Laborversuch zu Rechnernetze 1

Versuch 3: *Socket*-Schnittstelle

Versuchsziel: Benutzung der *Socket*-Schnittstelle mit Verwendung der Transportprotokolle TCP und UDP, Beeinflussung von sockets mit „setsockopt()“, Beobachtung der Arbeitsweise des ONC-RPC sowie Analyse von Datenkommunikation über Tunnel.

Die Aufgaben können in Gruppen zu 2 Personen oder auch allein bearbeitet werden. Für alle Teilaufgaben werden lauffähige Programme vorgegeben. Diese sind in der Versuchsvorbereitung mit Hilfe dieser Beschreibung sowie mit den vorgegebenen Source-Files zu analysieren, um die Funktion der einzelnen Programme zu verstehen. Die Sources müssen in einigen Teilaufgaben geringfügig modifiziert bzw. erweitert werden (s.u.). Fast alle Modifikationen sind bereits soweit vorbereitet, dass Sie über bedingte Kompilierung die Modifikation aktivieren können. Dazu ist allerdings eine Analyse des Source-Codes erforderlich. Außerdem werden fertige Skripte zum Übersetzen der Programme bereitgestellt.

Zur Beobachtung der Arbeitsweise aller Protokolle wird das Programm „Wireshark“ benutzt, welches alle Pakete, die auf der Ethernet-Schnittstelle eines Arbeitsplatzrechners gesendet oder empfangen werden, aufzeichnen und in lesbarer Form darstellen kann. Unter Linux kann man auch Pakete aufzeichnen, welche über die „localhost“-Schnittstelle von/zum eigenen Rechner übertragen werden.

Wireshark benutzt zum Aufzeichnen der Pakete das Programm „tcpdump“ (bzw. unter Windows „windump“). Wenn man bei Wireshark beim Aufzeichnen „capture filter“ benutzen möchte, so ist für die Filtereinstellung die Syntax für „capture filter“ zu verwenden, welche bei „tcpdump“ gilt. Die Syntax kann man über die entsprechenden Hilfefunktionen von „tcpdump“ (z.B. unter UNIX: man tcpdump) oder aus Wireshark erhalten.
Hinweis: Bei allen Daten, die mit „Wireshark“ aufgezeichnet werden, sind die Rohdaten zusätzlich in einer „capture“-Datei abzuspeichern (Endung: „.cap“). Damit ist auch noch nachträglich eine Analyse mit anderer Darstellungsart möglich. Benutzen Sie dazu die Option „**save as**“ in „Wireshark“.

Hinweis:
Der Versuch wird im RN-Labor an zwei benachbarten Rechnern unter dem Betriebssystem LINUX durchgeführt. Dabei können Sie ihren normalen RZ-Login benutzen (NIS).
(es wird hier kein lokaler Login mehr benötigt).

Hinweise zu Wireshark unter Betriebssystem Linux im RN-Labor:

Wireshark muss mit „sudo“ gestartet werden (wegen der notwendigen root-Rechte).

Speichern Sie ein Wireshark-Ergebnisse (capture-file) auf ihrem Home-Laufwerk bzw einem Unterverzeichnis ab.

Hinweis zur Wireshark-Darstellungen im Laborbericht:

Bei allen Aufgaben reicht in der Regel die Darstellung und Erläuterung der „packet summary line“ vollständig aus! Nur beim rpc (optionale Zusatzaufgabe 3) sind auch weitere Detaildarstellungen sinnvoll.

Für die Erstellung des Laborberichtes sollten Sie am besten das frei verfügbare Programm „wireshark“ auf einem eigenen PC installieren. Bitte beachten Sie dabei, dass folgende Einstellungen vorgenommen werden:

Einstellungen von Wireshark für Laborbericht:

Edit → Preferences.. → Protocols →

IP →

Reassemble fragmented IP datagrams: deaktivieren

Validate the IP checksum if possible: deaktivieren

TCP →

Validate the TCP checksum if possible: deaktivieren

Analyze TCP sequence numbers: aktivieren

Relative sequence numbers and window scaling: aktivieren

UDP →

Validate the UDP checksum if possible: deaktivieren

Edit → Preferences.. → name resolution→

Enable MAC name resolution: deaktivieren

Enable network name resolution: deaktivieren

Enable transport name resolution: deaktivieren

(alle 3 ohne Häkchen)

1.0 Allgemeine Angaben für alle Programme

TCP und UDP verwenden „Ports“ zur Unterscheidung verschiedener Anwendungen. Damit keine Konflikte bei gemeinsam genutzten Rechnern entstehen können, passen Sie jeweils die Portnummern auf Ihren Arbeitsplatz an.

Beispiel: In der Client-/Server-Anwendung aus Aufgabe 1 wählen Sie als **Serverport die Zahl 9000 + Endnummer (2 letzte Ziffern) ihres *Clients***. (Beispiel: *Client* = itpc3104 ➔ Serverport = 9004)

Für alle Aufgaben stehen fertige Skripte zum Übersetzen und Binden zur Verfügung. Die Skriptdateien (Endung „.sh“) müssen Ausführungsrechte haben, ggf. müssen Sie nach einem Entpacken von ZIP-Dateien die Rechte entsprechend setzen, z.B. mit „**chmod +x *.sh**“ .

Beim Ausführen von eigenen Skripten und selbst erstellten ausführbaren Programmen muss immer der Pfad mit angegeben werden, d.h. das Skript „coli_cli_serv_s.sh“ wird ausgeführt mit:

./coli_cli_serv_s.sh /* “**./** “ bedeutet: Datei im eigenen Verzeichnis */

Ausführen eines Programms mit:

./client_s.out [<server_name or server_ip>] [<server_port>]

Die Ausgaben von Client und Server kann man mit der Maus markieren und dann mit der mittleren Maustaste in einen Editor kopieren.

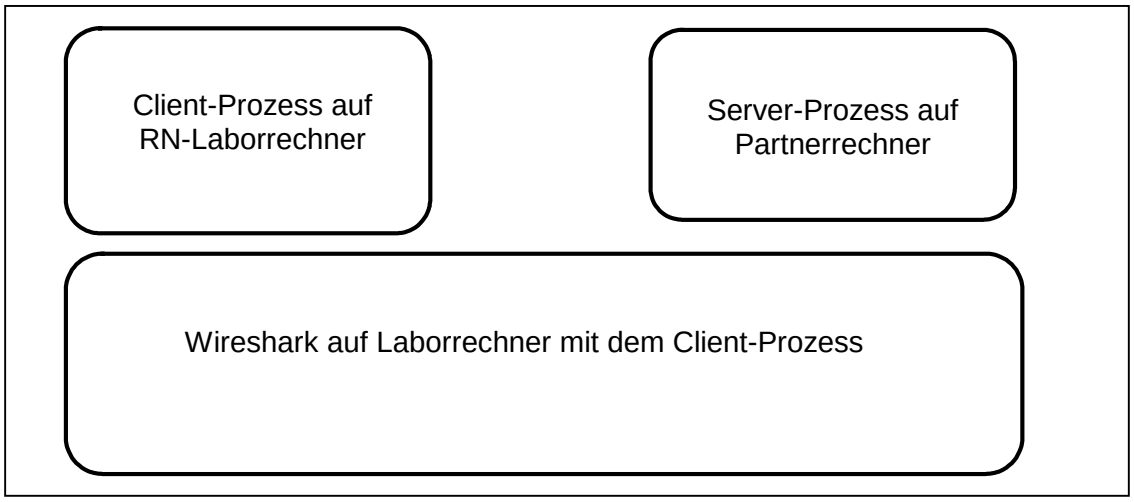

Abbildung 1: Anordnung der Fenster bei Versuchsdurchführung.

Hinweise:

Beachten Sie, dass bei Verlassen Ihres Arbeitsplatzes ggf. zuvor im Hintergrund gestartete Server-Prozesse, die nicht beendet wurden, mit "kill ..." oder mit Strg+C (bzw Ctrl+C), sofern die Maus im Fenster des laufenden Prozesses positioniert ist, abgebrochen werden.

Es ist sinnvoll, alle Programme direkt beim Start mit den benötigten Parametern zu versorgen, damit man die Programmaufrufe aus dem Befehlsspeicher wiederholen kann und nicht immer im Dialog innerhalb des Programms nachträglich die erforderlichen Parameter eingeben muss. Siehe folgende Beispiele zum Start des Servers mit der Portnummer als Parameter und Start des Clients mit IP-Adresse und Port als Parameter (s.u.):

./server_s.out 9007

./client_s.out 134.108.8.32 9007

(Näheres zu den Parametern: siehe Quellcode der Programme)

Aufgabe 1: TCP-Sockets

Es werden vollständige Programme mit Aufrufen aus der socket-Library für einen *Client* und einen nicht nebenläufigen Server vorgegeben. Kopieren Sie die vorgegebenen Quellprogramme in ein neues Arbeitsverzeichnis für diese Laboraufgaben (z.B. ~../rn/lab3/sock_tcp) und wechseln Sie bei Ausführung der Programme in allen Dialogfenstern in dieses Unterverzeichnis. Ändern sie nur die verwendeten Portnummern für Ihr Programm. Die Programme sind danach nur zu übersetzen und auszuführen. Die beim Ablauf der Programme übertragenen Daten sind zu beobachten und danach in Dateien abzuspeichern.

Durch Dialogeingaben kann man den Ablauf beeinflussen und durch gezielt eingesetzte Verzögerungen die Arbeitsweise der Protokolle beeinflussen, um möglichst viele Effekte darstellen zu können. Die Puffergrößen von 60000 Bytes bzw. 14000 Bytes sind ebenfalls so gewählt, dass unter geeigneten Bedingungen bestimmte Effekte erzielt werden können.

Aufgabe 1.1) Nicht nebenläufiger Server, 1 *Client*

Die Grobstruktur des Servers und des Clients sind im Folgenden dargestellt.

Server (server_s):

- Initialisierung der *Sockets* und Warten auf Verbindungen von *Clients*
- Bei ankommender Verbindung eines *Clients*:
  - -- Sofern im Server Lesen vom Client gewünscht ist (durch Dialogabfrage realisiert), großen Block lesen (60000 Bytes mit „*read*“) und am Bildschirm Anzahl der jeweils mit „read()“ gelesenen Zeichen sowie Anfang der übergebenen Daten ausgeben.
  - -- Sofern im Server Schreiben zum Client gewünscht ist (durch Dialogabfrage realisiert), senden einer langen Nachricht (14000 Bytes) zum *Client*. Diese beginnt mit einem Text (mit IP-Adresse, Portnummer und aktueller Uhrzeit), der Rest des Sendepuffers ist mit konstanten Zeichen aufgefüllt.
  - -- Anschließend liest der Server (ohne weiteren Dialog) eine Nachricht vom Client. Diese enthält eine Nachricht zum Beenden der Verbindung. Dann wird die Verbindung zum aktuellen Client geschlossen und erneut auf eingehende Verbindungsanforderungen von neuen Clients gewartet.

Der Server-Prozess kann nur durch einen „kill <process_id>“-Befehl beendet werden.

Client (client_s):

Zu obigem Server wird das entsprechende *Client*-Programm zur Verfügung gestellt. Bei Aufruf des Clients kann als optionaler Parameter der Name des Servers direkt angegeben werden.

Der *Client* stellt zu Anfang eine Verbindung zum Server her.

Nach Auffüllung des Sendefeldes (60000 Bytes) wird – über Dialogabfrage gesteuert – das Sendefeld mit *„write“* zum Server geschickt. Anschließend liest der Client – wiederum über Dialogabfrage gesteuert – vom Server einen Block von insgesamt 14000 Bytes. Nach Ausgabe eines Teils der empfangenen Daten sendet der Client noch eine letzte Nachricht zum Server. Diese Nachricht enthält eine Kennung zur Beendigung der aktuellen Verbindung zum Server. Anschließend schließt der Client die Verbindung und terminiert.

1.1.1 Ein Server / Ein Client

Übersetzen Sie die vorgegebenen Programme und führen Sie die Programme auf zwei verschiedenen Rechnern im RN-Labor aus. Beobachten Sie den Ablauf des Programms und die über das Netz ausgetauschten Pakete mit Hilfe des Programms Wireshark und speichern sie die beobachteten Pakete in einer Datei.

Schneiden sie alle PDU's mit. Empfohlene „capture"-Filter-Einstellung (bei Serverport = 9001): „**port 9001 or icmp**" /* statt „icmp" auch „ip proto 01" möglich */
Speichern Sie auch die Bildschirmausgaben des Servers in einer Textdatei.

Der Laborbericht soll nur einen Ausschnitt der PDU's enthalten (Verbindungsaufbau, Datenübertragung mit Flusskontrolle anhand von ca. 8-10 Paketen erläutern und Verbindungsabbau, nur TCP-Header und IP-Header in Kurzfassung, keine Detail-Darstellungen). Außerdem muss der Laborbericht alle erzeugten Ausgaben des Servers (von stdout) enthalten. Erklären Sie, woran eindeutig die nicht nebenläufige Arbeitsweise des Servers erkennbar ist.

Hinweis: Alle aktuellen TCP-Implementierungen können in jedem TCP-Paket eine 12 Bytes lange Timestamp-Option (im TCP-Header) benutzen. Ob diese Option benutzt wird, kann bei der Erstellung des Betriebssystems festgelegt werden (Voreinstellung: mit Timestamp-Option). Die Zeitstempel werden zur optimalen Einstellung der von TCP benutzten Timer verwendet.
Starten Sie zunächst den Server. Danach starten Sie einen einzigen Client, übertragen Sie Daten in beiden Richtungen und beenden Sie den Client.

1.1.2 Ein Server / Zwei Clients

Anschließend starten Sie bei weiter laufendem Server (oder nach Abbruch des Servers und Neustart des Servers) zunächst einen Client **A** und senden Daten zum Server. Starten Sie anschließend in einem weiteren Fenster einen weiteren Client **B** und schicken Sie Daten von **B** zum Server. Anschließend setzen sie Client **A** sowie den Server jeweils fort. Nach Terminierung des ersten Clients (**A**) können Sie den zweiten Client (**B**) vom Server bedienen lassen, dann den Client **B** terminieren. Der Laborbericht muss auch die Ausgaben des Servers (von stdout) enthalten. Erläutern Sie im Bericht, woran man eindeutig die sequenzielle Arbeitsweise des Servers erkennt. Beenden Sie den Server durch Abbruch des Serverprozesses (z.B. mit „kill <process_id>").

1.1.3 Ein Server / Ein Client (mit vorzeitiger Beendigung des Clients)

Starten Sie erneut zuerst den Server, dann einen einzigen Client. Im Server beantworten Sie zunächst die Frage nach Lesen mit „**n**" und die anschließende Frage nach Schreiben mit „**j**". Im Client beantworten Sie nun die Frage nach Schreiben mit „**n**" und die anschließende Frage nach Lesen ebenfalls mit „**n**". Kommentieren Sie vor allem die Terminierung des Clients (vor Auslesen der vom Server geschriebenen Daten) am Ende des Protokollmitschnitts (nur

die Terminierung in den Bericht aufnehmen, der Anfang mit Verbindungsaufbau und Datenübertragung kann weggelassen werden).

Aufgabe 1.2) *Client-Server*-Kommunikation über Tunnel

Verwenden Sie nun den Server-Prozess aus Aufgabe 1.1 auf dem Rechner mit der IP-Adresse „134.108.190.10“ im RN-Labor. Dieser Rechner steht in einem anderen Subnetz im RN-Labor und ist über einen GRE-Tunnel (siehe Labor 1) mit anderen Rechnern des Labors verbunden. Für den Tunnel sind weitere Header mit zusätzlich insgesamt 24 Bytes (neben IP und TCP) erforderlich, die vom einen Tunnel-Endpunkt jeweils eingefügt und auf der Gegenseite entfernt werden. Nehmen Sie an, dass innerhalb der verschiedenen Subnetze auf Data Link Layer jeweils maximal 1500 Bytes lange Pakete (oberhalb des MAC-Layers) transportiert werden können.
Melden Sie sich vom zweiten Arbeitsplatzrechner aus zunächst mit ssh auf dem Rechner „134.108.190.10“ mit Ihrer Arbeitsplatz-Kennung an.
Login/Passwort: rn-lab31xx bei itpc31xx
(z.B. bei itpc3105 **ssh rn-lab3105@134.108.190.10** Passwort rn-lab3105)

Dort haben Sie Zugriff auf das Home Verzeichnis /home/rn-lab31xx. Kopieren Sie Ihre Quelldateien dorthin und übersetzen Sie es auf dem 134.108.190.10 (häufig werden diese Dateien dort vorhanden sein). Kopieren z.B. mit:
scp -r meine_dateien rn-lab3105@134.108.190.10:/home/rn-lab3105

Starten Sie vom zweiten Arbeitsplatzrechner aus über die ssh-Verbindung den Server (aus Aufgabe 1.1) auf 134.108.190.10 und anschließend starten Sie einen Client auf ihrem Arbeitsplatzrechner. Zeichnen Sie die zwischen den beiden Rechnern bei Ausführung der Programme übertragenen PDU’s auf dem Rechner mit dem Client-Prozess auf (Capture Filter wie in Aufgabe 1.1) und erklären Sie nur die ersten aufgezeichneten 10 PDU’s (inklusive der ICMP-PDU). Analysieren Sie die übertragenen PDU’s mit Daten im Laborbericht und erläutern Sie kurz, warum die Datenpakete in relativ ungünstige Größen zerlegt werden.
Modifizieren Sie nun Ihren Client so, dass die **mss** auf den Wert eingestellt wird, der für diese Konstellation maximal möglich ist, sodass sofort alles ohne Fehler (keine ICMP-Nachricht) übertragen werden kann. Hinweis: Die **mss** wird zwar immer vom Empfänger vorgegeben und nur bei Verbindungsaufbau übertragen. Wenn Sie die **mss** im Client für den angegebenen socket ändern, so ist die Wirkung so, dass die beim Verbindungsaufbau vom Client empfangene **mss** mit dem geänderten Wert lokal überschrieben wird.

Vorbereitungsaufgabe zu 1.2:

Welche **mss** (maximum segment size) sollte am besten für diese Verbindung benutzt werden?

Hinweis zur Durchführung von Aufgabe 1.2:

Der TCP/IP-Stack erkennt schon bei Übertragung der ersten (großen) PDU mit Daten, dass das angesprochene Ziel nur über einen Tunnel erreichbar ist. Er erkennt dies daran, dass eine von ihm gesendete (große) Daten-PDU nicht mehr vollständig (unfragmentiert) in den Tunnel passt. In diesem Fall sendet der Tunneleingangsknoten eine ICMP-Nachricht (Packet to big, MTU of next hop is …). Der TCP/IP-Stack merkt er sich diesen Wert dann für alle nachfolgenden Datenübertragungen, indem er die MTU für Sendungen über diesen Tunnel entsprechend verkleinert. Falls Sie beim ersten Mal nicht alles richtig aufzeichnen, kann beim zweiten Versuch der beabsichtigte Effekt dann nicht mehr beobachtet werden. Daher muss vor neuen Versuchen das Interface zurückgesetzt werden. Dazu können Sie das vorbereitete Script „restartnetwork" benutzen, welches "root"-Rechte benötigt. Das Script muss daher mit „sudo" gestartet werden.

sudo restartnetwork

Aufgabe 1.3) Zustand *half closed* untersuchen

Aktivieren Sie nun im Client aus Aufgabe 1.1 den Code zum einseitigen Schließen des Clients, d.h. dass der Client nicht mehr schreiben, wohl aber noch lesen kann. Der Server bleibt unmodifiziert. Führen Sie dann Client und Server auf den benachbarten Rechnern so aus, dass nur noch vom Server zum Client geschrieben wird. Achten Sie bei der Bedienung des Clients darauf, dass Sie nach dem „Schließen zum Schreiben" nicht mehr versuchen zum Server zu schreiben. Bei dieser Versuchskonstellation kann der Server zwar die Ende-Bedingung vom Client nicht mehr empfangen, da der Client ja nicht mehr schreiben kann, dies spielt für die Versuchsdurchführung aber keine Rolle. Schneiden Sie die erzeugten PDU's mit und stellen Sie im Laborbericht nur die kommentierten PDU's (Nur TCP- und Header in der Kurzfassung der „packet summary line") dar, aus denen der Übergang zum Half-Close-Zustand ersichtlich ist und aus denen erkennbar ist, dass nur noch in einer Richtung Daten übertragen werden.

Aufgabe 1.4) *Client* mit "bind"

Der *Client* aus Aufgabe 1.1 wird nun modifiziert, um das Verhalten von TCP beobachten zu können. Dabei werden 2 Varianten erstellt (mit Fehlerbehandlung und ohne Fehlerbehandlung hinter „bind()“).

Es wird explizit mit "bind()" gearbeitet. Auf *Client*-Seite wird der Port "9000 + Endnummer ihrer Workstation" (siehe Aufgabe 1.1) fest benutzt. Aktivieren Sie zunächst den Code für die Benutzung von „bind()“ ((#define MIT_BIND 1) und führen Sie keine Fehlerbehandlung hinter dem „bind()“-Aufruf durch (#define BIND_MIT_FEHLERBEHANDLUNG 0). Erstellen Sie anschließend eine zweite Variante dieses Programms, indem Sie den Return-Code von „bind()“ auswerten (#define BIND_MIT_FEHLERBEHANDLUNG 1).

Starten Sie (bei zuvor gestartetem Server) zunächst den Client ohne Fehlerbehandlung zweimal parallel und übertragen Sie nach dem Start keine Daten zwischen Client und Server. Der zu erläuternde Effekt tritt nur dann auf, wenn Sie nach dem Start der Clients nur die Anfragen im Client direkt nacheinander beantworten (die Fragen nach „Lesen“ und „Schreiben“ jeweils mit „**n**“ beantworten und erst danach die entsprechenden Fragen im Server ebenfalls jeweils mit „**n**“ beantworten. Erklären Sie an Hand der Quellprogramme und an Hand der Wireshark-Aufzeichnungen den Effekt, der bei Parallellauf von zwei Clients ohne Fehlerbehandlung auftritt.

Warten Sie nach Beendigung aller Clients mindestens 2 Minuten und führen Sie dann die gleichen Abläufe wie zuvor mit dem Client mit Fehlerbehandlung durch, d.h. bei laufendem Server kurz nacheinander zweimal parallel den Client mit Fehlerbehandlung starten und genauso bedienen wie oben, ebenso alles aufzeichnen und die Ergebnisse im Bericht erläutern.

Hinweis: Zeichnen Sie die übertragenen PDU’s mit Wireshark auf. Für den Bericht benötigen Sie nur jeweils die 2 ersten aufgezeichneten PDU’s bei einem Programmstart. Alle erzeugten Ausgaben sind in eine Datei zu schreiben und das Programmverhalten ist im Laborbericht für beide Programmvarianten zu erläutern.

Aufgabe 1.5) Nebenläufiger Server , 2 *Clients*

Server und Client aus Aufgabe 1.1 wurden als Ausgangsprogramme zur Erstellung eines nebenläufigen Servers (server_n) mit zugehörigem Client (client_n) verwendet. Beide Programme wurden gegenüber den Versionen aus Aufgabe 1.1 geringfügig vereinfacht, indem die Lese-Schreibpuffer verkleinert wurden, der Server ohne Dialogeingaben läuft und im Client nur noch eine einzige Dialogabfrage vor dem ersten Schreibvorgang zum Server vorhanden ist. Damit kann man die Nebenläufigkeit durch gleichzeitigen Betrieb mehrerer Clients steuern. Bei den Serverausgaben wird zusätzlich die Prozessnummer mit ausgegeben, um

die verschiedenen nebenläufigen Server-Prozesse unterscheiden zu können. Außerdem werden im Client Schreib- und Lesevorgang (mit jeweils einer Anfrage vorher) zweimal hintereinander ausgeführt, damit man besser die nebenläufige Arbeitsweise des Servers aus den Ausgaben erkennen kann. Der Server liest und schreibt entsprechend zweimal.

Testen Sie die Nebenläufigkeit des Servers, indem zwei identische *Client*-Prozesse mit demselben Server nacheinander einen Kommunikationsvorgang starten (zwei Verbindungen gleichzeitig). Die Nebenläufigkeit der beiden Clients erreichen Sie auf jeden Fall, wenn Sie zunächst einen Client **A** in einem Fenster starten, die Dialogabfrage zum Senden im Client **A** zunächst nicht beantworten. Dann starten Sie den zweiten Client **B** in einem neuen Fenster und beantworten die Fragen nach Senden zum Server so, dass gesendet wird. Beantworten Sie erst jetzt die Dialogabfrage nach Senden in Client **A** und im nächsten Zyklus von Client **A** erneut so, dass jeweils gesendet wird. Erst danach beantworten Sie im zweiten Zyklus von Client **B** die Dialogabfrage nach Senden positiv. Die Beendigung des Servers kann wiederum durch Stoppen des Serverprozesses („kill <process_id>“) vorgenommen werden, wenn alle Clients zu Ende bedient sind. Nehmen Sie aus dem Protokollmitschnitt nur einige Pakete für den Laborbericht, aus denen die Nebenläufigkeit des Servers eindeutig ersichtlich ist (alle Verbindungsaufbau-Pakete für beide Verbindungen, jeweils 4 weitere Datenpakete je Verbindung). Außerdem muss der Laborbericht alle erzeugten Ausgaben des Servers (von stdout) enthalten. Erläutern Sie im Bericht, woran man eindeutig die nebenläufige Arbeitsweise des Servers erkennen kann.

Aufgabe 2: UDP-Sockets

Kopieren Sie die vorgegebenen Quellprogramme in ein neues Arbeitsverzeichnis für diese Laboraufgaben (z.B. ~/kt/lab3/sock_udp) und wechseln Sie bei Ausführung der Programme in allen Dialogfenstern in dieses Unterverzeichnis. In den Dateien „requester.c“ und „responder.c“ ist eine Kommunikation zwischen 2 Partnern über das UDP-Protokoll realisiert. Übersetzen Sie diese Programme mit dem zur Verfügung gestellten Skript „coli_req_resp_linux.sh“.

Beide Programme starten Sie am besten ebenfalls mit Parametern:
requester.out [responder-node] [own-port partner-port]
reponder.out [requester-node] [own-port partner-port]
Beispiele (responder läuft auf 134.108.8.40, port 9001, requester auf 134.108.8.41, 9002)
./requester.out 134.108.8.40 9002 9001
./responder.out 134.108.8.41 9001 9002

Aufgabe 2.1) Requester und Responder im gleichen Subnet

Passen Sie zunächst die verwendeten Ports (9000 + Arbeitsplatznummer ihres Rechners bzw. 9001 + Arbeitsplatznummer ihres Rechners) in requester,c und responder.c an. Übersetzen Sie die Programme mit dem gegebenen Script.

Lassen Sie zunächst Requester und Responder auf 2 Rechnern im RN-Labor laufen und übertragen Sie (über Dialog gesteuert) einmal vom Requester zum Responder sowie vom Responder zum Requester. Schneiden Sie die durch ihre Programmausführung übertragenen PDU's mit und kommentieren Sie das Geschehen im Laborbericht. Achten Sie darauf, dass durch Ihre Filter-Einstellungen keine wichtigen PDU's verloren gehen. Es sollen alle PDU's zwischen beiden Rechnern sowie evtl. auftretende ICMP-PDU's aufgezeichnet werden.

Mögliche Capture-Filter-Einstellung:

(host 134.108.8.xx) and (host 134.108.8.yy) or (ip proto 01)

Aufgabe 2.2) Requester und Responder in verschiedenen Subnetzen

Melden Sie sich wieder vom zweiten Arbeitsplatzrechner aus mit ssh auf dem Rechner mit der IP-Adresse „134.108.190.10" im RN-Labor an. Setzen Sie zunächst wieder die Netzkarte auf Ihrem Arbeitsplatzrechner in den Anfangszustand („sudo restartnetwork"). Lassen Sie den Requester auf Ihrem Arbeitsplatzrechner und den Responder über die ssh gesteuert auf dem Rechner mit der IP-Adresse „134.108.190.10" laufen. Schneiden Sie dabei die übertragenen PDU's auf Ihrem Arbeitsplatzrechner mit und kommentieren Sie die PDU's (jeweils ein voller Block mit insgesamt 1472 Bytes) Capture-Filter: beide IP-Adressen sowie zusätzlich alle ICMP-Nachrichten, analog zu Aufgabe 2.1.

Modifizieren Sie nun den Requester so, dass er die Pakete mit TTL = 1 abschickt (im Requester: **#define MY_TTL 1**) und lassen Sie erneut die Programme laufen. Beachten Sie, dass beide Rechner über einen Router miteinander verbunden sind. Auf einer Teilstrecke der Route ist eine **kleinere MTU** als die von Ethernet mit DIX-Rahmenformat.

Aufgabe 3) RPC (optional)

In den Dateien „onc.x", „onc_main.c", „onc_func.c" und „onc_lx.sh" ist eine lauffähige RPC-Anwendung vorgegeben. Mit dem Skript „onc_lx.sh" können Server- und Clientprogramm als ausführbare Dateien erzeugt werden. Erzeugen Sie mit Hilfe dieses Skriptes Server- und Client-Programm. Loggen Sie sich (z.B. remote) auf einen anderen Rechner ein und starten Sie dort das Serverprogramm im Hintergrund. Überzeugen Sie sich davon, ob das Serverprogramm läuft und ob der RPC registriert ist. Anschließend starten Sie auf Ihrem Client-

Rechner zunächst „Wireshark“ und dann ihr Client-Programm. Stellen Sie das Paketfilter so ein, dass nur die zwischen Client und Server ausgetauschten Pakete herausgefiltert werden. Stoppen Sie nach Beendigung der Aufgabe auch den Serverprozess und löschen Sie die registrierten RPC’s. Eigene Registrierungen im *Portmapper* können Sie mit folgendem Kommando unter LINUX löschen:

sudo rpcinfo -d 999999977 3 /* Programmnr = 999999977 , Version = 3 */
bzw.
sudo /usr/sbin/rpcinfo -d 999999977 3

Erläutern Sie die Pakete, die für diese Anwendung zwischen Client- und Server ausgetauscht werden (im Laborbericht). Lenken Sie die erzeugte Ausgabe in eine Datei und fügen Sie diese dem Bericht bei.

Vorgabedateien

Die Dateien befinden sich als ZIP-komprimierte Ordner zur Lehrveranstaltung RN1 im ELearn-System Moodle unter: **RN1 ➔Labor ➔ lab3_socket**

tcp.zip (für Aufgaben 1.1 bis 1.5)

Nicht nebenläufiger Server

client_s.c	**//** Source für Client, Partner zu server_s.c
server_s.c	**//** Source für nicht nebenläufigen Server, Partner für client_s.c
coli_cli_serv_s.sh	// Skript zum Übersetzen

Nebenläufiger Server

client_n.c	**//** Source für Client, Partner zu server_n.c
server_n.c	**//** Source für nebenläufigen Server, Partner für client_n.c
coli_cli_serv_n.sh	// Skript zum Übersetzen

udp.zip (für Aufgaben 2.1 bis 2.2)

requester.c	**//** Source für Requester
responder.c	// Source für Responder
coli_req_resp.sh.sh	// Skript zum Uebersetzen

rpc.zip (für optionale Aufgabe 3)

onc_lx/onc.x	// RPC-Definitionsdatei
onc_lx/onc_main.c	// RPC: Client-Main-Programm
onc_lx/onc_func.c	// RPC: Serverfunktionen
onc_lx/onc_lx.sh	// RPC: Übersetzungsskript

Versuchsdurchführung und Laborbericht:

Die Programme sind bei Versuchsdurchführung vorzuführen. Die mitprotokollierten Dateien sollen je Programmlauf zusammengefasst und durch geeignete Kommentare ergänzt werden. Dabei ist die Versuchskonstellation in die Protokolldateien einzuarbeiten und die Pakete sind kurz zu kommentieren. Die Kommentare stehen jeweils vor dem kommentierten Paket bzw. vor einer Gruppe von Paketen. Die erzeugten Ausgaben (von „stdout“) sind in eine Datei umzuleiten (z.B. mit „tee“) oder mit „cut and paste“ zu kopieren und mit abzugeben.
Abzugeben sind **Ausdrucke** (keine Emails) der **kommentierten** Protokollmitschnitte. In den Protokollmitschnitten ist auch zu vermerken, durch welche Operation im *Client* bzw. im Server (gesteuert durch die Programmeingaben zur Laufzeit) die jeweiligen PDU's verschickt wurden. Aus dekodierten Protokollmitschnitten sind die Rahmen zu entfernen, die nicht wesentlich für die jeweilige Aufgabe sind bzw. die in Wiederholung auftreten. (z.B. Flusskontrolle nur einmal exakt kommentieren).
Ebenso ist die Darstellung so zu wählen, dass nur die für die jeweilige Aufgabe notwendigen Details kompakt dargestellt werden. (Nicht alle Felder auf allen Ebenen dekodieren, z.B. wenn TCP-Flusskontrolle erläutert werden soll, so sind nur einige Felder aus dem TCP-Header darzustellen und die Werte zu erklären!)

Weitere Hinweise zur Erstellung des Laborberichts:

Moodle: RN1 ➔Labor ➔ lab3_socket: Labsock_Hinweise_Laborbericht.pdf