|  |  |
| --- | --- |
| **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**

|  |  |
| --- | --- |
| **Тема** Сокеты  **Студент** Сушина А.Д.  **Группа** ИУ7-61б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю, |  |

Москва.

2020 г

# Задание на лабораторную работу

Лабораторная работа состоит из двух частей:

1. Организовать взаимодействие параллельных процессов на отдельном компьютере.
2. Организовать взаимодействие параллельных процессов в сети (ситуацию моделируем на одной машине).

**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.

**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.

# Ход работы

## Задание 1

Код программ сервера и клиента представлен на листингах 1 и 2.

|  |
| --- |
| Листинг 1. Server.c  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <sys/types.h>  #include <signal.h>  #include <sys/socket.h>  #define MSG\_LEN 256  #define SOCKET\_NAME "socket.soc"  int sock;  void **sigint\_handler**(int signum)  {  close(sock);  unlink(SOCKET\_NAME);  printf("Socket was closed by ctrl+c!\n");  }  int **main**(void)  {  struct sockaddr addr;  sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);  if (sock < 0)  {  perror("Can't open socket!");  exit(1);  }  addr.sa\_family = AF\_UNIX;  strcpy(addr.sa\_data, SOCKET\_NAME);  if (bind(sock, &addr, sizeof(addr)) < 0)  {  printf("Can't bind name to socket!\n");  close(sock);  unlink(SOCKET\_NAME);  perror("Error in bind() ");  exit(-1);  }  printf("\nServer is waiting\n");  signal(SIGINT, sigint\_handler);  char msg[MSG\_LEN];  while(1)  {  int recievedSize = recv(sock, msg, sizeof(msg), 0);  if (recievedSize < 0)  {  close(sock);  unlink(SOCKET\_NAME);  perror("Error in recv(): ");  return recievedSize;  }  msg[recievedSize] = 0;  printf("Client send: %s\n", msg);  }    printf("Closing socket\n");  close(sock);  unlink(SOCKET\_NAME);  return 0;  } |
| Листинг 2. Client.c  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/socket.h>  #include "info.h"  int **main**(void)  {  int sockfd = socket(PF\_LOCAL, SOCK\_DGRAM, 0);  if (sockfd < 0)  {  perror("Can't open socket!");  exit(1);  }  struct sockaddr server\_addr;  server\_addr.sa\_family = AF\_UNIX;  strcpy(server\_addr.sa\_data, SOCKET\_NAME);  char msg[MSG\_LEN];  sprintf(msg, "Hello from client with pid %d\n", getpid());  sendto(sockfd, msg, strlen(msg), 0, &server\_addr, sizeof(server\_addr));  close(sockfd);  return 0;  } |

|  |
| --- |
| Рис 1. Пример работы программы. |

В процессе-сервере с помощью вызова socket() создается сокет семейства AF\_UNIX с типом SOCK\_DGRAM. С помощью системного вызова bind() происходит связка сокета с локальным адресом. Сервер блокируется на функции recv () и ждет сообщения от процессов-клиентов.

В процессе-клиенте создается сокет семейста AF\_UNIX с типом SOCK\_DGRAM с помощью системного вызова socket(). С помощью функции sendto() отправляется сообщение к процессу-серверу.

## Задание 2.

Код программ представлен на листингах 3 и 4.

|  |
| --- |
| Листинг 3. Server.c  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <errno.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <sys/select.h>  #include <arpa/inet.h>  #include <netdb.h>  #define MSG\_LEN 256  #define SOCK\_ADDR "localhost"  #define SOCK\_PORT 9999  #define MAX\_CLIENTS 10  int clients[MAX\_CLIENTS] = { 0 };  void **connectionHandler**(unsigned int fd)  {  struct sockaddr\_in addr;  int addrSize = sizeof(addr);  int incom = accept(fd, (struct sockaddr\*) &addr, (socklen\_t\*) &addrSize);  if (incom < 0)  {  perror("Error in accept(): ");  exit(-1);  }  printf("\nNew connection: \nfd = %d \nip = %s:%d\n", incom,  inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));  for (int i = 0; i < MAX\_CLIENTS; i++)  {  if (clients[i] == 0)  {  clients[i] = incom;  break;  }  }  }  void **clientHandler**(unsigned int fd, unsigned int client\_id)  {  char msg[MSG\_LEN];  memset(msg, 0, MSG\_LEN);  struct sockaddr\_in addr;  int addrSize = sizeof(addr);  int recvSize = recv(fd, msg, MSG\_LEN, 0);  if (recvSize == 0)  {  getpeername(fd, (struct sockaddr\*) &addr, (socklen\_t\*) &addrSize);  printf("User %d disconnected %s:%d \n", client\_id, inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));  close(fd);  clients[client\_id] = 0;  }  else  {  msg[recvSize] = '\0';  printf("Message from %d client: %s\n", client\_id, msg);  }  }  int **main**(void)  {  int sock = socket(AF\_INET, SOCK\_STREAM, 0);  if (sock < 0)  {  perror("Error in sock\n");  return sock;  }  struct sockaddr\_in addr;  addr.sin\_family = AF\_INET;  addr.sin\_port = htons(SOCK\_PORT);  addr.sin\_addr.s\_addr = INADDR\_ANY; //any address for binding  if (bind(sock, (struct sockaddr\*) &addr, sizeof(addr)) < 0)  {  perror("Error in bind\n");  return -1;  }  printf("Server is listening on the %d port!\n", SOCK\_PORT);  if (listen(sock, 3) < 0)  {  perror("Error in listen(): ");  return -1;  }  printf("Wait for the connections\n");  while (1)  {  fd\_set set;  int max\_fd = sock;  FD\_ZERO(&set);  FD\_SET(sock, &set);  for (int i = 0; i < MAX\_CLIENTS; i++)  {  if (clients[i] > 0)  {  FD\_SET(clients[i], &set);  }  max\_fd = (clients[i] > max\_fd) ? (clients[i]) : (max\_fd);  }  int active\_clients\_count = select(max\_fd + 1, &set, NULL, NULL, NULL);  if (active\_clients\_count < 0)  {  perror("No active clients");  return active\_clients\_count;  }  if (FD\_ISSET(sock, &set))  {  connectionHandler(sock);  }  for (int i = 0; i < MAX\_CLIENTS; i++)  {  int fd = clients[i];  if ((fd > 0) && FD\_ISSET(fd, &set))  {  clientHandler(fd, i);  }  }  }  return 0;  } |
| Листинг 4. Client.c  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <time.h>  #include <unistd.h>  #include <signal.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <arpa/inet.h>  #include <netdb.h>  #define MSG\_LEN 256  #define SOCK\_ADDR "localhost"  #define SOCK\_PORT 9999  int **main**(void)  {  srand(time(NULL));  int sock = socket(AF\_INET, SOCK\_STREAM, 0);  if (sock < 0)  {  perror("Error in sock\n");  return sock;  }  struct hostent\* host = gethostbyname(SOCK\_ADDR);  if (!host)  {  perror("Error in gethostbyname\n ");  return -1;  }  struct sockaddr\_in addr;  addr.sin\_family = AF\_INET;  addr.sin\_port = htons(SOCK\_PORT);  addr.sin\_addr = \*((struct in\_addr\*) host->h\_addr\_list[0]);  if (connect(sock, (struct sockaddr\*) &addr, sizeof(addr)) < 0)  {  perror("Error in connect\n");  return -1;  }  char msg[MSG\_LEN];  for (int i = 0; i < 10; i++)  {  memset(msg, 0, MSG\_LEN);  sprintf(msg, "%d message is here!\n", i);  printf("%s", msg);  if (send(sock, msg, strlen(msg), 0) < 0)  {  perror("Error in send(): ");  return -1;  }    printf("Sended %d message\n", i);  int wait\_time = 1 + rand() % 3;  sleep(wait\_time);  }  printf("Client app is over!\n");  return 0;  } |

|  |
| --- |
| Рис 2. Демонстрация работы программы |

В процессе-сервере с помощью вызова socket() создается сокет семейства AF\_INET с типом SOCK\_STREAM. С помощью системного вызова bind() происходит связка сокета с адресом, прописанным в SOCKET\_ADDRESS. С помощью вызова listen() сокету сообщается, что должны приниматься новые соединения. На каждой итерации цикла создается новый набор дескрипторов set. В него заносятся сокет сервера и сокеты клиентов с помощью функции FD\_SET. После этого сервер блокируется на вызове функции select(), она возвращает управление, если хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции. После выхода из блокировки, проверяется наличие новых соединений. При наличии таковых вызывается функция connectHandler. В этой функции с помощью accept() принимается новое соединение, а также создается сокет, который записывается в массив файловых дескрипторов. Затем происходит обход массива дескрипторов, и, если дескриптор находится в наборе дескрипторов, то запускается функция clientHandeler(). В ней осуществляется считывание с помощью recv() и вывод сообщения от клиента. Если recv() возвращает нулевое значение, значит соединение было сброшено. В таком случае выводится сообщение о закрытии сокета.

В процессе-клиенте создается сокет семейста AF\_INETс типом SOCK\_STREAM с помощью системного вызова socket(). С помощью функции gethostbyname() доменный адрес преобразуется в сетевой и с его помощью можно установить соединение, используя функцию connect(). Затем происходит отправка сообщений серверу.