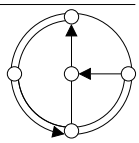
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



SS 2005 Prof. R. Wattenhofer / Keno Albrecht / Angela Nicoara

Vernetzte Systeme
Übung 5

Ausgabe: **28. April 2005**
Abgabe: **13. Mai 2005**

Bitte schreiben Sie immer Ihre(n) Namen auf die Lösungsblätter.

1 IM–Service

In den letzten beiden Übungen haben Sie einen einfachen Instant Messenger geschrieben. Die Arbeit umfasste die Registrierung beim Server sowie den Austausch von Nachrichten untereinander. In dieser Übung sollen Sie sich einen beliebigen Dienst ausdenken und implementieren. Wir haben auf unseren Servern z.B. einen einfachen ECHO-Dienst bereitgestellt, der an ihn gesendete Nachrichten an den Absender zurückschickt.

Der Dienst kann als *Computerbenutzer* angesehen werden, der sich wie gehabt beim Server registriert, aber auf Anfragen automatisch Antworten generiert und zurücksendet. Es ist ein eigenständiges Programm, das unabhängig von Ihrem Instant Messenger lauffähig sein soll. Natürlich kann der Dienst auf den bisherigen Quellcode aufgebaut werden. Wir geben daher in dieser Aufgabe kein neues Rahmengerüst für Sie vor! Sollten Sie mit Ihrer eigenen Lösung der Aufgabe 4 nicht zufrieden sein, so nehmen Sie die von uns bereitgestellte Beispiellösung als Vorlage.

Für einen *menschlichen* Benutzer erscheint der Dienst als ganz normaler Eintrag in der Benutzerliste. Dem Dienst können wie gewohnt Textnachrichten (als `MESSAGE`!) in einem von ihm spezifizierten Format (siehe c)) gesendet werden. Diese Nachrichten werden dann vom Dienst bearbeitet und eine Textnachricht als Antwort zurückgesendet.

Bearbeiten Sie nun die folgenden Teilaufgaben. Reichen Sie Ihre Lösungen per Email oder ausgedruckt bei Ihrer/Ihrem AssistentenIn ein.

a) Denken Sie sich einen beliebigen Dienst aus, und spezifizieren Sie seine Aufgabe. Welche Eingabe(n) (Parameter) benötigt er? Welche Ausgabe produziert er?

b) Nehmen Sie sich Ihre Lösung oder unsere Beispiellösung als Vorlage, und entwickeln Sie daraus ein Programm, das sich wie in Übung 4 wiederholt beim Server registriert (TCP) und Nachrichten empfangen und senden kann (UDP). Entfernen Sie unnötigen "Ballast" – eine GUI oder ein Menü ist wahrscheinlich nicht mehr notwendig! Der Dienst soll sich beim Server mit dem Benutzernamen `SERVICE:<service-name>_<übungsgruppe>_<real-names>` anmelden. Dabei steht `<service-name>` für einen aussagekräftigen, aber kurzen Namen des Dienstes, `<übungsgruppe>` für den Raum Ihrer Übungsgruppe (etwa `HG_F26.1`) und `<real-names>` für Ihre(n) Namen.

c) Sehen Sie in Ihrem Code vor, dass Sie bei Empfang einer Nachricht mit dem Text "help" eine kurze Beschreibung Ihres Dienstes zurücksenden. Geben Sie auch an, welche Nachricht(en) Ihr Dienst erwartet, um seine Aufgabe(n) auszuführen und welche Parameter benötigt werden.

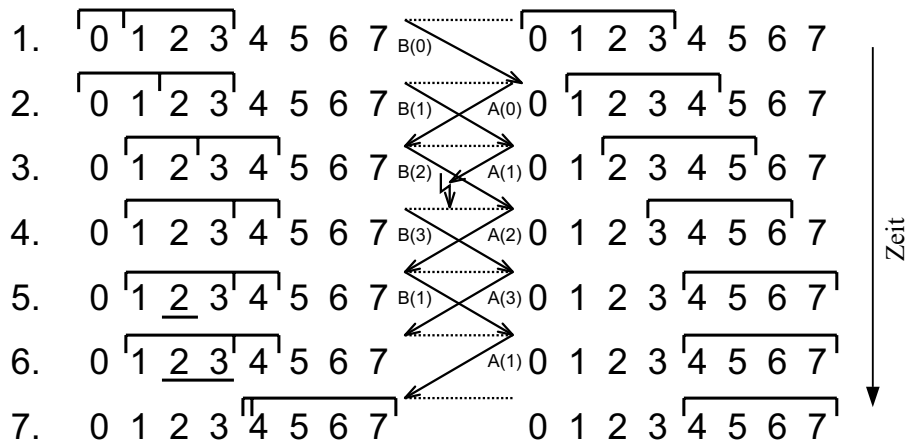
d) Implementieren Sie Ihren Dienst! Wenn Sie die von Ihnen erwartete Nachricht mit gültigen Parametern empfangen, dann bearbeiten Sie diese und schicken das Ergebnis als Nachricht an den Absender zurück. Empfangen Sie eine ungültige Nachricht, so senden Sie einfach die Beschreibung des Dienstes zurück. Seien Sie kreativ!

2 Schiebefenster und Sequenznummern

In der Vorlesung haben Sie *go-Back-N* und *selective repeat* als zwei Formen für zuverlässige Datentransferprotokolle mit Pipelining kennen gelernt. Pipelining bedeutet dabei, dass der Sender nacheinander mehrere Pakete verschicken kann und diese erst nach und nach vom Empfänger bestätigt (acknowledged) werden können. Damit dies funktioniert, verwenden diese Protokolle auf Sender- und Empfängerseite Puffer, sogenannte Schiebefenster (sliding windows). Für zuverlässigen Datentransfer (damit die Reihenfolge der Pakete erhalten bleibt und kein Paket verloren geht) werden Sequenznummern (sequence numbers) verwendet.

a) In dieser Aufgabe spielen Sie selber einmal das Selective-Repeat-Protokoll durch. Folgendes Beispiel zeigt den Zeitablauf für das Sliding-Window-Protokoll mit Sende- und Empfangspuffergrösse 4, bei dem die Bestätigung des Blocks B(1) verloren geht. Pro Zeiteinheit wird maximal ein Paket versandt. Im Sendefenster sind zusätzlich die Blöcke markiert, die schon bestätigt wurden. Als Fehlerbehandlungsverfahren wird *selective repeat* mit einem Timeout von 3 Zeiteinheiten verwendet, d.h. falls ein gesendeter Block nach drei Zeiteinheiten nicht bestätigt ist, wird er nochmals gesendet.

In jedem Schritt (zum Zeitpunkt der gepunkteten Linien) wird zuerst ein allfälliges Paket empfangen, dann entschieden, ob und wenn ja welches Paket versandt wird, dann dieses ggf. versandt und der neue Zustand dargestellt. Im Beispiel läuft die Zeit von oben nach unten. Dargestellt ist jeweils der Zustand des Sende- bzw. Empfangsfensters kurz nach dem Zeitpunkt der gepunkteten Linien. Der Abstand zwischen zwei gepunkteten Linien entspricht der Zeiteinheit T. Nachrichten sind jeweils genau eine Zeiteinheit (also T) unterwegs.



Nun zu Ihrer Aufgabe: Es sollen 3 Blöcke mit den Sequenznummern 0-2 übertragen werden. Dazu wird das Sliding-Window-Protokoll mit Sendefenstergrösse 3 und Empfangsfenstergrösse 2 und ein Sequenznummernbereich von [0, 5] verwendet. Als Fehlerbehandlungsverfahren verwenden wir *selective repeat* mit einem Timeout von 3 Zeiteinheiten. Block B(0) geht bei der ersten Übertragung verloren. Auch die Bestätigung A(0) der erneuten Übertragung von B(0) geht verloren. Alle anderen Übertragungen verlaufen erfolgreich.

Zeigen Sie mit Hilfe eines Diagramms analog zum Beispiel oben, wie die Fensterstellung nach jedem Erhalt eines Blocks bzw. einer Bestätigung aussieht. Zeichnen Sie nach jeder Übertragung bzw. Bestätigung den Zustand des Sende- und Empfangsfensters in die Schablone a) auf dem Zusatzblatt zum Übungsblatt ein. Markieren Sie auf Empfängerseite auch bereits empfangene, aber noch nicht weitergereichte Pakete.

b) Angenommen, der Sequenznummernbereich für Aufgabenteil a) sei [0, 3], ansonsten bleibe alles gleich. Zeichnen Sie den Ablauf in die Schablone b) ein. An welcher Stelle (in welcher Zeile) scheitert der Algorithmus? Warum scheitert er?

c) Wie viele Sequenznummern (k) sind mindestens nötig, so dass Probleme wie auf Folie 3/41 nicht vorkommen können? Begründen Sie ihr Resultat. Überlegen Sie sich dazu, wie gross der mögliche Bereich von Sequenznummern im Sende- und Empfangsfenster sein kann. Gehen Sie davon aus, dass Sende- und Empfangsfenster gleich gross sind, beide haben Grösse w. Nehmen Sie weiter an, dass der Empfänger Pakete, die in der richtigen Reihenfolge empfangen werden, direkt der Anwendungsschicht weiter reicht.

3 Minimale Fenstergrösse

Über eine Glasfaserstrecke von 5000 km mit einer Bandbreite von 2 Gbps werden Datenblöcke der Grösse 1000 Byte mit dem Sliding-Window-Protokoll gesendet.

a) Wieviele Blöcke sollte das Sendefenster mindestens fassen, um einen kontinuierlichen Datenstrom zu gewährleisten? Geben Sie zu Ihrer Berechnung die getroffenen Annahmen an.

b) Ist es sinnvoll, das Sendefenster grösser als den in a) gefundenen Wert zu machen?

4 TCP Fenstergrösse und Effizienz

Auf Folie 3/43 ist der TCP-Header dargestellt. In dieser Aufgabe beschäftigen wir uns genauer mit einem Feld davon.

a) Die gewünschte Fenstergrösse wird dem Kommunikationspartner im **Window**-Feld (rcvr-window-size) des TCP-Headers mitgeteilt. Überlegen Sie, ob die Grösse dieses Feldes mit 16 Bits genügend und optimal ist, besonders unter Berücksichtigung grosser Verzögerungen und hoher Bandbreiten. Erkennen Sie ein mögliches Problem? Notieren Sie Ihre Idee.

b) Berechnen Sie die Effizienz einer TCP-Verbindung mit einer Bandbreite von 100 Mbps über einen geostationären Satelliten. Nehmen Sie an, dass immer so viele Bytes gesendet werden, wie es das Window-Feld erlaubt. Wie gross können der tatsächliche Durchsatz und die Effizienz (Utilization) dieser Verbindung maximal sein? Treffen Sie vereinfachende Annahmen und notieren Sie diese.

c) Wie könnte man das Problem in einer Weise lösen, die mit dem ursprünglichen TCP-Protokoll, so wie es in RFC 793 definiert wurde, interoperabel ist.[1]

5 TCP-Zustandsdiagramm

Untenstehende Tabelle zeigt den Nachrichtenaustausch zwischen einem HTTP-Server (Port 80) und einem HTTP-Client. Es sind die TCP-Header der Nachrichten einer vollständigen TCP-Verbindung, inklusive Verbindungsauf- und -abbau, dargestellt. TCP ist im RFC 793 (`http://www.rfc-editor.org/rfc/rfc793.txt`) beschrieben.

Nr.	Data Length	Src. Port	Dst. Port	Seq. Nr.[2]	Ack. Nr.[3]	Flags
1	0	1000	80	3459	0	SYN
2	0	80	1000	8656	3460	ACK SYN
3	0	1000	80	3460	8657	ACK
4	676	1000	80	3460	8657	ACK PSH
5	0	80	1000	8657	4136	ACK
6	247	80	1000	8657	4136	ACK PSH
7	0	1000	80	4136	8904	ACK
8	171	80	1000	8904	4136	ACK PSH
9	0	80	1000	9075	4136	ACK FIN
10	0	1000	80	4136	9076	ACK
11	0	1000	80	4136	9076	ACK FIN
12	0	80	1000	9076	4137	ACK

a) Welche Nachrichten gehören zum Verbindungsaufbau (3-Way-Handshake)? Welche Aufgabe haben die SYN-Nachrichten und welche Sequenznummern werden ihnen jeweils zugeordnet?

b) Welche Nachrichten gehören zum Verbindungsabbau? Was bedeutet ein gesetztes FIN-Bit und welche Sequenznummern haben die FIN-Nachrichten?

[1] Hinweis: Der TCP-Header lässt sich um **Options** der Form (Optionstyp, Optionslänge, Parameter) erweitern.
[2] TCP ordnet jedem einzelnen Datenbyte (also nicht bloss jedem Segment) eine Sequenznummer zu. Auch SYN und FIN haben je eine eigene Sequenznummer.
[3] Eine Acknowledgement-Nummer gibt die Sequenznummer des nächsten erwarteten Datenbytes an und bestätigt kumulativ alle Datenbytes mit kleineren Sequenznummern.

c) Ein TCP-Modul kann als endlicher Automat aufgefasst werden. Die Eingaben dieses Automaten sind einerseits empfangene TCP-Segmente, andererseits (Benutzer-)Kommandos, wie z.B. OPEN, CLOSE, SEND und RECEIVE. Die Ausgaben des Automaten sind verschickte TCP-Segmente und Rückgabewerte der Kommandos. Die Zustände und Zustandsübergänge von TCP sind in Abb. 6 (TCP Connection State Diagram[4]) von RFC 793 dargestellt.

Vollziehen Sie den Austausch der Nachrichten in obiger Tabelle anhand des Zustandsübergangsdiagramms für Client und Server nach. Nehmen Sie an, dass Client und Server sich zu Beginn im **CLOSED**-Zustand befinden, der Server das Kommando **passive OPEN** und der Client das Kommando **active OPEN** erhält. Legen Sie dazu je eine Tabelle für Client und Server nach folgendem Muster an (Nummer der gesendeten/empfangenen Nachricht in Klammern angeben):

Server:

Zustand	**Eingabe**	**Ausgabe**	**Nachfolgezustand**
CLOSED	passive OPEN	—	LISTEN
...	...	...	...

Client:

Zustand	**Eingabe**	**Ausgabe**	**Nachfolgezustand**
CLOSED	active OPEN	send SYN (1)	SYN SENT
...	...	...	...

d) Eine mögliche Folge von Zustandsübergängen beim Verbindungsabbau führt über FIN WAIT-1, CLOSING und TIME WAIT. Unter welchen Umständen wird diese Folge durchlaufen?

[4] Auf der Vorlesungs-Homepage findet sich ein Link zu Abb. 6 (TCP-Zustandsübergangsdiagramm) von RFC 793.