

Sequential Consistency

7.1 Consider a multithreaded program written in a language with a syntax similar to C++ that provides (full) sequential consistency (not SC-DRF) and an assertion mechanism (i.e., `assert`) similar to the one in C++. The program has two threads, and these threads share the variables `x`, `y`, and `z` of type `int`, all of which are initially zero. Below, several different scenarios are given for the code of the two threads. In each case, the code contains a number of assertions. For each scenario, indicate whether each assertion will be true: always, sometimes, or never. Justify your answer in each case.

(a) First scenario.

- Thread 1 Code.

```
[A1] x = 1;  
[A2] y = 1;
```

- Thread 2 Code.

```
[B1] if (x == 1)  
[B2]     assert(y == 1);
```

(b) Second scenario.

- Thread 1 Code

```
[A1] x = 1;  
[A2] y = 1;
```

- Thread 2 Code

```
[B1] while (!y) {}  
[B2] assert(x == 1);
```

(c) Third scenario.

- Thread 1 Code

```
[A1] x = 1;  
[A2] y = 1;  
[A3] z = 1;
```

- Thread 2 Code

```
[B1] while (!y) {}  
[B2] assert(x == 1);  
[B3] assert(z == 1);
```

(d) Fourth scenario.

- Thread 1 Code.

```
[A1] x = 1;  
[A2] y = 1;
```

- Thread 2 Code.

```
[B1] if (y == 1)  
[B2]     assert(x == 1);
```

First scenario: Sometimes true. One possible of sequentially-consistent executions would be A1-B1-B2-A2. In this case, the assertion will be false.

Second scenario: Always true.

Third scenario: Sometime true. In the case of A1-A2-B1-B2-B3-A3 the assertion will be false.

Fourth scenario: Sometimes true. If the execution in the order that B1 executes before A2, the assertion will not be executed.

7.3 Consider the execution of the two-threaded program listed below. The program has four integer variables `a`, `b`, `x`, and `y`, all of which are initially zero. Enumerate all possible sequentially-consistent executions of this program. For each case, state the value of `a` and `b` upon completion of the program. Upon program completion, is there any combination of the values 0 and 1 that cannot be obtained for `a` and `b`?

- Thread A Code:

```
[A1] x = 1;  
[A2] a = y;
```

- Thread B Code:

```
[B1] y = 1;  
[B2] b = x;
```

Case 1: A1-A2-B1-B2, `a` = 0, `b` = 1.

Case 2: A1-B1-A2-B2, $a = 1, b = 1$.

Case 3: A1-B1-B2-A2, $a = 1, b = 1$.

Case 4: B1-B2-A1-A2, $a = 1, b = 0$.

Case 5: B1-A1-B2-A2, $a = 1, b = 1$.

Case 6: B1-A1-A2-B2, $a = 1, b = 1$.

No sequentially-consistent execution can result in both `a` and `b` being 0.

(a)

1) Assigns 0 to `x`, then assigns 0 to `b`. Or assigns 0 to `b`, then assigns 0 to `x` if reordering applied.

2) Not contains data race.

(b)

1) The program creates two threads, one is initializing the variables `x` and `y`, the other thread prints out the value of `y` and `x`.

- Possible output: "2 1" or "0 0".

2) Not contains data race.

(g)

1) `t1` assigns 42 to `x` and sets `done` to true, `t2` has a while loop and assertion for `x == 42`.

- Possible outcome: `x == 42` and assertion true.

2) Not contains data race.

(i)

1) `t1` assigns 1 to `y` if `x` greater than 0, `t2` assigns 1 to `x` if `y` greater than 0.

- Possible outcome: Both threads do nothing.

2) Not contains data race.

(l)

1) Both thread `t1` and thread `t2` increase the `counter` 100000 times respectively, then prints out `counter`.

- Possible outcome: the value of `counter` is not 200000.

2) Data race may happen since both threads try to increase the `counter`. We can use mutex to avoid this problem.

(m)

1) The program creates a Widget object `w`, and create two threads `t1` and `t2`. Thread `t1` creates a `scoped_lock` object `lock` and sets `w.x` to 1, `t2` creates a `scoped_lock` object `lock` and sets `w.y` to 1.

- Possible outcome: ``w.x == 1`` and ``w.y == 1``.

2) Not contains data race.

```
Data type: float
Height: 512
Width: 512
Max iterations: 255
number of threads: 1, time: 2324 ms
number of threads: 2, time: 1244 ms
number of threads: 4, time: 714 ms
number of threads: 8, time: 527 ms

Data type: double
Height: 512
Width: 512
Max iterations: 255
number of threads: 1, time: 3344 ms
number of threads: 2, time: 1820 ms
number of threads: 4, time: 1029 ms
number of threads: 8, time: 766 ms

Data type: long double
Height: 512
Width: 512
Max iterations: 255
number of threads: 1, time: 3441 ms
number of threads: 2, time: 1888 ms
number of threads: 4, time: 1105 ms
number of threads: 8, time: 779 ms
```

Base on the result, the preformance seems reasonable since more workers doing the same amount of job, the time they need will be shorter.