**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» Тема: Сокеты

### Студент: Мирзоян С.А. Группа: ИУ7-65Б. Преподаватель: Рязанова Н.Ю.

Москва, 2020г

Сокет – это абстракция конечной точки взаимодействия. Абстракция сокетов была введена в 4.2 BSD (Berkley Software Distribution) UNIX и были созданы для организации взаимодействия процессов, причем безразлично, где эти процессы выполняются: на одной машине или на нескольких машинах.

Сокеты поддерживают множество протоколов, поэтому была определена общая структура адреса sockaddr, так как при создании коммуникационных отношений нужно указывать адрес конечной точки коммуникационного партнера.

Структура sockaddr определяется следующим образом:

typedef unsigned short sa\_family\_t

|  |
| --- |
|  |
|  |  |
|  | /\* |
|  | \* 1003.1g requires sa\_family\_t and that sa\_data is char. |
|  | \*/ |
|  |  |
|  | struct sockaddr { |
|  | sa\_family\_t sa\_family; /\* address family, AF\_xxx \*/ |
|  | char sa\_data[14]; /\* 14 bytes of protocol address \*/ |
|  | }; |

}

Когда сокет создается с помощью socket (2), он существует в пространстве имен (семейство адресов), но ему не назначен адрес. Специальная функция bind () назначает адрес, указанный addr, сокету, указанному дескриптором файла sockfd. Параметр addrlen определяет размер в байтах структуры адреса, на которую указывает addr. Традиционно эта операция называется «присвоение имени сокету».

Параметр domain указывает домен связи. Домен определяет семейство протоколов, которое будет использоваться для связи. Эти семейства определены в <sys / socket.h>. Ядро Linux в настоящее время включает следующие форматы:

Структура sockaddr\_in определяется следующим образом:

struct sockaddr\_in

{

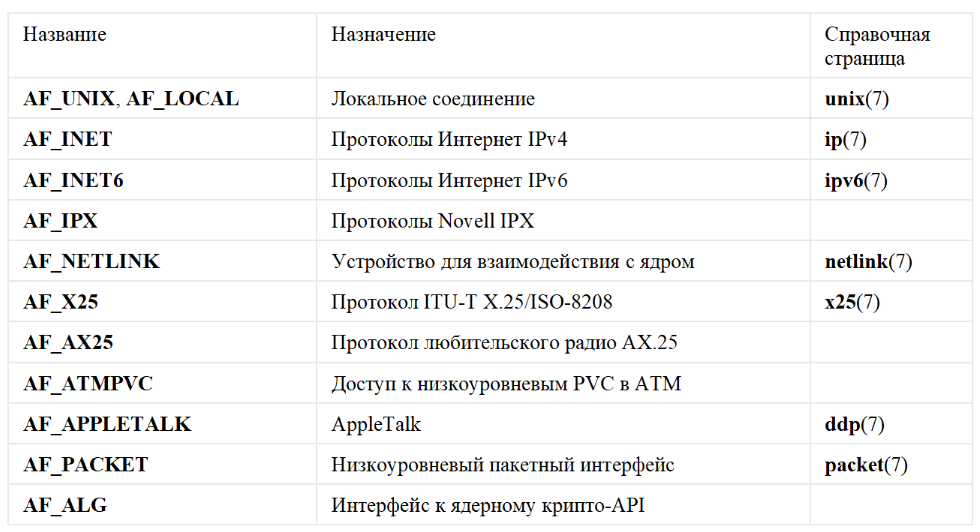
short int sin\_family; // Семейство адресов

unsigned short int sin\_port; // Номер порта

struct in\_addr sin\_addr; // IP-адрес

unsigned char sin\_zero[8]; // Дополнение до размера структуры sockaddr

};



Второй параметр функции socket() - тип сокета ( type). В настоящее время определены следующие типы:

* SOCK\_STREAM Обеспечивает создание двусторонних, надёжных потоков байтов на основе установления соединения. Может также поддерживаться механизм внепоточных данных.
* SOCK\_DGRAM Поддерживает дейтаграммы (ненадежные сообщения с ограниченной длиной без установки соединения).
* SOCK\_SEQPACKET Обеспечивает работу последовательного двустороннего канала для передачи дейтаграмм на основе соединений; дейтаграммы имеют постоянный размер; от получателя требуется за один раз прочитать целый пакет.
* SOCK\_RAW Обеспечивает прямой доступ к сетевому протоколу.
* SOCK\_RDM Обеспечивает надежную доставку дейтаграмм без гарантии, что они будут расположены по порядку.
* SOCK\_PACKET Устарел и не должен использоваться в новых программах; см. packet.

# Часть 1

## Организовать взаимодействие параллельных процессов на отдельном компьютере.

### Задание: 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 1: Код сервера

1. #include <sys/types.h>
2. #include <sys/socket.h>
3. #include <sys/un.h>
4. #include <stdio.h>
5. #include <unistd.h>
6. #include <stdlib.h>
7. #include <signal.h>
9. #define SOCK\_NAME "my\_socket"
10. #define SIZE 256
12. **void** reciever(**int** sock);
13. **void** sig\_handler(**int** sig);
15. **int** mysocket;
17. **int** main(**void**)
18. {
19. //int mysocket;
20. **struct** sockaddr addr;
22. mysocket = socket(AF\_LOCAL, SOCK\_DGRAM, 0); //socket - системный вызов, возвращает дескриптор сокета
23. //AF\_LOCAL - для связи процессов в одной машине
24. //SOCK\_DGRAM - датаграммные сокеты (без установления соединения, ненадежная передача сообщений; сообщения фиксированной максимальной длины).
25. **if** (mysocket < 0)
26. {
27. perror("Сообщение сервера: Невозможно открыть сокет!\n");
28. exit(1);
29. }
31. addr.sa\_family = AF\_LOCAL; //sa\_family - Семейство адресов, AF\_xxx
32. strcpy(addr.sa\_data, SOCK\_NAME) ;//sa\_data - 14 байтов для хранения адреса
34. **if** (bind(mysocket, &addr, **sizeof**(**struct** sockaddr)) < 0) //bind(2) привязывает сокет к локальному адресу,
35. {
36. printf("Закрыть сокет\n");
37. perror("Сообщение сервера: невозможно привязать имя к сокету!\n");
38. close(mysocket) ;
39. unlink(SOCK\_NAME) ;//unlink - системная функция, удаляет созданный файл
40. exit(2);
41. }
43. printf ("Сокет: %s\n", SOCK\_NAME) ;
44. signal(SIGINT, sig\_handler); // cntl+c
46. **while** (1)
47. reciever(mysocket); //прием сообщений
49. printf("Закрыть сокет\n");
50. close(mysocket) ;
51. unlink(SOCK\_NAME) ;
53. **return** 0;
54. }
56. **void** reciever(**int** sock)
57. {
58. **char** buffer[SIZE];
59. **int** size = recv(sock, buffer, **sizeof**(buffer), 0); //recv - прием данных из сокета
60. **if** (size < 0)
61. {
62. perror("Сообщение сервера: невозможно получить данные!\n");
63. close(sock) ;
64. unlink(SOCK\_NAME) ;
65. exit(3);
66. }
67. buffer[size] = 0;
68. printf("%s\n", buffer);
69. }
71. **void** sig\_handler(**int** sig) //удаление файла
72. {
73. puts("Завершение приема по сигналу (ctrl + c)\n");
74. close(mysocket) ;
75. unlink(SOCK\_NAME) ;
76. exit(1);
77. }

### Листинг 2: Код клиента

1. #include <sys/types.h>
2. #include <sys/socket.h>
3. #include <sys/un.h>
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <unistd.h>
8. #define SOCK\_NAME "my\_socket"
9. #define SIZE 256
11. **int** main(**int** argc, **char** \*argv[])
12. {
13. **int** sock;
15. **struct** sockaddr addr;
17. sock = socket(PF\_LOCAL, SOCK\_DGRAM, 0);
18. **if** (sock < 0)
19. {
20. perror("Сообщение клиента: невозможно открыть сокет!\n");
21. exit(1);
22. }
24. addr.sa\_family = AF\_LOCAL;
25. strcpy(addr.sa\_data, SOCK\_NAME);
27. **int** pid = getpid();
28. **char** mypid[6];   // ex. 34567
29. sprintf(mypid, "%d", pid);
31. strcat(argv[1], " ");
32. strcat(argv[1], mypid);
34. sendto(sock, argv[1], **sizeof**(argv[1]), 0, &addr, **sizeof**(addr));  //sendto -  отправляет данные в сокет,
36. close(sock) ;
37. **return** 0;
38. }

### Результат работы:

### - созданный сокет в ФС (ls –l).

### 

### 

### После завершения my\_socket удаляется.

***Часть 2***

**Организовать взаимодействие параллельных процессов в сети (ситуацию моделируем на одной машине).**

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.

TCP-порт **31337** использует протокол управления передачей данных (TCP), который является одним из основных протоколов в сетях TCP/IP. TCP является протоколом с установлением соединения и требует квитирования для установки сквозной связи. Только после установления соединения пользовательские данные могут пересылаться в обоих направлениях.  
TCP гарантирует доставку пакетов данных через порт **31337** в том же порядке, в котором они были отправлены. Гарантированная связь через TCP-порт **31337** является основным отличием TCP от UDP.

Листинг 3: Код сервера

1. #include <arpa/inet.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <sys/time.h>
5. #include <netinet/in.h>
6. #include <unistd.h>
7. #include <string.h>
8. #include <stdlib.h>
9. #include <stdio.h>
10. #include <fcntl.h>
12. #define SOCKET\_PORT 31337
13. #define BUFFER\_SIZE 256
14. #define LISTEN\_QUEUE\_SIZE 256
16. #define CLIENTS 10

19. **void** recieve\_data(**int** clients[], **int** i);
20. **int** max(**int** listener, **int** arr[], **int** count);
21. **int** insert(**int** s, **int** clients[], **int** count);
23. **int** main(**void**)
24. {
25. **int** sockL;
26. **struct** sockaddr\_in addr;
27. **int** clients[CLIENTS] = {0};
28. fd\_set rset;
30. sockL = socket(PF\_INET, SOCK\_STREAM, 0);
31. **if** (sockL < 0)
32. {
33. perror("Сообщение сервера: невозможно открыть сокет!\n");
34. **return** sockL;
35. }
36. addr.sin\_family = PF\_INET;
37. addr.sin\_port = htons(SOCKET\_PORT) ;
38. addr.sin\_addr.s\_addr = INADDR\_ANY;
39. **if** (bind(sockL, (**struct** sockaddr \*)&addr, **sizeof**(addr)) < 0)
40. {
41. perror("Сообщение сервера: невозможно привязать адрес!\n");
42. exit(2);
43. }
45. printf("Порт %d\n", SOCKET\_PORT) ;
47. listen(sockL, LISTEN\_QUEUE\_SIZE);
49. **while**(1)
50. {
51. FD\_ZERO(&rset) ;
52. FD\_SET(sockL, &rset);
53. **for** (**int** i = 0; i < CLIENTS; i++)
54. **if** (clients[i] != 0)
55. FD\_SET(clients[i], &rset);
57. **int** mx = max(sockL, clients, CLIENTS);
59. **if** (select(mx+1, &rset, NULL, NULL, NULL) <= 0)
60. {
61. perror("Выбор невозможен\n");
62. exit(3);
63. }
65. **if** (FD\_ISSET(sockL, &rset))
66. {
67. **struct** sockaddr\_in client\_addr;
68. **int** addrSize = **sizeof**(client\_addr);
70. **int** sock = accept(sockL, (**struct** sockaddr\*) &client\_addr, (socklen\_t\*) &addrSize) ;
71. **if** (sock < 0)
72. {
73. perror("Сообщение сервера: прием невозможен \n");
74. exit(4);
75. }
77. printf("\nНовое соединение: \nfd = %d \nip = %s:%d\n", sock,inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));
78. **if** (insert(sock, clients, CLIENTS) < 0)
79. {
80. perror("Сообщение сервера: слишком много клиентов!\n");
81. exit(5);
82. }
83. }
84. **for** (**int** i = 0; i < CLIENTS; i++)
85. **if** (FD\_ISSET(clients[i], &rset))
86. recieve\_data(clients, i);
87. }
88. **return** 0;
89. }
91. **void** recieve\_data(**int** clients[], **int** i)
92. {
93. **char** buffer[BUFFER\_SIZE];
94. **int** read = recv(clients[i], buffer, BUFFER\_SIZE, 0);
95. **if** (read <= 0)
96. {
97. close(clients[i]);
98. clients[i] = 0;
99. }
100. **else**
101. {
102. buffer[read] = 0;
103. printf("Сообщение от %d клиенту - %s\n", i, buffer);
104. }
105. }

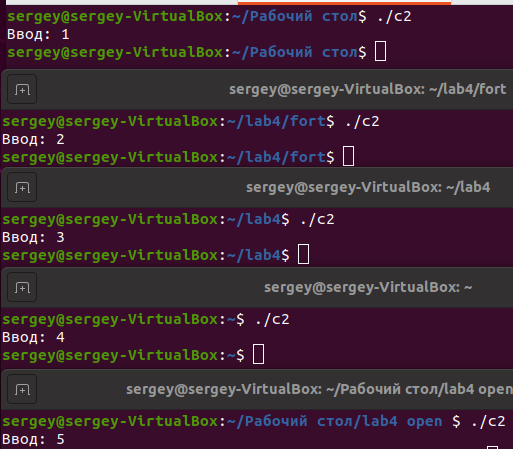
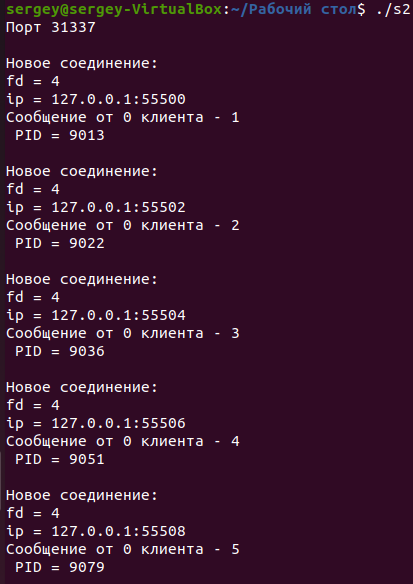
108. **int** max(**int** listener, **int** arr[], **int** count)
109. {
110. **int** res = listener;
111. **for** (**int** i = 0; i < count; i++)
112. **if** (arr[i] > res)
113. res = arr[i];
114. **return** res;
115. }

118. **int** insert(**int** s, **int** clients[], **int** count)
119. {
120. **for** (**int** i= 0; i < count; i++)
121. **if** (clients[i] == 0)
122. {
123. clients[i] = s;
124. **return** 0;
125. }
126. **return** -1;
127. }

Листинг 4: Код клиента

1. #include <sys/types.h>
2. #include <sys/socket.h>
3. #include <netinet/in.h>
4. #include <unistd.h>
5. #include <string.h>
6. #include <stdlib.h>
7. #include <stdio.h>
9. #define SOCKET\_PORT 31337
10. #define BUFFER\_SIZE 512
12. **int** main(**int** argc, **char**\*\* argv)
13. {
14. **int** sock;
15. **struct** sockaddr\_in addr;
17. sock = socket(PF\_INET, SOCK\_STREAM, 0);//SOCK\_STREAM - обеспечивает последовательное, надежное, двустороннее соединение
18. //PF\_INET - IPv6 Internet protocols
20. **if** (sock < 0)
21. {
22. perror("Сообщение клиента: невозможно открыть сокет!\n");
23. exit(1);
24. }
26. addr.sin\_family = PF\_INET;
27. addr.sin\_port = htons(SOCKET\_PORT); //Функция htons() преобразует узловой порядок расположения байтов положительного короткого
28. //целого hostshort в сетевой порядок расположения байтов.
29. addr.sin\_addr.s\_addr = htonl(INADDR\_LOOPBACK); //Функция htonl() преобразует узловой порядок расположения байтов положительного
30. //целого hostlong в сетевой порядок расположения байтов.
32. **if** (connect(sock, (**struct** sockaddr \*)&addr, **sizeof**(addr)) < 0) //connect - системный вызов соединение сокета с другим сокетом
33. {
34. perror("Соединение невозможно!\n");
35. exit (2);
37. }
39. **char** buf[BUFFER\_SIZE];
40. **char** nbuf[BUFFER\_SIZE] ;
41. printf("Ввод: ");
42. scanf("%s", buf);
44. **int** pid = getpid();
45. **char** mypid[6];   // ex. 34567
46. sprintf(mypid, "%d", pid);
48. strcat(buf, "\n PID = ");
49. strcat(buf, mypid);
50. send(sock, buf, strlen(buf), 0);
52. close(sock) ;
53. exit (0);
54. }

### Результат работы программы:





Порт 31337 в режиме «прослушивания», используется процессом «./s2».