OS4 - Callables, Futures, Intro to Synchronization, Mutex

Tuesday, 11 July 2023 1:50 AM

Merge Sort - Multithreaded

Divide → Conquer

Divide → sort both parts independently

https://github.com/KnightKnight27/scaler-os-batch

Runnables are used to only run a task thread.

Callables are used to run and return data from a thread.

Futures in Java

```
main () {
    print (" Something ");
    int i = Thread(55);
    print(" Bye")
}
```

```
Something
Bye
Thread Output
```

This is because the Thread is different from the thread of the Main() function.

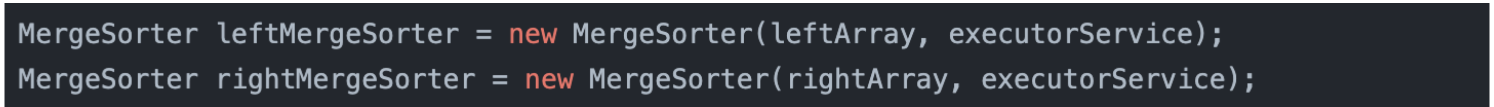
```
main () {
    print ("something");
    int i =  Thread(55);
    print(" Bye")
    print (i)
}
```

This will cause an error bcos the main thread executes it before Thread completes

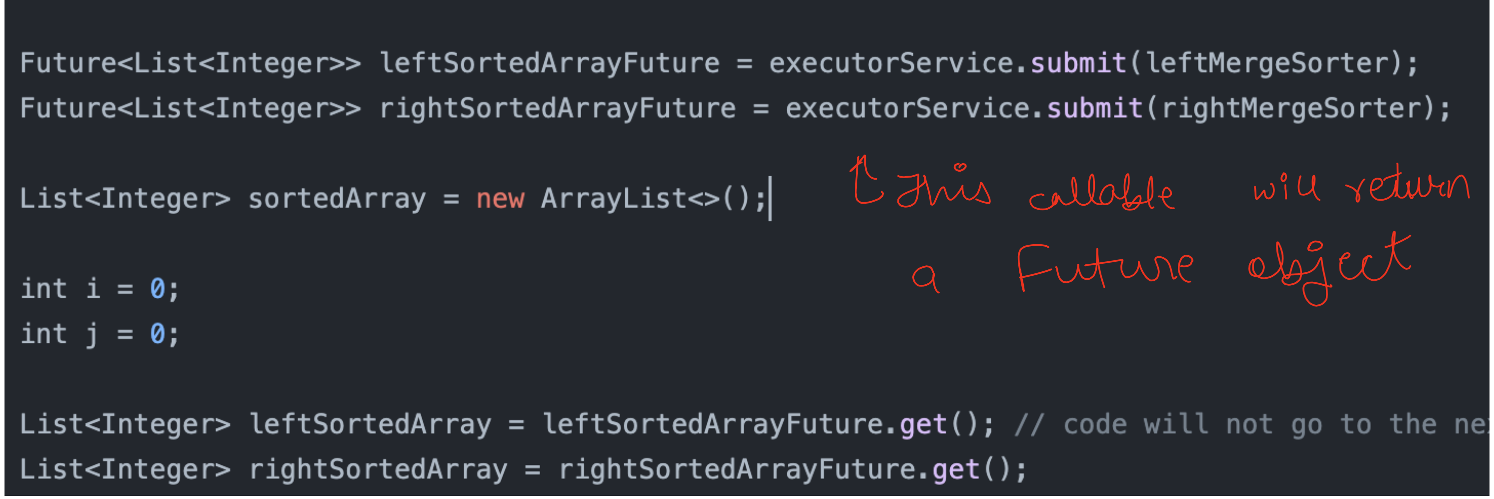
```
main () {
    print ("something");
    FutureObject i = Thread(55);
    print(" Bye");
    print (FutureObject.get(i))
}
```

This will stop the further execution of the whole program until i is calculated.

Callables $\xrightarrow{\text{return}}$ Future



```
MergeSorter leftMergeSorter = new MergeSorter(leftArray, executorService);
MergeSorter rightMergeSorter = new MergeSorter(rightArray, executorService);
```



```java
Future<List<Integer>> leftSortedArrayFuture = executorService.submit(leftMergeSorter);
Future<List<Integer>> rightSortedArrayFuture = executorService.submit(rightMergeSorter);

List<Integer> sortedArray = new ArrayList<>();

int i = 0;
int j = 0;

List<Integer> leftSortedArray = leftSortedArrayFuture.get(); // code will not go to the ne
List<Integer> rightSortedArray = rightSortedArrayFuture.get();
```

↳ This callable will return a Future object

https://github.com/KnightKnight27/scaler-os-batch/blob/main/MergeSorted2.java

↳ This will stop further execution of the program until the sorted array returns

Adder & Subtractor

Adder

```
for (int i=0; i<100; i++){
        count += i
}
```

Subtractor

```
for (int i=0; i<100; i++){
        count -= i
}
```

The count variable is commonly shared b/w the two functions

Count actually is an object of count variable.

```
public class Client {
    public static void main(...){
        SharedCount sharedCount = new SharedCount();

        Adder adder = new Adder(sharedCount);
        Subtractor sibtractor = new
        Subtractor(sharedCount);

        Thread t1 = new Thread(adder);
        Thread t2 = new Thread(subtractor);

        t1.start();
        t2.start();

        t1.join();
        t2.join();
    }
}
```

both started

wait for finish

join() function also stops further execution

```
public class SharedCount {
    this.count = 0;
}
```

But since both start one after another, they are working asynchronously and hence it is possible that race conditions occur and one function's preemption happens more than the other and hence it produces gibberish output.

Asynchronous

Parallel execution

```
public class Client {
    public static void main(...){
        SharedCount sharedCount = new SharedCount();

        Adder adder = new Adder(sharedCount);
```

Synchronous

```
        Subtractor sibtractor = new
        Subtractor(sharedCount);

        Thread t1 = new Thread(adder);
        Thread t2 = new Thread(subtractor);

        t1.start();
        t1.join();

        t2.start();
        t2.join();
    }
}
```

first start 4 finish t1 completely

then move on to t2

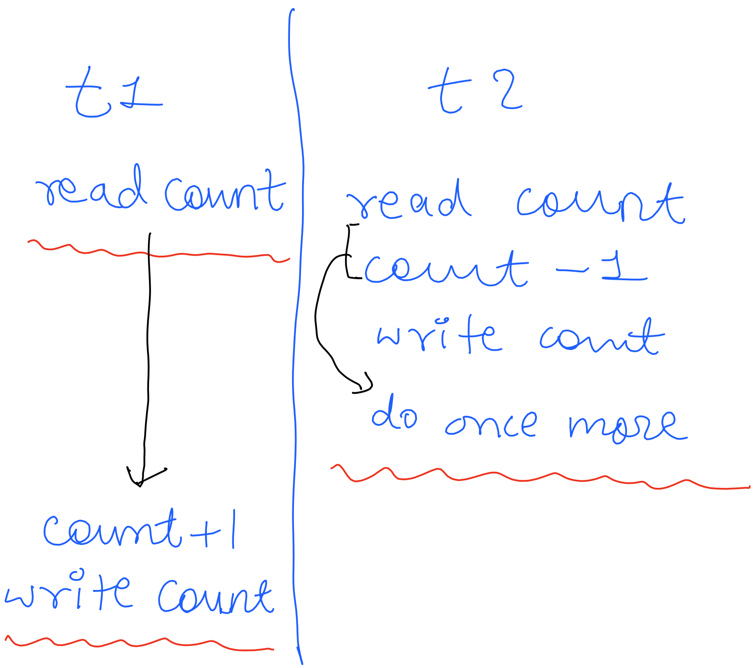
No parallelization

The problem with the gibberish output due to reace conditions is because of how the increment operation works.

count += 1 is actually comprised of 3 different ops:

1. Read count
2. calculate count+1
3. Overwrite count with count+1

So during execution, preemption may occur before any of these ops actually occur.



Assignment:

1. Implement Multithreaded Quick Sort
2. Read about Generics in Java
3. Read about Locks and Semaphores