**Java Programming - Simulation**

**Abstract**

A real-time system means that the system is subjected to real-time. i.e., the response should be guaranteed within a specified timing constraint or the system should meet the specified deadline. The aim of this research is to examine how the concurrent programming techniques can applied to address the real-time system’s constraints. I have made two simulations to determine the implications of concurrent programming on RT performance. The performance should be compared with normal programs. The result of the original codes and the refactored codes of both simulations are compared and determined whether there is an improvement or not. Using JBuilder, memory management of original codes and refactored codes are compared. In JBuilder, the heap memory usage, classes, threads, and CPU usage are measured. All in all, the result of the performance analysis shows that the refactored codes using Runnable and Callable interfaces achieve better result than the original codes. As a result, I proved that the refactored codes have made the system to be more efficiency and reliably.

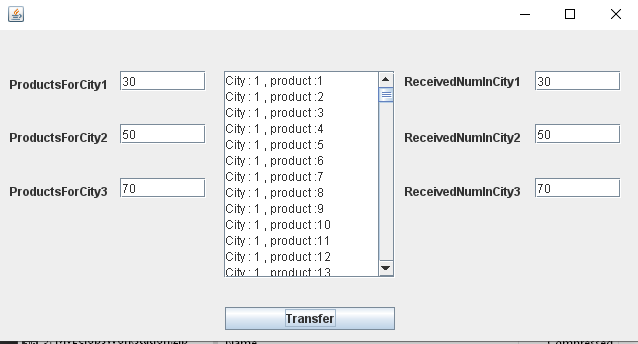
Index Term: Real-Time System, Concurrent Programming Techniques, JBuilder

**Introduction**

A real time system is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure.. This is because the real-time system designers will ensure that the system can execute tasks and provide responses in a specific period, and it is important for many projects or applications, especially those that are related to safety. In other words, a real-time system is a system that has time constraint to response to a given input stimuli. The response time is important for a real-time system, and the designers must ensure that the delay of the response must be as small as possible to obtain acceptable timeliness. There are three types of constraints required to be met by a real-time system. They are timing constraints, precedence constraints, and resource constraints. The time constraint refers to the time restriction that the task must be finished executing before the deadline. The precedence constraint refers to the priorities of tasks, which all tasks are scheduled to run in a specific order. The resource constraint refers to the access to the shared resource exclusively. Besides, the three key considerations for real-time developments, namely speed, reliability, and efficiency. Since a real-time system must meet all these constraints, thread management and interrupts, priority scheduling, and other concurrent programming techniques are required to implement to perform multitasking and generate responses on time. Real-time systems can be split into hard real-time and soft real-time. In a hard real-time system, the deadline is strict and rigid, which means that the task must be completed and give the response on time since there is no leeway to miss the deadline. On the other hand, the deadline for a soft real-time system is also important, but it is more flexible compared to a hard real-time system.

**Methodology**

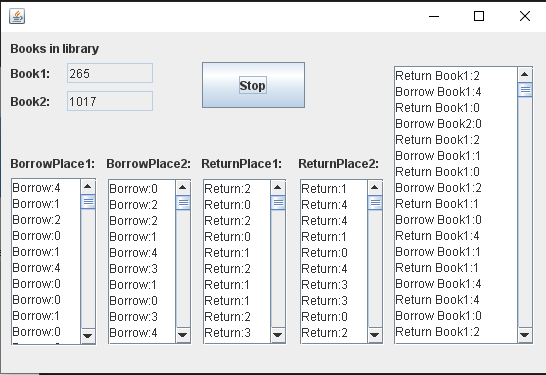
1.Overview of two simulations



In this paper, I made two applications and did its simulation

First , It's the application supplying products from multi-producer to multi-consumer.

The products are only supplied though one channel. They only use one memory place.

Second It is the application that borrow and return books in library.

There are two type of book.

There are also 4 place, borrowing and returning the books.

This also used selected one channel.

2.Implementation Environment

In this paper, both simulations are developed using Java programming language with MyEclipse.

Using JBuilder, powerful GUI was established and it became easy to control and consider real time performance.

The POM file is as follows:

<**project** xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<**modelVersion**>4.0.0</**modelVersion**>

<**groupId**>Concurrent\_Programming</**groupId**>

<**artifactId**>Concurrent\_Programming</**artifactId**>

<**version**>0.0.1-SNAPSHOT</**version**>

<**packaging**>ejb</**packaging**>

<**name**>Concurrent\_Programming</**name**>

<**description**/>

<**properties**>

<**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>

</**properties**>

<**dependencies**>

<**dependency**>

<**groupId**>javax</**groupId**>

<**artifactId**>javaee-api</**artifactId**>

<**version**>8.0</**version**>

<**scope**>provided</**scope**>

</**dependency**>

</**dependencies**>

<**build**>

<**plugins**>

<**plugin**>

<**artifactId**>maven-compiler-plugin</**artifactId**>

<**version**>2.3.2</**version**>

<**configuration**>

<**source**>1.7</**source**>

<**target**>1.7</**target**>

</**configuration**>

</**plugin**>

<**plugin**>

<**artifactId**>maven-ejb-plugin</**artifactId**>

<**version**>3.1.0</**version**>

<**configuration**>

<**ejbVersion**>3.2</**ejbVersion**>

</**configuration**>

</**plugin**>

</**plugins**>

</**build**>

</**project**>

3.Implementation Method

First, in two application, they have GUI part:

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(111, 112, 559, 377);

\_myContentPane = new JPanel();

\_myContentPane.setBorder(new EmptyBorder(6, 6, 6, 6));

\_myContentPane.setLayout(null);

setContentPane(\_myContentPane);

JLabel \_Label = new JLabel("Book1:");

\_Label.setBounds(10, 34, 46, 14);

\_myContentPane.add(\_Label);

\_234textField\_book1 = new JTextField();

\_234textField\_book1.setHorizontalAlignment(SwingConstants.LEFT);

\_234textField\_book1.setText("300");

\_234textField\_book1.setBounds(66, 31, 86, 20);

\_myContentPane.add(\_234textField\_book1);

\_234textField\_book1.setColumns(11);

JLabel lblBook = new JLabel("Book2:");

lblBook.setBounds(10, 62, 46, 14);

\_myContentPane.add(lblBook);

Also all operations used thread mode.

class ReturnThread extends Thread {

Book book1;

Book book2;

public ReturnThread(Book book1, Book book2) {

this.book1 = book1;

this.book2 = book2;

}

4. Performance Analysis

- Book application:

Response time 0.05s

CPU usage 0.011%

Runnable interface

-Product application

Response time 0.03s

CPU usage 0.014%

It is very efficiency way to use CPU and memory.

Runnable interface.

My application addressed these:

* Utilise an asynchronous messaging service
* involve multiple producers and consumers.
* incorporate appropriate concurrent mechanisms.
* Comprise appropriate error handling (exceptions)

**Conclusion**

In summary, there are two simulations that are used for redesigning using Runnable interfaces in this research. Using thread, all operations performed in real time mode. I analyze the drawback and advantages of this mode. Also I strengthen knowledge about thread and GUI programming. It was really big progress for me.

In the future, I’ll try to get more useful knowledge about Java programming and Real time performance.

**References**

Antony, A. (2019, March 19). Understanding Java Microbenchmark Harness or JMH Tool. Retrieved from Medium: https://medium.com/javarevisited/under standing-java-microbenchmark-harness- or-jmh-tool-5b9b90ccbe8d

Brooks, A. (2021, November 1). Java Runnable Interface. Retrieved from javaTpoint: https://www.javatpoint.com/runnable- interface-in-java

GeeksforGeeks. (2018, April 19). CountDownLatch in Java. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/countdo wnlatch-in-java/

Jenkov, J. (2018, October 27). Java Callable . Retrieved from jenkov: http://tutorials.jenkov.com/java-util- concurrent/java-callable.html

Koren, I., & Krishna, C. M. (2021). Checkpointing in Real-Time Systems. Fault-Tolerant Systems (Second Edition), 203-235.

Ojo, T. (2017, December 22). A Deep Dive Into the Java ExecutorService. Retrieved from DZone:

Staff, E. (2005, November 25). Back to the basics: How to measure real-time performance. Retrieved from embedded: https://www.embedded.com/back-to- the-basics-how-to-measure-real-time- performance/

Sunderman, K. (2018, June 7). Runnable interface in Java. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/runnable

Vats, R. (2021, May 2). Runnable Interface in Java: Implementation, Steps & Errors. Retrieved from upGrad: https://www.upgrad.com/blog/runnable- interface-in-java/

Walls, C. (2017). Real-Time Operating Systems. Embedded Software (Second Edition), 243-286.

Williams, L. (2021, November 1). Real-time operating system (RTOS): Components, Types, Examples. Retrieved from Guru99: https://www.guru99.com/real- time-operating-system.html

Zambrano, A. M. (2017). Community Early Warning Systems. Wireless Public Safety Networks 3, 39-66.