Визитка

МАКСИМ ЛОБОВ, *руководитель технического отдела в туристической фирме*

IRC: сохраняем сообщения в СУБД MySQL

В статье рассказывается о том, как организовать ведение лога сообщений для IRC-серверов на примере двух наиболее часто используемых программных продуктов

Мое руководство постоянно говорит: «Максим, если нет шага вперед, значит, есть шаг назад». Не могу с этим не согласиться: мы идем вперед, но не такими большими шагами, как хотелось бы. Бурный технологический рост сдерживается умеренным финансированием. Полагаю, что будущее нашей и других фирм – это внедрение CRM, единой интегрированной системы по обработке всего-всего начиная от звонков, писем, факсов, данных СКУД и заканчивая ценами на веб-сайте и статистикой on-line-продаж вплоть до специфической для конкретной сферы бизнеса информации вроде географического расположения гидов и курьеров на карте.

В реальности, пока CRM нет, мы используем множество мелких решений, таких как многопользовательские планировщики, система учета звонков, электронная почта, SMS-рассылки-уведомления для сотрудников.

С одной стороны, отдельные программы унифицированы и удобны в настройке, с другой, без доработки они не решают всех необходимых задач.

Как это было исторически

Итоговое используемое нами решение – это внутрикорпоративный доверенный IRC-чат с возможностью авторизованного внешнего подключения удаленных офисов, партнеров и откомандированных сотрудников. По внутреннему чату происходит не только закрытое общение руководства фирмы (по одному каналу), но и сотрудников с партнерами (по другому каналу), проходит много текущей внутренней информации, как, например, о том, что плановая летучка будет в 17:30, а какой-то отель попал в стоп-лист.

К использованию протокола IRC мы пришли не сразу. На рубеже 2000-х для внутреннего общения мы использовали небольшой чат собственной разработки, но со временем поняли, что удобнее – IRC, так как в общении с партнерами важнее иметь открытый стандарт (RFC 2810-2813) с доступными клиентами и не зависеть от платформы.

Большинство готовых IRC-серверов не сохраняло историю канала, то есть сотрудник, работающий во вторую смену, не видел новости, прошедшие по каналу до его прихода.

Также довольно часто руководство хотело знать, кто из сотрудников дал тому или иному клиенту скидку больше положенной или договорился «не на тех условиях». Для того чтобы подобных проблем не возникало, нами был переписан IRC-сервер, что позволило не только видеть историю каналов на intranet-сайте, но и выборочно интегрировать сообщения в стороннее коммерческое программное обеспечение по обработке заявок клиентов.

Почему был выбран IRC?

Видя, что отдельные, разрозненные решения приводят к косякам, руководство поставило задачу создания быстрого и безопасного обмена сообщениями между сотрудниками фирмы, руководством, партнерами и удаленными офисами с возможностью удобного просмотра истории сообщений.

Подразумевалось, что новое решение будет основным средством обмена информацией по важным вопросам внутри фирмы и с VIP-партнерами, при этом все остальные программы и способы взаимодействия останутся.

Мы проанализировали все электронные средства общения, что использовались сотрудниками нашей фирмы, ориентированные на обмен информацией в группе, а именно: телефонные конференции (в том числе и с видео), IP-телефония, электронная почта, факс, внутрикорпоративный чат, ICQ, Jabber, IRC, Skype, SMS-рассылки, RSS-ленты, веб-форум, даже в социальной сети имелась закрытая группа.

Почти во всех средствах для нас нашлись большие минусы. Проанализировав их, мы поняли, что IRC – самое подходящее, а создание журнала сообщений – легко исправимый недостаток, об устранении которого речь пойдет ниже. Во всех других случаях нам пришлось бы программировать в разы больше.

Выбранные версии

В силу специфичности используемого нами решения не будем его рассматривать, а разберем, как добавить необходимую функциональность на примере двух IRC-серверов. По ходу ответим на вопросы: «Как и где вносить правку в ис-

Очень просто, имея небольшие навыки программирования, организовать журналирование сообщений IRC-серверов

ходные коды? Как добиться того, чтобы программа скомпилировалась успешно?»

Эксперименты будем проводить на Linux Fedora 14. Дистрибутив специально не выбирался, был использован первый, находящийся под рукой, как и программы Unreal IRCD [1] и ngircd [2].

Вполне возможно, что многие описываемые подходы могут быть успешно перенесены читателями на другие системы. Читатель не получит конечное готовое решение «под ключ», но найдет для себя много интересной и полезной информации о том, как сделать такое решение самостоятельно.

Также следует отметить важность понимания того, что на правильную работу IRC-сервера в различных ситуациях могут оказывать влияние аспекты, не рассмотренные в данной статье, такие как конфигурация СУБД, веб-сервера, PHP-интерпретатора, пакетного фильтра и прочее.

Журналирование на базе unrealircd

Нам потребуется операционная система Linux с установленной и запущенной СУБД MySQL и средствами разработки (компилятор gcc, библиотеки и пр.).

Скачиваем исходные коды программы сервера (на момент написания статьи актуальной была версия 3.2.8.1):

```
$ wget http://www.unrealircd.com/downloads/ ↵
       Unreal3.2.8.1.tar.gz
```

и распакуем их в директорию /opt:

```
$ tar -xv -C /opt -f Unreal3.2.8.1.tar.gz
```

Найдем фрагмент кода, отвечающий за отправку сообщений, логично, что из этого места можно будет занести их копии в журнал. Можно предположить, что искать нужно словосочетание «send message». Постараемся найти все файлы, его содержащие. Так как в названии функций, структур и прочих вещей внутри кода не может быть пробелов, то при поиске рассмотрим также замену пробела знаком «_». Так как для нас Send и send имеют одинаковый смысл, будем использовать параметр -i для регистронезависимого поиска. На этот раз нам повезло, мы получили довольно короткий список:

```
$ grep -irl "send[_ ]message" /opt/Unreal3.2/src
```

```
/opt/Unreal3.2/src/s_user.c
/opt/Unreal3.2/src/timesynch.c
/opt/Unreal3.2/src/win32/editor.c
/opt/Unreal3.2/src/send.c
```

Путем просмотра файлов выясним, что нам нужен send.c, рядом с найденным сочетанием находим функцию sendbufto_one, внутри которой видим 263 строчку:

```
Debug((DEBUG_ERROR, "Sending [%s] to %s", msg, to->name));
```

Логично предположить, что msg – есть искомое сообщение. То же видим и в заголовке этой функции, где описываются ее параметры, в строчке 250: «msg: the message».

Если мы хотим сохранить журнал сообщений в /var/log/unreal/msg.log, то вставим до или после строчки Debug следующий код, который откроет файл, допишет туда сообщение и закроет файл:

```
int     f;
int     msg_size;
f = open("/var/log/unreal/msg.log", O_APPEND | O_WRONLY);
msg_size = strlen(msg);
write(f, msg, msg_size);
write(f, "\r\n", 2);
close(f);
```

Указанная операция будет производиться с каждым сообщением. Как видим, мы внесли довольно небольшой фрагмент, написание которого будет под силу даже начинающим программистам.

Чтобы при компиляции не было ошибок типа «first use in this function», касаемо O_APPEND и O_WRONLY, надо дополнительно дописать в заголовке файла:

```
#include <fcntl.h>
```

Код для записи сообщений в БД MySQL будет несколько больше. Допишем его (подробнее о том, как программировать и работать с СУБД MySQL, см. [3, стр.367-438] и [4]):

```
MYSQL *mysql_conn;
char *mysql_server = "localhost";
int mysql_port = 3306;
char *mysql_user = "mysql_user";
char *mysql_password = "mysql_password";
char *mysql_database = "irclog";

mysql_conn = mysql_init(NULL);

    if (mysql_real_connect(mysql_conn, mysql_server,
         mysql_user, mysql_password, mysql_database, ↵
                 mysql_port, NULL, 0))
{
        char query[1024];
        memset(query, 0, 1024);

        strcat(query, "INSERT INTO `irclog`.`table` ↵
                 (message) VALUES ('");
        strncat(query, msg,1000);
        strcat(query, "');");

        mysql_query(mysql_conn, query);
        mysql_close(mysql_conn);
}
```

В заголовке файла необходимо дописать:

```
#include </usr/include/mysql/mysql.h>
```

и установить пакет mysql-devel, содержащий заголовочный файл mysql.h:

```
# yum install mysql-devel
```

При этом будут также установлены сопутствующие пакеты: keyutils-libs-devel, krb5-devel, libcom_err-devel, libselinux-devel, libsepol-devel, mysql, mysql-libs, openssl-devel, zlib-devel.

Замечание 1

Про установку пакета mysql-devel не было сказано ранее лишь по той причине, что для ведения журнала в файл данный пакет не нужен, поэтому разумнее рассмотреть этот вопрос в отношении того фрагмента, для которого он необходим.

Код выше максимально нагляден, но следует заметить, что по правилам программирования сам пользователь (или программист) должен следить за функциями strcat и strncat, чтобы не произошло какого-либо переполнения или выхода за границы. Поэтому для получения правильного рабочего варианта программы ее лучше доработать. Значения чисел 1024 и 1000 – размеры выделяемой и копируемой памяти – следует подобрать исходя из максимального размера сообщений. В RFC2816 [7, стр. 5] по этому поводу написано следующее: «IRC messages ... SHALL NOT exceed 512 characters in length» (дословно: длина сообщения не должна превышать 512 знаков). Также тип int для значения порта можно заменить на более подходящий UINT16.

После внесения правки перейдем к компилированию. Для этого нам потребуются средства разработки: компилятор и сопутствующие ему пакеты.

Замечание 2

Ранее, в описании требований к системе, было отмечено, что данные средства должны быть установлены, однако если вы упустили этот момент, то их можно установить сейчас, запустив:

```
# yum install gcc
```

При этом будут установлены: gcc, binutils, cloog-ppl, cpp, glibc-devel, glibc-headers, kernel-headers, libgomp, libmpc, mpfr, ppl.

Также необходимо, чтобы была установлена утилита make. Она понадобится нам для групповой компиляции после конфигурирования программы. Установить ее можно командой:

```
# yum install make
```

Подробно об установке Unreal написано в [5]. В случае необходимости обратитесь к этому ресурсу, а также к документации, поставляемой в архиве (/opt/Unreal3.2/doc/unreal32docs.ru.html). Данная статья не рассказывает о том, как правильно настроить сервер в той или иной конфигурации. Подразумевается, что читатель это либо уже умеет, либо освоит самостоятельно.

Используя привычный подход:

```
$ cd /opt/Unreal3.2
$ ./configure
$ ./make
```

то есть когда configure запускается без параметров, получаем ошибку:

```
/usr/bin/ld: modules.o: undefined reference to symbol
'dlclose@@GLIBC_2.2.5'
/usr/bin/ld: note: 'dlclose@@GLIBC_2.2.5' is defined in DSO
/lib64/libdl.so.2 so try adding it to the linker command
line
/lib64/libdl.so.2: could not read symbols:
Invalid operation
collect2: выполнение ld завершилось с кодом возврата 1
```

Решение, как ее обойти, было найдено из [5], следует запускать программу Config, а не configure.

```
$ ./Config
```

После запуска вам будет задано много вопросов по настройке сервера. Если не знаете ответов, нажимайте <Enter>, оставляя значение по умолчанию. В конце получите сообщение: «Now all you have to do is type 'make' and let it compile. When that's done, you will receive other instructions on what to do next». То есть можем переходить к запуску make.

Из содержимого Config мы узнаем, что вызов configure происходит с множеством полезных параметров:

```
./configure --with-showlistmodes --enable-hub ↵
        --enable-prefixaq --with-listen=5 ↵
        --with-dpath=/opt/Unreal3.2 ↵
        --with-spath=/opt/Unreal3.2/src/ircd ↵
        --with-nick-history=2000 --with-sendq=3000000 ↵
        --with-bufferpool=18 --with-hostname=linux ↵
        --with-permissions=0600 --with-fd-setsize=1024 ↵
        -enable-dynamic-linking
```

Конфигурирование запустим с такими параметрами.

Изначально программа Unreal не рассчитана на работу с MySQL, не содержит нужных записей в Makefile и не знает, где искать те или иные библиотеки. Чтобы из-за отсутствия данной информации запуск make не прерывался ошибкой «undefined reference to `mysql_init'... выполнение ld завершилось с кодом возврата 1», после configure следует в /opt/Unreal3.2/Makefile дописать к параметру XCFLAGS (41-я строчка) через пробел:

```
`/usr/bin/mysql_config --cflags`
```

и добавить строчку:

```
LDFLAGS=`/usr/bin/mysql_config --libs`
```

Запустим make и подождем, пока не появится: «Compile is now complete». (Фраза будет в рамочке, среди других фраз.)

Это означает, что наши изменения правильно внесены, и все собралось.

Запуск сервера

Согласно [5] в целях безопасности для запуска сервера предлагается создать и использовать отдельную учетную запись с минимальными правами. Указанные действия, как и правку конфигурационных файлов, копирование (установку) программы, рассматривать не будем. Полагаю, что те, кто работал с данным продуктом ранее, не встретят проблем, а новые пользователи прочитают документацию. Если опустить из рассмотрения вышеуказанные шаги, то запуск сервера выполняется командой:

```
# ./unreal start
```

Немного критики

Программистам с большим стажем указанный пример доработки может показаться не оптимальным. Возможно, у читателей тоже возникнут различные вопросы, типа: почему log-файл каждый раз открывается и каждый раз закрывается? Почему аналогичное поведение происходит с базой: подключились, отключились? Оптимально ли это? Если сервер базы данных будет недоступен, сообщения пропадут, журналирования не будет? Можно ли совместить эти два способа (в файл и в базу)? Скажем, если база недоступна, логи ведутся в файл?

На все вопросы отвечу: цель статьи – не предложить конечное оптимальное решение, а заставить читателей на простом примере задуматься о том, что править коды не так уж и сложно.

Журналирование сообщений в ngircd

Допишем аналогичный код подключения к серверу MySQL для IRC-сервера ngircd. Для этого скачаем его и распакуем.

```
$ wget ftp://ftp.berlios.de/pub/ngircd/ngircd-17.1.tar.gz
$ tar -xv -C /opt -f ngircd-17.1.tar.gz
```

Выполним те же самые шаги, что и ранее. Поиск фразы, похожей на «send message», через команду:

```
$ grep -irl "send[_ ]message" /opt/Unreal3.2/src
```

Нашел три файла:

```
/opt/ngircd-17.1/src/ngircd/irc.c
/opt/ngircd-17.1/src/ngircd/irc-write.c
/opt/ngircd-17.1/src/ngircd/channel.c
```

Детальный просмотр их содержимого показал, что искать нужно в первом, в самом начале (строчка 41), определение функции:

```
static bool Send_Message PARAMS((CLIENT *Client, ↵
        REQUEST *Req, int ForceType, bool SendErrors));
```

а ее описание можно встретить ниже, примерно начиная с 320-й строчки.

Просмотрев код, находим вызов «Channel_Write» в единственном месте, строчка 477:

```
if (!Channel_Write(chan, from, Client, Req->command, ↵
        SendErrors, Req->argv[1]))
    return DISCONNECTED;
```

Значит, было бы разумно сохранить данные до вызова этой функции.

Чтобы не загромождать код, введем дополнительную функцию write_message_to_mysql, а также учтем некоторые замечания, отмеченные выше:

```
@@ -474,6 +488,7 @@
    return DISCONNECTED;
        } else if (ForceType != CLIENT_SERVICE && ↵
                (chan = Channel_Search(currentTarget))) {
// Write to MySQL or log file
+ write_message_to_mysql(Client, Req);
    if (!Channel_Write(chan, from, Client, Req->command, ↵
        SendErrors, Req->argv[1]))
        return DISCONNECTED;
```

Добавим включение заголовочного файла mysql.h, прототипа функции, ее описание также в irc.c, в итоге получим:

```
--- irc.c           2010-08-29 21:07:42.000000000 +0400
+++ irc_patched.c   2011-01-13 14:41:34.000000000 +0300
@@ -32,6 +32,7 @@
 #include "messages.h"
 #include "parse.h"
 #include "tool.h"
+#include <mysql/mysql.h>

 #include "exp.h"
 #include "irc.h"
@@ -43,6 +44,7 @@
 static bool Send_Message_Mask PARAMS((CLIENT *from, ↵
        char *command, char *targetMask, char *message, ↵
        bool SendErrors));
+void write_message_to_mysql PARAMS((CLIENT * Client, ↵
        REQUEST * Req));


 GLOBAL bool
@@ -474,6 +476,7 @@
    return DISCONNECTED;
        } else if (ForceType != CLIENT_SERVICE && ↵
                (chan = Channel_Search(currentTarget))) {
// Write to MySQL or log file
+ write_message_to_mysql(Client, Req);
    if (!Channel_Write(chan, from, Client, Req->command, ↵
        SendErrors, Req->argv[1]))
        return DISCONNECTED;
@@ -559,5 +562,59 @@
        return CONNECTED;
 } /* Send_Message_Mask */

+void write_message_to_mysql(CLIENT * Client, REQUEST * Req){
+    char * m_nick;
+    m_nick=(char *)Client;
+    m_nick+=4;
+    char * m_channel;
+    char * msg=Req->argv[1];
+    m_channel=Req->argv[0];
+    MYSQL *mysql_conn;
+
+    char *mysql_server = "localhost";
+    UINT16 mysql_port = 3306;
+    char *mysql_user = "mysql_user";
+    char *mysql_password = "mysql_password";
+    char *mysql_database = "irclog";
+    const char *nomysql_logfile="/var/log/ngircd/msg.log";
+
+    mysql_conn = mysql_init(NULL);
+
+    // Преобразуем время
+    struct tm* current_time_struct_tm;
```

```
+    char* current_time_string;
+    time_t my_time;
+    my_time=time(NULL);
+    current_time_struct_tm=localtime(&my_time);
+    current_time_string=asctime(current_time_struct_tm);
+
+    if (!mysql_real_connect(mysql_conn, mysql_server, ↵
       mysql_user, mysql_password, mysql_database, ↵
       mysql_port, NULL, 0))
+    {
+       //Если БД не работает, пишем в файл
+        FILE* f=fopen(nomysql_logfile, "a");
+        fprintf(f, "%s\t%s %s: %s\n", current_time_string, ↵
                m_channel, m_nick, msg);
+        fclose(f);
+    }
+    else
+    {
+        char query[1024];
+        memset(query, 0, 1024);
+
+        strcat(query, "INSERT INTO `irclog`.`table` ↵
                (time, channel, nick, message) VALUES ('");
+        strcat(query, current_time_string);
+        strcat(query,"','");
+        strcat(query,m_channel);
+        strcat(query,"','");
+        strcat(query,m_nick);
+        strcat(query,"','");
+        strncat(query,msg, 1000);
+        strcat(query,"');");
+
+        mysql_query(mysql_conn, query);
+    }
+
+    mysql_close(mysql_conn);
+
+} /* write_message_to_mysql */

 /* -eof- */
```

После того как внесена правка, запустим конфигурирование:

```
$ cd /opt/ngircd-17.1
$ ./configure
```

Изначально пакет ngircd не был рассчитан на использование функций, работающих с MySQL, поэтому их наличие в коде может вызвать ошибки при компилировании и сборке программы.

Для того чтобы не получить ошибку вроде «undefined reference to 'mysql_init', выполнение ld завершилось с кодом возврата 1», перед запуском:

```
$ ./make
```

допишем в /opt/ngircd-17.1/src/ngircd/Makefile в конце параметра CFLAGS через пробел:

```
`/usr/bin/mysql_config --cflags`
```

а в конце LDFLAGS:

```
`/usr/bin/mysql_config --libs`
```

Обратите внимание на то, что используются обратные кавычки. Для сбора специфических для каждой системы параметров запускается mysql_config.

В принципе то же самое можно было бы записать и через переменные окружения, но у кого-то это будет:

```
CFLAGS=-lmysqlclient LDFLAGS=-L/usr/lib/mysql
```

а у кого-то:

```
CFLAGS=-lmysqlclient LDFLAGS=-L/usr/lib64/mysql
```

Да и не всегда переменные считываются (например, если такая переменная уже определена в файле), поэтому вышеуказанный путь правки Makefile самый надёжный. Обратите внимание на полный путь, так как файлов Makefile несколько.

После компиляции мы получили готовую программу с дополнительными функциями, которую, как обычно, надо установить, настроить и запустить.

Замечание 3

Если мы решили вынести вывод в базу данных в отдельную функцию, то почему бы не сделать шаг дальше и не вынести настройки подключения к MySQL-серверу и данные расположения лог-файла журнала в конфигурационный файл? Тогда при их изменении не потребуется перекомпиляции программы. Сделать это просто, допишем в секцию [Global] конфигурационного файла следующие строчки, которые легко можно будет использовать, просто сняв с них знак комментария:

```
;mysql_host=
;mysql_port=
;mysql_login=
;mysql_password=
;mysql_database=
;nomysql_logfile=
```

В файл /opt/ngircd-17.1/src/ngircd/conf.h следует внести такие строчки, обозначающие глобальные переменные:

```
GLOBAL char Conf_mysql_host[HOST_LENGTH];
GLOBAL UINT16 Conf_mysql_port;
GLOBAL char Conf_mysql_login[LOGIN_LENGTH];
GLOBAL char Conf_mysql_password[PASSWORD_LENGTH];
GLOBAL char Conf_mysql_database[DATABASE_LENGTH];
GLOBAL char Conf_nomysql_logfile[LOGFILE_LENGTH];
```

Определения констант – в файл /opt/ngircd-17.1/src/ngircd/defines.h:

```
#define HOST_LENGTH 30
#define LOGIN_LENGTH 20
#define PASSWORD_LENGTH 20
#define DATABASE_LENGTH 50
#define LOGFILE_LENGTH 250
```

Внутри функции Set_Defaults (в /opt/ngircd-17.1/src/ngircd/conf.c) прописать значения по умолчанию:

```
strcpy(Conf_my_host, "localhost");
Conf_ mysql_port = 3306;
strcpy(Conf_my_login, "irc");
strcpy(Conf_my_password, "");
strcpy(Conf_my_database, "irclog");
strcpy(Conf_my_logfile, "/var/log/ngircd/messages.log");
```

А внутри функции Handle_GLOBAL (там же) по аналогии дописать проверку значений:

```
if( strcasecmp( Var, "mysql_host" ) == 0 ) {

    len = strlcpy( Conf_mysql_host, Arg, ↵
        sizeof( Conf_mysql_host ));
    if (len >= sizeof( Conf_mysql_host ))
        Config_Error_TooLong( Line, Var );
    return;
        }
...
if( strcasecmp( Var, "mysql_port" ) == 0 ) {
    port = atol( Arg );
    if( port > 0 && port < 0xFFFF )
```

```
        Conf_mysql_port = (UINT16)port;
    else
        Config_Error( LOG_ERR, "%s, ↵
                line %d (section \"Server\"): ↵
                Illegal MySQL port number %ld!",
NGIRCd_ConfFile, Line, port );
    return;
}
```

После этого внутри файла irc.c можно будет вместо:

```
char *mysql_server = "localhost";
UINT16 mysql_port = 3306;
char *mysql_user = "mysql_user";
char *mysql_password = "mysql_password";
char *mysql_database = "irclog";
const char *Conf_nomysql_logfile="/var/log/ngircd/msg.log";
```

записать обращение к переменным, значения которых будут считаны из конфигурационного файла:

```
char *mysql_server = Conf_mysql_host;
UINT16 mysql_port = Conf_mydql_port;
char *mysql_user = Conf_mysql_login;
char *mysql_password = Conf_mysql_password;
char *mysql_database = Conf_mysql_database;
char *nomysql_logfile = Conf_nomysql_logfile;
```

Настройка MySQL

Предполагается, что читатели уже знакомы с СУБД MySQL [4] и самостоятельно создадут необходимые учетные данные, базу данных, таблицы, выберут кодировку и дадут пользователю право вносить данные в таблицу. Для первого сервера (Unreal) подойдет довольно простая таблица, которую можно создать командой:

```
CREATE TABLE `irclog`.`table`(id INT NULL AUTO_INCREMENT ↵
        PRIMARY KEY ,`message` TEXT NULL);
```

Для второго сервера (ngircd) предлагается более сложный вариант таблицы, позволяющий раздельно хранить ники пользователей, имена каналов и сами сообщения, например:

```
CREATE TABLE `irclog`.`table` (
`id` BIGINT unsigned NOT NULL auto_increment,
`time` TIMESTAMP NOT NULL ,
`channel` TINYTEXT NOT NULL ,
`nick` TINYTEXT NOT NULL ,
`message` TEXT NOT NULL ,
PRIMARY KEY ( `id` )
);
```

Просмотр данных из MySQL

После того как сообщения попали в БД, мы встречаемся с еще одной проблемой: как их просматривать сотрудникам? Среди администраторов и веб-разработчиков она часто решается использованием различных MySQL клиентов или удобного веб-интерфейса phpMyAdmin [6]. Менеджерам и руководству такой инструментарий не дашь, поэтому предложим небольшой скрипт на php. Он будет подключаться к БД, делать запрос на получение сообщений за последние 24 часа и далее печатать полученные данные на экран в виде таблицы. Файл назовем view_messages.php. Указанный пример демонстрационный и содержит минимальный функционал. При желании скрипт можно доработать самостоятельно – добавить проверку на ошибки, изменить форматирование вывода, внести стили, интерактивный поиск и прочее. Файл view_messages.php:

```
<html>
<head>
    <title>Last IRC messages</title>
    <meta name="content" http-equiv="Content-Type" ↵
       CONTENT="text/html; charset=utf-8">
</head>
<body>
<?php
$server="localhost:3306";
$login="irc";
$password="**ircpassword**";
$mysql_link=mysql_connect($server,$login,$password);
mysql_select_db('irclog',$mysql_link);
mysql_query ("SET NAMES utf8");
$color=1;
$request="SELECT * FROM table WHERE unix_timestamp(now()) ↵
       -unix_timestamp(table.time)<1*24*60*60";
$result=mysql_query($request,$mysql_link);
echo "<table border=\"1\">";
echo "<tr align=\"center\" bgcolor=\"lightgrey\"><td>id</td>
          <td>timestamp</td>
          <td>channel</td>
          <td>nick</td>
          <td>message</td></tr>";
while($row=mysql_fetch_array($result))
  {
   if ($color) { echo "<tr bgcolor=\"#FBE0E0\">"; $color=0; }
          else { echo "<tr>"; $color=1; }
   echo "<td>".$row['time']."</td>";
   echo "<td>".$row['channel']."</td>";
   echo "<td>".$row['nick']."</td>";
   echo "<td>".$row['message']."</td>";
   echo "</tr>";
  }
echo "</table>";
?>
</body>
</html>
```

***

В данной статье было наглядно показано, насколько просто, имея лишь небольшие навыки программирования, организовать журналирование сообщений IRC-серверов. Было приведено два практических примера вместе с фрагментами кода, а также разобраны проблемы, с которыми может столкнуться читатель в процессе компиляции исходных кодов.

В качестве направления дальнейшего совершенствования приведенных фрагментов кода хочу порекомендовать читателям обратить внимание на проблемы использования различных кодировок для русского языка; проблемы SQL-инъекций; поведение программы при использовании слишком длинных сообщений, ников и названий каналов. EOF

1. Сайт программы Unreal (The next generation IRCD) – http://www.unrealircd.com.
2. Сайт программы ngIRCd: Next Generation IRC Daemon – http://ngircd.barton.de.
3. Мэтью Н. Основы программирования в Linux/пер. с англ./Н. Мэтью, Р. Стоунс. – 4-е изд., перераб. и доп. – СПб: «БХВ-Петербург», 2009, ISBN 978-5-9775-0289-4.
4. Дюбуа П. MySQL/пер с англ. – 3-е изд. – М.: Издательский дом «Вильямс», 2007, ISBN 5-8459-1119-2, 0-672-32673-6.
5. Установка UnrealIRCD – http://howtoforge.net.ua/ustanovka_unrealircd_anope.
6. PhpMyAdmin MySQL Database Administration Tool – http://www.phpmyadmin.net.
7. RFC 2812 Internet Relay Chat: Client Protocol – http://www.ietf.org/rfc/rfc2812.