

Федеральное агентство по образованию РФ
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО «АмГУ»)

УТВЕРЖДАЮ
Зав. кафедрой ИУС
А.В.Бушманов
« _____ » _____

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

по дисциплине «Системы реального времени»

для студентов специальности 230102 «Автоматизированные системы обработки информации и управления»

Составитель: ассистент кафедры ИУС Дрюков А.А.

Факультет математики и информатики

Кафедра информационных и управляющих систем

*Печатается по решению
редакционно-издательского совета
факультета математики и информатики
Амурского государственного
университета*

А.А. Дрюков

Учебно-методический комплекс по дисциплине «Системы реального времени». 230102 «Автоматизированные системы обработки информации и управления» очной формы обучения.- Благовещенск: Амурский гос. ун-т, 2008.

Пособие содержит рабочую программу, курс лекций, методические рекомендации по проведению и выполнению лабораторных работ. Составлено в соответствии с требованиями государственного образовательного стандарта.

© Амурский государственный университет, 2008

I. РАБОЧАЯ ПРОГРАММА

Курс 4	Семестр 8
Лекции 30 (час.)	Экзамен 8 семестр
Лабораторные работы 30 (час.)	
Самостоятельная работа 38 (час.)	
Всего часов 98 час.	

1. ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ, ЕЕ МЕСТО В УЧЕБНОМ ПРОЦЕССЕ

1.1. Цели и задачи дисциплины

Цель курса - ознакомить студентов с проблематикой встроенных систем реального времени, таких как: телефонные станции, роботы, военные приложения, а также с особенностями разработки ПО для них.

1.2. Задачи курса - изложение способов разработки ПО реального времени с учетом повышенных требований к надежности, эффективности, эргономичности и др.

1.3. Место курса в профессиональной подготовке выпускника. Для успешного освоения курса студенты должны быть знакомы с технологиями разработки ПО в обычных приложениях.

1.4. Требования к уровню освоения содержания курса. Этот курс необходимо прослушать тем студентам, которые собираются работать в области телекоммуникаций, компьютерных сетей и других приложений, в которых компьютеры управляют оборудованием в реальном масштабе времени.

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

2.1 Федеральный компонент

Программа курса «Системы реального времени» составлена в соответствии с требованиями Государственного образовательного стандарта высшего профессионального образования.

ГОСУДАРСТВЕННЫЙ СТАНДАРТ

СД.08 Системы реального времени. Особенности систем реального времени; аппаратурная среда, устройство связи с объектом; методы и средства обработки асинхронных событий; концепция процесса; ядро реального времени; ме-

ханизмы синхронизации и взаимодействия процессов; языки программирования реального времени; программирование синхронной и асинхронной обработки данных.

2.2 Наименование тем, их содержание, объем в лекционных часах

1. Тема 1 – 2 часа.

Определение СРВ. Жесткие и мягкие СРВ. Структура СРВ. Операционные системы реального времени (ОСРВ). Отличие ОСРВ от ОС общего назначения. Системы разработки и системы исполнения.

2. Тема 2 – 4 часа

Характеристики ОСРВ. Механизмы реального времени.

3. Тема 3 – 4 часа

Архитектура ОСРВ. Классы ОСРВ.

4. Тема 4 – 4 часа

Функции ядра ОСРВ. Процессы. Состояния процесса. Жизненный цикл процес-са. Поток. Приоритеты процессов.

5. Тема 5 – 4 часа

Организация вычислительного процесса в СРВ. Характеристики вычислительного процесса в СРВ. Требования к времени реакции системы. Многозадачность СРВ. Циклические и спорадические задачи.

6. Тема 6 – 4 часа

Средства взаимодействия с внешней средой в составе СРВ. Датчики. Исполнительные механизмы. Устройства связи с объектом.

7. Тема 7 – 4 часа

Структура СРВ. Аппаратная платформа СРВ. Организация ввода/вывода. Ввод/вывод по готовности. Ввод/вывод по прерыванию. ПДП. Таймеры. Watchdog. Real-time clock.

8. Тема 8 – 4 часа

Планирование задач. Алгоритмы планирования без переключения и с переключением. Схемы назначения приоритетов. FIFO диспетчеризация. Карусельная диспетчеризация. Адаптивная диспетчеризация.

2.3 Лабораторные занятия, их наименование и объем в часах

Среда разработки приложений реального времени

1. Основные понятия и панели LabVIEW.
2. Использование структур и построение графиков ввода.
3. Массивы и кластеры.
4. Подпрограммы.
5. Разработка VI для функционирования в сети Internet.
6. Разработка виртуального инструмента.
7. Взаимодействие vi по протоколу TCP.
8. Использование изображений.
9. Сбор данных с реального объекта.
10. Работа с файлами.
11. Разработка виртуального инструмента для поддержки CGI интерфейса.
12. Разработка виртуального инструмента используя DAQ Signal Generator.
13. Разработка виртуального инструмента позволяющего осуществлять передачу данных.
14. Моделирование систем управления производственными объектами в LabVIEW.

2.4 Вопросы для самостоятельного изучения

1. Проблемы взаимодействия процессов.
2. Семафоры.
3. Мьютексы.
4. Критические секции.
5. Задача «обедающих философов».

2.5 Вопросы к экзамену

1. Общая структура СРВ.
2. Определение СРВ.
3. Жесткие и мягкие СРВ.
4. Структура СРВ.
5. Операционные системы реального времени (ОСРВ).
6. Отличие ОСРВ от ОС общего назначения.
7. Системы разработки и системы исполнения.
8. Характеристики ОСРВ.
9. Механизмы реального времени.
10. Архитектура ОСРВ.
11. Классы ОСРВ.
12. Функции ядра ОСРВ.
13. Процессы в ОСРВ.
14. Состояния процесса в ОСРВ.

15. Жизненный цикл процесса в ОСРВ.
16. Потоки. Приоритеты процессов в ОСРВ.
17. Пример СРВ. Характеристики вычислительного процесса в СРВ.
18. Структура СРВ на примере задачи контроля испытательного оборудования.
19. Организация вычислительного процесса в СРВ.
20. Структура СРВ на примере задачи диспетчеризации.
21. Средства взаимодействия с внешней средой в составе СРВ.
22. Датчики как средство взаимодействия с внешней средой в составе СРВ.
23. Исполнительные механизмы как средство взаимодействия с внешней средой в составе СРВ.
24. Устройства связи с объектом.
25. УСО на примере ЦАП и АЦП.
26. Структура СРВ.
27. Аппаратная платформа СРВ.
28. Организация ввода/вывода. Ввод вывод по готовности.
29. Организация ввода/вывода. Ввод вывод по прерыванию.
30. Организация ввода/вывода. Прямой доступ в память.
31. Таймеры.
32. Планирование задач.
33. Алгоритмы планирования без переключения и с переключением.
34. Схемы назначения приоритетов.
35. FIFO диспетчеризация.
36. Карусельная диспетчеризация.
37. Адаптивная диспетчеризация.
38. Синхронизация процессов в СРВ.
39. Обмен информацией между процессами.
40. Требования к языкам и операционным системам реального времени.

3. ЛИТЕРАТУРА

Основная:

1. Б.С. Гольдштейн "Сигнализация в сетях связи", 2 тома, М.: Радио и связь, 1998
2. Сборник статей "Объектно-ориентированное визуальное моделирование" под редакцией проф. А.Н. Терехова, изд. СПбГУ, 1999
3. Журнал "СТА" (Современные Технологии Автоматизации).
4. Журнал "Открытые системы".

Дополнительная:

1. Основы автоматизации управления производством (Макаров И.Н. и др.)
М.: Высшая школа, 1983.

2. Основы управления технологическими процессами. (Анисимов А.С. и др., под ред. Райбмана Н.С.), М.: Наука, 1978.
3. Рей У. Методы управления технологическими процессами. М.: Мир, 1983.
4. Давиденко К.Я. Технология программирования АСУТП. М: Энергоатомиздат, 1986.
5. Первозванский А.А. Математические модели в управлении производством. М.: Наука, 1975.
6. Дегтярев Ю.И. Методы оптимизации. М.: Советское радио., 1980.
7. Карданский Л.А. и др. Централизованное управление машиностроительным оборудованием от ЭВМ. М.: Машиностроение, 1977.
8. Гроп Д. Методы идентификации систем. М.: Мир, 1979.
9. Басакер Р., Саати Т. Конечные графы и сети. М.: Наука, 1974.
10. Лысенко Э.В. Проектирование автоматизированных систем управления технологическими процессами. М.: Радио и связь, 1987.

II. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ И ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Лабораторные работы проводятся по подгруппам в компьютерном классе. Задания к лабораторным работам выполняются парой студентов на одном компьютере соответствии с вариантом.

Выполняя задание, студенты пользуются материалом, изложенным в тексте лабораторной работы; готовят письменный отчет, включающий краткое изложение проделанных действий, выводы.

Преподаватель, принимая лабораторную работу, проверяет навыки, полученные студентами при выполнении задания, отчет, задает дополнительные вопросы по теоретическому материалу.

III. ТЕХНОЛОГИЯ ВЫПОЛНЕНИЯ И ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Указание: сначала прочесть весь текст, иметь представление о содержимом, а затем выполнить задание. На каждом занятии студенты должны выполнить по одному заданию из приведенных в конце раздела. На последней паре проводится защита лабораторных работ.

1. Возможности и характеристики современных SCADA-систем

Разработка и выбор специализированного прикладного программного обеспечения для создания автоматизированных систем управления определенным технологическим процессом (АСУ ТП), в том числе и для решения образовательных задач, осуществляется по двум возможным направлениям:

- разработка программ на основе базовых традиционных языков программирования;
- использование коммерческих инструментальных проблемно ориентированных средств.

Использование уникального программного обеспечения для каждого конкретного проекта хотя и может быть наиболее оптимальным с точки зрения решения определенной задачи, но необходимость каждый разрешать задачу, практически, с нуля, рост временных и материальных затрат существенно снижает их достоинства. В данной связи все большее предпочтение промышленными, коммерческими и образовательными организациями отдается разработчикам специализированных операционных систем (ОС), аппаратного и программного обеспечения, предназначенных для конечных систем управления различными объектами типа SCADA-систем (от Supervisory Control And Data Acquisition – диспетчерское управление и сбор данных).

Перечислим некоторые популярные на западном и российском рынках SCADA-систем, имеющие поддержку в России [1]:

- Factory Link (United States DATA Co., USA);
- InTouch (Wonderware, USA);
- Genesis (Iconics, USA);
- WinCC (Siemens, Germany);
- Trace Mode (Ad Astra, Россия);
- RSView (Rockwell Software Inc, USA);
- LabVIEW, BridgeVIEW, LabVIEW RT, Lookout (National Instruments, USA) и др.

SCADA-системы, прежде всего, предназначены для получения и визуализацией информации от программируемых логических контроллеров (ПЛК), плат ввода-вывода информации, распределенных систем управления. Разработка на их основе комплексных, хорошо интегрированных инструментальных средств, обеспечивающих взаимодействие лабораторного оборудования различной степени сложности в автоматизированном режиме, позволяет реализовать на практике основные концепции использования современных информационно-коммуникационных технологий в образовательном процессе.

Рассмотрим основные возможности и характеристики современных SCADA-систем [1].

Функциональные возможности.

1. Разработка архитектуры всей системы автоматизации (на этом этапе определяется функциональное назначение каждого узла системы автоматизации).
2. Решение вопросов, связанных с возможной поддержкой распределенной архитектуры, необходимостью введения узлов с горячим резервированием и т.п.
3. Создание прикладной системы управления для каждого узла, где специалист в области автоматизируемых процессов наполняет узлы архитектуры алгоритмами, совокупность которых позволяет решать задачи автоматизации.
4. Приведение параметров прикладной системы в соответствие с информацией, которой обмениваются устройства нижнего уровня (ПЛС, АЦП, ЦАП) с внешним миром (датчиками температуры, давления и др.).
5. Отладка созданной прикладной программы в режиме эмуляции и реальном режиме.

Технические характеристики

1. Программно-аппаратные платформы. Анализ перечня таких платформ необходим, поскольку от него зависит распространение SCADA-системы на имеющиеся вычислительные средства, а также оценивание стоимости ее эксплуатации. Подавляющее большинство SCADA-систем реализовано на MS Windows-платформах (Windows NT).
2. Имеющиеся средства сетевой поддержки. Для эффективного функционирования системы автоматизации распределенных объектов SCADA-система должна обеспечивать высокий уровень сетевого сервиса. Необходима поддержка сетевых сред с использованием стандартных протоколов (Netbios, TCP/IP и др.), а также наиболее популярных сетевых стандартов из класса промышленных интерфейсов (Profibus, Canbus, LON, Modbus и т.д.).

3. Встроенные командные языки. Большинство SCADA-систем имеют встроенные языки высокого уровня, Basic-подобные языки, для создания фрагментов алгоритма, необходимых в решении задачи управления.
4. Поддерживаемые БД. Практически во всех SCADA-системах осуществлена поддержка SQL-синтаксиса, не зависящего от типа БД, что позволяет создавать независимые программы для анализа информации и использовать уже имеющееся ПО, ориентированное на обработку данных.
5. Графические возможности. Функционально графические интерфейсы SCADA-систем весьма похожи. В каждой из них существует графический объектно-ориентированный редактор с определенным набором анимационных функций. Используемая векторная графика дает возможность осуществлять широкий набор операций над выбранным объектом, а также быстро обновлять изображение на экране средствами анимации. Крайне важен вопрос о поддержке в рассматриваемых системах стандартных функций GUI (Graphic Users Interface). Поскольку большинство рассматриваемых SCADA-систем работает под управлением Windows, это и определяет тип используемого GUI.

Эксплуатационные характеристики

1. Удобство использования. Сервис, предоставляемый SCADA-системами на этапе разработки ППО, обычно очень развит. Почти все они имеют Windows-подобный пользовательский интерфейс, что во многом повышает удобство их использования, как в процессе разработки, так и в период эксплуатации прикладной задачи.
2. Наличие и качество поддержки. Возможны следующие уровни поддержки: услуги фирмы-разработчика, обслуживание региональными представителями фирмы-разработчика, взаимодействие с системными интеграторами, русификация программ и документации, горячая линия и решение проблем, связанных с индивидуальными требованиями заказчика и др.

В настоящем учебном пособии были изложены технологии разработки АЛП УД на базе SCADA-системы LabView (National Instruments). Рассмотрим основы создания ППО для сбора, обработки и представления в лабораторных практикумах в среде программирования LabVIEW.

2. Основы разработки ППО в среде программирования LabVIEW

Среда программирования LabView является продукцией компании National Instruments и представляет собой средство разработки ППО, близкое по своей логической структуре к конструкциям языков Си или Бейсик. Однако, LabView в отличие от них использует не текстовый язык программирования, а графиче-

ский – язык G. Он позволяет создавать программы в виде блок-схем.

LabView имеет обширные библиотеки функций для решения различных задач, таких как ввод/вывод, обработка, анализ и визуализация сигналов; контроль и управление технологическими объектами; статистический анализ и комплексные вычисления; взаимодействие процессов и сетевые технологии ActivX и TCP/IP; поддержка SQL запросов; работа с Internet и др.

Программные приложения, создаваемые в LabVIEW носят название **виртуальных инструментов (VI)**, включают две основные панели:

- **передняя или лицевая панель**, осуществляющая интерактивный интерфейс пользователя и имитирующая панель некоторого пульта управления с размещением на нем различных кнопок, графических индикаторов, диалоговых объектов, средств управления и индикации и т.д.;
- **функциональная панель или блок-схема**, в которой с помощью языка G осуществляется процесс разработки исходного кода виртуального инструмента в виде отдельных графических пиктограмм, осуществляющих различные функции, и связей между ними.

При этом виртуальные инструменты являются также аналогами функций языков программирования и подчиняются принципам иерархичности и модульности. В результате формируемые VI оказываются составленными из VI более низкого уровня (subVI), реализуя при этом концепцию модульного программирования. Возможно также накапливать и создавать собственные библиотеки виртуальных инструментов.

2.1. Основные понятия и панели LabVIEW

Запуск среды программирования LabVIEW осуществляется либо двойным кликом мыши на ярлыке LabVIEW, который находится на рабочем столе, либо из раздела Программы – National Instruments LabVIEW (изложение для Windows 9x, NT, 2000). При входе в главное меню LabVIEW (версия 5.1) пользователю предлагается создание нового виртуального инструмента (**New VI**) или открытие уже существующего (**Open VI**).

Разработка VI осуществляется на двух панелях, находящихся в двух окнах, - передней и функциональной. На передней панели визуально размещаются средства управления и индикации, на функциональной – составляется блок-схема или исходный код будущего VI. Структура панелей одинакова. Основным элементом каждой панели является рабочая область, снабженная горизонтальным и вертикальным скроллингами, в которой и размещаются элементы. Также на панелях имеется верхнее меню и набор функциональных кнопок. Размер окон может регулироваться пользователем. Размещение одновременно двух окон на экране – **Ctrl+T**. Активизация одной из панелей осуществляется посредством клика мыши в ее области или **Ctrl+E**. Имя панели соответствует имени загруженного в него VI. Если VI новый, то панель носит название

Untitled. Сохранение VI осуществляется через верхнее меню любой из панелей – **File-Save** или **File-Save As** для сохранения под новым именем.

Для обеих панелей доступна панель **Tools Palette** (рис.1), включающая набор управляющих кнопок для изменения режима редактирования. Перечислим некоторые из них:



Рис. 1. Панели Tools, Controls и Functions

- кнопка «указательный палец» – служит для изменения позиций выключателей и кнопок, управления значениями цифровых регуляторов, настройки виртуальных осциллографов и др.
- кнопка «стрелка» – выделение, перемещение объектов, изменение их размера.
- кнопка «А» – открытие и редактирование текстового окна.
- кнопка «катушка» – служит для соединения объектов на функциональной панели.
- кнопка «кисть» – раскрашивание объектов или фона.
- кнопка «рука» – перемещение рабочей области панели в окне.
- кнопка «пипетка» – выбор текущего цвета из имеющихся на панели.
- кнопка «красный круг» – для размещения и снятия точек остановки выполнения программы на функциональной панели.
- кнопка «Р» – для размещения на функциональной панели локальных окон для отображения текущих значений данных, передаваемых в ходе выполнения программы.

При активной передней панели становится доступной панель **Controls** (рис.1). С ее помощью осуществляется визуальное размещение регуляторов и

индикаторов на передней панели VI. **Регуляторы** предназначены для ввода информации в ходе выполнения программы, **индикаторы** – для вывода. В панели Controls они распределены по отдельным группам по некоторым признакам – числовые, логические, строковые, массивы, диалоговые, ActivX, Internet и др.

При активировании функциональной панели становится доступной панель **Functions** (рис.1), которая аналогично панели Controls включает систематизированные наборы стандартных элементов в виде отдельных пиктограмм, из которых осуществляется составление блок-схемы VI.

•

На передней и функциональной панелях также размещаются управляющие кнопки (рис. 2), такие как



- кнопка «стрелка» – пуск выполнения программы; если в программе имеются ошибки, то дан-

Рис. 2. Управляющие кнопки ная кнопка расколота на две части;
- кнопка «стрелки в цикле» – запуск программы в циклическом режиме;
- кнопка «красный круг» – остановка выполнения программы;
- кнопка «две вертикальные черты» – пауза в выполнении программы.

Процесс разработки VI включает:

1 Размещение регуляторов и индикаторов на передней панели VI. Для этого из панели Controls выбирается объект требуемого типа и внешнего вида и размещается в требуемом месте на передней панели. При этом его размер, цвет, описание и название могут в последующем меняться.

2 Добавление требуемых для прикладной задачи структур и функций на функциональной панели. Для этого из панели Functions выбираются соответствующие структуры и функции, пиктограммы которых размещаются на функциональной панели.

3 Соединение регуляторов, индикаторов, констант, функций и др. на функциональной панели при помощи проводки.

Регуляторы и индикаторы выполняют те же функции, что и входные и выходные параметры в текстовых языках программирования. При размещении регулятора/индикатора на передней панели, LabView создает соответствующую пиктограмму на блок-схеме. Символы на терминале соответствуют типу данных терминала. Например, **DBL** – терминал представляет данные в виде вещественных чисел с двойной точностью, **TF** – логический терминал, **I16** – терминал 16 – битных целых и др. (рис. 3).

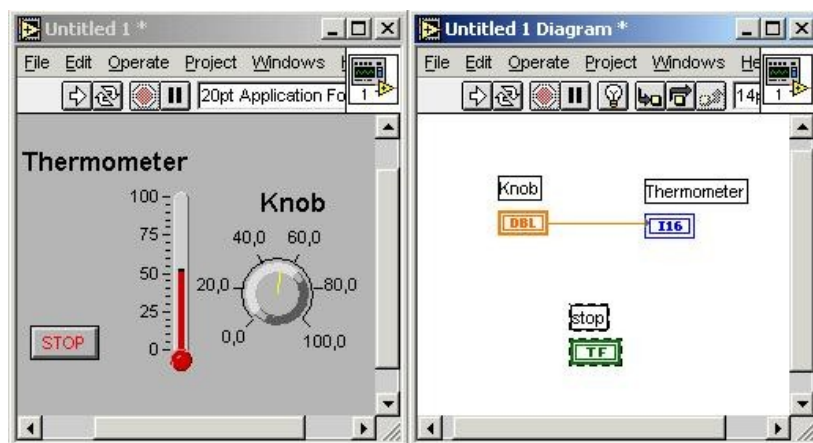


Рис. 3. Регуляторы и индикаторы

При нажатии правой кнопки мыши на регуляторе/индикаторе (как на передней, так и функциональной панели) появляется контекстное меню, с помощью которого возможно осуществить:

- замену индикатора на регулятор и наоборот (Change to Control, Change to Indicator);
- быстрый поиск терминала на функциональной панели (Find Terminal) и регулятора/индикатора на передней панели (Find Control, Find Indicator);
- демонстрацию или отказ от нее для названия и описания регулятора/индикатора (Show-Label, Show-Caption);
- настройку параметров регулятора/индикатора (Data Operations);
- замену на другой регулятор/индикатор (Replace);
- получение справки по используемой функции (Online Help);
- открытие для функций соответствующих им констант, индикаторов и регуляторов (Create Constant, Create Indicator, Create Control);
- и др.

Терминалы представляют собой области функции, через которые передается информация. Они аналогичны параметрам в текстовых языках программирования. Для того, чтобы увидеть какие терминалы включает данная функция необходима по правой кнопке мыши на пиктограмме из контекстного меню выбрать Show-Terminals (рис. 4).

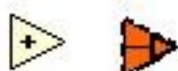


Рис. 4. Терминалы

Провода - пути данных между терминалами. Они аналогичны переменным на обычных языках (рис.3). Данные идут в только одном направлении, с исход-

ного терминала на один или более терминалов адресата. Провода имеют различную толщину и цвет. Синий цвет соответствует целым числам, оранжевый – вещественным числам, зеленый – логическим, лиловый – строковым данным. По мере перехода от скаляра к массиву и кластеру увеличивается толщина провода.

Для соединения терминалов необходимо подвести курсор мыши к исходному терминалу (из панели Tools выбрана кнопка «катушка»). При этом отдельные части пиктограммы, соответствующие различным терминалам начинают мигать, а также появляются всплывающие подсказки для облегчения идентификации терминала. После выбора нужного терминала на нем необходимо кликнуть левой кнопкой мыши. В этом случае один конец провода станет закрепленным за данным терминалом. Другой конец, перемещая курсор мыши, необходимо подвести к терминалу адресата и кликнуть левой кнопкой на нем. Если данное соединение возможно, то провод станет соответствующего типу передаваемых данных цвета, в противном случае он станет пунктирным черного цвета. Удаление всех некорректных соединений **Ctrl+B**.

В случае необходимости возможно удаление отдельных сегментов связей, ведение ответвлений от существующих проводов.

Пиктограмма VI соответствует каждому виртуальному инструменту и располагается в правом верхнем углу передней панели (рис. 3). Для редактирования пиктограммы используется упрощенный графический редактор, позволяющий создавать изображение, закрашивая его отдельные пиксели. Для этого необходимо вызвать контекстное меню на иконке в правом верхнем углу передней панели, и выбрать **Edit Icon**.

Коннектор представляет собой программный интерфейс виртуального инструмента. При использовании регуляторов или индикаторов на передней панели для передачи данных в VI, эти объекты должны иметь терминалы на панели коннектора. Он вызывается из контекстного меню на пиктограмме VI **Show Connector**. При этом выделяются терминалы для регуляторов на левой половине панели, а для индикаторов – на правой в соответствии с их количеством. Соответствие терминала индикатору или регулятору устанавливается щелчком левой кнопки мыши на терминале коннектора, а затем на соответствующем индикаторе или регуляторе. Это особенно важно при использовании разрабатываемого VI в других виртуальных инструментах для обеспечения возможности его подключения.

SubVI является аналогом подпрограммы. В создаваемом VI возможно использование любого виртуального инструмента, имеющего коннектор. Базовые настройки и тип разрабатываемого VI устанавливаются в контекстном меню пиктограммы – пункт **VI Setup**.

Панель Controls служит для добавления регуляторов и индикаторов к передней панели. Если панель Controls не видна на экране, ее можно открыть через верхнее меню **Windows – Show Controls Palette**. Панель Controls доступна, только если активно окно передней панели. Рассмотрим основные подпанели панели Controls.

- **Numeric** (числовые значения). Состоит из регуляторов и индикаторов для числовых данных.
- **Boolean** (Булевы значения). Состоит из регуляторов и индикаторов для булевых величин.
- **String&Table** (строковые значения и таблицы). Состоит из регуляторов и индикаторов для ASCII строк и таблиц.
- **List & Ring** (списки и закольцованные списки). Состоит из регуляторов и индикаторов для меню, выполненных в форме списков и закольцованных списков.
- **Array & Cluster** (массивы и кластеры). Состоит из регуляторов и индикаторов для группировки наборов типов данных.
- **Graph** (виртуальные осциллографы). Состоит из индикаторов, для построения графиков данных в графах или диаграммах в реальном масштабе времени.
- **Path & Refnum** (пути и ссылки). Состоит из регуляторов и индикаторов для путей и ссылок.
- **Decorations** (оформление). Состоит из графических объектов для настройки дисплеев передней панели.
- **Select Control** (выбор регулятора). Отображает диалоговое окно для загрузки самодельных элементов управления.
- **User Controls** (средства управления пользователем). Состоит из специальных средств управления, которые формирует сам пользователь.
- **ActiveX** (объекты ActiveX). Состоит из средств управления, позволяющих внедрить объекты ActiveX на переднюю панель.
- **Dialog** (диалоговая панель). Состоит из стандартных объектов для формирования диалога с пользователем.
- **IMAQ Vision** (обработка изображений). Состоит из средств обработки и анализа изображений.
- **Internet Toolkit** (работа с Internet). Состоит из средств управления, располагаемых на передней панели, позволяющих организовывать работу виртуальных инструментов в сети Internet (ftp, электронная почта, telnet, CGI и другие).

Панель Functions предназначена для формирования блок-схемы VI. Каждая пиктограмма на панели открывает подпанель пиктограмм нижнего уровня. Если панель Functions не видна на экране, ее можно открыть через верхнее меню Windows – Show Functions Palette. Панель Functions доступна, только если активно окно функциональной панели. Рассмотрим основные подпанели панели Functions.

- **Structures** (структуры). Состоит из управляющих структур программы, таких как циклы For Loop, While Loop и другие.
- **Numeric** (числовые функции). Состоит из тригонометрических, логарифмических и других функций.
- **Boolean** (Булевы функции). Состоит из логических и Булевых функций.

- **String** (строковые функции). Состоит из функций для работы со строковыми величинами.
- **Array** (массивы). Состоит из функций для обработки массивов.
- **Cluster** (кластеры). Состоит из функций для обработки кластеров.
- **Comparison** (сравнение). Состоит из функций для сравнения переменных.
- **Time & Dialog** (время и диалог). Состоит из функций для диалоговых окон, синхронизации, и обработки ошибок.
- **File I/O** (ввода/вывода файла). Состоит из функций для осуществления операций по вводу/выводу файлов.
- **Instrument I/O** (инструменты ввода/вывода). Состоит из VI для связи и управления приборами различной архитектуры.
- **Instrument Drivers** (драйверы приборов). Состоит из VI, способных управлять внешними приборами, осциллоскопами, генераторами, и т.д., через последовательный порт или интерфейс GPIB.
- **Data Acquisition** (сбор данных). Состоит из VI для использования плат сбора данных.
- **Signal Processing** (обработка сигналов). Состоит из VI для генерации и обработки сигналов.
- **Mathematics** (математические). Состоит из оптимизационных, алгебраических, интегральных, дифференциальных и других функций.
- **Graphics & Sound** (графика и звук). Состоит из VI для работы трехмерной графикой, изображениями и звуком.
- **Communication** (связи). Состоит из виртуальных приборов для работы с сетями TCP, DDE и др.
- **Application Control** (управление приложением). Состоит из VI, управляющих виртуальными приборами.
- **Advanced** (расширенная). Состоит из разных функций типа функции библиотечного запроса, манипуляции данными и др.
- **Report Generation** (генерация отчета). Состоит из VI, используемых для подготовки отчетных документов.
- **Tutorial** (обучающие программы). Состоит из VI, используемых в обучающей программе LabVIEW.
- **User Libraries** (пользовательские библиотеки). С помощью нее организуется быстрый доступ к нужному vi.
- **Select VI** (выбор VI). Состоит из диалогового окна для внедрения подпрограмм в текущий ВП.
- **IMAQ Vision** (обработка изображений). Состоит из VI, используемых для обработки и анализа изображений.
- **Image Acquisition** (получение изображения). Состоит из VI, используемых для получения и обработки изображений.
- **Internet Toolkit** (работа с Internet). Состоит из VI, используемых для работы в сети Internet (ftp, электронная почта, telnet, CGI и другие).
- **SQL** (SQL запросы). Состоит из VI, используемых для организации связи с SQL сервером и обработки запросов.

2.2. Использование структур и построение графиков ввода

Для графического отображения полученных данных используются диаграммы или виртуальные осциллографы. Диаграмма (**Chart**) - это виртуальный осциллограф, экран которого обновляется по мере поступления новых данных. Располагается в панели Controls-Graph-Waveform Chart. Настройка диаграммы осуществляется пользователем. При этом могут быть использованы полоса прокрутки (scrollbar), легенда (legend), палитра (palette), цифровой дисплей (digital display) и др. Возможно одновременное отображение на одной диаграмме нескольких зависимостей разным цветом или типом линии, имеющих одну вертикальную шкалу или несколько (контекстное меню на диаграмме Stack Plots/Overlay Plots). Для очистки экрана осциллографа необходимо в его контекстном меню выбрать Data Operations-Clear Chart.

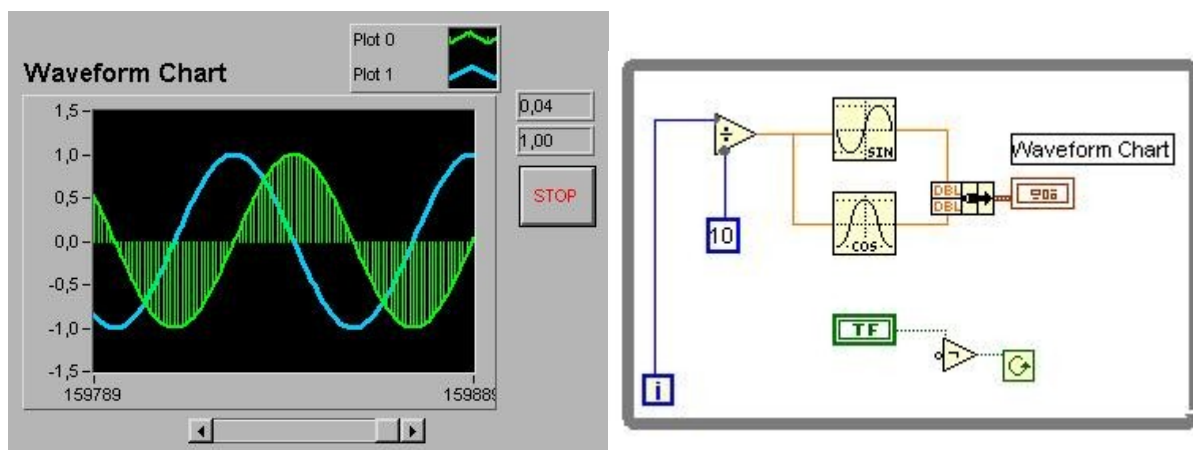


Рис. 5. Диаграмма Chart(Выводит на график значения \sin и \cos до нажатия кнопки stop).

Используется функция Bundle из панели Cluster) Возможны различные виды представления графиков в виртуальном осциллографе (Data Operations-Update Mode):

- Strip - отображение информации подобно действию самописца на бумажной ленте, т.е. новое значение наносится слева, если линия дошла до края области отображения, предыдущие значения начинают сдвигаться вправо.
- Score - отображение информации подобно работе осциллографа, т.е. когда линия достигает правого края экрана, экран обновляется, и линия снова идет с левого края.
- Sweep подобен режиму Score, но экран не очищается при достижении линией правой границы дисплея. Место начала нового цикла отмечает красная вертикальная черта, которая смещается влево по мере поступления новой информации.

Структура предназначена для управления прохождением данных в виртуальных инструментах. В языке G используется пять структур.

1. While Loop - условный цикл.
2. For Loop - счетный цикл.
3. Case Structure - выбор.
4. Sequence Structure - последовательность.
5. Formula Node - формульный блок.

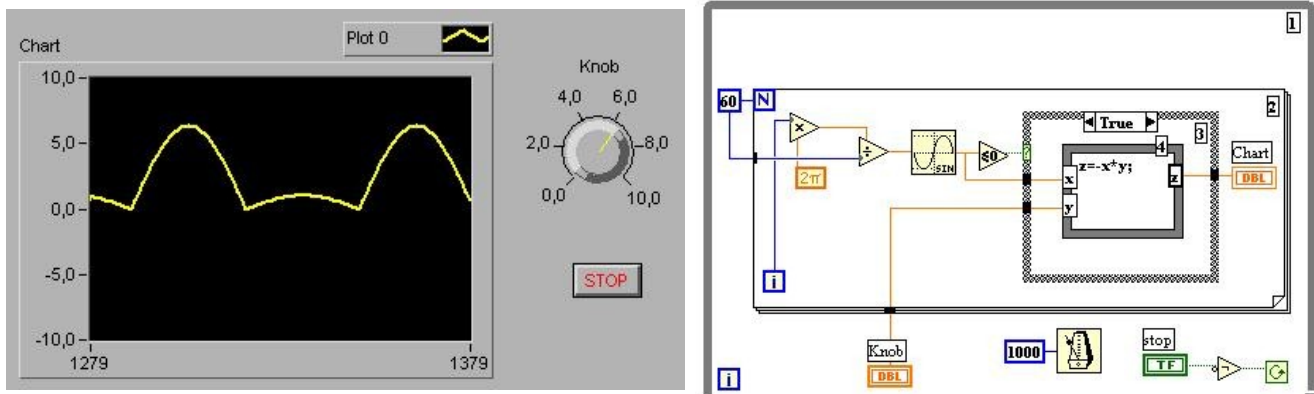


Рис. 6. Структуры While Loop, For, Case, Formula Node

Условный и счетный циклы (While Loop и For Loop) являются базовыми структурами языка G, как и многих других текстовых языков программирования.

Условный цикл (**While Loop**) осуществляет выполнение части программы определенное число раз, которое задается некоторым условием. Цикл While Loop включает (рис. 6, рамка 1).

- Ограниченную прямоугольную область, изменяемого размера, - тело цикла.
- Терминал условия, определяющий момент окончания работы цикла (момент, когда на него подается значение true). VI проверяет значение этого терминала после выполнения цикла, поэтому такой цикл выполняется, по меньшей мере, один раз.
- Терминал итераций (i), который показывает количество выполнений данного цикла. Если цикл выполнен 1 раз, то значением на этом терминале будет 0.

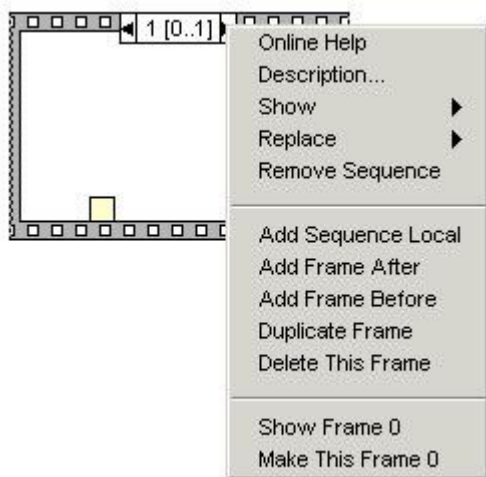
Счетный цикл (**For Loop**) выполняет тело цикла определенное число раз. Цикл For включает (рис. 6, рамка 2).

- Ограниченную прямоугольную область, изменяемого размера, - тело цикла.
- Терминал счетчик. Определяет сколько раз должен выполняться цикл (N).
- Терминал итераций, показывающий текущее число выполненных циклов (i).

В структуре выбор Case (рис. 6, рамка 3) имеется две или более встроенных блок-схемы. Выбор одной из них, которая будет выполнена определяется в зависимости от значения, поданного на вход данной структуры. Структура Case включает.

- Терминал выбора (?). Значение, подаваемое на него, может быть целым, логическим или строковым.
- Переключатель блок-схем (True \ False \ и т.д.). Позволяет переходить от одной блок-схемы к другой. Содержит по умолчанию два окна
- True и False. При необходимости количество блок-схем выбора может быть увеличено. Кроме True и False в качестве значений переключателя могут использоваться целые числа или строковые значения. В данном случае, значение, поданное на терминал выбора, будет сравниваться со значением переключателя данной блок-схемы. Всегда необходимо предусматривать блок-схему для False.

Формульный блок **Formula Node** (рис. 5, рамка 4) позволяет вводить



формулы в обычном виде прямо в блок-схему. Особенно это удобно, когда выражение имеет много переменных и сложный вид. Формулы вводятся как простой текст. При этом создаются терминалы на границе блока (контекстное меню Add Input или Add Output), в которые вписываются имена переменных. Каждое выражение заканчивается разделителем ";". Описание синтаксиса формул, а также используемых функций и операторов содержится в Help-Formula Node.

Структура последовательность **Sequence Structure** (рис. 7) выполняет встроенные в нее блок-схемы последовательно в определенном порядке. Количество встроенный блок-схем определяется числом фреймов данной структуры.

Рис. 7. Структура Sequence

Их количество добавляется при помощи контекстного меню - Add Frame After, Add Frame Before. Для передачи значений переменных из фрейма в фрейм используются локальные переменные структуры (контекстное меню - Add Sequence Local variable), создаваемые на границе фрейма. Данные, связан-

ные с такой переменной доступны во всех последующих фреймах и не доступны в предыдущих.

2.3. Массивы и кластеры

Массив - набор данных одного типа. Массив может иметь одно или несколько измерений. Доступ к элементу массива осуществляется по индексу. Индекс - это число от 0 до $n-1$, где n это число элементов массива. Для инициализации массива необходимо выбрать в панели Functions-Array-Array Constant или Controls-Array&Cluster. Используя Operating tool из палитры инструментов, вы можете выбрать числовую, логическую или строковую константу, которую нужно поместить в пустой массив.

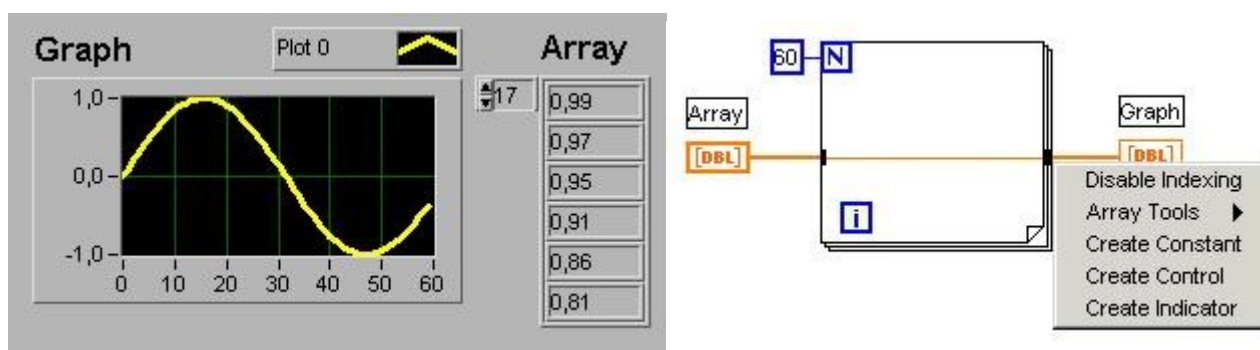


Рис. 8. Массивы и диаграмма Graph

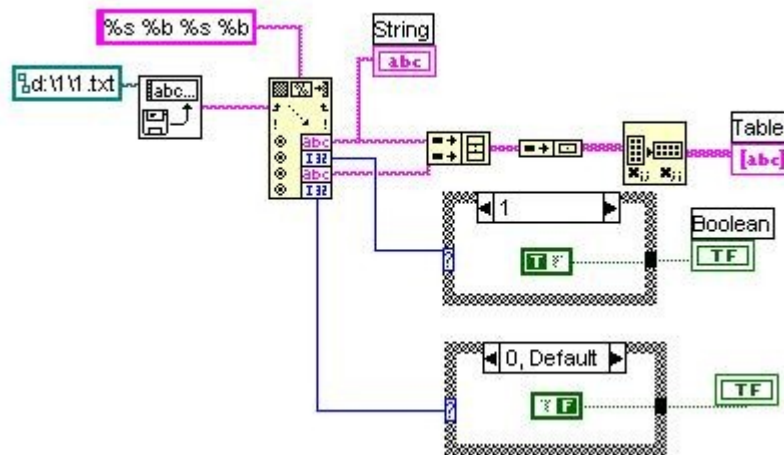
Структуры For Loop и While Loop могут использоваться для автоиндексации массивов (рис.8). Если связать внешний массив с блоком внутри цикла через входной канал, то цикл будет последовательно считывать элементы массива, по одному за цикл. Цикл будет считывать скаляры из одномерного массива, одномерные массивы из двумерного, и так далее. Если массив связан через выходной канал, то элементы будут записываться в массив.

Для включения и выключения автоиндексации (рис. 8) используется контекстное меню на входном/выходном каналах цикла - маленьких черных квадратов границы цикла (Disable Indexing / Enable Indexing). Для сборки элементов с образованием массива используются функция Build Array (Functions-Build Array).

Кластеры - упорядоченная совокупность элементов различного типа. Сборка и разборка кластера осуществляется функциями Functions-Bundle, Functions-Unbundle. Кластеры могут использоваться при выводе нескольких графиков на диаграмме Chart (рис. 5).

Осциллограф Waveform Graph (рис.8) позволяет наблюдать временные зависимости сигналов. Он регистрирует процесс за время одного пуска програм-

мы на числе выборок, которое устанавливается в программе. Он обновляется при новом запуске VI и может быть много лучевым



2.5. Подпрограммы

Любой vi может быть использован как подпрограмма при создании в последующем других виртуальных инструментов. Для объединения нескольких функциональных блоков разрабатываемой диаграммы в подпрограмму достаточно выделить их мышкой на диаграмме, удерживая клавишу Shift, и затем выбрать в верхнем меню пункт Edit - Create SubVI. При этом они объединятся в новую подпрограмму с новым значком на функциональной панели. Двойной клик на данном значке позволит вызвать созданную подпрограмму, настроить ее должным образом и сохранить с заданным именем. В последующем данный модуль может быть многократно использован в различных vi.

Для редактирования значка создаваемых vi и подпрограмм достаточно

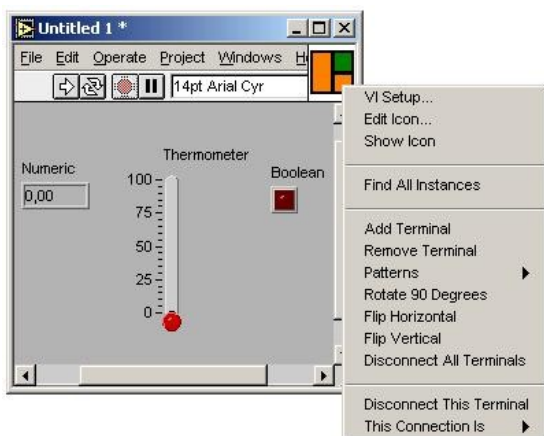


Рис. 10. Строки, таблицы, файлы

и вид которых можно редактировать с помощью всплывающего меню пиктограммы (добавить/удалить терминал - Add Terminal/Remove Terminal, поворот на 90 градусов - Rotate 90 Degrees, другой

кликнуть ПКМ на пиктограмме vi в правом верхнем углу и выбрать пункт Edit Icon... (рис. 10) С помощью простейших функций графического редактора можно создать собственный вариант иконки.

Настройка входов/выходов (терминалов) подпрограмм осуществляется следующим образом. Необходимо нажать ПКМ на пиктограмме vi в правом верхнем углу и выбрать пункт Show Connector... (рис. 10). При этом пиктограмма разделится на несколько прямоугольников, общий набор

вид - Patterns... и др.) Для того, что бы сопоставить каждый терминал с определенными данными необходимо ЛКМ кликнуть на нужном терминале, а затем на том индикаторе или регуляторе на передней панели, которой он будет соответствовать. При этом терминал окрасится в цвет соответствующий типу данных указанного индикатора или регулятора. В результате все терминалы будут связаны с определенными входными или выходными данными

3. Разработка VI для функционирования в сети Internet

При разработке лабораторных практикумов удаленного доступа на базе имитационных моделей и реального оборудования возникает проблема дистанционного управления лабораторным стендом и отображения пульта управления на экране обучающегося. Среда программирования LabVIEW предоставляет широкие возможности для создания программного обеспечения по управлению реальными объектами, в том числе с возможностью удаленного доступа через сеть Internet.

Для представления виртуальных инструментов в браузере пользователя используется G Web Server, входящий в состав Internet Developers Toolkit. После инсталляции данного пакета запуск веб сервера осуществляется из верхнего меню Project – Internet Toolkit – Start HTTP Server... При этом запускается виртуальный инструмент, осуществляющий поддержку http сервера на данной машине (рис. 11). После этого к ней можно обращаться через ее IP или DNS адрес в адресной строке браузера. При отладке программного обеспечения к http серверу можно обращаться локально через localhost.



Рис. 11. HTTP сервер

Сервер располагается в папке Program Files/National Instruments/LabVIEW/internet/. В папке home размещаются html страницы. Первоначально загружается файл index.htm, находящийся в папке home. Кроме того, в папке home находится папка cgi-bin, где размещаются виртуальные инструменты, осуществляющие CGI интерфейс.

Для размещения изображения передней панели vi на html странице используется следующий тэг:

```
<P><IMG SRC="http://62.76.177.133/.snap?teploten_otd.vi"
ALIGN="BOTTOM" BORDER="1" USEMAP="#panel" ISMAP></P>
```

Возможно вместо параметра snap использовать параметр monitor для анимаци-

онных изображений, тогда после имени vi задаются параметры &refresh=1&lifespan=60 (обновление будет через 1 сек в течение 60 сек). Не все браузеры поддерживают данный режим.

Отображение передней панели виртуального инструмента на машине клиента будет возможно только в том случае, если на сервере данный vi уже будет запущен. Для обеспечения возможности удаленного запуска через Internet браузер виртуального инструмента, обслуживающего лабораторную установку, а также последующего управления ею, необходима разработка vi, осуществляющего поддержку CGI интерфейса. Данный vi обязательно должен находиться внутри папки cgi-bin.

Рассмотрим набор основных функций, используемых при разработке vi для cgi интерфейса и управления другими приложениями (рис. 12).

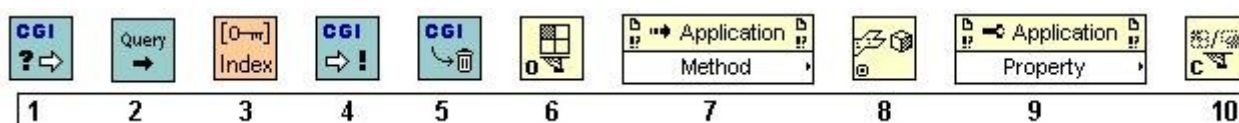


Рис. 12. Функции для CGI интерфейса и работы с другими приложениями

1. CGI Reade Request (Internet-CGI-CGI) – ожидает запрос и при подключении использует env - окружение (Keyed Array). Cgi connection inf - используется для передачи информации о соединении.
2. CGI Get Query Parameters (Internet-CGI)– возвращает параметры, пришедшие с cgi запросом (от окружения), которые в последующем и приходится разбирать по переменным.
3. Keyed Array Index (Internet-CGI-Keyed Array)- возвращает значение элемента массива (array in) по ключу -(key) – имени переменной окружения (page -- страница которой передается дальнейшее управление, temp– увеличение/уменьшение температуры, control– запуск/закрытие виртуального инструмента). valid key - указывает, что данная переменная получена; value - содержит значение полученной переменной.
4. CGI Write Reply (Internet-CGI-CGI)- пишет ответ на http соединение. В header его подается значение переменной page – страницы, на которую передается управление в последующем.
5. CGI Release (Internet-CGI-CGI)– информирует сервер, что обработка запроса закончена.
6. Open VI Reference (Application Control - Open VI reference) - возвращает ссылку на VI указанный в vi path -- локальный путь к запускаемому vi.
7. Invoke Node (Application Control) - вызов методов и действий с vi.
 - o установка работы с виртуальным инструментом – контекстное меню, Select VI Server Class - Virtual Instruments
 - o метод - контекстное меню - Methods -

- Get Control Value - получение значения от указанного vi по соответствующей переменной (по Control Name). Переменная указывается четко по ее названию в запущенном vi (можно русское название). Соединяются между собой через reference и ошибки.
 - Set Control Value - установка (передача) значения установленной переменной.
 - Run VI - запускает vi, необходимо подать False, чтобы не ждал завершения выполнения программы.
 - Abort VI - останавливает работу запущенного vi.
8. Unflatten From String / Flatten To String (Advanced - Data Manipulation) - конвертирует бинарную строку к приведенному типу, какой указывает пользователь (в вещественные, булевы и др.), или нечто (вещественные, булевы и др.) конвертирует в бинарную строку для передачи запущенному vi - через Set Control Value (Type Description - строка).
 9. Property Node (Application Control) - пишет или считывает информацию о свойствах vi.

- открытие передней панели - контекстное меню-Property-Front Panel Window-Open (true - открыть, false - закрыть), затем контекстное меню-Change to Write. Соединяется через reference.

1. Close Application or VI Reference (Application Control - Open VI reference) - закрывает открытые vi или соединения.

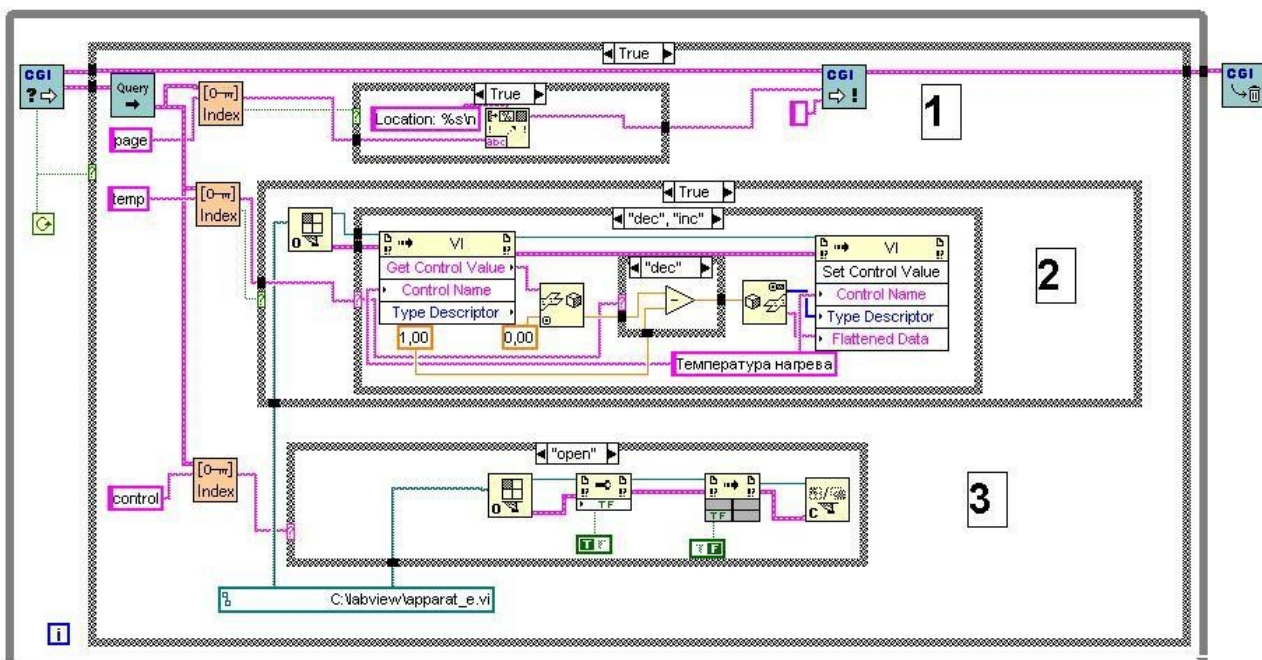


Рис. 13. Пример виртуального инструмента для CGI интерфейса.

На рис. 13 изображен вариант виртуального инструмента, который позво-

ляет осуществлять дистанционный запуск и управление виртуальным инструментом `apparat_e.vi`. В качестве входных параметров могут быть заданы:

- название Internet страницы, которая в последующем будет загружена клиенту (блок 1) – параметр `page`;
- команда на запуск двигателя (`dvig`), насоса (`nasos`), нагревателя (`nagrev`) или уменьшение/увеличение параметра регулирования (`dec`, `inc` именно данный вариант показан на рис. 13) (блок 2) – параметр `temp`;
- открытие или закрытие `vi` на сервере (блок 3) – параметр `control`.

Ссылка на данный `vi` из запускаемой Internet страницы имеет вид:

```
<a href="http://localhost/cgi-bin/apparat/apparat CGI_e.vi?control=open&page=/apparat/apparat_map.htm">
```

Текст Internet страницы, на которой отображается передняя панель пульта управления `vi` `apparat_e.vi` может быть представлена в виде:

```
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type"
CONTENT="text/html;CHARSET=win-1251">
  <META NAME="Author" Content="Internet Toolkit">
</HEAD>

<BODY>   <p align="center"><font color="#800000"
size="4"><strong>Опрос каналов и управление работой установки</strong></font></p>
<P><IMG SRC="http://localhost/.snap?apparat_e.vi"
ALIGN="BOTTOM" BORDER="1" USEMAP="#panel" ISMAP></P>
<MAP Name="panel">
<AREA Shape="Rect" coords = "408,18,680,162"
      HREF="http://localhost/cgi-bin/apparat/apparat CGI_e.vi?page=/apparat/apparat_map.htm">
<AREA Shape="Rect" coords = "408,182,680,326"
      HREF="http://localhost/cgi-bin/apparat/apparat CGI_e.vi?page=/apparat/apparat_map.htm">
<AREA Shape="Rect" coords = "408,344,680,489"
      HREF="http://localhost/cgi-bin/apparat/apparat CGI_e.vi?page=/apparat/apparat_map.htm">
```

```

<AREA      Shape="Rect"      coords      =      "280,453,386,483"
                                HREF="http://localhost/cgi-
bin/apparat/apparat_cgi_e.vi?control=close&page=/in-
dex.htm">
<AREA      Shape="Rect"      coords      =      "58,408,80,441"
                                HREF="http://localhost/cgi-
bin/apparat/apparat_cgi_e.vi?page=/apparat/apparat_map.ht
m&                                temp=dvig">

<AREA      Shape="Rect"      coords      =      "196,408,218,441"
HREF="http://localhost/cgi-
bin/apparat/apparat_cgi_e.vi?page=/apparat/apparat_map.ht
m&                                temp=nagrev">

<AREA      Shape="Rect"      coords      =      "328,408,350,441"
HREF="http://localhost/cgi-
bin/apparat/apparat_cgi_e.vi?page=/apparat/apparat_map.ht
m&                                temp=nasos">

</MAP>

</BODY>
</HTML>

```

В блоке MAP Name="panel" осуществляется картирование изображения передней панели vi, при помощи которого выделяются органы управления на пульте (кнопки, переключатели, регуляторы, экраны обновления и др.) Нажатие левой кнопки мыши на выделенные фрагменты рисунка вызывает передачу cgi файлу необходимых параметров, в соответствии с которыми он осуществляет определенные действия (открытие соответствующей Internet страницы, запуск двигателя, изменение величины параметра регулирования, открытие vi и др.)

4. Взаимодействие vi по протоколу TCP

Среда программирования LabVIEW позволяет создавать виртуальные инструменты, осуществляющие передачу данных между собой с использованием протокола TCP. Функции для создания таких приложений находятся в панели Functions–Communication–TCP. На рис. 14 и 15 приведен пример двух взаимосвязанных приложений.

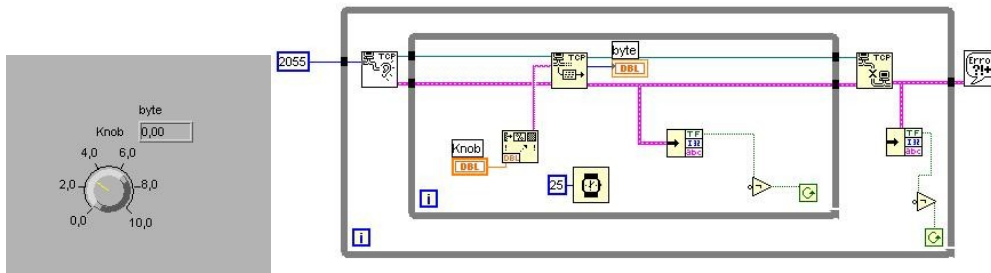


Рис. 14. Взаимодействие по протоколу TCP (серверная часть)

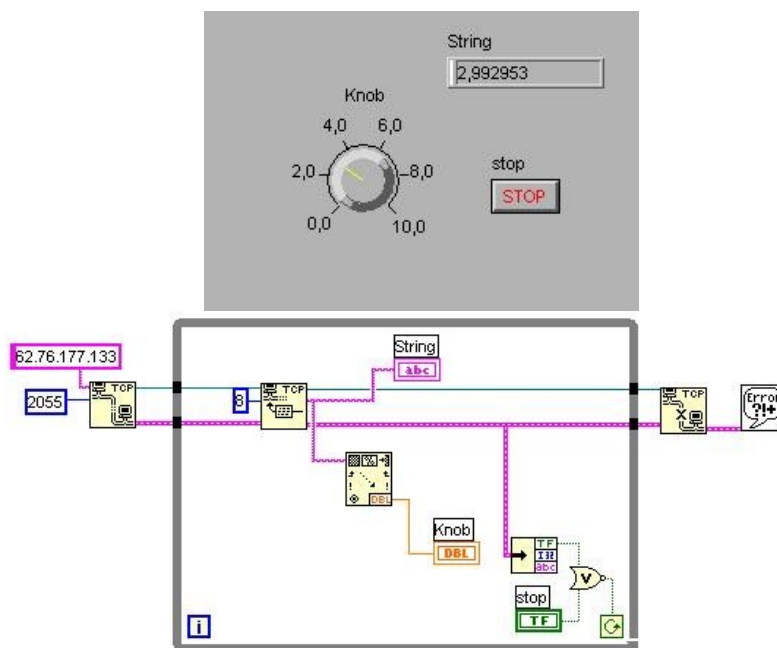


Рис. 15. Взаимодействие по протоколу TCP (клиентская часть)

Первый vi выполняет роль сервера, который ожидает подключения по одному из портов TCP (в данном случае взят порт 2055). Для этого использована функция TCP Listen. Когда клиент подключается к серверу с использованием функции TCP Open Connection (указывается IP адрес машины и номер порта), сервер передает (функция TCP Write) по созданному соединению данные (в данном случае это значение цифрового регулятора Knob). Клиент считывает (функция TCP Read) поступившие данные (в количестве 8 байт) и отображает их на своем цифровом индикаторе. При возникновении ошибки или остановке клиента соединение разрывается (функция TCP Close Connection).

Создание подобных приложений также может быть использовано при организации удаленного доступа к лабораторным практикумам как имитационного характера, так и на базе реального оборудования. Рассмотрим способы работы с реальными объектами в среде программирования LabVIEW.

5. Использование изображений

При использовании изображений, содержащихся в графических файлах (jpeg, png, bmp) на передней панели vi размещается индикатор Picture, расположенный на панели Controls–Graph–Picture. В данном случае используемый рисунок будет выводиться на экран в выделенную данным индикатором область. При необходимости на нее могут дополнительно устанавливаться требуемые органы управления и контроля.

Для считывания графического файла, например jpeg, используется функция Functions–Graphics&Sound–Graphics Formats–Read JPEG file (рис. 16).

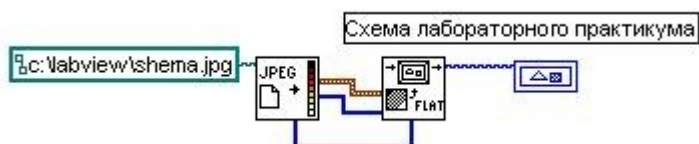


Рис. 16. Подключение графических файлов

На вход ей подается путь к выводимому файлу. На выходе функции формируется карта пиксел, в последующем преобразуемая в «картинку» с использованием функции Functions–Graphics&Sound–Picture Functions–Draw Flattened Bitmap. Сформированное изображение подается на индикатор Picture.

6. Сбор данных с реального объекта

В настоящее время использование компьютеров в научных исследованиях не ограничивается имитационным моделированием на основе математических моделей. Все чаще современная вычислительная техника применяется для приема, обработки и анализа сигналов от реальных физических объектов и управления ими [2]. При этом возникает потребность в электрических датчиках, преобразователях сигналов и специальном программном обеспечении. Среда программирования LabVIEW является удобным программно-аппаратным комплексом для разработки приложений, позволяющих осуществлять опрос датчиков, установленных на объекте исследования, обработку полученной информации, генерацию сигналов для его управления в диапазоне 10 В (от –5В до +5 В для знакопеременных сигналов; от 0 до 10 В для однополярных).

При обработке сигналов и их преобразовании из аналоговых в цифровые и наоборот могут быть использованы платы ЦАП/АЦП. Рассмотрим основные приемы сбора и обработки сигналов с помощью среды программирования LabVIEW на примере многофункциональной платы компании National Instruments PCI-MIO-16E-1. Плата устанавливается в свободный слот компьютера и при помощи соединительного кабеля (TYPE SH68-68-EP) подключается к коннектору (TBX-68). Датчики, установленные на объекте исследования, подключаются к компьютеру через коннектор в соответствии с настройкой каналов. В таблице 1 приведено соответствие основных каналов клеммам коннектора.

Для настройки каналов ЦАП/АЦП необходимо запустить приложение Measurement & Automation Explorer (рис. 17) и убедиться, что плата установлена и определена верно. При этом в папке Devices and Interfaces появится ЦАП/АЦП, а в скобках будет указан номер данного устройства (например, Device 1). В папке Data Neighborhood содержится список настроенных каналов. Выделив ЛКМ интересующий канал, можно запустить его тест (кнопка Test), просмотреть свойства (кнопка Properties...) или удалить его (кнопка Delete). Для настройки нового канала необходимо кликнуть ПКМ на папке Data Neighborhood, выбрать пункт Create New..., пункт Virtual Channel и затем нажать кнопку Finish. Далее будет предложено выбрать тип настраиваемого канала: аналоговый вход (Analog Input); аналоговый выход (Analog Output); цифровой вход/выход (Digital I/O).

Таблица 1

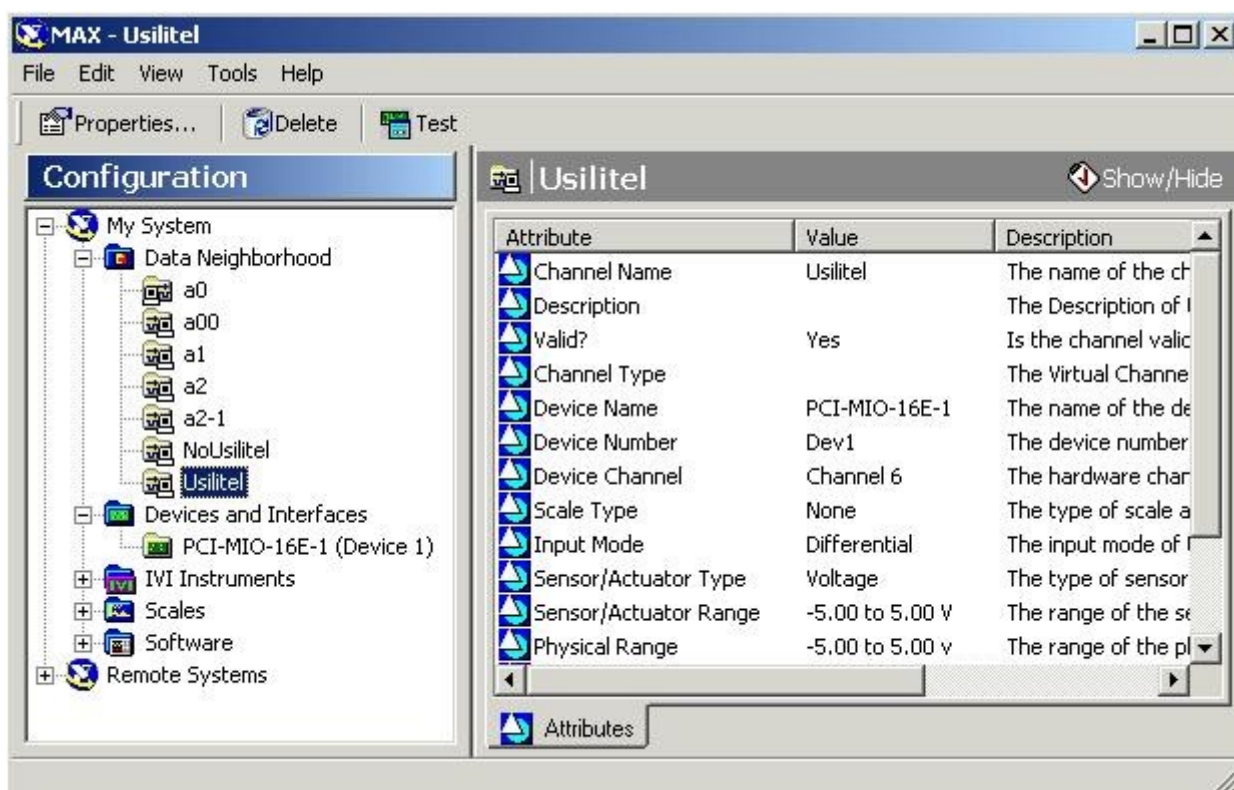


Рис. 17. Приложение Measurement & Automation Explorer

В следующем окне необходимо указать имя канала (Channel Name) и краткое его описание (Channel Description). Для Analog Input и Analog Output указываются далее:

- тип датчика (напряжение, температура, частота и др.);
- единицы измерения (Units) и диапазон измерения (Range);
- возможное масштабирование (Scaling);
- устройство, для которого данный канал настраивается (What DAQ hardware will be used?);

- номер канала и его соответствие по таблице 1 (Which channel on your DAQ hardware? и Pins);
- тип Analog Input (Which analog input mode will be used?)
 - дифференциальный (Differential);
 - с общим проводом, заземленным в конце (Referenced Single Ended);
 - с общим проводом, незаземленным в конце (Nonreferenced Single Ended).

Для Digital I/O указываются:

- тип цифрового входа/выхода (Read from Port, Read from Line, Write to Port, Write to Line);
- устройство, порт и номер линии с соответствием по таблице 1;
- линии, по которым осуществляется инвертирование цифрового сигнала (Invert Line).

После настройки канала его можно использовать при создании виртуальных инструментов. Функции для получения и формирования аналоговых и цифровых сигналов располагаются в панели Function–Data Acquisition. Перечислим основные из них:

Аналоговый вход (указываются номер устройства, имя одного или нескольких каналов):

- однократное считывание по одному из каналов (AI Sample Channel) – выдает значение полученного сигнала по одному каналу в виде числа;
- однократное считывание из группы каналов (AI Sample Channels) – выдает значения полученных сигналов по группе каналов в виде одномерного массива чисел;
- синхронное считывание по одному из каналов (AI Acquire Waveform) – выдает значения полученного сигнала по одному каналу в виде одномерного массива чисел (указывается число выборок и частота сканирования за секунду (1/с));
- синхронное считывание из группы каналов (AI Acquire Waveforms) выдает значения полученных сигналов по группе каналов в виде двумерного массива чисел (указывается число выборок и частота сканирования (1/с)).

Для изменения диапазона измерения используются терминалы high limit и low limit, которые автоматически изменяют коэффициент усиления сигнала (таблица 2).

Таблица 2

Коэффициенты усиления

Диапазон сигнала	Коэффициент усиления
$\pm 10 \text{ V}$	0.5

$\pm 5 \text{ V}$	1
$\pm 2.5 \text{ V}$	2
$\pm 1 \text{ V}$	5
$\pm 500 \text{ mV}$	10
$\pm 250 \text{ mV}$	20
$\pm 100 \text{ mV}$	50
$\pm 50 \text{ mV}$	100

Аналоговый выход. Набор функций подобен аналоговому входу (соответственно – AO Update Channel, AO Update Channels, AO Generate Waveform, AO Generate Waveforms). Также указывается номер устройства и имя канала (каналов). При их выполнении на соответствующие каналы подается заданное выходное напряжение в виде одного числа (AO Update Channel, AO Update Channels) или массива чисел (AO Generate Waveform, AO Generate Waveforms).

Цифровой вход/выход. Используются для считывания или записи логического выражения отдельных цифровых линий или цифрового канала в целом. Для считывания/записи логического выражения (true/false) отдельной цифровой линии настроенного канала используются функции Read from Digital Line и Write to Digital Line. Для них указывается номер устройства и имя цифрового канала, а также номер линии. При выводе данных задается подаваемое логическое значение (true/false), а при получении данных осуществляется его считывание. Для работы с каналом в целом (Read from Digital Port, Write to Digital port) ввод/вывод данных осуществляется аналогично отдельной цифровой линии, но данные имеют вид 8 битного шаблона.

Помимо перечисленных простейших функций для работы каналами ввода/вывода данных в LabVIEW содержится широкий спектр специальных функций, позволяющих осуществлять конфигурирование, сканирование, буферизированный ввод/вывод, очистку и множество других функций. Необходимость их использование при создании виртуальных инструментов определяется конкретной задачей сбора данных.

На рис. 18 приведен пример использования каналов аналогового входа/выхода. Первоначально были настроены каналы a0, как аналоговый выход (DAC0OUT), и a00, как аналоговый вход (канал 1 ACH1). В примере генерируется функция \sin переменного периода, значения которой подаются на аналоговый выход (a00), соединенный напрямую с аналоговым входом (a0). Коммутация каналов осуществляется на базе устройства DAQ Signal Generator, предназначенного для лабораторного изучения основных подходов обработки сигналов с использованием аппаратного обеспечения компании National Instruments и среды программирования LabVIEW. В Chart выводится сигнал, снимаемый с аналогового входа (a0).

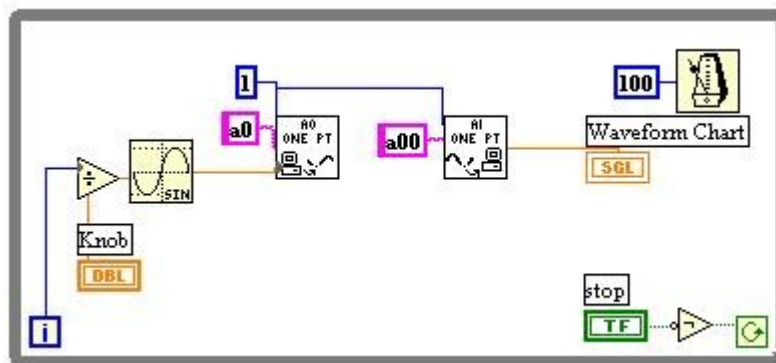


Рис. 18. Аналоговый вход/выход

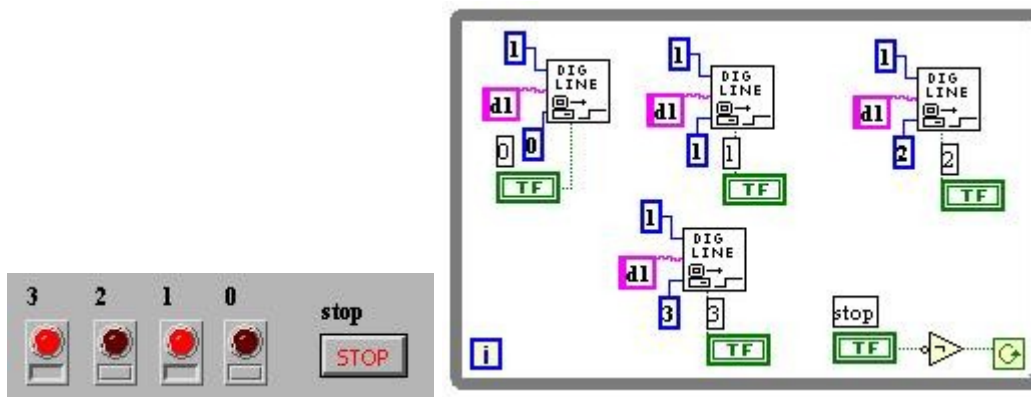


Рис. 19. Цифровой вход/выход

На рис. 19 приведен пример использования каналов цифрового входа/выхода. На устройстве DAQ Signal Generator имеется 4 светодиода, подключенных к цифровому порту. Канал d1 настроен как цифровой вход/выход для записи в порт с инверсией сигнала по всем линиям (DIO). Нажатие кнопки на передней панели v_i вызывает включение/выключение соответствующего ей светодиода на приборе.

В заключении отметим, что изложенный в данном пособии материал по среде программирования LabVIEW охватывает лишь необходимые начальные сведения по созданию виртуальных приборов и их применению в разработке автоматизированных лабораторных практикумов удаленного доступа на базе

имитационных математических моделей и реального оборудования. Более полную информацию читатель может получить из справки LabVIEW Help, на серверах www.ni.com, www.insysltd.ru, acs.levsha.ru, www.vitec.ru, phys.kemsu.ru и др.

7. Упражнения для лабораторных работ

1. Разработать виртуальный инструмент, включающий два цифровых регулятора и два цифровых индикатора, позволяющий выполнять сложение и умножение двух чисел.

2. Разработать виртуальный инструмент для измерения температуры и уровня жидкости в аппарате. Изменение параметров имитировать с помощью датчика случайных чисел. Предусмотреть как графическое отображение параметров, так и цифровое.

датчик случайных чисел – Functions–Numeric–Random number (0–1)

3. Смоделировать нагрев жидкости в баке, изменение температуры происходит по закону $T(t) = 0.1t^2 + t + 21$ в качестве модели часов используйте – Controls – Numeric – Knob. Необходимо что бы модель позволяла узнавать температуру в любой заданный момент времени. Часы представить в виде секундомера.

4. Построить график функции
 $y = ax^2 + bx + ab$

в диапазоне от -10 до 20 с шагом 2. Предусмотреть возможность изменения параметров a и b.

5. Разработать виртуальный инструмент для управления наполнением емкости жидкостью при помощи насоса. Предусмотреть возможность регулирования расхода жидкости, сигнализацию заполнения емкости (достижение уровня – 0.95) и автоматическое отключение насоса.

6. Разработать виртуальный инструмент, в котором осуществляется наполнение емкости насосом (аналогично работе 3), затем 5 секундная задержка на физико-химические превращения (горит сигнальный индикатор) и далее опорожнение емкости.

временная задержка – Functions–Time&Dialog–Wait Until Next ms Multiple.

создание локальной переменной – контекстное меню на индикаторе или регуляторе Create–Local Variable (в нее можно писать или из нее читать – контекстное меню на ней Change To Read Local / Change To Write Local).

7. Разработать виртуальный инструмент, отображающий значения функций \sin и \cos на одном Waveform Graph за 20 проходов цикла For.

8. Связать и затем развязать кластер, включающий строку (ФИО), массив числовой (оценки экзаменам за сессию), лампочку (горит – переведен на другой семестр, не горит – не переведен).

9. Записать в текстовый файл строку из 10 символов (название функции), 20 числовых значений (значения функции), текущее время, единицу и ноль (для отображения данных значений на двух лампочках). Затем считать данные значения из файла и отобразить их на соответствующих приборах.

10. Разработать виртуальный инструмент для поддержки CGI интерфейса, позволяющий осуществлять дистанционный запуск vi на другом компьютере, его закрытие, изменение значения его числового (увеличение/уменьшение значения переменной на) и булевого параметров. Продемонстрировать его работу на базе созданной Internet страницы.
11. Используя DAQ Signal Generator разработать виртуальный инструмент, позволяющий регистрировать значения датчика температуры, включение/ выключение и регулировку интенсивности накала светодиода.
12. Разработать виртуальные инструменты, позволяющие осуществлять передачу данных (включение/выключение тумблера и изменение значения цифрового регулятора) от одного к другому с использованием протокола TCP.
13. Создать модель системы регулирования уровня в баке.
14. Создать модель регулирования температуры и давления в котле.

8. Литература

1. Куцевич Н.А. SCADA-системы. Взгляд со стороны. (http://www.ipu.ru/period/asu/Contents/Number1/Contents/page_22-28.htm).
2. Жарков Ф.П., Каратаев В.В., Никифоров В.Ф., Панов В.С. Использование виртуальных инструментов LabVIEW – М.: Солон-Р, Радио и связь, Горячая линия – Телеком, 1999.–268 с.

IV. КОНСПЕКТ ЛЕКЦИЙ

Тема 1 (2 часа).

Определение СРВ. Жесткие и мягкие СРВ. Структура СРВ. Операционные системы реального времени (ОСРВ). Отличие ОСРВ от ОС общего назначения. Системы разработки и системы исполнения.

Существует несколько определений систем реального времени (СРВ), большинство из которых противоречат друг другу. Приведем несколько из них, чтобы продемонстрировать различные взгляды на назначение и основные задачи СРВ.

1. Системой реального времени называется система, в которой успешность работы любой программы зависит не только от ее логической правильности, но от времени, за которое она получила результат. Если временные ограничения не удовлетворены, то фиксируется сбой в работе системы.

Таким образом, временные ограничения должны быть гарантировано удовлетворены. Это требует от системы быть предсказуемой, т.е. вне зависимости от своего текущего состояния и загруженности выдавать нужный результат за требуемое время. При этом желательно, чтобы система обеспечивала как можно больший процент использования имеющихся ресурсов.

2. Стандарт POSIX (POSIX (Portable Operating System Interface for Computer Environment [for Unix]) интерфейс переносимой операционной системы, стандарт POSIX разработанный в 1988 г. IEEE развивающийся набор стандартов для UNIX (стандарт ISO/IEC 9945-1), описывающих интерфейсы этой ОС, а впоследствии дополненный расширениями для ОС реального времени (стандарт ISO/IEC 9945-1). Разработан для обеспечения переносимости приложений между различными версиями UNIX и других ОС. В частности, POSIX IEEE 1003.1 определяет базовый API для ядра ОС, IEEE 1003.2 (ISO/IEC 9945-2) - оболочку (shell) и утилиты, а IEEE 1003.4 - расширения для поддержки работы в реальном масштабе времени) 1003.1 определяет СРВ следующим образом: «Реальное время в опе-

рациональных системах – это способность операционной системы обеспечить требуемый уровень сервиса в заданный промежуток времени».

3. Иногда системами реального времени называются системы постоянной готовности (on-line системы), или «интерактивные системы с достаточным временем реакции». Обычно это делают по маркетинговым соображениям. Действительно, если интерактивную программу называют работающей в «реальном времени», то это просто означает, что она успевает обработать запросы от человека, для которого задержка в сотни миллисекунд даже незаметна.
4. Иногда понятие «система реального времени» отождествляют с понятием «быстрая система». Это не всегда правильно. Время задержки СРВ на событие не так уж важно (оно может достигать нескольких секунд). Главное, чтобы это время было достаточно для рассматриваемого приложения и *гарантировано*. Очень часто алгоритм с гарантированным временем работы менее эффективен, чем алгоритм, таким действием не обладающий. Например, алгоритм «быстрой» сортировки в среднем работает значительно быстрее других алгоритмов сортировки, но его гарантированная оценка сложности значительно хуже.
5. Во многих сферах приложения СРВ вводят свои понятия «реального времени». Например, процесс цифровой обработки сигнала называют идущим в «реальном времени», если анализ (при вводе) и/или генерация (при выводе) данных может быть проведен за то же время, что и анализ и/или генерация тех же данных без цифровой обработки сигнала. Например, если при обработке аудио данных требуется 2.01 секунд для анализа 2.00 секунд звука, то это не процесс реального времени. Если же требуется 1.99 секунд, то это процесс реального времени.

Назовем системой реального времени аппаратно-программный комплекс, реагирующий в предсказуемые времена на непредсказуемый поток внешних событий

Это определение означает, что:

- Система должна успеть отреагировать на событие, произошедшее на объекте, в течение времени, критического для этого события (meet deadline). Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается ошибкой для систем реального времени.
- Система должна успевать реагировать на одновременно происходящие события. Даже если два или больше внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение интервалов времени, критического для этих событий.

Хорошим примером задачи, где требуется СРВ, является управление роботом, берущим деталь с ленты конвейера. Деталь движется, и робот имеет лишь маленькое временное окно, когда он может ее взять. Если он опоздает, то деталь уже не будет на нужном участке конвейера, и, следовательно, работа не будет сделана, несмотря на то, что робот находится в правильном месте. Если он позиционируется раньше, то деталь еще не успеет подъехать, и робот заблокирует ей путь.

Другим примером может быть самолет, находящийся на автопилоте. Сенсорные серводатчики должны постоянно передавать в управляющий компьютер результаты измерений. Если результат какого-либо измерения будет пропущен, то это может привести к недопустимому несоответствию между реальным состоянием систем самолета и информацией о нем в управляющей программе.

Различают системы реального времени двух типов - системы жесткого реального времени и системы мягкого реального времени.

Системы жесткого реального времени не допускают никаких задержек реакции системы ни при каких условиях, так как:

- результаты могут оказаться бесполезны в случае опоздания,
- может произойти катастрофа в случае задержки реакции,
- стоимость опоздания может оказаться бесконечно велика.

Примеры систем жесткого реального времени - бортовые системы управления, системы аварийной защиты, регистраторы аварийных событий.

Системы мягкого реального времени характеризуются тем, что задержка реакции не критична, хотя и может привести к увеличению стоимости результатов и снижению производительности системы в целом.

Пример - работа сети. Если система не успела обработать очередной принятый пакет, это приведет к таймауту на передающей стороне и повторной отправке (в зависимости от протокола, конечно). Данные при этом не теряются, но производительность сети снижается.

Основное отличие между системами жесткого и мягкого реального времени можно выразить так: система жесткого реального времени никогда не опаздывает с реакцией на событие, система мягкого реального времени - не должна опаздывать с реакцией на событие.



Рисунок 1. Пример организации системы реального времени

Любая система реального масштаба времени может быть описана как состоящая из трех основных подсистем, как изображено на рисунке 1.

Управляемая (контролируемая) подсистема (например, промышленный завод, управляемое компьютером транспортное средство), диктует требования в реальном масштабе времени; **подсистема контроля (контролирующая)** управляет некоторыми вычислениями и связью с оборудованием с ОС (для использования от управляемой подсистемы); **подсистема оператора (операционная)** контролирует полную деятельность системы. Интерфейс между управляемыми и подсистемами контроля состоит из таких устройств как датчики и при-

воды. Интерфейс между управляющей подсистемой и оператором связывает человека с машинной.

Управляемая подсистема представлена задачами (в дальнейшем называемыми прикладными задачами) которые используют оборудование, управляемое подсистемой контроля. Эта последняя подсистема может быть построена из очень большого количества процессоров, управляющими такими местными ресурсами, как память и устройства хранения, и доступ к локальной сети в реальном масштабе времени. Эти процессоры и ресурсы управляются системой программного обеспечения, которую мы называем **операционной системой реального масштаба времени** (RTOS – real time operating system).

Назовем **операционной системой реального времени** такую систему, которая может быть использована для построения систем жесткого реального времени.

Это определение выражает отношение к операционным системам реального времени как к объекту, содержащему необходимые инструменты, но также означает, что этими инструментами еще необходимо правильно воспользоваться.

Большинство программного обеспечения ориентировано на «слабое» реальное время, а задача СРВ – обеспечить уровень безопасного функционирования системы, даже если управляющая программа никогда не закончит своей работы.

ОС общего назначения, особенно многопользовательские, такие как UNIX, ориентированы на оптимальное распределение ресурсов компьютера между пользователями и задачами (системы разделения времени). В операционных системах реального времени подобная задача отходит на второй план - все отступает перед главной задачей - успеть среагировать на события, происходящие на объекте.

Другое отличие - применение операционной системы реального времени всегда связано с аппаратурой, с объектом, с событиями, происходящими на объекте. Система реального времени, как аппаратно-программный комплекс,

включает в себя датчики, регистрирующие события на объекте, модули ввода-вывода, преобразующие показания датчиков в цифровой вид, пригодный для обработки этих показаний на компьютере, и, наконец, компьютер с программой, реагирующей на события, происходящие на объекте. Операционная система реального времени ориентирована на обработку внешних событий. Именно это приводит к коренным отличиям (по сравнению с ОС общего назначения) в структуре системы, в функциях ядра, в построении системы ввода-вывода. Операционная система реального времени может быть похожа по пользовательскому интерфейсу на ОС общего назначения (к этому, кстати, стремятся почти все производители операционных систем реального времени), однако устроена она совершенно иначе - об этом речь пойдет далее.

Кроме того, применение операционных систем реального времени всегда конкретно. Если ОС общего назначения обычно воспринимается пользователями (не разработчиками) как уже готовый набор приложений, то операционная система реального времени служит только инструментом для создания конкретного аппаратно-программного комплекса реального времени. И поэтому наиболее широкий класс пользователей операционных систем реального времени - разработчики комплексов реального времени, люди, проектирующие системы управления и сбора данных. Проектируя и разрабатывая конкретную систему реального времени, программист всегда точно знает, какие события могут произойти на объекте, знает критические сроки обслуживания каждого из этих событий.

Одно из коренных внешних отличий систем реального времени от систем общего назначения - четкое разграничение систем разработки и систем исполнения. **Система исполнения** операционных систем реального времени - набор инструментов (ядро, драйверы, исполняемые модули), обеспечивающих функционирование приложения реального времени.

Большинство современных ведущих операционных систем реального времени поддерживают целый спектр аппаратных архитектур, на которых работают системы исполнения (Intel, Motorola, RISC, MIPS, PowerPC, и другие). Это

объясняется тем, что набор аппаратных средств - часть комплекса реального времени и аппаратура должна быть также адекватна решаемой задаче, поэтому ведущие операционные системы реального времени перекрывают целый ряд наиболее популярных архитектур, чтобы удовлетворить самым разным требованиям по части аппаратуры. Система исполнения операционных систем реального времени и компьютер, на котором она исполняется называют "**целевой**" (target) системой. Система разработки - набор средств, обеспечивающих создание и отладку приложения реального времени.

Системы разработки (компиляторы, отладчики и всевозможные tools) работают, как правило, в популярных и распространенных ОС, таких, как UNIX и Windows. Кроме того, многие операционные системы реального времени имеют и так называемые резидентные средства разработки, исполняющиеся в среде самой операционной системы реального времени.

Заметим, что функционально средства разработки операционных систем реального времени отличаются от привычных систем разработки, таких, например, как Developers Studio, TaskBuilder, так как часто они содержат средства удаленной отладки, средства профилирования (измерение времен выполнения отдельных участков кода), средства эмуляции целевого процессора, специальные средства отладки взаимодействующих задач, а иногда и средства моделирования.

Тема 2 (4 часа)

Характеристики ОСРВ. Механизмы реального времени.

Как было сказано ранее, сердцем системы реального времени является ОСРВ (операционная система реального времени). В связи со специфичностью решаемых задач, ОСРВ должна обладать определенными свойствами.

Время реакции системы на внешние события. Согласно определению, ОСРВ должна обеспечить требуемый уровень сервиса в заданный промежуток времени. Этот промежуток времени задается обычно периодичностью и скоро-

стью процессов, которым управляет система. Приблизительное время реакции в зависимости от области применения ОСРВ может быть следующее:

- математическое моделирование - несколько **микросекунд**
- радиолокация - несколько **миллисекунд**
- складской учет - несколько **секунд**
- управление производством - несколько **минут**

Видно, что времена очень разнятся и накладывают различные требования на вычислительную установку, на которой работает ОСРВ.

Итак, как же определить время реакции системы? Поймем, какие времена мы должны знать для того, чтобы предсказать время реакции системы. События, происходящие на объекте, регистрируются датчиками, данные с датчиков передаются в модули ввода-вывода (интерфейсы) системы. Модули ввода-вывода, получив информацию от датчиков и преобразовав ее, генерируют запрос на прерывание в управляющем компьютере, подавая ему тем самым сигнал о том, что на объекте произошло событие. Получив сигнал от модуля ввода-вывода, система должна запустить программу обработки этого события. Интервал времени - от события на объекте и до выполнения первой инструкции в программе обработки этого события и является **временем реакции системы на события**, и, проектируя систему реального времени, разработчики должны уметь вычислять этот интервал. Из чего он складывается? Время выполнения цепочки действий - от события на объекте до генерации прерывания - никак не зависит от операционных систем реального времени и целиком определяется аппаратурой, а вот интервал времени - от возникновения запроса на прерывание и до выполнения первой инструкции обработчика определяется целиком свойствами операционной системы и архитектурой компьютера. Причем это время нужно уметь оценивать в худшей для системы ситуации, то есть в предположении, что процессор загружен, что в это время могут происходить другие прерывания, что система может выполнять какие-то действия, блокирующие прерывания. В ОСРВ заложен параллелизм, возможность одновременной обработки нескольких событий, поэтому все операционные системы реального времени являются

многозадачными (многопроцессными, многонитиевыми). Для того, чтобы уметь оценивать накладные расходы системы при обработке параллельных событий, необходимо знать время, которое система затрачивает на передачу управления от процесса к процессу (от задачи к задаче, от нити к нити), то есть **время переключения контекста**.

ОСРВ содержат механизмы, гарантирующие заранее вычисленное время реакции системы. Эта гарантия достигается знанием максимального времени блокировок прерываний в системе, времени переключения контекста, времен выполнения различных системных вызовов, применением нужных механизмов диспетчеризации и пр. Т.е. время реакции на события для операционных систем реального времени можно вычислить с большой точностью. Эти вычисления невозможны для операционных систем LINUX и Windows NT - здесь можно полагаться только на результаты тестирования, эмпирические оценки. Часто производительность компьютера подбирается таким образом, чтобы успевать в нужные времена, но сути дела это не меняет, поэтому риски в сложных комплексах могут оказаться велики - при небольшом изменении внешних условий или при модификации приложений времена могут "поплыть". Что касается расширений реального времени для LINUX и Windows NT, то здесь, по крайней мере, можно полагаться на времена, полученные при тестировании.

Время перезагрузки системы. Этот параметр кажется второстепенным, однако были свидетелями случаи, когда именно этот параметр целиком определял выбор дорогой операционной системы (время перезагрузки не должно было превышать 1 секунду). Этот параметр важен для систем, от которых требуется непрерывная работа; в этих случаях ставятся ловушки, отслеживающие зависание системы или приложений, и, если таковое произошло, автоматически перезагружающие систему (такие ловушки необходимы в системах повышенной надежности, т.к. от ошибок, по крайней мере, в приложениях никто не застрахован). В таких случаях важным является такое свойство системы как ее живучесть при незапланированных перезагрузках. Большинство операционных си-

стем реального времени устойчивы к перезагрузкам и могут быть прерваны и перезагружены в любое время.

Время загрузки для разных ОСРВ колеблется от секунды до нескольких десятков секунд. В большинстве систем (OS9, VxWorks) время загрузки можно регулировать, изменяя стартовые последовательности. В ОС LINUX время загрузки в стандартном варианте более минуты, система неустойчива к внезапным остановам - требуется стандартная процедура завершения работы с системой (shutdown). Однако LINUX достаточно гибок и можно создать конфигурации системы, в которых время загрузки будет уменьшено до десятка секунд и система будет устойчива к сбоям (использование специальной опции файловой системы). В Windows NT время загрузки более минуты, система неустойчива к внезапным сбоям. Использование расширений реального времени (RTX) позволяет детектировать зависания системы и выполнить в этих случаях необходимые операции по спасению данных и по выполнению каких-то страховочных действий.

Посмотрим, какие ОС могут использоваться в системах реального времени в зависимости от времени реакции системы.

Табл.1 Требования к реактивности системы и возможные используемые ОС

Время реакции	Используемые ОС
Менее 10 мкс	только ОСРВ. Но даже и они могут оказаться бессильны - это граница выбора между схемным и программным решением
10 - 100 мкс	операционные системы реального времени
100 мкс - 1 мс	ОСРВ, RTAI, RT LINUX, расширения реального времени для Windows NT, Windows CE
1 мс	можно пытаться что-то сделать с LINUX и Windows NT, но не для систем, где опоздания реакции могут привести к тяжелым последствиям

Вычислительные установки, на которых применяются ОСРВ, можно разделить на три группы:

- **«Обычные» компьютеры.** По логическому устройству совпадают с настольными системами. Аппаратное устройство несколько отличается. Для обеспечения минимального времени простоя в случае технической неполадки процессор, память и т.д. размещаются на съемной плате, вставляемой в специальный разъем так называемой «пассивной» основной платы. В другие разъемы этой платы вставляются платы периферийных устройств. Среди процессоров таких компьютеров преобладают процессоры Intel.
- **Промышленные компьютеры.** Состоят из одной платы, на которой размещены процессор, контроллер памяти, память. Память может быть нескольких видов – ПЗУ (в которой размещается сама ОСРВ), ОЗУ (там размещаются код и данные), Флеш-память (может играть роль диска). На плате также находятся контроллеры периферийных устройств, несколько программируемых таймеров. Среди процессоров этих компьютеров доминируют процессоры семейства PowerPC, Motorola 68xxx, SPARC, ARM, Intel 80x86, 80960x.
- **Встраиваемые системы.** Устанавливаются внутрь оборудования, которым они управляют. Для крупного оборудования могут по исполнению совпадать с промышленными компьютерами. Для оборудования поменьше могут представлять собой процессор с сопутствующими элементами, размещенными на одной плате с другими электронными компонентами оборудования. Для миниатюрных систем процессор с сопутствующими элементами может быть частью одной из интегральных схем оборудования.

В связи с особенностями оборудования ОСРВ (промышленные компьютеры и встраиваемые системы часто являются бездисковыми.) должны обладать следующими свойствами:

Размеры системы. Для систем реального времени важным параметром является размер системы исполнения, а именно суммарный размер минимально необходимого для работы приложения системного набора (ядро, системные мо-

дули, драйверы и т. д.). Хотя, надо признать, что с течением времени значение этого параметра уменьшается, тем не менее он остается важным и производители систем реального времени стремятся к тому, чтобы размеры ядра и обслуживающих модулей системы были невелики.

Примеры: размер ядра операционной системы реального времени OS-9 на микропроцессорах Motorola 68xxx - 22 KB, VxWorks - 16 KB.

Возможность исполнения системы из ПЗУ (ROM). Система должна иметь возможность осуществлять загрузку из ПЗУ. Для экономии места в ПЗУ часть системы может храниться в сжатом виде и загружаться в ОЗУ по мере необходимости. Часто система позволяет исполнять код как в ПЗУ, так и в ОЗУ. При наличии свободного места в ОЗУ система может копировать себя из медленного ПЗУ в более быстрое ОЗУ.

К дополнительным свойствам ОСРВ можно отнести следующие:

Наличие необходимых драйверов устройств. Если разрабатываемая система имеет обширную периферию, то наличие уже готовых драйверов может оказать большое влияние на выбор операционной системы. Естественно, самый большой набор драйверов создан для операционных системах LINUX и Windows NT. Наиболее популярные операционные системы реального времени, такие как VxWorks, OS9, QNX, также имеют обширные наборы готовых драйверов и, кроме того, содержат средства для их быстрой разработки.

Поддержка процессоров различной архитектуры. В связи с тем, что в промышленных компьютерах, серверах, встраиваемых системах широко распространены процессоры разной архитектуры с различной системой команд, ОСРВ по возможности должна поддерживать как можно более широкий ряд процессоров.

Одной из важных характеристик ОСРВ является наличие **специального кроссплатформенного инструментария разработчика.** Это связано с тем, что разработка СРВ часто проводится на «обычном» компьютере, отличном по архитектуре от компьютера, на котором будет устанавливаться СРВ. При этом ОС на этих двух компьютерах также может не совпадать.

Важнейшими характеристиками ОСРВ являются заложенные в операционную систему **механизмы реального времени**.

Процесс проектирования конкретной системы реального времени начинается с тщательного изучения объекта. Разработчики проекта исследуют объект, изучают возможные события на нем, определяют критические сроки реакции системы на каждое событие и разрабатывают алгоритмы обработки этих событий. Затем следует процесс проектирования и разработки программных приложений. Какие же механизмы в операционных системах реального времени делают систему реального времени (СРВ) предсказуемой?

Система приоритетов и алгоритмы диспетчеризации. Базовыми инструментами разработки сценария работы системы являются система приоритетов процессов (задач) и алгоритмы планирования (диспетчеризации) операционных системах реального времени.

В многозадачных ОС общего назначения используются, как правило, различные модификации алгоритма круговой диспетчеризации, основанные на понятии непрерывного кванта времени ("time slice"), предоставляемого процессу для работы. Планировщик по истечении каждого кванта времени просматривает очередь активных процессов и принимает решение, кому передать управление, основываясь на приоритетах процессов (численных значениях, им присвоенных). Приоритеты могут быть фиксированными или меняться со временем - это зависит от алгоритмов планирования в данной ОС, но рано или поздно процессорное время получают все процессы в системе.

Алгоритмы круговой диспетчеризации неприменимы в чистом виде в операционных системах реального времени. Основной недостаток - непрерывный квант времени, в течение которого процессором владеет только один процесс. Планировщики же операционных систем реального времени имеют возможность сменить процесс до истечения "time slice", если в этом возникла необходимость. Один из возможных алгоритмов планирования при этом "приоритетный с вытеснением". Мир операционных систем реального времени отличается богатством различных алгоритмов планирования: динамические, приоритет-

ные, монотонные, адаптивные и пр., цель же всегда преследуется одна - предоставить инструмент, позволяющий в нужный момент времени исполнять именно тот процесс, который необходим.

Механизмы межзадачного взаимодействия. Другой набор механизмов реального времени относится к средствам синхронизации процессов и передачи данных между ними. Для операционных систем реального времени характерна развитость этих механизмов. К таким механизмам относятся: семафоры, мьютексы, события, сигналы, средства для работы с разделяемой памятью, каналы данных (pipes), очереди сообщений. Многие из подобных механизмов используются и в ОС общего назначения, но их реализация в операционных системах реального времени имеет свои особенности - время исполнения системных вызовов почти не зависит от состояния системы, и в каждой операционной системе реального времени есть по крайней мере один быстрый механизм передачи данных от процесса к процессу.

Средства для работы с таймерами. Такие инструменты, как средства работы с таймерами, необходимы для систем с жестким временным регламентом, поэтому развитость средств работы с таймерами - необходимый атрибут операционных систем реального времени. Эти средства, как правило, позволяют:

- измерять и задавать различные промежутки времени (от 1 мкс и выше),
- генерировать прерывания по истечении временных интервалов,
- создавать разовые и циклические будильники

Здесь описаны только базовые, обязательные механизмы, использующиеся в ОСРВ. Кроме того, почти в каждой операционной системе реального времени имеется целый набор дополнительных, специфических только для нее механизмов, касающийся системы ввода-вывода, управления прерываниями, работы с памятью. Каждая система содержит также ряд средств, обеспечивающих ее надежность. Позже механизмы реального времени будут рассмотрены подробнее.

Тема 3 (4 часа)

Архитектура ОСРВ. Классы ОСРВ.

Все ОСРВ сегодня являются многозадачными системами. Задачи делят между собой ресурсы вычислительной системы, в том числе и процессорное время.

Четкой границы между ядром (KERNEL) и операционной системой нет. Различают их, как правило, по набору функциональных возможностей. *Ядра предоставляют пользователю такие базовые функции, как планирование синхронизация задач, межзадачная коммуникация, управление памятью, устройствами ввода-вывода и т. п.* Операционные системы в дополнение к этому имеют файловую систему, сетевую поддержку, интерфейс с оператором и другие средства высокого уровня. Они обеспечивают также взаимодействие системы и управляющего/управляемого оборудования.

По своей внутренней архитектуре ОСРВ можно условно разделить на:

- 1) монолитные ОС,
- 2) ОС на основе микроядра,
- 3) объектно-ориентированные ОС.

Графически различия в этих подходах иллюстрируются рисунками 1,2,3.



Рис 1. ОСРВ с монолитной архитектурой

ОСРВ с монолитной архитектурой можно представить в виде

- Прикладного уровня: состоит из работающих прикладных процессов;
- Системного уровня: состоит из монолитного ядра операционной системы, в котором можно выделить следующие части:

- интерфейс между приложениями и ядром (API)
- собственно ядро системы
- интерфейс между ядром и оборудованием (драйверы устройств).

API в таких системах играет двойную роль:

1. управление взаимодействием прикладных процессов и системы;
2. обеспечение непрерывности выполнения кода системы (т.е. отсутствие переключения задач во время исполнения кода системы).

Основным преимуществом монолитной архитектуры является относительная быстрота работы по сравнению с другими архитектурами. Однако, достигается это, в основном, за счет написания значительных частей системы на ассемблере.

Недостатки монолитной архитектуры:

1. Системные вызовы, требующие переключения уровней привилегий (от пользовательской задачи к ядру), должны реализовывать API как прерывания или ловушки (специальный тип исключений). Это сильно увеличивает время их работы.
2. Ядро не может быть прервано пользовательской задачей (non-preemptible). Это может приводить к тому, что высокоприоритетная задача может не получить управление из-за работы низкоприоритетной.

Например, низкоприоритетная задача запросила выделение памяти, сделала системный вызов, до окончания которого сигнал активизации высокоприоритетной задачи не сможет ее активизировать.

3. Сложность переноса на новые архитектуры процессора из-за значительных ассемблерных вставок.
4. Негибкость и сложность развития: изменение части ядра системы требует его полной перекомпиляции.

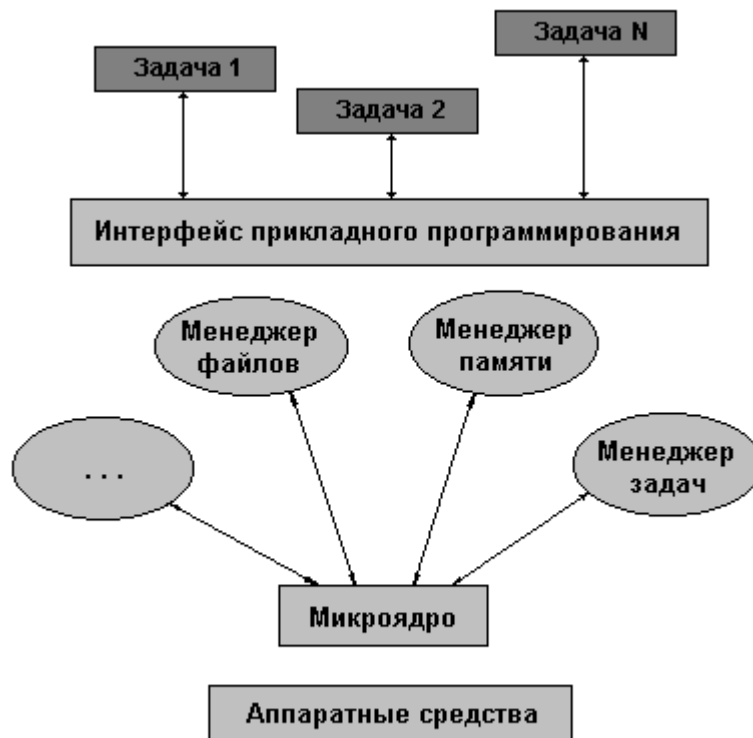


Рис 2. ОСРВ на основе микроядра

Модульная архитектура (на основе микроядра) появилась как попытка убрать узкое место – API и облегчить модернизацию системы и перенос ее на новые процессоры.

API в модульной архитектуре играет только одну роль: обеспечивает связь прикладных процессов и специального модуля – менеджера процессов. Однако, теперь микроядро играет двойную роль:

1. управление взаимодействием частей системы (например, менеджеров процессов и файлов)
2. обеспечение непрерывности выполнения кода системы (т.е. отсутствие переключения задач во время исполнения микроядра).

Недостатки у модульной архитектуры фактически те же, что и у монолитной. Проблемы перешли с уровня API на уровень микроядра. Системный интерфейс по-прежнему не допускает переключения задач во время работы микроядра, только сократилось время пребывания в этом состоянии. API по-прежнему может быть реализован только на ассемблере, проблемы с переносимо-

стью микроядра уменьшились (в связи с сокращением его размера), но остались.

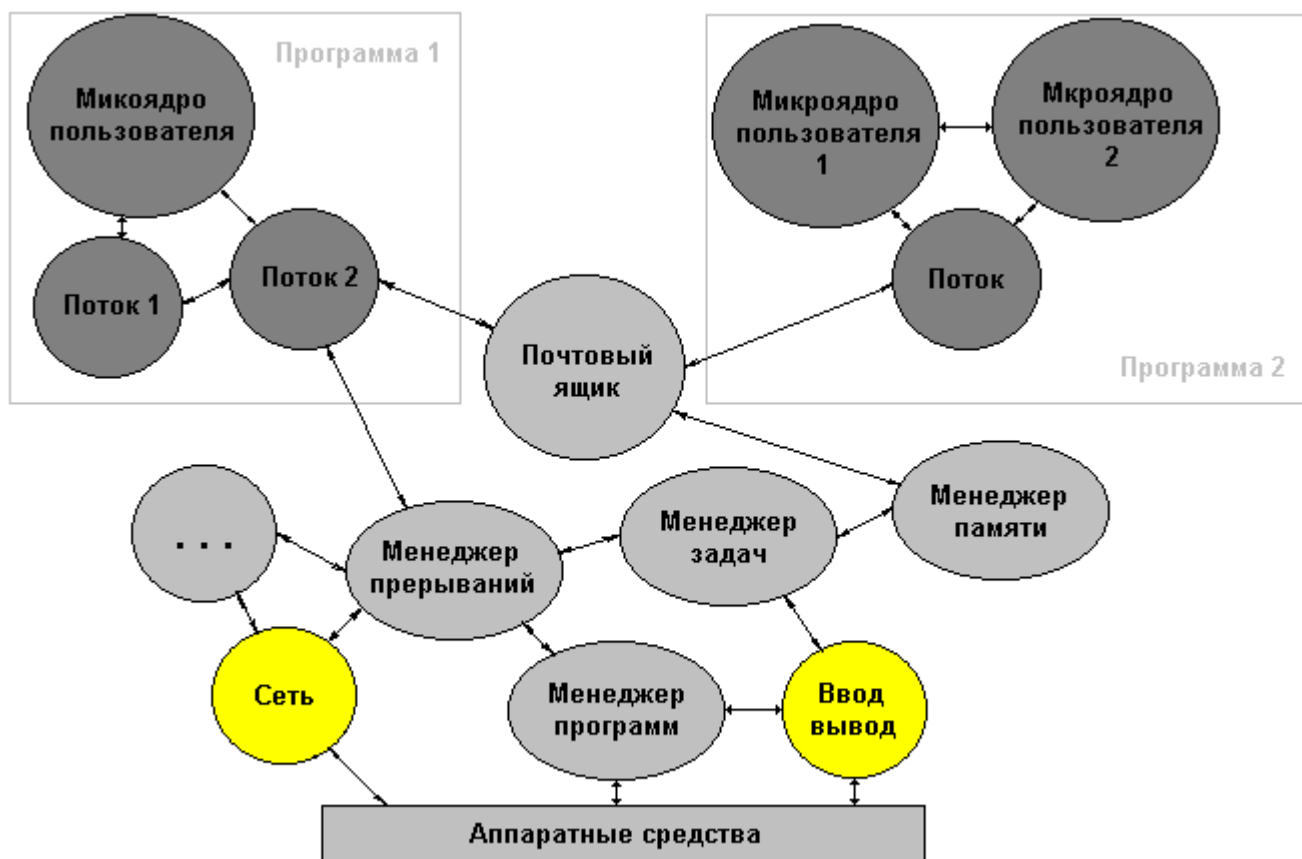


Рис 3. Объектно-ориентированная ОСРВ

Упомянутые выше архитектуры относятся к так называемой классической архитектуре ОСРВ и используют традиционный процедурный подход к программированию. Однако в силу преимуществ объектно-ориентированного подхода приложения создаются на его основе. Сочетание объектно-ориентированных приложений и процедурных операционных систем имеет ряд недостатков:

- В едином работающем комплексе (приложение + ОСРВ) разные компоненты используют разные подходы к разработке программного обеспечения.
- Не используются все возможности объектно-ориентированного подхода.
- Возникают некоторые потери производительности из-за разного типа интерфейсов в ОСРВ и приложении.

Уместенно, возникает идея строить саму ОСРВ, используя объектно-ориентированный подход.

Объектная архитектура на основе объектов-микроядер. В этой архитектуре API отсутствует вообще. Взаимодействие между компонентами системы (микроядрами) и пользовательскими процессами осуществляется посредством обычного вызова функций, поскольку и система, и приложения написаны на одном языке (Для ОСРВ SoftKernel это C++). Это обеспечивает максимальную скорость системных вызовов.

Фактическое равноправие всех компонент системы обеспечивает возможность переключения задач в любое время, т.е. система полностью preemptible.

Объектно-ориентированный подход обеспечивает модульность, безопасность, легкость модернизации и повторного использования кода.

Роль API играет компилятор и динамический редактор объектных связей (linker). При старте приложения динамический linker загружает нужные ему микроядра (т.е., в отличие от предыдущих систем, не все компоненты самой операционной системы должны быть загружены в оперативную память). Если микроядро уже загружено для другого приложения, оно повторно не загружается, а использует код и данные уже имеющегося ядра. Это позволяет уменьшить объем требуемой памяти.

Поскольку разные приложения разделяют одни микроядра, то они должны работать в одном адресном пространстве. Следовательно, система не может использовать виртуальную память и тем самым работает быстрее (так как исключаются задержки на трансляцию виртуального адреса в физический).

Поскольку все приложения и сами микроядра работают в одном адресном пространстве, то они загружаются в память, начиная с неизвестного на момент компиляции адреса. Следовательно, приложения и микроядра не должны зависеть от начального адреса (как по коду, так и по данным (последнее обеспечить значительно сложнее)). Это свойство автоматически обеспечивает возможность записи приложений и модулей в ПЗУ, с их последующим исполнением как в самом ПЗУ, так и оперативной памяти.

Микроядра по своим характеристикам напоминают структуры, используемые в других операционных системах. Однако есть и свои различия.

Взаимодействие микроядер и других структур ОС

- **Микроядра и модули.** Многие ОС поддерживают динамическую загрузку компонент системы, называемых модулями. Однако, модули не поддерживают объектно-ориентированный подход (напомним, микроядро фактически является представителем некоторого класса). Далее, обмен информацией с модулями происходит посредством системных вызовов, что достаточно дорого.
- **Микроядра и драйверы.** Многие ОС поддерживают возможность своего расширения посредством драйверов (специальных модулей, обычно служащих для поддержки оборудования). Однако, драйверы часто должны быть статически связаны с ядром (т.е. образовывать с ним связанный загрузочный образ еще до загрузки) и должны работать в привилегированном (суперпользовательском) режиме. Далее, как модули они не поддерживают объектно-ориентированный подход и доступны приложению только посредством системных вызовов.
- **Микроядра и DLL.** Многие системы оформляют библиотеки, из которых берутся функции при динамическом связывании, в виде специальных модулей, называемых DLL (Dynamically Linked Libraries – динамически связываемые библиотеки). DLL обеспечивает разделение своего кода и данных для всех работающих приложений, в то время как для микроядер можно управлять доступом для каждого конкретного приложения. DLL не поддерживает объектно-ориентированный подход, код DLL не является позиционно-независимым и не может быть записан в ПЗУ.

Системы реального времени можно разделить на 4 класса.

1-й класс: исполнительные СРВ.

Это программы для программируемых микропроцессоров, встраиваемых в различные устройства, очень небольшие и обычно написаны на языке низкого

уровня типа ассемблера или PLM. Внутрисхемные эмуляторы пригодны для отладки, но высокоуровневые средства разработки и отладки программ не применимы. Операционная среда обычно недоступна.

Признаки систем этого типа - различные платформы для систем разработки и исполнения. Приложение реального времени разрабатывается на host-компьютере (компьютере системы разработки), затем компонуется с ядром и загружается в целевую систему для исполнения. Как правило, приложение реального времени - это одна задача и параллелизм здесь достигается с помощью нитей (threads).

Системы этого типа обладают рядом достоинств, среди которых главное - скорость и реактивность системы. Главная причина высокой реактивности систем этого типа - наличие только нитей(поток) и, следовательно, маленькое время переключения контекста между ними (в отличие от процессов).

С этим главным достоинством связан и ряд недостатков: зависание всей системы при зависании нити, проблемы с динамической подгрузкой новых приложений

Наиболее ярким представителем систем этого класса является операционная система VxWorks. Область применения - компактные системы реального времени с хорошими временами реакций.

2-й класс: минимальное ядро системы реального времени.

На более высоком уровне находятся системы реального времени, обеспечивающие минимальную среду исполнения. Предусмотрены лишь основные функции, а управление памятью и диспетчер часто недоступны. Ядро представляет собой набор программ, выполняющих типичные, необходимые для встроенных систем низкого уровня функции, такие, как операции с плавающей запятой и минимальный сервис ввода/вывода. Прикладная программа разрабатывается в инструментальной среде, а выполняется, как правило, на встроенных системах.

В этот класс входят системы с монолитным ядром, где и содержится реализация всех механизмов реального времени этих операционных систем.

Системы этого класса, как правило, модульны, хорошо структурированы, имеют наиболее развитый набор специфических механизмов реального времени, компактны и предсказуемы. Наиболее популярные системы этого класса: OS9, QNX.

Одна из особенностей систем этого класса - высокая степень масштабируемости. На базе этих ОС можно построить как компактные системы реального времени, так и большие системы серверного класса.

3-й класс: ядро системы реального времени и инструментальная среда.

Этот класс систем обладает многими чертами ОС с полным сервисом. Разработка ведется в инструментальной среде, а исполнение - на целевых системах. Этот тип систем обеспечивает гораздо более высокий уровень сервиса для разработчика прикладной программы. Сюда включены такие средства, как дистанционный символьный отладчик, протокол ошибок и другие средства CASE. Часто доступно параллельное выполнение программ.

4-й класс: ОС с полным сервисом. Такие ОС могут быть применены для любых приложений реального времени. Разработка и исполнение прикладных программ ведутся в рамках одной и той же системы.

Область применения расширений реального времени - большие системы реального времени, где требуется визуализация, работа с базами данных, доступ в интернет и пр.

К таким системам можно отнести ОСРВ, написанные на основе UNIX и расширения Windows.

Расширения реального времени для WindowsNT

Появление в свое время UNIX'ов реального времени означало ни что иное, как попытку применить господствующую программную технологию для создания приложений реального времени. Появление расширений реального времени для Windows NT имеет те же корни, ту же мотивацию. Огромный набор прикладных программ под Windows, мощный программный интерфейс WIN32, большое количество специалистов, знающих эту систему. Конечно, соблазнительно получить в системе реального времени все эти возможности.

Несмотря на то, что Windows NT создавалась как сетевая операционная система, в нее при создании были заложены элементы реального времени, а именно - двухуровневая система обработки прерываний (ISR и DPC), классы реального времени (процессы с приоритетами 16-32 планируются в соответствии с правилами реального времени).

Конечно, даже поверхностный анализ Windows NT показывает, что эта система не годится для построения систем жесткого реального времени (система непредсказуема - время выполнения системных вызовов и время реакции на прерывания сильно зависит от загрузки системы; система велика; нет механизмов защиты от зависаний и пр. и пр.). Поэтому даже в системах мягкого реального времени Windows NT может быть использована только при выполнении целого ряда рекомендаций и ограничений.

Разработчики расширений пошли двумя путями:

1. Использовали ядра классических операционных систем реального времени в качестве дополнения к ядру Windows NT ("два в одном флаконе"). Таковы решения фирм "LP Elektronik" и "Radisyс". В первом случае параллельно с Windows NT (на одном компьютере!) работает операционная система VxWorks, во-втором случае - InTime. Кроме того, предоставляется набор функций для связи приложений реального времени и приложений Windows NT. Вот как, например это выглядит у LP Elektronik: вначале стандартным образом загружается Windows NT, затем с помощью специального загрузчика загружается операционная система VxWorks, распределяя под себя необходимую память Windows (что в дальнейшем позволяет избежать конфликтов памяти между двумя ОС). После этого главной в системе становится VxWorks, отдавая процессор ядру Windows NT только в случаях, когда в нем нет надобности для приложений VxWorks. В качестве канала для синхронизации и обмена данными между Windows NT и VxWorks служат псевдодрайверы TCP/IP в обеих системах. Технология использования двух систем на одном компьютере понятна - работу с объектом выполняет приложение реального времени, передавая затем

результаты приложениям Windows NT для обработки, передачи в сеть, архивирования и пр.

2. Вариант расширений реального времени фирмы VenturCom выглядит иначе: была сделана попытка "интегрировать" реальное время в Windows NT путем исследования причин задержек и зависаний и устранения этих причин с помощью подсистемы реального времени. Решения фирмы "VenturCom" (RTX 4.2) базируются на модификациях уровня аппаратных абстракций Windows NT (HAL - Hardware Abstraction Layer) - программного слоя, через который драйверы взаимодействуют с аппаратурой. Модифицированный HAL и дополнительные функции (RTAPI) отвечают также за стабильность и надежность системы, обеспечивая отслеживание зависания Windows NT, зависания приложений или блокировку прерываний. В состав RTX входит также подсистема реального времени RTSS, с помощью которой Windows NT расширяется дополнительным набором объектов (аналогичным стандартным, но с атрибутами реального времени). Среди новых объектов - нити (потoki, процессы) реального времени, которые управляются специальным планировщиком реального времени (256 фиксированных приоритетов, алгоритм - приоритетный с вытеснением). Побочным результатом RTX является возможность простого создания программ управления устройствами, т.к. среди функций RTAPI есть и функции работы с портами ввода-вывода и физической памятью. Предложения VenturCom характерны еще и тем, что они предоставляют совершенно экзотическую для NT возможность, а именно - возможность конфигурирования Windows NT и создания встроенных конфигураций (без дисков, клавиатуры и монитора) (Интегратор Компонентов - CI).

Тема 4 (4 часа)

Функции ядра ОСРВ. Процессы. Состояния процесса. Жизненный цикл процесса. Потoki. Приоритеты процессов.

1. Функции ядра ОСРВ

Как было указано ранее, основой любой среды исполнения в реальном времени является ядро. Все современные ОСРВ являются многозадачными. Таким образом, все программное обеспечение, включая также часть операционной системы, организуется в виде набора последовательных процессов. Исходя из этого, ядро может обеспечивать сервис пяти типов:

1. Синхронизация ресурсов. Метод синхронизации требует ограничить доступ к общим ресурсам (данным и внешним устройствам). Наиболее распространенный тип примитивной синхронизации - двоичный семафор, обеспечивающий избирательный доступ к общим ресурсам. Так, процесс, требующий защищенного семафором ресурса, вынужден ожидать до тех пор, пока семафор не станет доступным, что свидетельствует об освобождении ожидаемого ресурса, и, захватив ресурс, установить семафор. В свою очередь, другие процессы также будут ожидать доступа к ресурсу вплоть до того момента, когда семафор возвратит соответствующий ресурс системе распределения ресурсов. Системы, обладающие большей ошибкоустойчивостью, могут иметь счетный семафор. Этот вид семафора разрешает одновременный доступ к ресурсу лишь определенному количеству процессов.

2. Межзадачный обмен. Часто необходимо обеспечить передачу данных между программами внутри одной и той же системы. Кроме того, во многих приложениях возникает необходимость взаимодействия с другими системами через сеть. Внутренняя связь может быть осуществлена через систему передачи сообщений. Внешнюю связь можно организовать либо через датаграмму (наилучший способ доставки) (протокольная единица обмена (PDU) сетевого уровня. Представляет собой пакет данных, содержащий свой адрес доставки и передаваемый через сеть с коммутацией пакетов независимо от других пакетов без разрыва логического соединения и квитирования. Содержит ссылку на предыдущие пакеты, адресованные тому же самому получателю. Дейтаграммный способ передачи, основанный на анализе адреса получателя, в Интернете реализуется с помощью протокола IP), либо по линиям связи (гарантированная доставка). Выбор того или иного способа зависит от протокола связи.

3. Разделение данных. В прикладных программах, работающих в реальном времени, наиболее длительным является сбор данных. Данные часто необходимы для работы других программ или нужны системе для выполнения каких-либо своих функций. Во многих системах предусмотрен доступ к общим разделам памяти. Широко распространена организация очереди данных. Применяется много типов очередей, каждый из которых обладает собственными достоинствами.

4. Обработка запросов внешних устройств. Каждая прикладная программа в реальном времени связана с внешним устройством определенного типа. Ядро должно обеспечивать службы ввода/вывода, позволяющие прикладным программам осуществлять чтение с этих устройств и запись на них. Для приложений реального времени обычным является наличие специфического для данного приложения внешнего устройства. Ядро должно предоставлять сервис, облегчающий работу с драйверами устройств. Например, давать возможность записи на языках высокого уровня - таких, как Си или Паскаль.

5. Обработка особых ситуаций. Особая ситуация представляет собой событие, возникающее во время выполнения программы. Она может быть синхронной, если ее возникновение предсказуемо, как, например, деление на нуль. А может быть и асинхронной, если возникает непредсказуемо, как, например, падение напряжения. Предоставление возможности обрабатывать события такого типа позволяет прикладным программам реального времени быстро и предсказуемо отвечать на внутренние и внешние события. Существуют два метода обработки особых ситуаций - использование значений состояния для обнаружения ошибочных условий и использование обработчика особых ситуаций для прерывания ошибочных условий и их корректировки.

Кроме того, важнейшей функцией ядра является **диспетчеризация** (планирование). Планировщик должен определять, какому процессу должно быть передано управление, а также должен определить время, выделяемое каждому процессу.

2. Процессы.

Рассмотрим подробнее, что такое процесс. **Процесс** – это динамическая сущность программы, ее код в процессе своего выполнения. Имеет:

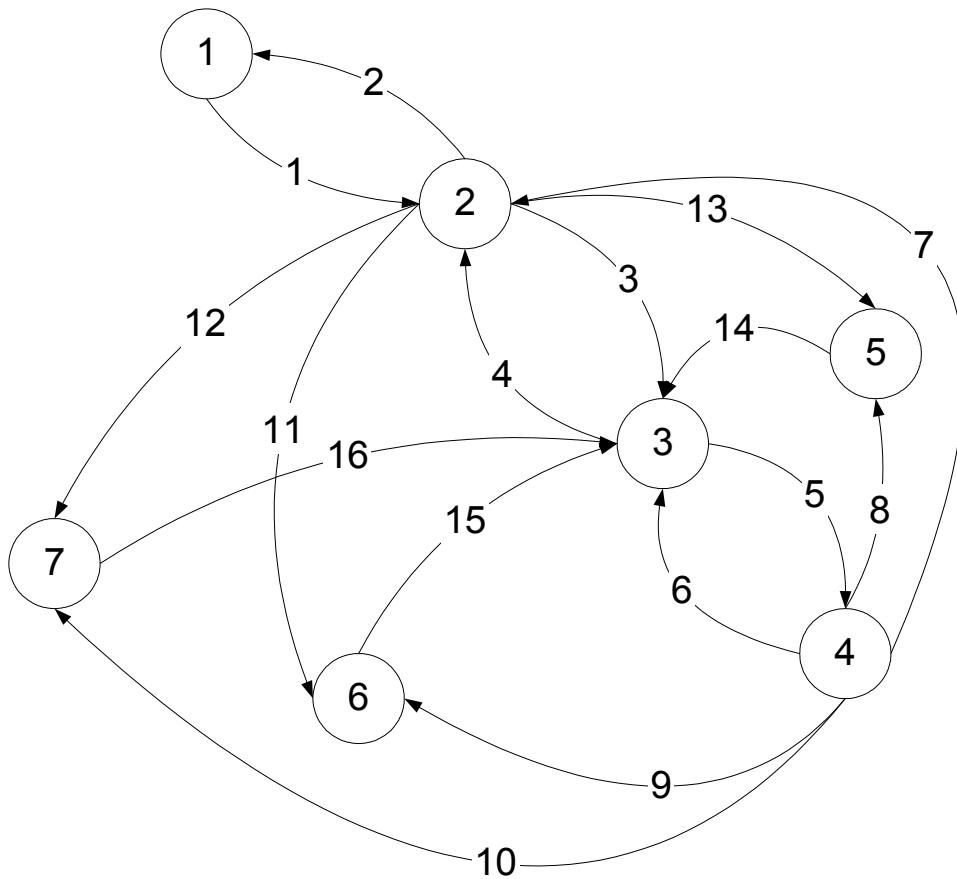
- собственные области памяти под код и данные, включая значения регистров и счетчика команд
- собственный стек
- собственное отображение виртуальной памяти (в системах с виртуальной памятью) на физическую
- собственное состояние.

3. Состояния процесса.

Процесс может находиться в одном из следующих типичных состояний:

- «остановлен» - процесс остановлен и не использует процессор (например, в таком состоянии процесс находится сразу после создания)
- «терминирован» - процесс терминирован и не использует процессор (например, процесс закончился, но еще не удален операционной системой)
- «ожидает» - процесс ждет некоторого события (им может быть аппаратное или программное прерывание, сигнал или другая форма межпроцессного взаимодействия)
- «готов» - процесс не остановлен, не терминирован, не ожидает, не удален, но и не работает (например, процесс не может получить доступ к процессору, если в данный момент выполняется другой, более высокоприоритетный процесс)
- «выполняется» - процесс выполняется и использует процессор. В ОСРВ это обычно означает, что этот процесс является самым приоритетным среди всех процессов, находящихся в состоянии «готов»

Рассмотрим более подробно состояния процесса и переходы из одного состояния в другое.



Состояния:

1. не существует
2. не обслуживается
3. готов
4. выполняется
5. ожидает ресурс
6. ожидает назначенное время
7. ожидает события

Переходы:

1. переход 1-2 создание процесса
2. переход 2-1 уничтожение процесса
3. переход 2-3 активизация процесса диспетчером
4. переход 3-2 деактивизация процесса
5. переход 3-4 загрузка на выполнение процесса диспетчером
6. переход 4-3 требование обслуживания от процессора другим процессом (preemption – приоритетное переключение)

7. переход 4-2 завершение процесса
8. переход 4-5 блокировка процесса до освобождения требуемого ресурса
9. переход 4-6 блокировка процесса до истечения заданного времени
10. переход 4-7 блокировка процесса до прихода события
11. переход 2-6 активизация процесса приводит к ожиданию временной задержки
12. переход 2-7 активизация процесса приводит к ожиданию события
13. переход 2-5 активизация процесса приводит к ожиданию освобождения ресурса
14. переход 5-3 активизация процесса из-за освобождения ожидавшегося ресурса
15. переход 6-3 активизация процесса по истечении заданного времени
16. переход 7-3 активизация процесса из-за прихода ожидавшегося события

4. Жизненный цикл процесса.

Таким образом, каждый процесс имеет свой жизненный цикл, состоящий из 4 стадий:

1. создание
2. загрузка
3. выполнение
4. завершение.

Создание процесса обычно состоит из присвоения новому процессу идентификатора процесса и подготовки информации, которая определяет окружение процесса.

Загрузка процесса означает загрузку в память кода процесса.

После того, как код программы загружен, процесс готов к выполнению. Он начинает конкурировать с другими процессами за ресурсы процессора. Процесс может выполняться, а может блокироваться по тем или иным причинам.

Завершение процесса означает освобождение всех ресурсов, выделенных процессу – файловых дескрипторов, памяти и т.д.

5. Потоки.

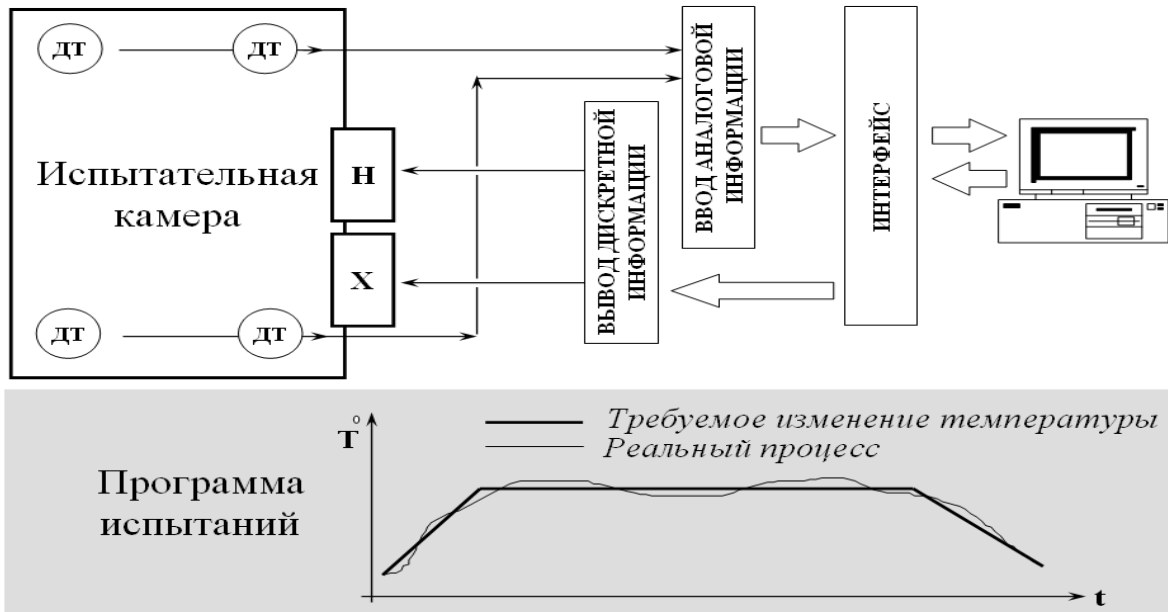
Модель потока базируется на двух независимых концепциях: группировании ресурсов и выполнении программы. С одной стороны, процесс можно рассматривать как способ группирования родственных ресурсов в одну группу. У процесса есть адресное пространство, содержащее текст программы и данные, а также другие ресурсы. Ресурсами могут быть открытые файлы, обработчики сигналов, учетная информация и многое другое. С другой стороны, процесс можно рассматривать как поток исполняемых команд, или просто **поток**. У потока есть счетчик команд, отслеживающий порядок выполнения действий. У него есть регистры, в которых хранятся текущие переменные. У него есть стек, содержащий протокол выполнения процесса, где на каждую процедуру, вызванную, но еще не вернувшуюся, отведен отдельный фрейм. У процесса есть свое состояние. Потоки одного процесса выполняются в одном адресном пространстве, используют одни и те же глобальные переменные, ресурсы.. Таким образом, процессы используются для группирования ресурсов, а потоки являются объектами, поочередно выполняющимися на процессоре.

6. Приоритеты процессов.

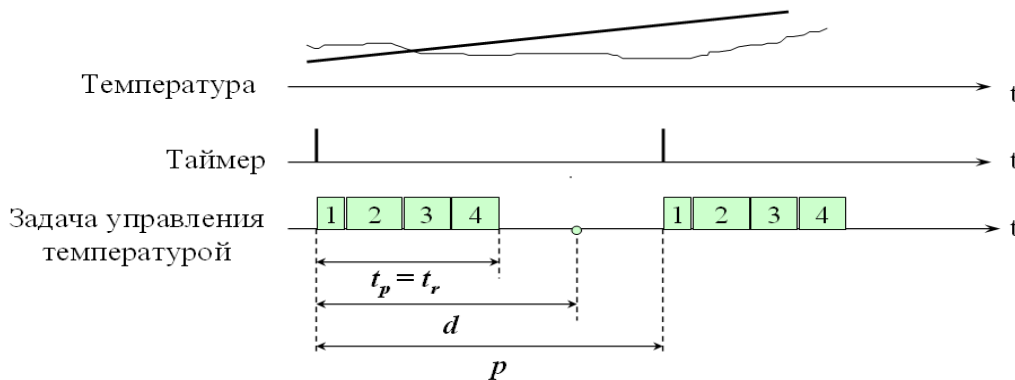
Каждому процессу и каждому потоку в ОСРВ приписывается некоторое число, называемое **приоритетом**. Чем больше это число, тем важнее процесс или поток. Приоритет может быть **фиксированным** (назначается при создании процесса и не меняется в течение его жизни). В зависимости от алгоритмов планирования приоритет также может изменяться в течение жизни.

Тема №5 (4 часа)

1.1.1. Задача управления температурой



Организация вычислительного процесса



- 1 - измерение;
- 2 - вычисление программного значения;
- 3 - алгоритм принятия решения;
- 4 - вывод управляющего воздействия;

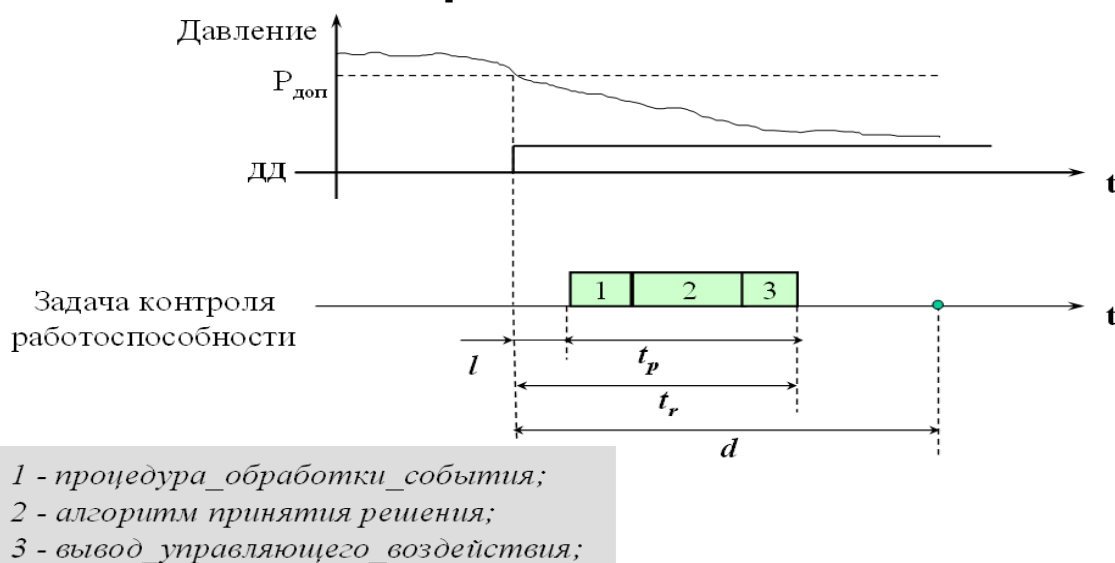
Характеристики вычислительного процесса

- t_p - время выполнения (execution time, processing time)
- t_r - время реакции системы (response time; в данном случае $t_p = t_r$)
- d - предельно допустимое время завершения (deadline)
- p - период активизации (period)
- **wcet** - время выполнения в наихудшем случае (worst case execution time)

1.1.2. Задача контроля работоспособности испытательного оборудования



Организация вычислительного процесса



1. Особенности систем реального времени. 2005 v.0.1

8



Характеристики вычислительного процесса

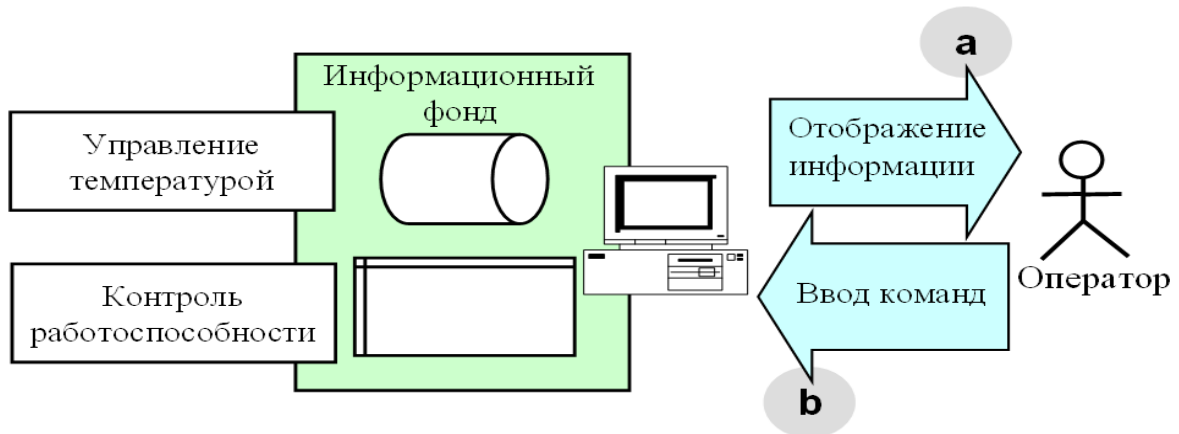
- **l** - задержка выполнения процедуры обработки события (latency)
- **t_p** - время выполнения (execution time, processing time)
- **t_r** - время реакции системы (response time $t_p + l$)
- **d** - предельно допустимое время завершения (deadline)
- **wcet** - время выполнения в наихудшем случае (worst case execution time)

1. Особенности систем реального времени. 2005 v.0.1

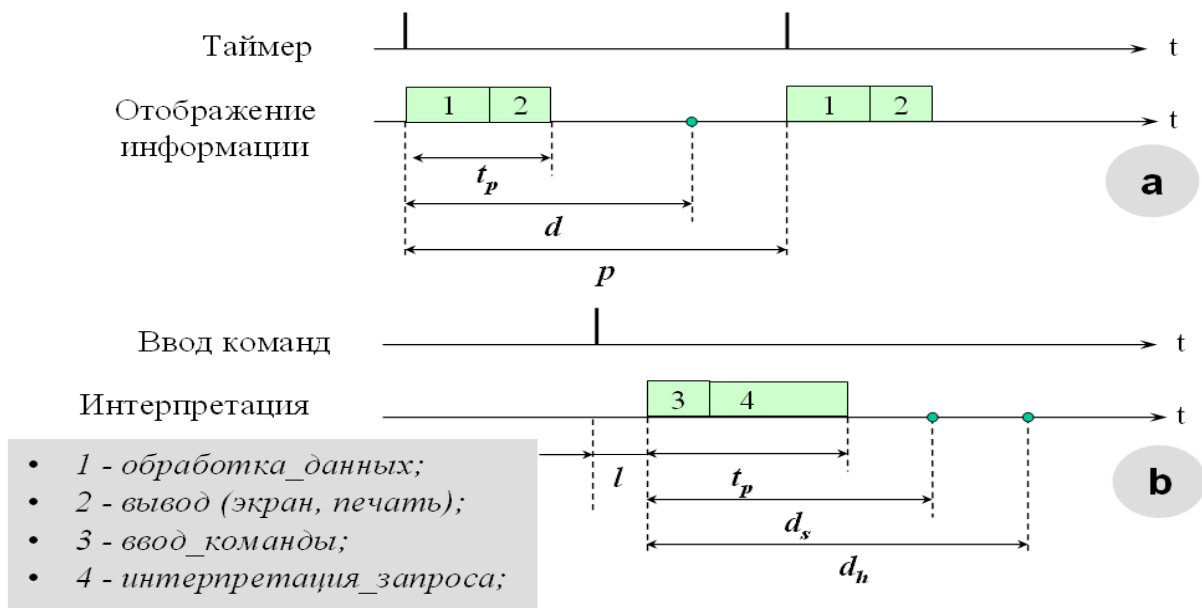
9



1.1.3. Задача диспетчеризации

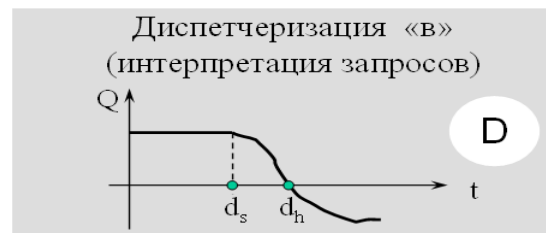
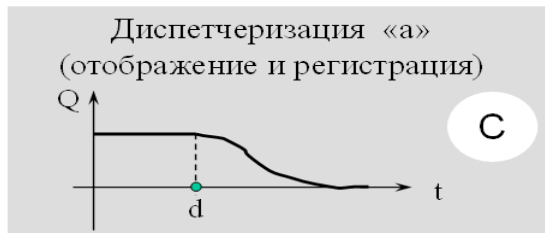
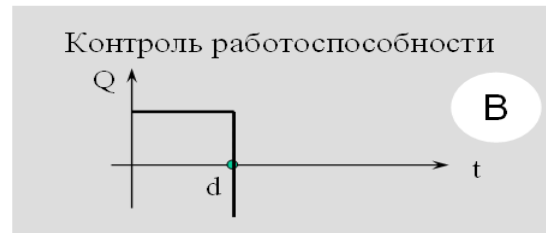
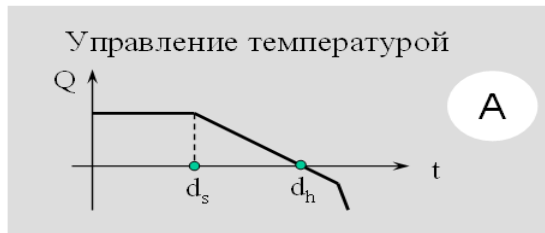


Организация вычислительного процесса



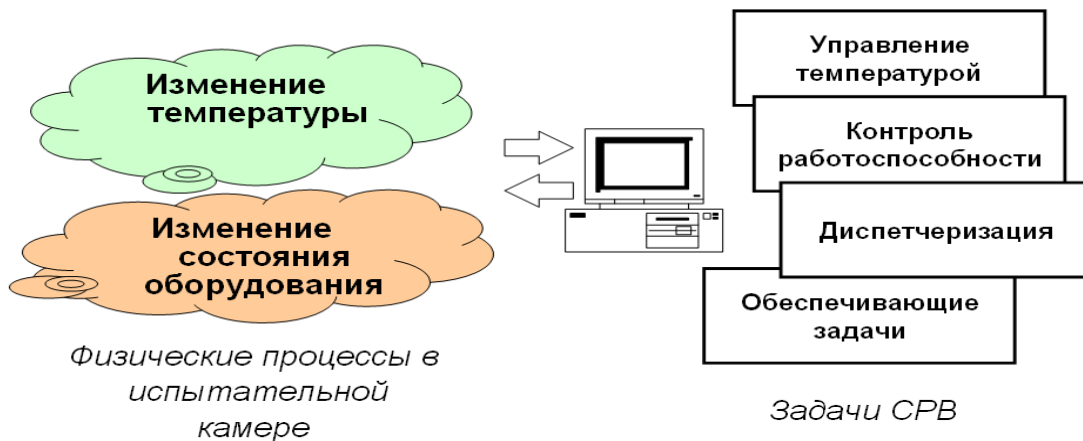
1.2. Требования к времени реакции

Q – «Значимость» результатов работы задачи



1.3. Многозадачность СРВ

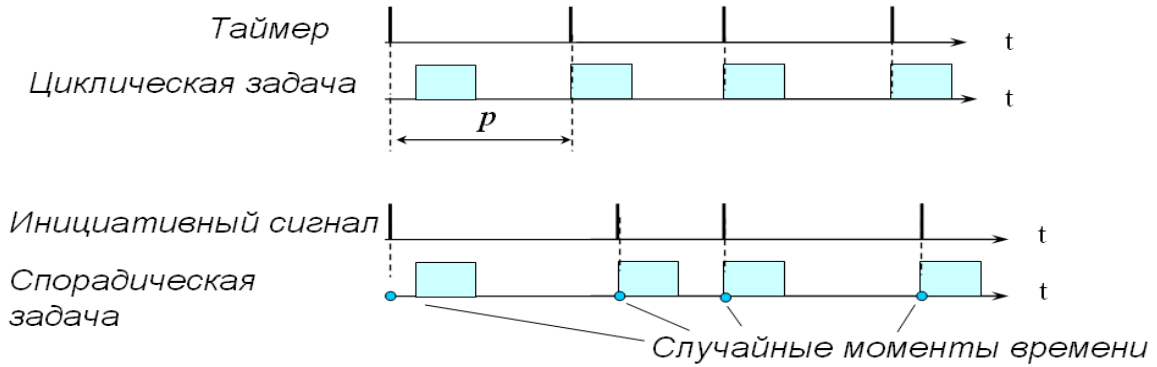
СРВ необходимо одновременно решать несколько задач, обслуживающих процессы внешней среды



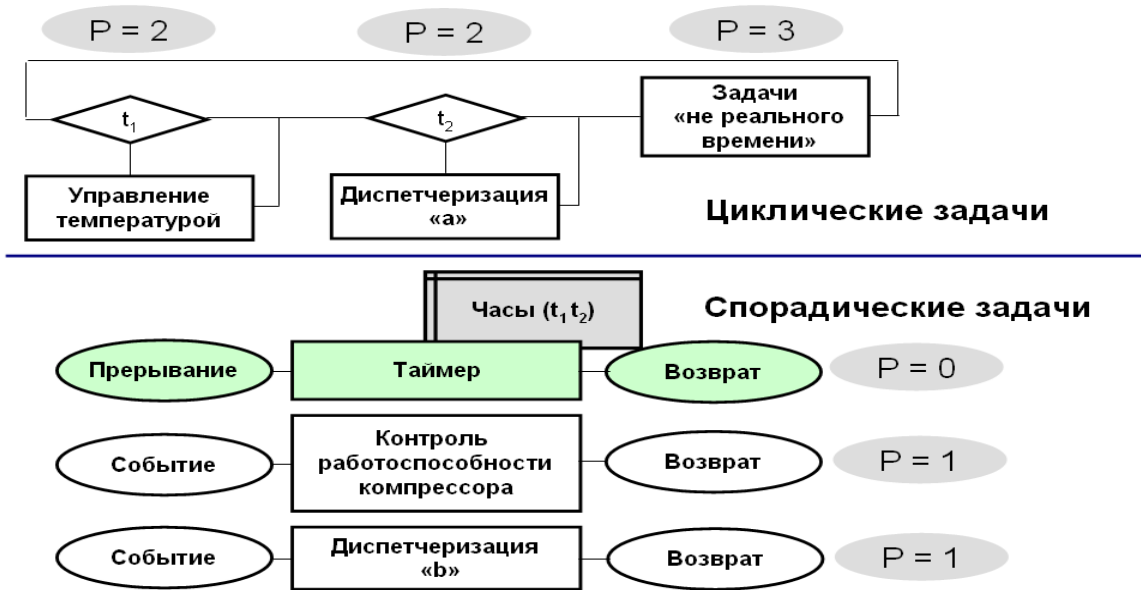
Циклические и спорадические задачи

Задачи различают

- по степени важности, присваивая им различные приоритеты
- по способу их активизации во времени – периодические (циклические) и спорадические (асинхронные)

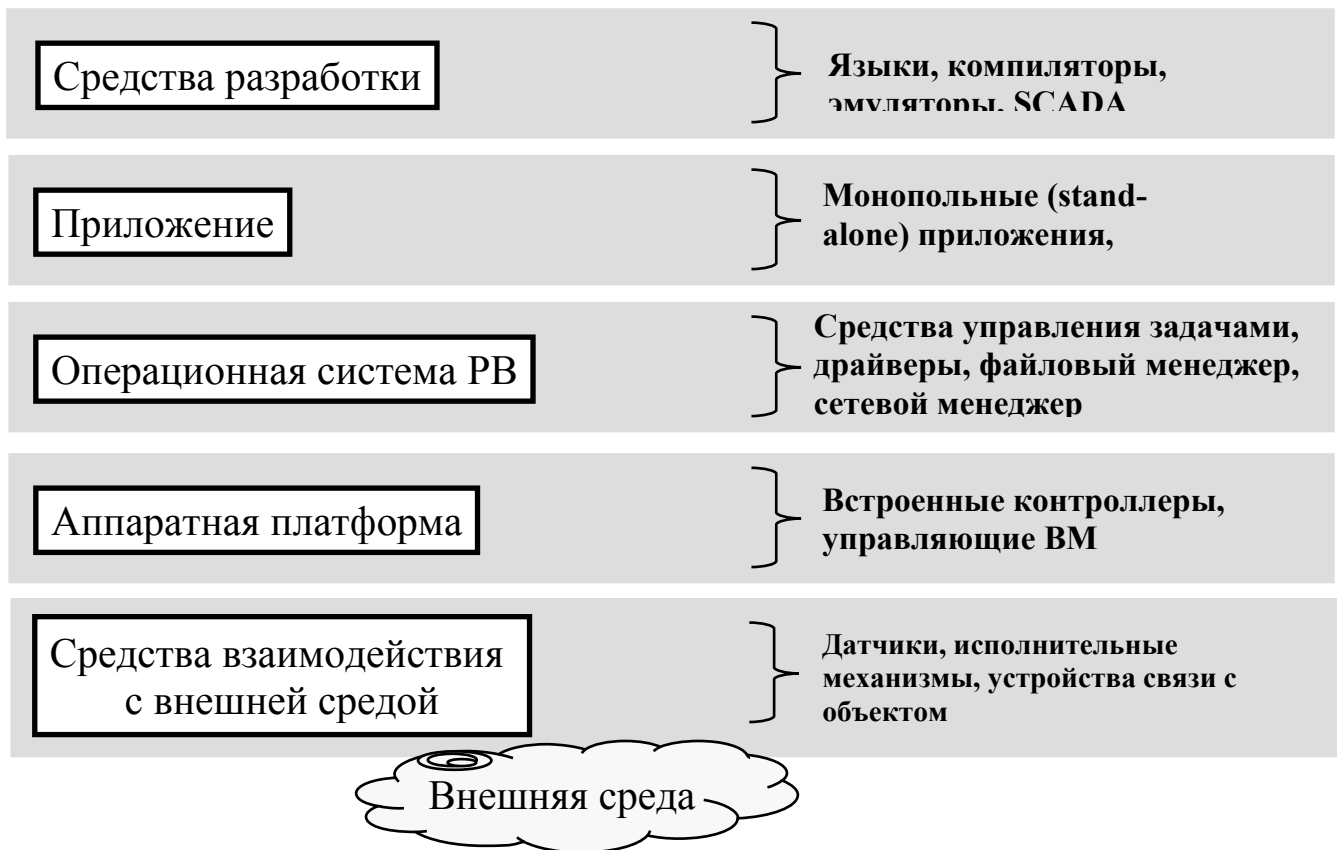


Пример организации многозадачности



Тема 6 (4 часа)

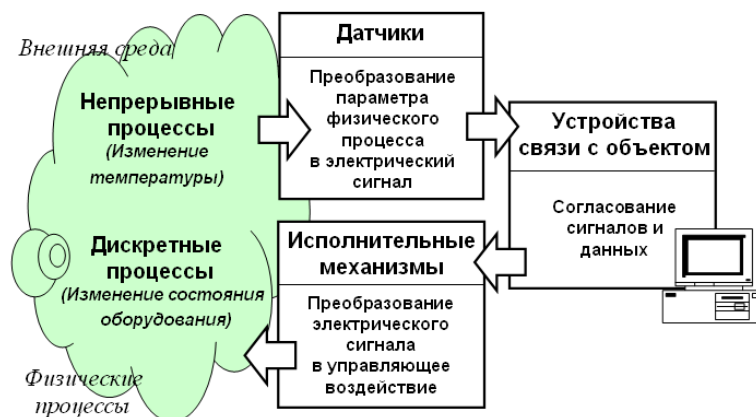
Средства взаимодействия с внешней средой в составе СРВ



Датчики – устройства для преобразования параметра физического процесса в электрический сигнал

Исполнительные механизмы предназначены для преобразования электрического сигнала в управляющее воздействие

2. Функции средств взаимодействия с внешней средой

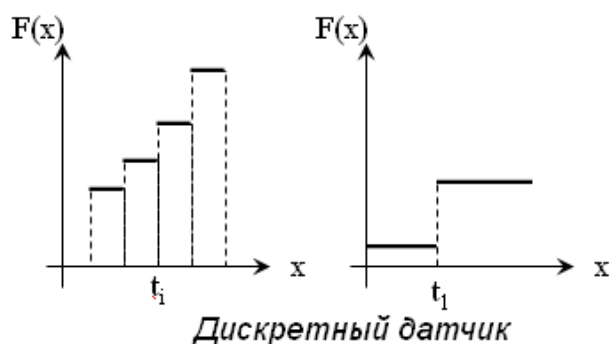
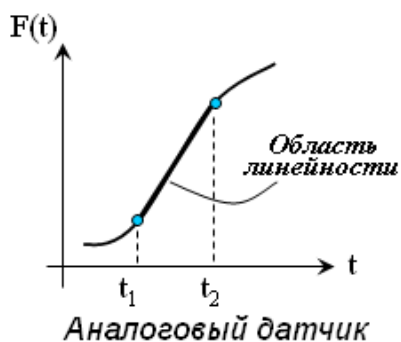
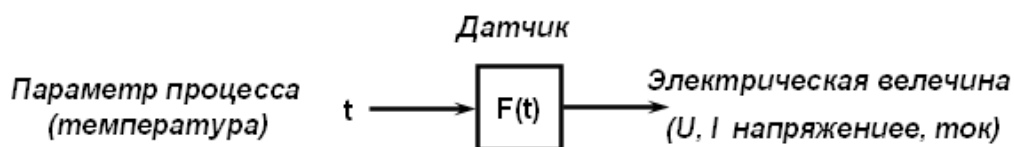


Датчики

- Аналоговые, дискретные, цифровые кодовые ...
- Абсолютные, относительные
- Контактные, бесконтактные
- Резистивные, индуктивные, емкостные

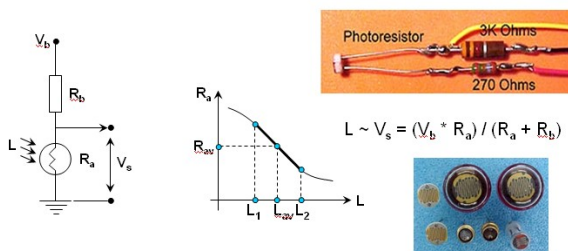
Характеристики датчиков

- Разрешение и точность
- Линейность
- Соответствие скорости измеряемого процесса
- Соответствие условиям применения (*герметичность, пыле-влаго защищенность ...*)
- Надежность
- Габариты и стоимость



Аналоговые датчики, пример

Сульфид-кадмиевый фотоэлемент (CdS Cell)



Выбор R_b :
 $R_b = R_a$ при среднем значении $L = L_{ав}$ (и, следовательно, $R_a = R_{ав}$)

Дискретные датчики, примеры



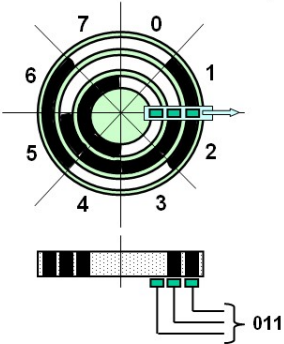
Инфракрасный излучатель + фототранзистор; электрическая цепь разрывается при наличии экранирующего элемента на пути светового потока



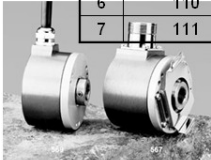
Контактные элементы - включен/выключен



Цифровые кодовые датчики, пример



	Позиционный код	Код Грея
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100



Исполнительные механизмы

Исполнительные механизмы

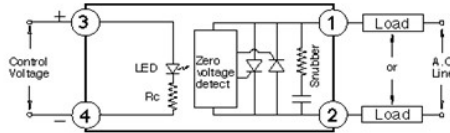
- **Контакты**
- **Шаговые двигатели**
- **Силовые привода**
- **Нагреватели, холодильные установки (управление температурой)**
- **Вентильные заслонки (управление давлением)**

Исполнительные механизмы управляются

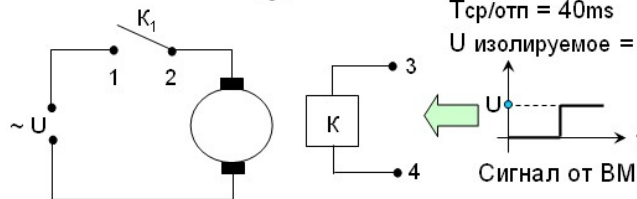
- **Аналоговыми,**
- **дискретными,**
- **цифровыми кодовыми ...**

воздействиями

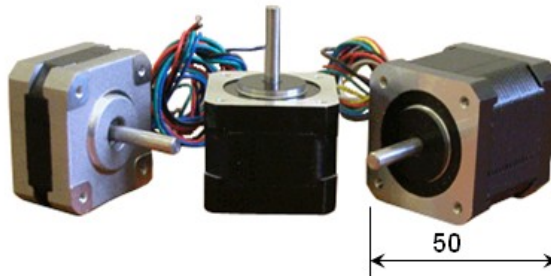
Контакты, пример



U срабатывания $\min = 3\text{v}$
 U отпускания $\max = 1\text{v}$
 I коммутируемый = 75а
 U коммутируемый = 330v
 $T_{\text{ср/отп}} = 40\text{ms}$
 U изолируемое = 4000v

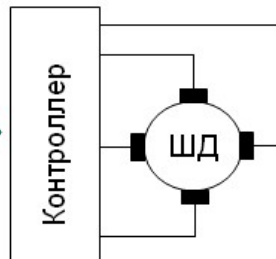


Шаговые двигатели, пример



Команда от ВМ (через LPT)

- Направление
- Количество шагов
- Скорость

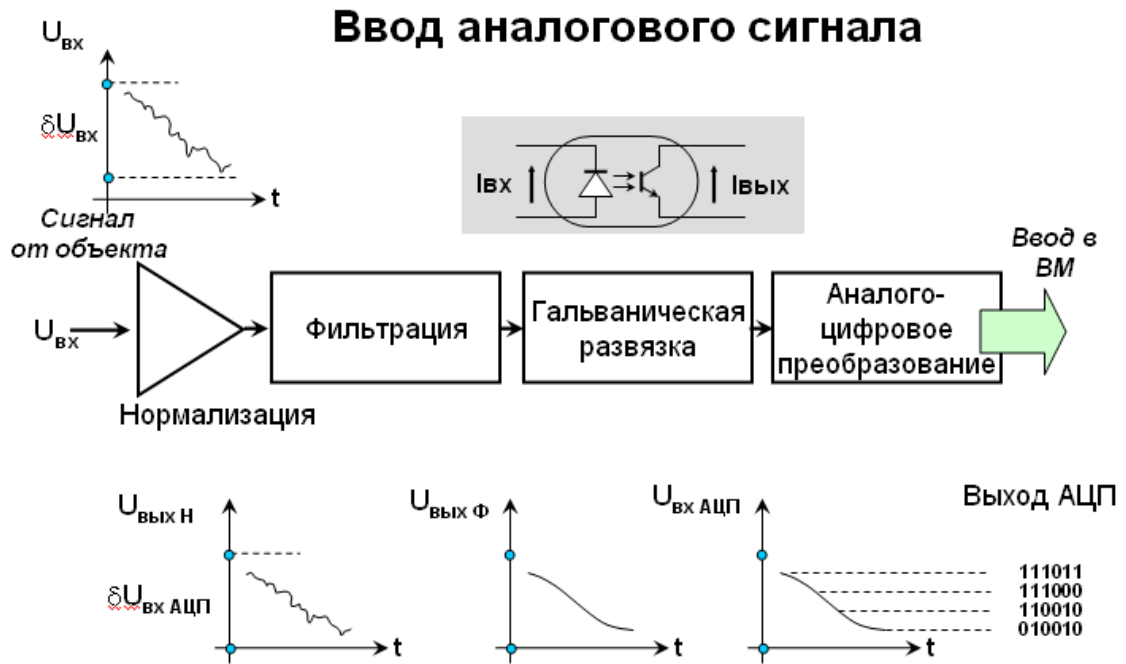


Устройства связи с объектом

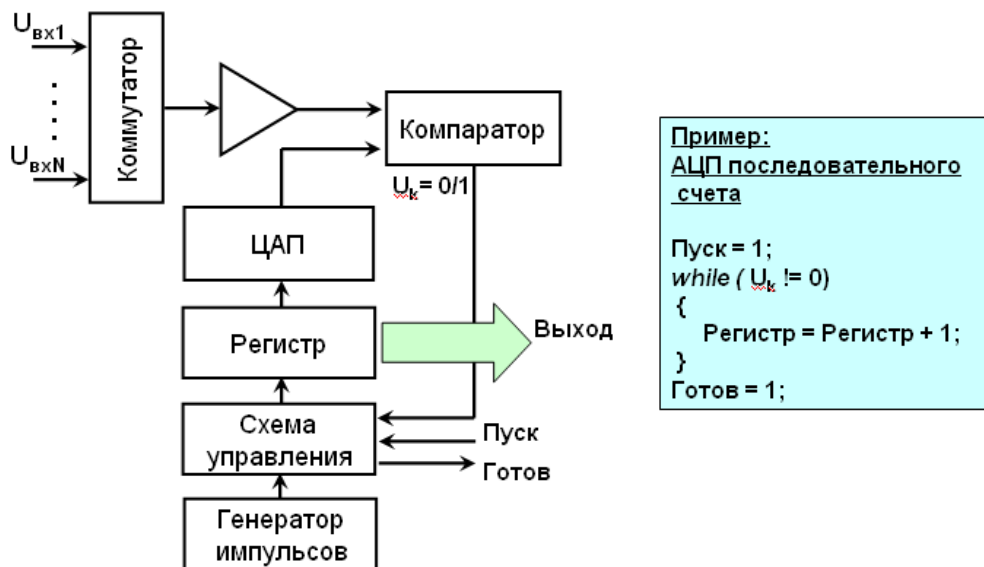
Функции:

- Нормализация сигналов согласование выходных (входных) сигналов датчиков (исполнительных механизмов) с входными (выходными) сигналами подсистем аналого-цифрового (цифро-аналогового) ввода-вывода
- Гальваническая развязка исключение электрической связи между силовыми агрегатами объекта и слаботочными схемами устройств связи

- Фильтрация подавление помех (импульсных наводок от срабатывания контакторов, 50 гц наводок от сети)
- Аналого-цифровое и цифро-аналоговое преобразование сигналов (АЦП, ЦАП)
- Ввод-вывод дискретной информации
- Поддержка интерфейсов связи с удаленными объектами



Аналого-цифровые преобразователи



Основные характеристики аналого-цифровых преобразователей

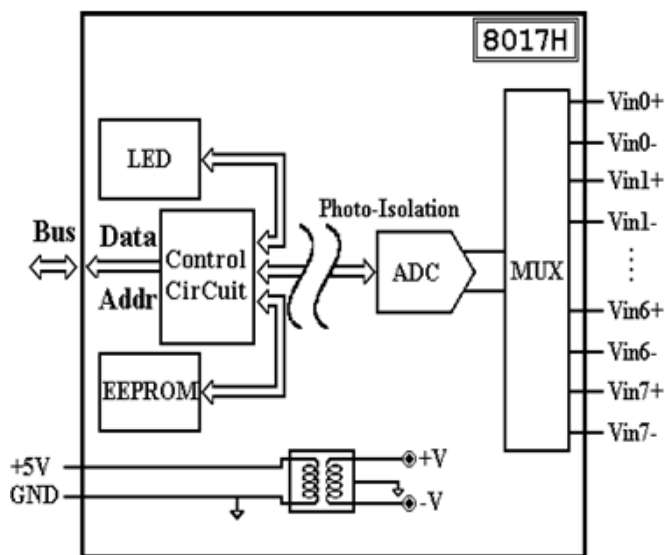
- Быстродействие - время преобразования – ($t = t_{\text{Готов}} - t_{\text{Пуск}}$, единицы – сотни микросекунд), частота (единицы – сотни КГц)

- Разрядность – 8 – 24, диапазон входного сигнала – 0 - 5v, 0 – 10v, -5 - +5v
- Точность - определяется диапазоном входного сигнала, разрядностью; представляется весом младшего разряда в % от диапазона (0.4 – 0.1%)
- Количество каналов – 4 – 128

Принципы преобразования (<http://www.gaw.ru/html.cgi/txt/doc/adc/>):

- АЦП последовательного счета (единицы килогерц)
- АЦП последовательного приближения (*в Регистр сразу заносится значение половины диапазона; компаратор выдает «0», если больше, «1» если меньше и т.д.*) (десятки килогерц)
- Параллельные АЦП
- Интегрирующие АЦП

Пример 1: АЦП ICOS i8017H

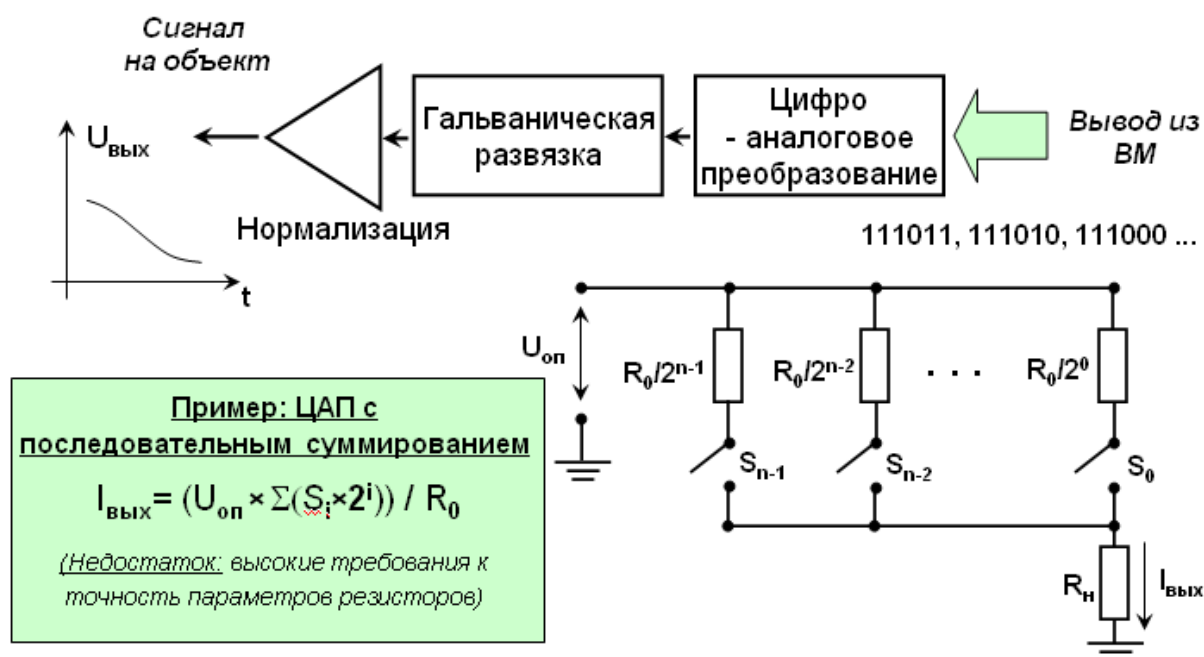


- 8 дифференциальных каналов
- Быстродействие – 100Кгц (при одном канале), 10Кгц – при 8-ми
- Точность - +/- 0.1%
- Изолируемое напряжение 3000v

Пример 2: Модуль аналогового ввода (National Instruments)



Вывод аналогового сигнала



Цифро-аналоговые преобразователи

Основные характеристики:

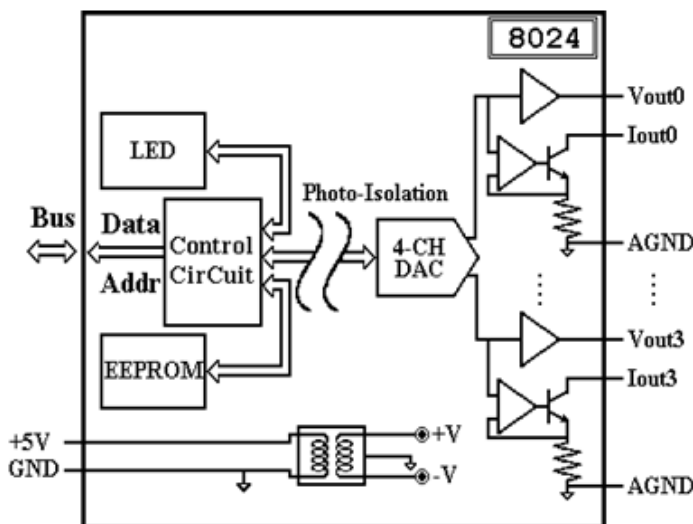
- Разрядность – 8 – 24
- Диапазон выходного сигнала – 0 - 5v, 0 – 10v, 5 – 100 ma

- Точность - определяется диапазоном входного сигнала, разрядностью; представляется весом младшего разряда в % от диапазона (0.4 – 0.1%)
- Количество каналов – 4 - 32

Принципы преобразования (<http://www.gaw.ru/html.cgi/txt/doc/dac/>):

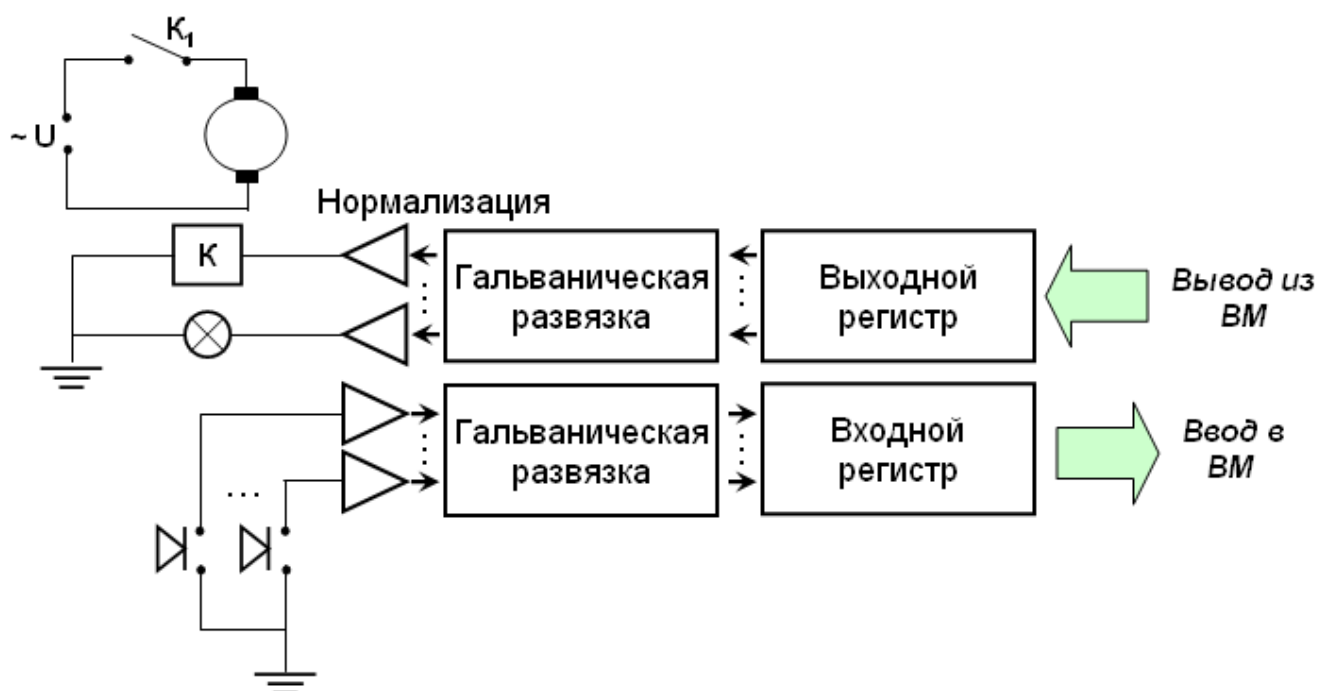
- **Последовательные ЦАП**
 - ЦАП с широтно-импульсной модуляцией
 -
- **Параллельные ЦАП**
 - ЦАП с суммированием выходных токов

Пример 1: ЦАП ICOS i8024H

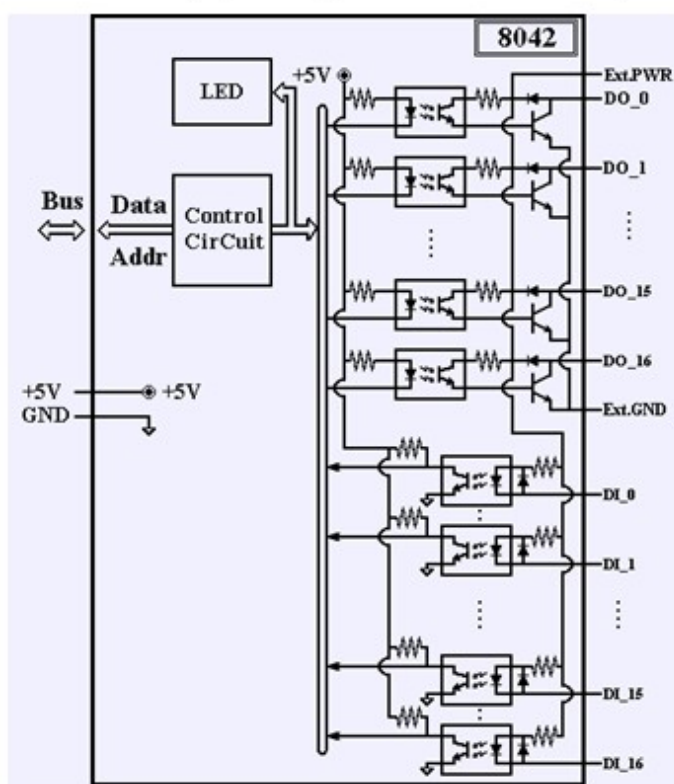


- 4 канала (по току и по напряжению)
- Точность - +/- 0.1%
- Изолируемое напряжение 3000v
- Максимальный ток нагрузки токового выхода – 5ma (при напряжении внешнего источника питания – 24v)
- Диапазон выходного напряжения – 0 - +10v

Ввод/вывод дискретной информации

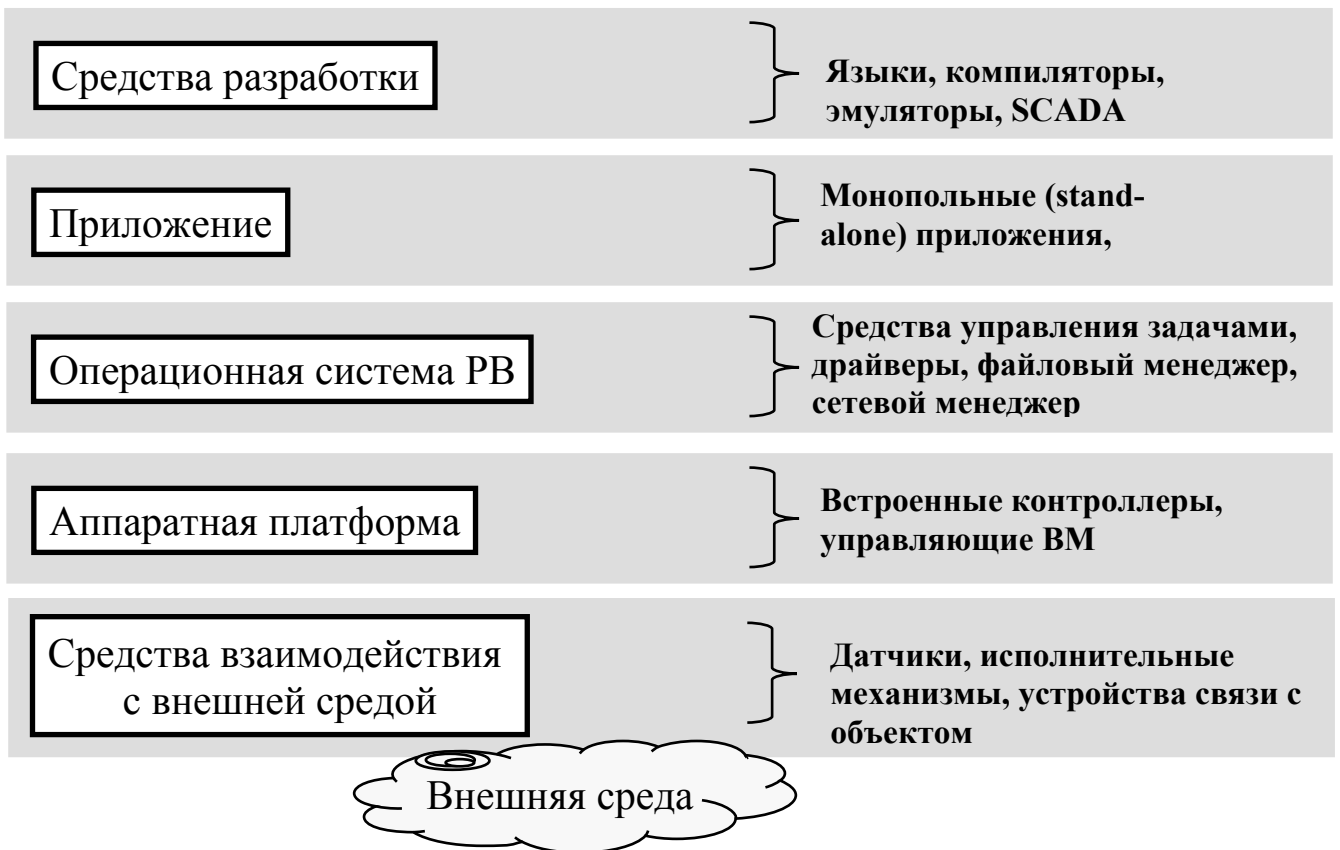


Пример: модуль ввода/вывода дискретной информации ICOS i8042



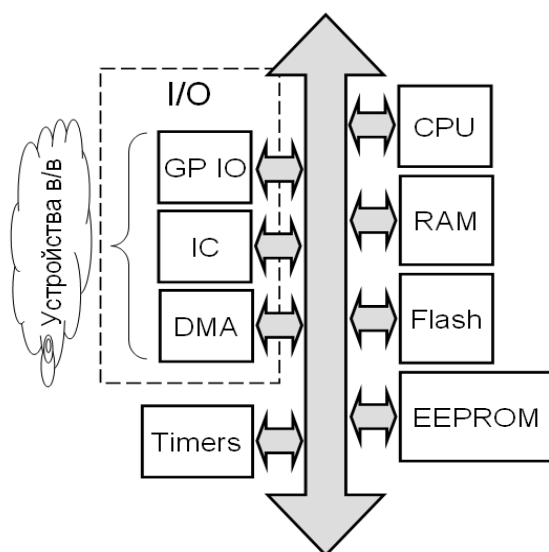
- Выход - 16 каналов, вход – 16 каналов
- Напряжение нагрузки на выходе 5 – 30v, ток нагрузки (max) – 100ma
- Входное напряжение включения (min) – 3,5v
- Входное напряжение выключения (max) – 1v
- Изолируемое напряжение 3750v

Структура СРВ



Аппаратная платформа СРВ

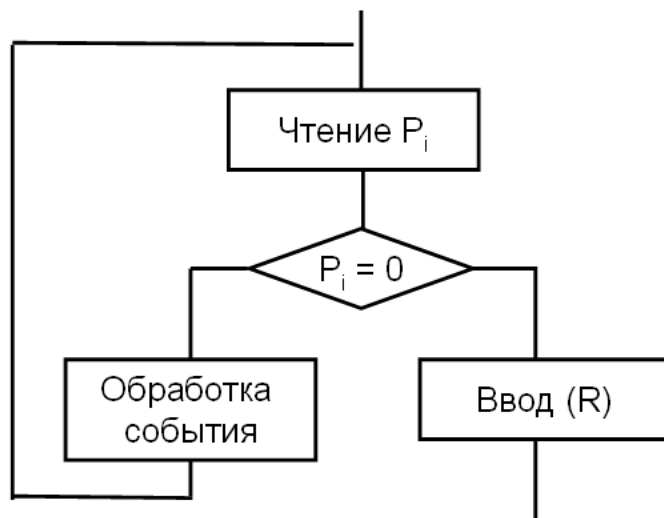
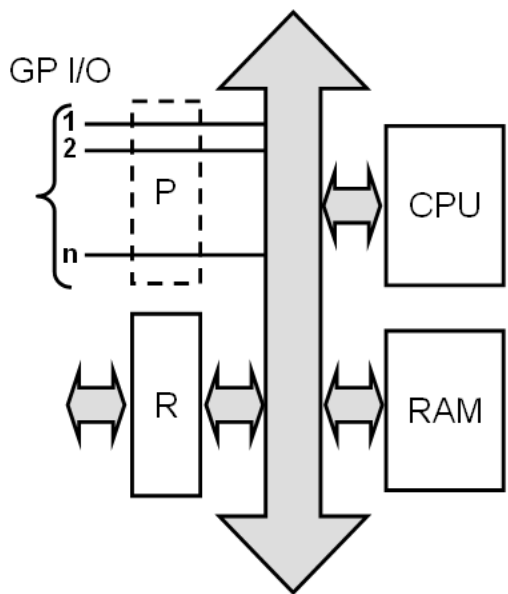
3. Аппаратная платформа СРВ



- CPU (Central Processing Unit): AMD80188-40MHz, ARM-11 – 300MHz
- RAM (Random Access Memory): 64Kb – 10Mb
- Flash: 64Kb – 10Mb
- EEPROM (Electrical Erasable Programmable Read-Only Memory): 2 – 10Kb
- Timers –
 - Interval Timer
 - Watchdog Timer
 - Real-Time Clock
- GP I/O (General Purpose I/O)
- IC (Interrupt Controller)
- DMA (Direct Memory Access)

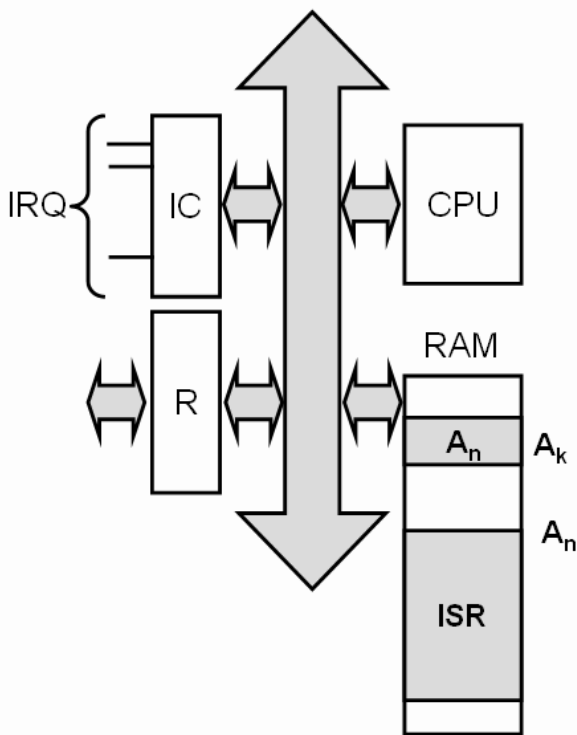
3.1. Организация ввода/вывода

3.1.1. Ввод/вывод по готовности



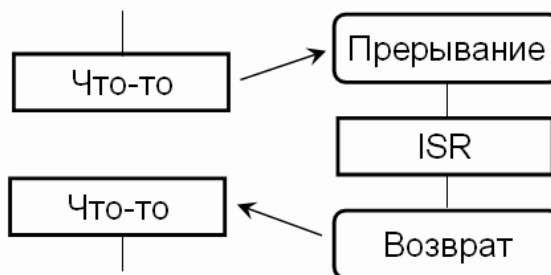
GP I/O – General Purpose I/O

3.1.2. Ввод/вывод по прерыванию



ISR – Interrupt Service Routine

1. Сигнал на входе IR_k
2. Процессор оканчивает текущую команду и запоминает контекст
3. Interrupt Controller (IC) передает адрес вектора прерывания A_k
4. Управление передается программе P , адрес точки входа которой (A_n) хранится в векторе
5. Программа P читает (записывает) содержимое регистра R
6. Восстановление контекста



ISR (Interrupt Service Routine) процедура обработки (обслуживания) прерыва-

ния

Прямой доступ к памяти (Direct Memory Access, DMA) – это обмен между системной памятью (ОЗУ) и устройством, выполняемый без непосредственного участия процессора. Обмен осуществляет *контроллер прямого доступа*.

3.1.3. Прямой доступ в память

Идея DMA – временное разделение внутренней магистрали процессора между потоком команд и вводом/выводом данных в память

Цикл процессора – действие, которое необходимо выполнить для передачи одного слова



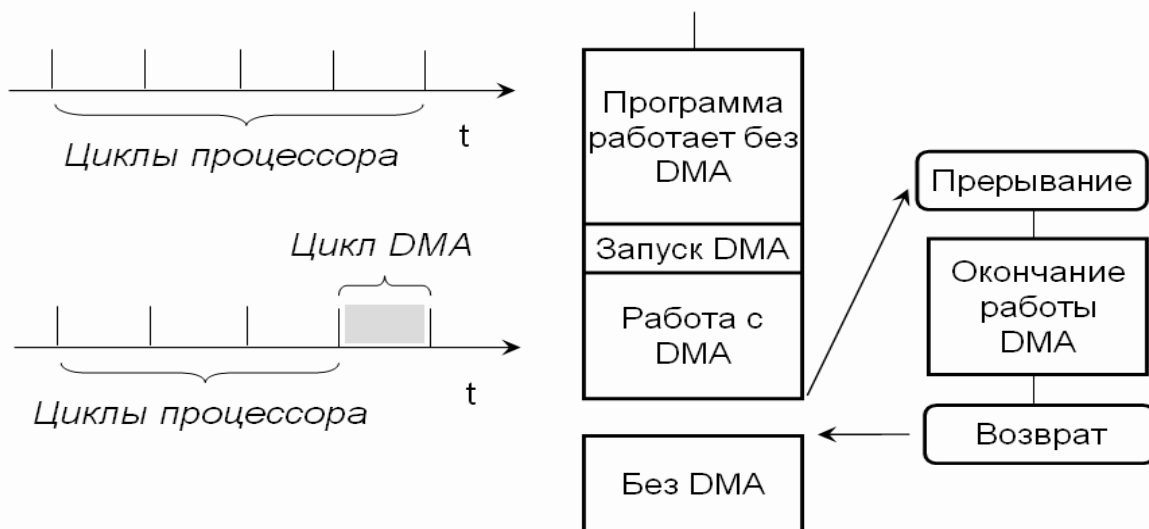
Пример: Команда **MOV AL, TOTAL** – два цикла:

- Считывание КОП
- Считывание **TOTAL** в младшую часть регистра **A**

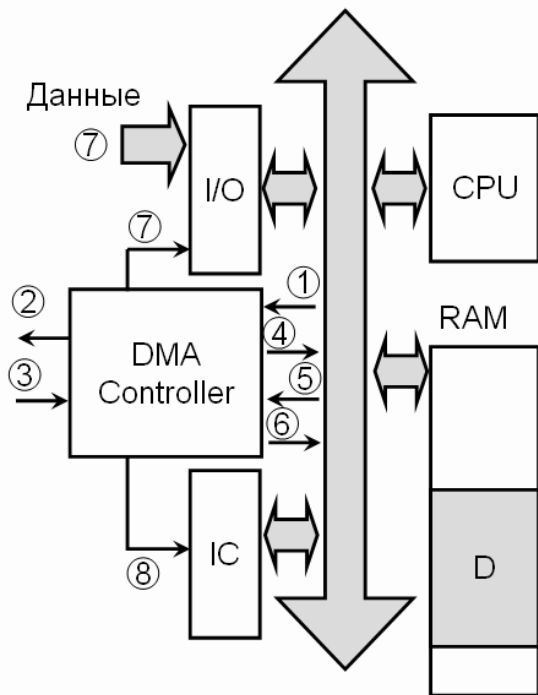
Команда процессора требует от 1 до 10 циклов

3.1.3.1. Процедура прямого доступа

Предлагается – при использовании DMA каждый n-й цикл отдавать под ввод/вывод по прямому доступу



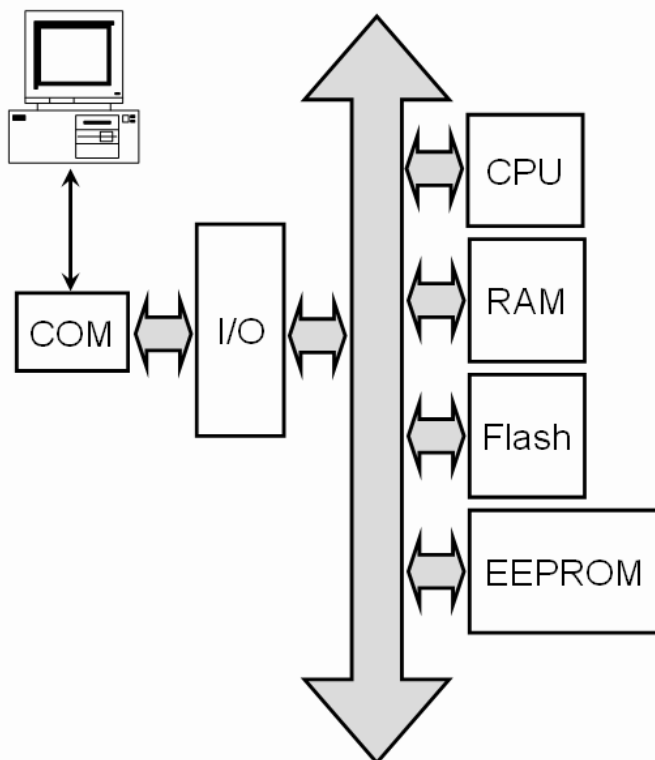
3.1.3.2. Организация канала прямого доступа



1. Инициирование DMA – установка начального адреса, количества передаваемых слов
2. Запрос ввода/вывода (*)
3. Разрешение ввода/вывода (*)
4. Запрос цикла
5. Разрешение цикла
6. Адрес ввода/вывода
7. Ввод слова
8. Запрос на прерывание по окончании ввода/вывода

(*) Установлены постоянно, пока идет обмен; снимаются по (8)

3.2. Память, доступная для чтения (ROM)



ROM – Read Only Memory

Flash

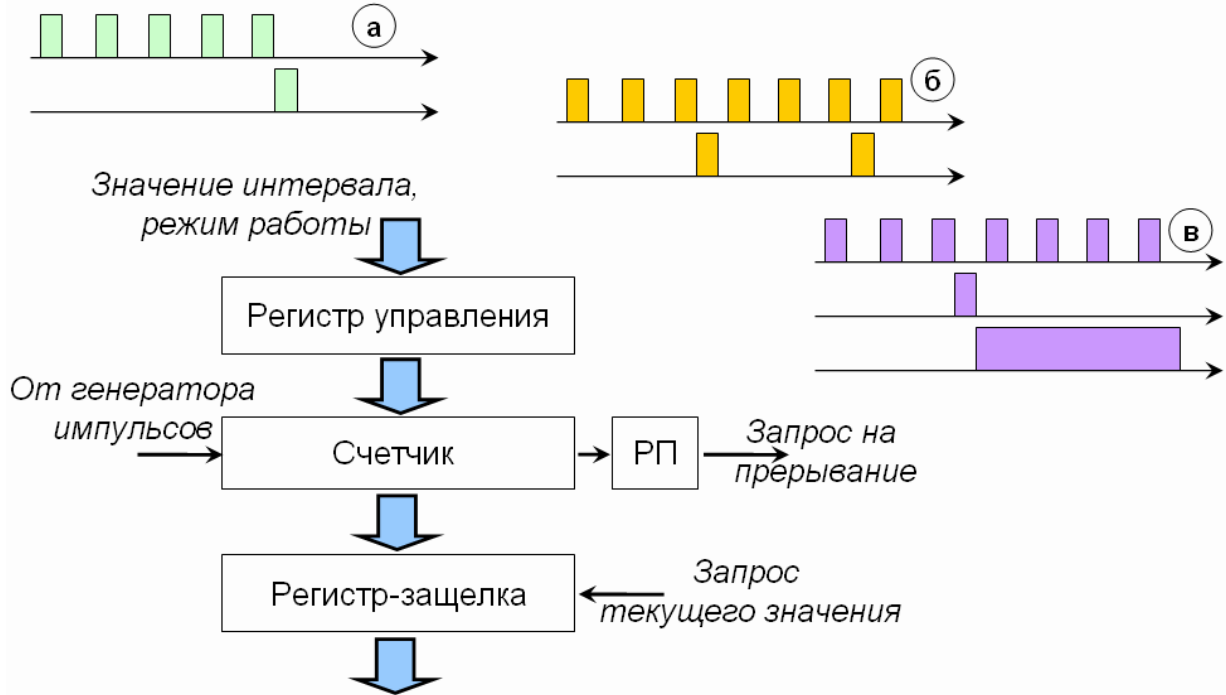
- Загружается с PC посредством соответствующих утилит
- Эмулируется файловая система
- Исполняемые программы грузятся в RAM

EEPROM

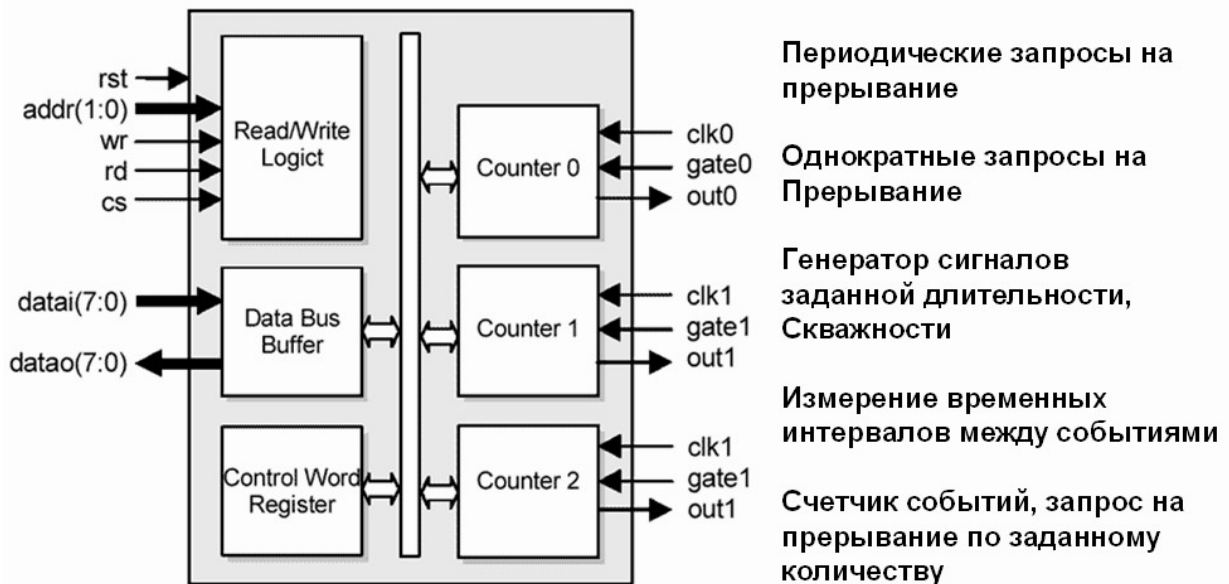
- Загружается с PC и посредством библиотечных функций
- Доступно побитное чтение
- Используется для хранения констант

3.3. Таймеры

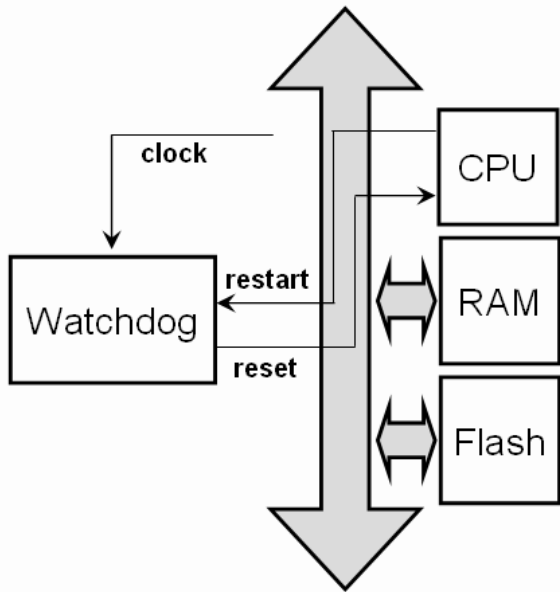
3.3.1. Интервальный таймер



3.3.1.1. Интервальный таймер, пример: i8254



3.3.2. Watchdog



```
# define interval 10

main ()
{
    while(true)
    {
        restartWD(interval)
        measure();
        takeControlDecision();
        output();
    }
}
```

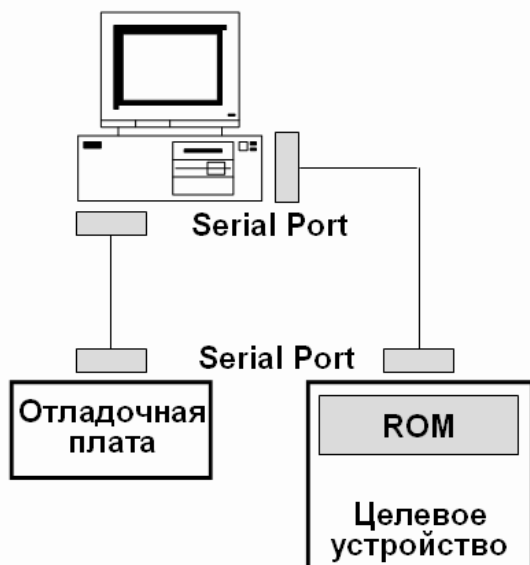
3.3.3. Real-Time Clock



- Управляется низкоуровневой процедурой (BIOS, OS)
- Библиотечные процедуры

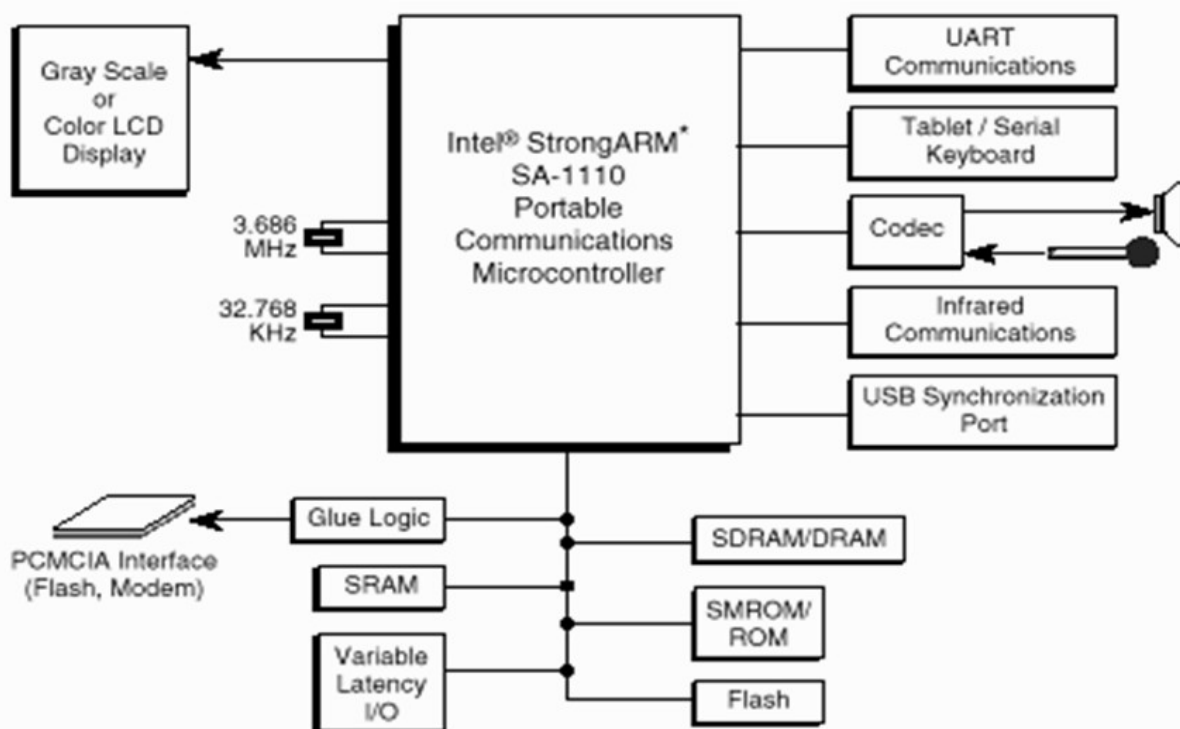

```
String MMHHDDMMYYYY
void loadRTC(String s)
String getRTS()
15:34, 24 January, 2003 –
341524012003
```

3.4. Целевая и отладочная платформы



1. Компиляция, линковка под целевую платформу
2. Отладка на симуляторе в автономном режиме
3. Загрузка в ROM отладочной системы (Evaluation board)
4. Отладка на целевой платформе (Target platform)
5. Загрузка ROM целевого устройства (Target device – телефон, PDA, контроллер системы управления ...)

3.4.1. Пример целевой платформы: SA-1110



UART (Universal Asynchronous Receiver/Transmitter) универсальный асинхронный приёмопередатчик, УАПП микросхема типа Intel 8251, 8250 или 16450, 16550A, преобразующая данные, поступающие по параллельным лини-

ям, в данные, передаваемые последовательно, и наоборот. Так как работа передатчика и приёмника не синхронизируется, поток данных должен содержать сигналы о начале и конце байта данных - стартовый и стоповый биты.

Инфракрасный, ИК тип электромагнитного излучения в диапазоне частот, расположенном между красным диапазоном видимого света и диапазоном сантиметровых волн, т. е. соответствует излучаемому теплу. Излучается некоторыми видами светодиодов.

Тема 8 (4 часа)

Планирование задач. Алгоритмы планирования без переключения и с переключением. Схемы назначения приоритетов. FIFO диспетчеризация. Карусельная диспетчеризация. Адаптивная диспетчеризация.

Необходимость планирования задач появляется, как только в очереди активных (готовых) задач появляются более одной задачи (в многопроцессорных системах - более числа имеющихся процессоров). Алгоритм планирования задач является основным отличием систем реального времени от операционных систем общего назначения. В последних целью планирования является обеспечение выполнения всех задач из очереди готовых задач, обеспечивая иллюзию их параллельной работы и не допуская монополизацию процессора какой-либо из задач. В ОСРВ целью планирования является обеспечение выполнения каждой готовой задачи *к определенному моменту времени*, при этом часто "параллельность" работы задач не *допускается*, поскольку тогда время исполнения задачи будет зависеть от наличия других задач.

Важнейшим требованием при планировании задач в ОСРВ является **предсказуемость** времени работы задачи. Это время не должно зависеть от текущей загруженности системы, количества задач в очередях ожидания (процессора, семафора, события, ...) и т.д. При этом желательно, чтобы длина этих очередей не была бы ограничена (т.е. ограничена только объемом памяти, доступной системе).

Определение. Планировщик задач (scheduler) - это модуль (программа), отвечающий за распределение времени имеющихся процессоров между выполняющимися задачами. Отвечает за коммутацию задач из состояния блокировки в

состояние готовности, и за выбор задачи (задач - по числу процессоров) из числа готовых для исполнения процессором (ами).

Главным вопросом планирования является выбор момента принятия решения:

Во-первых, когда создается новый процесс, необходимо решить, какой процесс запустить, родительский или дочерний. Поскольку оба процесса находятся в состоянии готовности, эта ситуация не выходит за рамки обычного и планировщик может запустить любой из двух процессов.

Во-вторых, планирование необходимо, когда процесс завершает работу. Этот процесс уже не существует, следовательно, необходимо из набора готовых процессов выбрать и запустить следующий.

В-третьих, когда процесс блокируется по какой-либо причине, необходимо выбрать и запустить другой процесс.

В-четвертых, решение по диспетчеризации должно приниматься после разблокировки процесса.

В-пятых, планировщик может принимать решение по истечении кванта времени, отпущенному процессу.

Алгоритмы планирования можно разделить на две категории согласно их поведению после прерываний. Алгоритмы планирования **без переключений**, иногда называемого также **неприоритетным** планированием, выбирают процесс и позволяют ему работать вплоть до блокировки либо вплоть до того момента, когда процесс сам не отдаст процессор. Процесс не будет прерван, даже если он работает часами. Соответственно, решения планирования не принимаются по прерываниям от таймера. После обработки прерывания таймера управление всегда возвращается приостановленному процессу.

Напротив, алгоритмы планирования **с переключениями**, называемого также **приоритетным** планированием, выбирают процесс и позволяют ему работать некоторое максимально возможное время. Если к концу заданного интервала времени процесс все еще работает, он приостанавливается и управление переходит к другому процессу. Приоритетное планирование тре-

бует прерываний по таймеру, происходящих в конце отведенного периода времени (решения планирования могут, например, приниматься при каждом прерывании по таймеру, или при каждом k -ом прерывании), чтобы передать управление планировщику.

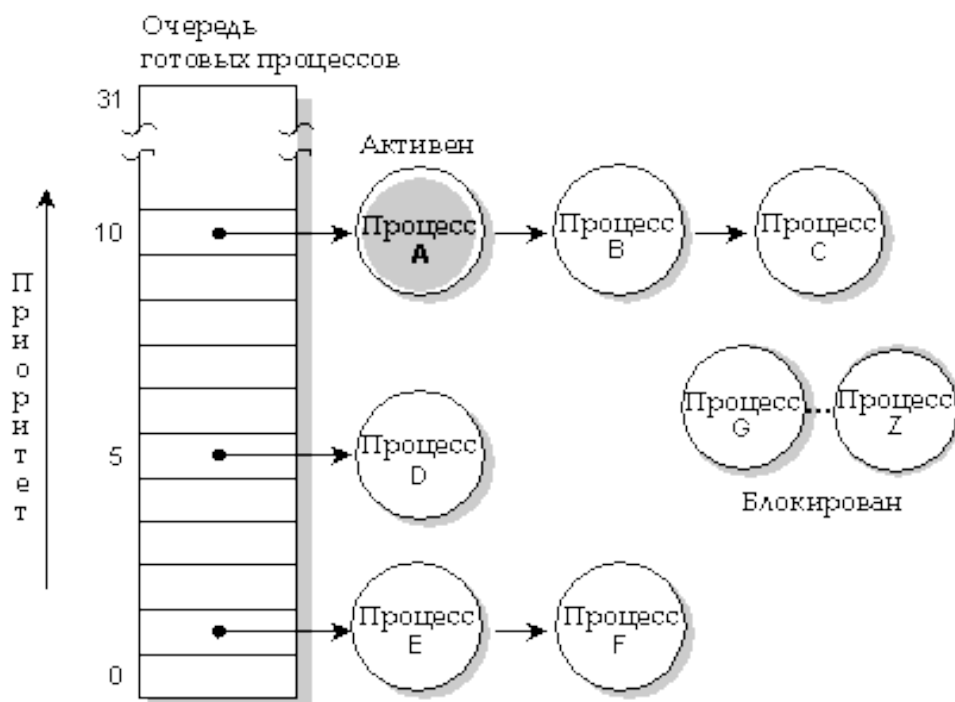
Напомним, что **приоритетом** называется число, приписанное операционной системой (а именно, планировщиком задач) каждому процессу и задаче. Существуют несколько схем назначения приоритетов.

- **Фиксированный** приоритет – приоритет, назначенный задаче при ее создании, и не меняющийся в течение ее жизни. Эта схема с различными дополнениями применяется в большинстве систем реального времени. В схемах планирования ОСРВ часто требуется, чтобы приоритет каждой задачи был уникальным, поэтому часто ОСРВ имеют большое число приоритетов (обычно 255 и более).

- **Турнирное** определение приоритета - приоритет последней исполнявшейся задачи понижается.

- Определение приоритета по алгоритму **round robin** - приоритет задачи определяется ее начальным приоритетом и временем ее обслуживания. Чем больше задача обслуживается процессором, тем меньше ее приоритет (но не опускается ниже некоторого порогового значения). Эта схема в том или ином виде применяется в большинстве UNIX систем.

Отметим, что в разных системах различные алгоритмы планирования задач могут вводить новые схемы изменения приоритетов. Например, в системе OS-9 приоритеты ожидающих задач увеличиваются для избежания слишком больших времен ожидания.

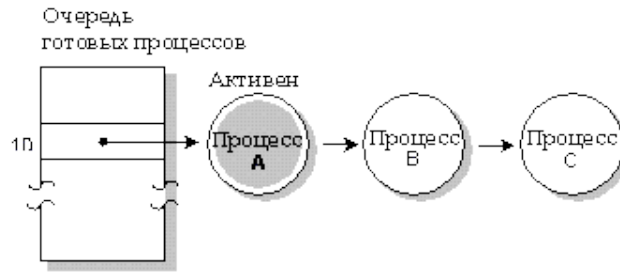


Как видно из рисунка, процессы А-Е находятся в состоянии готовности. Процессы G-Z блокированы. Процессы А, В, С имеют наивысший приоритет. Поэтому именно эти процессы будут разделять процессор в соответствии с алгоритмами диспетчеризации. Рассмотрим их.

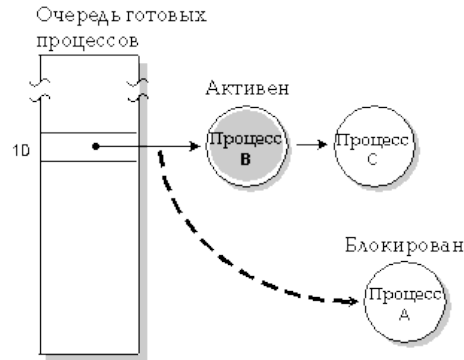
«Первым пришел – первым обслужен» (алгоритм FIFO). Является алгоритмом планирования без переключений. Процессам предоставляется доступ к процессору в том порядке, в котором они его запрашивают. При FIFO диспетчеризации процесс продолжает выполнение, пока не наступит момент, когда он:

- добровольно уступает управление (заканчивается, блокируется и т.п.);
- вытесняется процессом с более высоким приоритетом.

Заметим, что при отсутствии второго условия возможен случай, когда высокоприоритетная задача будет ожидать окончания работы низкоприоритетной.

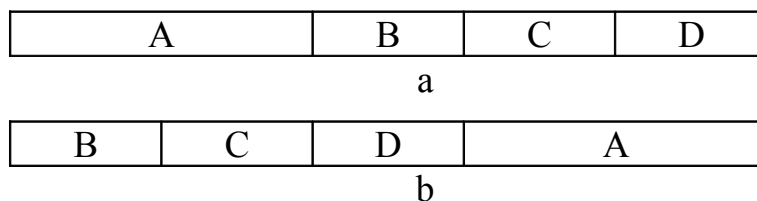


Процесс А выполняется до тех пор, пока не блокируется.



Процесс А блокируется, процесс В выполняется.

«Кратчайшая задача – первая». Этот алгоритм без переключений предполагает, что временные отрезки работы известны заранее. В этом алгоритме первым выбирается не самая первая, а **самая короткая задача**. Приведем пример.



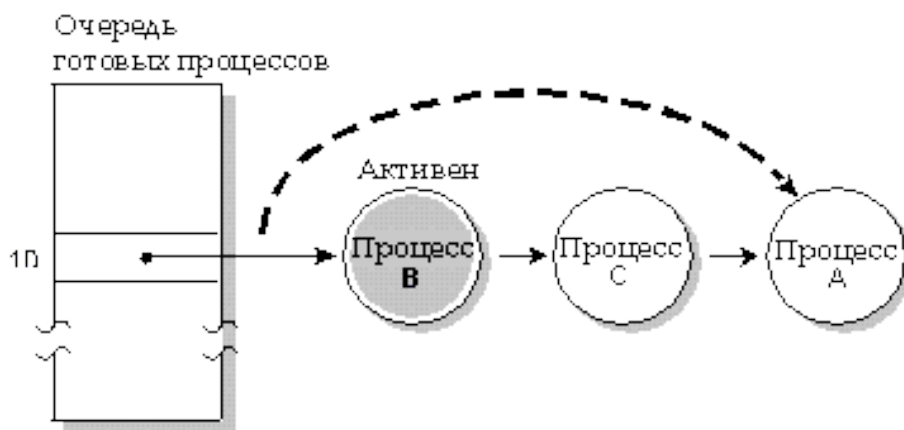
Пусть процессы А,В,С,Д имеют время выполнения 8,4,4,4 единиц времени (например, секунд). По алгоритму FIFO они должны быть запущены в том же порядке, в котором они стоят в очереди (вариант а). Тогда время выполнения процесса А будет равно 8, В – 12, С – 16 и D – 20. Среднее время выполнения для этих 4 процессов будет равно 14. Рассмотрим теперь очередь, отсортированную по времени выполнения (вариант б). Теперь время выполнения процесса В будет равно 4, С – 8, D – 12, А – 20. Среднее время в данном варианте будет равно 11.

Следует отметить, что данная схема работает только в случае одновременного наличия задач. Если не все процессы доступны с самого начала, можно привести пример, когда алгоритм ухудшает среднее время выполнения.

«Наименьшее оставшееся время выполнения». Является версией предыдущего алгоритма с переключениями. В соответствии с этим алгоритмом планировщик каждый раз выбирает процесс с наименьшим оставшимся временем выполнения. В этом случае также необходимо знать заранее время выполнения каждого процесса. Когда поступает новый процесс, его полное время выполнения сравнивается с оставшимся временем выполнения текущего процесса. Если время выполнения нового процесса меньше, текущий процесс приостанавливается и управление передается новому процессу. Эта схема позволяет быстро обслуживать короткие процессы.

«Карусельная диспетчеризация (циклическое планирование)». При карусельной диспетчеризации процесс продолжает выполнение, пока не наступит момент, когда он:

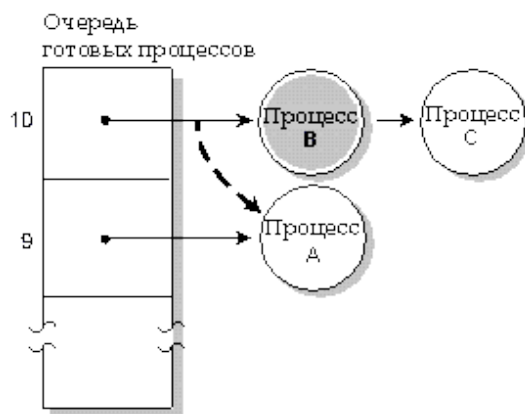
- добровольно уступает управление (т.е. блокируется);
- вытесняется процессом с более высоким приоритетом;
- использовал свой квант времени (*timeslice*). После того, как процесс использовал свой квант времени, управление передается следующему процессу, который находится в состоянии готовности и имеет такой же уровень приоритета.



Карусельная диспетчеризация. Процесс А выполняется до тех пор, пока он не использовал свой квант времени; затем выполняется следующий процесс, находящийся в состоянии готовности (процесс В).

«Адаптивная диспетчеризация». При адаптивной диспетчеризации процесс ведет себя следующим образом:

- Если процесс использовал свой квант времени (т.е. он не блокировался), то его приоритет уменьшается на 1. Это получило название *снижение приоритета* (priority decay). "Пониженный" процесс не будет продолжать "снижаться", даже если он использовал еще один квант времени и не блокировался - он снизится только на один уровень ниже своего исходного приоритета.
- Если процесс блокируется, то ему возвращается первоначальное значение приоритета.



Адаптивная диспетчеризация. Процесс А использовал свой квант времени; его приоритет снизился на 1. Выполняется следующий процесс в состоянии готовности (процесс В).

В системах реального времени наиболее распространенными являются алгоритмы FIFO, адаптивной диспетчеризации, карусельной диспетчеризации и их разновидности.

V. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ

В качестве самостоятельной работы рекомендуется знакомство с публикациями в периодических изданиях, поиск информации в Internet (например, по ссылке [www:/amursu.ru/citforum](http://www.amursu.ru/citforum)) по темам предлагаемым в качестве самостоятельного изучения и подготовки лабораторных работ.

Вопросы для самостоятельного изучения:

Проблемы взаимодействия процессов.

Семафоры.

Мьютексы.

Критические секции.

Задача «обедающих философов».

События. Использование событий с автосбросом.

К видам самостоятельной работы также относится подготовка к лабораторным работам, составление отчетов, разбор материалов лекций во время семестра.

Контроль за выполнением самостоятельной работы осуществляется на экзамене.

VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ

Межсессионный контроль осуществляется на основе выполнения домашних заданий, лабораторных работ, промежуточных тестов.

По итогам выполнения перечисленных видов работ в сроки, установленные деканатом (как правило, на 6-ой и 12-ой неделе семестра) преподавателем выставляется аттестационная оценка по пятибалльной системе.

VII. КОМПЛЕКТЫ ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ

Билет № 1

1. Общая структура СРВ.
2. Требования к языкам и операционным системам реального времени.

Билет № 2

1. Определение СРВ.
2. Обмен информацией между процессами.

Билет № 3

1. Жесткие и мягкие СРВ.
2. Средства взаимодействия с внешней средой в составе СРВ.

Билет № 4

1. Структура СРВ.
2. Датчики как средство взаимодействия с внешней средой в составе СРВ.

Билет № 5

1. Операционные системы реального времени.
2. Исполнительные механизмы как средство взаимодействия с внешней средой в составе СРВ.

Билет № 6

1. Отличие ОСРВ от ОС общего назначения.
2. Устройства связи с объектом.

Билет № 7

1. Системы разработки и системы исполнения.
2. УСО на примере ЦАП и АЦП.

Билет № 8

1. Характеристики ОСРВ.
2. Структура СРВ.

Билет № 9

1. Механизмы реального времени.

2. Аппаратная платформа СРВ.

Билет № 10

1. Архитектура ОСРВ.
2. Организация ввода/вывода. Ввод вывод по готовности.

Билет № 11

1. Классы ОСРВ.
2. Организация ввода/вывода. Ввод вывод по прерыванию.

Билет № 12

1. Процессы в ОСРВ.
2. Таймеры.

Билет № 13

1. Состояния процесса в ОСРВ.
2. Планирование задач.

Билет № 14

1. Жизненный цикл процесса в ОСРВ.
2. Алгоритмы планирования без переключения и с переключением.

Билет № 15

1. Потоки. Приоритеты процессов в ОСРВ.
1. 2. Схемы назначения приоритетов.

Билет № 16

1. Пример СРВ. Характеристики вычислительного процесса в СРВ.
2. FIFO диспетчеризация.

Билет № 17

1. Структура СРВ на примере задачи контроля испытательного оборудования.
2. Карусельная диспетчеризация.

Билет № 18

1. Организация вычислительного процесса в СРВ.
2. Адаптивная диспетчеризация.

Билет № 19

1. Функции ядра ОСРВ.
2. Организация ввода/вывода. Прямой доступ в память.

Билет № 20

1. Структура СРВ на примере задачи диспетчеризации.
2. Синхронизация процессов в СРВ.

VIII. КАРТА ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ КАДРАМИ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОГО СОСТАВА

Все виды занятий по дисциплине ведет ассистент Дрюков А.А.

СОДЕРЖАНИЕ

I.	РАБОЧАЯ ПРОГРАММА	3
II.	МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ И ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ	8
III.	ТЕХНОЛОГИЯ ВЫПОЛНЕНИЯ И ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ	9
IV.	КОНСПЕКТ ЛЕКЦИЙ	37
V.	МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ	96
VI.	МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ	96
VII.	КОМПЛЕКТ ЭКЗАМЕНАЦИОННЫХ БИЛЕТОВ	97
VIII.	КАРТА ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ КАДРАМИ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОГО СОСТАВА	100

Дрюков Александр Александрович,
ассистент кафедры ИиУС АмГУ

Учебно-методический комплекс по дисциплине «Системы реального времени» для спец. 230201 – Информационные системы и технологии

Тираж

Заказ