# Программирование сетевых приложений (TCP/IP) на C/C++ (http://blog.hoxnox.com/inet/tcp_programming.html)

Date Пн 02 Апрель 2012 Tags UML (http://blog.hoxnox.com/tag/uml.html) / fork (http://blog.hoxnox.com/tag/fork.html) / exec (http://blog.hoxnox.com/tag/exec.html) / system (http://blog.hoxnox.com/tag/system.html) / posix_spawn (http://blog.hoxnox.com/tag/posix_spawn.html) / C (http://blog.hoxnox.com/tag/c.html)

## Простейшие примеры

### TCP/IP

Сервер

```
/* определяет типы данных */
#include <sys/types.h>
/* "Главный" по сокетам */
#include <sys/socket.h>
/* sockaddr_in struct, sin_family, sin_port, in_addr_t, in_port_t, ...*/
#include <netinet/in.h>

#include <stdio.h>
#include <memory.h>
#include <string.h>
#include <errno.h>

/**@brief Получает от клиента последовательность байт, не длиннее 30 и печатает её на экран по
 * завершении соединения. Клиенту отправляет "Hi, dear!"*/
int main(int argc, char * argv)
{
	/*создаём сокет*/
	int s = socket(AF_INET, SOCK_STREAM, 0);
	if(s < 0)
	{
		perror("Error calling socket");
		return 0;
	}

	/*определяем прослушиваемый порт и адрес*/
	struct sockaddr_in addr;
	addr.sin_family = AF_INET;
	addr.sin_port = htons(18666);
	addr.sin_addr.s_addr = htonl(INADDR_ANY);
	if( bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0 )
	{
		perror("Error calling bind");
		return 0;
	}

	/*помечаем сокет, как пассивный - он будет слушать порт*/
	if( listen(s, 5) )
	{
		perror("Error calling listen");
		return 0;
	}

	/*начинаем слушать, для соединения создаём другой сокет, в котором можем общаться.*/
	int s1 = accept(s, NULL, NULL);
	if( s1 < 0 )
	{
		perror("Error calling accept");
		return 0;
	}

	/*читаем данные из сокета*/
	char buffer[31];
	int counter = 0;
	for(;;)
	{
		memset(buffer, 0, sizeof(char)*31);
		/*следует помнить, что данные поступают неравномерно*/
		int rc = recv(s1, buffer, 30, 0);
		if( rc < 0 )
		{
			/*чтение может быть прервано системным вызовом, это нормально*/
			if( errno == EINTR )
				continue;
			perror("Can't receive data.");
			return 0;
		}
		if( rc == 0 )
```

```
                        break;
                printf("%s\n", buffer);
        }
        char response[] = "Hi, dear!";
        if( sendto( s1, response, sizeof(response), 0, (struct sockaddr *)&addr, sizeof(addr) ) < 0 )
                perror("Error sending response");
        printf("Response send\n");
        return 0;
}
```

Клиент

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
/* hton, ntoh и проч. */
#include <arpa/inet.h>
#include <memory.h>
#include <stdio.h>

int main(int argc, char * argv[])
{

	/*объявляем сокет*/
	int s = socket( AF_INET, SOCK_STREAM, 0 );
	if(s < 0)
	{
		perror( "Error calling socket" );
		return 0;
	}

	/*соединяемся по определённому порту с хостом*/
	struct sockaddr_in peer;
	peer.sin_family = AF_INET;
	peer.sin_port = htons( 18666 );
	peer.sin_addr.s_addr = inet_addr( "127.0.0.1" );
	int result = connect( s, ( struct sockaddr * )&peer, sizeof( peer ) );
	if( result )
	{
		perror( "Error calling connect" );
		return 0;
	}

	/*посылаем данные
	 *
	 * Если быть точным, данные не посланы, а записаны где-то в стеке, когда и как они будут
	 * отправлены реализации стека TCP/IP виднее. Зато мы сразу получаем управление, не
	 * дожидаясь у моря погоды.*/
	char buf[] = "Hello, world!";
	result = send( s, "Hello, world!", 13, 0);
	if( result <= 0 )
	{
		perror( "Error calling send" );
		return 0;
	}
	/* закрываем соединения для посылки данных */
	if( shutdown(s, 1) < 0)
	{
		perror("Error calling shutdown");
		return 0;
	}

	/* читаем ответ сервера */
	fd_set readmask;
	fd_set allreads;
	FD_ZERO( &allreads );
	FD_SET( 0, &allreads );
	FD_SET( s, &allreads );
	for(;;)
	{
		readmask = allreads;
		if( select(s + 1, &readmask, NULL, NULL, NULL ) <= 0 )
		{
			perror("Error calling select");
			return 0;
		}
		if( FD_ISSET( s, &readmask ) )
		{
			char buffer[20];
```

```
                        memset(buffer, 0, 20*sizeof(char));
                        int result = recv( s, buffer, sizeof(buffer) - 1, 0 );
                        if( result < 0 )
                        {
                                perror("Error calling recv");
                                return 0;
                        }
                        if( result == 0 )
                        {
                                perror("Server disconnected");
                                return 0;
                        }
                        if(strncmp(buffer, "Hi, dear!", 9) == 0)
                                printf("Got answer. Success.\n");
                        else
                                perror("Wrong answer!");
                }
                if( FD_ISSET( 0, &readmask ) )
                {
                        printf( "No server response" );
                        return 0;
                }
        }
        return 0;
}
```

## UDP

# Что следует иметь ввиду при разработке с TCP

1. TCP не выполняет опрос соединения (не поможет даже keep alive - он нужен для уборки мусора, а не контроля за состоянием соединения). Исправляется на прикладном уровне, например, реализацией пульсации. Причем на дополнительном соединении.
2. Задержки при падении хостов, разрыве связи.
3. Необходимо следить за порядком получения сообщений.
4. Заранее неизвестно сколько данных будет прочитано из сокета. Может быть прочитано несколько пакетов сразу!
5. Надо быть готовым ко всем внештатным ситуациям:
    - постоянный или временный сбой сети
    - отказ принимающего приложения
    - аварийный сбой самого хоста на принимающей стороне
    - неверное поведение хоста на принимающей стороне
    - учитывать особенности сети функционирования приложения (глобальная или локальная)

# OSI и TCP/IP

| **OSI** | **TCP/IP** |
|---|---|
| Прикладной уровень | Прикладной уровень |
| Уровень представления | |
| Сеансовый уровень | Транспортный уровень |
| Транспортный уровень | Межсетевой уровень |
| Сетевой уровень | Интерфейсный уровень |
| Канальный уровень | |
| Физический уровень | |

# Порты

| **Порт** | **Кем контроллируется** |
|---|---|
| 0 - 1023 | Контролируется IANA |
| 1024 - 49151 | Регистрируется в IANA |
| 49152 - 65535 | Эфимерные |

Полный список зарегистрированных портов расположен по адресу: http://www.isi.edu/in-notes/iana/assignment/port-numbers (http://www.isi.edu/in-notes/iana/assignment/port-numbers). Подать заявку на получение хорошо известного или зарегистрированного номера порта можно по адресу http://www.isi.edu/cgi-bin/iana/port-numbers.pl (http://www.isi.edu/cgi-bin/iana/port-numbers.pl).

# Состояние TIME-WAIT

После активного закрытия для данного конкретного соединения стек входит в состояние TIME-WAIT на время 2MSL (максимальное время жизни пакета) для того, чтобы

1. заблудившийся пакет не попал в новое соединение с такими же параметрами.
2. если потерялся ACK, подтверждающий закрытие соединения, с активной стороны, пассивная снова пощлёт FIN, активная, игнорируя TIME-WAIT уже закрыла соединение, поэтому пассивная сторона получит RST.

Отключение состояния TIME-WAIT крайне не рекомендуется, так как это нарушает безопасность TCP соединения, тем не менее существует возможность сделать это - опция сокета SO_LINGER.

Штатная ситуация - перезагрузка сервера может пострадать из-за наличия TIME-WAIT. Эта проблема решается заданием опции SO_REUSEADDR.

# Отложенное подтверждение и алгоритм Нейгла.

Алгоритм Нейгла используется для предотвращения забивания сети мелкими пакетами и имеет очень простую формулировку - запрещено посылать второй маленький пакет до тех пор, пока не придет подтверждение на первый. Тем не менее данные отправляются при выполнении хотя бы одного из следующих условий:

- можно послать полный сегмент размером MSS (максимальный размер сегмента)
- соединение простаивает, и можно опустошить буфер передачи
- алгоритм Нейгла отключен, и можно опустошить буфер передачи
- есть срочные данные для отправки
- есть маленьки сегмент, но его отправка уже задержана на достаточно длительное время (таймер терпения persist timer на тайм-аут ретрансмиссии RTO)
- окно приема, объявленное хостом на другом конце, открыто не менее чем на половину
- необходимо повторно передать сегмент
- требуется послать ACK на принятые данные
- нужно объявить об обновлении окна

Кроме того, существует отложенное подтверждение. При этом хост, получивший сегмент, старается задержать отправку ACK, чтобы послать её вместе с данными.

Алгоритм Нейгла в купе с отложенным подтверждением в резонансе дают нежелательные задержки. Поэтому часто его отключают. Отключение алгоритма Нейгла производится заданием опции TCP_NODELAY

```
const int on = 1;
setsockopt( s, IPPROTO_TCP, TCP_NODELAY, &on, sizeof( on ) );
```

Но более правильным было бы проектировать приложение таким образом, чтобы было как можно меньше маленьких блоков. Лучше писать большие. Для этого можно объединять данные самостоятельно, а можно пользоваться аналогом write, работающим с несколькими буферами:

```
#include <sys/uio.h>
ssize_t writev( int fd, const struct iovec *iov, int cnt );
ssize_t readv( int fd, const struct iovec *iov, int cnt );

// Возвращают число переданных байт или -1 в случае ошибки

struct iovec {
      char *iov_base; /* Адрес начала буфера*/
      size_t iov_len; /* Длина буфера*/
}
```

В Winsock используется

```
int WSAAPI WSAsend( SOCKET s,
                    LPWSABUF,
                    DWORD cnt,
                    LPDWORD sent,
                    DWORD flags,
                    LPWSAOVERLAPPED ovl,
                    LPSWSAOVERLAPPED_COMPLETION_ROUTINE func );

// Возвращает 0 в случае успеха, в противном случае SOCKET_ERROR

typedef struct _WSABUF {
      u_long len;     /* Длина буфера */
      char FAR * buf; /* Указатель на начало буфера */
} WSABUF, FAR * LPWSABUF;
```

# Таймаут при вызове connect

## alarm

```
void alarm_hndlr( int sig )
{
      return;
}

int main( int argc, char **argv )
{
      // ...
      signal( SIGALRM, alarm_hndlr );
      alarm( 5 );
      int rc = connect( s, (struct sockaddr * )&peer, sizeof( peer ) );
      alarm( 0 );
      if( rc < 0 )
      {
              if( errno == EINTR )
                      error( 1, 0, "Timeout\n" );
      // ...
}
```

Способ простой, но имеет ряд проблем.

- Таймер, используемый в вызове alarm не должен больше нигде применяться.
- Перезапустить connect сразу не получится. Необходимо будет подождать, закрыть и заново открыть сокет.
- Некоторые системы могут возобновлять connect

## Неблокирующие соединения

Неплохо об асинхронной работе с сокетами расписано тут: http://www.wangafu.net/~nickm/libevent-book/01_intro.html (http://www.wangafu.net/~nickm/libevent-book/01_intro.html)

Суть в том, чтобы использовать неблокирующие сокеты и следить за ними с помощью системных вызовов. Жалко только, что переносимое решение "из коробки" можно реализовать только с тупым и медленным select.

## select

```
int main( int argc, char **argv )
{
	struct sockaddr_in peer;
	INIT() //
	set_address( argv[1], argv[2], &peer, "tcp" );
	SOCKET s = socket( AAF_INET, SOCK_STREAM, 0 );
	if( !isvalidsock( s ) )
		error( 1, errno, "Socket colling error" );

	/* Добавляет флаг "не блокирующий" к флагам сокета (как дескриптора файла)*/
	int flags = fcntl( s, F_GETFL, 0 ) );
	if( flags < 0 )
		error( 1, errno, "Error calling fcntl(F_GETFL)" );
	if( fcntl( s, F_SETFL, flags | O_NONBLOCK ) < 0 )
		error( 1, errno, "Error calling fcntl(F_SETFL)" );

	int rc = connect( s, (struct sockaddr * )&peer, sizeof( peer ) );
	if( rc < 0 && errno != EINPROGRESS )
		error( 1, errno, "Error calling connect" );

	if( rc == 0 ) // вдруг уже не надо ждать
	{
		if( fcntl( s, F_SETFL, flags ) < 0 )
			error( 1, errno, "Error calling fcntl (flags recovery)");
		client( s, &peer );
		return 0;
	}

	/* Если ждать надо, ждем с помощью select'а*/
	fd_set rdevents;
	fd_set wrevents;
	fs_set exevents;
	FD_ZERO( &rdevents );
	FD_SET( s, &rdevents );
	wrevents = rdevents;
	exevents = rdevents;
	struct timeval tv;
	tv.tv_sec = 5;
	tv.tv_usec = 0;
	rc = select( s + 1, &rdevents, &wrevents, &exevents, &tv );
	if( rc < 0 )
		error( 1, errno, "Error calling select" );
	else if( rc == 0 )
		error( 1, 0, "Connection timeout" );
	else if( isconnected( s, &rdevents, &wrevents, &exevents ) )
	{
		if( fcntl( s, F_SETFL, flags ) < 0 )
			error( 1, errno, "Error calling fcntl (flags recovery)" )
		client( s, &peer );
	}
	else
		error( 1, errno, "Error calling connect" );
	return 0;
}

/* В UNIX и WINDOWS разные методы уведомления об успешной попытке соединения, поэтому проверка
вынесена в отдельную функцию.*/

// UNIX
int isconnected( SOCKET s, fd_set *rd, fd_set *wr, fd_set *ex)
{
	int err;
	int len = sizeof( err );
	errno = 0; /*предполагаем, что ошибки нет*/
	if( !FD_ISSET( s, rd ), !FD_ISSET( s, wr ) )
		return 0;
	if( getsockopt( s, SOL_SOCKET, SO_ERROR, &err, &len ) < 0 )
```

```
                return 0;
        errno = err; /* если мы не соединились */
        return err == 0;
}

// Windows
int isconnected( SOCKET s, fd_set *rd, fd_set *wr, fd_set *ex)
{
        WSASetLastError( 0 );
        if( !FD_ISSET( s, rd ) && !FD_ISSET( s, wr ) )
                return 0;
        if( !FD_ISSET( s, ex ) )
                return 0;
        return 1;
}
```

## libevent

**TODO:**libevent

# Следует обнулять структуру sockaddr_in

Для дополнения структуры до 16 байт, в ней имеется блок sin_zero. Он должен быть нулевым.

# Преобразование порядка байт

```
0x12345678 => 12 34 56 78 - прямой порядок (big endian)
0x12345678 => 78 56 34 12 - братный порядок (little endian)
```

Сетевой порядок байт всегда прямой. Для преобразования числа из порядка платформы в сетевой порядок используют hton (host to network) группу функций (htonl, htons). Обратно, соответственно ntoh.

# Разрешение имен и служб

см. gethostbyaddr, gethostbyname, gethostbyname2 (IPv6), getservbyname, getservbyport

# boost::asio

Данная библиотека берет на себя ассинхронную передачу и получение сообщений по сети. Реализуется шаблоном проектирования Proactor. На каждое событие получения ответа или отправки запроса вешается обработчик. Операции вводавывода осуществляются асинхронно, но события формируются в очередь и выполняются последовательно. Таким образом, вы получаете возможность писать код не заморачиваясь на аспектах многопоточного программирования, собирая сливки с асинхронной передачи данных по сети.

Лучше всего о boost::asio написано в документации. В качестве простого примера выступает проект "Курсор".

# Количество открытых соединений

Итак, допустим мы хотим удержать максимально возможное число соединений. Сколько же это? Первым параметром, в который мы уткнёмся - число одновременно открытых файлов на процесс операционной системы.

```
ulimit -a | grep open
```

Следующая команда меняет это число в пределах сессии:

```
ulimit -n 1048576
```

Откуда берется константа - не знаю. В моей системе было так. Следует отметить, что для исполнения нужны специальные права.

Далее следует обратить внимание на количество оперативной памяти и иметь ввиду, что каждое открытое соединение откушает около 64Кб unswappable памяти. Таким образом, в моём случае:

```
python -c"print(`free | awk '/-\/+.*/ {print $4}'`/64)"
```

эта цифра была более 243422.0625. Значит будем пробовать открыть 200000 соединений. Не совсем всё так просто. Существует целый ряд настроек в net.ipv4.tcp (/proc/sys/net/ipv4).

- tcp_mem - векторная величина (минимум, нагрузка, максимум), характеризующая общие настройки потребления памяти для протокола TCP. Измеряется в страницах (обычно страница - 4Кб). До минимума операционная система ничего не делает, при среднем - старается ограничить использование памяти. Максимум - максимальное число страниц, разрешённое для всех TCP сокетов. Так как мы замахиваемся на 200000 соединений, нам надо бы минимум (200000*64)/4 = 3200000. Зачем заставлять нервничать операционную систему.

  ```
  sudo sysctl -w net.ipv4.tcp_mem="3200000 3300000 3400000"
  ```

- tcp_syncookies=0 - согласно рекомендациям разработчиков ядра (http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt (http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt)) этот режим лучше отключить на сильной нагрузке
- net.ipv4.netfilter.ip_conntrack_max=1048576 - максимальное число соединений для работы механизма connection tracking (используется, к примеру в iptables).
- net.ipv4.tcp_no_metrics_save=1 не сохранять результаты измерений TCP соединения в кеше при его закрытии. В некоторых случаях помогает повысить производительность.
- net.ipv4.tcp_tw_reuse=1 разрешаем повторное использование TIME-WAIT сокетов в случаях, если протокол считает это безопасным.
- net.core.somaxconn=200000 максимальное число открытых сокетов, ждущих соединения.
- net.core.netdev_max_backlog=1000 максимальное количество пакетов в очереди на обработку, если интерфейс получает пакеты быстрее, чем ядро может их обработать.

Итоговый скрипт подстройки ОС Linux

```
#!/bin/bash

ulimit -n 1048576
MAX_SOCKS=`python -c"print(\`free | awk '/-\/+.*/ {print $4}'\`//64"`
let MAX_SOCKS_MIDLE=$MAX_SOCKS+1000
let MAX_SOCKS_UP=$MAX_SOCKS+2000
sysctl -w net.ipv4.tcp_mem="$MAX_SOCKS $MAX_SOCKS_MIDLE $MAX_SOCKS_UP"
sysctl -w net.ipv4.tcp_syncookies=0
sysctl -w net.ipv4.netfilter.ip_conntrack_max=1048576
sysctl -w net.ipv4.tcp_no_metrics_save=1
sysctl -w net.ipv4.somaxconn=$MAX_SOCKS
sysctl -w net.ipv4.core.netdev_max_backlog=1000
sysctl -w net.ipv4.tcp_tw_recycle=0
sysctl -w net.ipv4.tcp_tw_reuse=0
```

## Отслеживание соединений

Выше для отслеживания состояния соединения мы использовали select. К сожалению он имеет жесткое ограничение по количеству отслеживаемых элементов. Существуют другие подходы, но к сожалению они не кросс-платформенные. Можно использовать библиотеку libevent. Заявляется, что она использует максимально эффективную реализацию для данной системы, но писать придётся в событийной модели.

Ещё один интересный момент. Стек TCP/IP Linux (3.3.8) в рамках одного потока идентифицирует соединение не только по локальному адресу, но и по удаленному. Это позволяет а одном потоке использовать два сокета с одним локальным адресом и портом, но разными удаленными адресами. К сожалению, при использовании большого количества соединений возникают различные сайд-эффекты. В связи с этим придется вручную контроллировать раздачу портов для reusable сокетов.

# Библиография

Эффективное программирование TCP/IP ( Йон Снейдер ) http://ru.fractalizer.ru/frpost_197 (http://ru.fractalizer.ru/frpost_197)/настройка-ядра-linux-для-поддержки-большо/