MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

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

---

**Systempraktikum — Projektaufgabe (Teil 3 von 4)**

---

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

1. Implementierung des vollständigen Protokolls
2. Anstoß des *Thinkers* bei Übermittlung des Spielfelds, Ausgabe des Spielfelds
3. Übermittlung eines Spielzugs (ein fiktiver Spielzug) 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 Wege 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 (vgl. erste Projektaufgabe). 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 im Aufgabenblatt zur ersten Projektaufgabe!

Wie bereits erwähnt wollen wir einen flexiblen Client implementieren, der allein durch Änderung der (in der nächsten Projektaufgabe zu implementierenden) künstlichen Intelligenz auf ein anderes Spiel adaptiert werden kann. Wir möchten daher auch die Spielfeldgröße dynamisch gestalten, d. h. evtl. auch auf einem größeren oder kleineren Spielfeld Quarto spielen können. Das Spielfeld wird, wie die anderen Spielinformationen auch, in einem SHM-Bereich abgelegt, so dass der *Thinker* darauf zugreifen kann. Wenn das Spielfeld vom Gameserver übermittelt wird, gibt das `FIELD`-Kommando als Argumente die Höhe und Breite des Spielfelds mit. Bei der ersten Übermittlung des Spielfelds muss an dieser Stelle der SHM-Bereich reserviert werden. Reservieren Sie also hier einen weiteren SHM-Bereich, in dem das Spielfeld abgelegt wird. Als Größe verwenden Sie die Anzahl der Felder multipliziert mit der Länge eines `int`. Beachten Sie, dass Sie auch dieses SHM-Segment beim Beenden Ihres Clients wieder freigeben!

Nach dem `MOVE`-Block 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 – zum einen das vom Gameserver empfangene `QUIT` im `GAMEOVER`-Block, zum anderen eine Negativantwort des Gameservers, die auch in jedem Fall die Verbindung beendet.

## 2. Anstoß des Thinkers und Ausgabe des Spielfelds

Nachdem der Gameserver das Spielfeld übertragen hat und mit dem `ENDFIELD`-Kommando abschließt, muss der Client sofort `THINKING` 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 ein Spielfeld vorliegt, zu dem 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 aufgesetzt werden (Hinweis: `signal`), welcher bei Empfangen eines `SIGUSR1` eine Methode ausführt, welche die zu setzende Spielsteinnummer und das Spielfeld aus dem SHM ließt und den Spielzug berechnet. Diese Methode sei im Folgenden `think()`-Methode genannt. Zusammenfassend schickt also der *Connector* dem *Thinker* das Signal `SIGUSR1`, sobald dem *Connector* das Spielfeld vom Gameserver übermittelt wurde. 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 Code nur den SHM-Bereich des Spielfelds lesen und das Spielfeld in ASCII sowie den die Nummer des nächsten zu setzenden Steins ausgeben[2], welches beispielsweise so aussehen könnte:

```
Next: 0001

    A     B     C     D
  +----------------------- +
4 | 1001  0111  ****  1000 | 4
  |                        |
3 | 0011  ****  0010  **** | 3
  |                        |
2 | 1111  ****  0110  0100 | 2
  |                        |
1 | ****  1011  ****  1010 | 1
  +----------------------- +
    A     B     C     D
```

Da es in Quarto auf die vier binären Eigenschaften eines Spielsteins (hoch/flach, breit/schmal[3], hol/solide und rund/eckig) ankommt, sind die Spielsteine binär dargestellt.

Dadurch, dass Ihre `think()`-Methode nun das Spielfeld ausgibt, sind Sie sich sicher, dass Sie das Spielfeld 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* ließt 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 – bei dem *Thinker*-Prozess die Leseseite und bei dem *Connector*-Prozess die Schreibseite.

[1] Sie können dem Prozess auch manuell über den Befehl `kill` auf der Kommandozeile ein `SIGUSR1` schicken: `kill -SIGUSR1` ⟨⟨ *PID* ⟩⟩

[2] Vergleichen Sie bitte auch die Protokolldefinition auf dem ersten Projektaufgabenblatt. Hier finden Sie insbesondere auch eine Beschreibung zur Syntax und Semantik eines vom Gameserver übertragenen Spielfelds.

[3] Im Orginal: weiß/schwarz

Nachdem in diesem Übungsblatt noch kein echter Spielzug anhand des Spielfelds berechnet wird, machen Sie einen Spielzug auf die Koordinate `A1` und dem als nächstes zu setzenden Stein, der sich aus der aktuell zu setzenden Spielsteinnummer plus eins modulo 16 berechnet. Den Spielzug schreiben Sie in der oben implementierten `think()`-Methode in die Pipe. Dies ist jeweils ein valider Spielzug für ein neues Spielfeld, 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 anderen kann es passieren, dass der *Thinker*-Prozess seinen Spielzug fertig berechnet hat und diesen in die Pipe schreibt. Sie müssen also auf beiden Filedeskriptoren (dem des Sockets, also der Gameserver-Verbindung, und auf dem der Pipe) überprüfen, ob zu lesende Daten anstehen (Hinweis: `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.