Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Studiengang Angewandte Informatik
Fakultät Elektrotechnik und Informatik

Systementwurf Gruppe 10

Systemprogrammierung

vorgelegt von:

Frank Klameth
xxxxx
frank.klameth@hs-weingarten.de

Simon Westphahl
20xxx
simon.westphahl@hs-weingarten.de

Michael Wydler
20168
michael.wydler@hs-weingarten.de

27. Oktober 2010

Inhaltsverzeichnis

1 Entwurf der Softwarekomponenten

1.1 Funktionalität Client

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

- -s - Servername oder IP (Servername wird in IP umgewandelt)
- -p - Port[1]
- -b - Benutzername
- -r - Rolle

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

Beispiele:

```
> ./client 127.0.0.1 50100 michael student
> ./client 127.0.0.1 michael dozent
```

1.1.1 Hauptprogramm

```
Start mit Parameter (Server-IP, Port, Username, Rolle)
Socket für Netzwerkkommunikation öffnen
Logindaten (Username und gewünschte Rolle) an Server senden
Login erfolgreich?
    wenn NEIN:
        Fehlermeldung ausgeben
        Kill: Client
    wenn JA:
        Userdaten und -rechte speichern (ID, Name, Rechte)
Mutex für lokalen Tafelzugriff initialisieren (gesperrt)
Initialisiere lokale Tafel
Starte Command-Thread
Starte Live-Agent
Starte GUI
Starte Listener-Thread
Starte Trigger für Live-Agent
    Mutex-Down für lokale Tafel
    Fordert aktuellen Tafelinhalt an
    Mutex-Up
```

1.1.2 Command-Thread

```
> quit (Client beenden)
    Sende Befehl "quit" an den Server
    Mutex-Down für lokale Tafel
    Beende Trigger für Live-Agent
    Kill: Listener-Thread
    Kill: GUI
    Kill: Live-Agent
    Kill: Command-Thread
    Lösche lokale Tafel
    Lösche Mutex für Tafelzugriff
```

[1] optional, wird sonst auf Default-Port (50000) gesetzt

```
> request (Schreibrecht anfordern)
    Ist Client Student?
        Wenn JA:
            Sende Befehl "request" an den Server
    Ist Client Dozent?
        Wenn JA:
            Dialog ob Benutzer schreibrecht bekommen soll
            Sende Antwort an Server
    Schreibrecht erteilt?
        Wenn JA:
            Deaktiviere Button 'Schreibrecht anfordern'
            Aktiviere Button 'Schreibrecht abgeben'
            Schreibrecht auf lokale Tafel gewähren
        Wenn NEIN:
            Hinweis das Anfrage abgeleht wurde.
```

```
> shutdown (System beenden)
    Ist Client Dozent?
        Wenn JA:
            Sende Befehl "shutdown" an den Server
```

```
> release (Schreibrecht abgeben)
    Ist Client Tutor?
        Wenn JA:
            Sende Befehl "release" an den Server
```

```
> acquire (Schreibrecht entziehen)
    Ist Client Dozent?
        Wenn JA:
            Sende Befehl "acquire" an den Server
```

```
> clear (Tafel löschen)
    Hat Client Schreibrechte?
        Wenn JA:
            Sende Befehl "clear" an den Server
```

1.1.3 Live-Agent

```
> modify (Tafel ändern)
    Hat Client schreibrecht?
        Wenn JA:
            Mutex–Down für lokale Tafel
            Ist Tafel voll?
                Wenn JA:
                    Fehlermeldung
                Wenn NEIN:
                    Schreibe Änderung in lokale Tafel
            Mutex–Up
            Trigger für Tafel starten.
    Trigger für Tafel sendet dann die Daten in bestimmten Intervallen.
    Trigger abgelaufen?
        Wenn JA:
            Mutex–Down für lokale Tafel
            Sende Tafel an Server
            Erfolgreiche Sendung?
                Wenn NEIN:
                    Tafel nochmals senden
            Mutex–Up
```

1.1.4 Listener-Thread

```
> board_modified (Tafel-Update)
    Mutex-Down für lokale Tafel
    Tafel aktuallisieren
    Mutex-Up
```

```
> states_changed (Statusänderung)
    GUI-Informationen aktuallisieren
```

```
> my_state_changed (eigene Rechte bekommen/entzogen)
    Schreibrecht erhalten?
        Wenn JA:
            Button "Schreibrecht anfordern" deaktivieren
            Tafel editierbar setzten
    Schreibrecht abgegeben/entzogen?
        Wenn JA:
            Tafel nicht-editierbar setzten
            Button "Schreibrecht anfordern" aktivieren
```

1.1.5 GUI

```
// Tafel wird als GtkTextView gespeichert.
GtkTextBuffer *gtkbuf = gtk_text_view_get_buffer(textview);

GtkTextIter startIter, endIter;
char *mybuf;

gtk_text_buffer_get_start_iter(gtkbuf, &startIter);
gtk_text_buffer_get_end_iter(gtkbuf, &endIter);

// Speichern in char*
mybuf = gtk_text_buffer_get_text(gtkbuf, &startIter, &endIter, FALSE);

// Tafel leeren
gtk_text_buffer_set_text(gtkbuf, "", -1);

// Tafel wieder befüllen
gtk_text_buffer_set_text(gtkbuf, mybuf, -1);
```

1.1.6 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.

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

1.2 Funktionalität Server

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

- -p - Port[2]
- -d - Debugging-Mode (ohne Argument)

Beispiele:

```
> ./server -p 8080 -d
> ./server
```

1.2.1 Hauptprogramm

```
Sicherstellen, dass noch kein Server läuft
Mutex für Tafelzugriff initialisieren (gesperrt)
Mutex für Zugriff auf Client-Liste initialisieren (gesperrt)
Initialisierung der Tafel (Shared Memory)
Initialisierung der Client-Liste (doppelt verkettete Liste)
Initialisierung Semaphore (Zähler) für aktive Clients (*** GEEIGNET??? ***)
Message Queue für Logging initialisieren
Initialisiere Trigger für Broadcasting-Agent (*** IMPLEMENTIERUNG??? ***)
Initialisiere Trigger "Tafel archiviert" (Condition Variable > pthread)
Signal registrieren für "System beenden"
Socket für Netzwerkkommunikation öffnen

Fork: Logger (externes Programm)
Fork: Archivierer (externes Programm), wenn Debugmodus mit Archivierungsintervall
Starte Broadcasting-Agent als Thread
Starte Login-Thread
Mutex für Tafelzugriff freigeben
Mutex für Clientliste freigeben
```

1.2.2 Signal System beenden

```
Mutex-Down: Clientliste
    Kill: Login-Thread
Mutex-Up: Clientliste
Trigger Broadcasting Agent: Clients beenden (quit)
Warte auf Sempahore (Clientzähler) == 0 (*** GEEIGNET??? ***)
    Kill: Broadcasting Agent
Netzwerksocket schliessen
Kill: Archivierer
Kill: Logger
Message Queue (Logger) löschen
Mutex-Down: Tafel
Freigabe: Shared Memory (Tafel)
```

1.2.3 Login-Thread

```
Warte auf Login von Client
Mutex-Down für Zugriff auf Client-Liste
    Prüfung: Clientname bereits in Liste?
        wenn Ja: Fehlermeldung an Client
    Schreibrecht: Nein
    Wenn Rolle Dozent: Prüfung, bereits ein Dozent angemeldet?
```

[2] optional, sonst wird Default-Port (50000) benutzt.

```
        wenn Ja: ändere Rolle Dozent -> Student
        wenn Nein: Schreibrecht zuweisen
    Wenn Rolle Tutor: ändere Rolle zu Student
    (IP/DNS-Name,) Client-Name, Rolle + Zugriffsrecht in Liste eintragen
    Semaphore (Clientzähler) Up
Mutex-Up für Zugriff auf Client-Liste
Starte Client-Thread für neuen Client
Trigger Broadcasting-Agent: Update Anzahl Clients
```

1.2.4 Client-Threads

```
Rückmeldung an Client: Login erfolgreich
Warte auf Befehle von verbundenem Client
> quit (Client beenden) bzw. Client schliesst Verbindung
    Mutex-Down für Zugriff auf Client-Liste
        Client aus Client-Liste austragen
        Semaphore (Client-Zähler) Down
    Mutex-Up
    Trigger Broadcasting-Agent: Sende neue Clientanzahl
    Tread beenden
```

```
> request (Schreibrechte anfordern)
    Mutex-Down für Zugriff auf Client-Liste
        hat Client bereits schreibrecht bzw. ist Dozent?
            wenn Ja: Fehlermeldung
        suche Dozent in Clientliste
            kein Dozent: Fehlermeldung
    Mutex-Up
    Anfrage für Schreibrecht an Dozent
    Warte auf Antwort von Dozent
        Antwort Nein: Fehlermeldung an anfragenden Benutzer
    Mutex-Down für Zugriff auf Client-Liste
        setze alter Benutzer mit Schreibrechten: keine Schreibrechte
        aktueller Benutzer: Schreibrecht
    Mutex-Up
    Statusänderung alter Benutzer: keine Schreibrechte
    Statusänderung anfragender Benutzer: Schreibrechte
    Trigger Broadcasting-Agent: Tutor = 1
```

```
> shutdown
    Mutex-Down für Zugriff auf Client-Liste
        Benuter ist Dozent?
            wenn Nein: Fehlermeldung
    Mutex-Up
    Signal senden: System beenden
```

```
> release
    Mutex-Down für Zugriff auf Client-Liste
        Benutzer ist Tutor?
            wenn Nein: Fehlermeldung
        aktueller Benutzer: Schreibrecht > Nein
        ändere Rolle Tutor -> Student
        Dozent: Schreibrecht Ja
    Mutex-Up
    Trigger Broadcasting-Agent: Tutoren = 0
    Sende Statusänderung Dozent: Schreibrecht erhalten
```

```
> acquire
    Mutex-Down für Zugriff auf Client-Liste
        aktueller Benutzer ist Dozent?
```

```
            wenn Nein: Fehlermeldung
        entziehe Tutor Schreibrecht
        setze Dozent Schreibrecht
    Mutex-Up
    Statusänderung vorheriger Tutor: keine Schreibrechte
    Trigger Broadcasting-Agent: Tutoren = 0
    Sende Statusänderung Dozent: Schreibrechte
```

```
> modify
    Mutex-Down für Zugriff auf Client-Liste
        Benutzer hat Schreibrechte?
            wenn Nein: Fehlermeldung
    Mutex-Up
    Mutex-Down für Zugriff auf Tafel
        Änderungen in Shared Memory schreiben
    Mutex-Up
    Trigger Broadcasting-Agent: Tafeländerung
```

```
> clear
    Mutex-Down für Zugriff auf Client-Liste
        Benutzer hat Schreibrechte?
            wenn Nein: Fehlermeldung
    Mutex-Up
    Mutex-Down für Zugriff auf Tafel
    Trigger Archivierer
        *** Im Archivierer ist kein Mutex-Down notwendig. Dies erfolgt vor
        *** der Triggerung des Archivierers, um sicherzustellen, das in der
        *** Zwischenzeit niemand anders auf die Tafel zugreift.
        *** Der Archivierer macht nach dem Sichern der Tafel einen Mutex-Up
    Mutex-Down für Zugriff auf Tafel
        Lösche Tafel
    Mutex Up
    Trigger Broadcasting-Agent: Tafel leer
```

1.2.5 Broadcasting-Agent

```
Warte auf Trigger
Wenn Tafeländerung
    Mutex-Down für Zugriff auf Tafel
        Lese Tafelinhalt
    Mutex-Up
Sende Nachricht an alle verbundenen Clients
```

1.3 Funktionalität Logger

```
Öffne Message Queue
Warte auf Messages
Schreibe Zeitstempel + Nachricht in Datei (zeilenweise)
```

1.4 Funktionalität Archivierer

```
Öffne Logfile (schreibbar)
Warte auf Trigger bzw. Ablauf von Timer (Debug-Modus)
Ausgelöst durch Timer?
    wenn Ja: Mutex
    *** Vor dem Auslesen der Tafel ist nur ein Mutex-Down notwendig, wenn
    *** der Auslöser für die Archivierung durch den Timer erfolgt ist.
```

```
7       *** Andernfalls ist dies bereits durch den Client-Thread 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)
```

1.5 Synchronisationsprotokoll

siehe Petrinetz.

1.6 Netzwerkschnittstellen

1.6.1 Header

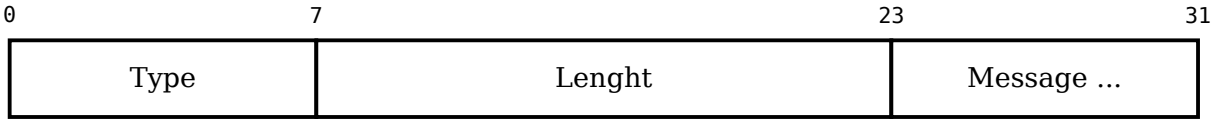

Abbildung 1: Header

Type	uint8_t
Lenght	uint16_t

1.6.2 Login

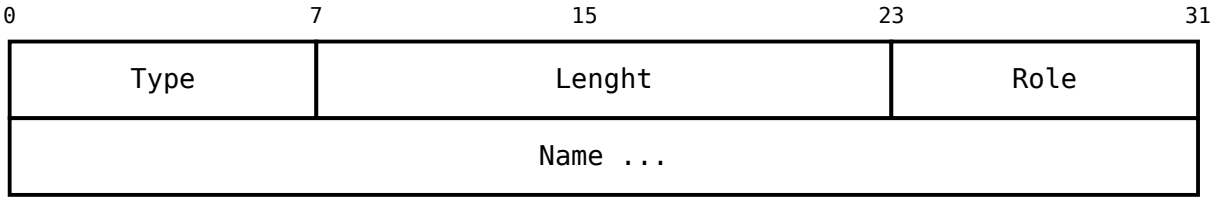

Abbildung 2: Login

1.6.3 Tafelinhalt

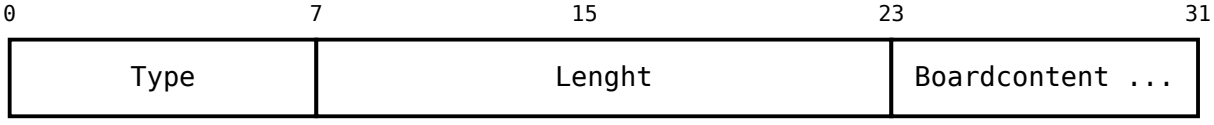

Abbildung 3: Tafelinhalt

1.6.4 Tafel löschen

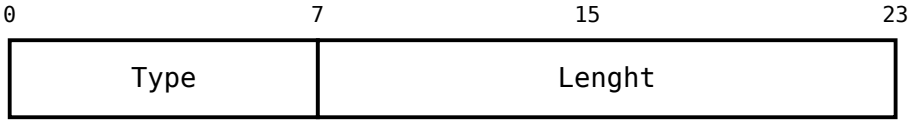

Abbildung 4: Tafel löschen

1.6.5 Schreibrechtanfrage

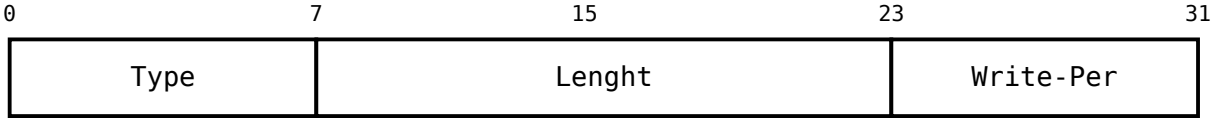

Abbildung 5: Schreibrechtanfrage C -> S

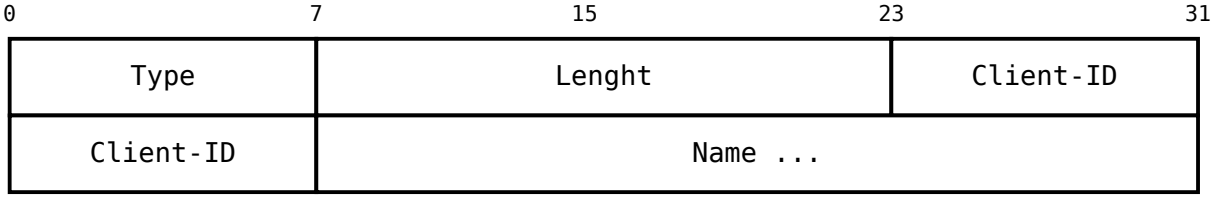

Abbildung 6: Schreibrechtanfrage S -> D

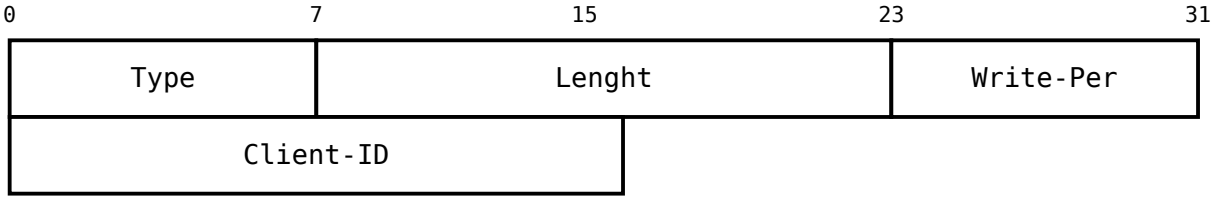

Abbildung 7: Schreibrechtanfrage D -> S

1.6.6 Beenden

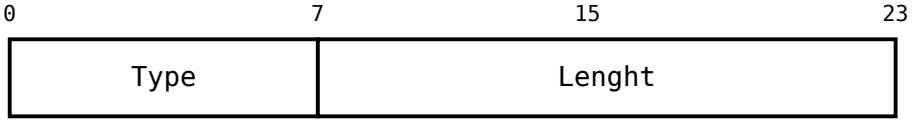

Abbildung 8: Beenden

1.6.7 Status-Nachricht

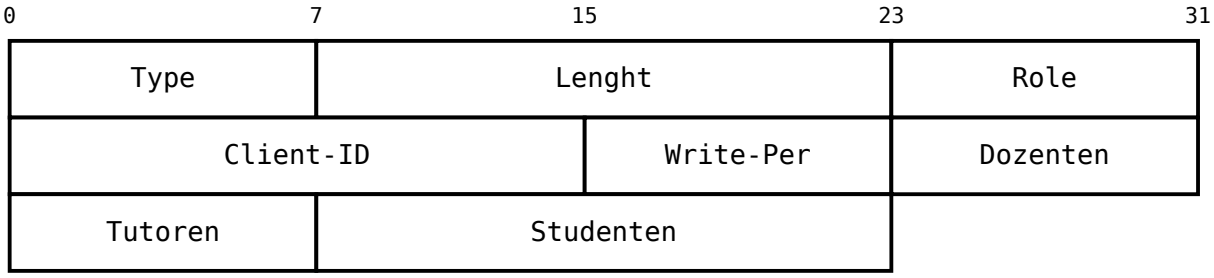

Abbildung 9: Status-Nachricht

1.6.8 Fehlernachricht

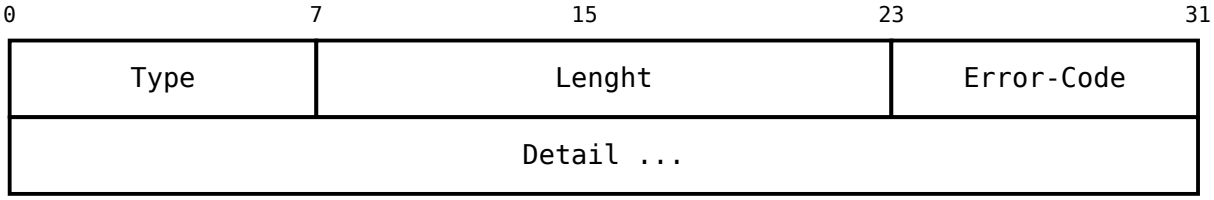

Abbildung 10: Fehlernachticht

2 Client

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

2.1 Module

- Login
- Benutzeroberfläche (GUI)
- Live-Agent
- Listener-Thread
- 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:

- 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`

2.3 Strukturen (intern)

2.3.1 Benutzerrollen

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

2.3.2 Schreib- und Leserecht

Als Datentyp wird `uint8_t` verwendet.

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

2.4 Abläufe

2.4.1 Programmstart

```
Start mit Parameter (Server-IP, Port, Username, Rolle)
Socket für Netzwerkkommunikation öffnen
Logindaten (Username und gewünschte Rolle) an Server senden
Login erfolgreich?
    wenn NEIN:
        Fehlermeldung ausgeben
        Kill: Client
    wenn JA:
        Userdaten und -rechte speichern (ID, Name, Rechte)
Mutex für lokalen Tafelzugriff initialisieren (gesperrt)
Initialisiere lokale Tafel
Starte Command-Thread
Starte Live-Agent
Starte GUI
Starte Listener-Thread
Starte Trigger für Live-Agent
    Mutex-Down für lokale Tafel
    Fordert aktuellen Tafelinhalt an
    Mutex-Up
```

2.4.2 Command-Thread

```
> quit (Client beenden)
    Sende Befehl "quit" an den Server
    Mutex-Down für lokale Tafel
    Beende Trigger für Live-Agent
    Kill: Listener-Thread
    Kill: GUI
    Kill: Live-Agent
    Kill: Command-Thread
    Lösche lokale Tafel
    Lösche Mutex für Tafelzugriff

> request (Schreibrecht anfordern)
    Ist Client Student?
        Wenn JA:
            Sende Befehl "request" an den Server
    Ist Client Dozent?
        Wenn JA:
            Dialog ob Benutzer schreibrecht bekommen soll
            Sende Antwort an Server
    Schreibrecht erteilt?
        Wenn JA:
            Deaktiviere Button 'Schreibrecht anfordern'
            Aktiviere Button 'Schreibrecht abgeben'
            Schreibrecht auf lokale Tafel gewähren
        Wenn NEIN:
            Hinweis das Anfrage abgeleht wurde.

> shutdown (System beenden)
    Ist Client Dozent?
        Wenn JA:
            Sende Befehl "shutdown" an den Server

> release (Schreibrecht abgeben)
    Ist Client Tutor?
        Wenn JA:
            Sende Befehl "release" an den Server

```

```
> acquire (Schreibrecht entziehen)
    Ist Client Dozent?
        Wenn JA:
            Sende Befehl "acquire" an den Server

> clear (Tafel löschen)
    Hat Client Schreibrechte?
        Wenn JA:
            Sende Befehl "clear" an den Server
```

2.4.3 Live-Agent

```
> modify (Tafel ändern)
    Hat Client schreibrecht?
        Wenn JA:
            Mutex-Down für lokale Tafel
            Ist Tafel voll?
                Wenn JA:
                    Fehlermeldung
                Wenn NEIN:
                    Schreibe Änderung in lokale Tafel
            Mutex-Up
            Trigger für Tafel starten.
    Trigger für Tafel sendet dann die Daten in bestimmten Intervallen.
    Trigger abgelaufen?
        Wenn JA:
            Mutex-Down für lokale Tafel
            Sende Tafel an Server
            Erfolgreiche Sendung?
                Wenn NEIN:
                    Tafel nochmals senden
            Mutex-Up
```

2.4.4 GUI

```
// Tafel wird als GtkTextView gespeichert.
GtkTextBuffer *gtkbuf = gtk_text_view_get_buffer(textview);

GtkTextIter startIter, endIter;
char *mybuf;

gtk_text_buffer_get_start_iter(gtkbuf, &startIter);
gtk_text_buffer_get_end_iter(gtkbuf, &endIter);

// Speichern in char*
mybuf = gtk_text_buffer_get_text(gtkbuf, &startIter, &endIter, FALSE);

// Tafel leeren
gtk_text_buffer_set_text(gtkbuf, "", -1);

// Tafel wieder befüllen
gtk_text_buffer_set_text(gtkbuf, mybuf, -1);
```

2.4.5 Listener-Thread

```
Wartet auf Nachrichten vom Server (Broadcasting-Thread)
Aktuallisierung der lokalen Tafel und der Statusinformationen.

> board_modified (Tafel-Update)
```

```
    Mutex-Down für lokale Tafel
    Tafel aktuallisieren
    Mutex-Up

> states_changed (Statusänderung)
    GUI-Informationen aktuallisieren

> my_state_changed (eigene Rechte bekommen/entzogen)
    Schreibrecht erhalten?
        Wenn JA:
            Button "Schreibrecht anfordern" deaktivieren
            Tafel editierbar setzten
    Schreibrecht abgegeben/entzogen?
        Wenn JA:
            Tafel nicht-editierbar setzten
            Button "Schreibrecht anfordern" aktivieren
```

2.5 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.

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

3 Server

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

3.1 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

3.2 Programmstart

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

- Port

Beispiel: `> ./server 8080`

3.3 Strukturen (intern)

3.3.1 Benutzerrollen

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

3.3.2 Clientliste

```
struct CLIENTLIST {
    int sockd;
    char name[25];
    enum ROLE role;
    /*
     * Modify 0 -> nur lesend
     * Modify 1 -> schreibzugriff (exclusiv)
     */
    int modify;
    struct CLIENTLIST *previous;
    struct CLIENTLIST *next;
}
```

3.3.3 Tafel

```
/*
 * 14 * 79 Zeichen + /n
 * 1 * 79 Zeichen + /0
 * >>> als Shared Memory anlegen
 */
char blackboard[15][80];
```

3.4 Abläufe

3.4.1 Programmstart

```
Sicherstellen, dass noch kein Server läuft
Mutex für Tafelzugriff initialisieren (gesperrt)
Mutex für Zugriff auf Client-Liste initialisieren (gesperrt)
Initialisierung der Tafel (Shared Memory)
Initialisierung der Client-Liste (doppelt verkettete Liste)
Initialisierung Semaphore (Zähler) für aktive Clients (*** GEEIGNET??? ***)
Message Queue für Logging initialisieren
Initialisiere Trigger für Broadcasting-Agent (*** IMPLEMENTIERUNG??? ***)
Initialisiere Trigger "Tafel archiviert" (Condition Variable > pthread)
Signal registrieren für "System beenden"
Socket für Netzwerkkommunikation öffnen

Fork: Logger (externes Programm)
Fork: Archivierer (externes Programm), wenn Debugmodus mit Archivierungsintervall
Starte Broadcasting-Agent als Thread
Starte Login-Thread
Mutex für Tafelzugriff freigeben
Mutex für Clientliste freigeben
```

3.4.2 Signal System beenden

```
Mutex-Down: Clientliste
    Kill: Login-Thread
Mutex-Up: Clientliste
Trigger Broadcasting Agent: Clients beenden (quit)
Warte auf Sempahore (Clientzähler) == 0 (*** GEEIGNET??? ***)
    Kill: Broadcasting Agent
Netzwerksocket schliessen
Kill: Archivierer
Kill: Logger
Message Queue (Logger) löschen
Mutex-Down: Tafel
Freigabe: Shared Memory (Tafel)
```

3.4.3 Login-Thread

```
Warte auf Login von Client
Mutex-Down für Zugriff auf Client-Liste
    Prüfung: Clientname bereits in Liste?
        wenn Ja: Fehlermeldung an Client
    Schreibrecht: Nein
    Wenn Rolle Dozent: Prüfung, bereits ein Dozent angemeldet?
        wenn Ja: ändere Rolle Dozent -> Student
        wenn Nein: Schreibrecht zuweisen
    Wenn Rolle Tutor: ändere Rolle zu Student
```

```
    (IP/DNS-Name,) Client-Name, Rolle + Zugriffsrecht in Liste eintragen
    Semaphore (Clientzähler) Up
Mutex-Up für Zugriff auf Client-Liste
Starte Client-Thread für neuen Client
Trigger Broadcasting-Agent: Update Anzahl Clients
```

3.4.4 Client-Thread

```
Rückmeldung an Client: Login erfolgreich
Warte auf Befehle von verbundenem Client
> quit (Client beenden) bzw. Client schliesst Verbindung
    Mutex-Down für Zugriff auf Client-Liste
        Client aus Client-Liste austragen
        Semaphore (Client-Zähler) Down
    Mutex-Up
    Trigger Broadcasting-Agent: Sende neue Clientanzahl
    Tread beenden
> request (Schreibrechte anfordern)
    Mutex-Down für Zugriff auf Client-Liste
        hat Client bereits schreibrecht bzw. ist Dozent?
            wenn Ja: Fehlermeldung
        suche Dozent in Clientliste
            kein Dozent: Fehlermeldung
    Mutex-Up
    Anfrage für Schreibrecht an Dozent
    Warte auf Antwort von Dozent
        Antwort Nein: Fehlermeldung an anfragenden Benutzer
    Mutex-Down für Zugriff auf Client-Liste
        setze alter Benutzer mit Schreibrechten: keine Schreibrechte
        aktueller Benutzer: Schreibrecht
    Mutex-Up
    Statusänderung alter Benutzer: keine Schreibrechte
    Statusänderung anfragender Benutzer: Schreibrechte
    Trigger Broadcasting-Agent: Tutor = 1
> shutdown
    Mutex-Down für Zugriff auf Client-Liste
        Benuter ist Dozent?
            wenn Nein: Fehlermeldung
    Mutex-Up
    Signal senden: System beenden
> release
    Mutex-Down für Zugriff auf Client-Liste
        Benutzer ist Tutor?
            wenn Nein: Fehlermeldung
        aktueller Benutzer: Schreibrecht > Nein
        ändere Rolle Tutor -> Student
        Dozent: Schreibrecht Ja
    Mutex-Up
    Trigger Broadcasting-Agent: Tutoren = 0
    Sende Statusänderung Dozent: Schreibrecht erhalten
> acquire
    Mutex-Down für Zugriff auf Client-Liste
        aktueller Benutzer ist Dozent?
            wenn Nein: Fehlermeldung
        entziehe Tutor Schreibrecht
        setze Dozent Schreibrecht
    Mutex-Up
    Statusänderung vorheriger Tutor: keine Schreibrechte
    Trigger Broadcasting-Agent: Tutoren = 0
    Sende Statusänderung Dozent: Schreibrechte
> modify
    Mutex-Down für Zugriff auf Client-Liste
        Benutzer hat Schreibrechte?
```

```
            wenn Nein: Fehlermeldung
    Mutex-Up
    Mutex-Down für Zugriff auf Tafel
        Änderungen in Shared Memory schreiben
    Mutex-Up
    Trigger Broadcasting-Agent: Tafeländerung
> clear
    Mutex-Down für Zugriff auf Client-Liste
        Benutzer hat Schreibrechte?
            wenn Nein: Fehlermeldung
    Mutex-Up
    Mutex-Down für Zugriff auf Tafel
    Trigger Archivierer
        *** Im Archivierer ist kein Mutex-Down notwendig. Dies erfolgt vor
        *** der Triggerung des Archivierers, um sicherzustellen, das in der
        *** Zwischenzeit niemand anders auf die Tafel zugreift.
        *** Der Archivierer macht nach dem Sichern der Tafel einen Mutex-Up
    Mutex-Down für Zugriff auf Tafel
        Lösche Tafel
    Mutex Up
    Trigger Broadcasting-Agent: Tafel leer
```

3.4.5 Broadcasting-Agent (Thread)

```
Warte auf Trigger
Wenn Tafeländerung
    Mutex-Down für Zugriff auf Tafel
        Lese Tafelinhalt
    Mutex-Up
Sende Nachricht an alle verbundenen Clients
```

3.5 Logger

```
Öffne Message Queue
Warte auf Messages
Schreibe Zeitstempel + Nachricht in Datei (zeilenweise)
```

3.6 Archivierer

```
Öffne Logfile (schreibbar)
Warte auf Trigger bzw. Ablauf von Timer (Debug-Modus)
Ausgelöst durch Timer?
    wenn Ja: Mutex
    *** Vor dem Auslesen der Tafel ist nur ein Mutex-Down notwendig, wenn
    *** der Auslöser für die Archivierung durch den Timer erfolgt ist.
    *** Andernfalls ist dies bereits durch den Client-Thread geschehen um
    *** sicherzustellen, dass der Tafelinhalt erst nach dem Archivieren
    *** gelöscht wird.
Tafel auslesen
Mutex-Up
Zeitstempel + Tafelinhalt in Datei schreiben (blockweise)
```

4 Netzwerk

4.1 Allgemeine Definitionen

4.1.1 Datentypen

Um hohe portabliltät 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.

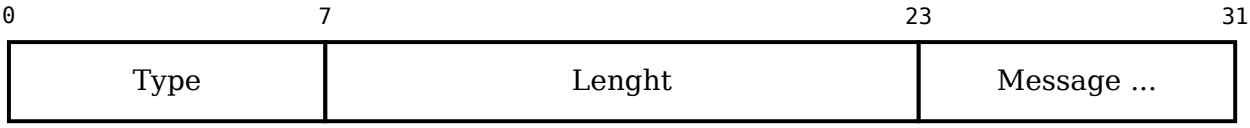

Abbildung 11: Header

4.1.2 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));
```

4.1.3 Benutzerrollen

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

4.1.4 Schreib- und Leserecht

Als Datentyp wird `uint8_t` verwendet.

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

4.1.5 Message-Types

```
enum M_TYPES {
    Login = 1,
    Quit, /*** eigentlich nicht nötig; Client bzw. Server */
          /*** schliesst Verbindung einfach */
    Request,
    Shutdown,
    Release,
    Aquire,
    Modify,
    Clear,
    Status,
};
```

4.2 Kommunikationsablauf

Dieser Abschnitt spezifiziert den Austausch der einzelnen Datenstrukturen über das Netzwerk.
Für die Beschreibung des Aufbaus der einzelnen Pakete: > siehe Abschnitt Datenstrukturen

4.2.1 Login

```
Login       >>>
            <<<     Login
```

4.2.2 Quit

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

4.2.3 Request

```
Request     >>>
            <<<     Request (an Dozent)
Bei Zustimmung des Dozenten die Schreibrechte abzugeben
Status      >>>
            <<<     Status (an anfragenden Client: Schreibrechte)
            <<<     Status (an alle Anderen: Tutor +1)
```

4.2.4 Shutdown

```
Shutdown    >>>     (Server schliesst alle Verbindungen)
```

4.2.5 Release

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

4.2.6 Aquire

```
Aquire      >>>
            <<<     Status (an Client der aktuell schreibberechtig ist: nur lesend)
            <<<     Status (an Dozent: Schreibrechte)
            <<<     Status (an alle Anderen: Tutoren -1)
```

4.2.7 Modify

```
Modify      >>>
            <<<     Modify (Änderungen an alle Clients verteilen)
```

4.2.8 Clear

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

4.2.9 Status

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

4.3 Datenstrukturen

```
pragmapack(1)
struct HEADER {
    uint16_t size; /* Gesamtgrösse in Bytes */
    uint8_t type; /* Wert aus M_TYPES */
};


/*
 * Packet:  Login
 *
 * Beschreibung:
 * Login-Versuch von Client bzw. Antwort von Server auf Login von Client.
 *
 *      >>> Richtung: Client > Server
 *      + "client_name" MUSS gesetzt und 0-terminiert sein und
 *        ist auf 25 Zeichen (inkl. 0-Byte) begrenzt
 *      + "role" SOLLTE gesetzt sein, kann aber in jedem Fall vom
 *        Server geändert werden
 *
 *      >>> Richtung: Server > Client
 *      + "client_name" KANN gesetzt sein
 *      + wenn vorhanden muss dieser 0-terminiert sein und
 *        ist auf 25 Zeichen (inkl. 0-Byte) begrenzt
 *      + "role" MUSS gesetzt sein und MUSS vom Client respektiert werden
 *      + im Fehlerfall ist role == 0
 */
struct LOGIN {
    struct NET_HEADER header;
    char[25] client_name;
    uint8_t role; /* Wert aus ROLE */
```

```
31 };
32
33
34 /*
35  * Packet:  Quit
36  *
37  * Beschreibung:
38  * *** eigentlich nicht nötig; Client bzw. Server
39  * *** schliesst Verbindung einfach
40  *
41  *      >>> Richtung: Server > Client
42  *      + Keine Nutzdaten/Payload
43  *      + Message Type in header ausreichend (kein "Payload Struct")
44  */
45
46
47 /*
48  * Packet:  Request
49  *
50  * Beschreibung:
51  * Client fordert Schreibrechte an bzw. Anfrage an Dozent für Schreibrechte
52  *
53  *      >>> Richtung: Client > Server
54  *      + "cid" wird nicht berücksichtigt
55  *      + "client_name" KANN gesetzt sein
56  *      + "client_name" muss 0-terminiert sein und ist auf 25 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 berücksichtigt)
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  *       + "nof_tut" (Anzahl Tutoren) KANN gesetzt sein und ist
163  *         entweder 0 oder 1 (Student ist nur Tutor, wenn er schreibrechte hat)
164  *       + "nof_std" (Anzahl angemeldeter Studenten) KANN gesetzt sein
165  */
166 struct STATUS {
167     struct NET_HEADER header;
168     char client_name[25];
169     uint8_t role;
170     uint32_t cid;
171     unit8_t write;
172     uint8_t nof_doz;
173     uint8_t nof_tut
174     uint32_t nof_std;
175 }
176
177 pragmapack(0)
```