MNM
TEAM
Munich Network Management Team

Ludwig-Maximilians-Universität München

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

Systempraktikum — Projektaufgabe (Teil 2 von 4)

Nach dem Abschluss des ersten Meilensteins implementiert Ihr Client mittlerweile die Protokollphase des Prologs. In diesem Übungsblatt sollen nun die folgenden drei Teilbereiche hinzugefügt werden:

1. Parametrisierbare Konfiguration über eine Konfigurationsdatei
2. Aufspaltung des Programmablaufs in zwei Prozesse – den *Thinker* und den *Connector*
3. Implementierung eines *Shared-Memory*-Bereichs für die Kommunikation der beiden Prozesse

1 Parametrisierbare Konfiguration

Der Client nimmt aktuell eine Kommandozeilenoption entgegen, die *Game-ID*. Implementieren Sie zunächst, dass Ihr Client als weiteres optionales Argument neben der Game-ID auch noch einen Dateinamen einer Konfigurationsdatei annimmt. Ist dieses Argument nicht angegeben, soll Ihr Client die Standardkonfigurationsdatei mit dem Namen `client.conf` verwenden.

Ihre Aufgabe ist es nun eine Struktur (`struct`) anzulegen, in die die ausgelesenen Konfigurationsparameter abgelegt werden. Im ersten Schritt sollen Sie mindestens die folgenden Konfigurationsparameter auslesen und im korrekten Datenformat in die Struktur ablegen können [1]:

- Hostname des Gameservers
- Portnummer des Gameservers
- Art des Spieles (hier: Reversi)

Hinweis: Legen Sie z. B. die Portnummer des Gameservers nicht als `char[]` ab, sondern in der Form, wie sie von `htons()` verwendet werden kann.

Die Form der Konfigurationsdatei ist ein Parameter pro Zeile in der Form

```
parameterName = parameterWert
```

Dabei soll es keine Rolle spielen, ob und wie viele Leerzeichen vor dem Parameternamen bzw. um das Gleichheitszeichen existieren.

Der Code, der die Konfigurationsdatei einließt und die Werte in Ihr `struct` schreibt, soll leicht auf weitere Parameter erweiterbar sein. Achten Sie der Übersichtlichkeit halber darauf die für das Interpretieren der Konfigurationsdatei erforderlichen Methoden in einer separaten Datei (z. B. `config.c`) auszulagern.

[1] Sie können so viel Sie möchten über die Konfigurationsdatei parametrisierbar machen.

2 Aufspaltung in zwei Prozesse

Wie Sie wissen, ist für das gesamte Projekt vorgesehen, dass Sie zwei Prozesse verwalten. Auf der einen Seite ist dies der *Thinker*, der anhand des übermittelten und im Shared-Memory (siehe nächste Teilaufgabe) abgelegten Spielfelds den nächsten Spielzug berechnet. Auf der anderen Seite benötigen Sie den *Connector*, welcher sich um die TCP-Verbindung zum Gameserver kümmert. Diese Aufteilung ist sinnvoll, da der Client stets auf Befehle vom Gameserver reagieren können soll, auch wenn er gerade mit der Berechnung des nächsten Spielzugs beschäftigt ist.

In dieser Teilaufgabe soll die Aufteilung in zwei Prozesse stattfinden, wobei der *Thinker*-Prozess noch keine Aufgaben übernimmt. Zum Erstellen eines weiteren Prozesses verwenden Sie `fork()`. Es ist Ihnen freigestellt, ob der Kindprozess die Aufgabe des *Thinkers* oder des *Connectors* übernimmt, es ist jedoch ausdrücklich empfohlen, dass der Elternprozess der *Thinker* und der Kindprozess der *Connector* wird [2].

Der *Thinker* hat zu diesem Zeitpunkt noch keine Aufgabe. Er existiert genau so lange wie der *Connector* und soll terminieren, wenn der *Connector* terminiert. Der *Connector* führt die im letzten Übungsblatt implementierte `performConnection()`-Methode aus.

Achtung: Es ist darauf zu achten, dass der Kindprozess nicht zu einem Waisen- oder Zombieprozess wird.

3 Implementierung eines Shared-Memory-Bereichs

Die beiden in der vorherigen Teilaufgabe erstellten Prozesse kommunizieren über mehrere Wege miteinander. Einer dieser Wege ist ein gemeinsam genutzter Speicherbereich (englisch *Shared Memory* oder mit SHM abgekürzt). Zunächst legen Sie ein `struct` an, welches initial die folgenden Daten enthalten soll:

- Vom Gameserver übermittelter, individueller Spielname
- Unsere Spielernummer
- Anzahl der Spieler
- Die Prozess-IDs (PID) der beiden Prozesse

Im weiteren Verlauf des Projekts wird diese Datenstruktur noch weiter ausgebaut, da Sie feststellen werden, dass Sie mehrere Werte zwischen den Prozessen teilen müssen. Insbesondere sollen alle Spielparameter im gemeinsam genutzten Speicherbereich hinterlegt werden, auch wenn sie nicht – oder noch nicht – vom *Thinker* genutzt werden. Bitte bauen Sie die Datenstruktur auch ohne explizite Aufforderung nach Ihren Anforderungen aus.

Die Struktur soll in einem SHM-Segment abgelegt werden. Hierfür reservieren Sie ein SHM-Segment mit der Größe Ihrer o. g. Datenstruktur (Hinweis: `shmget()`, `shmat()`). Es empfiehlt sich den SHM-Key `IPC_PRIVATE` zu verwenden und keine Magic-Numbers.

Unser Client soll so konstruiert sein, dass, sollten wir eines Tages die Lust am Reversi-Spiel verlieren, auch andere ähnliche Spiele implementiert werden können. Diese anderen Spiele müssen nicht zwangsweise zwei Spieler haben, sondern können auch mehr (z. B. Mensch ärgere dich nicht) oder weniger (z. B. Solitaire) oder eine dynamische Anzahl Spieler verwalten können müssen. Aus diesem Grund wird in der o. g. Datenstruktur auch das Feld für die Anzahl der Spieler benötigt.

Es existieren zu jedem Spieler mindestens drei Eigenschaften:

- Spielernummer
- Name
- Flag, ob Spieler bereits registriert / bereit ist oder nicht

Diese Eigenschaften sollen für jeden Spieler ebenfalls im geteilten Speicher abgelegt werden. Nachdem die Größe dieses SHM-Bereichs erst bekannt ist, wenn der Gameserver die Anzahl der Spieler übermittelt, müssen Sie sich Gedanken dazu machen, wie Sie Ihr SHM-Segment (oder auch Ihre SHM-Segmente) sinnvoll reservieren und verwalten.

[2] Der *Thinker* wartet nur auf ein Unix-Signal als Anstoß und kann somit warten, bis der *Connector* terminiert.

Achtung:

- Das dynamische Vergrößern eines SHM-Segments wird nicht von allen Systemen unterstützt und soll daher nicht verwendet werden.
- Es ist darauf zu achten, dass die SHM-Segmente bei Prozessende wieder explizit entfernt werden (Hinweis: `shmctl()`, `IPC_RMID`). Sie können über die Kommandozeile mittels `ipcs` die reservierten SHM-Bereiche einsehen und mittels `ipcrm` Speicherbereiche, die durch fehlerhafte Testversionen des Clients nicht entfernt wurden, manuell entfernen.