

Задание: Протестировать работу сокетов tcp при различных настройках setsockopt.

Task: Test tcp sockets with different setsockopt settings.

### Ход работы

На сайте <https://www.binarytides.com/server-client-example-c-sockets-linux/> представлены исходные файлы TCP сервера-клиента на C. Их и возьмем за основу собственных реализаций.

#### TCP-сервер:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<pthread.h>
#include <netinet/tcp.h>

//the thread function
void *connection_handler(void *);

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , *new_sock;
    struct sockaddr_in server , client;

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

#define OPTION1 SOL_SOCKET, SO_KEEPALIVE
#define OPTION2 IPPROTO_TCP, TCP_NODELAY
#define OPTION3 IPPROTO_TCP, TCP_QUICKACK
#define OPTION4 IPPROTO_TCP, TCP_FASTOPEN
#define OPTION5 SOL_SOCKET, SO_REUSEADDR
    int opt_val = 1;
    if (setsockopt(socket_desc, OPTION5, &opt_val, sizeof(opt_val))) {
        perror("setsockopt");
        exit(1);
    }

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );
```

```

//Bind
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
{
    //print the error message
    perror("bind failed. Error");
    return 1;
}
puts("bind done");

//Listen
listen(socket_desc , 3);

//Accept and incoming connection
puts("Waiting for incoming connections...");
c = sizeof(struct sockaddr_in);
while( (client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c))
)
{
    puts("Connection accepted");

    pthread_t sniffer_thread;
    new_sock = malloc(1);
    *new_sock = client_sock;

    if( pthread_create( &sniffer_thread , NULL ,  connection_handler , (void*)
new_sock) < 0)
    {
        perror("could not create thread");
        return 1;
    }

    //Now join the thread , so that we dont terminate before the thread
    //pthread_join( sniffer_thread , NULL);
    puts("Handler assigned");
}

if (client_sock < 0)
{
    perror("accept failed");
    return 1;
}

return 0;
}

/*
 * This will handle connection for each client
 */
void *connection_handler(void *socket_desc)
{
    //Get the socket descriptor

```

```

int sock = *(int*)socket_desc;
int read_size;
char *message , client_message[2000];

//Send some messages to the client
message = "Greetings! I am your connection handler\n";
write(sock , message , strlen(message));

message = "Now type something and i shall repeat what you type \n";
write(sock , message , strlen(message));

//Receive a message from client
while( (read_size = recv(sock , client_message , 2000 , 0)) > 0 )
{
    //Send the message back to client
    write(sock , client_message , strlen(client_message) + 1);
}

if(read_size == 0)
{
    puts("Client disconnected");
    fflush(stdout);
}
else if(read_size == -1)
{
    perror("recv failed");
}

//Free the socket pointer
free(socket_desc);

return 0;
}

```

## TCP-клиент

```

#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/time.h>

int main(int argc , char *argv[])
{
    int sock;
    struct sockaddr_in server;
    char    message[1000]    =    "asdadqwDSDQWD13123ewadSADASDA"    ,
    server_reply[2000];

    //Create socket
    sock = socket(AF_INET , SOCK_STREAM , 0);

```

```

if (sock == -1)
{
    printf("Could not create socket");
}
puts("Socket created");

server.sin_addr.s_addr = inet_addr("127.0.0.1");
server.sin_family = AF_INET;
server.sin_port = htons( 8888 );

//Connect to remote server
if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
{
    perror("connect failed. Error");
    return 1;
}

puts("Connected\n");

for(int j = 0; j < 1000; j++){
    struct timeval stop, start;
    gettimeofday(&start, NULL);
    for(int i = 0; i < 500; i++)
    {
        //Send some data
        if( send(sock , message , strlen(message) , 0) < 0)
        {
            puts("Send failed");
            return 1;
        }

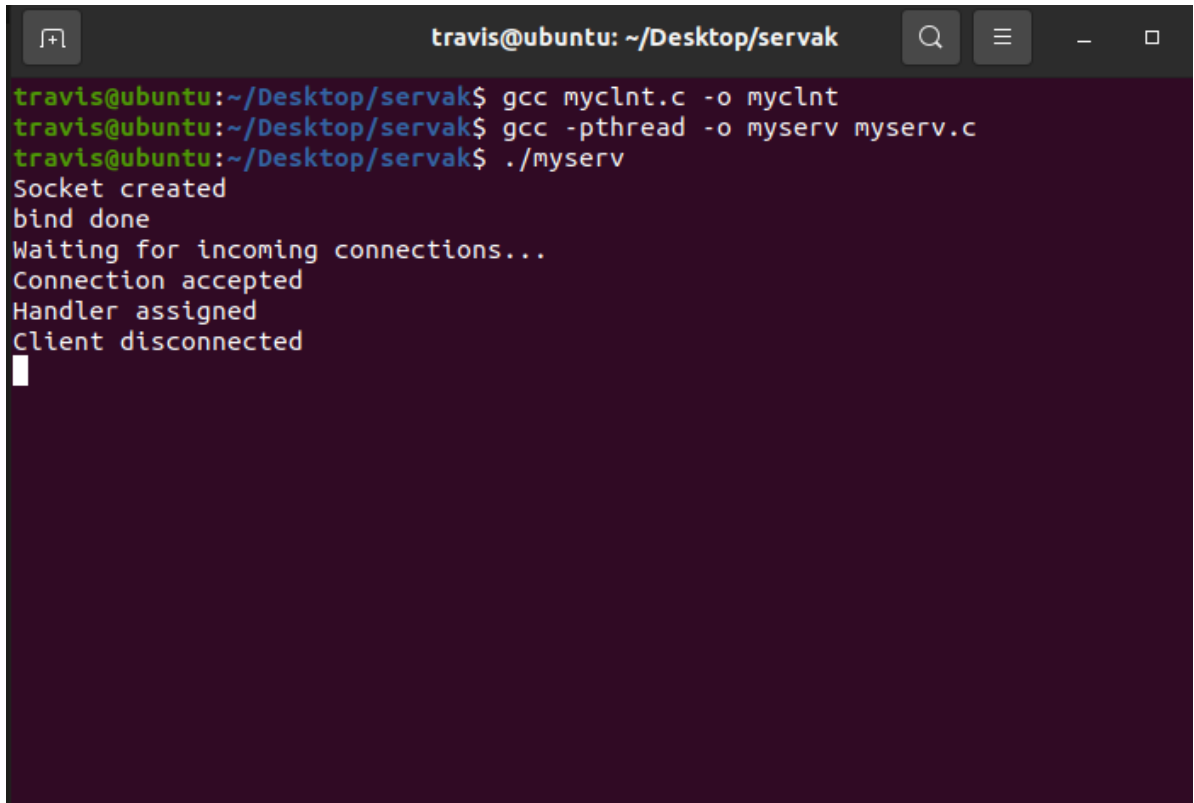
        //Receive a reply from the server
        if( recv(sock , server_reply , 2000 , 0) < 0)
        {
            puts("recv failed");
            break;
        }
    }
    gettimeofday(&stop, NULL);
    FILE *fout;
    fout = fopen("out.txt" , "ab");
    fprintf(fout, "%lu\n", (stop.tv_sec - start.tv_sec) * 1000000 + stop.tv_usec -
start.tv_usec);
    fclose(fout);
}

close(sock);
return 0;
}

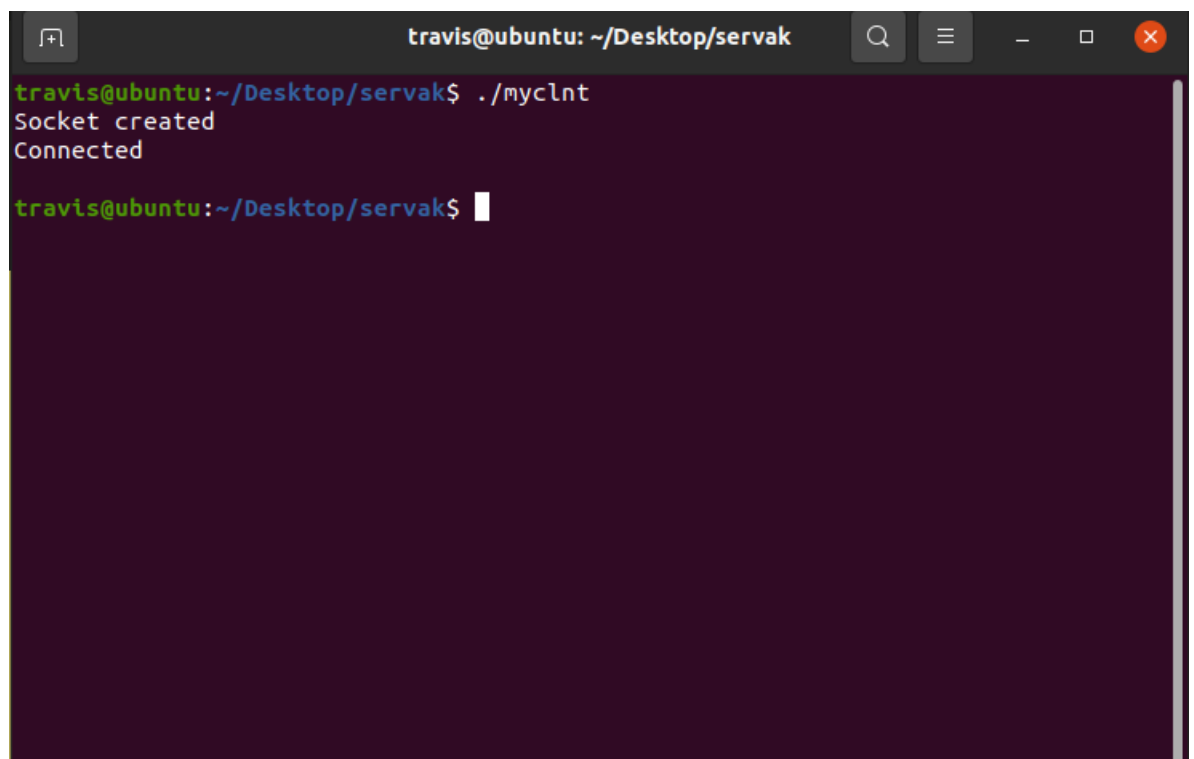
```

Для тестирования замерялось время в микросекундах, за которое происходят отправка клиентом 500 сообщений, а также получение ответа от сервера на каждое из них, и записывалось в файл out.txt. Данная процедура проделывалась 1000 раз для каждого набора настроек. Затем клиент отключался.

Компиляция файлов и запуск сервера-клиента.

A terminal window titled 'travis@ubuntu: ~/Desktop/servak' with standard window controls. It shows the compilation of 'myclnt.c' and 'myserv.c' using 'gcc', followed by running './myserv'. The output shows the server starting, listening for connections, and successfully handling a client connection before the client disconnects.

```
travis@ubuntu:~/Desktop/servak$ gcc myclnt.c -o myclnt
travis@ubuntu:~/Desktop/servak$ gcc -pthread -o myserv myserv.c
travis@ubuntu:~/Desktop/servak$ ./myserv
Socket created
bind done
Waiting for incoming connections...
Connection accepted
Handler assigned
Client disconnected
█
```

A terminal window titled 'travis@ubuntu: ~/Desktop/servak' with standard window controls. It shows the execution of './myclnt', which outputs 'Socket created' and 'Connected', indicating a successful connection to the server.

```
travis@ubuntu:~/Desktop/servak$ ./myclnt
Socket created
Connected

travis@ubuntu:~/Desktop/servak$ █
```

```
out.txt
~/Desktop/servak

1 112656
2 41937
3 11620
4 55948
5 49735
6 45779
7 35877
8 38176
9 41485
10 34926
11 64099
12 49411
13 46201
14 43555
15 50259
16 48627
17 50964
18 40483
19 41122
20 45974
21 58828
22 59214
23 41669
24 54465
25 42110
26 43540
27 39300
```

Для каждого набора настроек находились математическое ожидание и стандартное отклонение замеряемого времени.

Время, мкс	SOL_SOCKET, SO_KEEPALIV E	IPPROTO_TCP, TCP_NODELA Y	IPPROTO_TCP, TCP_QUICKA CK	IPPROTO_TCP, TCP_FASTOPE N	SOL_SOCKET, SO_REUSEAD DR
Математическо е ожидание	47607	44701	45032	53911	56237
Стандартное отклонение	12440	12410	9716	15370	17856

Таблица 1. Результаты исследования

Теперь приведем краткую информацию о каждой настройке:

- SO\_KEEPALIVE – включить отправку сообщений проверки активности на сокеты, ориентированные на соединение.
- TCP\_NODELAY – это отключит алгоритм Нагла. Это означает, что пакеты всегда отсылаются при первой же возможности, даже если к отправке назначено небольшое количество данных.
- TCP\_QUICKACK – включает режим quickack при установке или выключает при очищении. В этом режиме все уведомления отправляются немедленно, а не с некоторой задержкой в соответствии с обычными операциями TCP.

- TCP\_FASTOPEN – эта концепция была введена как решение для повышения производительности TCP-соединений, сокращающее количество циклов «рукопожатия».
- SO\_REUSEADDR – указывает, что правила, используемые при проверке адресов, предоставленных в вызове bind, должны разрешать повторное использование локальных адресов.

#### Выводы:

Как видно из таблицы 1, наилучшим образом себя показали настройки TCP\_NODELAY и TCP\_QUICKACK, что довольно ожидаемо, поскольку они обе предполагают включение режима с немедленной отправкой данных. Также заметим, что показатели для TCP\_QUICKACK имеют наименьший разброс. Остальные настройки, в соответствии с их описаниями, ориентированы на работу сервера с несколькими клиентами, в то время как в данной работе запускался лишь один. Таким образом, они только увеличили среднее время отправки и получения и его разброс.