IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication 23/05/2024

Digital Object Identifier

Operating Systems Project Report

Author: Baran Çakmak Demir (e-mail: 210315002@ogr.cbu.edu.tr).

I. INTRODUCTION

In operating systems, the concept of threads plays a significant role. Threads are code or processes that can run concurrently with each other. They are used to make a program run faster and more efficiently. Threads allow different pieces of code to execute simultaneously, thereby utilizing the processor more effectively.

Threads are capable of performing different tasks simultaneously within the same program, allowing for more efficient use of processor and memory resources. They are especially prevalent in processor-intensive programs and applications that require multiple threads.

The aim of this project is to use the thread structure to break down tasks into different parts and solve them. The programming language used in this project is Java. In the Java programming language, it is possible to create a new thread through the Thread class or the Runnable interface.

A. SCENARIO

In this project scenario, a party is being held at a house. A waiter from a catering company is assigned to serve at the party. There are "borek", "cake", and "drink" to be served to 10 guests. There are a total of 30 boreks, 15 slices of cake, and 30 glasses of drinks. Additionally, there are 3 types of trays, each with a capacity of 5. As guests consume the food and drinks, the waiter fill the trays. Each guest must eat all types of food and drink at least one beverage. A guest can eat a maximum of 4 boreks, drink a maximum of 4 glasses of drinks, but can only eat up to 2 slices of cake. This process continues until all the food and drinks are consumed.

II. METHODOLOGY

First, the libraries required for the program are imported. These libraries are necessary for the use of Thread and other functions. At the beginning of the program, final variables and other variables are defined. These variables hold the values for the number of guests, the number of boreks, the number of cakes, and the number of drinks. Additionally,

```
1  package os_project;
2
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5  import java.util.concurrent.locks.Lock;
6  import java.util.concurrent.locks.ReentrantLock;
7
8  public class PartySimulation {
9      private static final int NUM_GUESTS = 10;
10     private static final int NUM_BOREKS = 30;
11     private static final int NUM_CAKES = 15;
12     private static final int NUM_DRINKS = 30;
13
14     private static int borekTray = 5;
15     private static int cakeTray = 5;
16     private static int drinkTray = 5;
17
18     private static int remainingBoreks = NUM_BOREKS;
19     private static int remainingCakes = NUM_CAKES;
20     private static int remainingDrinks = NUM_DRINKS;
21
22     private static final Lock lock = new ReentrantLock();
```

FIGURE 1. Final variables and all the other variables are shown in figure 1.

the Lock object, which will be used later in the program, is created here. (Figure 1 shows the code part of the final and all the other variables.)

A. MAIN CLASS

In the Main Class, an object of the ExecutorService class is created using the Executors.newFixedThreadPool() method to create multiple threads. Inside the for loop, "Guest" objects are created for the number of guests and executed. The Guest class should implement the Runnable interface, allowing each guest to run in a separate thread. After creating threads for all guests, a new Waiter object is created and executed using the executor. The Waiter class should also implement the Runnable interface, allowing the waiter to run in its own thread. Executer.shutdown() method shuts down the ExecutorService after all the submitted tasks are completed. It prevents new tasks from being submitted but allows existing tasks to complete.

```
public static void main(String[] args) {
    ExecutorService executor = Executors.newFixedThreadPool(NUM_GUESTS + 1);

    for (int i = 1; i <= NUM_GUESTS; i++) {
        executor.execute(new Guest(i));
    }

    executor.execute(new Waiter());

    executor.shutdown();
}
```

FIGURE 2. Screenshot of Main Class is shown in figure 2.

B. GUEST CLASS

This code is part of the run method for a Guest class that implements the Runnable interface, allowing each guest to run in a separate thread. The method simulates a guest eating and drinking at a party until they have reached their consumption limits. lock.lock() method acquires the lock to ensure that the critical section (where the guest checks and consumes food and drinks) is executed by only one thread at a time. First "try" part contains the critical section code where the guest checks and consumes food and drinks. First "finally" part ensures that the lock is always released, even if an exception occurs within the try block. Thread.sleep() method makes the current thread sleep for a random period between 0 and 1000 milliseconds. This simulates the time taken by the guest to consume the food or drink. If the thread is interrupted during sleep, it catches the InterruptedException and interrupts the current thread.

```
lock.lock();
try {
    if (borekCount < 4 && borekTray > 0) {
        borekTray--;
        borekCount++;
        System.out.println("Guest " + id + " eats a borek. Borek count: " + borekCount + ". Borek tray: " + borekTray);
    }
    if (cakeCount < 2 && cakeTray > 0) {
        cakeTray--;
        cakeCount++;
        System.out.println("Guest " + id + " eats a slice of cake. Cake count: " + cakeCount + ". Cake tray: " + cakeTray);
    }
    if (drinkCount < 4 && drinkTray > 0) {
        drinkTray--;
        drinkCount++;
        System.out.println("Guest " + id + " drinks. Drink count: " + drinkCount + ". Drink tray: " + drinkTray);
    }
} finally {
    lock.unlock();
}
try {
    Thread.sleep((int)(Math.random() * 1000));
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
```

FIGURE 3. Critical section of the Guest Class is shown in figure 3.

C. WAITER CLASS

The Waiter class implements the Runnable interface, allowing it to run in its own thread. The Waiter's job is to monitor and refill the trays of boreks, cakes, and drinks as they are consumed by the guests. In the while (remainingBoreks > 0 || remainingCakes > 0 || remainingDrinks > 0) part, while loop continues to run as long as there are remaining boreks, cakes, or drinks to be refilled. The loop exits when all food and drinks are depleted. lock.lock acquires the lock to enter the critical section, ensuring that only one thread can modify the shared resources at a time. First "try" part contains the critical section code where the waiter checks and refills the trays. First "finally" part ensures that the lock is always released, even if an exception occurs within the try block. In "if""conditions all of the condition checking operations are handled. Thread.sleep() method makes the current thread (the waiter) sleep for a random period between 0 and 1000 milliseconds to simulate the time taken by the waiter to refill the trays. If the thread is interrupted during sleep, it catches the InterruptedException and interrupts the current thread.

```
lock.lock();
try {
    if (borekTray <= 1 && remainingBoreks > 0) {
        int refillAmount = Math.min(5 - borekTray, remainingBoreks);
        borekTray += refillAmount;
        remainingBoreks -= refillAmount;
        System.out.println("Waiter refills borek tray. Borek tray: " + borekTray + ". Remaining boreks: " + remainingBoreks);
    }
    if (cakeTray <= 1 && remainingCakes > 0) {
        int refillAmount = Math.min(5 - cakeTray, remainingCakes);
        cakeTray += refillAmount;
        remainingCakes -= refillAmount;
        System.out.println("Waiter refills cake tray. Cake tray: " + cakeTray + ". Remaining cakes: " + remainingCakes);
    }
    if (drinkTray <= 1 && remainingDrinks > 0) {
        int refillAmount = Math.min(5 - drinkTray, remainingDrinks);
        drinkTray += refillAmount;
        remainingDrinks -= refillAmount;
        System.out.println("Waiter refills drink tray. Drink tray: " + drinkTray + ". Remaining drinks: " + remainingDrinks);
    }
} finally {
    lock.unlock();
}
try {
    Thread.sleep((int)(Math.random() * 1000));
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
```

FIGURE 4. Critical section of the Waiter Class is shown in figure 4.

III. CONCLUSION

As a conclusion the provided Java code effectively simulates a party scenario where multiple guests are served by a waiter using multithreading. For this project, 10 pieces of threads are used. By utilizing the ExecutorService with a fixed thread pool, the program manages the concurrent actions of guests and the waiter, ensuring efficient and synchronized execution.

Each Guest thread simulates a guest consuming boreks, cake slices, and drinks, adhering to specified consumption limits. The critical sections, where shared resources are accessed and modified, are protected using a ReentrantLock, ensuring thread safety and preventing race conditions. The Waiter thread continuously monitors and refills the trays whenever their contents fall below a certain threshold, mimicking a real-time refilling process.

The use of random sleep intervals adds realism to the simulation, representing the varying time guests take to eat and drink. The synchronization mechanism ensures that the shared resources are accessed in a controlled manner, preventing any inconsistencies in the counts of boreks, cakes, and drinks.

Overall, this code demonstrates a practical application of multithreading and synchronization in Java, effectively managing concurrent tasks and ensuring consistent and expected outcomes in a simulated party environment.

```
Guest 10 eats a borek. Borek count: 2. Borek tray: 4
Guest 10 drinks. Drink count: 2. Drink tray: 4
Guest 8 eats a borek. Borek count: 3. Borek tray: 3
Guest 8 drinks. Drink count: 3. Drink tray: 3
Guest 3 eats a borek. Borek count: 3. Borek tray: 2
Guest 3 drinks. Drink count: 3. Drink tray: 2
Guest 6 eats a borek. Borek count: 4. Borek tray: 1
Guest 6 drinks. Drink count: 4. Drink tray: 1
Guest 10 eats a borek. Borek count: 3. Borek tray: 0
Guest 10 drinks. Drink count: 3. Drink tray: 0
Guest 7 eats a slice of cake. Cake count: 2. Cake tray: 1
Guest 9 eats a slice of cake. Cake count: 2. Cake tray: 0
Waiter refills borek tray. Borek tray: 5. Remaining boreks: 6
Waiter refills drink tray. Drink tray: 5. Remaining drinks: 6
Guest 4 eats a borek. Borek count: 3. Borek tray: 4
Guest 4 drinks. Drink count: 3. Drink tray: 4
Guest 3 eats a borek. Borek count: 4. Borek tray: 3
Guest 3 drinks. Drink count: 4. Drink tray: 3
Guest 5 eats a borek. Borek count: 3. Borek tray: 2
Guest 5 drinks. Drink count: 3. Drink tray: 2
Guest 8 eats a borek. Borek count: 4. Borek tray: 1
Guest 8 drinks. Drink count: 4. Drink tray: 1
Waiter refills borek tray. Borek tray: 5. Remaining boreks: 2
Waiter refills drink tray. Drink tray: 5. Remaining drinks: 2
Guest 1 eats a borek. Borek count: 4. Borek tray: 4
Guest 1 drinks. Drink count: 4. Drink tray: 4
Guest 7 eats a borek. Borek count: 2. Borek tray: 3
Guest 7 drinks. Drink count: 2. Drink tray: 3
Guest 10 eats a borek. Borek count: 4. Borek tray: 2
Guest 10 drinks. Drink count: 4. Drink tray: 2
Guest 7 eats a borek. Borek count: 3. Borek tray: 1
Guest 7 drinks. Drink count: 3. Drink tray: 1
Waiter refills borek tray. Borek tray: 3. Remaining boreks: 0
Waiter refills drink tray. Drink tray: 3. Remaining drinks: 0
Guest 9 eats a borek. Borek count: 1. Borek tray: 2
Guest 9 drinks. Drink count: 1. Drink tray: 2
Guest 5 eats a borek. Borek count: 4. Borek tray: 1
Guest 5 drinks. Drink count: 4. Drink tray: 1
Guest 2 eats a borek. Borek count: 4. Borek tray: 0
Guest 2 drinks. Drink count: 4. Drink tray: 0
```

FIGURE 5. Program output is shown in figure 5.

BARAN ÇAKMAK DEMIR was born on May 22, 2001, in Sakarya. He started his primary education at Bahçelievler Primary School and continued to Bahçelievler Middle School for his middle school education. After the high school entrance exam, he was admitted to Sakarya Anatolian High School, where he continued his high school education and graduated. Subsequently, he was admitted to the computer engineering department at Atılım University for his undergraduate program. After his first year, he transferred to the computer engineering department at Manisa Celal Bayar University, where he is currently continuing his studies in computer engineering.

He completed his summer internship at Tırsan Treyler Sanayi ve Ticaret A.Ş. There, he worked in the technology management department on tasks such as cloud management, network management, and server management.