|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

**по курсу “Операционные системы”**

**по теме “Сокеты”**

|  |  |
| --- | --- |
| Студент: | Уласик Е.А. |
| Группа: | ИУ7-61 |
| Преподаватель: | Рязанова Н.Ю. |

1. *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.

Листинг программы:

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <sys/types.h>
6. #include <signal.h>
7. #include <sys/socket.h>
8. #include "info.h"
10. **int** sock;
12. **void** sigint\_catcher(**int** signum)
13. {
14. printf("keyboard interrupt was catched\n");
15. close(sock);
16. unlink(SOCKET\_NAME);
18. }
20. **int** main(**void**)
21. {
22. **char** msg[MSG\_LEN];
23. **struct** sockaddr sock\_adr;
25. sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);
26. **if** (sock < 0)
27. {
28. perror("Error in socket(): ");
29. **return** sock;
30. }
32. sock\_adr.sa\_family = AF\_UNIX;
33. strcpy(sock\_adr.sa\_data, SOCKET\_NAME);
35. **if** (bind(sock, &sock\_adr, **sizeof**(sock\_adr)) < 0)
36. {
37. printf("Closing socket....\n");
38. close(sock);
39. unlink(SOCKET\_NAME);
40. perror("Error in bind(): ");
41. **return** -1;
42. }

Листинг 1. Код программы server.c (Часть 1)

Листинг 2. Код программы server.c (Часть 2)

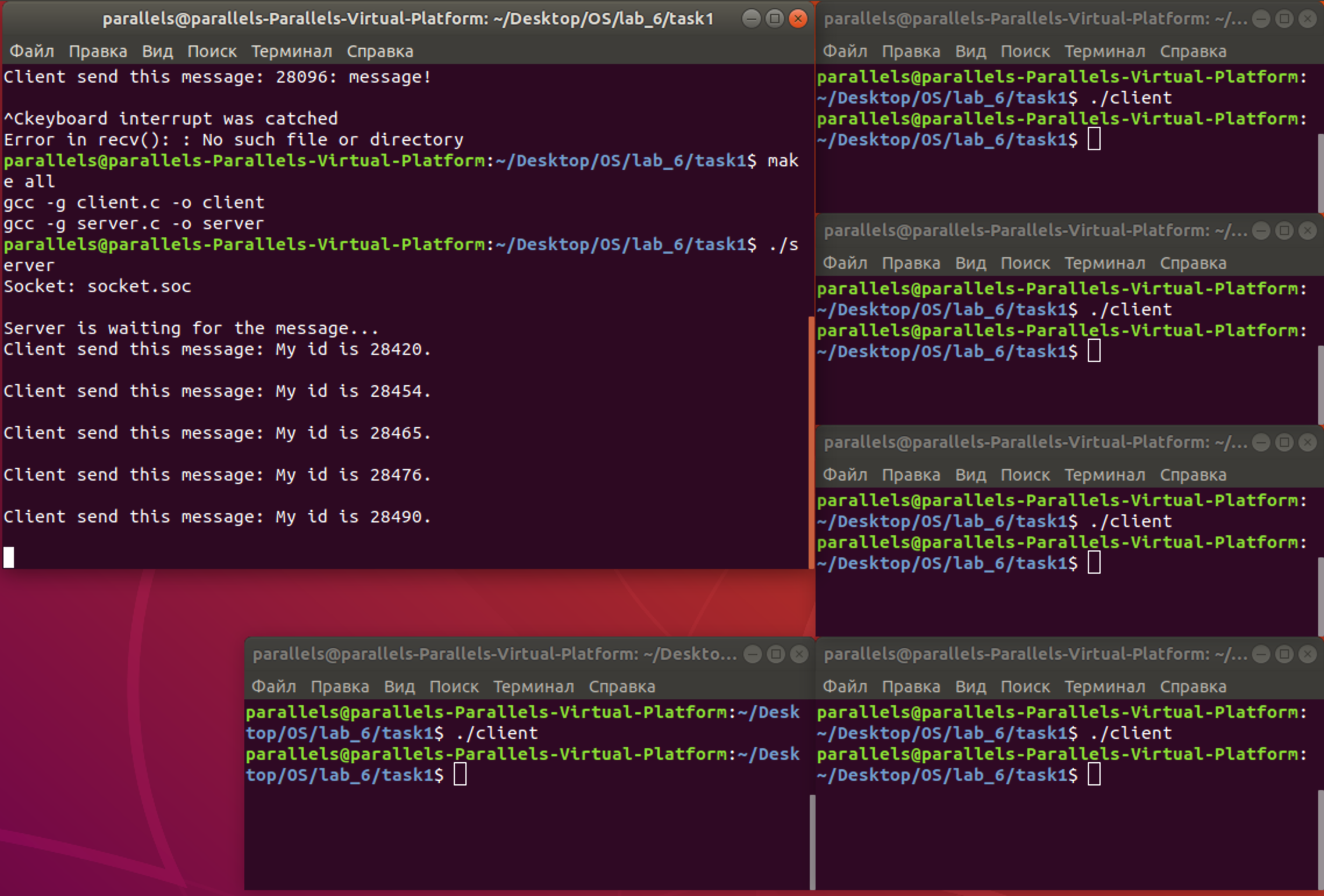
1. printf("\nServer is waiting for the message...\n");
2. signal(SIGINT, sigint\_catcher);
4. **while**(1)
5. {
6. **int** recieved\_size = recv(sock, msg, **sizeof**(msg), 0);
7. **if** (recieved\_size < 0)
8. {
9. close(sock);
10. unlink(SOCKET\_NAME);
11. perror("Error in recv(): ");
12. **return** recieved\_size;
13. }
15. msg[recieved\_size] = 0;
16. printf("Client send this message: %s\n", msg);
17. }
18. close(sock);
19. unlink(SOCKET\_NAME);
20. **return** 0;
21. }

Листинг 3. Код программы client.c

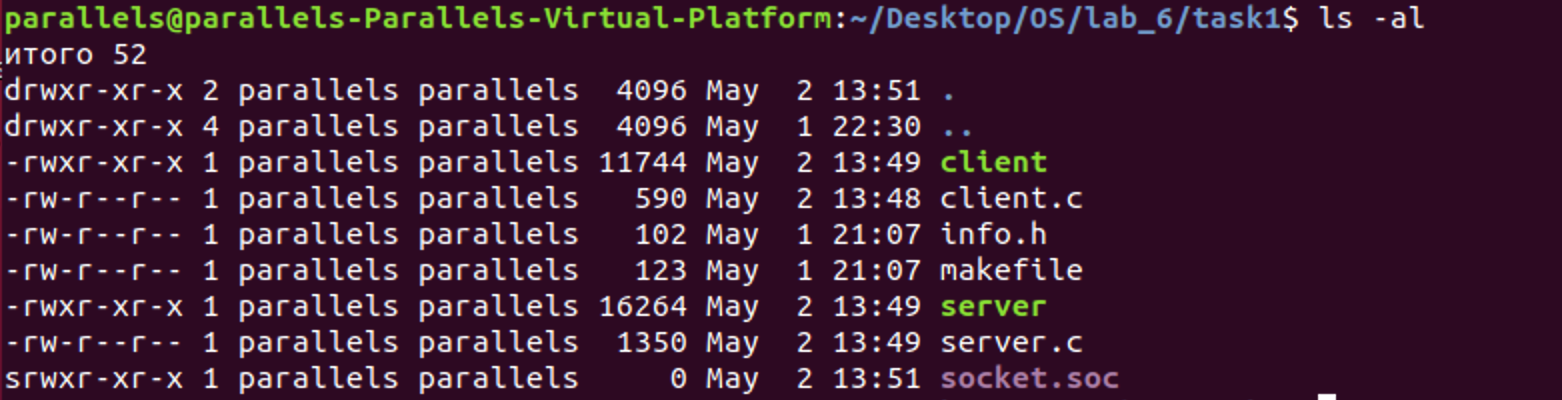
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <sys/types.h>
6. #include <sys/socket.h>
8. #include "info.h"
10. **int** main(**void**)
11. {
12. **int** sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);
13. **if** (sock < 0)
14. {
15. printf("Error in socket()\n");
16. **return** sock;
17. }
19. **struct** sockaddr server\_adr;
20. server\_adr.sa\_family = AF\_UNIX;
21. strcpy(server\_adr.sa\_data, SOCKET\_NAME);
23. **char** msg[MSG\_LEN];
24. strcpy(msg, "message!\n");
25. sendto(sock, msg, strlen(msg), 0, &server\_adr, **sizeof**(server\_adr));
27. close(sock);
28. **return** 0;
29. }
30. #ifndef INFO\_H
31. #define INFO\_H
33. #define MSG\_LEN 256
34. #define SOCKET\_NAME "socket.soc"
36. #endif // INFO\_H

Листинг 4. Код заголовочного файла info.h

Результат:



Картинка 1. Результат работы



Картинка 2. Сокет в файловой системе

В процессе сервере создаётся сокет с помощью системного вызова socket. Сокет семейства AF\_UNIX с типом SOCK\_DGRAM. С помощью системного вызова bind происходит связка сокета с локальным адресом. Сервер блокируется на функции recv в ожидание сообщения от процессов-клиентов.

В процессе клиенте создаётся сокет с помощью системного вызова socket. Сокет семейства AF\_UNIX с типом SOCK\_DGRAM. Затем с помощью функции sendto отправляется сообщение к процессу серверу.

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.

Листинг программы:

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <errno.h>
5. #include <unistd.h>
6. #include <sys/types.h>
7. #include <sys/socket.h>
8. #include <sys/select.h>
9. #include <arpa/inet.h>
10. #include <netdb.h>
11. #include "info.h"
13. #define MAX\_CLIENTS 10
14. **int** clients[MAX\_CLIENTS] = { 0 };
15. **int** sock;
17. **int** connection\_handling(**int** current\_fd)
18. {
19. **struct** sockaddr\_in addr;
20. **int** addr\_len = **sizeof**(addr);
21. **int** fd;
22. **if** ((fd = accept(sock, (**struct** sockaddr \*) &addr, (socklen\_t \*) &addr\_len)) < 0)
23. {
24. close(sock);
25. perror("Error in access\n");
26. exit(-1);
27. };
29. printf("server got new connection by:\n fd = %d \n ip %s : %d", fd, inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));
30. **for** (**int** i = 0; i < MAX\_CLIENTS; i++)
31. {
32. **if** (clients[i] == 0)
33. {
34. clients[i] = fd;
35. **return** 0;
36. }
37. }
38. **return** -1;
39. }
41. **void** client\_handling(**int** fd, **int** i)
42. {
43. **struct** sockaddr\_in addr;
44. **int** addrlen = **sizeof**(addr);

Листинг 5. Листинг программы server.c (часть 1)

Листинг 6. Листинг программы server.c (Часть 2)

1. **char** msg[MSG\_LEN];
3. **int** ms\_len = recv(fd, &msg, **sizeof**(msg), 0);
4. **if** (ms\_len == 0)
5. {
6. getpeername(fd, (**struct** sockaddr\*) &addr, (socklen\_t\*) &addrlen);
7. printf("User %d disconnected %s:%d \n", i, inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));
8. close(fd);
9. clients[i] = 0;
10. }
11. **else**
12. {
13. msg[ms\_len] = 0;
14. printf("Server got message from client: %d = %s", i + 1, msg);
15. }
16. }
18. **int** main (**void**)
19. {
20. sock = socket(AF\_INET, SOCK\_STREAM, 0);
21. **if** (sock < 0)
22. {
23. perror("socket creating error\n");
24. **return** sock;
25. }
26. **struct** sockaddr\_in serv\_sock;
27. serv\_sock.sin\_family = AF\_INET;
28. serv\_sock.sin\_addr.s\_addr = INADDR\_ANY;
29. serv\_sock.sin\_port = htons(SOCK\_PORT);
30. **if** (bind(sock, (**struct** sockaddr \*) &serv\_sock, **sizeof**(serv\_sock)) < 0)
31. {
32. perror("binding error \n");
33. **return** -1;
34. }
36. **if** (listen(sock, 3) < 0)
37. {
38. perror("server can't listen");
39. **return** -1;
40. }
41. printf("server active on ip %s, on port %d\n", inet\_ntoa(serv\_sock.sin\_addr), htons(serv\_sock.sin\_port));
42. **while** (1)
43. {
44. **int** max\_fd = sock;
45. fd\_set set;
46. FD\_SET(sock, &set);
48. **for** (**int** i = 0; i < MAX\_CLIENTS; i++)
49. {
50. **if** (clients[i] > 0)
51. {
52. FD\_SET(clients[i], &set);
53. }
55. **if** (clients[i] > max\_fd)
56. max\_fd = clients[i];
57. }
58. **int** active\_clients\_count = select(max\_fd + 1, &set, NULL, NULL, NULL);
59. **if** (active\_clients\_count < 0)
60. {
61. perror("there is no active clients\n");
62. **return** - 1;
63. }

Листинг 7. Листинг программы server.c (Часть 3)

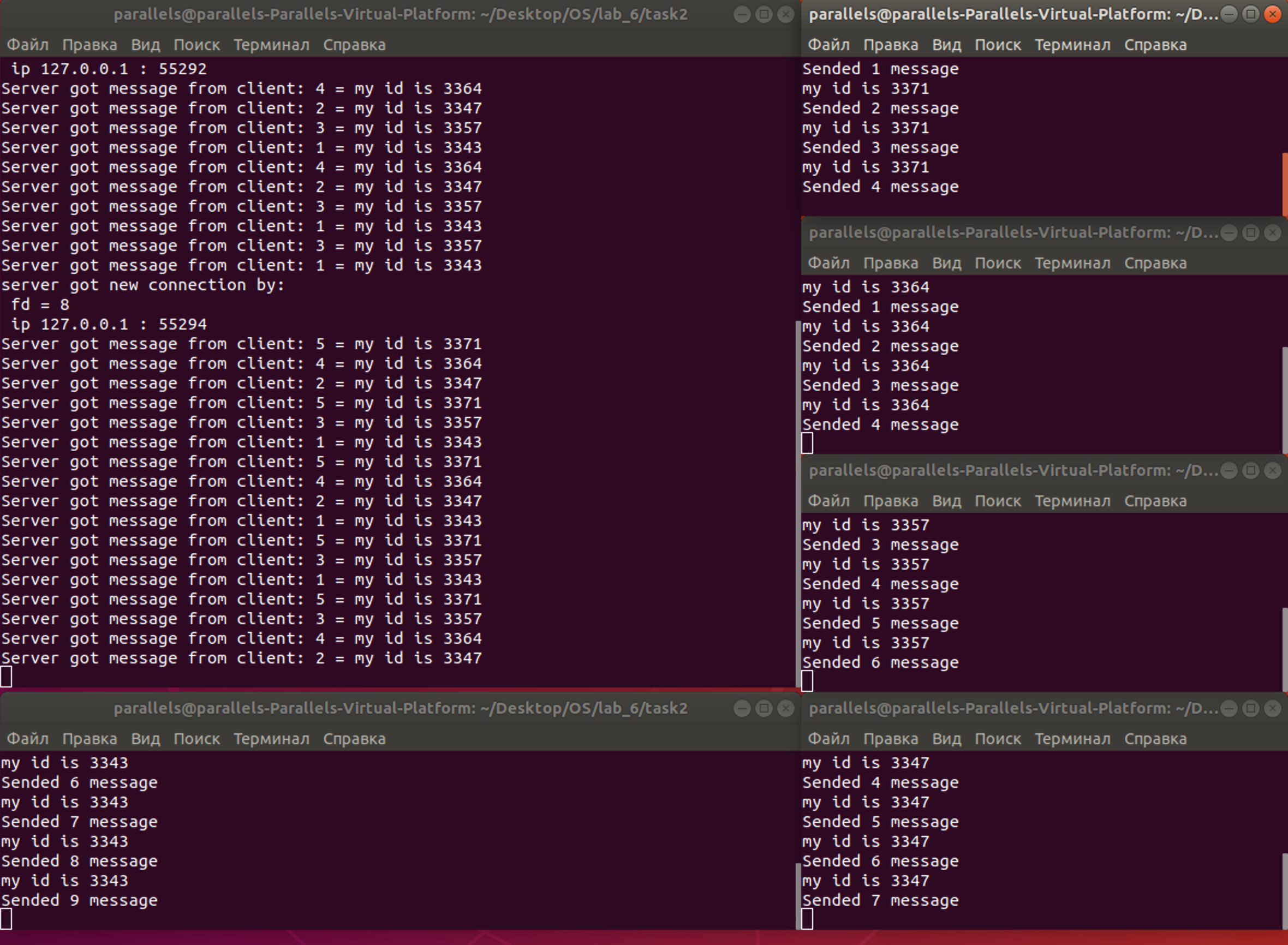
1. #ifndef INFO\_H
2. #define INFO\_H
4. #define MSG\_LEN 256
5. #define SOCK\_ADDR "localhost"
6. #define SOCK\_PORT 8088
8. #endif // INFO\_H
9. **if**(FD\_ISSET(sock, &set))
10. **if** (connection\_handling(sock) < 0)
11. {
12. perror("connection error[]\n");
13. **return** -1;
14. }
15. **for** (**int** i = 0; i < MAX\_CLIENTS; i++)
16. {
17. **int** client = clients[i];
18. **if** (client > 0 && FD\_ISSET(client, &set))
19. client\_handling(client, i);
20. }
21. }
23. close(sock);
24. **return** 0;
25. }

Листинг 8. Листинг заголовочного файла info.h

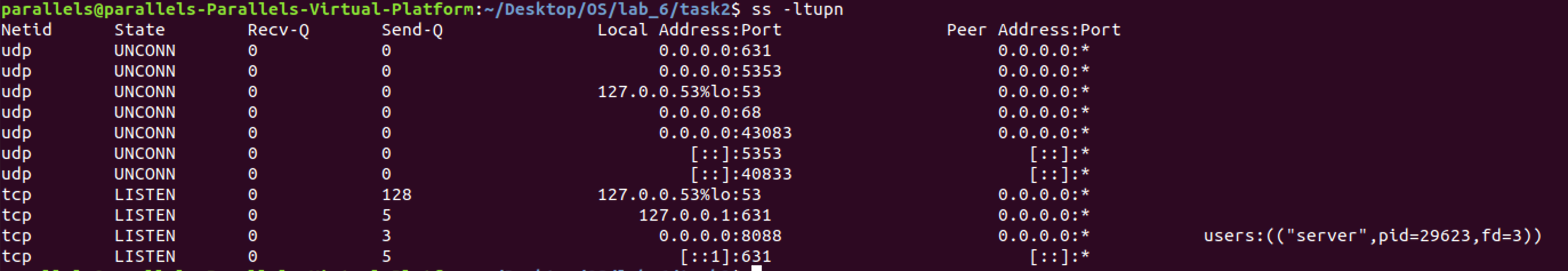
Листинг 9. Листинг программы client.c

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <time.h>
5. #include <unistd.h>
6. #include <signal.h>
7. #include <sys/types.h>
8. #include <sys/socket.h>
9. #include <arpa/inet.h>
10. #include <netdb.h>
12. #include "info.h"
14. **int** main(**void**)
15. {
16. srand(time(NULL));
18. **int** sock = socket(PF\_INET, SOCK\_STREAM, 0);
19. **if** (sock < 0)
20. {
21. perror("Error in sock(): ");
22. **return** sock;
23. }
25. **struct** hostent\* host = gethostbyname(SOCK\_ADDR);
26. **if** (!host)
27. {
28. perror("Error in gethostbyname(): ");
29. **return** -1;
30. }
32. **struct** sockaddr\_in server\_addr;
33. server\_addr.sin\_family = PF\_INET;
34. server\_addr.sin\_port = htons(SOCK\_PORT);
35. server\_addr.sin\_addr = \*((**struct** in\_addr\*) host->h\_addr\_list[0]);
37. **if** (connect(sock, (**struct** sockaddr\*) &server\_addr, **sizeof**(server\_addr)) < 0)
38. {
39. perror("Error in connect():");
40. **return** -1;
41. }
43. **char** msg[MSG\_LEN];
44. **for** (**int** i = 0; i < 10; i++)
45. {
46. memset(msg, 0, MSG\_LEN);
47. sprintf(msg, "%d message is here!\n", i);
48. printf("%s", msg);
50. **if** (send(sock, msg, strlen(msg), 0) < 0)
51. {
52. perror("Error in send(): ");
53. **return** -1;
54. }
56. printf("Sended %d message\n", i);
58. **int** wait\_time = 1 + rand() % 3;
59. sleep(wait\_time);
60. }
62. printf("Client app is over!\n");
63. **return** 0;
64. }

Результат:



Картинка 3. Результат работы.



Картинка 4. Порт и айди на предпоследней строчке (8088)

В процессе сервере создаётся сетевой сокет вызовом socket семейства AF\_INET с типом SOCK\_STREAM. Затем с помощью вызова bind сокет связывается с адресом, прописанным в SOCKET\_ADDRESS. После этого сервер переводится в режим ожидания на системном вызове listen, ожидая запроса на соединение. На каждом шаге цикла создаётся новый набор дескрипторов, в него заносятся сокет сервера с помощью макроса FD\_SET. Далее сервер блокируется на вызове функции select, которая возвращает управление в функцию при получении хотя бы одного запроса от клиента. При выходе из блокировки проверяется наличие новых соединений и при их наличии вызывается функция connection\_handling, внутри которой с помощью вызова accept принимается новое соединение и создаётся сокет, который записывается в массив дескрипторов. После осуществляется обход по массиву дескрипторов и если дескриптор находится в наборе дескрипторов, то с помощью функции client\_handling осуществляется считывание с помощью функции recv и вывод сообщения от клиента на сервере. Если recv вернула нулевое значение, значит соединение было сброшено. После этого выводится сообщение о закрытии соединения и закрытии сокета.

В процессе клиенте создаётся сетевой сокет семейства AF\_INET с типом SOCK\_STREAM. Функция gethostbyname преобразует доменный адрес в сетевой, благодаря которому можно установиться соединение с помощью функции connect. Затем в цикле происходит отправка сообщений серверу.