Design Notes
Assignment 3

Majid Ghaderi

Disclaimer

These notes are based on my own implementation. I do not claim that my implementation is the simplest or the best. Feel free to use or disregard any of these suggestions as you wish.

General Tips

- Start with basic UDP socket IO. Implement a simple client/server program to send and receive Segments using UDP.
- Start testing with small files that have just one segment (*e.g.*, file size is 1 byte).
- Add a lot of debug messages to your code. You can remove/disable them later when not needed. Do not reinvent the wheel here. Look at the Java debugging facilities specifically the `Logger` class.

Program Structure

The high-level structure of my `send()` method is as follows:

```
1: complete TCP handshake
2: start ACK receiving thread
3: while not end of file do
4:     create a segment with next sequence number
5:     wait while transmission queue is full (i.e., window is full)
6:     send the segment and do whatever is necessary for reliable transfer
7: end while
8: wait while there are un-Acked packets in transmission queue
9: cancel timer, ACK receiving thread, clean up
```

Rather than simply waiting in the while loops for the transmission queue to become 'empty' or

'not full', which wastes CPU time, a better strategy is to yield the execution so that other threads can run by calling `Thread.yield()`.

I use a separate thread to receive ACKs and a Timer task to schedule time-outs. I have also defined three helper methods to handle sending segments, receiving ACKs and retransmissions due to time-out:

```
public synchronized void processSend(Segment seg) {
  // send seg to the UDP socket
  // add seg to the transmission queue
  // if this is the first segment in transmission queue, start the timer
}

public synchronized void processACK(Segment ack) {
  // if ACK not in the current window, do nothing
  // otherwise:
  // cancel the timer
  // remove all segements that are acked by this ACK from the transmission queue
  // if there are any pending segments in transmission queue, start the timer
}

public synchronized void processTimeout() {
  // get the list of all pending segments from the transmission queue
  // go through the list and send all segments to the UDP socket
  // if there are any pending segments in transmission queue, start the timer
}
```

The methods `processACK()` and `processTimeout()` are called by the ACK receiving thread and the timer task, respectively, as explained below. As you can see, these three methods are synchronized to avoid running them simultaneously, which could results in unintended concurrency problems as they all modify the content of the transmission queue.

Setting up a Timer

Class `Timer` can be used to schedule a recurring timer. To use the `Timer` class, you need to define a timer task class as well. A timer task is similar to a `Thread` class with a `run()` method. The only difference is that it extends class `TimerTask`. Here is an example of a timer task class:

```
class TimeoutHandler extends TimerTask {

  // define constructor

  // the run method
  public void run() {
    // call processTimeout() in the main class
  }
}
```

The following piece of code demonstrates how to start a timer that goes off in 1000 milli-seconds,

```
  Timer timer = new Timer(true);
  timer.schedule(new TimeoutHandler(), 1000);
```

A timer can be canceled by calling its `cancel()` method. Alternatively, you can use a `ScheduledExecutorService` object to schedule a recurring task. This approach would provide a uniform solution for creating timer tasks based on the familiar `Runnable` interface.

Receving ACKs

Since arrival of ACK segments happens in parallel with sending segments, a separate thread can be started to receive ACKs from the same UDP socket that is used for sending segments.

```
public class ReceiverThread extends Thread {

  // define constructor

  // rhe run method
  public void run() {
    // while not terminated:
    // 1. receive a DatagramPacket pkt from UDP socket
    // 2. call processAck() in the main class to process pkt
  }
}
```