empty slots.
    - The hot dogs are packed in the first-come-first-serve manner.
    - The pool cannot deliver hot dogs to a packing machine when all slots are empty.

In the description above, note that you are to simulate some work (e.g. making hot dogs, packing hot dogs). For this, you can use the following function, where `n` represents the number of units of time

```
static void gowork(int n){
    for (int i=0; i<n; i++){
        long m = 300000000;
        while (m>0){
            m--;
        }
    }
}
```

The main thread should take in four parameters `N`, `S`, `M`, `P`, where:

- `N` is the number of hot dogs to make; `order`.
- `S` is the number of slots in the hot dog pool; `capacity`,
- `M` is the number making machines; `making machines`,
- `P` is the number of packing machines; `packing machines`.

Sample run:

```
$ java HotDogManager 10 3 2 2
```

Sample output of `log.txt`:

```
order:10
capacity:3
making machines:2
packing machines:2
m1 puts 1
m2 puts 2
p0 gets 1 from m1
m1 puts 3
m2 puts 4
p2 gets 2 from m2
p2 gets 3 from m1
m1 puts 5
m2 puts 6
p1 gets 4 from m2
p1 gets 5 from m1
p1 gets 6 from m2
m2 puts 7
m1 puts 8
p2 gets 7 from m2
p2 gets 8 from m1
m1 puts 9
m2 puts 10
p1 gets 9 from m1
p2 gets 10 from m2
summary:
m1 made 5
m2 made 5
p1 packed 5
p2 packed 5
```

4. Submission instructions

Submit **HotDogManager.java** and all ***.class** files resulting from its compilation, with no other files or subfolders in one ZIP archive to eLearn. The due date is indicated in eLearn.

5. Marking scheme

Thread-safe multithreaded programming	(10%)
Program correctness: Fulfil all the stated requirements	(80%)
Programming style, in-line documentation	(10%)

Note:

1. ***Non-multithreaded*** *implementation will receive* ***zero*** *marks.*
2. *You may use* ***only*** *the concurrency primitives covered in lab 2 for this assignment.*
3. *This is an individual assignment. While you are encouraged to discuss with classmates, submitted codes must be your own work.* ***No code generated from AI tools is allowed****.*