CS2040 Lab 5 (Week 7)

Additional Material: Synchronisation

Nathanael Seen

NUS

March 6, 2021

Introduction

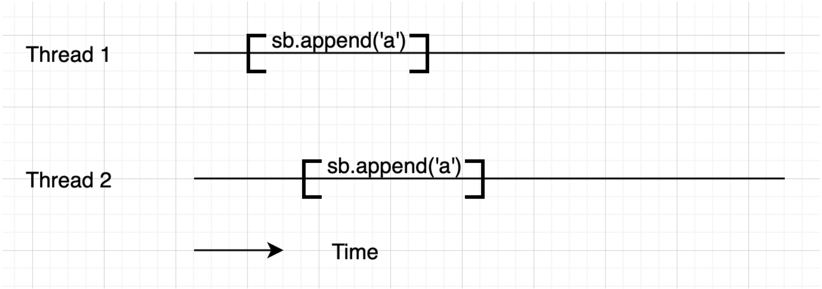
- Previously, we talked about how StringBuffer/HashTable are synchronised data structures
- As a result they could potentially be slower than its unsynchronised counterparts (namely; StringBuilder/HashMap)
- But what is synchronisation actually, and why do we need it?

Problem

- Consider a multithreaded program, where we have two threads; Thread1 and Thread2, and a shared StringBuilder; sb
- Both threads are executed *concurrently*
- Each thread tries to append one character 'a' to sb for 500 times
- Intuitively, the length of sb should be 1000 as there are two threads, each appending 'a' for 500 times
- However, due to concurrent execution and the fact the StringBuilder is not synchronised, we might get a length lesser than 1000!

Explanation

- Firstly, it is crucial to note that since StringBuilder's append() method is unsynchronised,
  it is possible for two threads to call append() in an interleaving fashion:



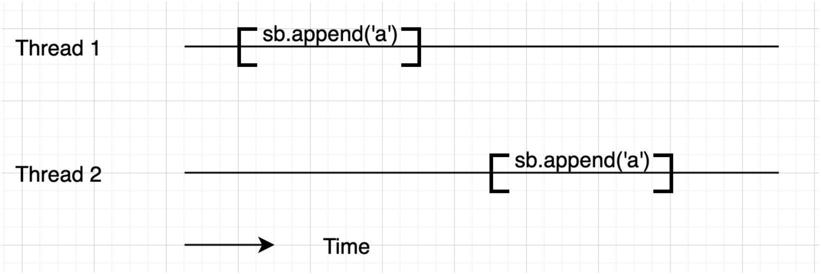
- Internally (in the append() methods), it could be the case where Thread1 modifies sb in some seperate local memory and then append 'a', but before it's able to write the updated String back to main memory, Thread2 accesses the main memory and gets the unmodified String
- Hence, Thread1's modification to sb, might not be visible to Thread2

Explanation (cont.)

- Moreover, from a black-box API perspective, such execution sequence is undefined (either both threads append, none append, or one of them append, we don't know for sure)

Explanation (cont.)

- In direct contrast, let's take a look at a *mutually-exclusive* invocation of append(), where only one thread can call append at any one time (if another thread wants to call append, it has to wait for the current thread to finish):



- This kind of execution sequence is 'good' and desired (more formally its called *sequential consistency*, more in CS4231 if you're interested!)
- To achieve this, we need to use StringBuffer (and not StringBuilder), as the synchronised methods help guarantee mutual-exclusion

Program Code

- Here's the code I used, try it yourself to see the difference!
- Copy the code to a file and name it Main.java, then run it!

```
public class Main {
    // Shared StringBuilder across threads
    volatile StringBuilder sb = new StringBuilder();

    // Defining our thread class
    class WorkerThread extends Thread {

        @Override
        public void run() {
            // Each thread prints 'a' 500 times
            for (int i = 0; i < 500; ++i) {
                sb.append("a");
            }
        }
    }
    ...
```

Program Code (cont.)

```
...
void test() {
    WorkerThread thread1 = new WorkerThread();
    WorkerThread thread2 = new WorkerThread();

    thread1.start();
    thread2.start();

    try {
        thread1.join();
        thread2.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println("sb.length() = " + sb.length());
}

public static void main(String[] args) throws Exception {
    Main main = new Main();
    main.test();
}
```

Program Output

- Here's the program output after running the original code (presented in previous slide) which uses StringBuilder (unsynchronised)
- As can be seen sometimes it prints the desired output of '1000' but other times it just prints a length of sb lesser than '1000'

```
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 999
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 980
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 989
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 998
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 999
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 1000
```

Program Output

- Here's the program output after replacing StringBuilder in the original code to StringBuffer (synchronised)
- As can be seen it always prints the desired output of '1000'

```
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 1000
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 1000
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 1000
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 1000
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 1000
nathanael@Nathanael-Seen HashTable % java Main
sb.length() = 1000
```

Thank you

Thank you!!