# АРХИТЕКТУРА СУРВЕРА

## Блок – функциональная архитектура сервера

Блок – функциональная архитектура сервера представима укрупненной блок – функциональной схемой типичного TCP/IP (Transmission Control Program / Internet Protocol) сервера. Такая схема представлена на следующем рисунке:

ПЛАНИРОВЩИК

ДИСПЕТЧЕР

АГЕНТ

КЛИЕНТА 1

АГЕНТ

КЛИЕНТА 2

АГЕНТ

КЛИЕНТА N

Библиотека

системных и

прикладных

сервисов

TCP/IP

Обобщенная блок – функциональная схема сервера

В данной схеме показано интеграция стандартных функциональных блоков типового TCP/IP сервера с прикладными функциями, которые реализуются разработчиком, когда определяются конкретное прикладное назначение сервера. При этом на соответствующие функциональные блоки возлагаются следующие задачи:

Таблица №ХХ

|  |  |
| --- | --- |
| **НАЗВАНИЕ БЛОКА** | **ВЫПОЛНЯЕМЫЕ ФУНКЦИИ ИЛИ ЗАДАЧИ** |
| БЛОКИ СЕРВЕРА, КОТОРЫЕ ИСПОЛЬЗУЮТСЯ БЕЗ МОДИФИКАЦИЙ | |
| **TCP/IP** | Данный блок представляет собой набор унифицированных библиотек, который, как правило, поставляется с некоторой технологией программирования. Этот набор представляет стандартные TCP/IP средства подключения и отключения клиентов в штатном и аварийном режимах, выполнения операций сетевого ввода вывода, подключения различных обработчиков стандартных и аварийных событий. |
| БЛОКИ СЕРВЕРА, ФУНКЦИИ КОТОРЫХ РАСШИРЯЮТСЯ ИЛИ МОДИФИЦИРУЮТСЯ ПРОГРАММИСТОМ | |
| ПЛАНИРОВЩИК | В сложных серверах на этот блок возлагаются задачи планирования выполнения самых различных запросов от клиентов, а также планирования использования ресурсов в условиях динамически изменяющейся нагрузки на сервер. В относительно простых серверах функции планирования не выделяются в отдельную подсистему, а реализуются в отдельных функциях или процедурах. |
| ДИСПЕТЧЕР | В сложных серверах на этот блок возлагаются задачи интерпретации планов, составляемых планировщиком, задачи измерения нагрузки и управление приоритетами, задачи обслуживания различных событий в системе. Однако основной задачей диспетчера является управление агентами клиентов. В более простых серверах, задача управления агентами клиентов является доминирующей. |
| АГЕНТ  КЛИЕНТА | Агенты клиентов являются самостоятельными паралельными процессами или задачами, которые создаются и диспетчируются диспетчером для каждого принятого на обслуживание клиента. Данные процессы или задачи являются системной оболочкой для прикладных данных и функций предоставляемых конкретному клиенту с конкретным набором прав доступа. |
| БЛОКИ СЕРВЕРА, ОПРЕДЕЛЯЮЩИЕ ЕГО ПРИКЛАДНОЕ НАЗНАЧЕНИЕ. | |
| **Библиотека системных и прикладных сервисов** | Как правило это совокупность прикладных функций или прикладных подсистем, которые доступны агентам клиента для выполнения текущих запросов от клиентов. К относительно небольшой системной группе сервисов относятся сервисы общие для всего многообразия клиентских запросов. |

В рамках данной разработки, в качестве блок – функциональной основы для реализации сервера, была использована технология INDY, которая стандартно входит в состав системы программирования DELPHI ver.7. Отображение блок – функциональной схемы на инструментарий предоставляемый технологией INDY представлено в описании модульной архитектуры разработанного сервера.

## Модульная архитектура сервера

### Unit MAIN (Главный модуль сервера)

**Использует стандартные юниты Delphi :** Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, CheckLst, ComCtrls, ExtCtrls,

**стандартные юниты технологии INDY и юниты собственной разработки.**

Unit IdComponent

Unit IdThreadMgr

Unit IdThreadMgrDefault

Unit IdBaseComponent

**Unit IdTCPServer**

#### Юниты сервера

Unit IdSocketHandle

Unit IdStack

Unit IdGlobal

Unit IdResourceStrings

#### Юниты сервера

**Unit Common1**

**Unit SrviceLibrary1**

Юниты интерфейсов связи сервера с прикладными подсистемами

**Unit SensorGroup01**

**Unit SensorGroup02**

**Unit SensorHorizon01**

Юниты прикладных подсистем сервера

**Группа библиотек, технологии INDY в составе DELPHI**

**Группа библиотек собственной разработки**

Таблица №ХХ

|  |  |
| --- | --- |
| **НАЗВАНИЕ ЮНИТА** | **НАЗНАЧЕНИЕ**  **ЮНИТА** |
| ОСНОВНЫЕ СИСТЕМНЫЕ ЮНИТЫ СЕРВЕРА | |
| **MAIN** | Main.pas. **Главный модуль разработанного сервера.** Содержит типы, процедуры и функции, необходимые для определения и реализации конкретной архитектуры и реализации TCP/IP сервера. |
| **IdTCPServer** | TIdTCPServer.pas содержит классы, типы, процедуры и функции, необходимые для определения и реализации многопоточного сервера с использованием транспорта TCP/IP (Transmission Control Program / Internet Protocol). |
| ИНТЕРФЕСНЫЕ ЮНИТЫ ИНТЕГРАЦИИ В СЕРВЕР ПРИКЛАДНЫХ ПОДСИСТЕМ | |
| **Common1** | Common1.pas содержит типы буферов и сами буфера для размещения результатов выполнения запросов от клиентов. |
| **SrviceLibrary1** | SrviceLibrary1.pas содержит процедуры планирующие и исполняющие операции по запросу клиентов. |
| ЮНИТЫ ПРИКЛАДНЫХ ПОДСИСТЕМ СЕРВЕРА | |
| **SensorGroup01** | SensorGroup01.pas. **Главный модуль прикладной подсистемы №01.** Данная подсистема является имитатором информационно-измерительной системы, которая оцифровывает показания группы датчиков №01. |
| **SensorGroup02** | SensorGroup02.pas. **Главный модуль прикладной подсистемы №01.** Данная подсистема является имитатором информационно-измерительной системы, которая оцифровывает показания группы датчиков №02. |
| **SensorHorizon01** | SensorHorizon01.pas. **Главный модуль прикладной подсистемы №03.** Данная подсистема является имитатором информационно-измерительной системы, которая оцифровывает показания текущих значений тангажа и крена. |

В нашем случае функции планирования и диспетчирования, которые требуют реализации сосредоточены в юните **MAIN:**

**Функция планирования**. В нашем случае функция планирования носит максимально простой характер и обеспечивается как возможность подключения обработчиков событий к стандартными событиям в **IdTCPServer** , таким как: **connect, disconnect, execute**. По сути названные обработчики событий представляют собой функции диспетчирования для выполнения процедур подключения или отключения клиента (connect, disconnect) либо активации процесса агента для конкретного клиента и обмена с этим клиентом информацией (execute).

**Функция диспетчирования**. В нашем случае основная функция диспетчера сводится к управлению агентами клиентов. Для коммутации запросов конкретного клиента к его агенту используется событие (execute), которое предоставляет агенту адрес нити и блока управления этим агентом. В теле обработчика такого события выполняется чтение запроса от клиента, его проверка, вызов прикладной системы обслуживания запроса и отправка клиенту результатов обслуживания.

## Функциональная структура главного модуля сервера

Функциональная структура модуля **MAIN** представлена в следующей таблице:

Таблица №ХХ

|  |  |
| --- | --- |
| **ФУНКЦИЯ** | **ОПИСАНИЕ** |
| ОБЩИЙ СЕРВИС | |
| **function** CutNextCmdField(var Cmd : string) : string; | Вырезать из текста команды очередное поле |
| **function** TServerForm.PortDescription(const PortNumber: integer): string; | По номеру порта получить описание соответствующей (Berkeley standard) службы. Используются юниты: IdStack |
| **procedure** TServerForm.PopulateIPAddresses(); | Сформировать список IP-адресов сетевых карт в chboxIPs, а также сформировать в cbboxViewPorts список номеров иописаний стандартных служб на этих портах. Используются юниты: IdGlobal, IdStack, IdResourceStrings |
| **procedure** TServerForm.cbboxViewPortsChange (Sender: TObject); | Изменить номер порта в edtPort |
| ИНИЦИИРОВАНИЕ / ТЕРМИНИРОВАНИЕ | |
| **function** TServerForm.UpdateBindings (RqListIP: TCheckListBox; RqPort : string) : boolean; | Обновление подключений сервера к Socket на выбранный порт. |
| **function** TServerForm.GetClientCount() : integer; | Получить количество активных клиентов (коннектов) |
| **function** TServerForm.ServerActivate(RqActivate: boolean) : boolean; | Активация и деактивация сервера. |
| **procedure** TServerForm.UpdateClientsList (RqList: TListBox; RqReport : TMemo); | Обновить список агентов на сервере |
| **procedure** TServerForm.TCPServerDisconnect (AThread: TIdPeerThread); | Обработчик события - отключение клиента |
| CONNECT / DISCONNECT | |
| **procedure** TServerForm.TCPServerConnect (AThread: TIdPeerThread); | Проверка и подключение нового клиента, создание объекта клиентского агента на сервере |
| **procedure** TServerForm.DisconnectClient (AThread: TIdPeerThread); | Выполнить Disconnect для указанного клиета |
| **function** TServerForm.RqDisconnectAllClients(): boolean; | Принудительное отключение всех клиентов |
| EXECUTE | |
| **procedure** TServerForm.TCPServerExecute (AThread: TIdPeerThread); | Обработчик события "Клиент выдал заявку на обслуживание" |
| **procedure** TServerForm.btnOpenSensorGroupClick (Sender: TObject); | Открыть выбранную форму для работы с имитаторами датчиков |
| УПРАВЛЕНИЕ СЕРВЕРОМ | |
| **procedure** TServerForm.FormCreate (Sender: TObject); | Инициализация приложения сервер |
| **procedure** TServerForm.Button1Click (Sender: TObject); | Активация сервера |
| **procedure** TServerForm.Button2Click (Sender: TObject); | Деактивация сервера |
| **procedure** TServerForm.lstboxAllClientsClick (Sender: TObject); | Отобразить последнюю операцию выбранного клиента |
| **procedure** TServerForm.FormClose (Sender: TObject; var Action: TCloseAction); | Обслуживание попыток терминировать сервер при подключенных клиентах |

### Полный исходный текст procedure TCPServerExecute (AThread: TIdPeerThread)

// Обработчик события "Клиент выдал заявку на обслуживание"

procedure TServerForm.TCPServerExecute(AThread: TIdPeerThread);

var wAgent : TAgent;

wMsgSRC : string;

wMsg : string;

wSysCmd : string;

wUserCmd : string;

wUCmdPrm : string;

wRep : string;

wRequest : TRequest; // Запрос к диспетчеру прикладных сервисов

begin

// ===========================================

// Получить ссылку на агента

wAgent := TAgent(AThread.Data);

// Прочитать полный текст заявки от клиента

wMsgSRC := AThread.Connection.ReadLn;

wMsg := wMsgSRC;

// Выделить в заявке системную командную часть

wSysCmd := CutNextCmdField(wMsg);

// Сохранить полный текст заявки от клиента

if wSysCmd <> cCmdLink

then if Assigned(wAgent)then wAgent.ClientMsg := 'Запрос : ' + wMsgSRC;

// ===========================================

// Запрос от клиента на подтверждение коннекта

if wSysCmd = cCmdLink

then begin

// Выдать клиенту позитивную квитанцию

wRep := cRepYES + cFieldSep + 'Сервер подтверждает коннект';

**AThread.Connection.WriteLn(wRep);**

Exit;

end;

// -------------------------------------------

// Запрос от клиента на дисконнект

if wSysCmd = cCmdDisConnect

then begin

// Выдать клиенту позитивную квитанцию

wRep := cRepYES + cFieldSep + 'Сервер принял запрос на дисконнект';

**AThread.Connection.WriteLn(wRep);**

// Выполнить дисконнект клиента

DisconnectClient(AThread);

// Обновить список агентов на сервере

UpdateClientsList(lstboxAllClients, Memo1);

Exit;

end;

// ===========================================

// Запрос от клиента на получение сервиса

if wSysCmd = cCmdService

then begin

// Выделить в заявке код команды пользователя

wUserCmd := CutNextCmdField(wMsg);

// Выделить в заявке параметры команды пользователя

wUCmdPrm := CutNextCmdField(wMsg);

// Подготовить запрос на прикладное обслуживание

wRequest.RqCmd := wUCmdPrm;

// Выполнить прикладное обслуживание

if UserDispatch(wRequest)

then begin

// Отправить клиенту позитивную квитанцию

wRep := cRepYES;

**AThread.Connection.WriteLn(wRep);**

// Если клиент запросил на блок данных ( cCmdGET )

if wUserCmd = cCmdGET

then begin

// Отправить клиенту результат обслуживания

**AThread.Connection.WriteBuffer(wRequest.Buffer^,**

**wRequest.BufSize,**

**True);**

end;

end

else begin

// Отправить клиенту негативную квитанцию

wRep := cRepNOT;

**AThread.Connection.WriteLn(wRep);**

end;

// Сохранить последнее сообщение клиенту от сервера

if Assigned(wAgent)then wAgent.ServerMsg := 'Квитанция : ' + wRep;

end;

// ===========================================

end;