OS Project - Fribourgeois Bakery

Group 1: Gabriel, Odermatt, Wey & Yang

May 26, 2022

Contents

1 Introduction

The present documentation reports on the solution to the *Fribourgeois Bakery* project elaborated by Group 1.

The discussion of the solution will be structured as follows: Chapter one will present the datastructure that represents our inventory as well as our solution to the task "Learning to make bread". Chapter two will focus on "Too good to go" and chapter four will present our additional feature. Our implementation decisions as well as the scenarios will be discussed in chapter one.

- CLI to invoke scripts.

2 Inventory and Making Bread

2.1 Inventory

- Binary tree where each node represents an ingredient type. Each of those nodes holds a linked list where each node represents one unit of that ingredient type.

- Building and searching of tree: Tree is lexographically sorted – meaning that that the left child of each parent node contains an ingredient type whose name is smaller and the right child contains an ingredient type whose name is bigger than the parents name. "Smaller" and "bigger" in this context refer to the ordering of characters proposed by the ASCII standard. Due to this strict ordering of ingredients, the tree can be traversed and searched

very efficiently[1].

- Traversal of the tree:
  - Iterativ oder rekursiv, richtig? (@Lukas). In getInvNode() ist es iterativ, in restock() ist es rekursiv.
- Useful functions provided:
  - registerIngredient()
  - addIngredient()
  - takeIngredient()
  - restock()
- Data structure size: Binary tree can grow in two dimensions: Regarding ingredient types and regarding ingredient units.
- Simultaneous access:
  - We chose to implement concurrency client-side, i.e. not in the datastructure itself but in the "Making bread" script.
  - We could have implemented mutual exclusion in the datastructure itself either on the entire datastructure or only on the linked list inside the ingredient node. However,

2.2 Making Bread

- Synchronization:
  - Mutex: The inventory is mutexed.
  - Semaphore: Coordination between baker, apprentices and shopper.
- Scheduler Strategy:

[1] Strictly speaking this does not always have to be the case. If the ingredients happen to be added in ASCII order (or in reverse) then the tree would be a normal linked list and searching the tree would perform accordingly.

- When an apprentice wants to access the inventory they announce their interest in a global array by modifying the field at the index that corresponds to their id. Depending on the schedular metric used, they insert differnt values. Access to this array may be concurrent since every index can only be written by one apprentice.

- Implemented four schedulers: (i) Predefined learners (ii) Arrival Order (based on time) (iii) Fast learners (apprentice with the most baked breads) (iv) Fairlearners (apprentice with the least baked breads).

- Explain when each of the strategies performs best and worst.

2.3 Scenarios

- Scenarios:

  - (i) Apprentices add the same item to the inventory: In func registerIngredient() we first check whether the ingredient has already been registered. If so, we restock to initial amount but do not add it again.

  - (ii) Apprentices access the same item in the inventory: Not possible by design, since the datastructure is made mutually exclusive and since we synchronize the apprentices using the baker as the scheduler.

  - (iii) Same as (ii) except in this case we synchronize between apprentice and shopper. But the approach stays the same.

3 Too Good To Go

On a high level, our implementation of "Too Good To Go" works as follows:

- The user inputs the parameters such as the name of the bread types, the number of units of each type, the paging alorithm to be used, the number of ticks between TGTG-decisions, etc.

- The bakery produces an initial amount of breads.

- The bakery sells the bread and bakes new breads as they run out.

- Every X number of ticks, the baker applies the paging algorithm to decide which breads to donate.

- As soon as the selling target for the day is reached, the bakery closes.

Let's have a look at the acutal implementation of some of those steps:

Datastructure: The bread types are represented as a dynamic array. Each bread type occupies an index in the array. Every index contains a linked list in which each node represents one unit of that type of bread. Each node is timestamped when it is added to the list. Furthermore, the linked list contains some additional information such as the timestamp of the oldest bread present in the list, whether this bread type was recently requested by a customer, etc.

Time of TGTG: The point in time when the baker has to decide which breads to donate is user-defined and enforced by a thread called 'tgtg_coordinator'. Every X seconds, the 'tgtg_coordinator' sets the flag 'tgtg_flag' to TRUE. After every time the baker sold a bread, they check the flag. If its set to TRUE we call the paging algorithm to decide which breads to donate. Nota bene: Since the flag 'tgtg_flag' is a shared variable we ensure mutual exclusion using the mutex 'mutTGTGFlag'.

- Datastructure: Dynamic Array of Bread Types. Where each bread type has a linked list.
- Procedure:
  - Bake initial amounts of bread.
  - Sell breads until the selling target for the day is reached.
  - Ever x seconds, tgtg is triggered and whe donate bread according to paging algorithm.
- Analogy to NRU.

4 Additional Feature

- Chairs: Queue, mutexted.
- Synchronization: Semaphores (one for the baker and one for each chair).