

Systementwurf Gruppe 10

Systemprogrammierung

vorgelegt von:

Frank Klameth

xxxxx

frank.klameth@hs-weingarten.de

Simon Westphal

20xxx

simon.westphal@hs-weingarten.de

Michael Wydler

20168

michael.wydler@hs-weingarten.de

22. Oktober 2010

Inhaltsverzeichnis

1	Client	4
1.1	Module	4
1.2	Programmstart	4
1.3	Strukturen (intern)	4
1.3.1	Benutzerrollen	4
1.3.2	Schreib- und Leserecht	5
1.4	Abläufe	5
1.4.1	Programmstart	5
1.4.2	Command-Thread	5
1.4.3	Live-Agent	6
1.4.4	GUI	7
1.4.5	Listener-Thread	7
1.5	Tafel-Trigger	8
2	Server	9
2.1	Module	9
2.2	Programmstart	9
2.3	Strukturen (intern)	9
2.3.1	Benutzerrollen	9
2.3.2	Clientliste	10
2.3.3	Tafel	10
2.4	Abläufe	10
2.4.1	Programmstart	10
2.4.2	Signal System beenden	11
2.4.3	Login-Thread	11
2.4.4	Client-Thread	11
2.4.5	Broadcasting-Agent (Thread)	13
2.5	Logger	13
2.6	Archivierer	13
3	Netzwerk	15
3.1	Allgemeine Definitionen	15
3.1.1	Datentypen	15
3.1.2	Serialisierung	15
3.1.3	Benutzerrollen	15
3.1.4	Schreib- und Leserecht	15
3.1.5	Message-Types	16
3.2	Kommunikationsablauf	16
3.2.1	Login	16
3.2.2	Quit	16

3.2.3	Request	16
3.2.4	Shutdown	16
3.2.5	Release	17
3.2.6	Acquire	17
3.2.7	Modify	17
3.2.8	Clear	17
3.2.9	Status	17
3.3	Datenstrukturen	17

1 Client

Der Client stellt eine Verbindung zum Server her. Es werden beim starten des Client die Server- und Logindaten angegeben.

1.1 Module

- Login
- Benutzeroberfläche (GUI)
- Live-Agent
- Listener-Thread
- Netzwerkkommunikation
 - Struct vor Senden Serialisieren
 - Nach Empfang wieder umwandeln

1.2 Programmstart

Das Programm wird aus der Konsole gestartet. Es müssen folgende Parameter angegeben werden:

- Servername oder IP (Servername wird in IP umgewandelt)
- Port
- Benutzername
- Rolle

Dabei können Benutzername und Rolle frei gewählt werden. Ist der Benutzername schon vergeben, wird Ist die angegebene Rolle schon belegt, wird der Benutzer automatisch als Student eingetragen.

Beispiel: > ./client 127.0.0.1 8080 michael student

1.3 Strukturen (intern)

1.3.1 Benutzerrollen

```
1 enum ROLE {  
2     Dozent = 1  
3     Tutor = 2  
4     Student = 3  
5 };
```

1.3.2 Schreib- und Leserecht

Als Datentyp wird `uint8_t` verwendet.

- Modify 0 = nur lesend
- Modify 1 = schreibzugriff (exklusiv)

1.4 Abläufe

1.4.1 Programmstart

```

1 Start mit Parameter (Server-IP, Port, Username, Rolle)
2 Socket für Netzworkkommunikation öffnen
3 Logindaten (Username und gewünschte Rolle) an Server senden
4 Login erfolgreich?
5     wenn NEIN:
6         Fehlermeldung ausgeben
7         Kill: Client
8     wenn JA:
9         Userdaten und -rechte speichern (ID, Name, Rechte)
10 Mutex für lokalen Tafelzugriff initialisieren (gesperrt)
11 Initialisiere lokale Tafel
12 Starte Command-Thread
13 Starte Live-Agent
14 Starte GUI
15 Starte Listener-Thread
16 Starte Trigger für Live-Agent
17     Mutex-Down für lokale Tafel
18     Fordert aktuellen Tafelinhalt an
19     Mutex-Up

```

1.4.2 Command-Thread

```

1 > quit (Client beenden)
2     Sende Befehl "quit" an den Server
3     Mutex-Down für lokale Tafel
4     Beende Trigger für Live-Agent
5     Kill: Listener-Thread
6     Kill: GUI
7     Kill: Live-Agent
8     Kill: Command-Thread
9     Lösche lokale Tafel
10    Lösche Mutex für Tafelzugriff
11
12 > request (Schreibrecht anfordern)
13     Ist Client Student?
14     Wenn JA:
15         Sende Befehl "request" an den Server

```

```

16     Ist Client Dozent?
17     Wenn JA:
18         Dialog ob Benutzer schreibrecht bekommen soll
19         Sende Antwort an Server
20     Schreibrecht erteilt?
21     Wenn JA:
22         Deaktiviere Button 'Schreibrecht anfordern'
23         Aktiviere Button 'Schreibrecht abgeben'
24         Schreibrecht auf lokale Tafel gewähren
25     Wenn NEIN:
26         Hinweis das Anfrage abgelehnt wurde.
27
28 > shutdown (System beenden)
29     Ist Client Dozent?
30     Wenn JA:
31         Sende Befehl "shutdown" an den Server
32
33 > release (Schreibrecht abgeben)
34     Ist Client Tutor?
35     Wenn JA:
36         Sende Befehl "release" an den Server
37
38 > acquire (Schreibrecht entziehen)
39     Ist Client Dozent?
40     Wenn JA:
41         Sende Befehl "acquire" an den Server
42
43 > clear (Tafel löschen)
44     Hat Client Schreibrechte?
45     Wenn JA:
46         Sende Befehl "clear" an den Server

```

1.4.3 Live-Agent

```

1 > modify (Tafel ändern)
2     Hat Client schreibrecht?
3     Wenn JA:
4         Mutex-Down für lokale Tafel
5         Ist Tafel voll?
6         Wenn JA:
7             Fehlermeldung
8         Wenn NEIN:
9             Schreibe Änderung in lokale Tafel
10        Mutex-Up
11        Trigger für Tafel starten.
12    Trigger für Tafel sendet dann die Daten in bestimmten
        Intervallen.
13    Trigger abgelaufen?
14    Wenn JA:
15        Mutex-Down für lokale Tafel

```

```

16         Sende Tafel an Server
17         Erfolgreiche Sendung?
18         Wenn NEIN:
19             Tafel nochmals senden
20         Mutex-Up

```

1.4.4 GUI

```

1 // Tafel wird als GtkTextView gespeichert.
2 GtkTextBuffer *gtkbuf = gtk_text_view_get_buffer(textview);
3
4 GtkTextIter startIter, endIter;
5 char *mybuf;
6
7 gtk_text_buffer_get_start_iter(gtkbuf, &startIter);
8 gtk_text_buffer_get_end_iter(gtkbuf, &endIter);
9
10 // Speichern in char*
11 mybuf = gtk_text_buffer_get_text(gtkbuf, &startIter, &endIter, FALSE
12 );
13 // Tafel leeren
14 gtk_text_buffer_set_text(gtkbuf, "", -1);
15
16 // Tafel wieder befüllen
17 gtk_text_buffer_set_text(gtkbuf, mybuf, -1);

```

1.4.5 Listener-Thread

```

1 Wartet auf Nachrichten vom Server (Broadcasting-Thread)
2 Aktualisierung der lokalen Tafel und der Statusinformationen.
3
4 > board_modified (Tafel-Update)
5     Mutex-Down für lokale Tafel
6     Tafel aktualisieren
7     Mutex-Up
8
9 > states_changed (Statusänderung)
10     GUI-Informationen aktualisieren
11
12 > my_state_changed (eigene Rechte bekommen/entzogen)
13     Schreibrecht erhalten?
14     Wenn JA:
15         Button "Schreibrecht anfordern" deaktivieren
16         Tafel editierbar setzten
17     Schreibrecht abgegeben/entzogen?
18     Wenn JA:
19         Tafel nicht-editierbar setzten
20         Button "Schreibrecht anfordern" aktivieren

```

1.5 Tafel-Trigger

Wenn auf die Tafel geschrieben wird, dann wird ein Timeout-Signal gestartet. Wenn dieses abgelaufen ist, wird die Tafel an den Server gesendet und somit an alle Clients verteilt. Bei jeder Änderung wird der Timeout zurückgesetzt. Wenn der Timeout 5x zurückgesetzt wurde, dann wird die Tafel dennoch zum Server gesendet und der Timeout-Counter zurückgesetzt.

```
1 Tafel wird geändert
2   Timeout (200ms) wird (neu) gestartet
3   Timeout-Counter +1
4   Timeout abgelaufen oder Timeout-Counter = 3?
5   Wenn JA:
6       Mutex-Down für lokale Tafel
7       Tafel an Server senden
8       Timeout-Counter = 0
9       Mutex-Up
```


2 Server

Der Client stellt eine Verbindung zum Server her. Es werden beim starten des Client die Server- und Logindaten angegeben.

2.1 Module

- Login-Thread
- Broadcasting-Agent
- Client-Thread
- Clientliste/Benutzerverwaltung/Rechteabfrage
- Tafel + Zugriffsfunktionen
- Schnittstelle: Server <-> Logger Kommunikation
- Schnittstelle: Server <-> Archivierer Kommunikation
- Netzwerkkommunikation
 - Struct vor Senden Serialisieren
 - Nach Empfang wieder umwandeln

2.2 Programmstart

Das Programm wird aus der Konsole gestartet. Es müssen folgende Parameter angegeben werden:

- Port

Beispiel: > ./server 8080

2.3 Strukturen (intern)

2.3.1 Benutzerrollen

```
1 enum ROLE {  
2     Dozent = 1  
3     Tutor = 2  
4     Student = 3  
5 };
```

2.3.2 Clientliste

```

1 struct CLIENTLIST {
2     int sockd;
3     char name[25];
4     enum ROLE role;
5     /*
6      * Modify 0 -> nur lesend
7      * Modify 1 -> schreibzugriff (exklusiv)
8      */
9     int modify;
10    struct CLIENTLIST *previous;
11    struct CLIENTLIST *next;
12 }

```

2.3.3 Tafel

```

1 /*
2  * 14 * 79 Zeichen + /n
3  * 1 * 79 Zeichen + /0
4  * >>> als Shared Memory anlegen
5  */
6 char blackboard[15][80];

```

2.4 Abläufe

2.4.1 Programmstart

```

1 Sicherstellen, dass noch kein Server läuft
2 Mutex für Tafelzugriff initialisieren (gesperrt)
3 Mutex für Zugriff auf Client-Liste initialisieren (gesperrt)
4 Initialisierung der Tafel (Shared Memory)
5 Initialisierung der Client-Liste (doppelt verkettete Liste)
6 Initialisierung Semaphore (Zähler) für aktive Clients (** GEEIGNET
   ??? **)
7 Message Queue für Logging initialisieren
8 Initialisiere Trigger für Broadcasting-Agent (** IMPLEMENTIERUNG???)
   (**))
9 Initialisiere Trigger "Tafel archiviert" (Condition Variable >
   pthread)
10 Signal registrieren für "System beenden"
11 Socket für Netzwerkkommunikation öffnen
12
13 Fork: Logger (externes Programm)
14 Fork: Archivierer (externes Programm), wenn Debugmodus mit
   Archivierungsintervall
15 Starte Broadcasting-Agent als Thread
16 Starte Login-Thread

```

```

17| Mutex für Tafelzugriff freigeben
18| Mutex für Clientliste freigeben

```

2.4.2 Signal System beenden

```

1| Mutex-Down: Clientliste
2|   Kill: Login-Thread
3| Mutex-Up: Clientliste
4| Trigger Broadcasting Agent: Clients beenden (quit)
5| Warte auf Semaphore (Clientzähler) == 0 (** GEEIGNET??? **)
6|   Kill: Broadcasting Agent
7| Netzwerksocket schliessen
8| Kill: Archivierer
9| Kill: Logger
10| Message Queue (Logger) löschen
11| Mutex-Down: Tafel
12| Freigabe: Shared Memory (Tafel)

```

2.4.3 Login-Thread

```

1| Warte auf Login von Client
2| Mutex-Down für Zugriff auf Client-Liste
3|   Prüfung: Clientname bereits in Liste?
4|     wenn Ja: Fehlermeldung an Client
5|     Schreibrecht: Nein
6|     Wenn Rolle Dozent: Prüfung, bereits ein Dozent angemeldet?
7|       wenn Ja: ändere Rolle Dozent -> Student
8|       wenn Nein: Schreibrecht zuweisen
9|     Wenn Rolle Tutor: ändere Rolle zu Student
10|    (IP/DNS-Name,) Client-Name, Rolle + Zugriffsrecht in Liste
    eintragen
11|   Semaphore (Clientzähler) Up
12| Mutex-Up für Zugriff auf Client-Liste
13| Starte Client-Thread für neuen Client
14| Trigger Broadcasting-Agent: Update Anzahl Clients

```

2.4.4 Client-Thread

```

1| Rückmeldung an Client: Login erfolgreich
2| Warte auf Befehle von verbundenem Client
3| > quit (Client beenden) bzw. Client schliesst Verbindung
4|   Mutex-Down für Zugriff auf Client-Liste
5|     Client aus Client-Liste austragen
6|     Semaphore (Client-Zähler) Down
7|   Mutex-Up
8|   Trigger Broadcasting-Agent: Sende neue Clientanzahl
9|   Tread beenden

```

```

10 > request (Schreibrechte anfordern)
11     Mutex-Down für Zugriff auf Client-Liste
12         hat Client bereits schreibrecht bzw. ist Dozent?
13         wenn Ja: Fehlermeldung
14         suche Dozent in Clientliste
15         kein Dozent: Fehlermeldung
16     Mutex-Up
17     Anfrage für Schreibrecht an Dozent
18     Warte auf Antwort von Dozent
19     Antwort Nein: Fehlermeldung an anfragenden Benutzer
20     Mutex-Down für Zugriff auf Client-Liste
21         setze alter Benutzer mit Schreibrechten: keine Schreibrechte
22         aktueller Benutzer: Schreibrecht
23     Mutex-Up
24     Statusänderung alter Benutzer: keine Schreibrechte
25     Statusänderung anfragender Benutzer: Schreibrechte
26     Trigger Broadcasting-Agent: Tutor = 1
27 > shutdown
28     Mutex-Down für Zugriff auf Client-Liste
29         Benutzer ist Dozent?
30         wenn Nein: Fehlermeldung
31     Mutex-Up
32     Signal senden: System beenden
33 > release
34     Mutex-Down für Zugriff auf Client-Liste
35         Benutzer ist Tutor?
36         wenn Nein: Fehlermeldung
37         aktueller Benutzer: Schreibrecht > Nein
38         ändere Rolle Tutor -> Student
39         Dozent: Schreibrecht Ja
40     Mutex-Up
41     Trigger Broadcasting-Agent: Tutoren = 0
42     Sende Statusänderung Dozent: Schreibrecht erhalten
43 > acquire
44     Mutex-Down für Zugriff auf Client-Liste
45         aktueller Benutzer ist Dozent?
46         wenn Nein: Fehlermeldung
47         entziehe Tutor Schreibrecht
48         setze Dozent Schreibrecht
49     Mutex-Up
50     Statusänderung vorheriger Tutor: keine Schreibrechte
51     Trigger Broadcasting-Agent: Tutoren = 0
52     Sende Statusänderung Dozent: Schreibrechte
53 > modify
54     Mutex-Down für Zugriff auf Client-Liste
55         Benutzer hat Schreibrechte?
56         wenn Nein: Fehlermeldung
57     Mutex-Up
58     Mutex-Down für Zugriff auf Tafel
59     Änderungen in Shared Memory schreiben
60     Mutex-Up

```

```

61 | Trigger Broadcasting-Agent: Tafeländerung
62 | > clear
63 |   Mutex-Down für Zugriff auf Client-Liste
64 |     Benutzer hat Schreibrechte?
65 |       wenn Nein: Fehlermeldung
66 |   Mutex-Up
67 |   Mutex-Down für Zugriff auf Tafel
68 |   Trigger Archivierer
69 |     *** Im Archivierer ist kein Mutex-Down notwendig. Dies
70 |       erfolgt vor
71 |       *** der Triggerung des Archivierers, um sicherzustellen, das
72 |         in der
73 |         *** Zwischenzeit niemand anders auf die Tafel zugreift.
74 |         *** Der Archivierer macht nach dem Sichern der Tafel einen
75 |           Mutex-Up
76 |   Mutex-Down für Zugriff auf Tafel
77 |   Lösche Tafel
78 |   Mutex Up
79 |   Trigger Broadcasting-Agent: Tafel leer

```

2.4.5 Broadcasting-Agent (Thread)

```

1 | Warte auf Trigger
2 | Wenn Tafeländerung
3 |   Mutex-Down für Zugriff auf Tafel
4 |     Lese Tafelinhalt
5 |   Mutex-Up
6 | Sende Nachricht an alle verbundenen Clients

```

2.5 Logger

```

1 | Öffne Message Queue
2 | Warte auf Messages
3 | Schreibe Zeitstempel + Nachricht in Datei (zeilenweise)

```

2.6 Archivierer

```

1 | Öffne Logfile (schreibbar)
2 | Warte auf Trigger bzw. Ablauf von Timer (Debug-Modus)
3 | Ausgelöst durch Timer?
4 |   wenn Ja: Mutex
5 |     *** Vor dem Auslesen der Tafel ist nur ein Mutex-Down notwendig,
6 |       wenn
7 |       *** der Auslöser für die Archivierung durch den Timer erfolgt
8 |         ist.
9 |       *** Andernfalls ist dies bereits durch den Client-Thread
10 |        geschehen um

```

```
8      *** sicherzustellen , dass der Tafelinhalt erst nach dem
      Archivieren
9      *** gelöscht wird.
10 Tafel auslesen
11 Mutex-Up
12 Zeitstempel + Tafelinhalt in Datei schreiben (blockweise)
```

3 Netzwerk

3.1 Allgemeine Definitionen

3.1.1 Datentypen

Um hohe portablilität zu gewährleisten verwenden wir nur sichere Datentypen. Insbesondere gilt dies für Zahlen. Statt den Datentypen short, int, long verwenden wir (u)int8_t, (u)int16_t,

Außerdem sollten immer Funktionen mit Längenbegrenzung benutzt werden, um Buffer-Overflows zu vermeiden.

3.1.2 Serialisierung

Alle structs werden zum senden serialisiert in ein char-Array.

Der Server deserialisiert diesen String wieder.

```
1 // PSEUDOCODE!
2 struct login_data {
3     char name[50];
4     uint8_t role;
5 }
6
7 // BUFFER
8 char buf[sizeof(login_data)];
9
10 // SERIALISIEREN
11 strncpy(buf, login_data, sizeof(buf));
12
13 // DESERIALISIEREN
14 strncpy(login_data, buf, sizeof(login_data));
```

3.1.3 Benutzerrollen

```
1 enum ROLE {
2     Dozent = 1
3     Tutor = 2
4     Student = 3
5 };
```

3.1.4 Schreib- und Leserecht

Als Datentyp wird uint8_t verwendet.

- Modify 0 = nur lesend
- Modify 1 = schreibzugriff (exklusiv)

3.1.5 Message-Types

```

1 enum MYPES {
2     Login = 1,
3     Quit, /** eigentlich nicht nötig; Client bzw. Server */
4         /** schliesst Verbindung einfach */
5     Request,
6     Shutdown,
7     Release,
8     Aquire,
9     Modify,
10    Clear,
11    Status,
12 };

```

3.2 Kommunikationsablauf

Dieser Abschnitt spezifiziert den Austausch der einzelnen Datenstrukturen über das Netzwerk.

Für die Beschreibung des Aufbaus der einzelnen Pakete: > siehe Abschnitt Datenstrukturen

3.2.1 Login

```

1 Login      >>>
2           <<<      Login

```

3.2.2 Quit

```

1 *** eigentlich nicht nötig; Client bzw. Server schliesst Verbindung
  einfach ***

```

3.2.3 Request

```

1 Request    >>>
2           <<<      Request (an Dozent)
3 Bei Zustimmung des Dozenten die Schreibrechte abzugeben
4 Status     >>>
5           <<<      Status (an anfragenden Client: Schreibrechte)
6           <<<      Status (an alle Anderen: Tutor +1)

```

3.2.4 Shutdown

```

1 Shutdown   >>>      (Server schliesst alle Verbindungen)

```


3.2.5 Release

```

1 Release      >>>
2              <<<      Status (an releasing Client: nur Leserechte)
3              <<<      Status (an Dozent: Schreibrechte)
4              <<<      Status (an alle Anderen: Tutoren -1)

```

3.2.6 Acquire

```

1 Acquire      >>>
2              <<<      Status (an Client der aktuell schreibberechtigt
3                          ist: nur lesend)
4              <<<      Status (an Dozent: Schreibrechte)
5              <<<      Status (an alle Anderen: Tutoren -1)

```

3.2.7 Modify

```

1 Modify       >>>
2              <<<      Modify (Änderungen an alle Clients verteilen)

```

3.2.8 Clear

```

1 Clear        >>>
2              <<<      Clear (an alle Clients: Tafel wurde gelöscht)

```

3.2.9 Status

```

1 *** Ein eigener Status-Befehl existiert nicht. Status-Nachrichten
2 *** dienen als Bestätigung für bestimmte Befehle an den Client
3 *** und als Update für den Client-Status.
4 *** Für die Verwendung: siehe andere Befehle

```

3.3 Datenstrukturen

```

1 #pragma pack(1)
2 struct HEADER {
3     uint16_t size; /* Gesamtgrösse in Bytes */
4     uint8_t type; /* Wert aus M_TYPES */
5 };
6
7
8 /*
9  * Packet: Login

```

```

10 *
11 * Beschreibung:
12 * Login-Versuch von Client bzw. Antwort von Server auf Login von
13 * Client.
14 *
15 * >>> Richtung: Client > Server
16 * + "client_name" MUSS gesetzt und 0-terminiert sein und
17 * + ist auf 25 Zeichen (inkl. 0-Byte) begrenzt
18 * + "role" SOLLTE gesetzt sein, kann aber in jedem Fall vom
19 * + Server geändert werden
20 *
21 * >>> Richtung: Server > Client
22 * + "client_name" KANN gesetzt sein
23 * + wenn vorhanden muss dieser 0-terminiert sein und
24 * + ist auf 25 Zeichen (inkl. 0-Byte) begrenzt
25 * + "role" MUSS gesetzt sein und MUSS vom Client respektiert
26 * + werden
27 * + im Fehlerfall ist role == 0
28 */
29 struct LOGIN {
30     struct NET_HEADER header;
31     char[25] client_name;
32     uint8_t role; /* Wert aus ROLE */
33 };
34
35 /*
36 * Packet: Quit
37 *
38 * Beschreibung:
39 * *** eigentlich nicht nötig; Client bzw. Server
40 * *** schliesst Verbindung einfach
41 *
42 * >>> Richtung: Server > Client
43 * + Keine Nutzdaten/Payload
44 * + Message Type in header ausreichend (kein "Payload Struct")
45 */
46
47 /*
48 * Packet: Request
49 *
50 * Beschreibung:
51 * Client fordert Schreibrechte an bzw. Anfrage an Dozent für
52 * Schreibrechte
53 *
54 * >>> Richtung: Client > Server
55 * + "cid" wird nicht berücksichtigt
56 * + "client_name" KANN gesetzt sein
57 * + "client_name" muss 0-terminiert sein und ist auf 25
58 * + Zeichen

```

```

57 *      begrenzt (inkl. 0-Byte)
58 *
59 *      >>> Richtung: Server > Client
60 *      + "cid" MUSS gesetzt sein
61 *      + "client_name" MUSS gesetzt sein
62 *      + "client_name" MUSS 0-terminiert sein und ist auf 25
        Zeichen
63 *      begrenzt (inkl. 0-Byte)
64 */
65 struct REQUEST {
66     struct NET_HEADER header;
67     uint8_t cid;
68     char client_name[25];
69 };
70
71
72 /*
73 * Packet:  Shutdown
74 *
75 * Beschreibung:
76 * System komplett beenden
77 *
78 *      >>> Richtung: Client > Server
79 *      + Keine Nutzdaten/Payload
80 *      + Message Type in header ausreichend (kein "Payload Struct")
81 */
82
83
84 /*
85 * Packet:  Release
86 *
87 * Beschreibung:
88 * Schreibrechte an Dozent zurückgeben
89 *
90 *      >>> Richtung: Client > Server
91 *      + Keine Nutzdaten/Payload
92 *      + Message Type in header ausreichend (kein "Payload Struct")
93 */
94
95
96 /*
97 * Packet:  Aquire
98 *
99 * Beschreibung:
100 * Aktuellem Benutzer Schreibrechte entziehen
101 *
102 *      >>> Richtung: Client > Server
103 *      + Keine Nutzdaten/Payload
104 *      + Message Type in header ausreichend (kein "Payload Struct")
105 *
106 */

```

```

107 |
108 |
109 | /*
110 |  * Packet: Modify
111 |  *
112 |  * Beschreibung:
113 |  * Änderungen an der Tafel übertragen
114 |  *
115 |  *   >>> Richtung: Client > Server
116 |  *   + "cid" MUSS gesetzt sein
117 |  *   + "board" MUSS gesetzt sein, ist 0-terminiert und max. 1200
      | Zeichen (inklusive 0-Byte)
118 |  *
119 |  *   >>> Richtung: Server > Client
120 |  *   + "cid" wird nicht berücksichtigt
121 |  *   + "board" MUSS gesetzt sein, ist 0-terminiert und max. 1200
      | Zeichen (inklusive 0-Byte)
122 | */
123 | struct MODIFY {
124 |     struct NET_HEADER header;
125 |     uint8_t cid;
126 |     char board[1200];
127 | };
128 |
129 |
130 | /*
131 |  * Packet: Clear
132 |  *
133 |  * Beschreibung:
134 |  * Löschen des Tafelinhalts
135 |  *
136 |  *   >>> Richtung: Client > Server
137 |  *   + Keine Nutzdaten/Payload
138 |  *   + Message Type in header ausreichend (kein "Payload Struct")
139 | */
140 |
141 |
142 | /*
143 |  * Packet: Status
144 |  *
145 |  * Beschreibung:
146 |  * Änderung des Status im Server bzw. Client
147 |  *
148 |  *   Richtung: Client > Server (als Antwort auf 'Request'
149 |  *   + "client_name" KANN gesetzt sein (vom Server nicht berü
      | cksichtigt)
150 |  *   + "cid" MUSS gesetzt sein
151 |  *   + restliche Elemente KÖNNEN gesetzt sein,
152 |  *   aber vom Server nicht weiter berücksichtigt
153 |  *
154 |  *   Richtung: Server > Client

```

```
155 *      + "client_name" MUSS gesetzt sein
156 *      + "role" KANN gesetzt sein
157 *      + "cid" KANN gesetzt sein
158 *      + "write" MUSS entweder 1 = schreibender, oder 0 = lesender
159 *          Zugriff sein
160 *      + "nof_doz" (Anzahl angemeldeter Dozenten) KANN gesetzt sein
161 *          und ist entweder 0 oder 1 (nicht mehr als 1 Dozent möglich
162 *      )
163 *      + "nof_tut" (Anzahl Tutoren) KANN gesetzt sein und ist
164 *          entweder 0 oder 1 (Student ist nur Tutor, wenn er
165 *          schreibrechte hat)
166 *      + "nof_std" (Anzahl angemeldeter Studenten) KANN gesetzt
167 *          sein
168 */
169 struct STATUS {
170     struct NET_HEADER header;
171     char client_name[25];
172     uint8_t role;
173     uint32_t cid;
174     uint8_t write;
175     uint8_t nof_doz;
176     uint8_t nof_tut;
177     uint32_t nof_std;
178 }
179 #pragma pack(0)
```