



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное  
учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ \_\_\_\_\_ «Информатика и системы управления»

КАФЕДРА \_\_\_\_\_ «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа № 6**

Дисциплина Операционные системы

Тема Сокеты

Студент Ильясов И. М.

Группа ИУ7-63Б

Оценка (баллы) \_\_\_\_\_

Преподаватель Рязанова Н. Ю.

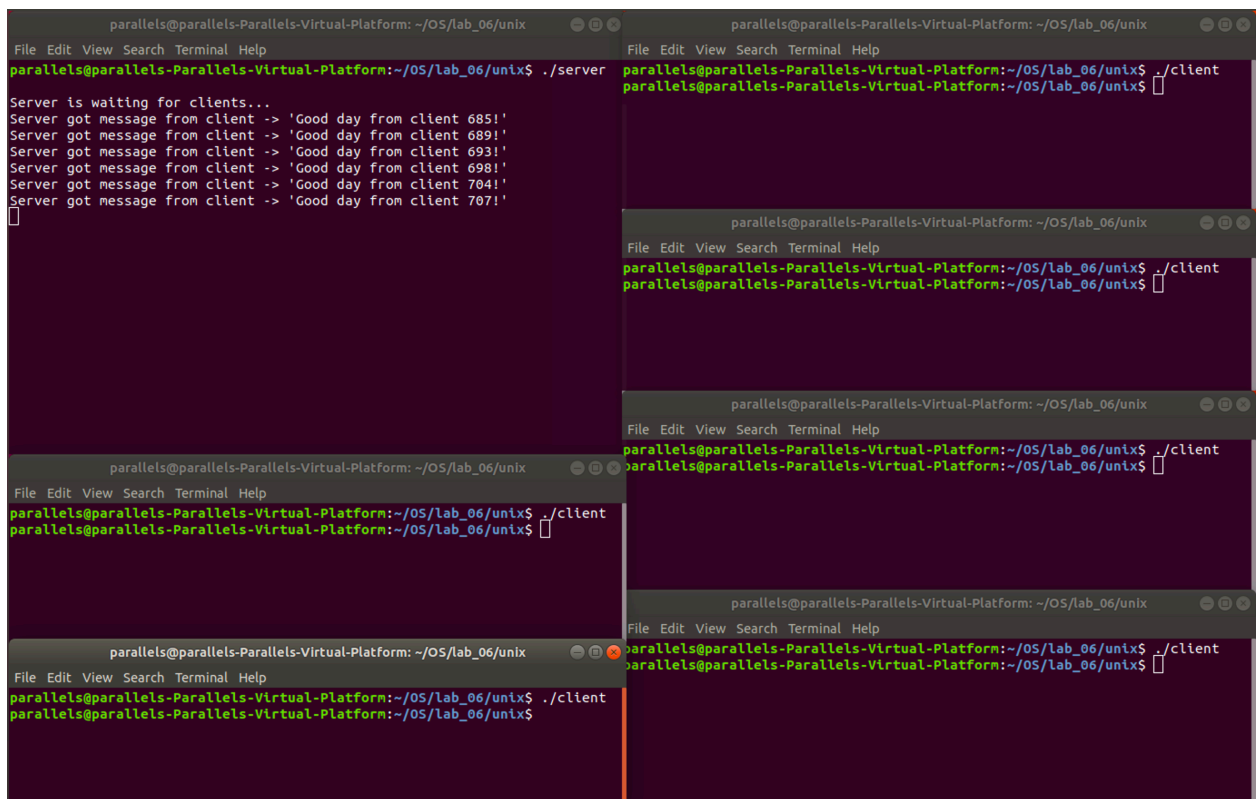
Москва, 2020 г.

## Задание 1.

Написать приложение по модели клиент-сервер, демонстрирующее взаимодействие параллельных процессов на отдельном компьютере с использованием сокетов в файловом пространстве имен: семейство – `AF_UNIX`, тип – `SOCK_DGRAM`. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

### Пример работы программы

Ниже приведен пример работы программы из задания №1. На данном рисунке 6 процессов-клиентов отправляют процессу-серверу сообщение типа “Good day from client X!”, где X – идентификатор процесса-клиента.



```
parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/unix
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$ ./server
Server is waiting for clients...
Server got message from client -> 'Good day from client 6851!'
Server got message from client -> 'Good day from client 6891!'
Server got message from client -> 'Good day from client 6931!'
Server got message from client -> 'Good day from client 6981!'
Server got message from client -> 'Good day from client 7041!'
Server got message from client -> 'Good day from client 7071!'
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/unix
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$ ./client
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/unix
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$ ./client
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/unix
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$ ./client
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/unix
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$ ./client
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/unix
File Edit View Search Terminal Help
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$ ./client
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_06/unix$
```

Рисунок 1. Пример работы программы из задания №1.

В процессе-сервере производится системный вызов `socket()`, при помощи которого создается сокет типа `SOCK_DGRAM` типа `AF_UNIX`. После этого производится системный вызов `bind()`, который привязывает сокет к локальному адресу. Затем процесс-сервер при помощи системного вызова

*recv()* происходит блокировка процесса-сервера (процесс-сервер ожидает ввод сообщений от процессов-клиентов).

В процессах-клиентах также производится системный вызов *socket()*, который создает сокет типа *SOCK\_DGRAM* семейства *AF\_UNIX*. Отправка сообщений процессами-клиентами процессу-серверу производится вызовом функции *sendto()*.

### Листинг программы

Ниже приведено 3 листинга – листинг с кодом программы клиента, листинг с кодом программы сервера и содержимое вспомогательного файла-хедера *info.h*.

#### Листинг 1. Код программы клиента.

```
#include "info.h"

int main()
{
    int sock = socket(AF_UNIX, SOCK_DGRAM, 0);

    if (sock < 0)
    {
        perror("Socket error\n");
        return(sock);
    }

    struct sockaddr_un server_addr;
    server_addr.sun_family = AF_UNIX;
    strcpy(server_addr.sun_path, NAME_SOCKET);

    char message[LEN_MESSAGE];
    sprintf(message, "Good day from client %d!", getpid());
    sendto(sock, message, strlen(message), 0,
           (struct sockaddr *)&server_addr, sizeof(server_addr));

    close(sock);
    return 0;
}
```

## Листинг 2. Код программы сервера.

```
#include "info.h"

int sock;

void catch_sigint(int signum)
{
    printf("\nSocket is closing (Ctrl+C signal was caught)!\n");
    close(sock);
    unlink(NAME_SOCKET);
    return 0;
}

int main(void)
{
    char message[LEN_MESSAGE];
    struct sockaddr_un addr;

    sock = socket(AF_UNIX, SOCK_DGRAM, 0);

    if (sock < 0)
    {
        perror("Socket error\n");
        return sock;
    }

    addr.sun_family = AF_UNIX;
    strcpy(addr.sun_path, NAME_SOCKET);

    if (bind(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("Bind error\n");
        close(sock);
        unlink(NAME_SOCKET);
        return -1;
    }

    printf("\nServer is waiting for clients...\n");
    signal(SIGINT, catch_sigint);

    for (;;)
    {
        int size = recv(sock, message, sizeof(message), 0);

        if (size < 0)
        {
            perror("Recv error\n");
            close(sock);
            unlink(NAME_SOCKET);
            return size;
        }

        message[size] = 0;
        printf("Server got message from client -> '%s'\n", message);
    }
    printf("Socket is closing\n");
    close(sock);
    unlink(NAME_SOCKET);
    return 0;
}
```

### Листинг 3. Содержимое вспомогательного файла *info.h*.

```
#ifndef _INFO_H_
#define _INFO_H_

#define LEN_MESSAGE 256
#define NAME_SOCKET "socket.soc"

#include <stdio.h>
#include <stdlib.h>
#include <sys/un.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <signal.h>
#include <unistd.h>

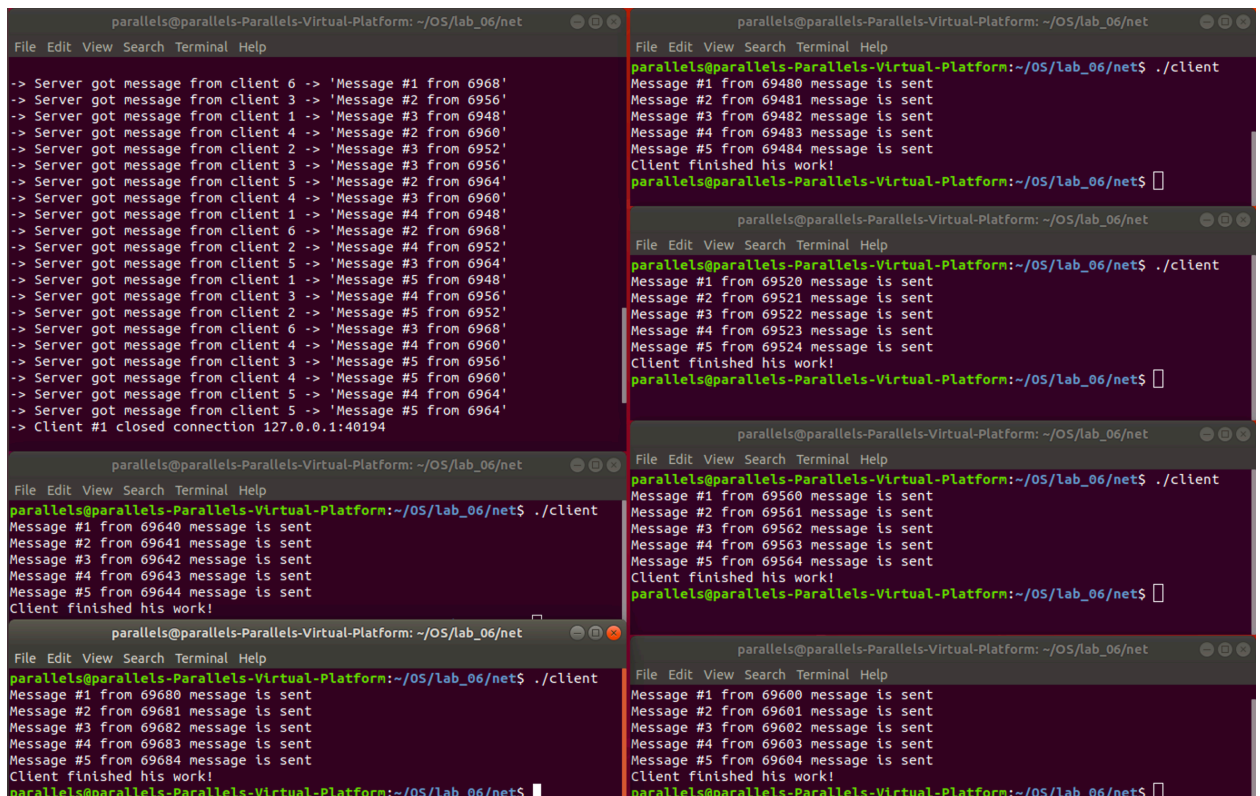
#endif
```

## Задание 2.

Написать приложение по модели клиент-сервер, осуществляющее взаимодействие параллельных процессов, которые выполняются на разных компьютерах. Для взаимодействия с клиентами сервер должен использовать мультиплексирование. Сервер должен обслуживать запросы параллельно запущенных клиентов. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

### Пример работы программы

Ниже приведен пример работы программы из задания №2. На данном рисунке 6 процессов-клиентов отправляют процессу-серверу сообщение типа "Message #X from Y!", где X – номер сообщения от процесса-клиента, Y – идентификатор процесса-клиента.



```
parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net
-> Server got message from client 6 -> 'Message #1 from 6968'
-> Server got message from client 3 -> 'Message #2 from 6956'
-> Server got message from client 1 -> 'Message #3 from 6948'
-> Server got message from client 4 -> 'Message #2 from 6960'
-> Server got message from client 2 -> 'Message #3 from 6952'
-> Server got message from client 3 -> 'Message #3 from 6956'
-> Server got message from client 5 -> 'Message #2 from 6964'
-> Server got message from client 4 -> 'Message #3 from 6960'
-> Server got message from client 1 -> 'Message #4 from 6948'
-> Server got message from client 6 -> 'Message #2 from 6968'
-> Server got message from client 2 -> 'Message #4 from 6952'
-> Server got message from client 5 -> 'Message #3 from 6964'
-> Server got message from client 1 -> 'Message #5 from 6948'
-> Server got message from client 3 -> 'Message #4 from 6956'
-> Server got message from client 2 -> 'Message #5 from 6952'
-> Server got message from client 6 -> 'Message #3 from 6968'
-> Server got message from client 4 -> 'Message #4 from 6960'
-> Server got message from client 3 -> 'Message #5 from 6956'
-> Server got message from client 4 -> 'Message #5 from 6960'
-> Server got message from client 5 -> 'Message #4 from 6964'
-> Server got message from client 5 -> 'Message #5 from 6964'
-> Client #1 closed connection 127.0.0.1:40194

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$ ./client
Message #1 from 69480 message is sent
Message #2 from 69481 message is sent
Message #3 from 69482 message is sent
Message #4 from 69483 message is sent
Message #5 from 69484 message is sent
Client finished his work!
parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$ ./client
Message #1 from 69520 message is sent
Message #2 from 69521 message is sent
Message #3 from 69522 message is sent
Message #4 from 69523 message is sent
Message #5 from 69524 message is sent
Client finished his work!
parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$ ./client
Message #1 from 69560 message is sent
Message #2 from 69561 message is sent
Message #3 from 69562 message is sent
Message #4 from 69563 message is sent
Message #5 from 69564 message is sent
Client finished his work!
parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$

parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$ ./client
Message #1 from 69600 message is sent
Message #2 from 69601 message is sent
Message #3 from 69602 message is sent
Message #4 from 69603 message is sent
Message #5 from 69604 message is sent
Client finished his work!
parallels@parallels-Parallels-Virtual-Platform: ~/OS/lab_06/net$
```

Рисунок 2. Пример работы программы из задания №2.

В процессе-сервере производится системный вызов *socket()*, при помощи которого создается сетевой сокет типа *SOCK\_STREAM* семейства *AF\_INET*. Далее системный вызов *bind()* связывает сокет с адресом, который

прописанным во вспомогательном файле *info.h*. Затем процесс-сервер при помощи системного вызова *listen()* переводится в режим ожидания и ожидает запрос на соединение от процессов-клиентов. При каждом входе в цикл производится очистка набора дескрипторов при помощи макроса *FD\_ZERO* и создание нового набора дескрипторов. В созданный набор заносится сокет сервера макросом *FD\_SET*. После этого процесс-сервер блокируется системным вызовом *select()*, который возвращает управление в функции при наличии запроса от процесса-клиента. Если блокировка снимается и управление возвращается процессу-серверу, он производит проверку на наличие новых соединений от процессов-клиентов и при их наличии вызывает функцию *connection\_handling()*. В этой функции принимается новое соединение при помощи вызова функции *accept()* и создается еще один сокет, который потом заносится в массив файловых дескрипторов. После этого осуществляется проход по массиву файловых дескрипторов и при условии если файловый дескриптор находится в наборе дескрипторов (проверка производится при помощи макроса *FD\_ISSET*), то вызывается функция *check\_info\_from\_clients*, в которой при помощи системного вызова *recv()* процессом-сервером считываются сообщения от процессов-клиентов. Далее при условии, что системный вызов *recv()* не вернул нулевое значение, выводятся сообщения, полученные процессом-сервером от процессов-клиентов. Если все же системный вызов *recv()* вернул нулевое значение, то получается, что соединение было сброшено, а значит этот сокет закрывается и удаляется из массива файловых дескрипторов.

В процессах-клиентах производится системный вызов *socket()*, при помощи которого создается сетевой сокет типа *SOCK\_STREAM* семейства *AF\_INET*. Функция *gethostbyname()* производит преобразование доменного адреса процесса-сервера в сетевой адрес. Далее вызывается функция *connect()*, которая устанавливает соединение с сокетом процесса-сервера, и в цикле при помощи системного вызова *send()* производится отправка сообщений процессами-клиентами сообщениями процессу-серверу.

## Листинг программы

Ниже приведено 3 листинга – листинг с кодом программы клиента, листинг с кодом программы сервера и содержимое вспомогательного файла-хедера *info.h*.

### Листинг 1. Код программы клиента.

```
#include "info.h"

int main()
{
    int sock = socket(AF_INET, SOCK_STREAM, 0);

    if (sock < 0)
    {
        perror("Socket error\n");
        exit(-1);
    }
    struct hostent* host = gethostbyname(SOCKET_ADDRESS);

    if (!host)
    {
        perror("Getting host by name error\n");
        close(sock);
        exit(-1);
    }
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SOCKET_PORT);
    server_addr.sin_addr = *((struct in_addr*) host->h_addr_list[0]);

    if (connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
    {
        perror("Connect error\n");
        close(sock);
        return -1;
    }
    char message[LEN_MESSAGE];

    for (int i = 0; i < QNT_MESSAGE; i++)
    {
        memset(message, 0, LEN_MESSAGE);
        sprintf(message, "Message #%d from %d", i + 1, getpid());
        printf("%s", message);

        if (send(sock, message, strlen(message), 0) < 0)
        {
            perror("Send error\n");
            close(sock);
            return -1;
        }
        printf("\n%d message is sent\n", i);
        sleep(rand() % 5 + 1);
    }
    printf("Client finished his work!\n");
    close(sock);
    return 0;
}
```



## Листинг 2. Код программы сервера.

```
#include "info.h"

void connection_closing();
int clients[MAX_CLIENTS] = {0};
int soc;

int connection_handling()
{
    struct sockaddr_in client_addr;
    int client_size = sizeof(client_addr);
    int client_socket = accept(soc, (struct sockaddr*)&client_addr,
(socklen_t*)&client_size);

    if (client_socket < 0)
    {
        perror("Accept error\n");
        connection_closing();
        close(soc);
        return -1;
    }

    printf("\n\n-> Server got new connection:\nip =
%s:%d\n",inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

    for (int i = 0; i < MAX_CLIENTS; i++)
    {
        if (clients[i] == 0)
        {
            clients[i] = client_socket;
            printf("It is client #%d\n", i+1);
            return 0;
        }
    }

    return -1;
}

void check_info_from_clients(int client_sock , int client_index)
{
    char message[LEN_MESSAGE];
    struct sockaddr_in client_addr ;
    int addr_size = sizeof(client_addr);
    int message_size = recv(client_sock, message, LEN_MESSAGE, 0);

    if (message_size == 0)
    {
        getpeername(client_sock, (struct sockaddr*)&client_addr,
(socklen_t*)&addr_size);
        printf("\n-> Client #%d closed connection %s:%d\n", client_index + 1,
            inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
        close(client_sock);
        clients[client_index] = 0;
    }
    else
    {
        message[message_size] = '\0';
        printf ("\n-> Server got message from client %d -> '%s'", client_index + 1,
message);
    }
}
```

```

void connection_closing()
{
    for (int i = 0; i < MAX_CLIENTS; i++)
    {
        if (clients[i] != 0)
        {
            close(clients[i]);
        }
    }
}

int main(void)
{
    soc = socket(AF_INET, SOCK_STREAM, 0);
    if (soc < 0)
    {
        perror("Socket error\n");
        return -1;
    }

    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(SOCKET_PORT);
    addr.sin_addr.s_addr = INADDR_ANY;

    if (bind(soc, (struct sockaddr*)&addr, sizeof(addr)) < 0)
    {
        perror ("Bind error\n");
        close(soc);
        return -1;
    }

    if (listen(soc, 3) < 0)
    {
        perror("Listen error\n");
        close(soc);
        return -1;
    }

    printf("Server launched on ip %s:%d\n", inet_ntoa(addr.sin_addr),
    ntohs(addr.sin_port));

    for (;;)
    {
        fd_set fds;
        int max_fd;

        FD_ZERO(&fds);
        FD_SET(soc, &fds);
        max_fd = soc;

        for ( int i = 0; i < MAX_CLIENTS; i++)
        {
            if (clients[i] > 0)
            {
                FD_SET(clients[i], &fds);
            }

            if (clients[i] > max_fd)
            {
                max_fd = clients[i];
            }
        }
    }
}

```

```

    }
}

if (select(max_fd + 1, &fds , NULL, NULL, NULL) < 0)
{
    perror("Select error\n");
    connection_closing();
    close(soc);
    return -1;
}

if (FD_ISSET(soc, &fds))
{
    if (connection_handling() < 0)
    {
        perror("No space for new connections\n");
        connection_closing();
        close(soc);
        return -1;
    }
}

for (int i = 0; i < MAX_CLIENTS; i++)
{
    if ((clients[i] > 0) && FD_ISSET(clients[i], &fds))
    {
        check_info_from_clients(clients[i], i);
    }
}

connection_closing();
close(soc);
return 0;
}

```

### Листинг 3. Содержимое вспомогательного файла *info.h*.

```

#ifndef _INFO_H_
#define _INFO_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>

#define LEN_MESSAGE 128
#define MAX_CLIENTS 10
#define SOCKET_ADDRESS "localhost"
#define SOCKET_PORT 6951
#define QNT_MESSAGE 5

#endif

```