MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. N. gentschen Felde

Systempraktikum — Projektaufgabe (Teil 1 von 4)

Willkommen in der Gruppenphase des Systempraktikums. Ihre Aufgabe in der Projektphase ist es, einen Client für ein Damespiel in der Programmiersprache `C` zu entwickeln. Die Übungsblätter werden Sie schrittweise zu diesem Ziel führen.

Der Lehrstuhl stellt im Rahmen des Systempraktikums einen Server bereit, mit Hilfe dessen das Brettspiel „Dame“ dargestellt und gespielt werden kann. Der Server ist für die Darstellung und den Spielablauf verantwortlich. Er implementiert keine Spielelogik sondern benötigt Informationen zu den Spielzügen von einer externen Quelle. Die Prüfung der Gültigkeit eines Spielzugs hingegen kann und wird vom Spiel-Server übernommen. Ein Spieler kann entweder ein humaner Spieler oder auch ein Computer-Client sein.

Über die Webseite `http://sysprak.priv.lab.nm.ifi.lmu.de`[1] können Sie auf das Webinterface des Spiel-Servers zugreifen. Erstellen Sie testweise ein neues Spiel und spielen Sie doch zum Einstieg gegen einen Freund oder sich selbst. Sie werden einen Einblick gewinnen wie das Spiel abläuft.

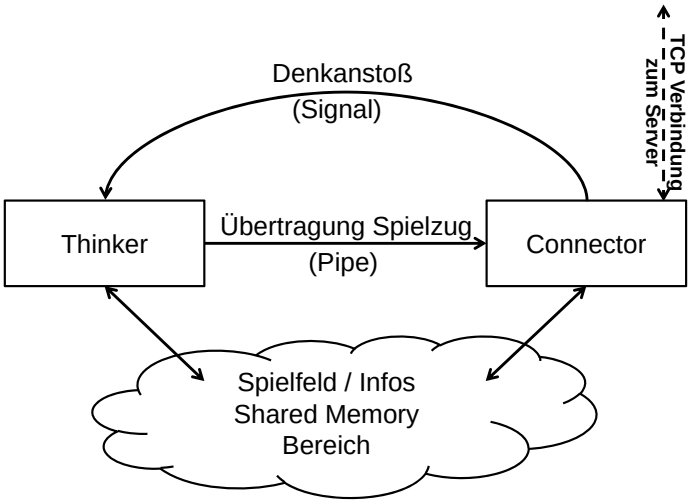

Abbildung 1: Übersicht über den Client

Der von Ihnen zu entwickelnde Client wird im Verlaufe des Systempraktikums schrittweise entwickelt. Maßgeblich besteht ihr Client aus zwei Prozessen und einem geteilten Speicherbereich. Abbildung 1 gibt Ihnen einen groben Überblick über den Aufbau des Clients.

Der *Thinker* und der *Connector* stellen jeweils einen eigenen Prozess dar, wobei der *Connector* die Kommunikation mit dem Spiel-Server übernimmt und der *Thinker*, sobald er durch ein Signal „aufgeweckt“ wird, den nächsten Spielzug berechnet. Das Spielfeld und weitere Informationen zum Spiel befinden sich in einem gemeinsam genutzten Speicherbereich, auf den sowohl der *Connector* als auch der *Thinker* zugreifen. Die Übertragung des Spielzugs zum *Connector* erfolgt über eine Pipe.

[1] Achtung: Der Server ist nur aus dem MWN erreichbar. (Informationen zum MWN: `http://www.lrz.de/services/netz/`)

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. N. gentschen Felde

Übungsaufgaben

Ihre Aufgabe für dieses Übungsblatt ist es die Kommunikation mit dem Spiel-Server der ersten Protokollphase („Prolog“) zu implementieren. In Abbildung 1 ist dies durch den rechten Prozess (*Connector*) dargestellt, der über eine TCP-Verbindung zum Spiel-Server verfügt. Die Protokolldefintion finden Sie ebenfalls in diesem Dokument ab Seite 3.

- Ihr Programm muss einen Kommandozeilenparameter (Hinweis: `argv, argc`) auslesen können, und zwar die Game-ID, welche 13-stellig ist und keine Leerzeichen enthalten darf. Weiterhin müssen (Hinweis: `#define`) die drei Konstanten *HOSTNAME* mit dem Wert `sysprak.priv.lab.nm.ifi.lmu.de`, *PORTNUMBER* mit dem Wert `1357`, und *GAMEKINDNAME*, welche mit dem Wert `Dame` belegt ist (siehe dazu auch die Protokollbeschreibung) definiert werden.
- Anschließend verbinden Sie sich mit dem Spiel-Server (Hinweis: `socket, gethostbyname, connect`) und rufen Sie die von Ihnen zu implementierende Methode *performConnection* auf, welche als Argument den File-Descriptor Ihres Sockets übergeben bekommt.
- Achten Sie bei all Ihren Aufrufen auf eine ordentliche Fehlerbehandlung (Hinweis: `perror`), da Ihr Programm Fehler wie z. B. ein nicht vorhandener Host oder ein nicht laufender Server erkennen und ausgeben sollte.
- Implementieren Sie nun die Methode *performConnection*. Diese Methode sollte sich der besseren Übersichtlichkeit halber in einer separaten Datei *performConnection.c* befinden. Sie können nach Belieben zusätzliche Methoden und Dateien erstellen, wenn Sie Ihnen helfen.
- In dieser Methode sollen Sie die „Prolog“-Phase der Kommunikation implementieren. Geben Sie bei dem *PLAYERS*-Kommando keine Werte mit und lassen Sie sich vom Spiel-Server einen Spieler zuweisen. Geben Sie alle vom Server erhaltenen Informationen wohl formatiert aus, d. h. nicht die Protokollzeile vom Server, sondern z. B.: „Spieler 1 mit der Farbe Weiss ist noch nicht bereit“. Achten Sie hierbei darauf, dass Integer-Werte wie z. B. die 1 auch als solche interpretiert werden.
- Testen Sie Ihren Client ausführlich mit dem Spiel-Server, versuchen Sie auch einem laufenden (offenen) Spiel beizutreten (Stichwort Game-ID), das nicht existiert, um zu sehen, ob Ihr Client die Fehlermeldungen des Servers auch richtig interpretiert und sich entsprechend verhält.
- Zum leichteren Übersetzen Ihres Programms erstellen Sie ein Makefile für Ihr Projekt, welches die einzelnen Quelldateien zu Objektdateien kompiliert und diese zu einer ausführbaren Datei „client“ linkt.

Achtung: Verwenden Sie für all Ihre Übersetzungen die gcc-Schalter `-Wall -Werror`. Dies führt dazu, dass auch Kleinigkeiten als Warnung ausgegeben werden und der Compiler eine Warnung als einen Fehler ansieht und abbricht. Dies dient dem Zweck Ihnen eine spätere, lästige Fehlersuche zu ersparen, die wesentlich aufwändiger ist als die Warnungen frühzeitig zu beseitigen bzw. zu vermeiden.

MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. N. gentschen Felde

Protokolldefinition des Gameservers

Der MNM-Spiel-Server ist wie folgt zu erreichen:

- *Hostname:* `sysprak.priv.lab.nm.ifi.lmu.de`
- *Port:* `1357` (TCP)

Die folgende Protokolldefinition kürzt eine Zeile, welche vom Client an den Server geschickt wird, mit **C:** für *Client* ab. Eine Zeile, welche vom Server an den Client übermittelt wird, wird mit **S:** für *Server* abgekürzt.

Wenn der Server eine Zeile mit einem + als ersten Buchstaben schickt, ist dies eine positive Antwort. Im Folgenden ist nur der positive Verlauf einer Kommunikation angegeben. An jedem Schritt kann eine Negativantwort auftreten, diese ist erkennbar an dem **-** als erstes Zeichen der Zeile. Ein „-“ ist stets gefolgt von einer aussagekräftigen Fehlermeldung. Im Anschluss an die Fehlermeldung wird die Verbindung getrennt.

In doppelten spitzen Klammern eingeschlossene Werte werden obligatorisch durch die ihnen entsprechenden Werte ersetzt, wie z. B. ⟨⟨ *Game-ID* ⟩⟩ durch die 13-stellige Game-ID. Werte, die in doppelten eckigen Klammern eingeschlossen sind, geben optionale Werte an, d. h. sie können auch weggelassen werden.

Es gibt drei Phasen in diesem Protokoll:

1. *Prolog* – hier wird dem Spiel beigetreten und Informationen über das Spiel ausgetauscht
2. *Spielverlauf* – hier wird gewartet bis man an der Reihe ist, bzw. das Spiel beendet wird
3. *Spielzug* – hier übermittelt der Server ein Spielfeld und erwartet einen Spielzug

Wenn nicht innerhalb von im Server festgelegten Zeitgrenzen auf Befehle geantwortet wird, oder eine zu lange „Denkzeit“ benötigt wird (s. u.), schickt der Server:
S: - TIMEOUT ⟨⟨ *Begründung* ⟩⟩

Abbildung 2 gibt eine grob-granulare Übersicht über den Ablauf der drei Protokollphasen.

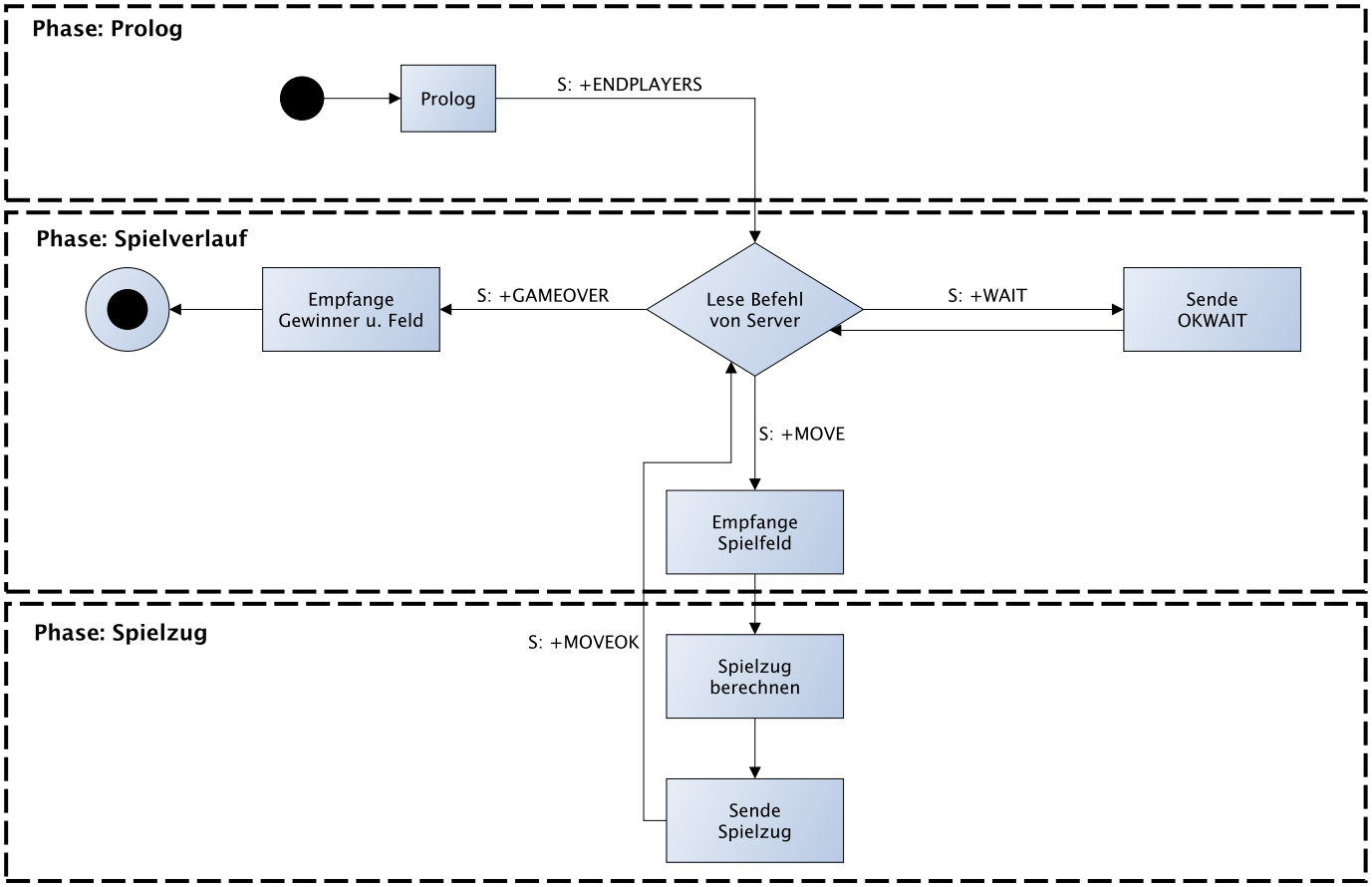

Abbildung 2: Protokollübersicht

1. Protokollphase „Prolog“

⟨⟨ *Verbindungsaufbau* ⟩⟩
S: + MNM Gameserver v1.0 accepting connections
C:[2] VERSION 1.0
S: + Client version accepted - please send Game-ID to join
C: ID ⟨⟨ *Game-ID* ⟩⟩
S:[3] + PLAYING ⟨⟨ *Gamekind-Name* ⟩⟩
S:[4] + ⟨⟨ *Game-Name* ⟩⟩
C:[5] PLAYER *[[Spielernummer]]*
S:[6] + YOU ⟨⟨ *Spielernummer* ⟩⟩ ⟨⟨ *Farbname* ⟩⟩
S:[7] + TOTAL ⟨⟨ *Anzahl der Spieler* ⟩⟩

Nun kommt für jeden der anderen Spieler die Zeile:
S:[8] + ⟨⟨ *Spielernummer* ⟩⟩ ⟨⟨ *Farbname* ⟩⟩ ⟨⟨ *Bereit* ⟩⟩
S:[9] + ENDPLAYERS

2. Protokollphase „Spielverlauf“

In dieser Phase können folgende drei Befehle vom Server empfangen werden:

1. **S:** + GAMEOVER ⟨⟨ *Spielernummer des Gewinners* ⟩⟩ ⟨⟨ *Farbname* ⟩⟩
   S:[10] + FIELD ⟨⟨ *Breite des Spielfelds in Anzahl Felder* ⟩⟩ , ⟨⟨ *Höhe des Spielfelds* ⟩⟩
   S:[11] + QUIT

2. **S:** + WAIT *[[Status]]*
   C:[12] OKWAIT

3. **S:**[13] + MOVE ⟨⟨ *Maximale Zugzeit in Millisekunden* ⟩⟩
   S: + STATUS ⟨⟨ *Statustext* ⟩⟩
   oder **S:**[14] + NOSTATUS
   S: + FIELD ⟨⟨ *Breite des Spielfelds in Anzahl Felder* ⟩⟩ , ⟨⟨ *Höhe des Spielfelds* ⟩⟩
   Die folgende Zeile wird nun für jede Zeile des Spielfeldes geschickt, beginnend bei der obersten Zeile des Spielfelds:
   S: + ⟨⟨ *Y* ⟩⟩ ⟨⟨ $Stein_{1Y}$ ⟩⟩ ⟨⟨ $Stein_{2Y}$ ⟩⟩ ... ⟨⟨ $Stein_{X_{max}Y}$ ⟩⟩

[2] Die Protokollversionen sind dann kompatibel, wenn die Major-Version von Client und Server identisch ist, d. h. 1.0 muss mit 1.3 kompatibel sein, nicht aber mit 2.0!
[3] Da der MNM-Gameserver eine generische Implementation ist, und nicht nur Dame spielen kann, wird hier ausgegeben, um welches Spiel es sich handelt. Der Client sollte dies verifizieren. Für dieses Praktikum wird als Gamekind-Name *Dame* ausgegeben.
[4] Jedes Spiel hat einen eindeutigen Namen der beim Eröffnen des Spiels festgelegt werden kann, wie z. B. Spiel zwischen Sepp und Franz.
[5] Wenn man fest einen Spieler mit einer bestimmten Nummer übernehmen möchte, kann dies hier angegeben werden. Ansonsten bekommt man einen freien Computerspieler vom Server zugeteilt.
[6] Mit dieser Zeile übermittelt der Server, welcher Spieler (welche Nummer, z. B. *2*) gespielt wird und welche Farbe (z. B. *schwarz*)
[7] Anzahl der Spieler wird bei einem Damespiel immer 2 sein.
[8] Spielernummer und Farbname wie oben, Bereit ist entweder 1 oder 0 und gibt an, ob der betreffende Spieler sich schon angemeldet hat.
[9] Mit ENDPLAYERS wird die Aufzählung der anderen Spieler abgeschlossen. Ab jetzt befindet sich das Protokoll in der Phase Spielverlauf
[10] Nach *FIELD* folgt die Ausgabe des gewinnenden Spielfeldes - siehe unten für Details
[11] An dieser Stelle beendet der Server die Verbindung.
[12] Diese *WAIT*-Befehle müssen zu jeder Zeit in dieser Phase mit einem *OKWAIT* quittiert werden sonst beendet der Server die Verbindung.
[13] Der *MOVE*-Befehl fordert zum Zug auf. Innerhalb der gegebenen Zeit erwartet der Server die Antwort. Bitte berücksichtigen Sie bei Ihrer Implementierung die Latenzzeiten der Verbindung.
[14] Als Statustext kann z. B. übermittelt werden: „Schwarz hat einen Spielstein verloren“. Wenn kein Statustext vorliegt wird *NOSTATUS* übermittelt

Im Damespiel kann der Wert für $\langle\langle\ Stein_{XY}\ \rangle\rangle$ folgende Werte annehmen:

- 0: Leeres weißes Feld
- 1: Leeres schwarzes Feld
- 3: Schwarzes Feld mit schwarzem Stein
- 5: Schwarzes Feld mit weißem Stein
- 7: Schwarzes Feld mit schwarzer Dame
- 9: Schwarzes Feld mit weißer Dame

D.h. die erste übermittelte Zeile könnte lauten: *+ 8 0 3 0 3 0 3 0 3*. Diese Zeile zeigt die Anfangsbelegung der obersten Reihe des Spielfelds. Die letzte übermittelte Zeile könnte lauten: *+ 1 5 0 5 0 5 0 5 0*. Diese Zeile zeigt die Anfangsbelegung der untersten Reihe des Spielfelds.

S:[15] + ENDFIELD
C: THINKING
S:[16] + OKTHINK

3. Protokollphase „Spielzug“

C:[17] PLAY $\langle\langle\ Quellfeld\ \rangle\rangle : \langle\langle\ Zielfeld\ \rangle\rangle$
S:[18] + MOVEOK

[15] Die Zeile *ENDFIELD* schließt die Übermittlung des Feldes ab

[16] Der Client muss direkt nach der Übermittlung des Spielfeldes *THINKING* schicken und hat hierfür wenig Zeit! Nach der Serverantwort *OKTHINK* befinden wir uns in der Protokollphase Spielzug

[17] Z. B. *PLAY A7:B6*. Es können auch mehrere Spielzüge hintereinander übermittelt werden, wenn im ersten ein Schlag getätigt wird. Hierfür wird dann z. B. *PLAY A7:C5;C5:D4* übermittelt. Wenn kein Schlag getätigt wird und zwei Züge übermittelt werden, wird nur der erste interpretiert. Wenn ein Schlag getätigt wird, aber nur ein Zug übermittelt wird, ist der Spieler gleich nach seinem Zug erneut an der Reihe und wird in der Protokollphase Spielverlauf mittels eines *MOVE*-Befehls ganz regulär zu einem nächsten Spielzug aufgefordert.

[18] Nach der *MOVEOK*-Quittung befinden wir uns wieder in der Protokollphase Spielverlauf und warten auf eines der dort validen drei Befehle. Sollte der übermittelte Zug nicht gültig sein, wird eine entsprechende Fehlermeldung übermittelt und die Verbindung getrennt.