Rechnernetze Beleg

January 9, 2023

Beleg von Lukas Kouba (s83783)

Erklaerung der einzelnen Funktionen:
Erklarung nur von Funktionen welche von Flashness123 geschrieben wurden
Also nicht von ARQ, Channel, CustomLoggingHandler, FileCopy, FT

Arten von Packets:

AcknowledgePacket:
- dient als ack Paket
- wird von server an client geschickt
- besteht aus sessionNumber, packetNumber, gbnN(1, also anzahl wie viele Pakete zu senden sind(aufeinmal))
- ganz zum schluss wird crc32 in data verschickt, deshalb auch 2 Konstruktoren

DataPacket:
- von Client an Server
- beinhalten Datenpakete bestehend aus bytes
- besteht aus sessionNumber, paketNumber, data
- in letztem DataPacket wird data mit der crc32 befullt

StartPacket:
-von Client an Server
- beginnt mit "Start"
- besteht aus Sessionnummer, PacketNumber(beim Startpacket naturlich 0), "Start", fileLength(Lange der zu ubertragenden Datei),

fileName(lange des Dateinamens an sich), fileName (eigentlicher Name der Datei)
- crc32 wird nur aus den letzten dreien gebildet
- crc wird 2mal gepruft, einmal nach dem startpaketm und einmal nachdem alle Dateien ubertragen wurden

Funktion von FileTransfer:

- hat 2 Konstruktoren, einen fur Client und einen fur Server
- Client benotigt host, socket, fileName und arq(arq entscheidet ob sw oder gbn ausgefuhrt wird, bei mir nur sw implementiert)
- moglicher Aufruf: client localhost 42069 "4922B.sql" sw
- Server benotigt socket und dir, moglich auch loss und delay
- moglicher Aufruf: server 42069 0.1 100

Funktion von file-init():

- wird aufgerufen, nachdem der server gestartet wurde, wird also von handleConnection aufgerufen
- wartet auf das Startpacket und offnet es
- vergleicht ob "Start" zwischen 3 und 8er Position in Bytekette
- allocated den benotigten speicher wie in Bytekette angegeben + von getMTU()(Bug)
- extrahiert die sessionnumber aus den ersten 3 bit
- falls alles passt wird ein acknowledgePacket zuruckgeschickt
- nun wird data-ind-req solange aufgerufen solange nicht numberOfBytesReceived mit der angegebenen Dateilange ubereinstimmt, Ergebnisse in data gespeichert und dann an
bytesReceived weitergegeben
- in data-ind-req werden weiter
- danach wird erneut ein AcknowledgePacket gebildet und verschickt
- File wird gespeichert mit utf8 coding (name aus Startpacket geholt), dir als Pfad
- falls existent, wird 1 angehangt, (macht teilweise dateiendung kaputt)

Funktion von file-req():

- wird gestartet nachdem Client gestartet wird(von sendFile)
- generiert random sessionNumber

- erstellt neues StartPacket und gibt sessionNumber, fileName, filelange weiter
- StartPacket wird dann in packetData gespeichert und an data-req() weitergegeben
- data-req() verarbeitet das startpacket und gibt true zuruck
- je nach grosse der MTU (bei uns wie bestpractice 1480) werden nun neue Daten geladen
- in schleife an data-req ubergeben, bis keine Daten mehr vorhanden
- packetNumber wird itteriert, damit der vergleich mit den acks klappt
- nun wird die crc32 gebildet und zum letzten packet verpackt
- dieses wird abgeschickt und lastTrasmission auf true gesetzt

Funktion von data-req():

- zieht sich aus hlData welches das Startpaket am Anfang ist die Session und Packet number
- setzt packetRetries auf null
- geht in Schleife, in der es versucht hlData(mit DataPackets gefullt) zu versenden, ein ack Packet zu empfangen, aus diesem die Session und Packet number zu lesen und mit den eigenen Werten zu vergleichen
- bei Fehlversuch wird packetRetries bis 10 itteriert und dann abgebrochen

Funktion von data-ind-req():

- funktion zum daten speichern und acks senden
- bekommt mit, wie viele bytes noch zu ubertragen sind
- empfangt ein packet und speichert dies in data
- bildet ackpacket und sendet es
- returned byte array von der grosse der MTU oder remaining

Lernportfolio:

- Fortschritte auf Github zu sehen

- Generelles vorgehen:

1. Beleg als zip-Datei runterladen
2. Nur lokal auf rechner bearbeiten und coden (grosster Zeitanspruch)
3. Nachdem das geklappt hat, die Dateien nach und nach auf Github hochladen und nochmal genau durchgehen ob der code so ausfuhrbar ist
4. Verbesserungen nacharbeiten und Bugs die ubersehen wurden fixen
5. Make-Datei schreiben und auf Linux System testen
6. Funktionen erklaren, teils mit bereits beim coden geschriebenen Kommentaren

Verbesserungsvorschlage:

Der Beleg war generell viel zu schwer und kaum nur mit der Erfahrung aus C++ machbar. Auch wenn man fruh genug anfangt, braucht man ewig dafur und das Debuggen macht nach den ersten Versuchen auch nur wenig Spass. An sich wurde das Ziel wahrscheinlich erreicht und zwar kann ich jetzt deutlich besser mit Java als Sprache umgehen und verstehe die Zusammenarbeit zwischen Client und Server deutlich besser. Das Problem bei dem Beleg ist, dass man so sehr ins kalte Wasser geworfen wird, dass selbst StackOverflow und diverse Videos einem nur wenig helfen. Ich musste oft andere nach Hilfe fragen oder mir von ihnen teilweise die Aufgabenstellung erklaren lassen. Ich denke ein etwas weniger intensiver Beleg wurde seinen Zweck besser erfullen. Generell fand ich es aber sehr gut, dass wir uns in ein teilweise geschriebenes Projekt einarbeiten mussten, mit LaTeX gearbeitet haben und ein neues Thema anhand einer neuen Sprache gelernt haben.