Multi-threaded Kitchen Simulation and IPC Using Pipes

Will Martin
Course Section: CS 3502
NETID: wmarti83

March 1, 2025

Contents

1 Introduction

This report documents the implementation of a multi-threaded kitchen simulation and an inter-process communication (IPC) mechanism using pipes. The project consists of two parts:

- **Project A:** A simulation where multiple chef threads interact with shared resources (ingredients and ovens), demonstrating threading, synchronization, and deadlock detection.
- **Project B:** A simple IPC demonstration where one process (the producer) sends data to another (the consumer) via pipes.

The objectives of this project were to implement thread management, apply synchronization mechanisms, handle inter-process communication, and overcome real-world challenges during environment setup and testing.

2 Implementation Details

2.1 Threading and Synchronization

Project A simulates a kitchen environment where chefs (threads) access shared resources. The implementation features:

- **Thread Management:** Each chef is implemented as a separate thread. The program creates multiple threads to simulate concurrent activity.
- **Synchronization Mechanisms:**
  - **Mutexes:** Used to protect access to shared resources such as ingredients and ovens. Two mutexes are maintained (one for ingredients and one for ovens) to ensure proper synchronization.
  - **Fixed Locking Order:** In safe mode, threads acquire locks in a consistent order to prevent deadlock.
- **Deadlock Detection and Recovery:** A dedicated thread monitors the progress of chef threads. If no progress is detected for more than 10 seconds, the system assumes a deadlock has occurred and triggers a recovery process that interrupts the stalled threads and resets the locks.

2.2 Inter-process Communication (IPC)

Project B demonstrates IPC using pipes:

- The **Producer** process writes sample data (e.g., strings) to the standard output.
- The **Consumer** process reads the data from standard input, processes it (in this case, converting text to uppercase), and outputs the result.

2.3 Code Snippets

Below is an excerpt from Project A illustrating the deadlock scenario and resource management:

```
public void DeadlockScenario(int chefNumber)
{
    if (chefNumber % 2 == 0)
    {
        // Chef A: Lock ingredients first, then oven
        ingredientMutex.WaitOne();
        Console.WriteLine($"Chef {chefNumber} locked
           ingredients, waiting for an oven...");
        Thread.Sleep(500);
        ovenMutex.WaitOne();
        Console.WriteLine($"Chef {chefNumber} got both
           locks! Cooking...");
        Thread.Sleep(2000);
        ovenMutex.ReleaseMutex();
        ingredientMutex.ReleaseMutex();
    }
    else
    {
        // Chef B: Lock oven first, then ingredients
        ovenMutex.WaitOne();
        Console.WriteLine($"Chef {chefNumber} locked an
           oven, waiting for ingredients...");
        Thread.Sleep(500);
        ingredientMutex.WaitOne();
        Console.WriteLine($"Chef {chefNumber} got both
           locks! Cooking...");
        Thread.Sleep(2000);
        ingredientMutex.ReleaseMutex();
        ovenMutex.ReleaseMutex();
    }
}
```

Listing 1: Deadlock Scenario in Project A

And an excerpt from Project B for the IPC mechanism:

```
using System;

namespace Producer
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] data = { "Hello", "This is a test",
                "IPC via pipes", "C# is awesome!" };
            foreach (var line in data)
            {
                Console.WriteLine(line);
            }
        }
    }
}
```

Listing 2: Producer Code in Project B

3 Environment Setup and Tool Usage

The development environment was set up on a Ubuntu virtual machine (VM) running on Parallels, hosted on a MacBook M2 Pro (ARM architecture). During the setup:

- I installed the necessary compilers and runtime environments for C# and .NET.
- I used Visual Studio Code as the integrated development environment (IDE).

- Some challenges arose due to ARM compatibility issues with VS Code and .NET, which required extra configuration steps.

4 Challenges and Solutions

The project presented several challenges:

- **Environment Setup:** Setting up Ubuntu on a VM via Parallels on an ARM-based MacBook M2 Pro was particularly challenging. I encountered issues with installing VS Code and .NET on the ARM architecture. I had to seek assistance from ChatGPT [1], as well as online tutorials [2] which provided helpful guidance to resolve these installation hiccups.

- **Deadlock Detection:** Initially, I attempted to analyze thread states to detect deadlocks. However, thread states proved too dynamic—they changed even when threads were simply waiting for a shared resource (in this case, the oven) [3]. To address this, I switched to a timing-based approach. If no progress was made by a thread within 10 seconds, it was considered deadlocked, prompting the system to initiate deadlock recovery.

5 Results and Outcomes

The final implementation successfully demonstrates:

- Effective multi-threading with proper synchronization.

- A working deadlock detection and recovery mechanism.

- A simple yet functional IPC mechanism using pipes.

Performance testing indicated that the timing-based deadlock detection method was more reliable than state analysis under dynamic conditions. Limitations remain in scaling the simulation further, and future work could focus on refining resource management and exploring alternative IPC methods.

6 Reflection and Learning

This project was a valuable learning experience. Key takeaways include:

- A deeper understanding of threading concepts, mutex usage, and the challenges associated with deadlock detection.
- Hands-on experience in setting up a development environment on non-traditional architectures (ARM-based MacBook).
- The importance of flexibility in problem-solving—when one approach (thread state analysis) proved unreliable, I pivoted to a timing-based solution.

References

[1] OpenAI, “ChatGPT: Conversational AI,” 2024, assistance with debugging and installation guidance. [Online]. Available: https://openai.com/chatgpt

[2] U. Community, “Installing ubuntu on arm-based macs,” 2024, available at: https://ubuntu.com/tutorials/install-ubuntu-on-arm.

[3] Microsoft, *Threading in C#*, 2023, available at: https://learn.microsoft.com/en-us/dotnet/standard/threading/.