New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Course Title: Data Communications & Networks
Course Number: CSCI-GA.2662-001
Instructor: Jean-Claude Franchitti
Session: 6

Assignment #6-2 (Programming)

I. Due

Thursday November 25, 2021.

II. Objectives

1. Get introduced to (or re-acquainted with) a very neat computational model called a finite state machine.
2. Build a finite state machine implementation of the TCP connection protocol (3-way handshake) for the both the client (active) side and the server (passive) side of a connection.
3. Leverage a JAVA package that implements a programmable finite state machine to write a program that uses these classes to implement the TCP connection protocol given in Figure 1 below.
4. Since you won't really get events from a communications link, a sequence of events will be simulated by reading and parsing commands from "standard input" (System.in in Java)

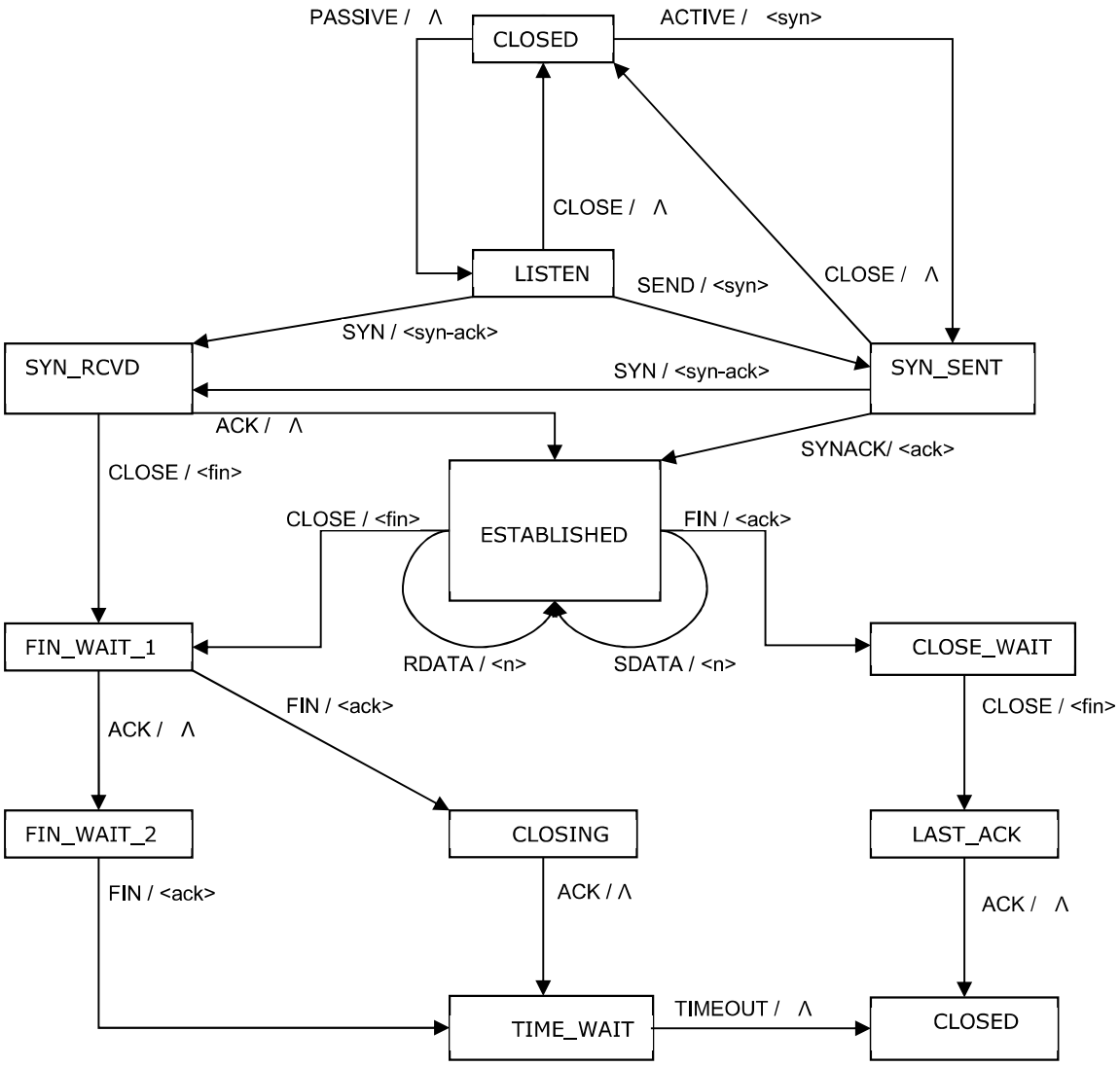

Figure 1 – TCP Connection Protocol (symbol Λ means "no action")

III. References

1. Slides and handouts posted on the course Web site
2. Textbook chapters as applicable

IV. Software Required

1. Microsoft Word.
2. Win Zip as necessary.
3. Java SE 8 (or different programming language platform if preferred)

V. Assignment

1. Get introduced (or reacquainted) with FSMs:

Finite state machines (fsm) are used to implement simulations, compilers, and, of most interest to us, communications protocols. You can learn about finite state machines from any text that covers Automata Theory or Theory of Switching Circuits, or from a myriad of sources on the World Wide Web. For our purposes, here is what you need to know:

Machines are devices that respond to a set of stimuli (events) by generating predictable responses (actions) based on a history of prior events (current state). Any object that behaves according to this stimulus/response relationship may be considered as a machine.

In order to represent a machine as a computational model, you need to define:

1. States
2. Events
3. Transitions
4. Start State

States represent the particular configurations that a machine can assume.

Events define the various inputs that a machine will recognize

Transitions represent a change of state from a current state to another (possibly the same) state that is dependent upon a specific event. In a Mealy machine, an FSM may generate an output upon a transition.

The **Start State** is the state of the machine before is has received any events.

Generally, finite a state machine is classified as either a Mealy machine - one that generates an output for each transition, or a Moore machine - one that generates an output for each state.

Moore machines can do anything a Mealy machine can do (and vice versa).

Practically, Mealy machines are more useful for implementing communications protocols.

The FSM that you will experiment with in this assignment is a Mealy machine.

2. Understand your program inputs requirements:

Your program MUST accept as input (from standard input) the Strings for events in the following table

EVENT	**Input string**
Passive Open	PASSIVE
Active Open	ACTIVE
SYN received	SYN
SYN + ACK received	SYNACK
ACK received	ACK
Data received from network	RDATA
Data to be sent from application	SDATA
FIN received	FIN
Client or Server issues close()	CLOSE
Timed wait ends	TIMEOUT

Events in standard input will be separated by white space **(note**: EOL is also white space!).

NOTE 1: Events **must** be specified exactly as shown in the table above. All events must be entered in **uppercase letters only**. Any parsed token (word) that does not exactly match one of these should be treated as invalid (incorrect) input.

NOTE 2: Figure 1 shows a transition from LISTEN state to SYN-SENT state driven by the SEND event. Your program should not handle this case so you should ignore this transition. A SEND event occurs when a server socket is converted implicitly from server to client by the owning application doing a write on the server socket.

3. **Understand the classes provided in the FSM package:**

The classes in the FSM package are:

FSM	The finite state machine class.
State	An abstract class that you will use to define the states for your FSM.
Event	A class that you will use to define the events for your FSM
Transition	A class that you will use to define the transitions in your FSM
Action	An abstract class that you will use to define the actions that you take on each transition.
FsmException	A class used to generate detectable errors in package classes.

The code for the FSM package is HERE. The package is in a jar file, so you will need to extract it or use the jar file in your classpath.

To use the classes, your JAVA program must import the classes in the Fsm package. The easiest way to use the package is to include the jar file in your class path when you compile and execute. For example, on Unix, you might try:

javac -cp .;./Fsm.jar myProgram.java to compile myProgram.java, and

java -cp .;./Fsm.jar myProgram to run myProgram

On Windows, the command is the same, but the classpath command argument is introduced by -cp rather than -classpath.

The documentation for the FSM package is HERE.

4. **Implement your program to operate as follows:**

Your program should loop, reading **ONE** event from standard in and processing that event **completely**, including display of any output, then continuing to read and process the next event, until end of input stream (end of file)

You MUST implement ALL of the states shown in the transition diagram.

You MUST Implement all of the transitions shown in the transition diagram AND the transitions for RDATA and SDATA.

Notice that there are two transitions in the Established state to handle data events. These are the SDATA (application request to send data) and the RDATA (data received from the network) events that can occur while in the ESTABLISHED State. You should handle these Events by writing an output message:

"DATA received **n**" when the event is RDATA

"DATA sent **n**" when the event is SDATA

Where **n** is a number representing the number of SDATA or RDATA Events received to date, including this (R|S)DATA Event. The transition for the (|S)DATA Event should leave the FSM in the ESTABLISHED State. The purpose of this requirement is to ask you to figure out how to associate a variable with a state. In practice, you would call a data transfer process which would have it's own FSM to implement flow control.

To simplify your task, you may treat ANY String that you read from standard input that is NOT one of the events defined here by writing:

*"Error: unexpected Event**: xxx***"

where **xxx** is the invalid event. Your program should then continue as if the "bad" input did not occur.

5. **Review the following FAQs:**

OK, so how do I use these classes to make an FSM?

1. Create classes for your states (by extending State) and allocate your states.
2. Allocate an FSM, giving it a name and a Start State (see the constructor). **NOTE:** Use ONLY the constructor that allows you to specify the FSM name and starting state:
   public FSM(String fsmName, State start)
3. Create your events (by allocating instances of Event).
4. Create classes for the actions to take on each transition (by extending Action) and allocate your actions.
5. Allocate instances of the Transaction class using the allocated State, Event, and Action objects, and add these Transition objects to your FSM object (see the addTransition() method in FSM).

How to I "send" events into the FSM?

The FSM class has a "doEvent" method that takes an Event object as its input.

What happens when I send an Event to the FSM?

Well, this is a Mealy machine, so the FSM will locate the Transition you defined that associates the Event with the current state of the FSM, then set the current state to the state defined in this Transition as the next state, then

it will execute the action method you specified in this Transition's Action object.

What do I do in my "Action" methods?

For all Transitions EXCEPT those caused by the (R|S)DATA Events, write the message:

"Event **eee** received, current State is **sss**"

where **eee** is the Event and **sss** is the current State.

For the Actions on the ESTABLISHED/(R|S)DATA Transitions, write the message:

"DATA received **n**" when the event is RDATA

"DATA sent **n**" when the event is SDATA

where **n** is a number representing the number of SDATA or RDATA Events received to date, including this (R|S)DATA Event.

What do I do if you send my program an Event that is not defined for the current State?

The FSM.doEvent() method will throw an FsmException if you pass an Event that is not defined for the current State. Make sure that you catch this exception and just display the exception (every Exception has a toString() method). Your program should then continue as if the "bad" Event did not occur.

How does my program terminate?

When you detect the end of the input stream on standard input, just exit.

6. **Email your assignment file to your TA.**

VI. Deliverables

1. Electronic:

Your assignment file must be submitted via NYU Brightspace. The file must be created and sent by the beginning of class. After the class period, the homework is late. The email clock is the official clock.

Your assignment file should include your program source code packaged as a JAR file along with a report document.

To create the JAR file containing your JAVA source code (please do not include the class files), change your working directory to the directory where your JAVA files are located and execute the command:

jar cvf **xxx**.jar *.java

where **xxx** is **YOUR STUDENT ID.**

Include the jar file in your assignment zip file and submit the zip file via NYU Brightspace.

The associated documentation provided in your zip file MUST include a readme file containing the name of the class that contains the main() method. It should also include a report document that elaborates on the design of your solution and other applicable details.

2. Report cover page and other formatting requirements:

The cover page supplied on the next page must be the first page of your report file.

Fill in the blank area for each field.

NOTE:

The sequence of the report file submission is:

1. **Cover sheet**
2. **Report Sheet(s)**

Report Layout (5%)

- Report is neatly assembled on 8 1/2 by 11 page layout.
- Cover page with your name (last name first followed by a comma then first name), and section number with a signed statement of independent effort is included.
- Program and documentation submitted for Assignment #6-2 are satisfactory.
- File name is correct.

VII. Sample Report Cover Sheet:

Name _____ Date: _____
(last name, first name)
Section: _____

Assignment 6-2 (Programming)

Total in points (100 points total): _____

Professor's Comments:

Affirmation of my Independent Effort: _____
(Sign here)