Библиотека libevent для асинхронного неблокирующего ввода/вывода: Часть 4. Сетевые приложения

Алексей Снастин
независимый разработчик ПО
начальник отдела

19.07.2011

Тема четвёртой, заключительной статьи - применение libevent в сетевых приложениях.

В этом цикле статей рассматривается библиотека libevent, предназначенная для обработки оповещений о событиях и организации асинхронного ввода/вывода. В первой статье описывается общая структура библиотеки. Вторая статья посвящена основам использования libevent: наибольшее внимание уделено обработке событий. В третьей статье речь идёт о механизмах буферизации ввода/вывода. Тема четвёртой, заключительной статьи - применение libevent в сетевых приложениях.

Введение

В библиотеке libevent, начиная с версии 2.0.2-alpha, появилась поддержка средств анализа, или мониторинга сетевых соединений (connection listeners), основанная на функциональных свойствах объекта evconnlistener. Все функции и типы данных, необходимые для работы с сетевыми соединениями, объявлены в заголовочном файле event2/listener.h.

1. Работа с TCP-соединениями

Объект evconnlistener предоставляет в распоряжение разработчика средства "прослушивания" (мониторинга) и "приёма" (ответа на запрос) входящих TCP-соединений.

1.1. Создание и удаление объекта evconnlistener

Для создания объекта типа evconnlistener используются две функции:

```
struct evconnlistener *evconnlistener_new( struct event_base *base,
                   evconnlistener_cb cb, void *ptr, unsigned flags, int backlog,
                   evutil_socket_t fd );

struct evconnlistener *evconnlistener_new_bind( struct event_base *base,
                   evconnlistener_cb cb, void *ptr, unsigned flags, int backlog,
                   const struct sockaddr *sa, int socklen );
```

Обе функции возвращают указатель на созданный объект "монитор соединений" (evconnlistener). Этот монитор использует уже известный разработчику набор событий base (тип event_base) для отслеживания появления новых запросов TCP-соединений на заданном "прослушиваемом" сокете. При поступлении запроса на установление TCP-соединения активизируется специальная функция обратного вызова cb, определяемая разработчиком. Если вместо указателя на функцию обратного вызова в качестве аргумента передаётся значение NULL, то создаваемый монитор соединений считается "запрещённым", то есть, фактически он не работает до тех пор, пока для него не будет установлена функция обратного вызова. В функцию обратного вызова будет передан указатель ptr.

Аргумент flags управляет функциональными свойствами объекта evconnlistener. Значение определяется комбинацией следующих флагов:

- LEV_OPT_LEAVE_SOCKETS_BLOCKING - по умолчанию сокет, связываемый с монитором соединений, переводится в неблокируемое состояние. Этот флаг позволяет сохранить возможность блокировки данного сокета;
- LEV_OPT_CLOSE_ON_FREE - если этот флаг установлен, то при удалении объекта evconnlistener автоматически закрывается связанный с ним сокет;
- LEV_OPT_CLOSE_ON_EXEC - если этот флаг установлен, то для связываемого с создаваемым монитором сокета устанавливается флаг close-on-exec, если это возможно на текущей платформе;
- LEV_OPT_REUSEABLE - пометить связываемый с монитором сокет, как "многократно используемый", то есть отменить таймаут, определяющий интервал времени, который должен пройти до того момента, когда на этом же порте можно будет открыть новый прослушиваемый сокет;
- LEV_OPT_THREADSAFE - включить механизм блокировок для данного монитора соединений, чтобы обеспечить его безопасное использование в многопоточной среде.

Аргумент backlog определяет максимальное количество ожидающих обработки соединений, которые в любой момент времени могут находиться в состоянии "запрос услышан, но пока ещё не принят" (not-yet-accepted). Более подробную информацию об установке значения этого параметра следует искать в документации по системному вызову listen() для используемой платформы. Если значением backlog является отрицательное число, то производится попытка подобрать оптимальное значение для этого параметра. Если backlog равен нулю, то считается, что для заданного сокета ранее уже был выполнен системный вызов listen(), и значение backlog определено разработчиком.

Удалить объект evconnlistener и освободить выделенную ему память можно с помощью функции

```
void evconnlistener_free( struct evconnlistener *ecl );
```

1.2. Функция обратного вызова для объекта evconnlistener

При получении запроса на установление нового соединения активизируется функция обратного вызова, для которой определён специальный тип:

```
typedef void (*evconnlistener_cb)( struct evconnlistener *listener,
             evutil_socket_t socket, struct sockaddr *addr, int len, void *ptr );
```

Аргумент listener указывает на монитор соединений, который обрабатывает запросы. Новый сокет определяется аргументом socket. В структуре, на которую ссылается аргумент addr, содержится адрес источника запроса на соединение, а в аргументе len передаётся длина этого адреса. Если необходимы дополнительные данные, то для их передачи предназначен указатель ptr, тот самый, который был передан пользователем в функцию создания монитора соединений.

В предыдущем разделе было отмечено, что монитор соединений (объект evconnlistener) может быть создан без указания функции обратного вызова. В этом случае функцию обратного вызова можно установить следующим образом:

```
void evconnlistener_set_cb( struct evconnlistener *listener,
                            evconnlistener_cb callback, void *arg );
```

1.3. Основные операции с объектом evconnlistener

Работу монитора соединений можно временно приостановить и возобновить с помощью пары функций

```
int evconnlistener_disable( struct evconnlistener *ecl );
int evconnlistener_enable( struct evconnlistener *ecl );
```

Кроме того, можно получить некоторую информацию о внутренних элементах монитора соединений: связанный с заданным монитором сокет возвращает функция

```
evutil_socket_t evconnlistener_get_fd( struct evconnlistener *ecl );
```

а указатель на набор событий, обрабатываемых заданным монитором, возвращает функция

```
struct event_base *evconnlistener_get_base( struct evconnlistener *ecl );
```

Для обработки ошибок разработчик может написать собственную функцию обратного вызова, для которой также определён специализированный тип:

```
typedef void (*evconnlistener_errorcb)( struct evconnlistener *listener, void *ptr );
```

Устанавливается обработчик ошибок для заданного монитора соединений с помощью функции

```
void evconnlistener_set_error_cb( struct evconnlistener *ecl,
                                  evconnlistener_errorcb error_callback );
```

1.4. Пример использования объекта evconnlistener

Для демонстрации работы монитора соединений самым простым и понятным вариантом является реализация эхо-сервера. В Листинге 1 приведён исходный код, в котором помимо

объекта evconnlistener использованы практически все объекты библиотеки libevent, которые рассматривались в предыдущих статьях цикла.

Листинг 1. Пример реализации эхо-сервера

```
#include <event2/listener.h>
#include <event2/bufferevent.h>
#include <event2/buffer.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

/* Функция обратного вызова для события: данные готовы для чтения в buf_ev */
static void echo_read_cb( struct bufferevent *buf_ev, void *arg )
{
  struct evbuffer *buf_input = bufferevent_get_input( buf_ev );
  struct evbuffer *buf_output = bufferevent_get_output( buf_ev );
  /* Данные просто копируются из буфера ввода в буфер вывода */
  evbuffer_add_buffer( output, input );
}

static void echo_event_cb( struct bufferevent *buf_ev, short events, void *arg )
{
  if( events & BEV_EVENT_ERROR )
    perror( "Ошибка объекта bufferevent" );
  if( events & (BEV_EVENT_EOF | BEV_EVENT_ERROR) )
    bufferevent_free( buf_ev );
}

static void accept_connection_cb( struct evconnlistener *listener,
                   evutil_socket_t fd, struct sockaddr *addr, int sock_len,
                   void *arg )
{
  /* При обработке запроса нового соединения необходимо создать для него
     объект bufferevent */
  struct event_base *base = evconnlistener_get_base( listener );
  struct bufferevent *buf_ev = bufferevent_socket_new( base, fd, BEV_OPT_CLOSE_ON_FREE );

  bufferevent_setcb( buf_ev, echo_read_cb, NULL, echo_event_cb, NULL );
  bufferevent_enable( buf_ev, (EV_READ | EV_WRITE) );
}

static void accept_error_cb( struct evconnlistener *listener, void *arg )
{
  struct event_base *base = evconnlistener_get_base( listener );
  int error = EVUTIL_SOCKET_ERROR();
  fprintf( stderr, "Ошибка %d (%s) в мониторе соединений. Завершение работы.\n",
           error, evutil_socket_error_to_string( error ) );
  event_base_loopexit( base, NULL );
}

int main( int argc, char **argv )
{
  struct event_base *base;
  struct evconnlistener *listener;
  struct sockaddr_in sin;
  int port = 9876;

  if( argc > 1 )  port = atoi( argv[1] );
  if( port < 0 || port > 65535 )
  {
    fprintf( stderr, "Задан некорректный номер порта.\n" );
    return -1;
```

```
  }

  base = event_base_new();
  if( !base )
  {
    fprintf( stderr, "Ошибка при создании объекта event_base.\n" );
    return -1;
  }

  memset( &sin, 0, sizeof(sin) );
  sin.sin_family = AF_INET;    /* работа с доменом IP-адресов */
  sin.sin_addr.s_addr = htonl( INADDR_ANY );  /* принимать запросы с любых адресов */
  sin.sin_port = htons( port );

  listener = evconnlistener_new_bind( base, accept_connection_cb, NULL,
                                      (LEV_OPT_CLOSE_ON_FREE | LEV_OPT_REUSEABLE),
                                      -1, (struct sockaddr *)&sin, sizeof(sin) );
  if( !listener )
  {
    perror( "Ошибка при создании объекта evconnlistener" );
    return -1;
  }
  evconnlistener_set_error_cb( listener, accept_error_cb );

  event_base_dispatch( base );
  return 0;
}
```

2. Поддержка DNS

Библиотека libevent предоставляет несколько API-интерфейсов для выполнения операций разрешения и преобразования доменных имён, а также инструменты для реализации простых DNS-серверов. Все необходимые функции, структуры и константы определены в заголовочных файлах <event2/util.h> и <event2/dns.h>. В данной статье рассматриваются только средства высокого уровня для работы с доменными именами.

2.1. Стандартная реализация интерфейса getaddrinfo()

Для тех платформ, на которых отсутствует функция getaddrinfo(), библиотека libevent предлагает собственную реализацию этого стандартного интерфейса (подробности см. в RFC 3493, раздел 6.1).

```
int evutil_getaddrinfo( const char *nodename, const char *servicename,
                        const struct evutil_addrinfo *hints,
                        struct evutil_addrinfo **res );
```

Эта функция пытается разрешить (resolve) и преобразовать данные, переданные в аргументах nodename (имя хоста) и servicename (обозначение сервиса), в соответствии с правилами, заданными в аргументе hints, и создать связанный список структур evutil_addrinfo (эта структура определена в заголовочном файле <event2/util.h>), место хранения которого указано в аргументе *res. При успешном завершении функция возвращает 0, ненулевое значение сообщает об ошибке.

Имя хоста nodename может быть задано в форме литерального IPv4-адреса ("127.0.0.1") или в форме литерального IPv6-адреса ("::1"), или в форме доменного имени ("www.google.com"). Обозначение сервиса servicename передаётся в символьной форме

("http") или в виде строки, содержащей номер порта ("80"). Либо аргумент nodename, либо аргумент servicename может содержать значение NULL, но не одновременно.

Правила поиска адресов hints определяются комбинацией флагов, набор которых определён в заголовочном файле <event2/util.h>.

2.2. Пример преобразования имени хоста с установлением блокирующего соединения

Ниже в Листинге 2 приведён пример исходного кода, в котором используется описанная выше функция определения адреса по имени хоста.

Листинг 2. DNS-клиент, устанавливающий блокирующее соединение

```
#include <event2/util.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/socket.h>

evutil_socket_t get_tcp_socket_for_host( const char *hostname, ev_uint16_t port )
{
  char port_buf[6];
  struct evutil_addrinfo hints;
  struct evutil_addrinfo *result = NULL;
  int error;
  evutil_socket_t sckt;

  evutil_snprintf( port_buf, sizeof(port_buf), "%d", (int)port );

  /* Создание набора правил для выполнения поиска */
  memset( &hints, 0, sizeof(hints) );
  hints.ai_family = AF_UNSPEC; /* рассматривать и IPv4, и IPv6 адреса */
  hints.ai_socktype = SOCK_STREAM; /* требуется потоковое соединение */
  hints.ai_protocol = IPPROTO_TCP; /* требуется TCP-сокет */
  /* IPv4-адреса возвращаются, только если для локального хоста сконфигурирован
     как минимум один IPv4-адрес; то же - для IPv6-адресов */
  hints.ai_flags = EVUTIL_AI_ADDRCONFIG;

  /* Выполнение операции поиска hostname */
  error = evutil_getaddrinfo( hostname, port_buf, &hints, &result );
  if( error < 0 )
  {
    fprintf( stderr, "Ошибка при выполнении операции разрешения '%s': %s\n",
             hostname, evutil_gai_strerror( error ) );
    return -1;
  }

  /* Если ошибок не было, то в result должен содержаться по крайней мере
     один найденный адрес */
  assert( result );
  /* Используется самый первый (или единственный) полученный адрес */
  sckt = socket( result->ai_family, result->ai_socktype, result->ai_protocol );

  if( sckt < 0 )
    return -1;
  if( connect( sckt, result->ai_addr, result->ai_addrlen ) )
  {
    /* Следует ещё раз отметить, что в данном случае устанавливается
```

```
     * блокирующее соединение. Если бы соединение было неблокирующим,
     * то потребовалась бы дополнительная специальная обработка
     * некоторых ошибок (например, EINTR, EAGAIN и т.д.) */
    EVUTIL_CLOSESOCKET( sckt );
    return -1;
  }
  return sckt;
}
```

Заключение

Средства для разработки сетевых приложений библиотеки libevent обеспечивают переносимость и удобство использования на различных платформах. Следует особо отметить гибкость и широкий диапазон настроек API-интерфейсов для работы с TCP-соединениями и для выполнения DNS-запросов.

В данной статье рассмотрено применение библиотеки libevent в сетевых приложениях.

Об авторе

Алексей Снастин

Алексей Снастин - независимый разработчик ПО, консультант и переводчик с английского языка технической и учебной литературы по ИТ. Принимал участие в разработке сетевых офисных приложений типа клиент/сервер на языке C в среде Linux.