Solutions
EDAF55/EDA040 Concurrent and Real-Time Programming

2019–08–22, 14.00–19.00

1. A race condition is said to occur when the result of the execution of a piece of code, and often even its correctness, is dependent on the order in which threads is actually executed.

2. a) Semaphore initialization

```
Semaphore mutex = new Semaphore(1);
```

Critical section

```
mutex.take();
...
mutex.give();
```

b) Semaphore initialization

```
Semaphore sem = new Semaphore(0);
```

Thread receiving signal

```
sem.take();
```

Thread sending signal

```
sem.give();
```

3. First, we need to determine the order of priority between the threads. DMS stipulates that we should assign priorities according to deadline (column D in the table). Then we get the order (from highest to lowest priorty): C, D, B, and A. We can now determine the response times either by drawing a scheduling diagram (since there are no blocking between the threads) or by using the given formula. Here we use the formula.

$R_C^0 = 1$

$R_D^0 = 3$
$R_D^1 = 3 + \lceil 3/6 \rceil \cdot 1 = 4$ $R_D^2 = 3 + \lceil 4/6 \rceil \cdot 1 = 4$
$R_B^0 = 2$
$R_B^1 = 2 + \lceil 2/6 \rceil \cdot 1 + \lceil 2/12 \rceil \cdot 3 = 6$ $R_B^2 = 2 + \lceil 6/6 \rceil \cdot 1 + \lceil 6/12 \rceil \cdot 3 = 6$
$R_A^0 = 4$
$R_A^1 = 4 + \lceil 4/6 \rceil \cdot 1 + \lceil 4/12 \rceil \cdot 3 + \lceil 4/10 \rceil \cdot 2 = 10$
$R_A^2 = 4 + \lceil 10/6 \rceil \cdot 1 + \lceil 10/12 \rceil \cdot 3 + \lceil 10/10 \rceil \cdot 2 = 11$
$R_A^3 = 4 + \lceil 11/6 \rceil \cdot 1 + \lceil 11/12 \rceil \cdot 3 + \lceil 11/10 \rceil \cdot 2 = 13$
$R_A^4 = 4 + \lceil 13/6 \rceil \cdot 1 + \lceil 13/12 \rceil \cdot 3 + \lceil 13/10 \rceil \cdot 2 = 17$
$R_A^5 = 4 + \lceil 17/6 \rceil \cdot 1 + \lceil 17/12 \rceil \cdot 3 + \lceil 17/10 \rceil \cdot 2 = 17$

Answer: $R_A = 17$, $R_B = 6$, $R_C = 1$, and $R_D = 4$

4. 1. mutual exclusion
   2. hold and wait
   3. no pre-emption
   4. circular wait

5. $B_A = 0.3 + 0.4 = 0.7ms$
   $B_B = 0.6 + max(max(0.4, 0.2), 0.4 + 0.3) = 1.3ms$
   $B_C = max(0.2, 0.3) + 0.6 = 0.9ms$
   $B_D = max(0.3, 0.2) = 0.3ms$
   $B_E = 0$

6.
```
public class DistanceSensing {

  private static final double DANGER_DIST = 0.8;

  private final Sample[] latest = new Sample[3];

  public synchronized void put(Sample s) {
    latest[s.getId()] = s;
    notifyAll();
  }

  // private helper: find least timestamp
  // returns Long.MIN_VALUE if at least one is missing
  private long leastTimestamp() {
    Sample min = latest[0];
    for (Sample s : latest) {
      if (s == null) {
        return Long.MIN_VALUE;  // means timestamp is "infinitely" old
      } else if (min.getTimestamp() < s.getTimestamp()) {
        min = s;
      }
    }
    return min.getTimestamp();
  }

  public synchronized Sample min(long time) throws InterruptedException {
    while (leastTimestamp() < time) {
      wait();
    }

    Sample min = latest[0];
    for (Sample s : latest) {
      if (s.getDistance() < min.getDistance()) {
        min = s;
      }
    }
    return min;
  }

  public synchronized Sample awaitDanger() throws InterruptedException {
    // wait for safe state
    while (min(0).getDistance() < DANGER_DIST) {
      wait();
    }

      // wait for unsafe state
    Sample min = min(0);
    while (min.getDistance() >= DANGER_DIST) {
      wait();
      min = min(0);
    }
    return min;
  }
}
```

7.
```
public class MessageBuffer {
  private static final long TIMEOUT = 1000;
  private static final int MAX_MESSAGES = 10;

  private final Deque<String> q = new LinkedList<>();

  private long sendtime = Long.MAX_VALUE;

  private Thread t = new Sender(this);

  /** Activates this message buffer. Must be called before the buffer is used. */
  public void launch() {
    t.start();
  }

  /** Submit a message for sending. */
  public synchronized void put(String s) {
    q.addLast(s);
    long now = System.currentTimeMillis();
    sendtime = q.size() >= MAX_MESSAGES ? now : now + TIMEOUT;
    notifyAll();
  }

  /** Retrieve up to 10 messages, ready to be sent. This method blocks
      until either 10 messages are available, or put() has not been
      called for the last 100ms. */
  public synchronized String[] fetch() throws InterruptedException {
    while (q.isEmpty() || System.currentTimeMillis() < sendtime) {
      wait(sendtime - System.currentTimeMillis());
    }

    int n = Math.min(q.size(), MAX_MESSAGES);
    String[] result = new String[n];
    for (int i = 0; i < n; i++) {
      result[i] = q.removeFirst();
    }

    sendtime = q.size() > 0 ? sendtime + TIMEOUT : Long.MAX_VALUE;

    return result;
  }
}

/** Thread for sending messages as they become available. */
public class Sender extends Thread {
  private final MessageBuffer b;

  public Sender(MessageBuffer b) {
    this.b = b;
  }

  public void run() {
    try {
      while (true) {
        String[] packetData = b.fetch();
        LowLevelNetwork.sendPacket(packetData);
      }
    } catch (InterruptedException unexpected) {
      throw new Error(unexpected);
    }
  }
}
```

```
public class BufferedNetwork {
  private static MessageBuffer buffer = new MessageBuffer();
  static {
    buffer.launch();
  }

  public static void sendMessage(String m) {
    buffer.put(m);
  }
}
```