IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

Preparation of Papers for IEEE ACCESS

BURAK TALHA MEMIS[1], (Member, Computer Engineering), YAVUZ SELIM ERDOGAN[2], (Member, Computer Engineering)

[1]Manisa Celal Bayar University Computer Engineering, Yunusemre/Manisa (e-mail: cbu.muhendislik.fakultesi@gmail.com)

Corresponding first author:(e-mail: talha45879@gmail.com) Corresponding second author:(e-mail: yavuz.selimm.erdogan@gmail.com).

ABSTRACT This study presents the design and implementation of software that models the food and beverage service processes of a waiter serving 8 guests at a house party. A simulation was conducted using the Java programming language to manage the consumption of "borek," "cake," and "drink" by the guests. Each guest can consume up to 4 borek, 4 drinks, and 2 slices of cake. The simulation was developed using a multithreaded structure to allow multiple guests to access the same food and drink simultaneously. Thread, lock, and try-catch mechanisms were used to prevent deadlock and incorrect consumption. The primary algorithm ensures that all guests consume each type of food and drink at least once, with a maximum of 2 cakes and 4 boreks and drinks. The developed software successfully simulates the consumption process of all food and beverages, showing a total duration of 1 minute and 5 seconds.

INDEX TERMS Multithreading, Java programming language, Deadlock prevention, Lock mechanisms, Thread synchronization, Parallel processing, Critical section management,

I. INTRODUCTION

A. SCENARIO

In this study, a party environment with 8 guests and a waiter is simulated. The guests consume borek, cake, and drink with the following quantities:

- 30 borek
- 15 slices of cake
- 30 glasses of drink

Each guest can consume up to 4 borek, 4 drinks, and 2 slices of cake. Each tray has a capacity of 5 units. The waiter refills the trays when the number of items on them drops to 1 or 0. If the waiter does not have enough items to fully refill the tray, they add as many items as they have. The eating and drinking process ends when all food and beverages are consumed.

B. TECHNOLOGIES USED

This study utilized the Java programming language. To implement the 'multithreading' structure, the Thread class was used, and synchronization was managed using 'Lock' and 'ReentrantLock' classes. These technologies were chosen to enable parallel processing and to prevent deadlock situations. The libraries used are:

```
import java.util.*;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;
```

II. METHODOLOGY OF OUR SYSTEM

In this section, we describe the methodology of our system for simulating the food and beverage service at a party with 8 guests and a waiter.

Our system consists of several classes and methods designed to simulate the behavior of guests and the waiter in a party environment.

1) Tray Class

The `Tray` class represents a tray that holds items, such as food or beverages, at the party. It has attributes for capacity, current item count, and the remaining items on the tray. Upon initialization, the tray is instantiated with a specific item count and name. It provides methods to retrieve the current item count and to decrement the item count when an item is taken from the tray.

```
public class Tray {
    private int capacity;
    private int itemCount;
    private int tepsideKalan = 5;
    public String name;
    static int finishCount = 3;

    public Tray(int itemCount, String name) {
        this.capacity = 5;
        this.itemCount = itemCount;
        this.name = name;
```

```
    }

    public int getItemCount() {
        return itemCount;
    }

    public void decrementItemCount() {
        if (itemCount > 0)
            itemCount--;
    }
```

The isEmpty() method checks if the tray is empty by examining the item count.

```
    public boolean isEmpty() {
        return itemCount == 0;
    }
```

The remainingItems() method returns the number of remaining items on the tray.

```
    public int remainingItems() {
        return tepsideKalan;
    }
```

The fillRemainingItems() method refills the tray with items if its count drops to 1 or 0.

```
    public void fillRemainingItems(){
        if (itemCount >= 5){
            tepsideKalan = capacity;
        }
        else if((itemCount < 5) &&
        (itemCount >= 1)){
            tepsideKalan = itemCount;
        }
        else if(itemCount == 0){
            System.out.println("The " + name +
            "s tray is out of products");
            finishCount--;
        }
        if(finishCount == 0){
            System.out.println("Party is over!!!");
            System.exit(0);
        }
    }
```

The decrementRemainingItems() method decreases the count of remaining items on the tray by 1.

```
    public void decrementRemainingItems(){
        tepsideKalan = tepsideKalan - 1;
    }
}
```

2) Guest Class

The `Guest` class represents each guest at the party. It includes methods for eating and drinking, ensuring that each guest consumes food and beverages according to the specified limits.

The Guest class represents a guest at the party, keeping track of their consumption of borek, cake, and drink. Upon initialization, the guest's counts for borek, cake, and drink are all set to zero.

```
public class Guest {
    private int borekCount;
    private int cakeCount;
    private int drinkCount;

    public Guest() {
        this.borekCount = 0;
        this.cakeCount = 0;
        this.drinkCount = 0;
    }
```

The getBorekCount() method returns the current count of borek consumed by the guest.

```
    public int getBorekCount() {
        return borekCount;
    }
```

The eatBorek() method increments the borek count if it's less than 4, indicating the guest's consumption of borek.

```
    public void eatBorek() {
        if (borekCount < 4)
            borekCount++;
    }
```

The getCakeCount() method returns the current count of cake slices consumed by the guest.

```
    public int getCakeCount() {
        return cakeCount;
    }
```

The eatCake() method increments the cake count if it's less than 2, indicating the guest's consumption of cake.

```
    public void eatCake() {
        if (cakeCount < 2)
            cakeCount++;
    }
```

The getDrinkCount() method returns the current count of drinks consumed by the guest.

```
    public int getDrinkCount() {
        return drinkCount;
    }
```

The drink() method increments the drink count if it's less than 4, indicating the guest's consumption of drinks.

```
    public void drink() {
        if (drinkCount < 4)
            drinkCount++;
    }
```

The isSatisfied() method checks if the guest is satisfied by ensuring that they have consumed 4 borek, 2 cake slices, and 4 drinks.

```
    public boolean isSatisfied() {
        return borekCount == 4 &&
        cakeCount == 2 && drinkCount == 4;
    }
}
```

III. CRITICAL SECTION AND THREAD MANAGEMENT

In this section, we explain the critical section management and thread synchronization mechanisms used in our system to ensure proper and concurrent access to shared resources, such as food trays.

A. THREAD MANAGEMENT

Each guest at the party is represented by a separate thread, which allows them to perform eating and drinking actions independently. The following code snippet shows how guest threads are created and managed:

"This code initializes a Thread array where each thread represents a guest. It iterates through the guest list, creates a new thread for each guest, and assigns it to the array. The thread's behavior is defined using a lambda expression."

```
Thread[] guestThreads =
new Thread[guests.size()];
for (int i = 0; i < guestThreads.length; i++)
{
    Guest guest = guests.get(i);
    guestThreads[i] = new Thread(() -> {
```

B. CRITICAL SECTION MANAGEMENT

"This code checks if the cake tray is not empty and locks it for thread safety. The guest eats a slice of cake, and the tray's item counts are updated. It prints the updated cake count and makes the thread sleep for 1 second to simulate eating."

```
if (!cakeTray.isEmpty()) {
    cakeLock.lock();
    try {
        if (!cakeTray.isEmpty()) {
            guest.eatCake();
            cakeTray.decrementRemainingItems();
            cakeTray.decrementItemCount();
            System.out.println("Guest " + guests.indexOf(guest) + " ate a slice of cake.");
            Thread.sleep(1000);
            System.out.println("Total number of cakes remaining = " + cakeTray.getItemCount() +
                    " / Number of cakes left on tray = " + cakeTray.remainingItems());

            if (cakeTray.remainingItems() <= 1) {
                waiter.fillTray(cakeTray);
                Thread.sleep(3000);
                cakeTray.fillRemainingItems();
[           }                                                                                   ]
```

Critical sections are used to prevent concurrent access to shared resources, such as the food trays, ensuring that only one thread can access the tray at a time. This is achieved using locks.

1) Lock Mechanism

We use the `Lock` and `ReentrantLock` classes to synchronize access to the trays. The `borekLock` is used to protect access to the borek tray, as shown in the code snippet above. Here is how the lock mechanism is implemented:

- Before accessing the borek tray, the thread acquires the `borekLock` using `borekLock.lock()`.
- The thread checks if the tray is not empty and performs the eating action.
- The item count is decremented, and the remaining items on the tray are updated.
- If the tray has 1 or 0 items left, the waiter refills the tray.
- The thread releases the lock using `borekLock.unlock()` in the `finally` block to ensure that the lock is released even if an exception occurs.

C. WAITER FUNCTIONALITY

The waiter thread continuously monitors the trays and refills them when necessary. The following method illustrates how the waiter refills the trays:

```
public void fillTray(Tray tray) {
    tray.fillRemainingItems();
}
```

D. HANDLING INTERRUPTIONS

The code includes exception handling to manage interruptions using the `try-catch` block. If a thread is interrupted while sleeping or waiting, an `InterruptedException` is caught and logged.

E. SUMMARY

This section has detailed the thread management and critical section handling mechanisms used to ensure the safe and concurrent consumption of food and beverages by the guests at the party. By using multithreading and synchronization, we prevent race conditions and ensure that the system operates smoothly.

IV. RESULTS

A. PERFORMANCE AND EXECUTION TIME

In this study, a total of 8 threads were used. The average and total execution times for each thread are as follows:

Total number of threads: 8 Total execution time: 1 minute and 5 seconds Guests' eating and drinking time: Guests perform eating and drinking actions at random intervals. Waiter's refilling time: The waiter refills the trays when the number of items drops to 1 or 0. Process waiting time: Average waiting time, minimum and maximum waiting times.

V. CONCLUSION

This study successfully developed software that simulates the food and beverage consumption processes of guests at a party. A multithreaded structure was implemented using the Java programming language, and synchronization mechanisms were used to prevent deadlock and race conditions. The results show that guests successfully consumed food and beverages within the specified limits, and the entire process took a total of 1 minute and 5 seconds.

REFERENCES

[1] A. S. Tanenbaum and H. Bos, "Multithreading and synchronization," in *Modern Operating Systems*, 3rd ed., Upper Saddle River, NJ, USA: Pearson, 2007, pp. 1–56.
[2] D. B. Johnson and A. C. Weaver, "The Lock Mechanism for Multithreaded Applications," *Journal of Parallel and Distributed Computing*, vol. 62, no. 2, pp. 123–136, Feb. 2002, 10.1016/S0743-7315(02)00005-4.

BURAK TALHA MEMIS I am an individual who has been fascinated by technological devices since my childhood and focus on improving myself in this field, particularly in computers. Spending most of my time coding and experimenting with computers, I possess a personality that embraces innovation, which greatly aids me in my studies and projects within the realm of computer engineering. I have acquired proficiency in Python in 2024, and I also have knowledge in Java, SQL, and MATLAB programming languages. My skill set includes strong problem-solving abilities, effective communication skills, adeptness in teamwork and leadership, crisis management capabilities, and proficiency in public speaking.

YAVUZ SELIM ERDOGAN I am Yavuz Selim Erdoğan, I am currently a 3rd year student of Computer Engineering at Manisa Celal Bayar University and I have a strong foundation in various technologies such as Kotlin, Java, Flutter, Python, Firebase, PostgreSQL, Git and Android Studio. My self-learning journey has been greatly supported by leveraging resources like Stack Overflow and official documentation, allowing me to continually improve my programming skills. I excel in teamwork, leadership and crisis management, developed through a variety of academic and extracurricular activities. Additionally, I have an intermediate level of English proficiency, which allows me to interact with international technical resources and collaborate effectively in different environments.