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 Набиев Ф.М.

Группа ИУ7–63Б

Оценка (баллы)

Преподаватель Рязанова Н.Ю.

Москва, 2020 г.

1 Задание №1

1.1 Условие

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.

1.2 Реализация

Сокеты, входящие в домен AF_UNIX используют имя сокет-файла в качестве адреса. Программа-сервер создает сокет при помощи функции socket с параметрами AF_UNIX и SOCK_DGRAM. Далее функция bind привязывает сокет к локальному адресу. После этого при помощи системного вызова recvfrom происходит блокировка процесса-сервера в ожидании сообщения от одного из процессов-клиентов. Текст программы-сервера приведён в листинге 1.1.

Листинг 1.1 – Текст программы-сервера

```
1  #include <stdio.h>
2  #include <errno.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <unistd.h>
6  #include <signal.h>
7  #include <sys/types.h>
8  #include <sys/socket.h>
9
10 #define SOCKET_NAME "socket.soc"
11
12 int socket_fd;
13
14 void sigint_handler(int signum);
15
16 int main(void)
```

```
17 {
18     int bytes;
19     char buf[BUFSIZ];
20     struct sockaddr server_name;
21 
22     if (signal(SIGINT, &sigint_handler) == SIG_ERR)
23     {
24         fprintf(stderr, "%s: %s\n", "signal sigint", strerror(errno));
25         exit(1);
26     }
27 
28     if ((socket_fd = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0)
29     {
30         fprintf(stderr, "%s: %s\n", "socket", strerror(errno));
31         exit(1);
32     }
33 
34     server_name.sa_family = AF_UNIX;
35     strcpy(server_name.sa_data, SOCKET_NAME);
36 
37     if (bind(socket_fd, &server_name, sizeof(server_name.sa_family) +
38                                       strlen(server_name.sa_data) + 1) < 0)
39     {
40         fprintf(stderr, "%s: %s\n", "socket", strerror(errno));
41         exit(1);
42     }
43 
44     for(;;)
45     {
46         bytes = recvfrom(socket_fd, buf, sizeof(buf), 0, NULL, NULL);
47 
48         if (bytes < 0)
49         {
50             fprintf(stderr, "%s: %s\n", "recvfrom", strerror(errno));
51             exit(1);
52         }
53 
54         buf[bytes] = 0;
55         printf("%s", buf);
56     }
57 }
58 
59 void sigint_handler(int signum)
60 {
61     if (close(socket_fd) != 0)
62     {
63         fprintf(stderr, "%s: %s\n", "close socket_fd", strerror(errno));
64         exit(1);
```

```
65      }
66
67      if (unlink(SOCKET_NAME) != 0)
68      {
69          fprintf(stderr, "%s: %s\n",
70                  "unlink " SOCKET_NAME, strerror(errno));
71          exit(1);
72      }
73
74      exit(0);
75  }
```

Программа-клиент создает сокет с такими же параметрами, как и программа-сервер. Далее она задаёт тип домена и имя сокет-файла в структуре sockaddr. Отправка сообщения процессу-серверу происходит при помощи функции sendto. Текст программы-клиента приведён в листинге 1.2.

Листинг 1.2 – Текст программы-клиента

```
1   #include <errno.h>
2   #include <stdio.h>
3   #include <string.h>
4   #include <stdlib.h>
5   #include <unistd.h>
6   #include <sys/types.h>
7   #include <sys/socket.h>
8
9   #define SERVER_SOCKET_NAME "socket.soc"
10
11  static inline int get_input(char *buf, size_t len);
12
13  int main(void)
14  {
15      int socket_fd;
16      char buf[BUFSIZ];
17      char input[BUFSIZ];
18      struct sockaddr server_name;
19
20      if ((socket_fd = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0)
21      {
22          fprintf(stderr, "%s: %s\n", "signal sigint", strerror(errno));
23          exit(1);
24      }
25
```

```
26      server_name.sa_family = AF_UNIX;
27      strcpy(server_name.sa_data, SERVER_SOCKET_NAME);
28
29      while (get_input(input, sizeof(input)))
30      {
31          snprintf(buf, sizeof(buf), "Client (pid %d): %s", getpid(), input);
32          sendto(socket_fd, buf, strlen(buf), 0, &server_name,
33                  sizeof(server_name.sa_family) +
34                  strlen(server_name.sa_data) + 1);
35      }
36
37      if (close(socket_fd) != 0)
38      {
39          fprintf(stderr, "%s: %s\n", "close socket_fd", strerror(errno));
40          exit(1);
41      }
42
43      return 0;
44  }
45
46  static inline int get_input(char *buf, size_t len)
47  {
48      printf("> ");
49      fgets(buf, len, stdin);
50
51      return !feof(stdin);
52  }
```

1.3 Пример работы

На рисунке 1.1 изображён пример работы реализованных программ.

Рис. 1.1 – Задание №1

На рисунке 1.2 демонстрируется, что во время выполнения процесса-сервера сокет-файл существует в файловой системе. Для этого используем команду ls с флагом -l.

```
 ~/Documents/Repositories/bmstu/OperatingSystems/lab_16/task01  master ! ? ↑1
# ./server &
[1] 490

 ~/Documents/Repositories/bmstu/OperatingSystems/lab_16/task01  master ! ? ↑1
# "ls" -l
total 68
-rwxr-xr-x 1 faris faris 17344 May 21 16:30 a.out
-rwxr-xr-x 1 faris faris 17344 May 21 16:35 client
-rw-r--r-- 1 faris faris  1175 May 21 16:24 client.c
-rwxr-xr-x 1 faris faris 17288 May 21 01:43 server
-rw-r--r-- 1 faris faris  1576 May 16 22:59 server.c
srwxr-xr-x 1 faris faris     0 May 21 16:52 socket.soc
```

Рис. 1.2 – Проверка существования сокет-файла

2 Задание №2

2.1 Условие

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.

2.2 Реализация

В процессе-сервере при помощи системного вызова socket создается сетевой сокет семейства AF_INET и типа SOCK_STREAM. Системный вызов bind связывает сокет с адресом. Далее процесс-сервер при помощи listen переводится в режим ожидания запроса на соединение от процессов-клиентов. После этого процесс-сервер блокируется системным вызовом select, который возвращает управление при поступлении запроса от процесса-клиента. В этом случае процесс-сервер производит проверку на появление нового соединения от процесса-клиента и при его наличии вызывает функцию _connect. В этой функции устанавливается новое соединение при помощи вызова функции accept и создается еще один сокет, который затем заносится в массив файловых дескрипторов. После этого происходит вызов функции _serve_clients, в которой производится обход массива файловых дескрипторов и проверка, находится ли конкретный файловый дескриптор в наборе дескрипторов (проверка производится при помощи макроса FD_ISSET). Если находится, то при помощи функции read процессом-сервером считывается сообщение от процесса-клиента. Если вызов read вернул нулевое значение, то сокет закрывается и удаляется из массива файловых дескрипторов. Иначе выводится полученное сообщение. Текст программы-сервера приведён в листинге 2.1.

Листинг 2.1 – Текст программы-сервера

```
1  #include <stdio.h>
2  #include <errno.h>
3  #include <unistd.h>
4  #include <signal.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <sys/types.h>
8  #include <sys/socket.h>
9  #include <sys/select.h>
10 #include <netinet/in.h>
11
12 #define LISTENQTY 256
13 #define FEEDBACK_MSG "Ok"
14
15 static int socket_fd;
16 static int greatest_fd;
17 static int greatest_idx;
18
19 static void show_usage(char  *pname);
20 static void handle_sigint(int signum);
21
22 static int _connect(int *client_fds, int *client_pids,
23                     fd_set *fdset);
24 static int _serve_clients(int *client_fds, int *client_pids,
25                           fd_set *fdset, fd_set *reset);
26
27 int main(int argc, char **argv)
28 {
29     int port;
30     struct sockaddr_in server_addr;
31
32     signal(SIGINT, &handle_sigint);
33
34     if (argc < 2)
35     {
36         show_usage(argv[0]);
37         exit(1);
38     }
39     else
40     {
41         char *endptr;
42
43         port = strtol(argv[1], &endptr, 10);
44
45         if (*argv[1] == 0 || *endptr != 0)
```

```
46          {
47              show_usage(argv[0]);
48              exit(1);
49          }
50      }
51
52      greatest_fd = socket_fd = socket(AF_INET, SOCK_STREAM, 0);
53
54      if (socket_fd < 0)
55      {
56          fprintf(stderr, "%s: %s\n", "socket", strerror(errno));
57          exit(1);
58      }
59
60      memset(&server_addr, 0, sizeof(server_addr));
61
62      server_addr.sin_family = AF_INET;
63      server_addr.sin_addr.s_addr = INADDR_ANY;
64      server_addr.sin_port = htons(port);
65
66      if (bind(socket_fd,
67              (struct sockaddr *)&server_addr,
68              sizeof(server_addr)) < 0)
69      {
70          fprintf(stderr, "%s: %s\n", "bind", strerror(errno));
71          exit(1);
72      }
73
74      listen(socket_fd, LISTENQTY);
75
76      int client_fds[FD_SETSIZE];
77      int client_pids[FD_SETSIZE];
78      fd_set fdset;
79      fd_set reset;
80
81      for (int i = 0; i < FD_SETSIZE; ++i)
82          client_fds[i] = -1;
83
84      FD_ZERO(&fdset);
85      FD_SET(socket_fd, &fdset);
86
87      for (;;)
88      {
89          reset = fdset;
90          select(greatest_fd + 1, &reset, NULL, NULL, NULL);
91
92          if (FD_ISSET(socket_fd, &reset))
93              if (_connect(client_fds, client_pids, &fdset) != 0)
```

```
94                  {
95                      fprintf(stderr, "%s: %s\n",
96                              "_connect", strerror(errno));
97                      exit(1);
98                  }
99
100         if (_serve_clients(client_fds, client_pids, &fdset, &reset) != 0)
101         {
102             fprintf(stderr, "%s: %s\n", "_serve_clients", strerror(errno));
103             exit(1);
104         }
105     }
106 }
107
108 static int _connect(int *client_fds, int *client_pids, fd_set *fdset)
109 {
110     int idx;
111     int accepted_fd;
112     char buf[BUFSIZ] = { 0 };
113
114     accepted_fd = accept(socket_fd, NULL, NULL);
115
116     if (accepted_fd < 0)
117         return 1;
118
119     for (idx = 0; idx < FD_SETSIZE && client_fds[idx] >= 0; ++idx);
120     if (idx == FD_SETSIZE)
121     {
122         errno = ENOMEM;
123         return 1;
124     }
125
126     FD_SET(accepted_fd, fdset);
127     client_fds[idx] = accepted_fd;
128
129     if (greatest_fd < accepted_fd)
130         greatest_fd = accepted_fd;
131     if (greatest_idx < idx)
132         greatest_idx = idx;
133
134     client_pids += idx;
135     idx = read(accepted_fd, buf, sizeof(buf) - 1);
136     *client_pids = atoi(buf);
137
138     printf("Client (pid %d) just connected\n", *client_pids);
139
140     return 0;
141 }
```

```
142
143 static int _serve_clients(int *client_fds, int *client_pids,
144                           fd_set *fdset, fd_set *reset)
145 {
146     int read_cnt;
147     char buf[BUFSIZ] = { 0 };
148
149     for (int i = 0; i <= greatest_idx; ++i)
150         if (FD_ISSET(client_fds[i], reset))
151         {
152             read_cnt = read(client_fds[i], buf, sizeof(buf) - 1);
153
154             if (read_cnt > 0)
155             {
156                 write(client_fds[i], FEEDBACK_MSG, strlen(FEEDBACK_MSG));
157
158                 buf[read_cnt] = '\0';
159                 printf("Client (pid %d): %s", client_pids[i], buf);
160             }
161             else
162             {
163                 FD_CLR(client_fds[i], fdset);
164                 close(client_fds[i]);
165                 client_fds[i] = -1;
166                 printf("Client (pid %d) just disconnected\n",
167                        client_pids[i]);
168             }
169         }
170
171     return 0;
172 }
173
174 static void show_usage(char *pname)
175 {
176     fprintf(stderr, "Usage:\n\t%s <port-number>\n", pname);
177 }
178
179 static void handle_sigint(int signum)
180 {
181     close(socket_fd);
182
183     exit(0);
184 }
```

Программа-клиент создает сокет с такими же параметрами, как и программа-сервер. Далее вызывается функция connect, которая устанавливает соединение с сокетом процесса-сервера. При помощи си-

стемного вызова write производится отправка сообщений сообщений процессу-серверу, а при помощи read — получение. Текст программы-клиента приведён в листинге 2.2.

Листинг 2.2 – Текст программы-клиента

```
1  #include <stdio.h>
2  #include <errno.h>
3  #include <netdb.h>
4  #include <unistd.h>
5  #include <signal.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <sys/types.h>
9  #include <sys/socket.h>
10 #include <sys/select.h>
11 #include <netinet/in.h>
12
13 static void show_usage(char  *pname);
14 static int get_input(char *buf, size_t bufsize);
15
16 int main(int argc, char **argv)
17 {
18     int port;
19     int socket_fd;
20     struct hostent *host;
21     struct sockaddr_in server_addr;
22
23     if (argc < 3)
24     {
25         show_usage(argv[0]);
26         exit(1);
27     }
28     else
29     {
30         char *endptr;
31
32         port = strtol(argv[2], &endptr, 10);
33
34         if (*argv[2] == 0 || *endptr != 0)
35         {
36             show_usage(argv[0]);
37             exit(1);
38         }
39
40         host = gethostbyname(argv[1]);
41
```

```
42          if (!host)
43          {
44              fprintf(stderr, "%s: %s\n", "gethostbyname", strerror(errno));
45              exit(1);
46          }
47      }
48
49      socket_fd = socket(AF_INET, SOCK_STREAM, 0);
50
51      if (socket_fd < 0)
52      {
53          fprintf(stderr, "%s: %s\n", "socket", strerror(errno));
54          exit(1);
55      }
56
57      memset(&server_addr, 0, sizeof(server_addr));
58
59      server_addr.sin_family = AF_INET;
60      memcpy(&server_addr.sin_addr, host->h_addr_list[0], host->h_length);
61      server_addr.sin_port = htons(port);
62
63      if (connect(socket_fd,
64                  (struct sockaddr *)&server_addr,
65                  sizeof(server_addr)) < 0)
66      {
67          fprintf(stderr, "%s: %s\n", "connect", strerror(errno));
68          exit(1);
69      }
70
71      int read_cnt;
72      char buf[BUFSIZ] = { 0 };
73
74      snprintf(buf, sizeof(buf), "%d", getpid());
75      write(socket_fd, buf, strlen(buf));
76
77      while (get_input(buf, sizeof(buf)))
78      {
79          write(socket_fd, buf, strlen(buf));
80
81          read_cnt = read(socket_fd, buf, sizeof(buf) - 1);
82          buf[read_cnt] = '\0';
83          printf("feedback: %s\n", buf);
84      }
85
86      close(socket_fd);
87  }
88
89  static int get_input(char *buf, size_t bufsize)
```

```
90  {
91      printf("> ");
92      fgets(buf, bufsize, stdin);
93
94      return !feof(stdin);
95  }
96
97  static void show_usage(char *pname)
98  {
99      fprintf(stderr, "Usage:\n\t%s <host-name> <port-number>\n", pname);
100 }
```

2.3 Пример работы

На рисунке 2.1 изображён пример работы реализованных программ.

Рис. 2.1 – Задание №2

На рисунке 2.2 демонстрируется слушающий сокет и связанный с ним порт. Для этого используется команда ss с флагом -l, который включает отображение только слушающих сокетов, и флагом -t, который включает отображение только TCP сокетов.

```
 ~/Documents/Repositories/bmstu/OperatingSystems/lab_16/task02  master ! ? ↑1
 # ./server 1024&
[1] 17558

 ~/Documents/Repositories/bmstu/OperatingSystems/lab_16/task02  master ! ? ↑1
 # ss -lt
State      Recv-Q      Send-Q           Local Address:Port                 Peer Address:Port
LISTEN     0           128                  127.0.0.1:postgresql               0.0.0.0:*
LISTEN     0           20                   127.0.0.1:smtp                     0.0.0.0:*
LISTEN     0           128                  127.0.0.1:5433                     0.0.0.0:*
LISTEN     0           128                    0.0.0.0:1024                     0.0.0.0:*
LISTEN     0           128                    0.0.0.0:9999                     0.0.0.0:*
LISTEN     0           50                           *:1716                           *:*
LISTEN     0           128                    [::1]:postgresql                 [::]:*
LISTEN     0           20                     [::1]:smtp                       [::]:*
LISTEN     0           128                    [::1]:5433                       [::]:*

 ~/Documents/Repositories/bmstu/OperatingSystems/lab_16/task02  master ! ? ↑1
 # fg
[1]  + 17558 running    ./server 1024
^C%

 ~/Documents/Repositories/bmstu/OperatingSystems/lab_16/task02  master ! ? ↑1
 # ss -lt
State      Recv-Q      Send-Q           Local Address:Port                 Peer Address:Port
LISTEN     0           128                  127.0.0.1:postgresql               0.0.0.0:*
LISTEN     0           20                   127.0.0.1:smtp                     0.0.0.0:*
LISTEN     0           128                  127.0.0.1:5433                     0.0.0.0:*
LISTEN     0           128                    0.0.0.0:9999                     0.0.0.0:*
LISTEN     0           50                           *:1716                           *:*
LISTEN     0           128                    [::1]:postgresql                 [::]:*
LISTEN     0           20                     [::1]:smtp                       [::]:*
LISTEN     0           128                    [::1]:5433                       [::]:*
```

Рис. 2.2 – Демонстрация слушающего сокета