Protocol Implementation Lab Sessions.
Part 2. Stop and Wait Flow Control.
Laboratori d'Aplicacions i Serveis Telemàtics

Josep Cotrina, Marcel Fernandez, Jordi Forga, Juan Luis Gorricho, Francesc Oller

1 Flow control

In this second series of exercises we take a first step towards reliable transmission and discuss a simple Stop and Wait protocol. We will transmit data over a channel that does neither corrupt nor loose packets. This allows us to solely focus on flow control. A new data segment is not to be transmitted until an acknowledgment of the previously transmitted segment has been received.

Note again that you only need to extend the `TCBlock` class and make the appropriate changes in the Properties file.

2 Stop and Wait protocol

Complete the following class that implements a simple Stop and Wait protocol. Invoking the `sentData()` method will block the application sending thread until all data transmitted in the previous invocation has been aknowledged.

```
class TCBlock_StopWait1 extends TCBlock {
    protected int state;
    protected CircularQueue<TCBlock_StopWait1> acceptQueue;

    // States of FSM (see description in TProtocol_StopWait1):
    protected final static int CLOSED = 0,
                               LISTEN = 1,
                               SYN_SENT = 2,
                               ESTABLISHED = 3,
                               FIN_WAIT = 4,
                               CLOSE_WAIT = 5;

    // Sender variables:
    protected int sndMSS;         // Send maximum segment size
    protected boolean sndIsUna; // (true when bytes in (2))
        /* 1: Transmitted and acknowledged bytes
           2: Transmitted but not yet acknowledged bytes
                1       2
            ------------+-------+---------
                        :       first byte not yet transmitted
                        first transmitted byte not yet acknowledged
         */
    // Receiver variables:
    protected ByteQueue rcvQueue;
```

```
    @Override
    public void sendData(byte[] data, int data_off, int data_len) throws IOException {
        lk.lock();
        try {
            log.debug("%1$s->sendData(length=%2$d)", this, data_len);
```

```
            int n = 0;
            while (data_len > n) {
                if (state != ESTABLISHED && state != CLOSE_WAIT) {
                    // transmission is closed
                    throw new IOException("connection does not exist");
                }
                while (sndIsUna) {  // wait sender is not expecting an acknowledg
                    try { appCV.await();
                    } catch (InterruptedException ie) {
                        log.warn("Interrupted exception in sendData()");
                    }
                }
                int seglen = data_len - n;
                if (seglen > sndMSS) seglen = sndMSS;
                // seglen == min(data_len - n, sndMSS)
                byte[] segdata = new byte[seglen];
                System.arraycopy(data, data_off + n, segdata, 0, seglen);
                TCPSegment segment = new TCPSegment();
                    segment.setSourcePort(localPort);
                    segment.setDestinationPort(remotePort);
                    segment.setFlags(TCPSegment.PSH);
                    segment.setData(segdata, 0, seglen);
                sendSegment(segment);
                sndIsUna = true;
                n += seglen;
                logDebugState();
            }
        } finally {
            lk.unlock();
        }
    }

    @Override
    protected void processReceivedSegment(int sourceAddr, TCPSegment rseg) {
        lk.lock();
        try{
            switch (state) {
            case LISTEN: {
                if (rseg.isAck()) {
                    // Any acknowledgment is bad if it arrives on a connection still in
                    // the LISTEN state. Ignore it
                    return;
                }
                if (rseg.isSyn()) {
                    if (acceptQueue.full()) {
                        log.warn(
                          "%1$s->processReceivedSegment: Backlog queue is full. SYN IS LOST !!!",
                          this);
                        return;
                    }
                    // create TCB for new connection
                    TCBlock_StopWait1 ntcb = (TCBlock_StopWait1) newTCB();
                    ntcb.initActive(sourceAddr, rseg.getSourcePort(), ESTABLISHED);
                    // To be completed by the student:
                    ...
                    // prepare the created connection for accept
                    ...
                    // send SYN segment for new connection
                    ...
                    ntcb.logDebugState();
                }
                break;
            }
            case SYN_SENT: {
```

```
            if (rseg.isSyn()) {
                // To be completed by the student:
                ...
                // Change state and wake up connect() thread
                ...
                logDebugState();
            }
            break;
        }
        case ESTABLISHED:
        case FIN_WAIT:
        case CLOSE_WAIT: {
            // Check SYN bit
            if (rseg.isSyn()) {
                // A SYN is bad if it arrives on a connection in an active state. Ignore it
                return;
            }
            // Check ACK
            if (rseg.isAck()) {
                // To be completed by the student:
                ...
                // change sndIsUna accordingly, etc.
                ...
                logDebugState();
            }
            // Process segment text
            if (rseg.getDataLength() > 0) {
                if (state == ESTABLISHED || state == FIN_WAIT) {
                    // To be completed by the student:
                    ...
                    logDebugState();
                } else {
                    // This should not occur, since a FIN has been received from the
                    // remote side.  Ignore the segment text.
                }
            }
            // Check FIN bit
            if (rseg.isFin()) {
                if (state == ESTABLISHED) {
                    state = CLOSE_WAIT;
                } else if (state == FIN_WAIT) {
                    state = CLOSED;
                    removeActiveTCB(this);
                }
                appCV.signalAll(); // wake up receiveData() thread
                logDebugState();
            }
            break;
        }
        default:
            // Segment is ignored
        }
    } finally {
        lk.unlock();
    }
}

@Override
public int receiveData(byte[] buf, int off, int len) throws IOException {
    lk.lock();
    try {
        log.debug("%1$s->receiveData()", this);
        if (state == ESTABLISHED || state == FIN_WAIT) {
        } else if (state == CLOSE_WAIT && rcvQueue.empty()) {
```

```
                throw new IOException("connection closing");
            } else {
                throw new IOException("connection does not exist");
            }
            assert state == ESTABLISHED || state == FIN_WAIT
                    || (state == CLOSE_WAIT && !rcvQueue.empty());
            // wait until receive buffer is not empty
            while (rcvQueue.empty() && !(state == CLOSE_WAIT || state == CLOSED)) {
                try { appCV.await(); } catch (InterruptedException e) {}
            }
            assert !rcvQueue.empty() || state == CLOSE_WAIT || state == CLOSED;
            if (rcvQueue.empty()) {
                // remote endpoint is closed (state is CLOSE_WAIT or CLOSED)
                return -1;
            } else {
                // To be completed by the student:
                ...
                // get data from receive queue and decide when to send an ACK
                ...
            }
        } finally {
            lk.unlock();
        }
    }
```

3 Implementation details

In the `TCBlock_StopWait1` class identify the number of different waiting conditions and reimplement the class using a different *Condition Variable* for each waiting condition.

4 Stop and Wait with an "unsent" Queue

Note that in the protocol of the previous section a slow receiver will keep the application's sender thread blocked most of the time. It is important that you check this. To do so, make the appropriate changes in the configuration file. In order to overcome this undesired situation, one can use a queue of unsent bytes in the sending side of the protocol. Bytes to be sent are queued. At the reception of an ACK segment dequeue as many bytes as possible, pack them in a data segment and send it!.

```
class TCBlock_StopWait2 extends TCBlock_StopWait1 {
    // Sender variables:
    protected static final int SND_MAXUNSENT = 10000; // Max number of bytes in unsent queue
    protected ByteQueue sndUnsentQueue; // (3)
        /* 1: Transmitted and acknowledged bytes
           2: Transmitted but not yet acknowledged bytes
           3: bytes in sndUnsentQueue (not yet transmitted)
                 1       2         3
              ----------+-------+----------+----------
                        :       :          last buffered byte
                        :       first byte not yet transmitted
                        first transmitted byte not yet acknowledged
         */

    TCBlock_StopWait2(TProtocol proto, int port) {
        super(proto, port);
    }

    // initialize for new connection
    @Override
    protected void initActive(int remAddr, int remPort, int st) {
```

```
        remoteAddr = remAddr;
        remotePort = remPort;
        state = st;
        // sndMSS = IP message size (IP packet size - IP header size) - 20 (TCP header size)
        sndMSS = ipLayer.getMMS(remoteAddr) - 20;
        sndUnsentQueue = new ByteQueue(SND_MAXUNSENT);
        rcvQueue = new ByteQueue(sndMSS);
        sndIsUna = false;
        addActiveTCB(this);
    }

    @Override
    public void close() throws IOException {
        // To be completed by the student:
        ...
        // Hint: Take into account if there are bytes left in sndUnsentQueue
        ...
    }

    @Override
    public void sendData(byte[] data, int data_off, int data_len) throws IOException {
        lk.lock();
        try {
            log.debug("%1$s->sendData(length=%2$d)", this, data_len);
            int n = 0;
            while (data_len > n) {
                if (state != ESTABLISHED && state != CLOSE_WAIT) {
                    // transmission is closed
                    throw new IOException("connection does not exist");
                }
                // To be completed by the student:
                ...
                // wait send buffer space (sndUnsentQueue) is available
                ...
                // p == min(data_len - n, sndUnsentQueue.free())
                ...
                // put p bytes of data in sndUnsentQueue. Add p to n
                ...
                // Transmit segment if sender is not expecting an acknowledg
                if (! sndIsUna) {
                    // To be completed by the student:
                    ...
                    // get data from sndUnsentQueue, create segment and send it
                    ...
                }
                logDebugState();
            }
        } finally {
            lk.unlock();
        }
    }

    @Override
    protected void processReceivedSegment(int sourceAddr, TCPSegment rseg) {
        lk.lock();
        try{
            switch (state) {
            case LISTEN: {
                if (rseg.isAck()) {
                    // Any acknowledgment is bad if it arrives on a connection still in
                    // the LISTEN state. Ignore it
                    return;
                }
                if (rseg.isSyn()) {
```

```
            if (acceptQueue.full()) {
                log.warn(
                  "%1$s->processReceivedSegment: Backlog queue is full. SYN IS LOST !!!",
                  this);
                return;
            }
            // create TCB for new connection
            TCBlock_StopWait2 ntcb = (TCBlock_StopWait2) newTCB();
            ntcb.initActive(sourceAddr, rseg.getSourcePort(), ESTABLISHED);
            // To be completed by the student:
            ...
            // prepare the created connection for accept
            ...
            // send SYN segment for new connection
            ...
            ntcb.logDebugState();
        }
        break;
    }
    case SYN_SENT: {
        if (rseg.isSyn()) {
            // To be completed by the student:
            ...
            // Change state and wake up connect() thread
            ...
            logDebugState();
        }
        break;
    }
    case ESTABLISHED:
    case FIN_WAIT:
    case CLOSE_WAIT: {
        // Check SYN bit
        if (rseg.isSyn()) {
            // A SYN is bad if it arrives on a connection in an active state. Ignore it
            return;
        }
        // Check ACK bit
        if (rseg.isAck()) {
            // To be completed by the student:
            ...
            // send unsent data if any, change sndIsUna accordingly, etc.
            ...
            logDebugState();
        }
        // Process segment text
        if (rseg.getDataLength() > 0) {
            if (state == ESTABLISHED || state == FIN_WAIT) {
                // To be completed by the student:
                ...
                logDebugState();
            } else {
                // This should not occur, since a FIN has been received from the
                // remote side.  Ignore the segment text.
            }
        }
        // Check FIN bit
        if (rseg.isFin()) {
            if (state == ESTABLISHED) {
                state = CLOSE_WAIT;
            } else if (state == FIN_WAIT) {
                state = CLOSED;
                removeActiveTCB(this);
            }
```

```
                    appCV.signalAll(); // wake up receiveData() thread
                    logDebugState();
                }
                break;
            }
            default:
                // Segment is ignored
            }
        } finally {
            lk.unlock();
        }
    }
```

5 Testing.

Test the classes above. Check that some data might be lost. Change both the size of transport segments and the `rcvBuffer` size and comment on the outcome.

Note: A compiled solution is available in the class `ast.protocols.transportCO.impl.TProtocolStopWait` with protocol number equal to 202. You can run tests with your implementation in one end against our compiled solution in the other end.