Bounded-Buffer Project

CSS226

By

Group 3

Members :

61090500407 CHAKKRIN YONGYUTHWICHAI

61090500414 THANAPON DONLOY

61090500419 BOONYAPAT BOONSAENG

61090500422 PHASIT THATUCHIRANGKUL

61090500425 SUANGRUTHAI TITIPATTANANONE

61090500431 JUNJIRA KAOWICHAKORN

King Mongkut's University of Technology Thonburi (KMUTT)

ความหมายของ Bounded-buffer problem

เมื่อ thread 2 ตัวขึ้นไปสามารถประมวลผลพร้อมกันได้โดยแชร์ทรัพยากรกันใน 1 process การทำงานได้แบบนี้ถ้าเราใช้มันแก้ไขข้อมูลร่วมกันโดยไม่ได้คิดถึงผลในหลายๆ กรณีอาจทำให้เกิดปัญหา inconsistency หรือการไม่สอดคล้องกันของข้อมูลได้ ปัญหาคลาสสิคที่ทำให้เกิด inconsistency คือ Consumer-producer problem หรืออีกชื่อหนึ่งคือ Bounded-buffer problem เป็นปัญหาการใช้ buffer ร่วมกันถ้า buffer ไม่เต็ม producer ก็จะผลิตค่าเติมลงไปใน buffer จนกว่าจะเต็มเมื่อ buffer เต็ม producer ก็จะรอส่วน consumer จะกินข้อมูลจาก buffer ที่ละหนึ่งไปเรื่อย ๆ หาก buffer หมด consumer ก็จะรอ

วัตถุประสงค์โปรแกรม

โปรแกรมนี้สร้างขึ้นมาเพื่อจำลองการทำงานของหลักการ Bounded-Buffer problem ซึ่งเป็นหลักการที่ไว้ใช้แก้ปัญหาการเข้าถึงข้อมูลที่อยู่ในรูปแบบของ Buffer หลายช่องของหลาย Thread ที่ทำงานอยู่พร้อม ๆกัน และต้องการที่จะเข้าถึงข้อมูลนี้ร่วมกัน

การนำ Bounded-Buffer มาประยุกต์ใช้กับโปรแกรม

โปรแกรมนี้เป็นโปรแกรมที่จะให้ Producer ใส่ข้อมูลเข้าไปก่อน โดยที่จะให้ Consumer รอในระหว่างที่ Producer กำลังใส่ข้อมูลชุดแรก ซึ่ง Producer ก็จะใส่ข้อมูลจนกว่า Buffer จะเต็ม และเมื่อมีช่องว่าง Producer ก็จะทำการเติมข้อมูลลงไปทันที โดยไม่ต้องรอให้ Consumer ทำงานจนเสร็จ และ Consumer ก็จะเช็คว่ามีข้อมูลใน Buffer อยู่หรือไม่หากมีข้อมูลก็จะดึงข้อมูลออกมาใช้ทันที แต่ถ้าหากไม่มีข้อมูลอยู่ภายใน Buffer ก็จะรอจนกว่าจะมีข้อมูลเข้ามาเติม วิธีการนี้จะทำให้ความเร็วในการส่งข้อมูลและความถูกต้องของข้อมูลนั้นมีมากขึ้น

การทำงานของโปรแกรม

โปรแกรมนี้เป็นโปรแกรมแปลง “ตัวเลข” ให้เป็น “ตัวอักษร” โดยจะมีผู้ใส่ข้อมูลตัวเลขเข้าไปยัง Buffer คือ Producer และ ผู้ที่จะดึงข้อมูลจาก Buffer มาแปลงเป็นตัวอักษร คือ Consumer

Producer จะป้อนข้อมูลตัวเลขไปเรื่อย ๆ จนกว่า Buffer จะเต็ม ถ้าหาก Buffer เต็ม Producer ก็จะทำการรอจนกว่าจะมีการนำข้อมูลไปใช้ทำให้ Buffer นั้นว่าง

Consumer มีหน้าที่ดึงข้อมูลมาจาก Buffer ตั้งแต่ช่องแรก และไปเรื่อย ๆเมื่อถึงจุดสิ้นสุดของ Buffer, Consumer ก็จะย้อนกลับไปดูที่ Buffer ช่องแรกว่ามีข้อมูลอยู่หรือไม่ ถ้ามีก็ดึงข้อมูลไปเรื่อย ๆ แต่ถ้าหากไม่มีก็ทำการรอจนกว่าจะมีข้อมูลเข้ามา

Code

ใน Code ของโปรแกรมนี้จะเป็น Code ภาษา Java และเป็นการเขียนโปรแกรมในรูปแบบของ การเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) โดยจะมีการทำงานของ Class ทั้งหมด 4 Class ได้แก่

1. Class BoundedBuffer ซึ่งมี Method main() อยู่ข้างใน มีไว้เพื่อสร้าง Object และรัน Thread
2. Class Buffer มีไว้เพื่อสร้าง Buffer กลางที่ไว้ให้ 2 Thread ได้ใช้งานข้อมูลร่วมกัน
3. Class Producer มีไว้เพื่อสร้างผู้ป้อนข้อมูลลงไปยัง Buffer
4. Class Consumer มีไว้เพื่อสร้างผู้ดึงข้อมูลมาจาก Buffer

สามารถ **ดาวน์โหลด** Project ได้ที่ : https://github.com/paedayz/OS_Project_BoundedBuffer

คำแนะนำ : เพื่อความสะดวก แนะนำให้ใช้ Visual Studio Code ในการเปิดและรัน Project

Class BoundedBuffer:

```
1   class BoundedBuffer {
2
3       public static void main(String[] args) throws InterruptedException {
4
5           System.out.println("program starting\n");
6           System.out.println("Thread\t\tStatus\t\tPut\t\tGot\n");
7           Buffer buffer = new Buffer(4);
8           Producer prod = new Producer(buffer);
9           Consumer cons = new Consumer(buffer);
10
11          prod.start();
12          cons.start();
13      }
14  }
```

บรรทัดที่ 3 : ประกาศ Method main()

บรรทัดที่ 7 : สร้าง Object ชื่อ buffer จาก Class Buffer โดยกำหนด Argument เป็น 4 Argument ในที่นี้จะเป็นตัวกำหนด Buffer max size

บรรทัดที่ 8 : สร้าง Object ชื่อ prod จาก Class Producer โดยกำหนด Argument เป็นตัวแปร buffer เพื่อที่จะให้ Producer ใช้ Buffer ที่ถูกสร้างมาในบรรทัดที่ 7

บรรทัดที่ 9 : สร้าง Object ชื่อ cons จาก Class Consumer โดยกำหนด Argument เป็นตัวแปร buffer เพื่อที่จะให้ Consumer ใช้ Buffer ที่ถูกสร้างมาในบรรทัดที่ 7

บรรทัดที่ 11 : สั่งการเริ่มการทำงาน Thread ที่มีชื่อว่า prod

บรรทัดที่ 12 : สั่งการเริ่มการทำงาน Thread ที่มีชื่อว่า const

Class Buffer :

```java
public class Buffer {
    private final int MaxBuffSize;
    private String[] store;
    private int BufferStart, BufferEnd, BufferSize;

    public Buffer(int size) {
        MaxBuffSize = size;
        BufferEnd = -1;
        BufferStart = 0;
        BufferSize = 0;
        store = new String[MaxBuffSize];
    }

    public synchronized void put(String num) {

        while (BufferSize == MaxBuffSize) {
            try {
                wait();
            } catch (InterruptedException e) {
                Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, e);
            }
        }
        BufferEnd = (BufferEnd + 1) % MaxBuffSize;
        store[BufferEnd] = num;
        BufferSize++;
        notifyAll();
    }

    public synchronized String got() {

        while (BufferSize == 0) {
            try {
                wait();
            } catch (InterruptedException e) {
                Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, e);
            }
        }
        String word = store[BufferStart];
        BufferStart = (BufferStart + 1) % MaxBuffSize;
        BufferSize--;
        notifyAll();
        return word;
    }
}
```

บรรทัดที่ 6 : สร้าง Constructor ของ Class Buffer ขึ้นมา

บรรทัดที่ 14 : สร้าง synchronized method ชื่อ put เพื่อใช้สำหรับการนำเข้าข้อมูลมาใส่ไว้ใน Buffer

บรรทัดที่ 16 : หาก BufferSize หรือตัวแปรที่ใช้บอกขนาดของ Buffer ใช้ไปปัจจุบัน มีค่าเท่ากับ MaxBuffSize หรือตัวแปรที่ใช้บอกขนาดความจุทั้งหมดของ Buffer ก็จะทำให้ Thread ทำการรอ จนกว่า Buffer จะว่างด้วยคำสั่ง wait() ในบรรทัดที่ 18

บรรทัดที่ 29 : สร้าง synchronized method ชื่อ got เพื่อใช้สำหรับการนำข้อมูลจาก Buffer ออกไปใช้

บรรทัดที่ 31 : หาก BufferSize มีค่าเท่ากับ 0 หรือแปลว่า ไม่มีข้อมูลอยู่ใน Buffer ก็จะทำให้ Thread ทำการรอ จนกว่า Buffer จะมีข้อมูลเข้ามาใส่ ด้วยคำสั่ง wait()

Tips 1 : คำสั่ง try-catch เป็นบล็อกคำสั่งที่ใช้ครอบกลุ่มคำสั่งสำหรับดักจับ ***ข้อผิดพลาด Error ของ Runtime Error*** *โดยเมื่อเกิด Error หรือข้อผิดพลาดขึ้น โปรแกรมในภาษา* ***Java*** *จะกระโดดออกจากบล็อก* ***try*** *แล้วจัดการความผิดพลาดตามคำสั่งที่ระบุไว้ในบล็อก* ***catch*** *โดยเราสามารถตรวจสอบชนิดของข้อผิดพลาด หรือจะ อ่านรายละเอียดข้อผิดพลาดที่เกิดขึ้นในขณะนั้นก็ได้เช่นเดียวกัน*

Class Producer :

```
1   class Producer extends Thread {
2       private final Buffer buffer;
3
4       public Producer(Buffer b) {
5           buffer = b;
6       }
7
8       public void run() {
9           String num = "123456789";
10          String[] word = num.split("");
11
12          while (true) {
13
14              for (int i = 0; i < word.length; i++) {
15                  buffer.put(word[i]);
16                  System.out.println("Producer\trunning\t\t" + word[i] + "\t\t--");
17              }
18
19              break;
20
21          }
22
23          System.out.println("Producer\tEND\t\t--\t\t--");
24      }
25  }
```

บรรทัดที่ 4 : กำหนด Constructor สำหรับ Class Producer

บรรทัดที่ 8 : Method run() มีไว้เพื่อ run คำสั่งทุกคำสั่งภายใน Method นี้เมื่อมีการสร้าง Thread ด้วยคำสั่ง start()

บรรทัดที่ 9 : ตัวแปรตัวอย่างข้อมูล ในความเป็นจริงหากต้องการที่จะเปลี่ยนแปลง แก้ไข หรือเพิ่มเติมข้อมูลตัวเลขในตัวแปรนี้ก็สามารถทำได้

บรรทัดที่ 10 : คำสั่ง split(" ") มีไว้เพื่อแยกตัวอักษรออกทีละ 1 bit และนำมาเก็บไว้ใน Array ชื่อ word

บรรทัดที่ 14 : รันลูป for ตามจำนวนช่องของ Array word โดยใช้คำสั่ง word.length เป็นตัวกำหนด

บรรทัดที่ 15 : นำค่าของตัวเลขในตัวแปร word ในลำดับที่ i เข้าไปยัง Method put() ที่อยู่ใน Buffer กลาง

บรรทัดที่ 16 : แสดงค่าที่ทำการใส่เข้าไปใน Buffer ออกมาหลังจากทำการใส่เข้าไปเป็นที่เรียบร้อยแล้ว

บรรทัดที่ 19 : เมื่อทำลูปจบก็ใช้คำสั่ง Break เพื่อหยุดการทำงานของ Thread

Class Consumer :

```
class Consumer extends Thread {

    private final Buffer buffer;

    public Consumer(Buffer b) {
        buffer = b;
    }

    public void run() {

        try {
            sleep(3000);
        } catch (InterruptedException ex) {
            Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, ex
        }

        String word;

        while (true) {

            String work = buffer.got();
            switch (work) {
                case "0":
                    word = "Zero";

                    break;
                case "1":
                    word = "One";

                    break;
                case "2":
                    word = "Two";

                    break;
                case "3":
                    word = "Three";

                    break;
                case "4":
                    word = "Four";

                    break;
                case "5":
                    word = "Five";

                    break;
                case "6":
                    word = "Six";

                    break;
                case "7":
                    word = "Seven";

                    break;
                case "8":
                    word = "Eigth";

                    break;
                case "9":
                    word = "Nine";

                    break;

                default:
                    word = "";
                    break;
            }
            System.out.println("Consumer\trunning\t\t--\t\t" + word);
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

บรรทัดที่ 5 : กำหนด Constructor สำหรับ Class Consumer

บรรทัดที่ 9 : Method run() มีไว้เพื่อ run คำสั่งทุกคำสั่งภายใน Method นี้เมื่อมีการสร้าง Thread ด้วยคำสั่ง start()

บรรทัดที่ 12 : เมื่อเริ่มการทำงานของ Thread ให้ Thread นี้รอไปก่อน 3 วินาทีด้วยคำสั่ง sleep(3000) โดยตัวเลข 3000 นั้นมีหน่วยคือ Millisecond

บรรทัดที่ 21 : สร้างตัวแปร String work มาไว้เก็บค่าที่ได้จากการไปดึงข้อมูลมาจาก Buffer ด้วยคำสั่ง buffer.got()

บรรทัดที่ 22 : ใช้คำสั่ง switch-case เพื่อเช็คว่าค่าที่นำมาจาก Buffer นั้นเป็นค่าใด

บรรทัดที่ 68 : ทำการแสดงผลลัพธ์ที่ได้หลังการนำข้อมูลมาจาก Buffer และทำการแปลงเป็นที่เรียบร้อย

บรรทัดที่ 70 : สั่งให้ Consumer รอ 1 วิก่อนที่จะไปดึงข้อมูลครั้งถัดไป

Output :

```
program starting

Thread          Status          Put             Got

Producer        running         1               --
Producer        running         2               --
Producer        running         3               --
Producer        running         4               --
Producer        running         5               --
Consumer        running         --              One
Consumer        running         --              Two
Producer        running         6               --
Consumer        running         --              Three
Producer        running         7               --
Consumer        running         --              Four
Producer        running         8               --
Consumer        running         --              Five
Producer        running         9               --
Producer        END             --              --
Consumer        running         --              Six
Consumer        running         --              Seven
Consumer        running         --              Eigth
Consumer        running         --              Nine
```

จะเห็นได้ว่า Producer ทำการ put ตัวเลขลงไปก่อน เพราะ Consumer ยังรอด้วยคำสั่ง sleep(3000) อยู่ ทำให้ Producer ทำงานไปก่อน จนกว่า Buffer จะเต็ม เมื่อเต็มแล้วจึงรอให้ Consumer มาดึงข้อมูลออกจาก Buffer ก่อนค่อย put ข้อมูลเข้าไปใหม่ และเมื่อ Producer put ข้อมูลครบจนถึงเลข 9 แล้ว ก็แสดง Status ว่า END และไม่มีการทำงานใด ๆอีก แต่ Consumer ยังคงดึงข้อมูลจาก Buffer ออกมาจนกว่าจะหมด และที่ Consumer ไม่มี END ก็เพราะโปรแกรมนี้ไม่ได้กำหนดไว้ว่า Consumer จะหยุดทำงานเมื่อใด Consumer จะรอการ input ของข้อมูลใส่ Buffer ต่อไปเรื่อย ๆ