SYSPROG BEISPIEL 1

Aufgabenstellung B

Würfel-Wettbewerb: Möge die/der Beste gewinnen! Sie sollen im Rahmen dieses Beispiels einen computergesteuerten Würfelspieler (Bot/Client) implementieren, der gegen andere Bots spielt. Derjenige, der die meisten Spiele gewinnt, ist Sieger des Wettbewerbs. Sofern der Bot gültige Spielzüge in einer vorgegebenen Zeitspanne tätigt und den Punktestand für sich und seine Gegenspieler kennt, gilt das Beispiel als funktional korrekt und es können alle Punkte erreicht werden (sofern die restlichen Richtlinen eingehalten wurden!).
Wir stellen den Würfelserver zur Verfügung. Clients können sich per TCP/IP mit ihm verbinden und an einem Spiel teilnehmen. Der Server leitet das Spiel, würfelt für die Bots und achtet darauf, dass sich alle an das Protokoll halten.
Es gibt 2 Phasen:

- die Trainingsphase: Bis zur Abgabedeadline können Sie Ihren Bot entwickeln und gegen andere (gerade in der Entwicklung befindlichen) Bots spielen lassen. Wir werden von Zeit zu Zeit Zwischenstände bekannt geben. Freilich fliesst kein Spielergebnis von dieser Phase in die 2. Phase ein.
- die Wettbewerbsphase: Nach der Abgabe werden wir den Wettbewerb intern durchführen: Jeder Bot spielt eine hohe Anzahl an Spielen gegen alle anderen Bots um Zufallseffekte nach Möglichkeit auszugleichen (es bleibt aber letztendlich ein Glücksspiel!). Verstösst ein Bot gegen das Protokoll (insb. in Bezug auf eine korrekten *BYE* Nachricht (siehe Protokollbeschreibung)) wird er disqualifiziert. Es wird eine Rangordnung entstehen, die wir im Wiki veröffentlichen werden.

Die Gewinnerin/Der Gewinner erhält 20 Bonuspunkte, jeder weitere Platz um 5 Bonuspunkte weniger (es gibt also insgesamt 4 Plätze und der 4. Platz erhält 5 Bonuspunkte).
Der Server-Hostname und die Portnummer lauten: sysprog-dicer.tilab.tuwien.ac.at:9001
Der Würfelserver ist auch extern erreichbar.

Das Spiel: Jeweils 3 Bots spielen mit 2 Würfeln gegeneinander. Der Bot, der gerade an der Reihe ist, würfelt immer mit beiden Würfel gleichzeitig. Die Augenzahl der Würfel wird zusammengezählt und immer solange zur eigenen Rundensumme addiert, bis der Bot einen Pasch (d.h. Augenzahl der beiden Würfeln ist gleich) erwürfelt. Mit einem Pasch endet für ihn die Runde und er verliert alle in dieser Runde erwürfelten Punkte. Das kann aber vermieden werden, denn vor dem Würfeln kann sich der Bot entscheiden:

- Würfeln und möglicherweise alle in der aktuellen Runde erspielten Punkte verlieren, oder
- an den nächsten abgeben und die bisherig erspielten Punkte auf jeden Fall behalten?

Die behaltenen Punkte werden immer zur aktuellen Rundensumme addiert. Wer so zuerst 150 Punkte oder mehr erkämpft hat, gewinnt!

Der Würfelbot

```
Client: SYNOPSIS
  dicebot -n <bot-name> [-p <server-port>] <server-hostname>
```

Implementieren Sie einen Würfelbot (Client), der mittels TCP/IP mit dem Würfelserver kommunizieren kann und dabei das vorgegebene Protokoll einhält.

Anleitung

Dem Würfelbot wird beim Aufruf zumindest der Server Hostname und der eigene Botname (Option *-n*) übergeben. Optional soll der Server Port per *-p* konfigurierbar sein; standardäßig soll der Bot auf den Port 9001 verbinden. Achten Sie auf eine korrekte Parameterbehandlung mittels *getopt(3)* Legen Sie zuerst einen TCP/IP Socket an. Stellen Sie dann die zum Host-Namen des Servers zugehörige IPv4 IP-Adresse mittels *getaddrinfo(3)* fest. Verbinden Sie den Socket mittels *connect(2)*. Interagieren Sie mit dem Würfelserver gemäß dem Protokoll (siehe Tabelle 1) mittels *read* und *write* oder ähnlichem. Nachdem das Spiel entschieden ist, schließen Sie den Socket mittels *close(2)*. Weiters soll der Client bei Empfang eines der Signale SIGINT, SIGQUIT und SIGTERM die Meldung "Received termination request - terminating..." auf *stderr* ausgeben und sauber terminieren (d.h. eventuell offene Verbindungen mit dem Server müssen geschlossen werden). Unser Würfelserver wird Ihnen kaum genug Zeit geben, diese Art der Terminierung zu testen - aber Sie können netcat[1] lokal als 'manuellen' Würfelserver betreiben:

```
# nc -l <port-num>
```

Netcat wartet nun auf eine Verbindung und gibt alle Nachrichten, die es vom Bot erhält, auf *stdout* aus und verschickt alle Nachrichten an den Bot, die es über *stdin* bekommt. Starten Sie nun Ihren Bot und übergeben Sie ihm *localhost* und <*port-num*> – letzteres muss eine nicht priviligerte[2] und nicht benutzte Portnummer sein. Da netcat die Verbindung nicht schliesst, haben Sie ausreichend Zeit eventuelle Protokollprobleme zu debuggen oder die Terminierung mittels Signal zu testen.

Für die Trainingsphase: Um den Client wiederholt gegen andere testen zu können, empfiehlt es sich ihn per Bash Script rasch hintereinander zu starten. Bitte sehen Sie aber davon ab, mehr als 2 Instanzen Ihres Bots gleichzeitig auf unserem Würfelserver spielen zu lassen:

```
# while true; do ./dicebot -n <bot-name> sysprog-dicer.tilab.tuwien.ac.at; sleep 0.1; done
```

Das Protokoll:

Keine der Nachrichten (weder vom Server noch vom Client) ist länger als 100 Zeichen und jede Nachricht ist mit einem *Newline* terminiert. Da manche Nachrichten Parameter enthalten, könnte Ihnen die Funktion *strtok* bzw. *strtok_r* von Nutzen sein. Als Delimiter innerhalb der Nachrichten kommt das <*space*> (Leerzeichen) zur Anwendung. "Freitext" enthält ebenfalls Leerzeichen, kann aber bei allen Nachrichten ignoriert werden. Der Client soll innerhalb von max. 2 Sekunden antworten. Weiters ist die Anzahl der Runden pro Spiel aus praktischen Gründen auf 100 beschränkt. Bei Überschreitung erhalten alle beteiligten Clients eine entsprechende *ERR* Nachricht. Die Nachrichten des Protokolls sind in Tabelle 1 zusammengefasst. Die Reihenfolge entspricht dem Vorkommen der Nachrichten in einer typischen Session.
Hinweis: Es empfiehlt sich *fgets(3)* für den Empfang der einzelnen Nachrichten zu nutzen. Sie können per *fdopen(3)* einen File Descriptor als File Stream öffnen.
Eine typische Session (mitgeschnitten am Server) kann so aussehen:

```
0> HELO This is the SYSPROG dice game server (v0.1 [train mode]). Welcome player!
0< AUTH dice1
1> HELO This is the SYSPROG dice game server (v0.1 [train mode]). Welcome player!
1< AUTH dice2
2> HELO This is the SYSPROG dice game server (v0.1 [train mode]). Welcome player!
2< AUTH dice3
2> TURN 0 Your turn!
2< ROLL
0> THRW 0 4 5 dice3
```

[1] `man nc`
[2] Port >1023

```
1> THRW 1 4 5 dice3
2> THRW 2 4 5 dice3
2> TURN 9 Your turn!
[...]
0> THRW 0 3 5 dice2
1> THRW 2 3 5 dice2
2> THRW 1 3 5 dice2
1> WIN You win, because you have reached 150 points or more first.
2> DEF You lose, because another player has reached 150 points or more first.
0> DEF You lose, because another player has reached 150 points or more first.
1< BYE 203 0 145
2< BYE 0 145 203
0< BYE 145 203 0
0> BYE
1> BYE
2> BYE
```

Hinweis: Jede Zeile repräsentiert eine Nachricht. Die beiden ersten Zeichen jeder Zeile geben den Kommunikationspartner (*Player ID*) und die Richtung der Kommunikation (> zum Client, < vom Client) an.

Hinweis zum Stoff des 1. Computertests

Auch wenn Sie bei diesem Beispiel keinen TCP/IP Server implementieren mussten, besteht die Möglichkeit, dass zum 1. Computertest ein Server zu entwickeln ist - d.h. Sie müssen in der Lage sein, einen TCP/IP Socket mit *socket(AF_INET, SOCK_STREAM,0)* zu erstellen, an eine Portnummer zu binden (mittels *bind*), den Socket mit dem Kommando *listen* auf das Akzeptieren von Verbindungen vorzubereiten und beispielsweise in einer Endlosschleife per *accept* Verbindungsanfragen der Clients zu akzeptieren (für jede Verbindungsanfrage generiert *accept* einen neuen Socket).

Sender	Nachricht	Beschreibung
Server	HELO "Freitext"	Clientbegrüssung. Der Server erwartet nach dieser Nachricht eine *AUTH* Nachricht.
Client	AUTH "Botname"	Der Client soll seinen Namen dem Server übermitteln. Der Name *Botname* darf keine Sonderzeichen und nur Buchstaben bzw. Zahlen aus dem englischen Alphabet enthalten.
Server	TURN "akt. Punkte" "Freitext"	Der Server signalisiert so einem Client, dass er an der Reihe ist. Diese Nachricht erhält nur der Client der gerade an die Reihe kommt und beinhaltet die eigene Punktezahl. Der Client muss mit *ROLL* oder *SAVE* antworten.
Client	SAVE	Diese Nachricht kann an den Server als Antwort auf die *TURN* Nachricht verschickt werden. Der Client gibt damit seine Entscheidung "Sichern" bekannt. Der aktuelle Rundenstand wird zu den gespeicherten Punkten addiert und der nächste Spieler kommt an die Reihe.
Client	ROLL	Diese Nachricht kann an den Server als Antwort auf die *TURN* Nachricht verschickt werden. Der Client gibt damit seine Entscheidung "Weiterwürfeln" bekannt.
Server	THRW "Spieler ID" "W1" "W2" "Bot Name"	Kommt es zu einem Wurf (als Reaktion auf die *ROLL* Nachricht eines Clients) verschickt der Server das Ergebnis an alle Bots. Dabei hat der Bot der gerade am Zug ist immer die *Spieler ID* 2 während gegnerische Ergebnisse die *Spieler IDs* 0 bzw. 1 haben. Die Gegner *Spieler ID* bleibt im Laufe eines Spiels immer gleich. *W1* bzw. *W2* beziehen sich auf die Augenzahlen der beiden Würfel. *Bot Name* enthält den in der *AUTH* Nachricht angegebenen Namen des jeweiligen Bots, der gerade am Zug ist.
Server	WIN "Freitext"	Der Client, der gewinnt, erhält, sobald er als erster >= 150 Punkte erreicht hat, diese Nachricht vom Server.
Server	DEF "Freitext"	Die Clients die verlieren erhalten nach dem Wurf des gewinnenden Bots diese Nachricht.
Client	BYE "my points" "E1 points" "E2 points"	Der Client verabschiedet sich mit dieser Nachricht nach einer *DEF* oder *WIN* Nachricht des Servers. Dabei gibt er den aktuellen Spielstand zur Kontrolle an den Server zurück.
Server	BYE	Die letzte Nachricht des Servers wird nach der *BYE* Nachricht des Clients an jeden Client verschickt, sofern alle den korrekten Spielstand übermittelt haben. Hat sich ein Bot geirrt, ist das gesamte Spiel ungültig und alle erhalten eine *ERR* Nachricht.
Server	ERR "Freitext"	Es ist ein Fehler aufgetreten und das Spiel kann nicht fortgesetzt werden. Der Grund ist im Freitext enthalten (e.g. Verbindungsunterbrechungen, Protokollfehler, ...) .

Tabelle 1: Nachrichten des Würfelprotokolls