Praktikum Simulative Untersuchung von Protokollfunktionen des Internet

Aufgabenblatt 10 [1]

Vorbereitung

Aufgabe 1: State Machine

Am Ende dieses Aufgabenblattes finden Sie die Abbildung eines endlichen Automaten (Finite State Machine). Jeder Kasten stellt einen Zustand dar. Die gerichteten Verbindungen zwischen den Zuständen sind die zulässigen Zustandsübergänge. Ein Zustandsübergang wird von einem bestimmten Ereignis ausgelöst und verursacht bestimmte Aktionen. Ereignis und Aktionen stehen durch Schrägstrich getrennt am Zustandsübergang.

Zur Vorbereitung sollen Sie sich mit dem Zustandsdiagramm, das Sie in der praktischen Aufgabe implementieren sollen, vertraut machen. Nehmen Sie hierzu zwei verschiedene Münzen o.Ä. und fahren Sie die möglichen Abläufe der beiden Kommunikationpartner ab (jeder der beiden Kommunikationpartner hat eine solche Zustandsmaschine). Notieren Sie alle möglichen Abläufe vom Zustand `eCLOSED` bis wieder der Zustand `eCLOSED` erreicht ist (Tipp: Es sind insgesamt zehn).

Praktischer Teil

Denken Sie bitte – wie immer bei den praktischen Aufgaben– an aussagekräftige Debug-Ausgaben. Die Sourcen für diese Aufgabe befinden sich im Verzeichnis `protsim10`.

Ihre Aufgabe im praktischen Termin ist es, die Zustandsmaschine für Verbindungsauf- und abbau zu implementieren. Hierfür müssen Sie die `handleStateXXX()`-Funktionen in der Datei `transport/TransportEndpoint.cc` ergänzen. Beachten Sie bitte, dass den Konstanten im Zustandsdiagramm jeweils das Präfix `e` (für Enumeration) voranzustellen ist.

Ereignisse

Die folgenden Ereignisse können auftreten. Das Diagramm unterschlägt einige Ereignisse (z.B. RST-Nachrichten, Datenkommunikation), für welche die Verarbeitung schon für Sie implementiert wurde.

Ereignis	Beschreibung
`eSYS_OPEN`	Ankunft eines System-Calls vom Typ `SysCallOpen` von der Applikation.
`eSYS_*`	Ankunft eines System-Calls vom Typ `SysCall` mit dem angegebenen Kommando-Code.
`eSYN`	Ein `TransportPacket` mit der ID `eSYN` (Verbindungsaufbau), aber ohne Acknowledgement ist eingetroffen.
`eSYN,eACK`	Ein `TransportPacket` mit der ID `eSYN` und mit Acknowledgement ist eingetroffen.
`eACK`	Ein `TransportPacket` mit der ID `eACK`, d.h. ein Paket, das nur ein Acknowledgement enthält, ist eingetroffen. Denken Sie bitte ggf. an das Aktualisieren von `lastAckSeqNo` (die als nächstes *erwartete* Sequenznummer).
`eFIN`	Ein `TransportPacket` mit der ID `eFIN` (Verbindungsabbau) ist eingetroffen.
`eACK of eFIN`	Ein beliebiges `TransportPacket`, das ein Acknowledgement für das gesendete `eFIN` enthält, ist eingetroffen. Erkennbar ist dies daran, dass die Sequenznummer des `eFIN`-Pakets bestätigt wurde.
`eTIMER`	Der Timer ist abgelaufen.

[1] Stand: 12. Oktober 2023

Die mit R gekennzeichneten Zustände unterstützen das Empfangen von Daten durch die Applikation (eSYS_RECEIVE), die mit S gekennzeichneten zusätzlich das Senden (eSYS_SEND). Diese System-Calls sind in der Vorgabe schon implementiert. Das gleiche gilt für die Verarbeitung von Datenpaketen und Acknowledgements für Daten (eDATA und eACK).

Aktionen

Folgende Aktionen sollen bei den Zustandsübergängen ausgeführt werden. In der Regel können Sie sich zur Implementierung der Hilfsfunktionen im nächsten Abschnitt bedienen.

Aktion	Beschreibung
OKAY	Beantworte den eingehenden System-Call mit dem Ergebnis-Code eOKAY.
set parameters	Initialisiere Verbindungsparameter (peerNode, peerAppl, lastReceivedSeqNo (-1, falls noch kein Paket empfangen), lastAckSeqNo und nextSendSeqNo).
send *	Sende ein entsprechendes TransportPacket an das Gegenüber.
SYS_*	Sende einen entsprechenden System-Call an die Applikation.
set timer	Setze Timer auf timeout bzw. timewait.
clear timer	Lösche Timer.
reset parameters	Setze Verbindungsparameter zurück.

Hilfsfunktionen

Die Klasse TransportEndpoint, die Sie erweitern sollen, beinhaltet schon einige Hilfsfunktionen, die Ihnen helfen sollen, die Zustandsmaschine zu implementieren. Details zu den Methoden finden Sie in der API-Dokumentation.

Funktion	Beschreibung
createPacket()	Erzeugt ein TransportPacket mit der angegebenen ID, Sequenznummer und Acknowledgement-Nummer. Wird letztere nicht angegebenen (oder mit -1), enthält das Paket kein Acknowledgement-Feld.
selectInitialSeqNo()	Generiert eine initiale Sequenznummer für die Verbindung.
setTimer()	Setzt den Timer auf das relativ angegebene Timeout.
sendAck()	Sendet eine Acknowledgement-Nachricht mit der entsprechenden erwarteten Sequenznummer. Da Acknowledgements unter Umständen huckepack auf Datenpaketen reisen, sollte für Acknowledgements immer diese Methode verwendet werden.
sysCall()	Sendet einen System-Call mit dem angegebenen Kommando und Ergebnis (eOKAY, wenn weggelassen) an die Applikation.
sysCallReply()	Beantwortet den angegebenen System-Call mit dem angegebenen Ergebnis-Code (Default: eOKAY).
sendData()	Sendet ein Datenpaket. Alle Pakete ab Zustand eESTABLISHED (also auch eFIN) sollten so gesendet werden. Wenn der zweite Parameter true ist, wird das Sendefenster ignoriert (z.B. für eFIN).
resetConnection... ...Parameters()	Setzt die Verbindungsparameter zurück.
check()	Wird in der Regel in der Vorgabe schon aufgerufen. Prüft, ob die Nachricht vom richtigen Gegenüber kommt, eliminiert Duplikate und Out-of-Order-Pakete. Nur wenn diese Funktion true zurückgibt, sollte die Nachricht weiter verarbeitet werden. Ansonsten hat sich schon check() darum gekümmert.
reject()	Wird in der Regel in der Vorgabe schon aufgerufen. Lehnt die angegebene Nachricht (SysCall oder TransportPacket) mit dem angegebenen Fehler ab.
handleReset()	Wird in der Vorgabe schon aufgerufen. Behandelt ankommende eRST-Nachrichten angemessen.

Aufgabe 2: Implementierung der Zustandsmaschine
Implementieren Sie die Zustandsmaschine unter Benutzung der angegebenen Hilfsmethoden.

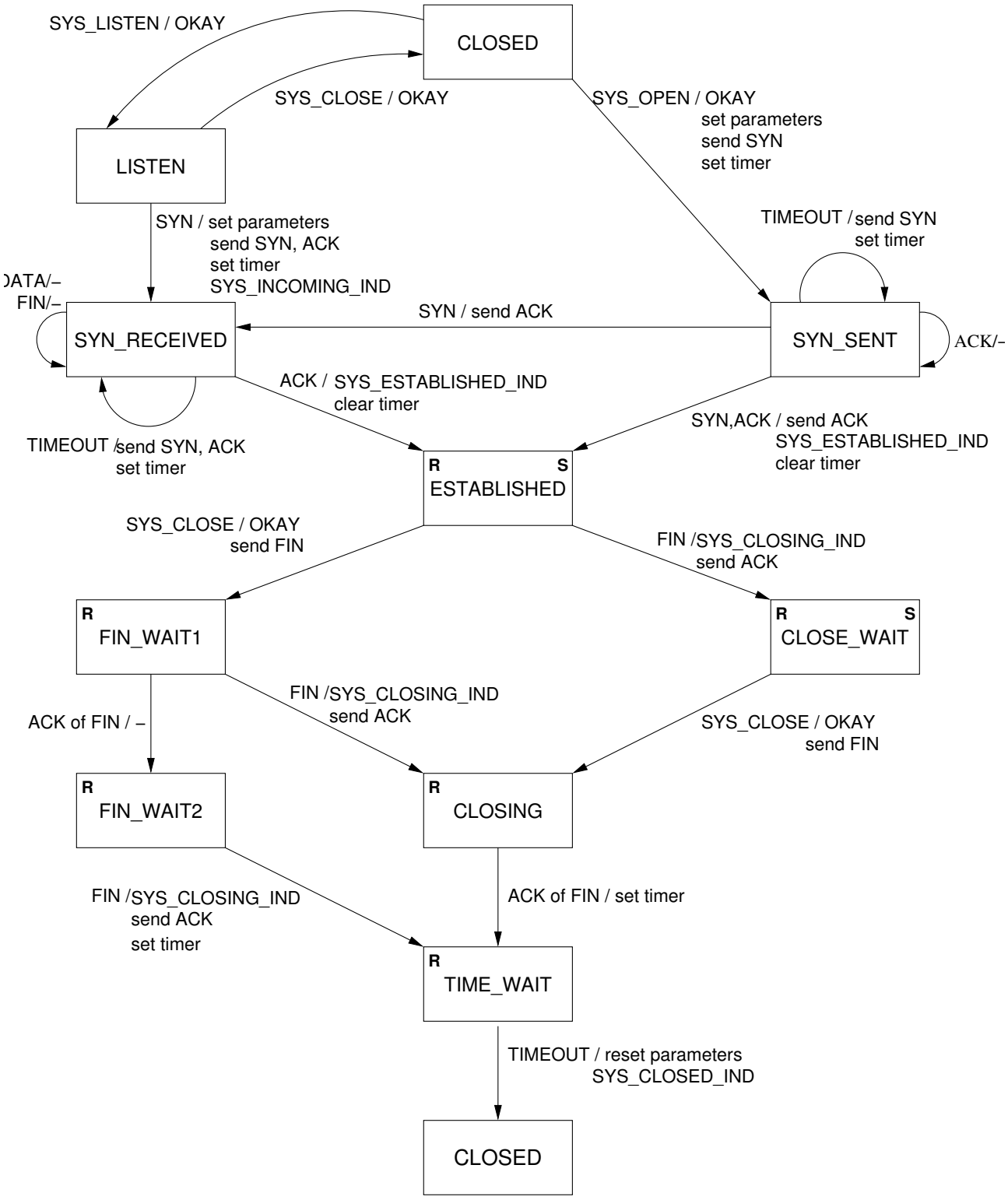
Aufgabe 3: Testen
Rufen Sie Run 1 auf. Die schon implementierten `PeerToPeerAppl`-Applikationen werden nun versuchen dynamisch Verbindungen auf- und abzubauen. Die Zustandsautomaten schreiben Ihren aktuellen Zustand und das jeweils auftretende Ereignis in Vektoren.

Nach Durchlauf der Simulation wechseln Sie bitte in das Verzeichnis `eval` und zerlegen Sie die Vektor-Datei durch Aufruf von `./splitpsvec.py ../results/transpA01.vec.psOV out` in alle (Eingabe: `*`) einzelnen Vektoren. Und rufen Sie anschließend das Test-Skript mit `./protsim_tcp_verify.pl $(seq -f out-%g.vec 0 79)` auf. Dieses prüft, ob nur gültige Abläufe auftraten und wie oft verschiedene Sequenzen von Zuständen durchlaufen wurden. Prüfen Sie, ob die Ergebnisse plausibel sind (welche Abläufe treten sehr häufig auf, welche sehr selten).

Hinweise zu `./protsim_tcp_verify.pl $(seq -f out-%g.vec 0 79)`:

- Sie können beim Aufruf `stderr` umleiten (z.B. nach `/dev/null`), um eine bessere Übersicht zu erhalten (auch bei einer richtigen Lösung gibt es Fehlermeldungen, da nicht alle Vektoren vom implementierten Transport-Modul generiert werden).
- Die Ausgabe jeder Zeile ist als Vektor zu lesen: welcher Index im Vektor für welchen Ablauf steht, wird am Ende ausgegeben.

Sie werden feststellen, dass neben den von Ihnen in der Vorbereitung identifizierten Abläufen einige weitere auftreten können, z.B. wenn versucht wird eine Verbindung mit einer Applikation aufzubauen, die schon mit einem anderen Gegenüber kommuniziert.


CLOSED
SYS_LISTEN / OKAY
SYS_CLOSE / OKAY
SYS_OPEN / OKAY
set parameters
send SYN
set timer
LISTEN
SYN / set parameters
send SYN, ACK
set timer
SYS_INCOMING_IND
TIMEOUT / send SYN
set timer
DATA/–
FIN/–
SYN_RECEIVED
SYN / send ACK
SYN_SENT
ACK/–
ACK / SYS_ESTABLISHED_IND
clear timer
TIMEOUT / send SYN, ACK
set timer
SYN,ACK / send ACK
SYS_ESTABLISHED_IND
clear timer
R S
ESTABLISHED
SYS_CLOSE / OKAY
send FIN
FIN / SYS_CLOSING_IND
send ACK
R
FIN_WAIT1
R S
CLOSE_WAIT
FIN / SYS_CLOSING_IND
send ACK
ACK of FIN / –
SYS_CLOSE / OKAY
send FIN
R
FIN_WAIT2
R
CLOSING
FIN / SYS_CLOSING_IND
send ACK
set timer
ACK of FIN / set timer
R
TIME_WAIT
TIMEOUT / reset parameters
SYS_CLOSED_IND
CLOSED