Socket-Programmierung 3

Übungen zu Netzwerktechnologie
Bachelor-Studiengang Software Engineering
Vollzeitform
im Wintersemester 2019

Gerhard Jahn Stephan Leitner

2. Dezember 2019

1 Lehrziele

Studierende sollen durch diese Übung

- erlernen, wie ein gut skalierbares Serverprogramm praktisch umgesetzt werden kann;
- die Funktionsweise von `select()` kennenlernen.

2 Einführung

`select()` erlaubt die parallele Überwachung mehrerer Sockets. Die Funktion unterscheidet dabei drei Mengen von überwachbaren Sockets, gegliedert nach folgenden Ereignissen: lesebereite, schreibbereite und fehlerhafte Sockets.

Die Funktion blockiert solange, bis ein Ereignis auf einem der überwachten Sockets aufgetreten ist. Nach dem Retournieren von `select()` sind in den übergebenen Mengen nur noch jene Sockets enthalten, bei denen das entsprechende Ereignis aufgetreten ist. Alle anderen Sockets wurden von `select()` aus der Menge entfernt.

3 Durchführung

Aufgabe *1 – Ereignisorientierter Server mit `select()` (12 Punkte)*

Ziel dieser Übung ist es, den Hangman Server, den Sie bereits aus den vorhergehenden Übungen kennen, unter Verwendung der Funktion `select()` zu reimplementieren.

3.1 Funktionsweise des Servers

Bei dem ereignisorientierten Ansatz akzeptiert der Server mehrere Verbindungen in einem einzigen Prozess. Intern muss daher am Server zu jeder dieser Verbindungen der Zustand gespeichert werden – bei uns ist das der Spielstand des jeweiligen Client. Die eindeutige Identifikation des jeweiligen Spielstandes erfolgt über den dazugehörigen Socket-Deskriptor. Der Spielstand und der Socket-Deskriptor werden am Server in der Datenstruktur `state` abgelegt:

```
typedef struct state {
    char   *whole_word;       /*the word to be guessed*/
    int    word_len;
    char   part_word[WORDLEN];/*the part guessed already*/
    int    lives;             /*lives the player has left*/
    int    fd;                /*socket descriptor of client*/
    struct state* next;       /*pointer to next client */
} state;
```

Das eigentliche Spiel und die Verwaltung der Spielstände müssen Sie **nicht selbst implementieren**. Beides wird ihnen in Form der Dateien `service.c` und `service.h` zur Verfügung gestellt. Es geht bei dieser Aufgabe ausschließlich um die Programmierung der Netzwerklogik.

Das Modul `service` stellt über seine Schnittstelle in `service.h` die folgenden Funktionen zur Verwaltung der Spielstände bereit:

- `void service_init(int fd);`
  → Ein neuer Client hat sich verbunden und möchte das Spiel beginnen. `service_init(int fd)` legt hierzu einen neuen Datensatz vom Typ `state` an, und speichert ihn unter dem Schlüssel `fd` in der internen Liste ab. Auch die erste Nachricht an den Client wird hier versendet.

- `int service_do(int fd);`
  → Lädt den diesem Socket – identifiziert durch `fd` – zugeordneten Datensatz, liest die Benutzereingabe vom Socket und führt den nächsten Spielzug aus. Wenn `service_do()` den Wert 0 zurückliefert, ist das Spiel beendet. Mögliche Ursachen sind: Das Wort wurde erraten oder das letzte Leben wurde verspielt oder der Client hat die Verbindung beendet. Sie müssen dann `service_exit(int fd)` aufrufen. Keine dieser beiden Funktionen schließt den Socket `fd`, das müssen Sie selber machen.

- `void service_exit(int fd);`
  → Löscht den (dem Socket `fd` zugeordneten) Datensatz aus der internen Liste.

3.2 Vorgehensweise

- Kopieren Sie sich die Dateien `service.c` und `service.h` aus dem Moodle-Kurs.
- Implementieren Sie ein C-Modul, in dem die `main()`-Funktion folgendes macht:
  - Starten Sie einen Socket-Server, wie Sie es bereits gemacht haben (also `getaddrinfo()`, `socket()`, `bind()`, `listen()`).
  - Warten Sie mittels `select()` auf einen lesebereiten Socket. Wenn es sich beim lesebereiten Socket um den Master-Socket handelt, müssen Sie die neue Client-Verbindung mittels `accept()` annehmen und `service_init()` aufrufen. Wenn es sich beim lesebereiten Socket um eine Client-Verbindung handelt, rufen Sie `service_do()` für den nächsten Spielzug auf bzw. am Ende des Spiels auch noch `service_exit()`.

3.3 Hinweise

- Die Datei `WordCheck.c` wird in dieser Übung nicht benötigt, die gesamte Programmlogik des Hangman-Spieles ist in der Datei `service.c` implementiert.
- Als Ausgangspunkt für diese Übung kann ihnen das Tutorial `http://beej.us/guide/bgnet/` dienen, insbesondere der Abschnitt und das Quellcode-Beispiel zu select().