

Entwicklung eines IRC-Bots

**Nils Gutsche
(aka Vellas)**

Vorwissen und Werkzeuge

Für dieses Tutorial sollte man zumindest Grundkenntnisse mit C++, der STL (Standard Template Library) und der Socketprogrammierung (Client-Side) haben. Ich werde in diesem Tutorial nicht auf Funktionen der Socket-API oder der STL eingehen. Wem eine Funktion nicht bekannt ist, kann auf den nachfolgend genannten Seiten nachschlagen.

Socket-Programmierung:

<http://www.c-worker.ch/dokuwsck/index.php>

<http://www.c-worker.ch/tuts.php> (Winsock Tutorial)

C++ Referenz / STL Referenz:

<http://www.cppreference.com/>

<http://www.kuzbass.ru:8086/docs/isocpp/>

Den Bot werde ich hier mit Hilfe der bedingten Kompilierung für UNIX-Betriebssysteme und für Windows-Betriebssysteme schreiben.

In Windows wird das Platform-SDK benötigt. Eine Beschreibung zur Installation und Einrichtung mit Visual C++ 2005 Express finden Sie hier:

<http://msdn.microsoft.com/vstudio/express/visualc/usingpsdk/>

Des Weiteren muss die Anwendung in Windows gegen die ws2_32.lib gelinkt werden.

Auf UNIX-Systemen wird zur Kompilierung lediglich der g++ benötigt.

Da ich davon ausgehe, dass Sie sich mit Ihrem Compiler auskennen, werde ich nicht weiter auf betriebssystem- oder compilerspezifische Details eingehen.

Was ist das IRC?

IRC, Internet Relay Chat, ist ein rein textbasiertes Chat-System. Es ermöglicht Gesprächsrunden zwischen mehreren Teilnehmern in einem oder mehreren Channels gleichzeitig, aber auch Privatgespräche zwischen zwei Teilnehmern.

Das ursprüngliche Protokoll ist im RFC 1459 beschrieben:

<http://www.ietf.org/rfc/rfc1459.txt>

Es gibt noch weitere RFCs zur Erweiterung des RFC 1459, allerdings haben diese in der Praxis kaum eine Bedeutung, da es keinen IRC-Server gibt, der die weiteren RFCs auch nur annäherungsweise vollständig implementiert hat. Daher konzentrieren wir uns auf das o.g. RFC.

Was ist ein Bot?

Ein Bot ist ein Computerprogramm, das Aufgaben, mit denen ein Mensch zeit- und/oder mengenmäßig überfordert wäre, weitestgehend automatisch erledigt.

Was ist ein IRC-Bot?

Im IRC werden Bots eingesetzt um Channels zu kontrollieren und um Channels offen zu halten. IRC-Bots können die Nachrichten von Usern prüfen um dann auf diese zu reagieren, so könnten Sie beispielsweise einen, den Channel betretenden, User begrüßen. IRC-Bots haben in der Regel, für die User eines Channels, nützliche Befehle eingebaut (z.B. google-Suche). Einige bieten sogar Spiele (z.B. Hangman) an.

Im Folgenden werde ich für IRC-Bot nur Bot verwenden.

Der Verbindungsaufbau

Die wohl wichtigste Frage ist sicher: Wie verbinden wir uns mit einem IRC-Server?

Die meisten Seiten im Netz, die einen Channel im IRC haben, geben über ihre Seite auch die Verbindungsdaten zu ihrem IRC-Channel bekannt. Wenn man in keiner Community mit einem solchen Channel ist, kann man einen Blick auf die folgende Seite werfen:

<http://irc.netsplit.de/networks/>

Dort finden wir zumindest schonmal eine Liste von IRC-Netzwerken. Bei einem Klick auf ein Netzwerk können wir uns die genaueren Verbindungsdaten wie Hostnamen und Port ansehen. Für unser Beispiel verwenden wir das insiderZ.de Netzwerk:

<http://irc.netsplit.de/networks/details.php?net=insiderZ.de>

Wir werden uns also zu dem Host "irc.insiderZ.de" auf den Port 6667 (Standardport) verbinden. Für den Verbindungsaufbau und -abbau schreiben wir uns zwei entsprechende Funktionen.

Für den gesamten Bot werden folgende Headerdateien benötigt, diese sollten Sie am Anfang ihrer CPP-Datei einfügen. Die Liste wird nicht mehr erweitert:

```
#include <iostream>      /* std::cout, std::cerr, std::endl */
#include <cstdlib>        /* exit */
#include <cstring>        /* memcpy, memset */
#include <string>         /* std::string */
#include <cerrno>         /* perror */
```

```
#ifdef WIN32
#include <winsock2.h>
#else
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#endif
```

Um nicht immer „std:“ schreiben zu müssen habe ich am Anfang der Anwendung `using namespace std;` eingefügt.

Konstanten:

```
const unsigned int MAX_LINE = 1024; // Groesse des Empfangspuffers
const int PORT = 6667;
const char *HOST = "irc.insiderZ.DE";
```

Globale Variablen:

```
#ifdef WIN32
SOCKET sockfd;
#else
int sockfd;
#endif
```

Globale Variablen sind natürlich nicht schön, allerdings wird sockfd immer benötigt, wenn eine Socketfunktion aufgerufen wird und trägt somit nur zur besseren Übersicht bei und ist in diesem Fall auch eine Programmoptimierung, da die Variable nicht bei jedem Funktionsaufruf übergeben werden muss.

Verbindungsaufbau:

```
void irc_connect() {
#ifdef WIN32
    // Windows-Socket initialisieren
    WSADATA wsa;
    if(WSAStartup(MAKEWORD(2,0),&wsa)!=0)
        exit(1);
#endif

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    // Cast eigentlich nur in Windows noetig, fuer bessere Uebersicht hier ohne bedingte Kompilierung
    if(static_cast<int>(sockfd) < 0) {
        perror("socket()");
        irc_disconnect();
        exit(1);
    }

    hostent *hp = gethostbyname(HOST);
    if(!hp) {
        cerr << "gethostbyname()" << endl;
        irc_disconnect();
        exit(1);
    }

    sockaddr_in sin;
    memset((char*)&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    memcpy((char*)&sin.sin_addr, hp->h_addr, hp->h_length);
    sin.sin_port = htons(PORT);
    memset(&(sin.sin_zero), 0, 8*sizeof(char));

    if(connect(sockfd, (sockaddr*) &sin, sizeof(sin))== -1) {
        perror("connect()");
        irc_disconnect();
        exit(1);
    }
}
```

Diese Funktion enthält wenig besonderes. Zunächst wird der Socket mit der `socket()` Funktion initialisiert. Anschließend wird die Adresse des Hosts mit `gethostbyname()` ermittelt. Danach wird die Verbindung aufgebaut. An dieser Funktion ist nichts besonderes, alles was sie enthält sind normale Standard-Funktionsaufrufe, wie man sie bei jeder Client-Anwendung durchführt. Diese Anweisungen wird man so oder so ähnlich auch in jedem Socket-Tutorial vorfinden.

Wie man sieht, wird bei Fehlschlägen die Funktion `irc_disconnect()` aufgerufen. Diese gibt den Socket wieder frei.

Verbindungsabbau:

```
void irc_disconnect() {
#ifdef WIN32
    // Windows-Socket freigeben
    closesocket(sockfd);
    WSACleanup();
#else
    close(sockfd);
#endif
}
```

Die Funktion `irc_connect` ist die erste Funktion in der Anwendung die aufgerufen wird und die Funktion `irc_disconnect` sollte die letzte sein, damit würde unsere Anwendung bisher

wie folgt aussehen:

```
int main() {
    irc_connect();
    irc_disconnect();
}
```

Wie man sieht, sieht man nicht viel. Die Verbindung wird aufgebaut und wieder abgebaut.

Senden von Nachrichten

Zum Senden von Nachrichten braucht man lediglich die Socketfunktion `send()`. Damit wir nicht immer so viel zu schreiben haben, verpacken wir uns diese ein wenig:

```
void s2u(const char *msg) { //send to uplink
    send(sockfd, msg, strlen(msg), 0);
}
```

Diese Funktion werde ich für alle Nachrichten an den Server verwenden. Der übergebene Puffer muss immer mit einem `“\r\n”` abgeschlossen sein.

Ping – Pong

Der IRC-Server prüft zwischendurch ob unser Client noch aktiv ist. Dies macht der Server mit dem Senden eines PINGs, was im IRC lediglich eine Nachricht mit dem Inhalt “PING” und einem String (Parameter von “PING”), den wir zurück senden müssen, ist. Damit der Server uns nicht rauswirft, müssen wir ihm mit einem “PONG” und dem Parameter antworten. Wenn der Server uns also “PING :ircServer” sendet, müssen wir mit “PONG :ircServer” antworten.

Auch dafür schreiben wir uns eine einfache Funktion:

```
void ping_parse(const string &buffer) {
    size_t pingPos = buffer.find("PING");
    if(pingPos!=string::npos) {
        string pong("PONG"+buffer.substr(pingPos+4)+"\r\n");
        cout << pong;
        s2u(pong.c_str());
    }
}
```

Wenn in buffer “PING” gefunden wurde, schneiden wir einfach alles hinter “PING” ab, hängen es an “PONG” und senden es an den Server zurück.

Empfangen von Nachrichten

Zum Empfangen von Nachrichten schreiben wir uns zunächst eine einfache Schleife und fügen diese in die Hauptroutine (main) unserer Anwendung ein:

```
for(;;) {
    char buffer[MAX_LINE+1] = {0};
    if(recv(sockfd, buffer, MAX_LINE*sizeof(char), 0)<0) {
        perror("recv()");
        irc_disconnect();
        exit(1);
    }
    cout << buffer;
    irc_parse(buffer);
}
```

Wir warten mit der `recv`-Socketfunktion auf eine Antwort des Servers. Wenn alles ok ist,

geben wir die empfangene Nachricht aus und übergeben den Puffer an eine Parser-Funktion, die allerdings wie nachfolgend zu sehen, nicht wirklich parst:

```
void irc_parse(string buffer) {
    if(buffer.find("\r\n") == buffer.length()-2)
        buffer.erase(buffer.length()-2);
    ping_parse(buffer);
    bot_functions(buffer);
}
```

Wir schneiden erstmal die String-Endekennung für die spätere Behandlung der empfangenen Nachricht ab. Die Funktion `bot_functions` benutzen wir später zum Parsen von Nachrichten um dann auf diese mit einer Antwort unseres Bots reagieren zu können.

Einem Channel betreten

Nun könnte unser Bot sich immerhin zum IRC-Server verbinden und auf die PINGS antworten. Das ist aber noch nicht wirklich viel. Damit unser Bot uns auch wirklich was nützt, sollte er eine Identität annehmen und auch einen Channel betreten können. Damit dieser Nickname nicht zurückgesetzt wird, sollte man sich eventuell erstmal mit einem IRC-Client zum IRC-Server verbinden und einen Nickname für seinen Bot registrieren, sofern in dem IRC-Netzwerk die Registrierung eines Nicknames möglich ist. Bei `insiderZ.de` ist dies möglich, daher habe ich für die Identifizierung und das Betreten eines Channels eine Routine geschrieben, die das nach dem Verbinden erledigt:

```
void irc_identify() {
    s2u("NICK Bot\r\n"); // Nickname
    s2u("USER Bot 0 0 :VellasBot\r\n"); // Userdaten
    s2u("PRIVMSG NickServ IDENTIFY password\r\n"); // Identifizieren
    s2u("JOIN #channel\r\n"); // Channel betreten
    s2u("PRIVMSG #channel :Hallo!\r\n"); // Begrüßungsnachricht
}
```

Mit `NICK` und `USER` werden die Benutzerdaten festgelegt. Mit `PRIVMSG` sendet man Nachrichten an einen Dienst (bei der Identifizierung z.B. `NickServ`, der `Nick-Server` bei dem sich die Benutzer anmelden), eine Person oder einen Channel. Mit `JOIN` betreten (joinen) wir einen Channel, damit wir Nachrichten an diesen Channel senden und Nachrichten von diesem Channel empfangen können. Mit `PRIVMSG` senden wir dann eine Begrüßungsnachricht in den Channel.

Botfunktionen

Für die Realisierung der Bot-Funktionen habe ich bereits eine entsprechende Routine vorgesehen. Dieser übergeben wir einfach den empfangenen Puffer und suchen einfach nach bestimmten Wörtern. Wenn zum Beispiel eine Person den Channel mit "Hallo" begrüßt, können wir beispielsweise ebenfalls mit einem "Hallo" antworten:

```
void bot_functions(const string &buffer) {
    if(buffer.find("Hallo") != string::npos) {
        s2u(("PRIVMSG #channel :Hallo!\r\n").c_str());
    }
}
```

Diese Funktion können wir natürlich beliebig ändern und anpassen. Um beispielsweise sicherzustellen, dass ein bestimmtes Wort am Anfang der Benutzernachricht steht, könnten wir z.B. nachsehen ob sich vor dem Wort ein Doppelpunkt befindet. Natürlich ist diese Methode nicht 100%ig richtig, man müsste zumindest sicherstellen, dass es sich bei dem Doppelpunkt um den zweiten Doppelpunkt im Puffer handelt. Wenn der Benutzer

einem anderen Benutzer einfach was nachplappern soll, können wir dies beispielsweise wie folgt realisieren, wobei unser Schlüsselwort am Anfang der Zeile stehen muss:

```
size_t pos = 0;
if((pos=buffer.find("say "))!=string::npos) {
    s2u("PRIVMSG #channel :"+buffer.substr(pos+5)+"\r\n").c_str());
}
```

Wenn jetzt ein Benutzer "say Ich bin toll" schreibt, sendet unser Bot "Ich bin toll" zurück.

Wenn unser Bot nur auf Nachrichten von uns reagieren soll, könnten wir beispielsweise prüfen ob sich folgendes am Anfang des Puffers befindet:

```
buffer.find("User!User@User.user.insiderZ.DE")==0
```

"User" ist in diesem Fall mit den Daten die bei NICK und USER gesetzt wurden zu ersetzen. Natürlich nicht mit den Bot-Daten, sondern mit den Daten unseres Nicknames. Das diese Daten entsprechend gesetzt sind, kann man über den verwendeten Client sicherstellen. Wie der String am Anfang genau aussieht, muss man durch ausprobieren auf dem Server herausfinden, im insiderZ-Netzwerk sollte der String ähnlich wie oben angegeben aussehen. Damit könnten wir beispielsweise sorgen, dass wir die einzigen sind, die unseren Bot beenden können:

```
if(buffer.find("User!User@User.user.insiderZ.DE")==0 && buffer.find("exit")!=string::npos) {
    s2u("PRIVMSG #channel :Cya\r\n");
    irc_disconnect();
    exit(0);
}
```

Natürlich nicht das Beenden der Verbindung vergessen. :)

Abschlusswort

Ich hoffe das dieses Tutorial einen Einblick in die Programmierung eines IRC-Bots mit C++ gegeben hat. Natürlich kann man noch viel mehr mit dem IRC-Protokoll machen, als es hier besprochen wurde. Um sich näher damit zu beschäftigen, lesen Sie sich einfach das am Anfang genannte RFC zum IRC-Protokoll durch. Dort finden Sie alle benötigten Informationen zur Entwicklung eines Bots oder gar eines Clients. Im Anhang finden Sie den kompletten Beispiel-Quellcode.

Wenn Sie nur das PDF-Dokument haben und den Beispiel-Quellcode in einer CPP-Datei möchten, finden sie dieses Tutorial in einem RAR-Archiv mit diesem Dokument und der CPP-Datei auch auf meiner Homepage:

<http://www.vellas.de/>

Anhang

// RFC zu IRC: <http://www.ietf.org/rfc/rfc1459.txt>

```
#include <iostream>      /* std::cout, std::cerr, std::endl */
#include <cstdlib>        /* exit */
#include <cstring>        /* memcpy, memset */
#include <string>         /* std::string */
#include <cerrno>         /* perror */

#ifdef WIN32
#include <winsock2.h>
#else
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#endif

const unsigned int MAX_LINE = 1024; // Groesse des Empfangspuffers

using namespace std;

#ifdef WIN32
SOCKET sockfd;
#else
int sockfd;
#endif

void irc_disconnect() {
#ifdef WIN32
    // Windows-Socket freigeben
    closesocket(sockfd);
    WSACleanup();
#else
    close(sockfd);
#endif
}

void s2u(const char *msg) { //send to uplink
    send(sockfd, msg, strlen(msg), 0);
}

void irc_connect() {
#ifdef WIN32
    // Windows-Socket initialisieren
    WSADATA wsa;
    if(WSAStartup(MAKEWORD(2,0),&wsa)!=0)
        exit(1);
#endif

    const int PORT = 6667;
    const char *HOST = "irc.insiderZ.DE";
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    // Cast eigentlich nur in Windows noetig, fuer bessere Uebersicht hier ohne bedingte Kompilierung
    if(static_cast<int>(sockfd) < 0) {
        perror("socket()");
        irc_disconnect();
        exit(1);
    }
}
```

```

hostent *hp = gethostbyname(HOST);
if(!hp) {
    cerr << "gethostbyname()" << endl;
    irc_disconnect();
    exit(1);
}

sockaddr_in sin;
memset((char*)&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
memcpy((char*)&sin.sin_addr, hp->h_addr, hp->h_length);
sin.sin_port = htons(PORT);
memset(&(sin.sin_zero), 0, 8*sizeof(char));

if(connect(sockfd, (sockaddr*)&sin, sizeof(sin))!=-1) {
    perror("connect()");
    irc_disconnect();
    exit(1);
}
}

void irc_identify() {
    s2u("NICK Bot\r\n"); // Nickname
    s2u("USER Bot 0 0 :Bot\r\n"); // Userdaten
    s2u("PRIVMSG NickServ IDENTIFY password\r\n"); // Identifizieren
    s2u("JOIN #channel\r\n"); // Channel betreten
    s2u("PRIVMSG #channel :Hello!\r\n"); // Begrueßungsnachricht
}

void ping_parse(const string &buffer) {
    if(buffer.find("PING")==0) {
        string pong("PONG"+buffer.substr(4)+"\r\n");
        cout << pong;
        s2u(pong.c_str());
    }
}

void bot_functions(const string &buffer) {
    size_t pos = 0;
    if((pos=buffer.find(":say "))!=string::npos) {
        s2u(("PRIVMSG #channel :"+buffer.substr(pos+5)+"\r\n").c_str());
    }
    else if(buffer.find(":User!User@User.user.insiderZ.DE")==0 && buffer.find("exit")!=string::npos) {
        s2u("PRIVMSG #channel :Cya\r\n");
        irc_disconnect();
        exit(0);
    }
}

void ping_parse(const string &buffer) {
    size_t pingPos = buffer.find("PING");
    if(pingPos!=string::npos) {
        string pong("PONG"+buffer.substr(pingPos+4)+"\r\n");
        cout << pong;
        s2u(pong.c_str());
    }
}

```

```
int main() {
    irc_connect();
    irc_identify();
    for(;;) {
        char buffer[MAX_LINE+1] = {0};
        if(recv(sockfd, buffer, MAX_LINE*sizeof(char), 0)<0) {
            perror("recv()");
            irc_disconnect();
            exit(1);
        }
        cout << buffer;
        irc_parse(buffer);
    }
    irc_disconnect();
}
```