Aufgabe 2:

legs (12 Punkte)

Implementieren Sie ein einen einfachen „Spieleserver“ **legs** (low end game server), und einen geeigneten Client (**legs-client**). Dem Client wird dabei ein ASCII-Zeichen-Labyrinth übertragen, durch das der Anwender eine Figur (= roter Punkt) hindurch steuern kann. Leerzeichen repräsentieren dabei betretbare Bereiche, alle anderen Symbole (mit Außnahme von @) stellen dabei ein Hindernis dar, welches nicht betreten werden darf. Erreicht der Spieler ein @ Symbol ist das Spiel beendet. Aufgerufen werden die Programme mit:

```
legs portnummer Spielfeld [n]

legs-client host portnummer
```

wobei Spielfeld den Pfad zu einer Datei zeigt, die ein entsprechendes ASCII-Labyrinth enthält. Jede Zeile eines Spielfeldes besteht aus 80 Zeichen. Insgesamt gibt es 23 Zeilen je Feld. Es wird bei Start des Servers einmalig eingelesen. Der Server muss dabei folgendes Protokoll unterstützen:

Client sendet		**Server Antwort**
`GET GAME\n`	Sitzung wird initialisiert	übermittelt das Spielfeld zeilenweise (23 Zeilen, 80 Spalten pro Feld)
`MOVE UP\n`	Server reduziert aktuelle Position des Spielers in y-Richtung um 1 sofern dort kein Hindernis ist und überträgt die Koodinaten bzw. signalisiert das Spielende.	`<x> <y>\n` oder `END GAME\n`
`MOVE DOWN\n`, `MOVE LEFT\n`, `MOVE RIGHT\n`	analog zu `MOVE UP`	analog zu `MOVE UP`

Bricht der Spieler vorzeitig a, wird vom Client einfach die Verbindung geschlossen.

a) Makefile

Erstellen Sie ein Makefile mit dem sich das Programm legs und der legs-client erstellen lässt. Das Makefile soll außerdem die Pseudo-Targets all, clean und install enthalten.

b) Server

Der legs-Server bindet sich an einen übergebenen Port (**socket(2)**, **bind(2)**, **listen(2)**) und nimmt an diesem Anfragen entgegen (**accept (2)**). Für jede Verbindung startet der Server einen neuen Kindprozess (**fork(2)**), der die Anfrage bearbeitet und sich anschliessend beendet. Der Vaterprozess bereitet sofort die Entgegennahme der nächsten Anfrage vor. Beendete Kindprozesse sollen unmittelbar durch einen Signalhandler aufgeräumt werden (**waitpid(2)**, **sigaction(2)**).

c) Anzahl der Anfragen beschränken

Nun soll die Anzahl der gleichzeitig möglichen Anfrage-Bearbeitungen auf eine übergebene Zahl n reduziert, also die Anzahl der laufenden Kindprozesse entsprechend beschränkt werden. Da der Signalhandler nebenläufig ausgeführt wird, ist hier für geeignete Synchronisation zu sorgen (**sigprocmask(2)**).

d) Client

Zusätzlich zum Server soll nun noch ein passender Client erstellt werden. Hierzu muss lediglich eine Funktion `int getConnection(char *url, int port)` in der Datei legs-connect.c implementiert werden, die eine Verbindung zu der übergebenen URL auf dem entsprechenden Port herstellt(**gethostbyname(3)**, **connect(2)**), und einen entsprechenden File-Deskriptor zurück liefert (oder -1 im Fehlerfall).
Die übrige Funktionalität wird in Form einer Objektdatei von uns bereitgestellt.

Allgemeine Hinweise

- Die .o-Datei für den Client, sowie Spielfelder zum Testen finden Sie unter /proj/i4sp/pub/aufgabe2
- Der Client muss mit -lcurses gelinkt werden.
- Um sigemptyset nutzen zu können, benötigen Sie -D_POSIX_SOURCE, für SA_RESTART -D_BSD_SOURCE.
- Zum Testen finden Sie an oben genannter Stelle ausserdem einen fertigen Server und Client zum Testen.
- Um das Protokoll einsehen zu können, eignet sich ausserdem das Programm nc. nc **url port** bietet einen simplen Client, nc -l -p **port** einen Server.
- Die Abgabe erfolgt bis 5.6.2009 14:00.