acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . OOP-Her_beginFr :

TCP/IP сокеты.

Рассмотрим общение двух процессов через сокеты и опишем интерфейсы классов для организации такого общения.

На одном компьютерее сначала запускается программа сервер. Изначально мы знаем ip-адрес и номер порта, по которому работает этот сервер.

С другого компьютера запускается программа клиента. Этой программе передается ip-адрес машины (хоста) на которой запущен сервер. Порт известен.

Сервер (программа на С)

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

#define PORTNUM 15000

main(int argc, char **argv)
{
    int s, ns;
    int ppid;
    int nport;
    struct sockaddr_in serv_addr, clnt_addr;
    struct hostent *hp;
    char buf[80], hname[80];

    nport = PORTNUM;
// меняет порядок байт в целом из машинного представления в сетевой
    nport = htons((u_short)nport);

 /*
 socket() - создание коммуникационного интерфейса (гнезда)

      #include

      int socket(int af, int type, int protocol);

      возвращает дескриптор гнезда или -1 в случае ошибки

      af - коммуникационный домен, в зависимости от которого
           интерпретируются адреса в послед. операциях:

           AF_INET   домен взаимодействия удаленных систем.
           PF_INET   использ. протоколы TCP/IP

           AF_UNIX   домен локального межпроцессного взаимодействия
      PF_UNIX   внутри одной опер.системы.
                    использ. внутренние протоколы

      type - тип сокета

           SOCK_STREAM  надежная последовательная двунаправленная
```

```
                        передача потока байтов

          SOCK_DGRAM   передача датаграмм (ненадежная, несвязная
         передача сообщений фиксированной или ограниченной
         длины (обычно небольшой). Только для домена
                        AF_INET

      protocol - протокол, используемый данным гнездом
                 0 - система сама выбирает протокол


 */
    if((s = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("Error in socket() call"); exit(1);
    }
// заполняет нулями  область
    bzero(&serv_addr, (size_t)sizeof(serv_addr));
 // связь по семейству tcp/ip

 /*
 В AF_INET адрес представляется структурой sockaddr_in

      struct sockaddr_in {
          short            sin_family;  # коммуникационный домен - AF INET
          u_short        sin_port;    # номер порта, если 0, назначается
                                      #   автоматически
          struct in_addr sin_addr;    # IP - адрес хоста
          char           sin_zero[8];
      };
 */
    serv_addr.sin_family = AF_INET;
/*    inet_aton("127.0.0.1", &serv_addr.sin_addr);*/

// запросы с любого адреса
    serv_addr.sin_addr.s_addr = INADDR_ANY;
// порт, по которому слушает
    serv_addr.sin_port = nport;
/*
 bind - связывает адрес с гнездом

      #include
      #include
      int bind(int s, const void *addr, int addrlen);

      s       - дескриптор гнезда
      addr    - указатель на адресную структуру
      addrlen - ее длина



      bind() возвращает

            0   при успешном завершении.
           -1   ошибка (устанавливает errno)


*/
    if(bind(s, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == -1)
    {
        perror(" Error in bind() call"); exit(1);
    }


    fprintf(stderr, "Server ready: %s\n", inet_ntoa(serv_addr.sin_addr));

 /*
  listen - информирует систему о том, что сервер готов принимать запросы
               (т.е. просит организовать очередь для запросов к серверу)

      #include
      int listen(int s, int backlog);

      s       - дескриптор гнезда
      backlog - желаемое значение длины очереди
 */
    if(listen(s, 5) == -1)
    {
        perror("Error in listen() call"); exit(1);
    }
```

```
    while(1)
    {
        int addrlen, pid;

        bzero(&clnt_addr, sizeof(clnt_addr));
        addrlen = sizeof(clnt_addr);

/*
  accept - извлекает первый запрос из очереди запросов на
               соединение и создает для него коммуникационное гнездо

      #include
      int accept(int s, void *addr, int *addrlen);

      s       - дескриптор гнезда
      addr    - указатель структуры, в которую будет записан адрес клиента,
                с которым устанавливается соединение
      addrlen - размер адр. структуры

      при успешном выполнении возвращает неотрицательное число,
      которое является дескриптором установленного соединения от
      принятого запроса.
      -1 при ошибке (устанавливается errno)

*/
        if((ns = accept(s, (struct sockaddr *)&clnt_addr, &addrlen)) == -1)
        {
            perror("Error in accept() call"); exit(1);
        }

        fprintf(stderr, "Client @ %s\n", inet_ntoa(clnt_addr.sin_addr));


// Создание дочернего процесса для организации связи с клиентом

        if((pid = fork()) == -1)
        {
            perror("Error in fork() call"); exit(1);
        }
        if(!pid)
        {
       int nbytes;
            int fout;
            close(s);

/*
recv - прием сообщения от гнезда

      #include

      int recv(int s, void *buf, int len, int flags);

      s - дскриптор гнезда

      buf - указатель на буфер, в который принимается сообщение

      len - размер буфера


      recv(), recvfrom() возвращает:

            n   успешное завершение - длина принятого сообщения
            0   гнездо блокировано
           -1   ошибка (устанавливается  errno)


*/
       while((nbytes = recv(ns, buf, sizeof(buf), 0)) != 0)
       {
      if(!strcmp("terminate", buf))
      {
          pid_t  ppid = getppid();
          if(kill(ppid, SIGKILL) == -1)
          {
         perror("Shutdown request failed");
#define FAILED "Failed to shut the server down.\n"
         send(ns, FAILED, strlen(FAILED) + 1, 0);
          }
          else
          {
         fprintf(stderr, "Server has been shut down...\n");
```

```
#define OK "Server has been shut down.\n"
         send(ns, OK, strlen(OK) + 1, 0);
          }
      }
      else
/*
send() - отправить сообщение

      #include

      int send(int s, const void *msg, int len, int flags);

      s - дескриптор гнезда
      msg - указатель на буфер с сообщением
      len - длина сообщения
      flags:
   MSG_OOB - экстренные данные

      send(), sendto() возвращают:

            n   успешное завершение, передано n байтов
           -1   ошибка (устанавливается  errno)

*/
          send(ns, buf, sizeof(buf), 0);
       }
           close(ns);
       exit(0);
        }
        close(ns);
    }
}
```

Клиент (программа на C)

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

#define PORTNUM 15000

main(int argc, char **argv)
{
    int s, ns;
    int pid;
    int i, j;
    struct sockaddr_in serv_addr;
    struct hostent *hp;
    char buf[80];

    int nport = PORTNUM;
    nport = htons((ushort)nport);

    if(argc < 3)
    {
   fprintf(stderr, "USAGE: client <host> <message>\n\t Send 'terminate' to shut the server down\n");
   return 1;
    }

// Преобразует строку имени хоста в ip-адрес
    if((hp = gethostbyname(argv[1])) == 0)
    {
        perror("Error gethostbyname()"); exit(3);
    }
    strcpy(buf, argv[2]);

    bzero(&serv_addr, (size_t)sizeof(serv_addr));
    bcopy(hp->h_addr, &(serv_addr.sin_addr), hp->h_length);
    serv_addr.sin_family = hp->h_addrtype;
    serv_addr.sin_port = nport;

    if((s = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
```

```
        perror(" Error socket()"); exit(1);
    }

    fprintf(stderr, "Server's host address: %s\n", inet_ntoa(serv_addr.sin_addr));

// Соединение с сервером
    if(connect(s, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == -1)
    {
        perror("Error connect()"); exit(1);
    }

    send(s, buf, sizeof(buf), 0);

    if (recv(s, buf, sizeof(buf), 0) < 0)
    {
        perror("Error recv()"); exit(1);
    }
     printf("Received from server: %s\n", buf);
     close(s);
     printf("Client completed\n\n");
}
```

Интерфейсы классов для работы по протоколу TCP/IP

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

#include <iostream>
#include <cstdlib>

#define PORTNUM 15000

class Server{
    int ppid; // родительский pid
    int nport; // номер порта
    struct sockaddr_in serv_addr; // ip сервера
    // массив адресов клиентов
    struct sockaddr_in clnt_addr[2];
// для получения IP-адреса по имени хоста
    struct hostent *hp;
//  для сокета "слушать"
    int s;
// для сокетов клиентов
     int ns[2];
    char buf[80], hname[80];
// когда подключились два клиента
// запускаем дочерний процесс для
// обслуживания игры
// далее сервер продолжает слушать
   void new_game();
// проверка правильности хода
   int check_turn(int x, int y);
public:
//создает порт чтобы слушать подключения
// заполняет информацию serv_addr
   Server(int port_num);
// запускает бесконечный цикл приема запросов
   void start();
// останавливает сервер
// для остановки запускается еще один процесс
// сервера с ключом -stop
// и посылает ему сообщение terminate
// сервер может получить сообщение terminate
// также от клиента
   void stop();
};

class Client{
    struct sockaddr_in serv_addr; // ip сервера
// для получения IP-адреса по имени хоста
    struct hostent *hp;
```

```
    char buf[80];
    int nport;// номер порта
public:
// запускает сервер и подключается к хосту
// с адресом addr
   Client(string addr, int port);
// сообщает серверу о выходе из игры
// закрывает соединение
   ~Client();
// предоставляет возможность делать ходы
// получает и обрабатывает сообщения от
// сервера
   void play();
};
```

Signals

```
#include <iostream>
#include <signal.h>
#include <unistd.h>
#include <cstring>
#include <atomic>

std::atomic<bool> quit(false);    // signal flag

void got_signal(int)
{
    quit.store(true);
};

class SigCheck
{
public:
      SigCheck();
      void zzz();
    ~SigCheck();
};

SigCheck::SigCheck(){
 std::cout<<"Конструктор\n";
};

void SigCheck::zzz(){
   while(1){
      if( quit.load() )
         return;
   };
};

SigCheck::~SigCheck()
{
   std::cout << "\nDestructor\n";
};

int main(void)
{
    struct sigaction sa;
    memset( &sa, 0, sizeof(sa) );
    sa.sa_handler = got_signal;
 //   sigfillset(&sa.sa_mask);
    sigaction(SIGINT,&sa,NULL);

    SigCheck a;    // needs destruction before exit

        // do real work here...
       a.zzz();

    return 0;
}
```

```
#include <iostream>
#include <signal.h>
#include <unistd.h>
#include <cstring>
#include <atomic>

class SigEx{
   int signal;
   public:
```

```
      SigEx(int);
      int getSignal();
};

SigEx::SigEx(int a)
{
   signal = a;
};

int SigEx::getSignal()
{

   std::cout<<"Signal\n";
   return signal;
};


std::atomic<bool> quit(false);    // signal flag

void got_signal(int)
{
    quit.store(true);
};

class SigCheck
{
public:
      SigCheck();
      void zzz();
    ~SigCheck();
};

SigCheck::SigCheck(){
 std::cout<<"Конструктор\n";
};

void SigCheck::zzz(){
   int p;
   while(1){
      if( (p = quit.load()) )
      {
         throw(SigEx(p));
      }
   };
};

SigCheck::~SigCheck()
{
   std::cout << "\nDestructor\n";
};

int main(void)
{
    struct sigaction sa;
    memset( &sa, 0, sizeof(sa) );
    sa.sa_handler = got_signal;
    sigfillset(&sa.sa_mask);
    sigaction(SIGINT,&sa,NULL);

    SigCheck a;    // needs destruction before exit
         try
         {
        // do real work here...
        a.zzz();
      } catch (SigEx & e) {
              std::cout<< e.getSignal();
            return 0;
         }
    return 0;
}
```

Задачи

Задача 1.

Реализовать все функции всех классов и проверить их работу. Сервер дожен запускаться в Вашем контейнере, а клиент на любой другой машине в сети mipt. Оформить все функции в библиотеки (статические).

Задача 2.

Реализовать игру в "крестики-нолики" или "морской бой" по сети.

Задача 3.*

На шахматной доске (8x8) на первой линии в центре на черной клетке стоит фишка-титаник. Титаник может ходить на одну любую ближайшую клетку по диагонали. На последней линии на 4 черных клетках стоят айсберги. Каждый айсберг может ходить по диагонали на черную клетку только вперед. Очередность ходов: первых ход делает титаник, затем айсберги делают по одному ходу. Пропускать ход не может никто.

Задача титаника – добраться до последней линии. Задача айсбергов запереть титаник так, чтобы он не мог больше сделать ход.

Реализовать игру по сети. Каждый айсберг и титаник – отдельный клиент.

-- TatyanaOvsyannikova2011 - 24 Nov 2016

(c) Материалы раздела "Язык Си" публикуются под лицензией GNU Free Documentation License.