**«Санкт-Петербургский государственный электротехнический университет**

**«ЛЭТИ» им. В.И.Ульянова (Ленина)»**

**(СПбГЭТУ «ЛЭТИ»)**

|  |  |  |
| --- | --- | --- |
| **Направление** | 09.04.04 – Программная инженерия | |
| **Профиль** | Разработка распределенных программных систем | |
| **Факультет** | ФКТИ | |
| **Кафедра** | МО ЭВМ | |
| *К защите допустить* |  | |
| Зав. кафедрой |  | А.А. Лисс |

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

магистра

**Тема:** Проектирование и разработка программного обеспечения системы внешних связей ЦВК ГАС на отечественных вычислительных средствах

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  |  |  | Ф.Д. Бочаров |
|  |  | *подпись* |  |  |
| Руководитель |  |  |  | Н.В. Мальцева |
|  | *(к.т.н., доцент)* | *подпись* |  |  |
| Консультанты |  |  |  | А.М. Шаторная |
|  |  | *подпись* |  |  |
|  |  |  |  | И.И. Иванов |
|  | *(Уч. степень, уч. звание)* | *подпись* |  |  |
|  |  |  |  | И.И. Иванов |
|  | *(Уч. степень, уч. звание)* | *подпись* |  |  |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на выпускную квалификационную работу**

|  |  |
| --- | --- |
|  | Утверждаю |
|  | Зав. кафедрой МО ЭВМ |
|  | \_\_\_\_\_\_\_\_\_\_\_\_ А.А. Лисс |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент | Бочаров Ф.Д. | | | |  | Группа | 8303 | |
| Тема работы: Проектирование и разработка программного обеспечения системы внешних связей ЦВК ГАС на отечественных вычислительных средствах. | | | | | | | | |
| Место выполнения ВКР: АО «Концерн «Океанприбор» | | | | | | | | |
| Исходные данные (технические требования): Проанализировать аппаратное и программное обеспечение комплекса. Изучить возможности интерфейсов связи, протоколы связей. Исследовать возможные варианты структуры программного обеспечения системы внешних связей. Спроектировать, реализовать и протестировать программное обеспечение. | | | | | | | | |
| Содержание ВКР: Исследование предметной области и постановка задачи, интерфейсы каналов связи, разработка программного обеспечения, разработка имитатора, тестирование программного обеспечения, специальные вопросы обеспечения безопасности, заключение. | | | | | | | | |
| Перечень отчетных материалов: пояснительная записка, иллюстративный материал. | | | | | | | | |
| Дополнительные разделы: специальные вопросы обеспечения безопасности. | | | | | | | | |
| Дата выдачи задания | | | Дата представления ВКР к защите | | | | | |
| «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. | | | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. | | | | | |
| Студент | |  | | Ф.Д. Бочаров | | | |  |
| Руководитель | |  | | Н.В. Мальцева | | | |  |
| *(к.т.н., доцент)* | |  | |  | | | |  |
| Консультант | |  | | А.М. Шаторная | | | |  |

**календарный план выполнения**

**выпускной квалификационной работы**

|  |  |
| --- | --- |
|  | Утверждаю |
|  | Зав. кафедрой МО ЭВМ |
|  | \_\_\_\_\_\_\_\_\_\_\_\_ А.А. Лисс |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | Бочаров Ф.Д. |  | Группа | 8303 |
| Тема работы: Проектирование и разработка программного обеспечения системы внешних связей ЦВК ГАС на отечественных вычислительных средствах. | | | | |

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование работ | Срок выполнения |
| 1 | Обзор литературы по теме работы | 00.00 – 00.00 |
| 2 | Исследование предметной области и постановка задачи | 00.00 – 00.00 |
| 3 | Интерфейсы каналов связи | 00.00 – 00.00 |
| 4 | Разработка программного обеспечения | 00.00 – 00.00 |
| 5 | Разработка имитатора |  |
| 6 | Тестирование ПО |  |
| 7 | Специальные вопросы обеспечения безопасности |  |
| 8 | Оформление пояснительной записки | 00.00 – 00.00 |
| 9 | Оформление иллюстративного материала | 00.00 – 00.00 |

|  |  |  |
| --- | --- | --- |
| Студент |  | Ф.Д. Бочаров |
| Руководитель |  | Н.В. Мальцева |
| *(к.т.н., доцент)* |  |  |
| Консультант |  | А.М. Шаторная |

**РЕФЕРАТ**

Пояснительная записка 65 стр., 19 рис., 16 табл., 24 ист.

ЦВК, ГАС, СВС, многопоточность.

**Объектом исследования** является ....

**Предметом исследования** является ....

**Цель работы:** является проектирование и разработка ПО СВС для ЦВК гидроакустической станции, используя язык программирования Си.

В работе проводится анализ характеристик отечественных вычислительных средств, интерфейсов каналов связи. Проводится анализ требований к программному обеспечению. Исходя из этого избирается модель разработки и структура ПО. После разработки для первоначального тестирования программного обеспечения выбирается структура имитатора данных. При помощи разработанного имитатора производится автономное и комплексное тестирование программного обеспечения системы внешних связей.

**ABSTRACT**

**Содержание**

[Определения, обозначения и сокращения 9](#_Toc166017147)

[Введение 11](#_Toc166017148)

[1. Глава 1. Исследование предметной области и постановка задачи 14](#_Toc166017149)

[1.1. Характеристики ВК-27.07 16](#_Toc166017150)

[1.2. Многопоточность 22](#_Toc166017151)

[1.3. Система обмена 27](#_Toc166017152)

[2. Глава 2. Интерфейсы каналов связи 29](#_Toc166017153)

[2.1. Интерфейс Манчестер (Manchester)/ГОСТ Р 52070-2003 29](#_Toc166017154)

[2.2. Интерфейс Ethernet, протоколы UDP, TCP 36](#_Toc166017155)

[2.3. Интерфейс RS-232 43](#_Toc166017156)

[3. Глава 3. Разработка программного обеспечения 46](#_Toc166017157)

[3.1. Анализ требований к программному обеспечению и выбор структуры 47](#_Toc166017158)

[3.2. Разработка ПО 53](#_Toc166017159)

[4. Разработка имитатора 62](#_Toc166017160)

[4.1. Анализ требований 62](#_Toc166017161)

[4.2. Выбор структуры имитатора 62](#_Toc166017162)

[5. Тестирование ПО 67](#_Toc166017163)

[5.1. Автономное тестирование 67](#_Toc166017164)

[5.2. Тестирование ПО на одном и двух вычислительных комплексах 67](#_Toc166017165)

[5.3. Тестирование ПО на взаимодействие с задачами ЦВК ГАС и СОРДиУ 69](#_Toc166017166)

[5.4. Результаты тестирования 70](#_Toc166017167)

[Заключение 73](#_Toc166017168)

[Список использованных источников 74](#_Toc166017169)

**А глава 2 не входит в исследование предметной области?**

# Определения, обозначения и сокращения

СВС – система внешних связей

ГАК – гидроакустический комплекс

ЦВК – цифровой вычислительный комплекс

ГАС – гидроакустическая станция

ПО – программное обеспечение

ОС – операционная система

ОПО – общее программное обеспечение

CПО – системное программное обеспечение

ФПО – функциональное программное обеспечение

СД – слова данных

КШ – Контроллер шины (Bus Controller(BC)) – устройство интерфейса, управляющее обменом информации в интерфейсе

ОУ – Оконечное устройство (Remote Terminal(RT)) – устройство интерфейса, функционирующее в соответствии с командами КШ

ИУ – интерфейсные устройства

ЛВС – локальная вычислительная сеть

СЕВ – система единого времени

НК – навигационный комплекс

ПУ – пульт управления

НК – надводный корабль

ВК – вычислительный комплекс

ТУ – технические условия

КС – командное слово (Command Word, CW).

СД – слово данных (Data Word, DW).

ОС – ответное слово (Status Word, SW).

КУ – команда управления (Mode code).

СКВ – система контроля версий.

UDP – User Datagram Protocol (Протокол пользовательских дейтаграмм)

TCP – Transmission Control Protocol (Протокол управления передачей)

ЯП – язык программирования

Сообщение – последовательность передаваемых командных и ответных слов, а также слов данных (при их наличии).

ЛП – легковесный процесс

СОРДиУ – система отображения, регистрации, документирования и управления

ЦП – центральный процессор

# Введение

В судостроении гидроакустика – это область, связанная с изучением и применением акустических явлений. Она изучает распространение звука в воде и его взаимодействие с окружающей средой. Гидроакустика играет важную роль в судостроении и позволяет обеспечить безопасность на воде. Гидроакустические системы судна позволяют определять его местоположение, а также используются в оборонных целях и могут обнаруживать и рассчитывать дистанцию до подводных лодок, мин и препятствий.

К гидроакустическим средствам относят эхолоты, гидроакустические антенны и локаторы, гидроакустические комплексы. Гидроакустические комплексы, которые могут содержать в своем составе одну или несколько гидроакустических станций, используют в различных целях, обнаружение подводных и надводных объектов, обработка и анализ гидроакустических данных, исследование морской среды, измерения гидроакустических параметров и т.п.

Цифровой вычислительный комплекс (ЦВК) [1] современной гидроакустической станции (ГАС) представляет собой многопроцессорную, многофункциональную вычислительную систему, обеспечивающую прием и обработку цифровых сигналов, управление ГАС, выделение полезной информации для оператора, взаимодействие с другими системами. Аппаратное и программное обеспечение, входящее в состав комплекса, имеет высокие показатели производительности, большую пропускную способностью каналов обмена, а также параллельную обработку данных.

На судне существуют различные корабельные системы. Каждая из этих корабельных систем вырабатывает определенную информацию в зависимости от своего назначения, например, навигационные параметры движения, метеоданные и т.д. Системам судна для их корректной работы необходимы как раз эти данные, выработанные другими системами. Вследствие этого определенные корабельные системы связывают между собой: прокладывают требуемые линии связи. Обмен данными ГАС с другими системами корабля обеспечивает программное обеспечение (ПО) системы внешних связей (СВС). Среди всех задач комплекса, эта система является крайне важной частью ПО ЦВК. Обмен данными между ЦВК и внешними системами осуществляется по основному и/или резервному каналу интерфейсов Манчестер, Ethernet и RS-232.

Для работы ГАС необходимы данные, от других систем корабля, такие как время, метеоданные, навигационные данные, глубина, скорость звука и обнаруженные цели другими средствами. Эти данные используются при обработке информации. Для этого используется трехуровневый подход:

• *первичная обработка* используется для пространственно-временного спектрального анализа формирования диаграмм направленности. На этом уровне происходит обнаружение сигнала и координат;

• *вторичная обработка* служит для определения параметров движения целей, классификации целей, уточнения координат, построения траекторий движений целей;

• *третичная обработка* – это совместная обработка данных от нескольких систем и комплексов, выработка комплексных формуляров целей.

Полученные формуляры обнаруженных объектов передаются в другие системы корабля, например, для целеполагания.

ЦВК ГАС построено на базе отечественных вычислительных средств «Эльбрус-90микро» ВК-27.07 (ТВГИ.466535.015-27.07) (далее – ВК-27.07), производства АО «МЦСТ». ЦВК состоит из нескольких приборов обработки информации и нескольких пультовых приборов.

Работа является актуальной, поскольку в разных заказах ПО СВС может значительно отличаться, из-за различий в возможностях и составе оборудования, внешних систем, протоколов связи, поэтому на этапе разработки необходимо заново проектировать и реализовывать ПО СВС.

**Целью работы** является проектирование и разработка ПО СВС для ЦВК гидроакустической станции, используя язык программирования Си. Для достижения цели необходимо решить следующие задачи:

* Анализ аппаратного и программного обеспечения комплекса.
* Изучение возможностей интерфейсов и программного обеспечения, используемого при обращении к ним.
* Изучение протоколов связей, структур входных и выходных данных, формата их представления, частоты обмена данными.
* Исследование возможных вариантов структуры ПО СВС и имитатора внешних связей. Проектирование ПО СВС и имитатора внешних связей.
* Реализация и тестирование ПО СВС.
* Реализация и тестирование имитатора ПО СВС.

Работа является практически ценной, так как разработанное ПО будет использовано в актуальном заказе.

В *главе 1* проводится исследование предметной области и постановка задачи.

*Глава 2* посвящена интерфейсам каналов связи.

В *главе 3* проектируется и разрабатывается программное обеспечение системы внешних связей.

В *главе 4* проектируется и разрабатывается имитатор СВС.

В *главе 5* производится отладка разработанного ПО СВС и имитатора

В *главе 6* …доп. Раздел

# Глава 1. Исследование предметной области и постановка задачи

*Система внешних связей* – это элемент изделия, обеспечивающий взаимодействие гидроакустической станции с другими системами, приборами и комплексами корабля. От качества приема информации от внешних систем и своевременности ее доставки абонентам цифрового вычислительного комплекса зависит качество обработки гидроакустической информации, а значит и потенциал комплекса. Своевременная отправка пользователям выработанной в комплексе информации определяется эффективностью работы гидроакустической станции, поэтому необходимо уделить особое внимание грамотному проектированию и реализации ПО СВС.

Обмен с внешними системами должен быть реализован согласно протоколами обмена с внешними системами, при использовании средств связи таких как технологии пакетной передачи данных Ethernet, стандарт физического уровня для асинхронного интерфейса RS-232/485, а так же интерфейс магистральный последовательной системы электронных модулей Манчестер/ГОСТ Р 52070-2003 [2].

На рисунке 1 изображен пример схемы взаимодействия устройств гидроакустической станции. Прибор включает в себя ВК модуль Манчестер, модуль RS, модуль Ethernet, и имеет связь с одним прибором через модуль Манчестер, а с другим через модуль RS. Также существует внешний абонент – прибор, для связи с которым используется модуль Ethernet и протоколы UDP/TCP.

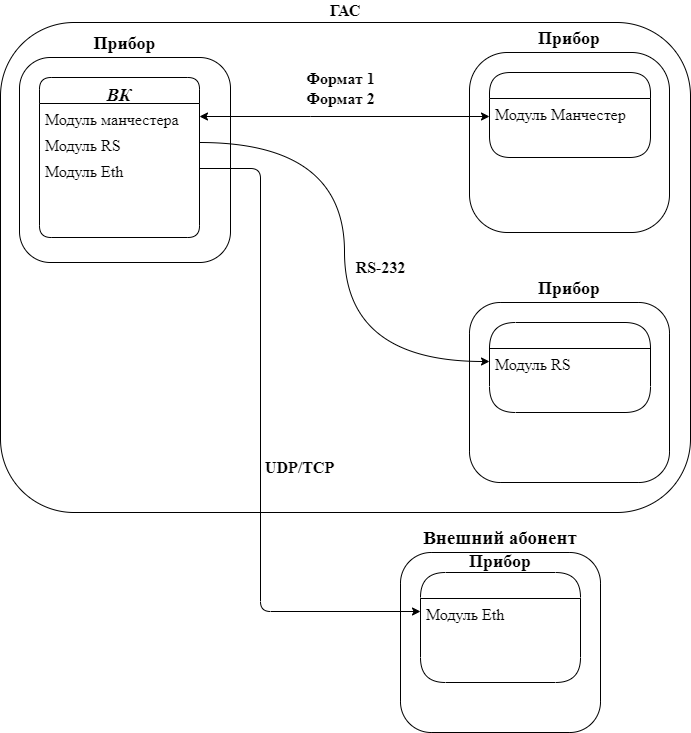


Рисунок 1 – Условная схема взаимодействия ГАС

При приеме внешней информации необходимо преобразовать её из сетевого вида и доставить прикладным задачам. Для выдачи информации необходимо принять её от прикладных задач, преобразовать в сетевой вид и выдать во внешние системы. Прием и передачу информации выполняется в реальном времени. Проверка связей с внешними системами производится на этапе комплексного тестирования изделия на заказе.

На новых заказах изменяются внешние системы, протоколы связи, вследствие этого его необходимо заново проектировать и разрабатывать, даже если сохраняется прежняя аппаратно-программная платформа. Для предварительной проверки корректности работы ПО необходимо разработать имитатор внешних систем, который позволит протестировать ПО до этапа комплексного тестирования изделия на надводном корабле (НК). Во время тестирования ПО с помощью имитатора можно обнаружить и устранить ошибки реализации алгоритмов и функций, что облегчит отладку ПО на испытаниях в реальных условиях. Имитатор должен обеспечить реализацию в реальном времени протоколов связи.

Программная часть ЦВК ГАС реализована на ВК-27.07, программно-аппаратная структура этого вычислительного средства предоставляет более широкие возможности для проектирования и разработки многопоточного ПО. Далее рассмотрим характеристики отечественного вычислительного средства ВК-27.07.

## 1.1. Характеристики ВК-27.07

«Эльбрус-90микро» ВК-27.07 (ТВГИ.466535.015-27.07) представляет собой многопроцессорный вычислительный комплекс, обеспечивающий многопользовательский, многозадачный режим вычислений в реальном времени. В настоящее время является последней моделью в ряду вычислительных комплексов, разрабатываемых АО «МЦСТ» по заказу АО «Концерн «Океанприбор».

Размещение составных частей ВК приведено на рисунке 2.



Рисунок 2 – Размещение составных частей ВК-27.07

1 – субблок

2 – панель передняя 6U 4HP (2 шт.)

3 – панель ПRS232 (2 шт.)

4 – ячейка МПК2/М

5 – ячейка МFE/М

6 – ячейка ММР/М (4 шт.)

7 – модуль МС76.03 (3 шт.)

8 – модуль МНМХ/С

9 – модуль МП1СА/С

10 – модуль МКEG/С

11 – модуль МHSA/С

12 – модуль электропитания БТРС-010 (2 шт.)

13 – ячейка МГАR/MX-K

14 – ячейка МПВ/М;

15 – модуль МНМ/С (3 шт.)

ВК оснащается центральным процессором «R2000» с тактовой частотой до 2000 МГц и имеет микросхему ЦП 1891ВМ01А8. Основные технические данные отображены в таблице 1.

Таблица 1 – Технические данные ВК

|  |  |
| --- | --- |
| Наименование параметра | Значение |
| Количество интегральных микросхем 1891ВМ01А8 | 1 шт. |
| Количество процессорных ядер в микросхеме 1891ВМ01А8 | 8 шт. |
| Рабочая тактовая частота процессорного ядра | 2 ГГц |
| Пиковая производительность | Не менее 64 Гфлопс |
| Объем оперативной памяти | 32 Гбайт |
| Объем внешней памяти | Не менее 400 Гб |
| Номинальное напряжение питания | 27 В |
| Установившееся отклонение напряжения | +2,7; –4,86 |
| Коэффициент пульсаций напряжения | Не более 5% |
| Потребляемая мощность | Не более 500 Вт |
| Пусковой ток | Не более 42 А |
| Суммарное сопротивление цепи заземления | Не более 0,2 Ом |
| Габаритные размеры | 483×299×265,8 мм |
| Система охлаждения | Внешняя, воздушного типа |
| Каналы ввода/вывода (количество) | Gb Ethernet (11),  Fast Ethernet (4),  RS-232 (9); USB (7), Audio (3), Display Port (2), SATA (2),  «Манчестер» (4), Link (12), ВОЛС (12) |

ПО вычислительного комплекса состоит из функционального программного обеспечения (ФПО), системного программного обеспечения (СПО) и общего программного обеспечения (ОПО).

ФПО и СПО разрабатываются в процессе создания изделия и являются его неотъемлемой частью. ОПО обеспечивает разработку и функционирование СПО и ФПО, является заимствуемым и входит в состав поставляемого вычислительного комплекса.

Достоинствами средств «ВК-27.07» является:

* Наличие в комплекте поставки предустановленного ОПО.
* Разработка и поставка аппаратуры и ОПО одной организацией, что положительно сказывается на устойчивости работы приборов.

В состав Общего программного обеспечения ВК входят:

1) Операционная система (ОС) «Эльбрус-Д»:

* Ядро операционной системы «Эльбрус-Д»
* Специальные модули и утилиты ОС «Эльбрус-Д»
* Средства поддержки пользовательского интерфейса
* Базовые средства
* Компоненты графической подсистемы
* Средства работы с сетью
* Средства отладки
* Средства работы с периферийными устройствами

2) Система программирования:

* Оптимизирующий компилятор языков С/С++

3) Система тестовых и диагностических программ.

4) Система тестирования ОС.

5) Комплекс сервисных и пользовательских программ.

6) Высокопроизводительные библиотеки.

7) Программа начального старта.

Операционная система для вычислительной техники на основе отечественных микропроцессоров архитектуры «Эльбрус-Д» является комплексом прикладных программных пакетов, работающих на базе ядра ОС Linux kernel-4.9, доработанного для функционирования на архитекторах «Эльбрус», SPARC V9 и с разработанными новыми основными средствами защиты информации от несанкционированного доступа «Эльбрус». Также, ОС обеспечивает многозадачный режим работы, осуществляющий сетевое взаимодействие с использованием стандартных сетевых протоколов, может работать со стандартными типами файловых систем и предоставляет возможности управления драйверами специальных устройств.

В состав ОС «Эльбрус-Д» входят:

* Доработанное ядро ОС Linux kernel-4.9;
* Специально разработанные программы;
* Программы из состава дистрибутива ОС Debian 8.

ОС предназначена для управления вычислительными процессами в многопроцессорных и многоядерных средствах вычислительной техники на основе микропроцессоров с архитектурами «Эльбрус» и SPARC V9;

ОС «Эльбрус-Д» применяют в следующих областях:

* управляющие вычислительные комплексы (ВК) и робототехнические комплексы;
* вычислительные комплексы для оснащения АСУ и информационные системы обработки и хранения информации;
* автоматизированные рабочие места;
* сервера и рабочие станции;
* маршрутизаторы;

Комплекс программ «Ядро ОС «Эльбрус-Д» и специальные модули» является частью дистрибутива ОПО ОС «Эльбрус-Д» и предназначен для работы в составе ОС «Эльбрус-Д».

Назначение ядра ОС — обеспечить прикладным программам ФПО и всем прикладным программам доступ к ресурсам ВК, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации.

Ядро ОС представляет собой наиболее низкий уровень абстракции для доступа приложений к ресурсам системы, необходимым для их работы. Ядро предоставляет такой доступ исполняемым процессам соответствующих приложений за счёт использования механизмов межпроцессного взаимодействия и обращения приложений к системным вызовам ОС.

Специальные модули и утилиты ядра ОС «Эльбрус-Д» предназначены для реализации комплекса средств защиты информации от несанкционированного доступа.

Ядро ОС и специальные модули обеспечивает следующие свойства:

* Соответствие интерфейсу ядра ОС Linux 4.9;
* Функционирование на архитектурах «Эльбрус» и SPARC V9;
* Возможность работы в многоядерных, многопроцессорных и многомашинных вычислительных системах;
* Выполнение функций загрузчика программ СПО;
* Возможность контроля целостности программного обеспечения;
* Возможность управления параллельными потоками вычислений для организации параллельной деятельности на ядрах микропроцессора;
* Возможность приоритетного планирования вычислительных процессов из СПО;
* Привязка потоков вычислений к определенному процессору (ядру);
* Возможность работы с драйверами устройств;
* Возможность трассировки основных событий ядра (переключений процессов).

## 1.2. Многопоточность

ВК 27.07 и ядро ОС «Эльбрус-Д» позволяют разрабатывать многопоточное ПО, а для корректной работы комплекса необходима передача данных на разных частотах, поэтому необходимо разбить процесс на потоки (нити).

Процесс – это набор ресурсов и данных, использующихся при выполнении программы.

Поток или нить – это объект, выполняющий последовательность инструкций внутри процесса. Нитей может быть несколько, и все они имеют доступ к ресурсам процесса, таким как адресное пространство, открытые файлы, полномочия и т. д. Существуют различные их типы:

• нити ядра;

• легковесные процессы;

• прикладные нити.

*Нити ядра* не требуют связи с каким-либо прикладным процессом. Они создаются и уничтожаются ядром по мере необходимости и отвечают за выполнение определенных функций. Такие нити используют совместно доступные области кода и глобальные данные ядра, но обладают собственным стеком в ядре.

Нити ядра являются малозатратными при создании и дальнейшем использовании. Единственными используемыми ими ресурсами являются стек ядра и область, в которой сохраняется контекст регистров на период приостановки работы нити.

В ОС «Эльбрус-Д» нить ядра – это основной легковесный объект, который может независимо планироваться и отправляться на выполнение одному из процессоров системы. Такой объект не нуждается в ассоциации с каким-либо процессом, он может быть создан, запущен и уничтожен ядром при помощи специальных функций. Нить ядра требует минимального количества ресурсов в виде небольшой структуры данных и стека. Структура данных нити ядра содержит следующую информацию:

* сохраненная копия регистров ядра
* приоритет и информация
* указатели на местонахождение нити в очереди планировщика или, если выполнение нити блокировано, в очереди ожидания ресурсов
* указатель на стек
* указатель на очередь всех нитей процесса и очередь всех нитей в системе

*Легковесный процесс (ЛП)* – это прикладная нить, поддерживаемая ядром. Каждый процесс может иметь один или более легковесный процесс, любой из которых поддерживается отдельной нитью ядра. ЛП разделяют адресное пространство и другие ресурсы процесса. Каждый легковесный процесс потребляет значительные ресурсы ядра, включая физическую память, отводимую под стек ядра, из-за этого система не может поддерживать большое количество таких процессов. Применение ЛП не подходит для приложений, использующих большое количество нитей или часто создающих и уничтожающих их.

В ОС «Эльбрус-Д» появляется новая структура – lwp, которая содержит следующую информацию:

* указатель на нить ядра;
* указатель на структуру *proc*;
* значение времени работы в режиме задачи и использование процессора;
* информацию об обработке сигналов;
* данные об использовании ресурсов и данные профиля процесса.

Для легковесных процессов доступны основные средства синхронизации. Легковесные процессы не имеют глобальных имен и вследствие этого невидимы для других процессов.

Структура *proc* содержит следующие информационные поля:

* уникальным идентификатор процесса;
* текущее состояние процесса;
* предыдущий и следующий указатели, связывающие процесс с очередью планировщика;
* приоритеты планирования задач;
* информация об обработке сигналов.

Также существует поддержка нитей полностью на прикладном уровне, при этом ядру об их существовании ничего известно не будет. Для этих целей создаются специальные библиотечные пакеты, которые содержат все необходимые средства для создания, синхронизации, планирования и обработки нитей без какой-либо специальной помощи ядра. Реализация прикладных нитей возможна по причине того, что прикладной контекст нити может сохраняться и восстанавливаться без вмешательства ядра. Каждая прикладная нить обладает собственным стеком в адресном пространстве процесса, областью для хранения контекста регистров и другой важной информации, такой как маски сигналов.

Библиотека планирует выполнение и переключает контекст между прикладными нитями, сохраняя стек и состояние регистров текущей нити и загружая стек и состояние регистров следующей по расписанию нити.

Ядро системы планирует выполнение процесса, содержащего в себе прикладные нити, который в свою очередь, использует библиотечные функции для планирования выполнения своих нитей. Если процесс или ЛП вытеснен кем-то, то вытесняются и все его нити.

В ОС «Эльбрус-Д» каждая прикладная нить должна поддерживать информацию следующего содержания:

* идентификатор нити;
* сохраненное состояние регистров;
* стек в приложении;
* маска сигналов;
* приоритет;
* локальная область хранения нити.

Для создания прикладных нитей будет использован интерфейс потоков выполнения POSIX.1c. Функции POSIX.1c приведены в таблице 2.

Таблица 2 – Функции POSIX.1c.

|  |  |
| --- | --- |
| **Название функции** | **API потоков выполнения POSIX.1c** |
| Создание потока | pthread\_create |
| Получение идентификатора потока | pthread\_self |
| Завершение потока | pthread\_exit |
| Посылка сигнала | pthread\_kill |
| Ожидание завершения потока | pthread\_join |

Приоритеты прикладным нитям назначаются при помощи функции *pthread\_setprio()*. Несвязанные нити с более низким приоритетом получают доступ к ЛП, только когда недоступны несвязанные нити с более высоким приоритетом. Связанные нити имеют свои собственные ЛП, поэтому не нуждаются в установке приоритета уровня нитевой библиотеки.

Приоритеты, устанавливаемые прикладным нитям на уровне нитевой библиотеки, не гарантируют того, что нить с наивысшим приоритетом будет выполняться ранее всех остальных. Это происходит, когда большое количество несвязанных нитей пытаются получить доступ к ЛП.

Для того чтобы этого избежать, необходимо при создании нити связывать ее с ЛП. За управление приоритетами легковесных процессов отвечает ядро системы. Прикладная нить, связанная с легковесным процессом, имеющим наибольший приоритет, гарантировано получит доступ к ЦП перед всеми остальными нитями в системе.

С другой стороны, когда библиотека создает по одному ЛП для каждой нити, необходимо большое количество ресурсов ядра и участия ядра во всех операциях синхронизации и выполнения. Это ведет к снижению производительности системы.

Для создания нитей существуют следующие функции:

1. int thr\_create(void \*stack\_base, size\_t stack\_size, void \*(\*start\_func) (void\*), void \*arg, long flags, thread\_t \*new\_thread\_ID)

Аргументы функции:

* + stack\_base– начало выделяемой под стек области памяти
  + stack\_size – размер этой области
  + start\_func – функция-обработчик thread
  + arg – аргумент этой функции
  + flags – флаги
  + new\_thread\_ID – идентификатор новой нити.

1. int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routne) (void\*),void \*arg)

Аргументы функции:

* + thread – идентификатор новой нити
  + attr – атрибуты нового pthread
  + start\_routne – функция-обработчик
  + arg – аргумент функции-обработчика

Создать легковесный процесс можно с помощью функции \_l p\_create.

Для ОС «Эльбрус-Д» время, затрачиваемое на порождение нитей, передачу управления между нитями, является минимальным в случае применения технологии легковесных процессов.

## 1.3. Система обмена

На предприятии АО «Концерн «Океанприбор» был разработан программный продукт, обеспечивающий более удобную работу с обменом данными между процессами и нитями ПО, как внутри приборов, так и между приборами ЦВК.

Система обмена (СО) – это набор библиотечных функций, которые подключаются к образу на этапе компоновки и выполняются в контексте задачи пользователя. «Система обмена» обеспечивает:

* Передачу информации через подготовленные массивы от одной задачи к другой.
* Передачу данных от одной задачи к другим задачам
* Ожидание прихода данных от одного или нескольких абонентов в определенное время
* Проверка обновления данных

Система обмена имеет следующие функции:

**ioOpenInit** – начало раздела инициализации системы обмена.

**ioCloseInit** – конец раздела инициализации системы обмена.

**ioOpen** – открытие массива в системе обмена.

**ioClose** – закрытие массива в системе обмена.

**ioWrite** – передача массива данных.

**ioRead** – прием массива данных.

**ioScan** – копирование массива данных из общего буфера без ожидания, с возвратом признака обновления информации.

**ioLotAdd** – добавление массива в множество ожидаемых массивов.

**ioLotAdds** – добавление нескольких массивов в множество.

**ioLotAddFunc** – добавление массива в множество с определением функции обработки.

**ioLotRead** – ожидание прихода информации по нескольким массивам.

**ioLotRem** – обнуление множества ожидаемых массивов.

**ioCancel** – снятие текущего запроса на чтение.

**ioExit** – завершение работы системы обмена, с автоматическим закрытием всех дескрипторов.

**ioGetPrm** – чтение параметра без ожидания, с копированием значения параметра в буфер, определенный пользователем.

**ioSetPrm** – установка значения параметра без передачи массива параметров.

# Глава 2. Интерфейсы каналов связи

Так как нити процесса и процессы производят обмен данными между собой при помощи СО, а прием и передача данных в другие системы НК происходит через интерфейсы каналов связи Манчестер, Ethernet, RS-232, рассмотрим эти интерфейсы.

## 2.1. Интерфейс Манчестер (Manchester)/ГОСТ Р 52070-2003

*Манчестер (Manchester)* **–** используемый для связи между различными электронными модулями в системе. Был разработан в 1949 году для использования в первых компьютерах. Структурная схема технических средств интерфейса в ее основной (базовой) конфигурации приведена на рисунке 3.

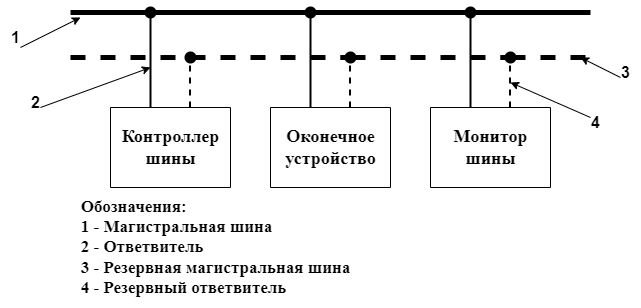


Рисунок 3 – Структурная схема технических средств интерфейса

Интерфейс должен функционировать, асинхронно, в режиме «команда-ответ». Передача информации должна осуществляться полудуплексным способом. Инициирование обмена информацией и управление передачей должен осуществлять только один КШ. Поток информации, передаваемый по информационной магистрали, должен состоять из сообщений, содержащих КС, ОС, СД, формат передаваемых слов изображен на рисунке 4.

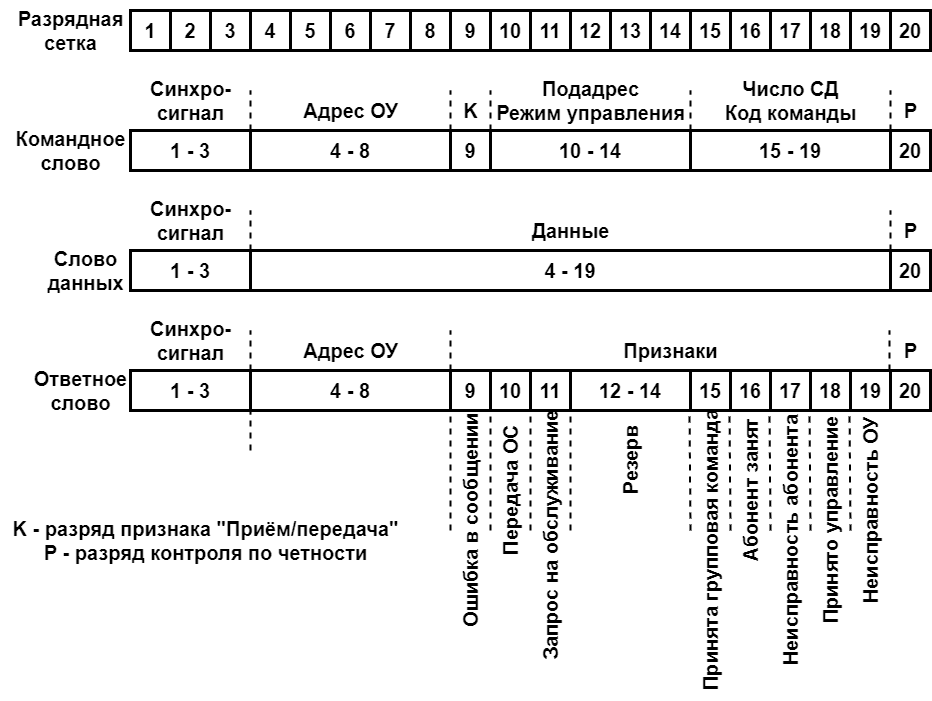


Рисунок 4 – Структурная схема технических средств интерфейса

Одной из особенностей является использование контроля синхронизации. Благодаря четкому разделению временных интервалов, обеспечивает синхронизацию между передатчиком и приемником. Это позволяет избежать ошибок при передаче и обеспечить высокую скорость передачи информации. Также, его можно использовать для передачи данных на большие расстояния без потерь качества сигнала. Еще одной особенностью является возможность передавать в обе стороны, но только в одно и то же время.

Однако интерфейс имеет некоторые недостатки. Для передачи потребуется двойной пропускной способности канала связи. Также, может возникнуть задержка в получении информации, при передаче информации, так как приемник должен декодировать каждый бит сигнала.

Данные кодируются с использованием изменения в уровне напряжения сигнала. Каждый бит данных представлен изменением уровня напряжения за определенный период времени. Единица передается как биполярный кодированный сигнал 1/0 (за положительным импульсом следует отрицательный импульс). Нуль передается как биполярный кодированный сигнал 0/1 (за отрицательным импульсом следует положительный импульс). Переход через нулевой уровень осуществляется в середине интервала времени, в течение которого передается информационный разряд, изображено на рисунке 5.

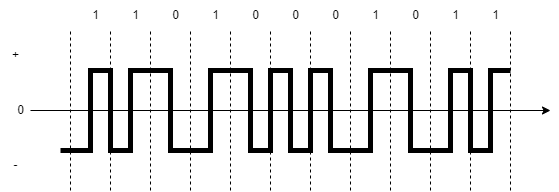


Рисунок 5 – Кодирование Манчестера

Ячейка Манчестера предназначена для обмена информацией между вычислительным комплексом и различными устройствами, подсоединяемыми к вычислительному комплексу через канал обмена, по стандарту «Интерфейс магистральный последовательный системы электронных модулей ГОСТ Р 52070–2003».

Ячейка Манчестера представляет собой модуль в конструктиве РМС, устанавливаемый в вычислительный комплекс с помощью соединителя мезонина РМС, имеющий 2 канала ввода-вывода (основной и резервный), с темпом обмена 1 Мбит/с и возможным удалением 100 м. Связь с вычислительным комплексом осуществляется по шине PCI.

Для написания тестовых программ были использованны следующие стандартные библиотеки языка Си: *stdio.h, stdlib.h.* При работе с устройством используются стандартные функции языка Си: open(), close(), read(), write(), poll(), ioctl(), mmap() [6]. Перед передачей данных, в каждой из них был проинициализирован модуль Манчестер.

Были разработаны две программы *BC.c* (bus controller, Контроллер шины) и *RT.c* (remote terminal, Оконечное устройство). Устройство КК отправляет данные в ОУ, в случае успеха BC получает ответное слово (ОС), рисунок 6.

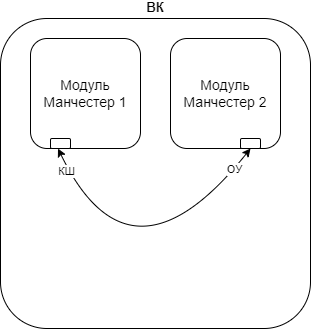


Рисунок 6 – Схема обмена данными между двумя устройствами соедененными через Манчестер, в одном ВК

Обмен производился с использованием следующих форматов:

* Формат 1 – передача данных от КШ к ОУ;
* Формат 2 – передача данных от ОУ к КШ.

При обмене данными через Формат 1 КШ отправляет Сообщение в ОУ. При успешном выполнении ОУ возвращает ОС. При обмене данными через Формат 2 КШ также отправляет Сообщение в ОУ, но теперь при успешном выполнении ОУ отправляет свое Сообщение. Форматы всех Сообщений изображены на рисунке 6.



Рисунок 6 – Форматы сообщений

*Программа BC.c* представляет собой *Контроллер шины* и функционирует следующим образом. В системной памяти драйвер выделяет адресное пространство для блока буферов обмена размером 4Кб. Блок буферов состоит из двух частей: 32-х буферов по приему и 32-х буферов по передаче. Каждый такой буфер имеет размер 64 байта (32 слова по 16 бит), что соответствует максимальному объему данных, передаваемых в одном сообщении. Начальный адрес блока буферов выровнен по границе 4Кб. Таким образом, младшие 5 разрядов адреса определяют номер буфера обмена. Шестой разряд определяет направление обмена: (0 - выдача в магистраль, 1 - прием из магистрали).

Для начала работы с устройством необходимо открыть файл драйвера в режиме «Чтение-запись»:

int f\_BC;

f\_BC = open ("Устройство Манчестера", O\_RDWR);

Для дальнейшей работы с буферами необходимо отразить их в адресное пространство процесса:

unsigned short \*bc\_pool;

bc\_pool = (uint16\_t \*) mmap (отображение адресного пространства для f\_BC);

Для выдачи данных по формату пакета 1(«прием данных от КШ в ОУ») необходимо заполнить данными нужный буфер. Первый(0) и последний(31) буферы не используются. Адрес буфера был получен при помощи специального макроса:

uint16\_t \*s\_buf;

saddr = 1; //номер буфера

s\_buf[0] = данные;

s\_buf = Макрос для получение адреса буфера;

Для отправки данных нужно инициализировать структуру, содержащую *командное слово* для передачи, при помощи специального макроса. Вызвать функцию *ioctl* для отправки сообщения.

mmr\_cmd\_t cmd;

{Инициализация структуры}

cmd.cw = Создание командного слова;

int ret = ioctl(отправка сообщения);

Если переменная *ret* ≥ *0* – выдача данных прошла успешно, в таблице 3 отображены коды завершения транзакции.

Таблица 3 – Коды завершения транзакций

|  |  |
| --- | --- |
| Код завершения | Описание |
| \_OK | Успешно, получено ОС |
| \_ECRC | Ошибка четности |
| \_ETO | Нет ответа от ОУ |
| \_ESPEC | Успешно, спец. признаки в ОС |
| \_ECMD | Некорректная команда |
| \_ERP | Пауза при приеме данных |
| \_NONE | Успешно, без ОС |

Перед завершением необходимо снять отображение буферов и закрыть файл драйвера.

munmap(снятие отображения для bc\_pool);

close(f\_BC);

*Программа RT.c* представляет собой *Оконечное устройство*. При работе в режиме ОУ данные для передачи и принятые данные хранятся во внутренней памяти ПЛИС модуля. Для приема данных используются 32 буфера размером 64 байта каждый. Для отправки данных используются 32 буфера размером 64 байта каждый (32 слова). Для начала работы с устройством необходимо открыть файл драйвера в режиме «Чтение-запись»:

int f\_RT;

f\_RT = open("Устройство Манчестера", O\_RDWR);

Для работы с ОУ необходимо инициализировать структуру устройства и вызвать функцию *ioctl* со специальным кодом:

mmr\_rt\_init\_t rt\_i;

{Инициализация структуры}

int ret = ioctl(инициализация ОУ);

Таблица 4 – Флаги ОУ

|  |  |
| --- | --- |
| Флаг | Значение |
| \_TFV | валидны поля ta, flags, vw. |
| \_IMODE | валидно поле imode |
| \_OMODE | валидно поле omode |

Для чтения принятой от КШ команды нужно вызвать функцию ioctl со специальным кодом:

ret = ioctl(чтение принятой команды от КШ);

Для чтения принятых данных от ОУ и записи данных для отправки на запрос от ОУ — используются стандартные функции *lseek*, *read*, *write*.

uint16\_t buf[32];

ret = lseek(прием данных от ОУ);

ret = read(f\_RT, &buf, (32 \* 2));

lseek(заполнение буфера для отправки на запрос от ОУ);

write(f\_RT, &buf, (32 \* 2));

Перед завершением необходимо закрыть файл драйвера.

close(f\_RT);

## 2.2. Интерфейс Ethernet, протоколы UDP, TCP

*Ethernet* – это технология локальной сети, которая использует проводное соединение для передачи данных между устройствами. Интерфейс был разработан компанией Xerox в середине 1970-x годов, затем улучшен компаниями Intel и DEC. Он использует метод передачи с помощью коллизионной сети, данные передаются с помощью экранированного кабеля, который использует протокол *CSMA/CD*. Передаваемые данные проверяются на наличие коллизий. Для передачи данных используется программный интерфейс *Сокет.*

*Сокет (socket)* – был впервые реализован в операционной системе Berkley UNIX. Сейчас этот программный интерфейс доступен практически в любой модификации Unix. Изначально сокеты использовались в программах на языках программирования C и C++, но в настоящее время средства для работы с ними предоставляют многие ЯП. Сокеты предоставляют мощный и гибкий механизм межпроцессного взаимодействия (IPC). Они могут использоваться для организации взаимодействия программ на одном компьютере, по локальной сети или через Internet, что позволяет вам создавать распределённые приложения различной сложности. Кроме того, с их помощью можно организовать взаимодействие с программами, работающими под управлением других операционных систем [5].

*TCP сокет* и *UPD сокет* для передачи данных используют протоколы транспортного уровня *TCP* и *UDP* соответственно [3]. Данные представляют собой пакет. Пакет это структура, описанная в специальном протоколе связи.

*UDP* – этот протокол, который обеспечивает доставку данных без подтверждения получения, не устанавливает соединение. Он не обеспечивает контроль целостности и точности всех переданных данных, не следит за доставкой пакетов. Используется, где скорость важнее надежности.

Рассмотрим две тестовые программы, работающие на протоколах *TCP* и *UDP*, которые одновременно передают друг другу пакеты данных.

*Программа X1*, реализованная как Клиент, передает и получает данные по локальной сети *программе X2*, реализованной как Сервер. Для связи программы используют протокол UDP, рисунок 7.

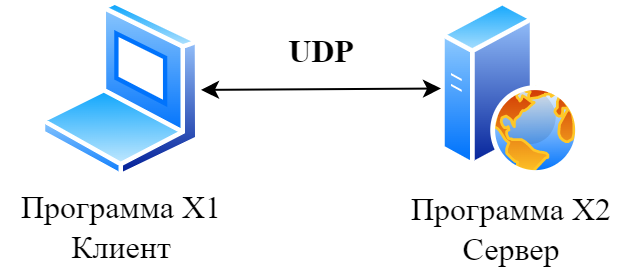


Рисунок 7 – Взаимодействие ПО на основе Клиент-Серверной реализации с использованием протокола *UDP*

Были использованы стандартные библиотеки языка Си: *stdio.h*, *stdlib.h*, *sys/types.h*, *sys/socket.h*. Прежде, чем передавать данные через сокет, его необходимо связать с адресом в выбранном домене. При помощи стандартной библиотеки Си (*socket.h*) были инициализированы *сокеты* по заданным портам.

int socket;

socket = socket(AF\_INET, SOCK\_DGRAM, 0);

Функция принимает аргументы указанные в таблице Т.

Таблица – аргументы функции *socket ()*

|  |  |  |
| --- | --- | --- |
|  | Название | Значение |
| Первый аргумент | AF\_UNIX | Межпроцессорное взаимодействие |
|  | AF\_INET | Взаимодействие по сети |
| Второй аргумент | SOCK\_STREAM | С установкой соединения |
|  | SOCK\_DGRAM | Без установки соединения |

Для связывания *сокета* с локальным адресом протокола используется функция *bind(),* только в программе-сервере*.* В качестве первого параметра передаётся дескриптор сокета, который мы хотим привязать к заданному адресу. Второй параметр содержит указатель на структуру с адресом, а третий – длину этой структуры.

int bind(int socket, struct sockaddr \*addr, int addrlen);

Структура имеет следующий вид:

struct sockaddr {

short int sin\_family; // Семейство адресов

unsigned short int sin\_port; // Номер порта

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

};

Поле *sin\_family* содержит идентификатор домена, тот же, что и первый параметр функции *socket*. В *sin\_port* записывается номер порта, а в *sin\_addr* – IP-адрес хоста. Поле *sin\_addr* само является структурой, которая имеет вид:

struct in\_addr {

unsigned long s\_addr;

};

Структура была заполнена данными о *сокете*:

sockaddr addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(2221);

addr.sin\_addr.s\_addr = htonl(INADDR\_LOOPBACK);

Для передачи информации используется функция:

int sendto(int sockfd, const void \*msg, int len, unsigned int flags, const struct sockaddr \*to, int tolen);

int sockfd – *сокет,* для отправки данных

const void \*msg – указатель на буфер данных

int len – длина сообщения

unsigned int flags – параметры отправки

const struct sockaddr \*to – указатель на структуру получателя

int tolen – длина структуры получателя

Для приема информации используется функция:

int recvfrom(int sockfd, void \*buf, int len, unsigned int flags, struct sockaddr \*from, int \*fromlen);

int sockfd – *сокет,* для чтения данных

void \*buf – указатель на буфер для хранения данных

int len – длина буфера

unsigned int flags – параметры получения

struct sockaddr \*from – адрес для структуры отправителя

int \*fromlen – длина структуры отправителя

После завершения работы необходимо закрыть сокет.

close(socket);

Схема работы клиента и сервера *UDP* изображена на рисунке У.

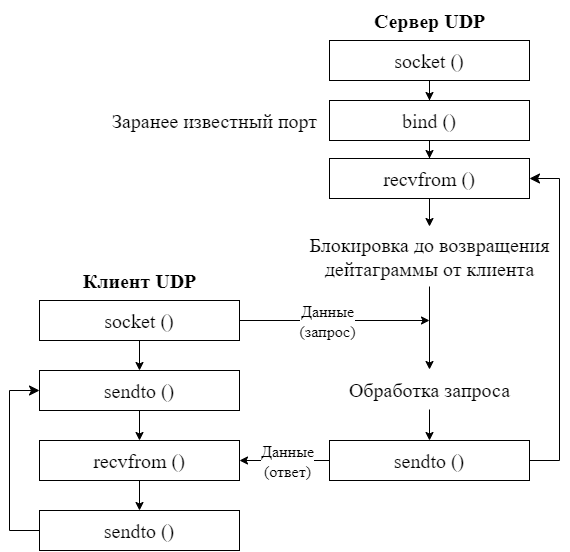


Рисунок У – Схема сценария взаимодействия клиент-сервера *UDP*

*Программа Х3* будет передавать и принимать данные от *программы Х4.* Программы реализованы с использование протокола связи *TCP.* Схема взаимодействия программ отображена на рисунке Е.

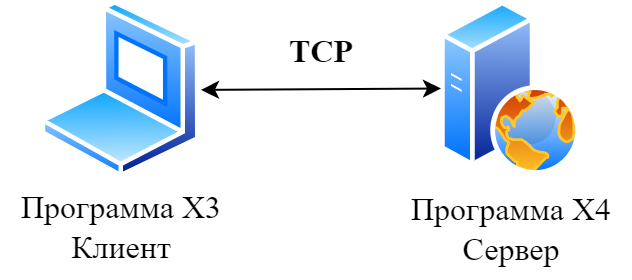


Рисунок Е – Взаимодействие ПО на основе Клиент-Серверной на основе протокола *UDP*

*TCP* – этот протокол устанавливает соединение перед передачей и обеспечивает надежную доставку данных с контролем целостности и контролем передаваемых данных. Гарантирует порядок доставки пакетов и автоматически повторно передает их до тех пор, пока они не будут доставлены. Протокол гарантирует доставку данных в том же порядке, в котором были отправлены. Использует трехэтапный процесс установления соединения между клиентом и сервером, что обеспечивает безопасность передачи данных.

int socket;

socket = socket(AF\_INET, SOCK\_DGRAM, 0);

Для связывания сокета с локальным адресом протокола необходимо также использовать функцию *bind().*

Для передачи данных по протоколу TCP необходимо установить соединение с сервером:

int connect(int sockfd, struct sockaddr \*serv\_addr, int addrlen);

int sockfd – дескриптор *сокета*

struct sockaddr \*serv\_addr – адрес

int addrlen – длина

Сервер должен ожидать сообщение от клиента:

int listen(int sockfd, int backlog);

Далее подтверждать получение:

int accept(int sockfd, void \*addr, int \*addrlen);

Для отправки сообщения используется функция *sendto:*

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

Для получения данных служат функция *recv:*

int recv(int sockfd, void \*buf, int len, int flags);

В отличие от функции *sendto()* и *recvto(),* функции передачи и приема в протоколе *TCP* не имееют 5 и 6 аргумент функции, так как было установлено соединение.

После завершения работы необходимо также необходимо закрыть сокет.

close(socket);

Схема сценария взаимодействия клиент-сервера *TCP* показа на рисунке О.

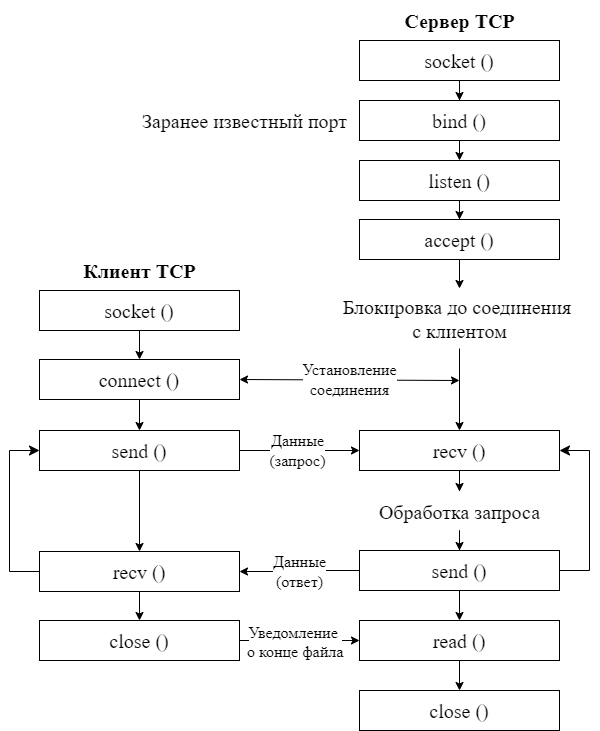


Рисунок О – Сценарий взаимодействия клиент-сервера *TCP*

## 2.3. Интерфейс RS-232

*RS-232(Recommended Standard 232)* – это стандарт передачи данных, разработанный в 1960-х годах, используемый для связи между устройствами. Стандарт использует два провода для передачи данных: *TX* – для отправки данных, *RX* – для приема данных. *RS-232* также определяет линии управления передачей данных, такие как *RTS (Ready to Send*) и *CTS (Clear to Send)*, *DTR (Data Terminal Ready)* и *DSR (Data Set Ready)*.

Стандарт определяет физические, электрические и протокольные характеристики передачи данных через последовательный порт. Характеристики *RS-232* включают скорость передачи, набор битов контроля ошибок, контроль четности и другие параметры, которые позволяют обеспечить надежную передачу данных между устройствами. Каждый бит данных отправляется последовательно, от старшего к младшему. Он может работать на расстоянии до 15 метров, для более дальних расстояний необходимы дополнительные устройства. Также существует необходимость использования проводников для предотвращения помех.

Данные в *RS-232* передаются в последовательном коде побайтно. Для синхронизации битам данных предшествует специальный стартовый бит, после битов данных следует бит паритета и один или два стоповых бита. Такая группа битов совместно со стартовым и стоповым битом, а также битом паритета носит название старт–стопного символа. Таким образом, полный асинхронно передаваемый символ данных состоит из 10–11 бит при том, что собственно пользовательские данные состоят из 7–8 бит. На рисунке 8 приведен формат данных по *RS-232*.

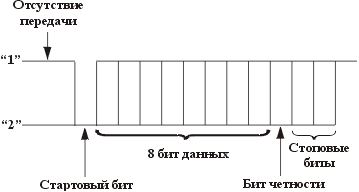


Рисунок 8 – Формат данных *RS-232*

Драйвер состоит из стандартных библиотечных процедур:

**open** – открытие устройства;

**tcgetattr** – получение настроек устройства *RS-232*;

**tcsetattr** – изменение настроек устройства *RS-232*;

**read** – чтение по каналу обмена данных от приемника;

**write** – запись данных;

**close** – закрытие канала.

При включенном использовании *CTS/RTS* для устройства *RS-232* производит выдачу данных в порт только при наличии сигнала *CTS*. Данный сигнал устанавливается сигналом *RTS* другого конца соединения. Изменение сигнала *CTS* вызывает прерывание и драйвер порта автоматически продолжит вывод данных, когда *CTS* примет единичное значение.

# Глава 3. Разработка программного обеспечения

Учитывая специфику будущего программного обеспечения была выбрана водопадная модель разработки.

Водопадная модель разработки ПО – это последовательный процесс разработки, который разделяет процесс разработки на последовательные фразы, где каждая следующая начинается только после завершения предыдущей. Основные этапы такой модели [14]:

* **Анализ требований:** на этом этапе определяется требования к системе со стороны заказчика и пользователя.
* **Проектирование:** на основе выбранных требований разрабатывается архитектура системы. Проектирование включает в себя определение структур данных, модулей программы и архитектурных особенностей.
* **Реализация:** на этом этапе происходит написание кода программы на выбранном ЯП с учетом всех деталей, заложенных на этапе проектирования.
* **Тестирование:** после написания кода, следует этап проверки для выявления и исправления ошибок.
* **Внедрение:** готовое ПО устанавливается у пользователей.

Схема модели изображена на рисунке П.

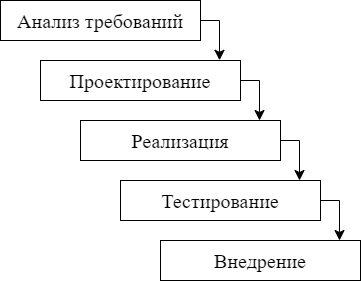


Рисунок П – Водопадная модель разработки ПО

## 3.1. Анализ требований к программному обеспечению и выбор структуры

Программное обеспечение для обмена с внешними системами по требованиям протоколов связи должно обеспечивать взаимодействие задач ЦВК с четырьмя внешними система системами:

* Метеорологическая система использует основной канал интерфейса связи RS-232 на частоте 1 Гц. Производится только передача данных.
* «БИУС» использует основной и резервный каналы интерфейса связи Ethernet на частоте 1 Гц. Передает и принимает данные.
* Навигационная система использует основной канал интерфейса связи Манчестер на частоте 0.25 Гц. Производится только передача данных.
* Система единого времени использует основной и резервный каналы интерфейса связи Манчестер на частоте 1 Гц. Производится только передача данных.

Основные требования по протоколу обмена между системами указаны в таблице 5.

Таблица 5 – Требования к обмену между ЦВК и внешними системами.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название системы | Интерфейс канала обмена | Прием данных | Передача данных | Частота приема | Частота обмена |
| Метеорологическая система | RS-232 | Да | Нет | 1 Гц | 1 Гц |
| БИУС | Ethernet 1  Ethernet 2 | Да | Да | 1 Гц | 1 Гц |
| Навигационная система | Манчестер | Да | Нет | 0.25 Гц | 0.25 Гц |
| Система единого времени | Манчестер 1  Манчестер 2 | Да | Нет | 1 Гц | 1 Гц |

Реализация ПО СВС предполагает несколько подходов:

* однопоточным процессом
* многопоточным процессом
* несколькими процессами

Необходимо оценить достоинства и недостатки этих способов, для того чтобы выбрать наиболее подходящую структуру.

В рамках однопоточного процесса задачи обмена с подсистемами выполняются последовательно без приоритетности, так как они реализуются в одном потоке и имеют общий приоритет. Обмен с внешними системами может использовать ресурсы только одного процессора. Недостаток такой структуры заключается в том, что данная структура не использует в полной мере возможности многопроцессорной системы, а задачи зависят друг от друга. В тоже время один процесс с множеством потоком имеет преимущества, такие как высокая эффективность и меньшее потребление системных ресурсов благодаря общему доступу к данным. Однако, многопоточность требует точную синхронизацию потоков для предотвращения ошибок и потери данных в каждой программе.

Потоки обмена, взаимодействующие с внешними системами, должны быть организованы таким образом, чтобы обеспечить работу в режиме реального времени и должны иметь собственные таймеры для обеспечения частоты обмена, соответствующие протоколу. Для этого необходимо каждый поток процесса связать со своей прикладной нитью, для оптимального управления.

Исходя из анализа, структура ПО СВС, основанная на одном процессе с множеством связных потоков, реализующих задачи обмена с внешними системами, не является наилучшим решением из-за необходимости активного участия ядра в синхронизации и планирования. Учитывая функциональную независимость задач обмена данными с внешними системами, предпочтительным решением будет создание отдельного процесса для каждой задачи, имеющего несколько потоков. Такая структура ПО наилучшим образом подходит для решения данной проблемы. Такой подход не только улучшит производительность, но и облегчит отладку программы, позволяя тестировать модули по отдельности без необходимости перекомпилирования всего приложения в случае изменений в многопоточной структуре.

Так как по протоколу системы должны обмениваться данными по разным интерфейсам связи, для их взаимодействия будут использоваться отдельные задачи, отображено в таблице Т.

Таблица Т – Названия систем и задач

|  |  |
| --- | --- |
| Название системы | Название задачи |
| Метеорологическая система | svs\_mk |
| Навигационная система | svs\_nk |
| Система единого времени | svs\_sev |
| БИУС | svs\_bius |
| Рассылка общекомплексных данных | common\_svs |

Схема взаимодействия систем ПО СВС по протоколам связи изображена на рисунке 9.

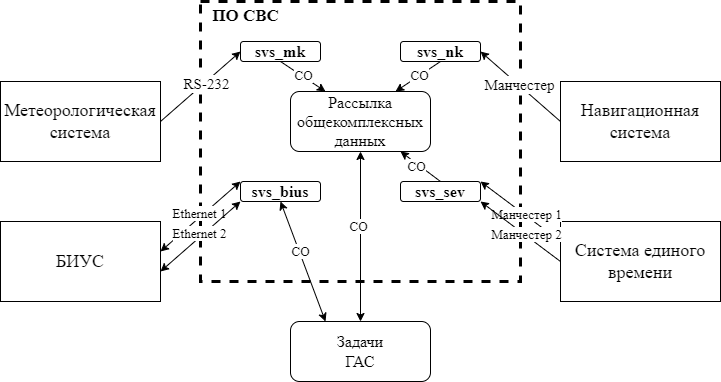


Рисунок 9 – Схема взаимодействия систем по протоколам связи

ПО обмена должно обеспечивать контроль принимаемых данных и каналов связи с внешними системами. Данные контроля, фиксирующие состояния каналов обмена с внешними системами, должны передаваться задачей обмена в систему контроля комплекса. Информация, которой обменивается ЦВК и внешние системы, отображается на табло системы отображения, регистрации, документирования и управления (СОРДиУ). Примерный вид отображаемой информации приведен на рисунке 10.

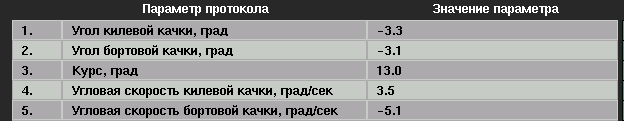


Рисунок 10 – Данные обмена с системой единого времени отображенные на табло СОРДиУ

Задачи ЦВК Задача обмена с навигационной системой и «БИУС» состоит из нескольких потоков. Задача обмена с системой «БИУС» принимает данные по основному и резервному одновременно. Для организации такой структуры понадобится четыре потока, работающих на одинаковой частоте – 1 Гц. Частота обмена с функциональными задачами обусловлена требованиями протокола связи, частота приема данных от внешней системы равна 1 Гц. Для этого используются потоки *t\_recv\_1* и *t\_recv\_2*. Потоком *t\_bius* формирует данные для функциональных задач ЦВК. Поток *t\_send* используется для отправки данных обратно в систему.

Процесс обмена с навигационной системой производится по основному каналу и состоит из двух нитей, одна из которых 1 раз в 4 секунды читает данные по каналу Манчестер с помощью потока *t\_recv.* Второй поток *t\_svs\_nk* ожидает запроса оператора, после чего передает данные функциональным задачам ЦВК.

Задача обмена системы единого времени получает данные по основному и резервному каналам, организуется процессом, состоящим из двух нитей , которые работают на частоте 1 Гц.

Задача обмена с метеорологической системой принимает данные по основному каналу. Состоит из одного процесса с одним потоком. Поток *t\_recv* принимает данные с частотой 1 Гц.

На рисунке 11 изображена схема ПО СВС с потоками.

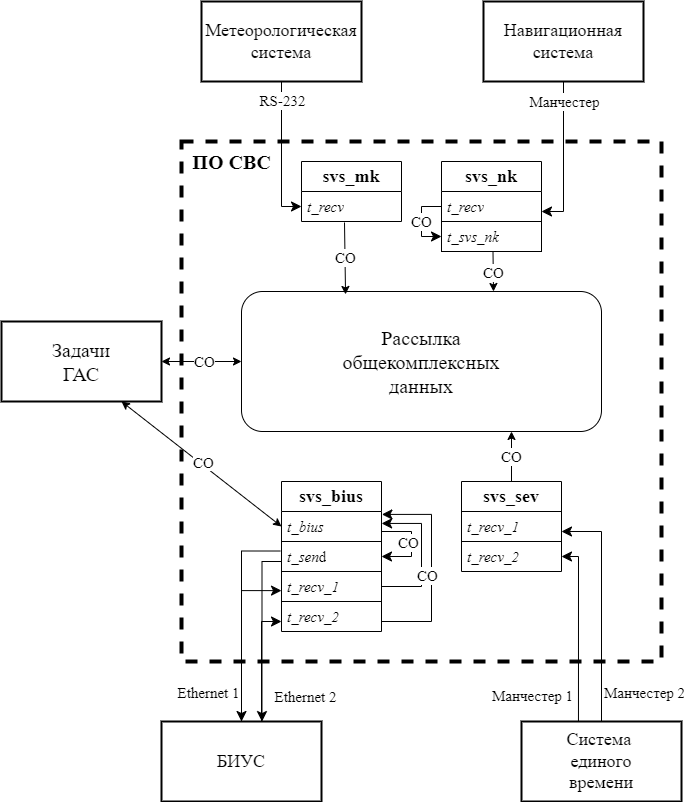


Рисунок 11 – Схема ПО СВС с потоками.

Для обеспечения заданных частот приема и передачи массивов данных можно лучше использовать интервальные таймеры стандарта POSIX.1b, по сравнению с задержкой системного *nanosleep*. Так как использование таймеров более точно.

Более целесообразно разрабатывать процессы, задействованные в обмене с внешними системам, как процессы реального времени, чтобы гарантировать передачу данных на определённых частотах. Управление приоритетами таких процессов осуществляется непосредственно через приложение. В случае наличия в системе хотя бы одного процесса реального времени, он получает преимущество в выполнении перед другими процессами, которые работают на основе разделения времени.

## 3.2. Разработка ПО

После организации структуры ПО получено распределение задач по процессам и потокам. На рисунках 12 и 13 изображены блок-схемы работы передающих и принимающих устройств.

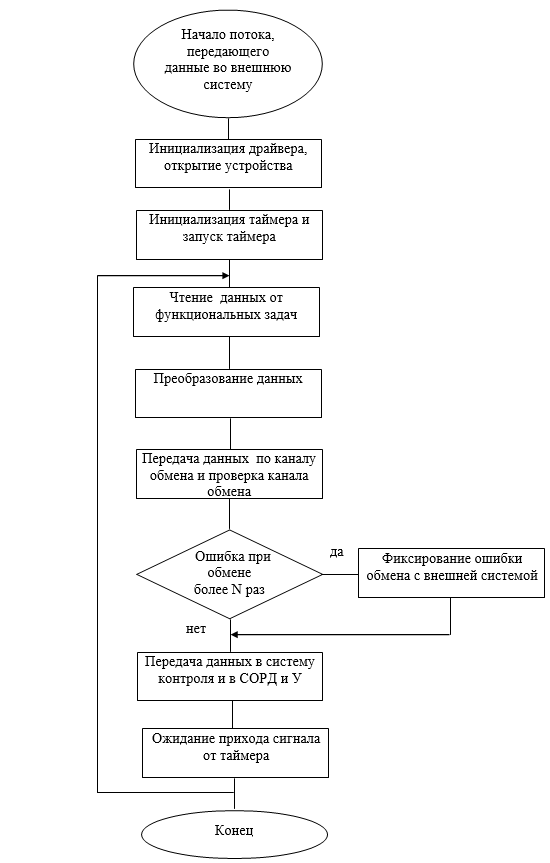


Рисунок 12 – Поток, передающий данные во внешние системы

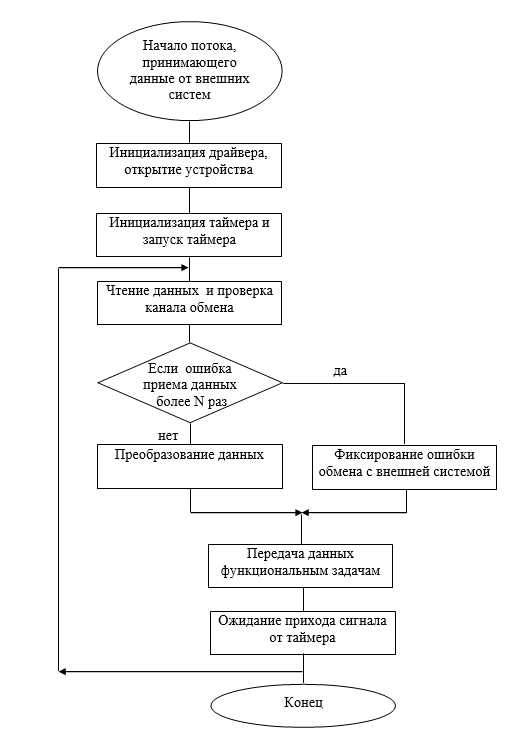


Рисунок 13 – Поток, принимающий данные из внешней системы

Работа программы с каналом связи Манчестер происходит с помощью правильно сконфигурированного драйвера. Инициализировать драйвер нужно отдельно из прикладной программы, затем создаются и инициализируются интерфейсные устройства. При инициализации какого-либо интерфейсного устройства (ИУ) все ИУ с меньшими номерами должны быть проинициализированы. После инициализации необходимо открыть файлы на созданных устройствах. Файлы должны открываться с флагом *O\_RDWR*. Все операции настройки контроллера соответствующего ИУ на требуемый режим работы (контролер канала, оконечное устройство, монитор) и операции приема или передачи данных (в соответствии с выбранным режимом) выполняются с помощью функции *ioctl*. В таблице 6 приведены возможные команды для работы разных устройств.

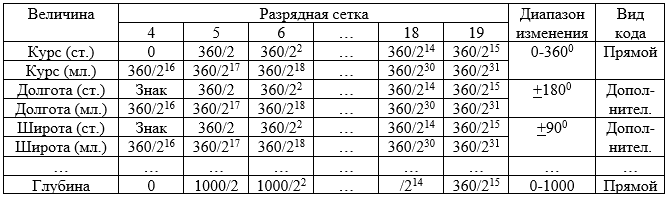
Таблица 6 – Возможные команды для разных форматов работы

|  |  |  |
| --- | --- | --- |
| Формат сообщения | Команда драйвера в контроллере | Команда драйвера в оконечном устройстве |
| КК → ОУ | FIOSEND | FIOGETBLOCK |
| ОУ → КК | FIORECEIVE | FIOSETBLOCK |
| КК → КУ | FIOSENDCOM | FIOWAITCOM |
| КК → КУ ← СД | FIOSENDCOM | FIOWAITCOM  FIODROPVECWORD |
| КК → КУ + СД | FIOSENDCOM | FIOGETBLOCK |

Для реализации функций целесообразно объединить их и создать библиотеку, в которой реализовываются функция инициализации контролера, функция инициализации оконечного устройства и функция закрытия устройства. Созданная библиотека подключается к программе, на этапе компоновки и выполняется в контексте задачи, использующей обмен по каналу Манчестер.

Данные от навигационной системы и системы единого времени, поступающие по каналу Манчестер, представляют собой шестнадцатиразрядные целые слова. Для того чтобы перевести их в числовой вид с плавающей точкой, необходимо создать процедуры преобразования, руководствуясь разрядной сеткой, заданной в технических требованиях для поступающих величин, отображено на таблице 7.

Таблица 7 – Формат данных от навигационной системы.



Каждое шестнадцатиразрядное слово, полученное навигационной системы, содержит несколько информационных величин, показано на таблице 8.

Таблица 8 – Формат данных, поступающих от системы СЕВ



В ОС для обмена по каналу *RS-232* прием и передача обеспечивается драйвером устройства. Данные, передаваемые метеорологической системой, по интерфейсу канала связи, представляют собой 11 битные пакеты. При включенном использовании *Clear To Send/Ready To Send (CTS/RTS)* устройство *RS-232* производит выдачу данных в порт, но только при наличии сигнала *CTS*.

Данный сигнал *CTS* устанавливается сигналом *RTS* другого конца соединения (при соответствующем кабеле). Изменение сигнала *CTS* вызывает прерывание и драйвер порта автоматически продолжит вывод данных, когда *CTS* примет единичное значение. С последовательными портами работают два драйвера: терминальный (не зависит от конкретного контроллера портов) и драйвер контроллера портов – на процессорном модуле.

Настройка и инициализация драйвера проходит с помощью диалоговой системы конфигурации. При вызове указываются номер порта, имя создаваемого устройства ОС, размеры входного и выходного буферов, скорость. В прикладной программе открывается файл устройства, проводятся необходимые настройки драйвера порта, например, использование сигналов *CTS/RTS* для управления потоком данных. При включенном использовании *CTS/RTS* драйвер порта производит выдачу данных в порт только при наличии сигнала *CTS*. Данный сигнал устанавливается сигналом *RTS* другого конца соединения (при соответствующем кабеле). Изменение сигнала *CTS* вызывает прерывание и драйвер порта автоматически продолжит вывод данных, когда *CTS* примет единичное значение.

Если включено использование *CTS/RTS*, драйвер терминала при приеме данных от драйвера порта проверяет уровень заполнения буфера ввода. Если число заполненных позиций в буфере ввода стало больше значения элемента массива *c\_cc[VHIWATER]* структуры *termios*, драйвер терминала устанавливает специальный признак, по которому драйвер порта снимает сигнал *RTS*, запрещая тем самым другому концу соединения передавать новые байты. Драйвер терминала снимет этот специальный признак, когда число занятых позиций в буфере ввода станет меньше значения элемента массива *c\_cc[VLOWATER]* структуры *termios* и после этого драйвер порта поднимет сигнал *RTS*.

Для работы с интерфейсом *RS-232* используются следующие функции:

***auxRawDevInit()*** – инициализация драйвера прямого ввода/вывода и создание устройства;

***auxRtsOn()*** – установка сигнала *RTS*;

***auxRtsOff()*** – снятие сигнала *RTS*;

***getCTSState()*** – функция возвращает, состояние *CTS*;

***tcgetattr()*** – получение параметров порта;

***tcsetattr()*** – установка параметров порта;

***write()*** – запись данных в порт;

***read()*** – чтение данных.

Команды функции ***ioctl()***:

**CTS\_RTS\_ON** - включить использование *RTS/CTS*;

**CTS\_RTS\_OFF** – выключить использование *RTS/CTS*;

**FIO\_FIONREAD** – опрос числа байт в буфере ввода/вывода;

**FIO\_RAW\_FLUSH** – очистка буферов ввода и вывода;

При открытии и инициализации канала *RS-232* функции драйвера удобнее использовать непосредственно в основной задаче. Реализация функций обмена по интерфейсу *RS-232* представляет собой отдельную библиотеку. Создание и запуск таймеров требует нескольких операций, которые повторяются в каждой задачи, использующей таймер, поэтому целесообразнее создать библиотеку с реализацией функций работы с таймерами. В основной функции каждого из процессов назначается приоритет процесса, формируются сигнальные маски для таймеров, создаются потоки процесса.

В отдельном процессе обмена с «БИУС» происходит инициализация по каналу Ethernet. На программном уровне, чтобы отправлять и принимать необходимо создать сокет и настроить его соответствующим образом. На транспортном уровне «БИУС» работает по стандарту *UDP*:

sockt = socket (AF\_INET, SOCK\_DGRAM, 0);

После создания *сокета* его необходимо заполнить структуру *сокета.* Установить семейство адресов для связи по сети, связать с известным портом, выбрать тип приема данных:

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

Указание в структуре на месте IP-адреса *INADDR\_ANY*, говорит о том, что независимо от интерфейса, на который пришли данные, *UDP*-пакеты будут приняты, если они пришли на указанный порт.

Функция ПО СВС *svs\_bius* выступает в роли сервера поэтому в задаче необходимо связать *сокет* с локальным адресом протокола:

bind (sock, (sockaddr \*)&addr, sizeof(addr));

Дальнейшая настройка *сокета* производится с помощью функции:

setsockopt (int sock, int level, int optname, void \*optval, void optlen);

*sock* – идентификатор *сокета*; *level* – номер протокола управляющего флагами; *optname* – название опции; *optval* – указатель на структуру устанавливаемых параметров; *optlen* – размер структуры.

Данные отправляются при помощи функции:

int sendto(int sockfd, const void \*msg, int len, unsigned int flags, const struct sockaddr \*to, int tolen);

Данные принимаются при помощи функции:

int recvfrom(int sockfd, void \*buf, int len, unsigned int flags, struct sockaddr \*from, int \*fromlen);

# Разработка имитатора

## 4.1. Анализ требований

На этапе комплексного тестирования изделия, когда производится настройка взаимодействия с внешними системами, часто работы проводятся в ограниченные сроки. Такой имитатор должен обладать следующими свойствами:

* Формировать информацию аналогичную той, которая предоставляется внешними системами, включая недостоверные данные, для тщательной проверки ПО.
* Гарантировать соответствие частоты обмена данными с ЦВК, с частотой заданной в протоколах связи.
* Функционировать как на одном приборе вместе с ПО СВС, так и на отдельном.

## 4.2. Выбор структуры имитатора

Для аппаратного обеспечения имитатора необходимо три соединения по каналам *Манчестер* и одно соединение между последовательными портами *RS-232*, а также порт *Ethernet*. Реализация такой аппаратной структуры на одном приборе возможна, так как он имеет 6 ячеек с интерфейсом *Манчестера*, 8 с интерфейсом *RS-232* и 2 с интерфейс *Ethernet*. Схема реализация имитатора на одном ВК 27.07 представлена на рисунке 14.

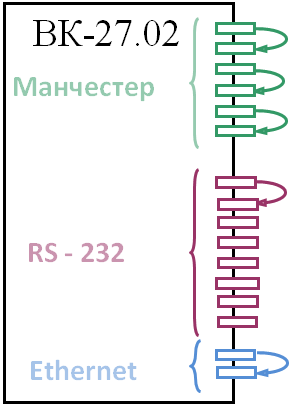


Рисунок 14 – Схема реализации имитатора на одном ВК-27.07

Схема реализация имитатора на двух ВК 27.07 представлена на рисунке 15.

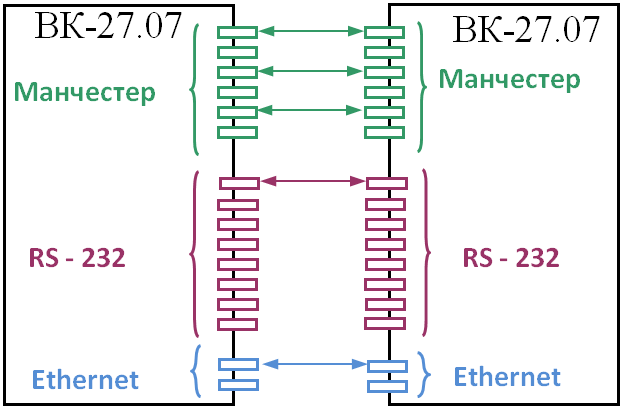


Рисунок 15 – Схема реализации имитатора на двух ВК-27.07

Реализация программного имитатора может осуществляться в рамках однопоточного или многопоточного процесса, либо с использованием нескольких независимых процессах. Так же как для ПО СВС, целесообразно выбрать структуру, состоящую из нескольких процессов, каждый из который имитирует работу одной внешней системы.

Управление приоритетами задач осуществляется системным вызовом *priocntl()*, он также может задать режим круговой диспетчеризации и определить квант времени для процесса. Квант времени определяет максимальное количество времени, в течение которого процесс нити может выполняться, при условии, что за это время он не закончился, не затребовал некоторый ресурс и не перешел в состояние *sleep*.

Планировщик ОС организует процессы в очереди по приоритетам, используя принцип очереди (FIFO), и учитывает состояние процессов для распределения процессорного времени. Процесс, выполняющийся после вызова *sleep*, остальные нити которые были переведены в *realtime* из другого класса планирования, процессы, которые полностью использовали свой квант времени и процессы, приоритет которых был изменен при помощи *priocntl()*, помещаются в конец очереди с соответствующим приоритетом. Нить, прерванная процессом с более высоким приоритетом, помещается в начало очереди, он будет выполняться раньше любого другого процесса с тем же приоритетом.

Приоритеты процессов ПО СВС установлены в соответствии со временем ответа. Учтено, что их значения не должны быть выше, чем значение приоритетов системных процессов, обеспечивающих работу всего ЦВК, и некоторых функциональных задач, чье выполнение по времени выполнения. Для функционирования в одном приборе ПО СВС и имитатора целесообразно установить приоритеты таким образом, чтобы приоритеты процессов, передающих данные, были ниже, чем приоритеты соответствующих задач приема информации. При переносе имитатора на отдельный прибор приоритеты процессов не изменяются.

Таблица 9 – Приоритеты процессов СВС и имитатора СВС.

|  |  |  |  |
| --- | --- | --- | --- |
| Название процесса обмена с системой | Приоритет | Название процесса обмена имитатора с системой | Приоритет |
| Метеорологическая система | 33 | Имитация метеорологической системы | 34 |
| Навигационная система | 41 | Имитация навигационной системы | 42 |
| БИУС | 37 | Имитация «БИУС» | 38 |
| Система единого времени | 53 | Имитация системы единого времени | 54 |

Данные, поступающие от имитатора, ПО СВС передает в функциональные задачи, которые их используют для своих алгоритмов. Важным требованием является формирование и передача имитатором информации, поступающей от внешних систем в реальных условиях, в том числе и недостоверной. Для обеспечения этого начальные данные от задач, имитирующих внешние системы, записываются в файлы перед запуском имитатора, и в процессе работы изменяются двумя способами:

* разовое обновление данных, перед запуском имитатора
* обновление, на заданную величину, каждой из передаваемых величин, с некоторой установленной частотой.

Считанные данные из файла проверяется каждая величина, если она не попала в диапазон, величина изменяется на значение по умолчанию.

Файлы с данными для внешних подсистем должны содержать следующую информацию:

1. Начальная величина
2. Значение, на которое будет изменяться величина
3. Частота изменения величины
4. Признак достоверности величины

При передаче данных первым способом параметры 2 и 3 имеют значения «0».

При изменении данных в процессе работы имитатора, если величина достигает минимального или максимального значения, то величина станет равна начальному значению из файла. После этого изменения величин повторяются с прежними параметрами.

Задачи ЦВК ГАС используют не всю информацию, приходящую от внешних систем, поэтому файлы с имитационной информацией, содержат значения только необходимых величин.

# Тестирование ПО

Тестирование производится для проверки работоспособности ПО. В ходе тестирования решаются следующие задачи:

* Определение наличия ошибок
* Диагностика и локализация ошибок
* Устранение ошибок ПО

Тестирование включает в себя следующие этапы:

1. Автономное тестирование ПО
2. Тестирование ПО совместно с имитатором на одном и двух ВК
3. Тестирование ПО с функциональными задачами ЦВК и СОРДиУ

## 5.1. Автономное тестирование

Первый этап заключается в индивидуальном тестировании ПО и имитатора. Этот этап проводится на одном компьютере с установленной ОС «Эльбрус-Д». На этом этапе осуществляется проверка архитектуры программного обеспечения, точность формирования и корректность преобразования данных.

## 5.2. Тестирование ПО на одном и двух вычислительных комплексах

Второй этап представляет собой совместную отладку ПО и имитатора на одном и двух вычислительном комплексе. На этом этапе происходит тестирование связи по каналам обмена, проверка современности передачи и приема данных, корректность преобразования данных. Сначала производится тестирование каждой внешней системы и ее имитатора по отдельности, после чего проверяется совместная работа всех задач ПО СВС и имитатора. Хотя использование встроенного имитатора удобно на начальных этапах, его применение при комплексной отладке может снизить производительность системы из-за использования общих ресурсов. Схема тестирования на одном приборе изображена на рисунке 16.

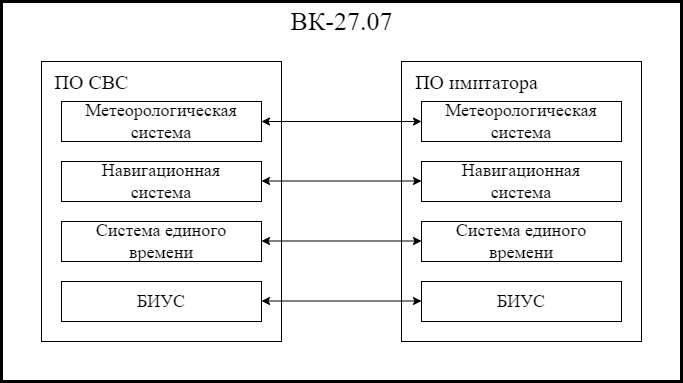


Рисунок 16 – Схема тестирования на одном ВК-27.07

Далее тестирование происходит на двух ВК-27.07. ПО СВС расположено на ВК-27.07.(1), ПО имитатора расположено на ВК-27.07(2). При переносе имитатора на внешний прибор осуществляется без изменения программного обеспечения. Схема тестирования на двух ВК-27.07 отображена на рисунке 17

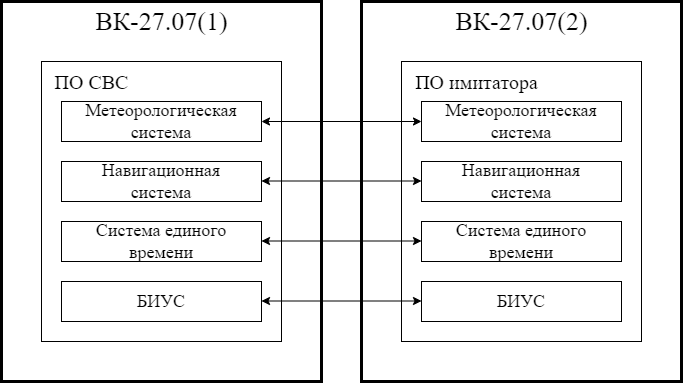


Рисунок 17 – Схема тестирования на двух ВК-27.07

## 5.3. Тестирование ПО на взаимодействие с задачами ЦВК ГАС и СОРДиУ

Третий этап включает в себя проверку взаимодействия ПО СВС с задачами ЦВК ГАС и СОРДиУ. На этом этапе осуществляется проверка передачи данных через систему обмена. Проверяется переключение режимов и каналов связи через отправку контрольных сообщений для получения дополнительной информации о состоянии сети. Все данные передаются через систему обмена в СОРДиУ. Схема тестирования отображена на рисунке 18.

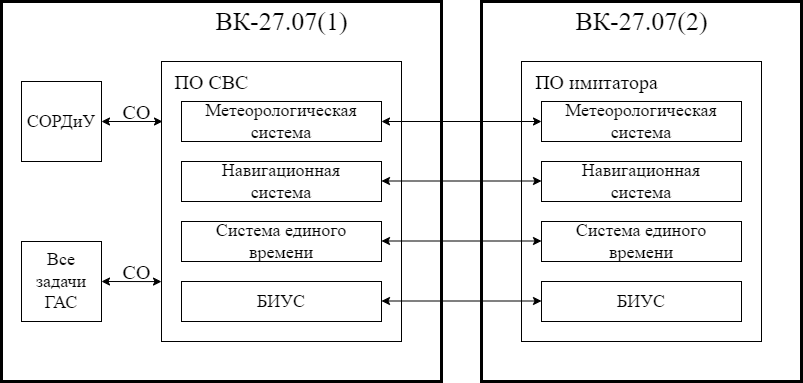


Рисунок 18 – Схема тестирования на двух ВК-27.07

## 5.4. Результаты тестирования

Целью тестирования на использование ресурсов системы являлось выявление неэффективного использования ресурсов и проверка корректности освобождения ресурсов после завершения работы. Благодаря консольной команде *top,* была оценена нагрузка на центральный процессор (ЦП) для каждого процесса, как в отсутствие имитатора, так и при его активности, а также при взаимодействии ПО обмена с внешними системами. Данные о нагрузке на ЦП отображены на таблице 10.

Таблица 10 – Нагрузка на ЦП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название процесса обмена | Нагрузка ЦП без вкл. имитатора | Нагрузка ЦП с вкл. имитатором | Нагрузка ЦП при взаимодействии с задачами ГАС | Макс.  значение |
| Метеорологическая система | ~ 0,07 % | ~ 0,67 % | ~ 1,35 % | < 1,5% |
| Навигационная система | ~ 0,1 % | ~ 0,89 % | ~ 0,91 % | < 1,3% |
| Система единого времени | ~ 0,02 % | ~ 0,34 % | ~ 0,47 % | < 1% |
| БИУС | ~ 0,2 % | ~ 2,55 % | ~ 3,33 % | < 4,5% |

Тестирование выполнения функциональных требований проводилось в три этапа:

* Автономное тестирование на ПК
* Тестирование на двух и одном ВК-27.07
* Тестирование с задачами ЦВК ГАС

При автономном тестировании и тестировании на одном и двух ВК-27.07 на ЦВК с ОС «Эльбрус-Д» использовался имитатор различных данных для метеорологической системы, навигационной системы, системы единого времени, системы «БИУС». При комплексном тестировании данные вводились вручную, используя управление СОРДиУ. Тестировался каждый модуль разработанного пакета прикладных программ, отображено на таблице Н.

Таблица Н – тестирование модулей

|  |  |  |  |
| --- | --- | --- | --- |
| Название модуля | Описание модуля | Количество тестов | Количество успешных тестов |
| NAV\_S | Навигационная система | 16 | 16 |
| MET\_S | Метеорологическая система | 12 | 11 |
| SEV | Система единого времени | 24 | 24 |
| BIUS | БИУС | 64 | 63 |

В ходе тестирования у «БИУС» и метеорологической системы при отправке данные не доходили до конечного абонента. После анализа программного кода и исправления найденных ошибок системы прошли все тесты, и работает корректно.

Таким образом, протестированные системы в автономном режиме, на одном и на двух ВК и при взаимодействии с задачами ЦВК ГАС прошли все проверки. Нагрузка на ЦП не превышает максимальные значения. Данные приходят с корректной задержкой у всех систем.

# Заключение

В ходе выполнения работы были изучены принципы работы Ethernet, RS и Манчестер. Произведена работа с драйверами устройств.

# Список использованных источников

1) Цифровой вычислительный комплекс современных гидроакустических систем, А.Р.Лисс, А.В.Рыжиков, А.В.Челпанов, АО «Концерн «Океанприбор» Журнал: «Гидроакустика». Санкт-Петербург: http://www.oceanpribor.ru/docs/SbGA2.pdf, 2000г.

2) ГОСТ Р 52070-2003, Интерфейс магистральный последовательный системы электронных модулей, ГОССТАНДАРТ РОССИИ Москва, 2003г.

3) Компьютерные сети, Э.Таненбаум, издательство «Питер», 2010г.

4) Аппаратные средства локальных сетей, Михаил Гук, издательство «Питер», 2005г.

5) Мониторинг и анализ сетей, Эд Уилсон, издательство «Лори», 2002г.

6) Программирование на С, Б.В. Керниган, Д.М. Ричи, издательство «Вильямс», 1998г.

7) Unix изнутри, Ю. Вахалия, издательство «Питер», 2003г.