Concurrency Without Locks

GURUTEAM
specialist ICT learning

Concurrency without Locks

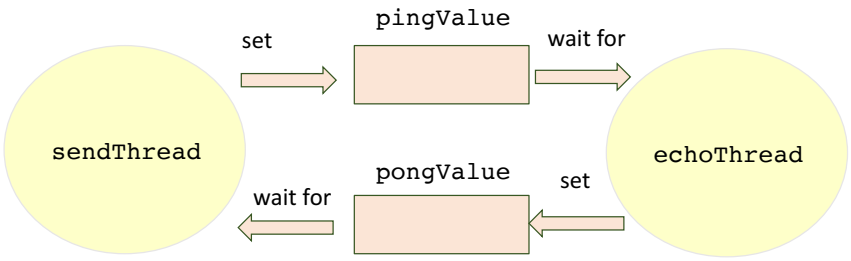
- Synchronisation is required in all concurrent applications
- Conventional model based around 3rd party enforcement
  - lock
  - condition variable

- Synchronisation possible without locks
  - threads manage synchronisation themselves
  - relies on specific hardware behaviour

Concurrency without Locks

- Synchronisation carries an overhead
  - block/unblock thread
  - requires OS intervention
- No option on single core systems
  - multicore systems now make alternatives possible
  - spin rather than block
- Algorithms changing
  - require special low level support
  - require understanding of hardware, especially memory architectures
- "Safe memory operations"
  - volatile

Example: Ping Pong



Example: Ping Pong

```
public class PingPong {
 …
 private static final long REPETITIONS = 1L * 1000L * 1000L;
 public static void main(final String[] args) throws Exception {
  final Thread echoThread = new Thread(new EchoRunner());
  final Thread sendThread = new Thread(new SendRunner());
  echoThread.start();
  sendThread.start();
  final long start = System.nanoTime();
  echoThread.join();
  final long duration = System.nanoTime() – start;
  out.printf("duration %,d (ns)\n", duration);
  out.printf("%,d ns/op\n", duration / (REPETITIONS * 2L));
  out.printf("%,d ops/s\n",
              (REPETITIONS * 2L * 1000L * 1000L * 1000L) / duration);
  out.println("sendValue = "+sendValue+", echoValue = "+echoValue);
 }
}
```

- Driver program
  - start threads, measure time (1,000,000 iterations)

Example: Ping Pong

- Conventional synchronisation
  - condition variable and associated lock

```
public class CVPingPong {
 …
 private static final Lock sendLock = new ReentrantLock();
 private static final Condition sendCondition =
                                        sendLock.newCondition();
 private static final Lock echoLock = new ReentrantLock();
 private static final Condition echoCondition =
                                        echoLock.newCondition();
 private static long sendValue = -1;
 private static long echoValue = -1;
 …

}
```

Example: Ping Pong

```
public static class SendRunner implements Runnable {
 public void run() {
  for (long i = 0; i < REPETITIONS; i++) {
   sendLock.lock();
   try {
    sendValue = i;
    sendCondition.signal();
   } finally { sendLock.unlock(); }
   echoLock.lock();
   try {
    while (echoValue != i) {
     echoCondition.await();
    }
   } catch (final InterruptedException ex) {
    break;
   } finally { echoLock.unlock(); }
  }
 }
}
```

- Worker
  - `EchoRunner` class is similar

Example: Ping Pong

```
public class PingPong {
 …
 private static volatile long sendValue = -1;
 private static volatile long echoValue = -1;
 …
 public static class SendRunner implements Runnable {
  public void run() {
   for (long i = 0; i < REPETITIONS; i++) {
    sendValue = i;
    while (echoValue != i) {
     // busy spin
    }
   }
  }
 }
 // Similar for EchoRunner
}
```

- Lock free
  - volatile variable and spin threads

Example: Ping Pong

- Comparison of performance

```
$ time java CVPingPong
duration 14,255,025,000 (ns)
7,127 ns/op
140,301 ops/s
sendValue = 999999, echoValue = 999999

real    0m14.410s
user    0m5.383s
sys     0m11.443s
```

```
$ time java PingPong
duration 100,420,000 (ns)
50 ns/op
19,916,351 ops/s
sendValue = 999999, echoValue = 999999

real    0m0.227s
user    0m0.336s
sys     0m0.035s
```


Example: Ping Pong

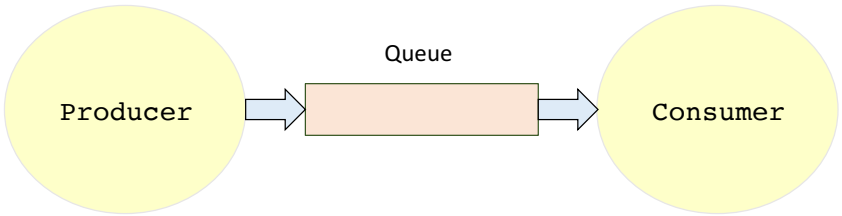
- Comparison of performance

```
$ time java CVPingPong
duration 14,255,025,000 (ns)
7,127 ns/op
140,301 ops/s
sendValue = 999999, echoValue = 999999

real    0m14.410s
user    0m5.383s
sys     0m11.443s
```

© J&G Services Ltd, 2017

Example – Single Writer/Reader Queue



Example – Single Writer/Reader Queue

- A driver program for the test

```
public class QueuePerfTest {
 public static final Integer TEST_ELEMENT = Integer.valueOf(777);
 public static final int REPS = 10 * 1000 * 1000;
 public static final int QUEUE_SIZE = 64 * 1024;
 public static void main(final String[] args) throws Exception {
  final Queue<Integer> queue = new
     java.util.concurrent.ArrayBlockingQueue<Integer>(QUEUE_SIZE);
  for (int i  = 0; i < 5; i++) {
   System.gc();
   Thread.sleep(1000);
   runTest(i, queue);
  }
 }
 …
}
```

Example – Single Writer/Reader Queue

```
private static void runTest( final int runNumber,
                final Queue<Integer> queue) throws Exception {
 final CyclicBarrier barrier = new CyclicBarrier(2);
 final Runnable runner = new Producer(barrier, queue);
 final Thread t = new Thread(runner);
 t.start();
 barrier.await();
 final long start = System.nanoTime();
 int i = REPETITIONS + 1;
 while (0 != --i) {
  while (null == queue.poll()) {
   Thread.yield();
  }
 }
 final long finish = System.nanoTime();
 final long duration = finish - start;
 final long ops = (REPS * 1000L * 1000L * 1000L) / duration;
 System.out.format("%d - ops/sec = %,d\n",
            Integer.valueOf(runNumber), Long.valueOf(ops) );
}
```

- The test

Example – Single Writer/Reader Queue

```
private static class Producer implements Runnable {
 private final CyclicBarrier barrier;
 private final Queue<Integer> queue;
 public Producer( final CyclicBarrier barrier,
                  final Queue<Integer> queue ) {
  this.barrier = barrier; this.queue = queue;
 }
 public void run() {
  try {
   barrier.await();
  } catch (final Exception ex) { ex.printStackTrace(); }
  try {
   int i = REPETITIONS + 1;
   while (0 != --i) {
    while (!queue.offer(TEST_ELEMENT)) { Thread.yield(); }
   }
  } catch (final Exception ex) { ex.printStackTrace(); }
 }
}
```

- The test - Producer

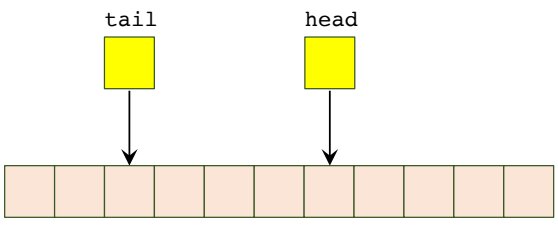
Example – Single Writer/Reader Queue

- Results from `ArrayBlockingQueue<T>`

```
$ java QueuePerfTest
0 - ops/sec = 4,133,628
1 - ops/sec = 4,103,580
2 - ops/sec = 4,767,521
3 - ops/sec = 4,367,569
4 - ops/sec = 4,324,988
```


Alternative `Queue<T>` Implementation

- Lamport Queue
- Use array, and `volatile long` values for `head`/`tail`
  - elements added at `tail` index
  - elements removed from `head` index
  - use mod (`%`) to ensure values can operate within array length
  - `tail - head == 0` => empty
  - `tail - head >= arraylength` => full



Alternative `Queue<T>` Implementation

- Results from `NonBlockingQueue<T>`

```
public class QueuePerfTest {
 … // As before
 public static void main(final String[] args) throws Exception {
  final Queue<Integer> queue = new
                          NonBlockingQueue<Integer>(QUEUE_SIZE);
 … // As before
}
```

```
$ java QueuePerfTest
0 - ops/sec = 14,257,260
1 - ops/sec = 13,256,744
2 - ops/sec = 15,750,710
3 - ops/sec = 14,330,219
4 - ops/sec = 13,932,020
```