

Ludwig-Maximilians-Universität München

Prof. Dr. D. Kranzlmüller
Dr. Karl Furlinger
Jan Schmidt
Amir Kabouteh

Systempraktikum — Projektaufgabe (Teil 3 von 4)

Ihr Client implementiert bis jetzt nur die Protokollphase des Prologs, besteht aber bereits aus zwei Prozessen, die über einen gemeinsamen Speicherbereich (SHM) verfügen, und legt dort die wichtigsten Spielstand- sowie Mitspielerinformationen ab. In diesem Übungsblatt sollen die folgenden drei Teilbereiche hinzugefügt werden:

1. Implementierung des vollständigen Protokolls
2. Anstoß des *Thinkers* bei Übermittlung des Spielstandes, Ausgabe des Spielstandes
3. Übermittlung eines (fiktiven) Spielzugs vom *Thinker* zum *Connector*

Ziel ist, dass die gesamte Interprozesskommunikation soweit fertig ist, so dass Sie sich im kommenden und letzten Übungsblatt komplett auf Ihre Spieltaktik konzentrieren können.

1 Implementierung des vollständigen Protokolls

Bis jetzt haben Sie die Protokollphase des Prologs implementiert. Wie Sie in der Protokollbeschreibung sehen können, gibt es noch zwei weitere Phasen – die des Spielverlaufs und die des Spielzugs. Implementieren Sie zunächst die Phase des Spielverlaufs. Diese Phase muss drei mögliche Arten des Spielverlaufs abbilden, d. h. Ihr Client muss auf drei verschiedene Mitteilungen des Gameservers reagieren:

- **GAMEOVER** – Der Gameserver schickt die **GAMEOVER** Mitteilung, wenn das Spiel beendet ist.
- **WAIT** – Der Gameserver schickt periodisch **WAIT** Mitteilungen, auf die Ihr Client reagieren muss.
- **MOVE** – Wenn Sie am Zug sind, schickt der Gameserver Ihnen ein **MOVE**-Kommando.

Die genaue Protokolldefinition und die Kommunikation, die auf die jeweiligen Kommandos folgt, entnehmen Sie der Protokollbeschreibung. Nachdem einer der oben genannten Befehle geschickt und der dazu gehörige Protokollblock abgearbeitet wurde, befinden Sie sich wieder am Anfang der Protokollphase „Spielverlauf“, d. h. es können nun wieder die drei o. g. Mitteilungen vom Gameserver empfangen werden. Einen grafischen Überblick hierzu finden Sie ebenfalls in der Protokolldefinition.

Wie bereits erwähnt wollen wir einen flexiblen Client implementieren, der allein durch Änderung der (in der nächsten Projektaufgabe zu implementierenden) künstlichen Intelligenz auf ähnliche Spiele adaptiert werden kann (z. B. *Ataxx*). Dazu wird der Spielstand, wie die Informationen der Partie auch, in einem SHM-Bereich abgelegt, so dass der *Thinker* darauf zugreifen kann. Bei der ersten Übermittlung des Spielstandes muss ein weiterer SHM-Bereich reserviert werden, da erst dann die Größe des benötigten Speicherbereichs zum Speichern der Spielstände endgültig feststeht. Beachten Sie, dass Sie auch dieses SHM-Segment beim Beenden Ihres Clients wieder freigeben!

Nach dem **MOVE**-Kommando gelangen wir in die (sehr kurze) Protokollphase des Spielzugs und, sobald das **MOVEOK**-Kommando vom Gameserver empfangen wird, wieder in die Protokollphase des Spielverlaufs. Hier befinden wir uns also in einer Art Endlosschleife, aus der es nur zwei Auswege gibt: Entweder sendet der Gameserver ein **GAMEOVER** mit anschließendem **QUIT** sobald das Partieende erreicht ist oder der Gameserver sendet sofort ein **QUIT** nach einem Protokollfehler.

2 Anstoß des Thinkers und Ausgabe des Spielstandes

Nachdem der Gameserver den Spielstand übertragen hat, muss der Client sofort das `THINKING`-Kommando schicken. Hierfür bleibt nicht sehr viel Zeit, da vermieden werden soll, dass Teile des Nachdenkens hier ausgelagert werden können. Daher sollte Ihr Client sofort `THINKING` schicken und anschließend dem *Thinker* Bescheid geben, dass nun der Spielstand vorliegt, zu welchem ein Spielzug berechnet werden soll. Dieses Bescheidgeben erfolgt durch UNIX-Signale. Der *Connector* schickt dem *Thinker* ein `SIGUSR1` (Hinweis: `kill`). Der *Thinker* implementiert bis jetzt nur eine Warteschleife, die den Prozess so lange aktiv hält wie der *Connector* auch aktiv ist. Bevor der *Thinker* also nun in diese Warteschleife einsteigt muss ein Signal-Handler eingerichtet werden (Hinweis: `signal`), welcher beim Empfangen eines `SIGUSR1` eine Methode ausführt, die die Spielinformationen aus dem SHM liest und einen (gültigen) Spielzug berechnet. Diese Methode sei im Folgenden `think()`-Methode genannt. Zusammenfassend schickt also der *Connector* dem *Thinker* das Signal `SIGUSR1`, sobald dem *Connector* die Spielinformationen vom Gameserver übermittelt wurden. Der *Thinker* ruft daraufhin die `think()`-Methode auf.

Um zu vermeiden, dass jemand anders als der *Connector* dem *Thinker* ein `SIGUSR1` schickt ¹, sollten Sie ein Flag in Ihre SHM-Struktur, die die Spielinformation hält, integrieren, das nur gesetzt ist, wenn der *Thinker* auch wirklich einen neuen Zug liefern soll. Sobald der *Thinker* dieses Flag ausgewertet hat, soll dieser es auch gleich wieder zurücksetzen. Somit fängt der *Thinker* nur mit der Berechnung des neuen Spielzugs an, wenn zum einen das Flag im SHM gesetzt ist und zum anderen das Signal `SIGUSR1` empfangen wird.

In diesem Teil der Projektaufgabe sollen Sie noch kein Spielverhalten bzw. keine Spielintelligenz implementieren. Wenn also die `think()`-Methode aufgerufen wird, soll Ihr Programm nur den SHM-Bereich lesen und den Spielstand in ASCII ausgeben ², welches beispielsweise so aussehen könnte:

```
  A B C D E F G H
+-----+
1| . . . . . |1
2| . . . . . |2
3| . 0 0 . # . . |3
4| # 0 # 0 0 0 0 . |4
5| 0 # # # # # . . |5
6| 0 0 0 0 0 0 . . |6
7| . . # 0 0 . . . |7
8| . . # . . . . |8
+-----+
  A B C D E F G H
```

Dadurch, dass Ihre `think()`-Methode nun den Spielstand ausgibt, sind Sie sich sicher, dass Sie den Spielstand richtig in das SHM-Segment schreiben können und richtig interpretiert auch wieder auslesen können.

3 Übermittlung eines Spielzugs vom Thinker zum Connector

Der letzte Schritt der Interprozesskommunikation besteht darin den vom *Thinker* berechneten Spielzug zum *Connector* zu schicken, so dass dieser den Zug an den Gameserver weiterleiten kann. Dies soll mittels einer *unnamed Pipe* geschehen. Der *Thinker* schreibt den berechneten Spielzug in die Pipe hinein und der *Connector* liest ihn aus der Pipe aus. Diese Pipe muss erstellt werden, bevor sich der Client-Prozess mittels `fork()` in den *Connector*- und den *Thinker*-Prozess aufteilt (Hinweis: `pipe()`). Die Pipe ist unidirektional und besitzt eine Schreibseite und eine Leseseite. Als erste Handlung, nachdem Sie den `fork()` durchgeführt haben, sollten Sie in den beiden Prozessen die Seite der Pipe schließen, die Sie nicht mehr benötigen – beim *Thinker*-Prozess die Leseseite, beim *Connector*-Prozess die Schreibseite.

Nachdem in diesem Übungsblatt noch kein echter Spielzug anhand des Spielstandes berechnet wird, machen Sie einen Spielzug auf die Koordinate `E6` als heller und auf `E3` als dunkler Spieler. Den Spielzug schreiben Sie in der oben implementierten `think()`-Methode in die Pipe. Dies ist jeweils ein valider Spielzug für eine neue Partie mit den Standardeinstellungen, mit dem Sie die Kommunikation mit dem Gameserver testen können.

In der `performConnection()`-Methode im *Connector*-Prozess haben Sie nun zwei Stellen gleichzeitig zu überwachen. Zum einen kann es passieren, dass der Gameserver dem Client einen Timeout schickt (eine Negativmeldung), zum

¹Sie können dem Prozess auch manuell über den Befehl `kill` auf der Kommandozeile ein `SIGUSR1` schicken: `kill -SIGUSR1 <<PID>>`

²Vergleichen Sie bitte auch die Protokolldefinition. Hier finden Sie insbesondere auch eine Beschreibung zur Syntax und Semantik eines vom Gameserver übertragenen Spielstandes.

anderen kann es passieren, dass der *Thinker*-Prozess seinen Spielzug fertig berechnet hat und diesen in die Pipe schreibt. Sie müssen also auf beiden Datei-Deskriptoren (dem des Sockets, also der Gameserver-Verbindung, und dem der Pipe) überprüfen, ob zu lesende Daten anstehen (Hinweis: `epoll()` oder `select()`). Anschließend holen Sie die gelesenen Daten ab und verarbeiten entweder den Gameserver-Fehler, indem Sie den Client terminieren, oder Sie schicken den von der Pipe gelesenen Spielzug unter Beachtung der Protokollvorschriften zum Gameserver.