Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Studiengang Angewandte Informatik
Fakultät Elektrotechnik und Informatik

Systementwurf Gruppe 10

Systemprogrammierung

vorgelegt von:

Frank Klameth
20688
frank.klameth@hs-weingarten.de

Simon Westphahl
20146
simon.westphahl@hs-weingarten.de

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

29. 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.
Die Darstellung der Tafel, die Eingabe von Text auf der Tafel und die Anwahl von Kommandos durch Auswahl der Buttons wird durch unterschiedliche Threads realisiert.

Beispiele:

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

1.1.1 Command-Thread

Der Command-Thread ist für das Senden von Aufträgen an den Client-Thread im Server zuständig. Aufträge werden durch Anwahl eines Buttons getätigt und durch den Command-Thread an den Server geschickt.

1.1.2 Live-Agent

Der Live-Agent wird aktiv, wenn sich der Tafelinhalt ändert, d.h. wenn der Eingabefokus auf der Tafel steht und der Anwender Text eingibt oder verändert. Der Live Agent schickt die Änderungen in der lokalen Tafel an den Client-Thread im Server. Der Client-Thread übernimmt die Änderungen in die virtuelle Tafel im Server und sorgt dafür, dass alle angemeldeten Clients die geänderte Tafel erhalten.

1.1.3 Listener-Thread

Der Listener-Thread hört auf Nachrichten von seinem Client-Thread im Server, wertet die Nachrichten aus und übernimmt die Verarbeitung der Nachricht im Client. Er ist daher auch für die Aktualisierung der lokalen Tafel und der Statusinformationen in der GUI zuständig.

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

1.1.4 GUI

- Die virtuelle Tafel wird im mittleren Bereich in der Farbe Grün mit weißer Schrift dargestellt.
- Status- und Fehlermeldungen für den Client werden in einem Bereich unter der Tafel angezeigt.
- Die Aufträge des Clients an den Server können über Buttons angewählt werden. Die Buttons werden in einem Bereich links neben der Tafel angezeigt. Es sind immer nur die Buttons aktiviert, die vom Client in seiner jeweiligen Rolle auch angewählt werden können

1.1.5 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.

1.2 Funktionalität Server

Der Server darf nur genau einmal gestartet werden. Ein wiederholtes Starten des Servers soll erkannt und zu einer Fehlermeldung führen.
Der Server erzeugt zwei Sohnprozesse, den Logger und den Archivierer. Falls beim Archivierer der Testmodus eingeschaltet werden soll, muss der Server dem Archivierer die Periodendauer übergeben. Diese Periodendauer erhält der Server beim Starten als Kommandoparameter und gibt den eingegeben Wert an den Archivierer weiter.
Für jeden Auftrag, der im Server abgearbeitet wird, erzeugen die beteiligten Threads eine oder mehrere Protokollnachricht(en) und senden diese über eine Message-Queue an den Logger.

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
```

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

1.2.1 Login-Thread

Zur Verwaltung der angemeldeten Clients existiert im Server ein Login-Thread. Jeder Client meldet sich über diesen Login-Thread beim Server an und wird bei Erfolg in einer globalen Datenstruktur „Clientliste“ eingetragen. Danach wird für jeden Client ein Client-Thread im Server gestartet. Alle Clients-Threads im Server haben Zugriff auf die Clientliste.

1.2.2 Client-Thread

Der Server erstellt für jeden Client einen eigenen Client-Thread, der auf Aufträge von seinem Client wartet und diese Aufträge ausführt. Der Client-Thread hat Zugriff auf die virtuelle Tafel. Er kennt folgende Kommandos:

- login
- quit
- request
- shutdown
- release
- acquire
- modify
- clear

1.2.3 Broadcasting-Agent

Der Broadcasting-Agent sendet die Statusnachrichten an alle Clients bei einer Statusänderung. Außerdem ist er für das versenden der Tafel bei einer Änderung an alle angemeldeten Clients zuständig.

1.3 Funktionalität Logger

Der Logger protokolliert alle Protokoll-Meldungen in einer Logdatei.

1.4 Funktionalität Archivierer

Der Archivierer ist ein Prozess zum Speichern von Tafelinhalten und ist als Sohnprozess des Servers zu realisieren.
Der Archivierer hat 2 Funktionen:

- Archivieren des Tafelinhaltes in eine Archivdatei, wenn die virtuelle Tafel gelöscht werden soll
- Speichern der Tafel periodisch in eine Debugdatei, wenn der Testmodus eingestellt ist. Die Zeitdauer soll einstellbar sein. Der Testmodus und die Zeitdauer wird beim Starten des Archivierers eingestellt.

Der Archivierer wird vom Server einmal erzeugt und wartet dann auf einen Trigger. Dieser Trigger wird vom Client-Thread gesetzt, wenn die Tafel gelöscht werden soll oder aber von einem Timer, wenn der Testmodus eigestellt ist.

1.5 Synchronisationsprotokoll

siehe Petrinetz.

1.6 IPC-Schnittstellen

1.6.1 Shared-Memory

Shared-Memory wird an folgenden Stellen benutzt:

- Tafelinhalt

1.6.2 Message-Queues

Message-Queues werden an folgenden Stellen benutzt:

- Logger

1.6.3 Semaphore

Semaphore werden an folgenden Stellen benutzt:

- Tafel (binär)
- Client-Liste (binär)
- Trigger Archivierer

1.6.4 Condition-Variablen

Condition-Variablen werden für den Trigger des Broadcasting-Agent verwendet.

1.7 Netzwerkschnittstellen

1.7.1 Typenübersicht

Nr.	Name	Beschreibung
0	Login	Login-Nachricht
1	Status-Msg	Status-Nachricht
2	Board-Content	Tafel-Inhalt
3	Clear-Board	Tafel löschen
4	Shutdown	System herunterfahren
5	Write-Request	Schreibrecht-Anfrage (Client -> Server)
6	Write-Requested	Schreibrecht-Weiterleitung (Server -> Dozenten)
7	Request-Reply	Schreibrecht-Antwort (Dozenten Client -> Server)
255	Error-Msg	Fehler-Nachricht

1.7.2 Header

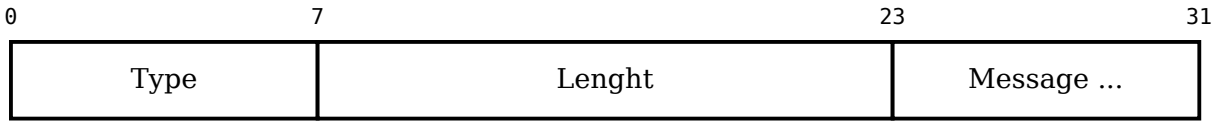

Abbildung 1: Header

Type:	uint8_t	beschreibt den Typ der folgenden Nachricht
Lenght:	uint16_t	gibt die Anzahl der folgenden Bytes an

1.7.3 Login

Der Client sendet eine Login Nachricht
Der Server antwortet mit einer Status Nachricht oder Fehler Nachricht –> Auf beides reagieren!

- Kommt eine ungültige Rolle zurück, dann muss sich der Client beenden
- Die Rolle die zurück kommt ist verpflichtend
- Der Server schickt die Tafel dem Client die Tafel

Nach jedem Login wird eine Status Message an jeden Client gesendet, da sich die Zahl der Studenten oder Dozenten geändert hat (das selbe gilt für einen Logout).

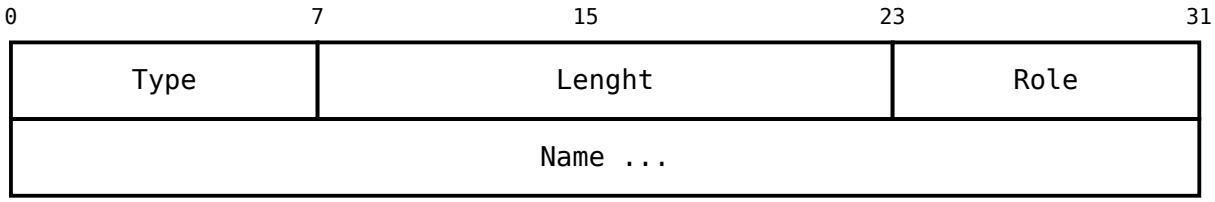

Abbildung 2: Login

Role:	uint8_t	Egal = 0; Student = 1; Dozent = 2; Alles andere Ungültig!
Name:	Char-Array	Nicht null-terminiert (Length - 1) Bytes lang

1.7.4 Tafelinhalt

Der Client tippt, und schickt die ganze neue Tafel, der Server nimmt die neue Tafel auf und verteilt die Tafel mit einer Board-Content Nachricht an alle, die keine Schreibrechte haben.

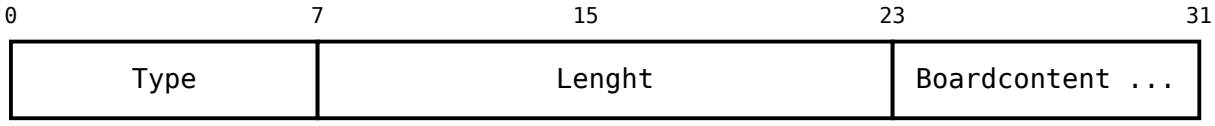

Abbildung 3: Tafelinhalt

Board-Content:	Char-Array	nicht null-terminiert Length Bytes lang

1.7.5 Tafel löschen

Der Client mit Schreibrechte schickt eine Clear-Board Nachricht an den Server. Der Server triggert den Archivierer, löscht den Tafelinhalt und schickt eine leere Tafel an alle Clients.

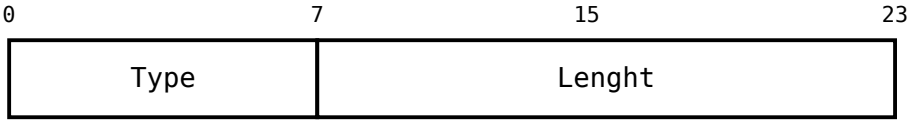

Abbildung 4: Tafel löschen

1.7.6 Schreibrechtanfrage

Der Student wünscht sich Schreibrecht:
Es wird eine Write Request Nachricht an den Server geschickt, dieser leitet eine Write-Requested Nachricht an den Dozenten Client weiter. Der Dozent beantwortet die Anfrage und es wird vom Dozenten Client eine Request-Reply Nachricht an den Server geschickt, dieser verarbeitet diese dann.

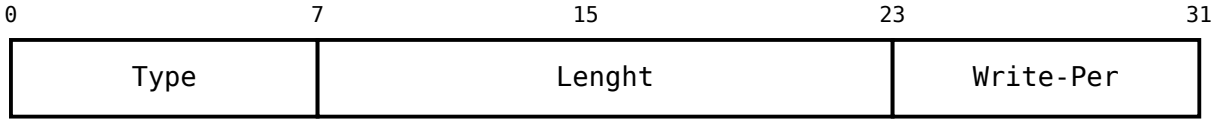

Abbildung 5: Schreibrechtanfrage C -> S

Write-Per: uint8_t | Gewünschtes Schreibrecht

Der Dozent entzieht einem Client das Schreibrecht:
Vom Dozenten Client wird eine Write-Request an den Server geschickt. Der Server entzieht dem Client mit Schreibrecht das Schreibrecht.

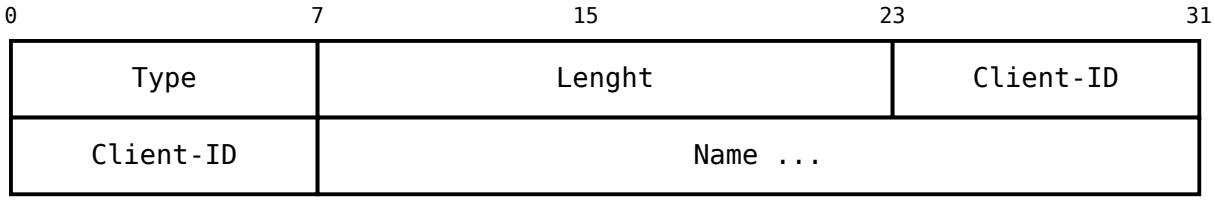

Abbildung 6: Schreibrechtanfrage S -> D

Client-ID: uint16_t | Die ID des Clients der Schreibrecht angefordert hat
Name: Char-Array | nicht null-terminiert, (Lendth - 2) Bytes lang

Der Dozent wünscht sich Schreibrecht:
Vom Dozenten Client wird eine Write-Request an den Server geschickt. Der Server entzieht dem Client mit Schreibrecht das Schreibrecht und setzt dem Dozenten das Schreibrecht.

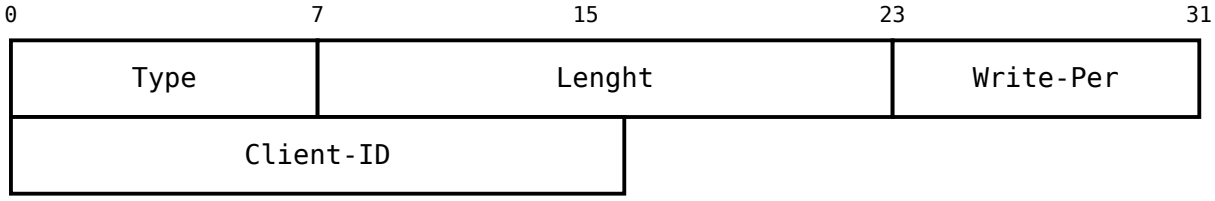

Abbildung 7: Schreibrechtanfrage D -> S

Write-Per: uint8_t | Das neue Schreibrecht
Client-ID: uint16_t | Die Identifizierungsnummer

1.7.7 Beenden

Der Dozenten Client sendet den Befehl das System herunter zu fahren. Der Server sendet eine Error-Msg mit Error-Code 0 (Servernachricht) und einem Shutdown Text an alle Clients

über den Broadcast Agent. Der Client zeigt diese Nachricht an. Der Server fährt sich herunter und schließt dabei alle Verbindungen. Der Client wartet bis die Verbindung schlussendlich getrennt wird.

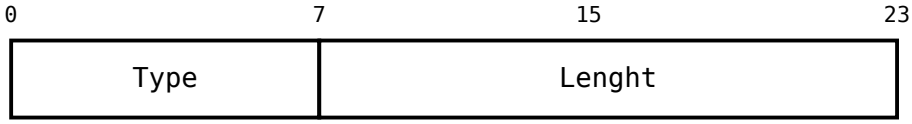

Abbildung 8: Beenden

1.7.8 Status-Nachricht

Nach jeder Rechteänderung wird eine Statusmeldung an alle Clients gesendet.

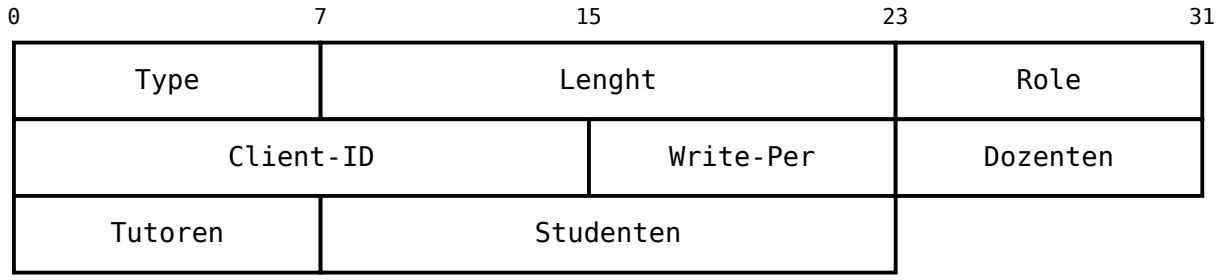

Abbildung 9: Status-Nachricht

Role:	uint8_t	Student = 1; Dozent = 2
Client-ID:	uint16_t	Die Identifizierungsnummer
Write-Per:	uint8_t	ohne Schreibrecht = 0; mit Schreibrecht = 1 –> alles andere ungültig
Dozenten:	uint8_t	Anzahl der Dozenten
Tutoren:	uint8_t	Anzahl der Tutoren
Studenten:	uint16_t	Anzahl der Studenten

1.7.9 Fehlernachricht

Fehlermeldungen können über Error-Msg gesendet werden, sowie Benachrichtigungen an den Client.

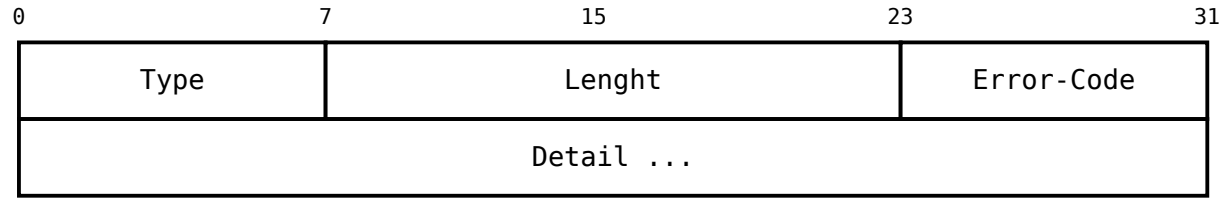

Abbildung 10: Fehlernachtichte

Error-Code:	uint8_t	Grobe Beschreibung des Fehlers
Detail:	Char-Array	nicht null-terminierter (length - 1) Bytes lang; Leer möglich!

2 Client

2.1 Programmabläufe

2.1.1 Hauptprogramm

```
Start mit Parameter (Server-IP, Port, Username, Rolle)
Socket für Netzwerkkommunikation öffnen
Mutex für lokalen Tafelzugriff initialisieren (gesperrt)
Initialisiere lokale Tafel
Starte Listener-Thread
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)
Starte Command-Thread
Starte Live-Agent
Starte GUI
Starte Trigger für Live-Agent
    Mutex-Down für lokale Tafel
    Fordert aktuellen Tafelinhalt an
    Mutex-Up
```

2.1.2 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
```

```
> 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
```

2.1.3 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
```

2.1.4 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
```

2.1.5 Tafel-Trigger

```
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
```

2.2 Funktionshierarchie

2.3 Modulhierarchie

2.4 Quellcode

Der Quellcode ist auf der CD zu finden.

3 Server

3.1 Programmabläufe

3.1.1 Hauptprogramm

```
1  Sicherstellen, dass noch kein Server läuft
2  Mutex für Tafelzugriff initialisieren
3  Mutex für Zugriff auf Client-Liste initialisieren
4  Initialisierung der Tafel (Shared Memory)
5  Initialisierung der Client-Liste (doppelt verkettete Liste)
6  Initialisierung Semaphore (Zähler) für aktive Clients
7  Message Queue für Logging initialisieren
8  Initialisiere Trigger für Broadcasting-Agent (Condition Variable > pthread)
9  Initialisiere Trigger "Tafel archiviert" (Semaphore)
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
```

3.1.2 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 Sempahore (Clientzähler) == 0
6      Kill: Broadcasting Agent
7  Netzwerksocket schliessen
8  Kill: Archivierer
9  Kill: Logger
10 Message Queue (Logger) löschen
11 Freigabe: Shared Memory (Tafel)
```

3.1.3 Login-Thread

```
1 Warte auf Login von Client
2 Starte für jeden Login Client-Thread
```

3.1.4 Client-Thread

```
Mutex-Down für Zugriff auf Client-Liste
    Wenn Rolle Dozent: Prüfung, bereits ein Dozent angemeldet?
        wenn Ja: ändere Rolle Dozent -> Student
        wenn Nein: Schreibrecht zuweisen
    Wenn Rolle egal: ändere Rolle zu Student oder Dozent falls noch nicht
       vorhanden
    Client-ID, Client-Name, Rolle + Zugriffsrecht in Liste eintragen
    Semaphore (Clientzähler) Up
Mutex-Up für Zugriff auf Client-Liste
Bei Fehler: Fehlernachticht an Client und Client-Thread beenden
Trigger Broadcasting-Agent: Statusnachricht an Clients
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
```

```
> write_request (Schreibrecht-Anfrage (Client -> Server))
    Mutex-Down für Zugriff auf Client-Liste
        hat Client bereits schreibrecht?
            wenn Ja: Fehlermeldung
    ist Client Dozent?
      wenn Ja:
          Schreibrecht geht automatisch zum Dozent
          Statusnachricht an Clients
        suche Dozent in Clientliste
            kein Dozent: Fehlermeldung
    Mutex-Up
    Anfrage für Schreibrecht an Dozent
```

```
> write_reply (Schreibrecht-Weiterleitung (Server -> Dozenten))
    Antwort Nein: Fehlermeldung an anfragenden Benutzer
  Antwort Ja:
    Mutex-Down für Zugriff auf Client-Liste
      setze alter Benutzer mit Schreibrechten: keine Schreibrechte
      aktueller Benutzer: Schreibrecht
    Mutex-Up
      Trigger Broadcasting-Agent: Statusnachricht an alle Clients
```

```
> 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
        Hat Client schreibrecht & nicht Dozent?
            wenn Nein: Fehlermeldung
        aktueller Benutzer: Schreibrecht > Nein
        Dozent: Schreibrecht Ja
    Mutex-Up
    Trigger Broadcasting-Agent: Statusnachricht an alle Clients
```

```
> acquire
    Mutex-Down für Zugriff auf Client-Liste
        aktueller Benutzer ist Dozent & kein Schreibrecht?
            wenn Nein: Fehlermeldung
        entziehe Tutor Schreibrecht
        setze Dozent Schreibrecht
    Mutex-Up
    Trigger Broadcasting-Agent: Statusnachricht an alle Clients
```

```
> 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 löschen
```

3.1.5 Broadcasting-Agent

```
1  Warte auf Trigger
2  Wenn Tafeländerung
3      Mutex–Down für Zugriff auf Tafel
4          Lese Tafelinhalt
5      Mutex–Up
6    Sendet Tafelinhalt
7  Wenn Tafel löschen
8    Sende clear an alle Clients
9  Wenn Statusänderung
10   Sende Statusnachricht an alle Clients
```

3.2 Funktionshierarchie

3.3 Modulhierarchie

3.4 Quellcode

Der Quellcode ist auf der CD zu finden.

4 Logger

4.1 Programmablauf

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

4.2 Funktionshierarchie

4.3 Modulhierarchie

4.4 Quellcode

Der Quellcode ist auf der CD zu finden.

5 Archivierer

5.1 Programmablauf

```
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, wenn
6      *** 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)
```

5.2 Funktionshierarchie

5.3 Modulhierarchie

5.4 Quellcode

Der Quellcode ist auf der CD zu finden.

6 Client-Liste

6.1 Datenstrukturen

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

6.2 Exportierte Funktionen

7 Tafel

7.1 Datenstrukturen

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

7.2 Exportierte Funktionen