Міністерство світи і науки України
Дніпровський національний університет імені Олеся Гончара
фізико-технічний факультет
кафедра радіоелектронної автоматики

В.Б.Мазуренко

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КОМП`ЮТЕРНО-ІНТЕГРОВАНИХ ПРИСТРОЇВ ТА СИСТЕМ

Конспект лекцій

Дніпро

2020

Наведено конспект лекцій з курсу «Програмне забезпечення комп`ютерно-інтегрованих пристроїв та систем», який розроблено відповідно до освітньо-професійної програми другого рівня вищої освіти «Автоматизація та комп’ютерно-інтегровані технології». Для студентів фізико-технічного факультету ДНУ, що навчаються за спеціальністю 151 «Автоматизація та комп’ютерно-інтегровані технології» на другому рівні вищої освіти.

Укладач: доцент кафедри радіоелектронної автоматики фізико-технічного факультету Дніпровського національного університету ім. Олеся Гончара Мазуренко Валерій Борисович.

Лекция № 1

Тема: Компьютерно-интегрированные технологии. Введение.

Оглавление

Определение технологии

Современное общество – это общество технологий. Все предметы, изделия, что нас окружают, средства и системы коммуникации, сервисы и услуги, которыми пользуется современный человек созданы путем применения тех или иных технологий.

Технология в общем случае – это совокупность методов обработки, изготовления, изменения состояния, свойств, формы сырья, материала или полуфабриката в процессе производства.

Все известные технологии можно классифицировать по сфере их приложения, например:

- Сельскохозяйственные технологии
- Производственные технологии
- Технологии научных исследований
- Военные технологии
- Транспортные технологии
- Информационные технологии
- Телекоммуникационные технологии
- Социальные технологии
- Политтехнологии
- Технологии в сфере медицины: лечение, профилактика, оздоровление и др.

Приведенные примеры - это обобщение, которое указывает на совокупность, целый набор отдельных технологий. Так, например, промышленные технологии включают в себя следующие разделы, которые в свою очередь также являются обобщением:

- Технология металлов
- Химическая технология
- Машиностроительные технологии
- Технология строительства и др.

Среди промышленных технологий можно выделить группу под общим названием "Технологии неразрушающего контроля", в которую входят технологии радиоволнового, акустического, оптического и других видов НК.

Если продолжать углубляться, то придём в конечном итоге к конкретной технологии, которая реализуется при производстве конкретного вида продукции, например, "технология контроля сварных соединений трубопроводов радиографическим методом".

Общее число применяемых во всех сферах человеческой деятельности конкретных технологий, по всей видимости, неизвестно.

Зачастую технологии различают по физическим (или другим) явлениям, на основе которых они функционируют, например:

- Технологии, связанные с электричеством
- Акустические технологии

Существует еще одна классификация технологий: по степени вовлечения в технологический процесс труда человека. Роль человека при производстве продукции различна. Наиболее простой классификацией по степени вовлечения труда человека можно считать следующее разделение технологий:

- ручные (используется только труд человека);
- механизированные (механизация тяжелого физического труда)
- автоматизированные (частичное участие человека, чаще всего для манипулирования техникой);
- автоматические (без какого-либо участия человека во время производства).

Средства массовой информации выделяют такое понятие, как "высокие технологии" - это технологии, которые определяют направление научно-технического прогресса современного общества, которые повышают, подымают вверх технический уровень цивилизации, да и сами эти технологии находятся на вершине самых передовых научных знаний. Высокими эти технологии называют еще и потому, что не только создание, но и применение этих технологий требует вовлечения специалистов высочайшей квалификации, требует наличия высокоточного, специализированного оборудования, а также требует высоких финансовых затрат. В силу указанных факторов высокими технологиями обладают только высокоразвитые государства, с высоким экономическим и научным потенциалом. К высоким технологиям, в частности, относят:

- Космические технологии
- Нанотехнологии
- Биотехнологии и др.

Информационные технологии

Среди всех видов технологий отдельно выделяются информационные технологии (ИТ). Что это такое? Напомним, что технология - это совокупность методов обработки, изготовления, изменения состояния, свойств, формы сырья, материала или полуфабриката в процессе производства. Конкретная технология предполагает наличие трех основных составных элементов: материал, средства обработки материала, методы обработки материала. Когда сырьем для обработки и результатом является *информация в форме данных*, а средствами обработки являются программы вычислительной техники, то под

информационной технологией следует понимать совокупность методов (способов) в форме программ вычислительной техники для сбора, накопления, хранения, поиска, обработки и выдачи информации потребителю.

Компьютерно-интегрированные технологии

Применяемые технологии постоянно совершенствуются в направлении уменьшения доли ручного труда, повышения производительности и качества.

Появление компьютеров позволило не только механизировать технологические процессы, но и перейти к автоматизированным и даже автоматическим технологиям, управление механизмами в которых берет на себя компьютер. Компьютер, а точнее компьютерная техника, становится составной частью технологического процесса, иначе говоря, интегрируется в него. Многие технологии уже в принципе не реализуются без компьютерных систем. К таким технологиям относятся многие телекоммутационные технологии, технологии работы с видео и фото изображениями, технологии управления космической и ракетной техникой, энергетическими установками, системами распределения электроэнергии и, конечно же, информационные технологии. Однако, многие технологии по-прежнему не требуют, применения компьютерной техники, а некоторые до сих пор остаются ручными, например, технологии художественных промыслов.

Таким образом, под компьютерно-интегрированной технологией можно было бы понимать любую технологию, которая реализуется с применением компьютерной техники. Однако, если пользоваться данным определением, возникает противоречие. Получается, что если выполняется глажка ткани обычным электрическим утюгом, то это технология глажки ткани. Если в утюге используется микропроцессор для управления нагревом, то это - компьютерно-интегрированная технология глажки?

Для того, чтобы разобраться с данным вопросом, следует в первую очередь выяснить, для решения каких задач применяется компьютерная техника в технологическом процессе? На самом деле, число этих задач совсем невелико:

1) получение данных об объекте, то есть измерение,
2) выработка управляющего воздействия.

В автоматических и автоматизированных технологиях - это все, других задач нет. Таким образом, компьютер имеет дело не с технологией изготовления, лечения или глажки, а только с технологиями измерения и управления, без которых данная конкретная технология изготовления, лечения или глажки не может быть реализована.

Обобщая вышесказанное, можно дать следующее определение компьютерно-интегрированных технологий.

Компьютерно-интегрированными технологиями (КИТ) называются технологии измерения, контроля, принятия решений и управления, которые реализуются с применением компьютерной техники.

Примечание: Компьютерно-интегрированные технологии для решения большого спектра задач используют информационные технологии, при этом ИТ не являются частью КИТ, также как и КИТ не является частью ИТ.

Возвращаясь к приведенному выше примеру можно говорить не о компьютерно-интегрированной технологии глажки, а о компьютерно-интегрированной технологии управления нагревом утюга. Терминологически и лексически применительно к какой-либо технологии, устройству, агрегату правильно говорить в отношении КИТ только об управлении, то есть: компьютерно-интегрированная технология управления автомобилем, КИТ управления ткацким станком и т.д. При этом подразумевается, что данная КИТ включает и измерение, так как управление без получения данных об объекте, то есть без измерения, невозможно.

В настоящем курсе технологии научного исследования и генерации объектов не рассматриваются, так как КИТ научного исследования изучается в целом ряде других читаемых на кафедре курсах, а КИТ генерации объектов в нестоящее время находится в разработке и не может рассматриваться как установившаяся.

Программно-аппаратные средства компьютерно-интегрированных технологий

Компьютерно-интегрированные технологии реализуются в виде управляющих и измерительных систем. Можно сказать, что системы, в которых реализуются компьютерно-интегрированные технологии, представляют собой следующее поколение измерительных систем, а также АСУ ТП - автоматизированных систем управления технологическими процессами. То есть - это те же системы, но построенные на основе компьютерной техники.

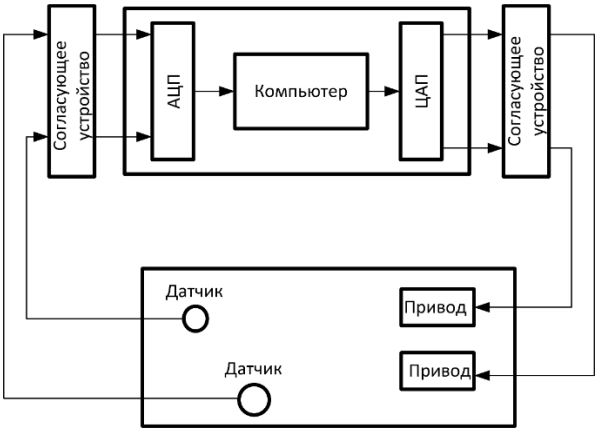


Рис. 1 Сосредоточенная система

Классическая архитектура управляющей системы воплощена в моносистеме (рис.1) – сосредоточенной системе, которая обеспечивает управление сосредоточенным объектом управления. Сосредоточенными в данном случае можно называть системы, в которых расстояние от датчиков и приводов до компьютера исчисляется метрами или десятками метров.

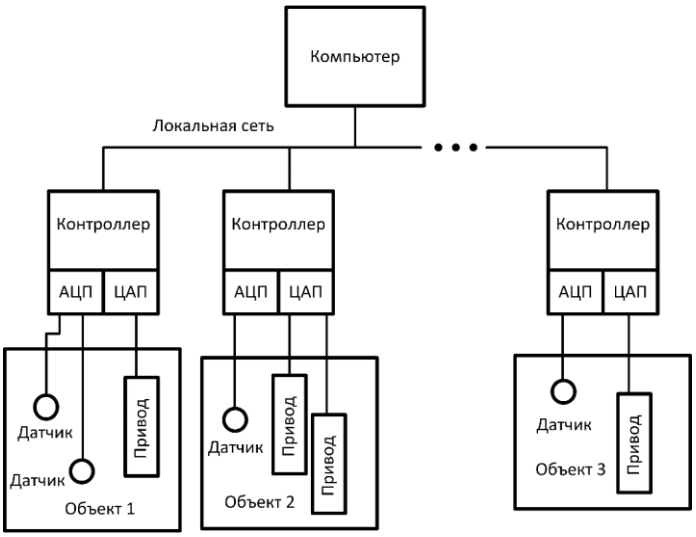


Рис.2 Распределенная система

Со временем стали развиваться распределенные системы (рис. 2), которые обеспечивают измерение и управление на совокупности объектов. В этом случае расстояния обычно составляют сотни метров и километры, а в некоторых случаях - десятки километров.

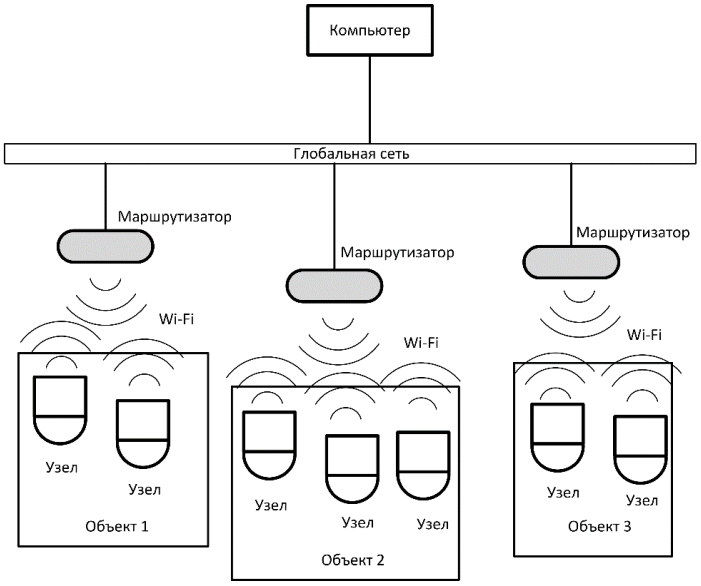


Рис. 3 Глобально распределенная система

Класс решаемых при помощи компьютерно-интегрированных технологий постоянно расширяется. В последнее время все более широкое применение находят глобально распределенные системы - системы в которых

объекты контроля отстоят на значительные расстояния - расстояния, которые уже не ограничены какими-либо пределами. Такие системы строятся по схеме, представленной на следующем рисунке. Они, в основном, являются измерительными.

Как можно видеть, системы, реализующие компьютерно-интегрированные технологии, в основном строятся с применением таких аппаратных средств:

- компьютер,
- аналогово-цифровой преобразователь (АЦП),
- цифро-аналоговый преобразователь (ЦАП),
- датчики (узлы),
- привода (или исполнительные устройства),
- локальная сеть,
- глобальная сеть.

В настоящем курсе, который носит название "Програмне забезпечення комп`ютерно-інтегрованих пристроїв та систем" будут рассмотрены программные средства, под управлением которых функционируют аппаратные средства. Для освоения компьютерно-интегрированных технологий необходимо получить знания о следующих программных средствах:

- системное программное обеспечение (ПО),
- технология "клиент-сервер",
- базы данных,
- системы программирования контроллеров,
- SCADA-системы.

Структура программного обеспечения

Совокупность программ, предназначенная для решения задач на компьютере, называется программным обеспечением. Программное обеспечение (ПО), можно условно разделить на три категории:

1. **Системное** (программы общего пользования), выполняющие различные вспомогательные функции, например создание копий используемой информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и т.д.

2. **Прикладное**, обеспечивающее выполнение задач, необходимых пользователю: обработка информационных массивов, математическое моделирование, редактирование текстовых документов, создание и редактирование изображений, и т.д.

3. **Инструментальное** (системы программирования), обеспечивающее разработку новых программ для компьютера на языке программирования.

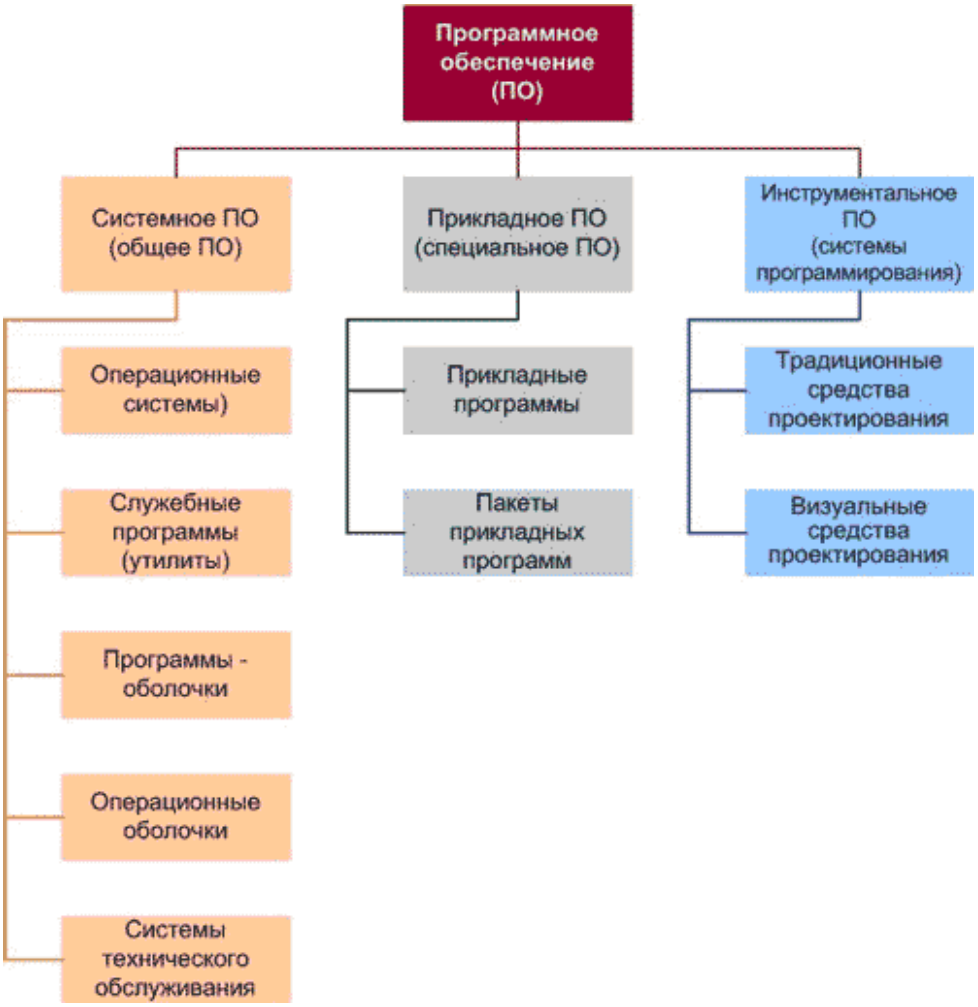


Рис. 4. Классификация программного обеспечения

В данном курсе рассматривается системное и инструментальное ПО, которое необходимо для создания и функционирования прикладного ПО. Прикладное ПО в данном курсе не рассматривается, так как в задачах измерения и управления каждая программа (прикладное ПО), которая выполняется в системе для решения конкретной задачи измерения, контроля или управления, является уникальной, поскольку уникальной является либо сама задача, либо оборудование (аппаратная часть) на котором данная задача должна быть решена. Количество созданного и используемого прикладного ПО на всех существующих технических средствах не поддается исчислению. Систематизация прикладного ПО компьютерно-интегрированных приборов и систем не проводится в силу нецелесообразности. На практике важно научиться применять системное и инструментальное ПО, а отдельные экземпляры прикладного ПО рассматривать в качестве примеров.

Контрольные вопросы по теме

Уровень модуля

1. Дайте определение понятия "технология".
2. Приведите пример классификации технологий по сфере приложения.
3. Приведите примеры технологий, различающихся по виду физических явлений, которые положены в основу их функционирования?
4. Каким образом классифицируются технологии по степени вовлечения человеческого труда?
5. Какие технологии называют высокими?
6. Почему, по вашему мнению, применяется термин высокие технологии?
7. Какие технологии можно отнести к высоким технологиям?
8. Какие технологии называют информационными?
9. Для решения каких задач применяется компьютерная техника в технологическом процессе?
10. Дайте определение понятию "компьютерно-интегрированные технологии".
11. Информационные технологии и компьютерно-интегрированные технологии – это одно и то же? Почему?
12. Представьте на рисунке функциональную схему сосредоточенной управляющей системы.
13. Представьте на рисунке функциональную схему распределенной управляющей системы.
14. Представьте на рисунке функциональную схему глобально распределенной измерительной системы.
15. Перечислите основные аппаратные средства компьютеризированных систем управления и контроля.
16. Перечислите основные программные средства компьютеризированных систем управления и контроля.

Уровень курса

1. Понятие компьютерно-интегрированных технологий.
2. Архитектура сосредоточенной компьютерно-интегрированной системы.
3. Архитектура распределенной компьютерно-интегрированной системы.

Лекция №2

Тема: Операционные системы. (Часть первая)

Оглавление

Источники:

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.: ил.

   https://rutracker.org/forum/viewtopic.php?t=4956359

2. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил.
3. Бабак В.П. Теоретические основы информационно-измерительных систем: Учебник / В.П. Бабак, С.В. Бабак, В.С. Ерёменко и др. – К., 2014. – 832 с.

Структура программного обеспечения

Совокупность программ, предназначенная для решения задач на компьютере, называется программным обеспечением. Программное обеспечение (ПО), можно условно разделить на три категории:

1. **Системное** (программы общего пользования), выполняющие различные вспомогательные функции, например создание копий используемой информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и т.д.

2. **Прикладное**, обеспечивающее выполнение задач, необходимых пользователю: обработка информационных массивов, математическое моделирование, редактирование текстовых документов, создание и редактирование изображений, и т.д.

3. **Инструментальное** (системы программирования), обеспечивающее разработку новых программ для компьютера на языке программирования.

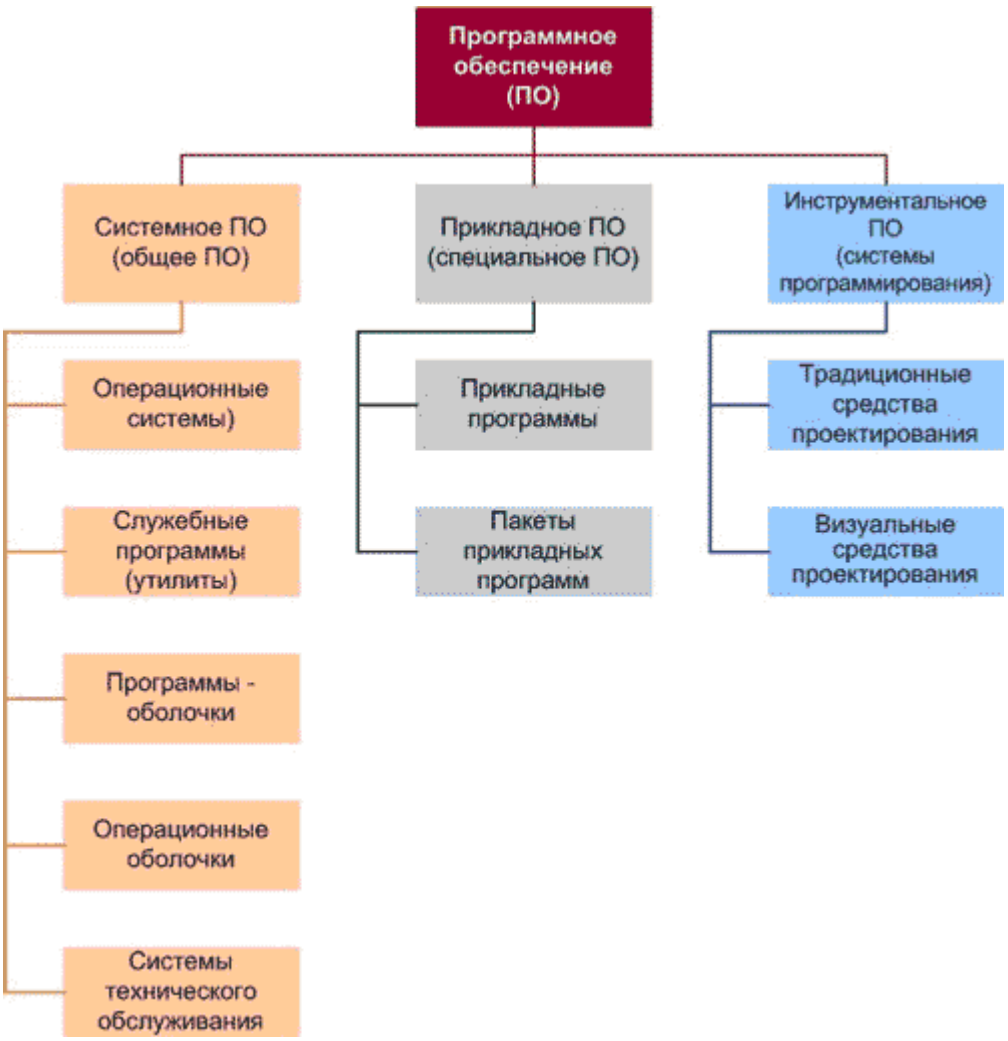


Рис. 1. Классификация программного обеспечения

Системное ПО

Системное программное обеспечение — комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование, выступая как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой — приложения пользователя. В отличие от прикладного программного обеспечения, системное не решает конкретные практические задачи, а лишь обеспечивает работу других программ, предоставляя им сервисные функции, абстрагирующие детали аппаратной и микропрограммной реализации вычислительной системы, управляет аппаратными ресурсами вычислительной системы.

К системному ПО относятся:

- операционные системы;
- драйверы (программы, предназначенные для управления портами периферийных устройств);
- утилиты.

Утилита – вспомогательная компьютерная программа в составе общего программного обеспечения для выполнения специализированных типовых задач, связанных с работой оборудования и операционной системы. Утилиты предоставляют доступ к возможностям (параметрам, настройкам, установкам), недоступным без их применения, либо делают процесс изменения некоторых параметров проще (автоматизируют его).

Утилиты могут входить в состав операционных систем, идти в комплекте со специализированным оборудованием или распространяться отдельно. К утилитам, в частности, относятся:

- утилиты управления процессами
- диспетчеры файлов или файловые менеджеры;
- архиваторы;
- средства динамического сжатия данных;
- средства просмотра и воспроизведения;
- средства контроля и диагностики – позволяют проверить конфигурацию компьютера и проверить работоспособность устройств компьютера, прежде всего жестких дисков;
- утилиты восстановления после сбоев;
- Оптимизатор диска – вид утилиты для оптимизации размещения файлов на дисковом накопителе, например, путём дефрагментации диска;
- деинсталлятор;

- средства обеспечения компьютерной безопасности (резервное копирование, антивирусное ПО).

Необходимо отметить, что часть утилит входит в состав операционной системы, а другая часть функционирует автономно. Большая часть общего (системного) ПО входит в состав ОС. Часть общего ПО входит в состав самого компьютера – часть программ ОС и контролирующих тестов записана в постоянном запоминающем устройстве системной платы. Часть общего ПО относится к автономными программам и поставляется отдельно.

Прикладное ПО

Прикладные программы могут использоваться либо автономно, либо в составе программных комплексов и пакетов.

Пакеты прикладных программ – это система программ, которые по сфере применения делятся на проблемно – ориентированные, пакеты общего назначения и интегрированные пакеты. К прикладному ПО, например, относятся:

- комплект офисных приложений MS OFFICE,
- бухгалтерские системы и системы управления предприятиями,
- CAD – системы (системы автоматизированного проектирования),
- браузеры – средства просмотра Web - страниц,
- графические редакторы,
- редакторы HTML или Web – редакторы,
- системы выполнения математических расчетов и моделирования,
- экспертные системы.

Инструментальное ПО

Инструментальное ПО или системы программирования – это системы для автоматизации разработки новых программ на языках программирования.

В самом общем случае для создания программы на выбранном языке нужно иметь следующие компоненты:

1. Текстовый редактор для создания файла с исходным текстом программы.
2. Компилятор или интерпретатор. Исходный текст с помощью программы-компилятора переводится в промежуточный объектный код. Исходный текст большой программы состоит из нескольких *модулей* (файлов с исходными текстами). Каждый модуль компилируется в отдельный файл с объектным кодом, которые затем надо объединить в одно целое.
3. Редактор связей или сборщик, который выполняет связывание объектных модулей и формирует на выходе работоспособное

приложение – исполнимый код. Исполнимый код – это законченная программа, которую можно запустить на любом компьютере, где установлена операционная система, для которой эта программа создавалась. Как правило, итоговый файл имеет расширение .EXE или .COM.

Операционная система, ее назначение и состав

Современный компьютер представляет собой сложную систему. Компьютер состоит из одного или нескольких процессоров, оперативной памяти, дисков, принтера, клавиатуры, мыши, дисплея, сетевых интерфейсов и других разнообразных устройств ввода-вывода. Если каждому программисту, создающему прикладную программу, нужно будет разбираться во всех тонкостях работы всех этих устройств, то он не напишет ни строчки кода. Более того, управление всеми этими компонентами и их оптимальное использование представляет собой очень непростую задачу. По этой причине компьютеры оснащены специальным уровнем программного обеспечения, который называется операционной системой, в чью задачу входит управление пользовательскими программами, а также всеми упомянутыми ресурсами.

Программы, с которыми непосредственно взаимодействуют пользователи, обычно называемые оболочкой, когда они основаны на применении текста, и графическим пользовательским интерфейсом (Graphical User Interface (GUI)), когда в них используются значки, фактически не являются частью операционной системы, хотя задействуют эту систему в своей работе. Схематично основные рассматриваемые здесь компоненты представлены на рис. 2.

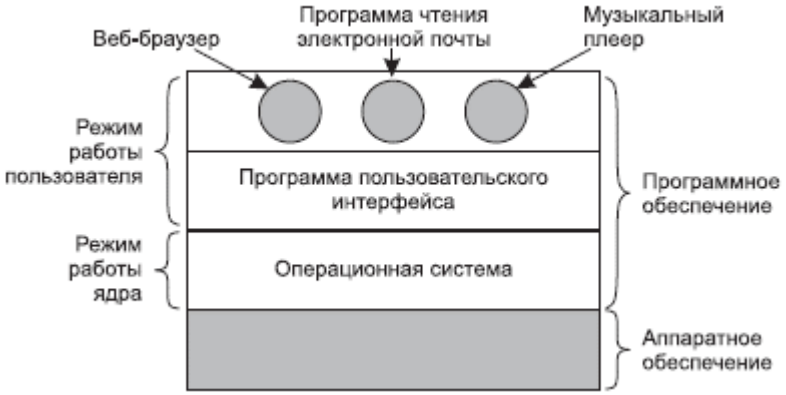


Рис. 2. Место операционной системы в структуре программного обеспечения

В нижней части рисунка показано аппаратное обеспечение. Оно состоит из микросхем, плат, дисков, клавиатуры, монитора и других физических объектов. Над аппаратным обеспечением находится программное обеспечение. Большинство компьютеров имеют два режима работы: режим

ядра и режим пользователя. Операционная система — наиболее фундаментальная часть программного обеспечения, работающая в режиме ядра (этот режим называют еще режимом супервизора). В этом режиме она имеет полный доступ ко всему аппаратному обеспечению и может задействовать любую инструкцию, которую машина в состоянии выполнить. Вся остальная часть программного обеспечения работает в режиме пользователя, в котором доступно лишь подмножество инструкций машины. В частности, программам, работающим в режиме пользователя, запрещено использование инструкций, управляющих машиной или осуществляющих операции ввода-вывода (Input/Output — I/O).

Программы пользовательского интерфейса — оболочка или GUI — находятся на самом низком уровне программного обеспечения, работающего в режиме пользователя, и позволяют пользователю запускать другие программы, такие как веб-браузер, программа чтения электронной почты или музыкальный плеер. Эти программы также активно пользуются операционной системой.

Местонахождение операционной системы показано на рис. 2. Она работает непосредственно с аппаратным обеспечением и является основой остального программного обеспечения. Важное отличие операционной системы от обычного (работающего в режиме пользователя) программного обеспечения состоит в следующем: если пользователь недоволен конкретной программой чтения электронной почты, то он может выбрать другую программу или, если захочет, написать собственную программу, но не может написать собственный обработчик прерываний системных часов, являющийся частью операционной системы и защищенный на аппаратном уровне от любых попыток внесения изменений со стороны пользователя.

Во многих системах также есть программы, работающие в режиме пользователя, но помогающие работе операционной системы или выполняющие особые функции. К примеру, довольно часто встречаются программы, позволяющие пользователям изменять их пароли. Они не являются частью операционной системы и не работают в режиме ядра, но они выполняют важную функцию и должны быть особым образом защищены. В некоторых системах эта идея доведена до крайней формы, и те области, которые традиционно относились к операционной системе (например, файловая система), работают в пространстве пользователя. В таких системах трудно провести четкую границу. Все программы, работающие в режиме ядра, безусловно, являются частью операционной системы, но некоторые программы, работающие вне этого режима, возможно, также являются ее частью или, по крайней мере, имеют с ней тесную связь.

Операционные системы отличаются от пользовательских программ (то есть приложений) не только местоположением. Их особенности – довольно большой объем, сложная структура и длительные сроки использования. Исходный код основы операционной системы типа Linux или Windows занимает порядка 5 млн строк. И это касается только той части, которая работает в режиме ядра. При включении необходимых общих библиотек объем Windows превышает 70 млн строк кода, не считая основных прикладных программ (таких, как Windows Explorer, Windows Media Player и т. д.). Поэтому операционные системы живут достаточно долго, – их очень трудно создавать. После создания одной такой системы нет смысла ее выбрасывать и приступать к созданию новой. Поэтому операционные системы развиваются в течение долгого периода времени. Семейство Windows 95/98/Me по своей сути представляло одну операционную систему, а семейство Windows NT/2000/XP/Vista/Windows 7 – другую. Для пользователя они были похожи друг на друга, поскольку Microsoft позаботилась о том, чтобы пользовательский интерфейс Windows 2000/XP/ Vista/Windows 7 был очень похож на ту систему, которой он шел на замену, а чаще всего это была Windows 98. Другим примером является операционная система UNIX, ее варианты и клоны. Она также развивалась в течение многих лет, существуя в таких базирующихся на исходной системе версиях, как System V, Solaris и FreeBSD. А вот Linux имеет новую программную основу, хотя ее модель весьма близка к UNIX и она обладает высокой степенью совместимости с этой системой.

Функции операционной системы

Дать точное определение операционной системы довольно трудно. Можно сказать, что это программное обеспечение, которое работает в режиме ядра. Операционные системы осуществляют две значительно отличающиеся друг от друга функции: предоставляют прикладным программистам (и прикладным программам, естественно) вполне понятный абстрактный набор ресурсов взамен неупорядоченного набора аппаратного обеспечения, а также управляют этими ресурсами. Рассмотрим эти функции.

Абстрактные ресурсы

Архитектура большинства компьютеров (система команд, организация памяти, ввод-вывод данных и структура шин) на уровне машинного языка слишком примитивна и неудобна для использования в программах, особенно это касается систем ввода-вывода. Поэтому оборудованием, например жестким диском, занимается не пользовательская программа (для чего в каждую пользовательскую программу необходимо было бы вводить специальные сложные блоки программных кодов), а та часть системного

программного обеспечения, которая называется драйвером диска и предоставляет, не вдаваясь в детали, интерфейс для чтения и записи дисковых блоков. Операционные системы содержат множество драйверов для управления устройствами ввода-вывода.

Но для большинства приложений слишком низким является даже этот уровень. Поэтому все операционные системы предоставляют еще один уровень абстракции для использования дисков — файлы. Используя эту абстракцию, программы могут создавать, записывать и читать файлы, не вникая в подробности реальной работы оборудования.

Эта абстракция является ключом к управлению сложностью. Хорошая абстракция превращает практически неподъемную задачу в две, решить которые вполне по силам. Первая из этих задач состоит в определении и реализации абстракций, а вторая — в использовании этих абстракций для решения текущей проблемы. Одна из абстракций, понятная практически любому пользователю компьютера, — это уже упомянутый ранее файл. Он представляет собой полезный объем информации, скажем, цифровую фотографию, сохраненное сообщение электронной почты или веб-страницу. Работать с фотографиями, сообщениями электронной почты и веб-страницами намного легче, чем с особенностями SATA-дисков (или других дисковых устройств). Задача операционной системы заключается в создании хорошей абстракции, а затем в реализации абстрактных объектов, создаваемых в рамках этой абстракции, и управлении ими.

Существующие процессоры, блоки памяти, диски и другие компоненты представляют собой слишком сложные устройства, предоставляющие трудные, неудобные, не похожие друг на друга и не обладающие постоянством интерфейсы. Одна из главных задач операционной системы – скрыть аппаратное обеспечение и существующие программы под создаваемыми взамен них и приспособленными для нормальной работы неизменными абстракциями. Следует отметить, что в действительности «заказчиками» операционных систем являются прикладные программы (разумеется, не без помощи прикладных программистов). Именно они непосредственно работают с операционной системой и ее абстракциями. А конечные пользователи работают с абстракциями, предоставленными пользовательским интерфейсом, – это или командная строка оболочки, или графический интерфейс. Абстракции пользовательского интерфейса могут быть похожими на абстракции, предоставляемые операционной системой, но так бывает не всегда. Чтобы пояснить это положение, рассмотрим обычный рабочий стол Windows и командную строку. И то и другое — программы, работающие под управлением операционной системы Windows и использующие

предоставленные этой системой абстракции, но они предлагают существенно отличающиеся друг от друга пользовательские интерфейсы.

Операционная система в качестве менеджера ресурсов

Представление о том, что операционная система главным образом предоставляет абстракции для прикладных программ, — это взгляд сверху вниз. Сторонники альтернативного взгляда, снизу вверх, придерживаются того мнения, что операционная система существует для управления всеми частями сложной системы. Современные компьютеры состоят из процессоров, памяти, таймеров, дисков, манипуляторов, сетевых интерфейсов, принтеров и широкого спектра других устройств. Сторонники взгляда снизу вверх считают, что задача операционной системы заключается в обеспечении упорядоченного и управляемого распределения процессоров, памяти и устройств ввода-вывода между различными программами, претендующими на их использование.

Современные операционные системы допускают одновременную работу нескольких программ. Представьте себе, что будет, если все три программы, работающие на одном и том же компьютере, попытаются распечатать свои выходные данные одновременно на одном и том же принтере. Первые несколько строчек распечатки могут быть от программы № 1, следующие несколько строчек — от программы № 2, затем несколько строчек от программы № 3 и т. д. В результате получится полный хаос. Операционная система призвана навести порядок в потенциально возможном хаосе за счет буферизации на диске всех выходных данных, предназначенных для принтера. После того как одна программа закончит свою работу, операционная система сможет скопировать ее выходные данные с файла на диске, где они были сохранены, на принтер, а в то же самое время другая программа может продолжить генерацию данных, не замечая того, что выходные данные фактически (до поры до времени) не попадают на принтер.

Когда с компьютером (или с сетью) работают несколько пользователей, потребности в управлении и защите памяти, устройств ввода-вывода и других ресурсов значительно возрастают, поскольку иначе пользователи будут мешать друг другу работать. Кроме этого, пользователям часто требуется совместно использовать не только аппаратное обеспечение, но и информацию (файлы, базы данных и т. п.). Первичной задачей операционной системы является отслеживание того, какой программой какой ресурс используется, чтобы удовлетворять запросы на использование ресурсов, нести ответственность за их использование и принимать решения по конфликтующим запросам от различных программ и пользователей.

Управление ресурсами включает в себя **мультиплексирование** (распределение) ресурсов двумя различными способами: во времени и в пространстве. Когда ресурс разделяется во времени, различные программы или пользователи используют его по очереди: сначала ресурс получают в пользование одни, потом другие и т. д. К примеру, располагая лишь одним центральным процессором и несколькими программами, стремящимися на нем выполняться, операционная система сначала выделяет центральный процессор одной программе, затем, после того как она уже достаточно поработала, центральный процессор получает в свое распоряжение другая программа, затем еще одна программа, и, наконец, его опять получает в свое распоряжение первая программа. Определение того, как именно ресурс будет разделяться во времени — кто будет следующим потребителем и как долго, — это задача операционной системы. Другим примером мультиплексирования во времени может послужить совместное использование принтера. Когда в очереди для распечатки на одном принтере находятся несколько заданий на печать, нужно принять решение, какое из них будет выполнено следующим.

Другим видом разделения ресурсов является пространственное разделение. Вместо поочередной работы каждый клиент получает какую-то часть разделяемого ресурса. Например, оперативная память обычно делится среди нескольких работающих программ, так что все они одновременно могут постоянно находиться в памяти (например, используя центральный процессор по очереди). При условии, что памяти достаточно для хранения более чем одной программы, эффективнее разместить в памяти сразу несколько программ, чем выделять всю память одной программе, особенно если ей нужна лишь небольшая часть от общего пространства. Разумеется, при этом возникают проблемы равной доступности, обеспечения безопасности и т. д., и их должна решать операционная система. Другим ресурсом с разделяемым пространством является жесткий диск. На многих системах на одном и том же диске могут одновременно храниться файлы, принадлежащие многим пользователям. Распределение дискового пространства и отслеживание того, кто какие дисковые блоки использует, — это типичная задача операционной системы по управлению ресурсами.

Контрольные вопросы по теме

Уровень модуля

Уровень курса

1. Структура программного обеспечения компьютерно-интегрированных систем.
2. Системное программное обеспечение, его назначение и состав.
3. Прикладное и инструментальное программное обеспечение.
4. Операционная система, ее назначение и состав.
5. Понятие абстрактных ресурсов.
6. Операционная система в качестве менеджера ресурсов.

Лекция №3

Тема: Операционные системы. (Часть вторая)

Оглавление

Источники:

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.: ил.

https://rutracker.org/forum/viewtopic.php?t=4956359

2. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил.
3. Бабак В.П. Теоретические основы информационно-измерительных систем: Учебник / В.П. Бабак, С.В. Бабак, В.С. Ерёменко и др. – К., 2014. – 832 с.

Функции операционной системы

Дать точное определение операционной системы довольно трудно. Можно сказать, что это программное обеспечение, которое работает в режиме ядра. Операционные системы осуществляют две значительно отличающиеся друг от друга функции: предоставляют прикладным программистам (и прикладным программам, естественно) вполне понятный абстрактный набор ресурсов взамен неупорядоченного набора аппаратного обеспечения, а также управляют этими ресурсами. Рассмотрим эти функции.

Абстрактные ресурсы

Архитектура большинства компьютеров (система команд, организация памяти, ввод-вывод данных и структура шин) на уровне машинного языка слишком примитивна и неудобна для использования в программах, особенно это касается систем ввода-вывода. Поэтому оборудованием, например жестким диском, занимается не пользовательская программа (для чего в каждую пользовательскую программу необходимо было бы вводить специальные сложные блоки программных кодов), а та часть системного программного обеспечения, которая называется драйвером диска и предоставляет, не вдаваясь в детали, интерфейс для чтения и записи дисковых блоков. Операционные системы содержат множество драйверов для управления устройствами ввода-вывода.

Но для большинства приложений слишком низким является даже этот уровень. Поэтому все операционные системы предоставляют еще один уровень абстракции для использования дисков — файлы. Используя эту абстракцию, программы могут создавать, записывать и читать файлы, не вникая в подробности реальной работы оборудования.

Эта абстракция является ключом к управлению сложностью. Хорошая абстракция превращает практически неподъемную задачу в две, решить которые вполне по силам. Первая из этих задач состоит в определении и реализации абстракций, а вторая — в использовании этих абстракций для решения текущей проблемы. Одна из абстракций, понятная практически любому пользователю компьютера, — это уже упомянутый ранее файл. Он представляет собой полезный объем информации, скажем, цифровую фотографию, сохраненное сообщение электронной почты или веб-страницу. Работать с фотографиями, сообщениями электронной почты и веб-страницами намного легче, чем с особенностями SATA-дисков (или других дисковых устройств). Задача операционной системы заключается в создании хорошей абстракции, а затем в реализации абстрактных объектов, создаваемых в рамках этой абстракции, и управлении ими.

Существующие процессоры, блоки памяти, диски и другие компоненты представляют собой слишком сложные устройства, предоставляющие трудные, неудобные, не похожие друг на друга и не обладающие постоянством интерфейсы. Одна из главных задач операционной системы – скрыть аппаратное обеспечение и существующие программы под создаваемыми взамен них и приспособленными для нормальной работы неизменными абстракциями. Следует отметить, что в действительности «заказчиками» операционных систем являются прикладные программы (разумеется, не без помощи прикладных программистов). Именно они непосредственно работают с операционной системой и ее абстракциями. А конечные пользователи работают с абстракциями, предоставленными пользовательским интерфейсом, – это или командная строка оболочки, или графический интерфейс. Абстракции пользовательского интерфейса могут быть похожими на абстракции, предоставляемые операционной системой, но так бывает не всегда. Чтобы пояснить это положение, рассмотрим обычный рабочий стол Windows и командную строку. И то и другое — программы, работающие под управлением операционной системы Windows и использующие предоставленные этой системой абстракции, но они предлагают существенно отличающиеся друг от друга пользовательские интерфейсы.

Операционная система в качестве менеджера ресурсов

Представление о том, что операционная система главным образом предоставляет абстракции для прикладных программ, — это взгляд сверху вниз. Сторонники альтернативного взгляда, снизу вверх, придерживаются того мнения, что операционная система существует для управления всеми частями сложной системы. Современные компьютеры состоят из процессоров, памяти, таймеров, дисков, манипуляторов, сетевых интерфейсов, принтеров и широкого спектра других устройств. Сторонники взгляда снизу вверх считают, что задача операционной системы заключается в обеспечении упорядоченного и управляемого распределения процессоров, памяти и устройств ввода-вывода между различными программами, претендующими на их использование.

Современные операционные системы допускают одновременную работу нескольких программ. Представьте себе, что будет, если все три программы, работающие на одном и том же компьютере, попытаются распечатать свои выходные данные одновременно на одном и том же принтере. Первые несколько строчек распечатки могут быть от программы № 1, следующие несколько строчек — от программы № 2, затем несколько строчек от программы № 3 и т. д. В результате получится полный хаос. Операционная система призвана навести порядок в потенциально возможном

хаосе за счет буферизации на диске всех выходных данных, предназначенных для принтера. После того как одна программа закончит свою работу, операционная система сможет скопировать ее выходные данные с файла на диске, где они были сохранены, на принтер, а в то же самое время другая программа может продолжить генерацию данных, не замечая того, что выходные данные фактически (до поры до времени) не попадают на принтер.

Когда с компьютером (или с сетью) работают несколько пользователей, потребности в управлении и защите памяти, устройств ввода-вывода и других ресурсов значительно возрастают, поскольку иначе пользователи будут мешать друг другу работать. Кроме этого, пользователям часто требуется совместно использовать не только аппаратное обеспечение, но и информацию (файлы, базы данных и т. п.). Первичной задачей операционной системы является отслеживание того, какой программой какой ресурс используется, чтобы удовлетворять запросы на использование ресурсов, нести ответственность за их использование и принимать решения по конфликтующим запросам от различных программ и пользователей.

Управление ресурсами включает в себя **мультиплексирование** (распределение) ресурсов двумя различными способами: во времени и в пространстве. Когда ресурс разделяется во времени, различные программы или пользователи используют его по очереди: сначала ресурс получают в пользование одни, потом другие и т. д. К примеру, располагая лишь одним центральным процессором и несколькими программами, стремящимися на нем выполняться, операционная система сначала выделяет центральный процессор одной программе, затем, после того как она уже достаточно поработала, центральный процессор получает в свое распоряжение другая программа, затем еще одна программа, и, наконец, его опять получает в свое распоряжение первая программа. Определение того, как именно ресурс будет разделяться во времени — кто будет следующим потребителем и как долго, — это задача операционной системы. Другим примером мультиплексирования во времени может послужить совместное использование принтера. Когда в очереди для распечатки на одном принтере находятся несколько заданий на печать, нужно принять решение, какое из них будет выполнено следующим.

Другим видом разделения ресурсов является пространственное разделение. Вместо поочередной работы каждый клиент получает какую-то часть разделяемого ресурса. Например, оперативная память обычно делится среди нескольких работающих программ, так что все они одновременно могут постоянно находиться в памяти (например, используя центральный процессор по очереди). При условии, что памяти достаточно для хранения более чем одной программы, эффективнее разместить в памяти сразу несколько программ, чем выделять всю память одной программе, особенно если ей

нужна лишь небольшая часть от общего пространства. Разумеется, при этом возникают проблемы равной доступности, обеспечения безопасности и т. д., и их должна решать операционная система. Другим ресурсом с разделяемым пространством является жесткий диск. На многих системах на одном и том же диске могут одновременно храниться файлы, принадлежащие многим пользователям. Распределение дискового пространства и отслеживание того, кто какие дисковые блоки использует, — это типичная задача операционной системы по управлению ресурсами.

Основные понятия и абстракции операционной системы

Большинство операционных систем используют определенные основные понятия и абстракции, такие как процессы, адресные пространства и файлы, которые играют главную роль в осмыслении самих систем.

Процессы

Ключевым понятием во всех операционных системах является процесс. Процессом, по существу, является программа во время ее выполнения. С каждым процессом связано его адресное пространство – список адресов ячеек памяти от нуля до некоторого максимума, откуда процесс может считывать данные и куда может записывать их. Адресное пространство содержит выполняемую программу, данные этой программы и ее стек. Кроме этого, с каждым процессом связан набор ресурсов, который обычно включает регистры(в том числе счетчик команд и указатель стека), список открытых файлов, необработанные предупреждения, список связанных процессов и всю остальную информацию, необходимую в процессе работы программы. Таким образом, процесс – это контейнер, в котором содержится вся информация, необходимая для работы программы.

Для того чтобы выработать интуитивное представление о процессе, рассмотрим систему, работающую в мультипрограммном режиме. Пользователь может запустить программу редактирования видео и указать конвертирование одночасового видеофайла в какой-нибудь определенный формат (процесс займет несколько часов), а затем переключиться на блуждания по Интернету. При этом может заработать фоновый процесс, который периодически «просыпается» для проверки входящей электронной почты. И у нас уже будет (как минимум) три активных процесса: видеоредактор, веб-браузер и программа получения (клиент) электронной почты. Периодически операционная система будет принимать решения остановить работу одного процесса и запустить выполнение другого, возможно, из-за того, что первый исчерпал свою долю процессорного временив предыдущую секунду или две.

Если процесс приостанавливается таким образом, позже он должен возобновиться именно с того состояния, в котором был остановлен. Это означает, что на период приостановки вся информация о процессе должна быть явным образом где-то сохранена. Например, у процесса могут быть одновременно открыты для чтения несколько файлов. С каждым из этих файлов связан указатель текущей позиции (то есть номер байта или записи, которая должна быть считана следующей). Когда процесс приостанавливается, все эти указатели должны быть сохранены, чтобы вызов read, выполняемый после возобновления процесса, приводил к чтению нужных данных. Во многих операционных системах вся информация о каждом процессе, за исключением содержимого его собственного адресного пространства, хранится в таблице операционной системы, которая называется **таблицей процессов** и представляет собой массив (или связанный список) структур, по одной на каждый из существующих на данный момент процессов.

Таким образом, процесс (в том числе приостановленный) состоит из собственного адресного пространства, которое обычно называют образом памяти, и записи в таблице процессов с содержимым его регистров, а также другой информацией, необходимой для последующего возобновления процесса.

Системный вызов – это обращение прикладной программы (процесса) к операционной системе для выполнения какой-либо операции.

Главными системными вызовами, используемыми при управлении процессами, являются вызовы, связанные с созданием и завершением процессов. Рассмотрим простой пример. Процесс, называемый интерпретатором команд, или оболочкой, считывает команды с терминала. Пользователь только что набрал команду, требующую компиляции программы. Теперь оболочка должна создать новый процесс, запускающий компилятор. Когда этот процесс завершит компиляцию, он произведет системный вызов для завершения собственного существования.

Если процесс способен создавать несколько других процессов (называющихся дочерними процессами), а эти процессы в свою очередь могут создавать собственные дочерние процессы, то перед нами предстает дерево процессов, подобное изображенному на рис. 1.

Связанные процессы, совместно работающие над выполнением какой-нибудь задачи, зачастую нуждаются в обмене данными друг с другом и синхронизации своих действий. Такая связь называется межпроцессным взаимодействием. Другие системные вызовы, предназначенные для управления процессом, позволяют запросить выделение дополнительной памяти (или освобождение незадействованной), организовать ожидание

завершения дочернего процесса или загрузку какой-нибудь другой программы поверх своей.

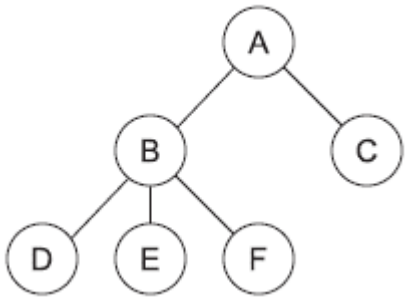


Рис. 1. Дерево процессов. Процесс A создал два дочерних процесса, B и C. Процесс B создал три дочерних процесса, D, E и F

Временами возникает потребность в передаче информации запущенному процессу, который не находится в состоянии ожидания этой информации. Можно привести в пример процесс, который обменивается информацией с другим процессом, запущенным на другом компьютере, и посылает удаленному процессу сообщение по сети. Чтобы застраховаться от возможной утраты сообщения или ответа на него, отправитель может запросить собственную операционную систему уведомить его по истечении определенного интервала времени, чтобы он мог повторно отправить сообщение, если не получит подтверждения его получения раньше. После установки такого таймера программа может продолжить выполнение другой работы.

Когда истечет заданный интервал времени, операционная система посылает процессу сигнал тревоги. Этот сигнал заставляет процесс приостановить выполняемую работу, сохранить в стеке состояние своих регистров и запустить специальную процедуру обработки сигнала тревоги, для того чтобы, к примеру, заново передать предположительно утраченное сообщение. Когда обработчик сигнала завершит свою работу, запущенный процесс возобновится в том самом состоянии, которое было до поступления сигнала. Сигналы являются программными аналогами аппаратных прерываний. Они могут генерироваться в различных ситуациях, а не только по истечении времени, установленного в таймере. Многие аппаратные прерывания (например, выполнение недопустимой команды или обращение по неверному адресу) также транслируются процессу, при выполнении которого произошла ошибка.

Напомним (лекция 4), что *прерывание* (англ. interrupt) – это сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания.

Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние, и выполняя функцию, называемую обработчиком прерывания (или программой обработки прерывания), который реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

Адресные пространства

Каждый компьютер обладает определенным объемом оперативной памяти, используемой для хранения исполняемых программ. В самых простых операционных системах в памяти присутствует только одна программа. Для запуска второй программы сначала нужно удалить первую, а затем на ее место загрузить в память вторую.

Более сложные операционные системы позволяют одновременно находиться в памяти нескольким программам. Чтобы исключить взаимное негативное влияние программ друг на друга из-за неправильного обращения к памяти (одна программа обращается к участку оперативной памяти, где в это время хранятся данные или коды совершенно другой программы или даже самой операционной системы), что приведет к общему сбою, необходим защитный механизм. Этот механизм управляется операционной системой.

Кроме необходимости защиты памяти управление адресным пространством процессов необходимо и по другой причине. Обычно каждому процессу отводится для использования некоторый непрерывный набор адресов, как правило, от нуля и до некоторого максимума. В самом простом случае максимальный объем адресного пространства, выделяемого процессу, меньше объема оперативной памяти. Таким образом, процесс может заполнить свое адресное пространство и для его размещения в оперативной памяти будет достаточно места. При этом на многих компьютерах используется 32- или 64-разрядная адресация, позволяющая иметь адресное пространство размером 2^{32} или 2^{64} байт соответственно. Что произойдет, если адресное пространство процесса превышает объем оперативной памяти, установленной на компьютере, а процессу требуется использовать все свое пространство целиком? На первых компьютерах такой процесс неизменно терпел крах. В наше время, на современных компьютерах, применяется технология виртуальной памяти, которая состоит в том, что операционная система хранит часть адресного пространства в оперативной памяти, а часть – на диске, по необходимости меняя их фрагменты местами. По сути, операционная система создает абстракцию адресного пространства в виде набора адресов, на которые может ссылаться процесс. Адресное пространство

отделено от физической памяти машины и может быть как больше, так и меньше нее. Управление адресными пространствами и физической памятью является важной частью работы операционной системы.

Файлы

Другим ключевым понятием, поддерживаемым практически всеми операционными системами, является файловая система. Основная функция операционной системы – скрыть специфику дисков и других устройств ввода-вывода и предоставить программисту удобную и понятную абстрактную модель, состоящую из независимых от устройств файлов. Вполне очевидно, что для создания, удаления, чтения и записи файлов понадобятся системные вызовы. Перед тем, как файл будет готов к чтению, он должен быть найден на диске и открыт, а после считывания – закрыт. Для проведения этих операций предусмотрены системные вызовы.

Чтобы предоставить место для хранения файлов, многие операционные системы используют каталог как способ объединения файлов в группы. Для создания и удаления каталогов нужны системные вызовы. Они также нужны для помещения в каталог существующего файла и удаления его оттуда. Элементами каталога могут быть либо файлы, либо другие каталоги.

Каждый файл, принадлежащий иерархии каталогов, может быть обозначен своим полным именем с указанием пути к файлу, начиная с вершины иерархии – корневого каталога. Этот абсолютный путь состоит из списка каталогов, которые нужно пройти от корневого каталога, чтобы добраться до файла, где в качестве разделителей компонентов служат символы косой черты (слеша).

В любой момент времени у каждого процесса есть текущий рабочий каталог, относительно которого рассматриваются пути файлов, не начинающиеся с косой черты. Процесс может изменить свой рабочий каталог, воспользовавшись системным вызовом, определяющим новый рабочий каталог. Перед тем как с файлом можно будет работать в режиме записи или чтения, он должен быть открыт. На этом этапе происходит также проверка прав доступа. Если доступ разрешен, система возвращает целое число, называемое дескриптором файла, который используется в последующих операциях. Если доступ запрещен, то возвращается код ошибки.

Ввод-вывод данных

У всех компьютеров имеются физические устройства для получения входной и вывода выходной информации. Существует масса разнообразных устройств ввода-вывода: клавиатуры, мониторы, принтеры и т.д., а также АЦП и ЦАП. Управление всеми этими устройствами возлагается на операционную систему. Поэтому у каждой операционной системы для управления такими

устройствами существует своя подсистема ввода-вывода. Некоторые программы ввода-вывода не зависят от конкретного устройства, то есть в равной мере подходят для применения со многими или со всеми устройствами ввода-вывода. Другая часть программ, например драйверы устройств, предназначена для определенных устройств ввода-вывода.

Безопасность

Компьютеры содержат большой объем информации, и часто пользователям нужно защитить ее и сохранить ее конфиденциальность. Возможно, это электронная почта, бизнес-планы, налоговые декларации и многое другое. Управление безопасностью системы также возлагается на операционную систему: например, она должна обеспечить доступ к файлам только пользователям, имеющим на это право.

Чтобы понять сам замысел возможной организации работы системы безопасности, обратимся в качестве примера к системе UNIX. Файлам в UNIX присваивается 9-разрядный двоичный код защиты. Этот код состоит из трехбитных полей. Одно поле – для владельца, второе – для представителей группы, в которую он входит (разделяет пользователей на группы системный администратор), третье – для всех остальных. В каждом поле есть бит, определяющий доступ для чтения, бит, определяющий доступ для записи, и бит, определяющий доступ для выполнения. Эти три бита называются rwx-битами (read, write, execute). Например, код защиты rwxr-x--x означает, что владельцу доступны чтение, запись или выполнение файла, остальным представителям его группы разрешается чтение или выполнение файла (но не запись), а всем остальным разрешено выполнение файла (но не чтение или запись). Дефис (минус) означает, что соответствующее разрешение отсутствует.

Кроме защиты файлов существует множество других аспектов безопасности. Один из них – это защита системы от нежелательных вторжений как с участием, так и без участия людей (например, путем вирусных атак).

Оболочка

Операционная система представляет собой программу, выполняющую системные вызовы. Не являясь частью операционной системы, оболочка нашла широкое применение как средство доступа ко многим ее функциям. Когда не применяется графический пользовательский интерфейс, она также является основным интерфейсом между пользователем, сидящим за своим терминалом, и операционной системой. Существует множество оболочек, Оболочка запускается после входа в систему любого пользователя. В качестве стандартного устройства ввода и вывода оболочка использует терминал1. Свою работу она начинает с вывода приглашения — знака доллара,

сообщающего пользователю, что оболочка ожидает приема команды. Например, если теперь пользователь наберет на клавиатуре В наши дни на большинстве персональных компьютеров используется графический пользовательский интерфейс. По сути, графический пользовательский интерфейс — это просто программа (или совокупность программ), работающая поверх операционной системы наподобие оболочки. В системах Linux этот факт проявляется явным образом, поскольку у пользователя есть выбор по крайней мере из двух сред, реализующих графический пользовательский интерфейс: Gnome и KDE. Или он может вообще не выбрать ни одну из них, воспользовавшись окном терминала из X11. В Windows также есть возможность заменить стандартный менеджер рабочего стола (Windows Explorer) какой-нибудь другой программой путем внесения изменений в некоторые значения реестра, хотя этой возможностью практически никто не пользуется.

Редакторы, компиляторы, ассемблеры, компоновщики, утилиты и интерпретаторы команд по определению не являются частью операционной системы. Они относятся к инструментальному ПО.

Классы операционных систем

История операционных систем насчитывает уже более полувека. За это время было разработано огромное количество разнообразных операционных систем, но не все они получили широкую известность. В данном разделе вкратце, для сведения, рассмотрим классы операционных систем. Для компьютерно-интегрированных технологий наиболее важное значение имеют операционные системы жесткого реального времени.

Операционные системы мейнфреймов

К высшей категории относятся операционные системы мейнфреймов (больших универсальных машин) — компьютеров, занимающих целые залы и до сих пор еще встречающихся в крупных центрах обработки корпоративных данных. Такие компьютеры отличаются от персональных компьютеров объемами ввода-вывода данных. Мейнфреймы, имеющие тысячи дисков и петабайты данных, – весьма обычное явление. Мейнфреймы также находят применение в качестве мощных веб-серверов, серверов крупных интернет-магазинов и серверов, занимающихся межкорпоративными транзакциями. Операционные системы мейнфреймов ориентированы преимущественно на одновременную обработку множества заданий, большинство из которых требует колоссальных объемов ввода-вывода данных. Работа в режиме разделения времени дает возможность множеству удаленных пользователей одновременно запускать на компьютере свои задания, например запросы к большой базе данных. Все эти функции тесно связаны друг с другом, и

зачастую операционные системы универсальных машин выполняют их в комплексе. Примером операционной системы универсальных машин может послужить OS/390, наследница OS/360. Однако эти операционные системы постепенно вытесняются вариантами операционной системы UNIX, например Linux.

Серверные операционные системы

Чуть ниже по уровню стоят серверные операционные системы. Они работают на серверах, которые представлены очень мощными персональными компьютерами, рабочими станциями или даже универсальными машинами. Они одновременно обслуживают по сети множество пользователей, обеспечивая им общий доступ к аппаратным и программным ресурсам. Серверы могут предоставлять услуги печати, хранения файлов или веб-служб. Интернет-провайдеры для обслуживания своих клиентов обычно задействуют сразу несколько серверных машин. При обслуживании веб-сайтов серверы хранят веб-страницы и обрабатывают поступающие запросы. Типичными представителями серверных операционных систем являются Solaris, FreeBSD, Linux и Windows Server 201x.

Многопроцессорные операционные системы

Сейчас все шире используется объединение множества центральных процессоров в единую систему, что позволяет добиться большой вычислительной мощности. В зависимости от того, как именно происходит это объединение, а также каковы ресурсы общего пользования, эти системы называются параллельными компьютерами, мультикомпьютерами или многопроцессорными системами. Им требуются специальные операционные системы, в качестве которых часто применяются особые версии серверных операционных систем, оснащенные специальными функциями связи, сопряжения и синхронизации. С появлением многоядерных процессоров для персональных компьютеров операционные системы даже обычных настольных компьютеров и ноутбуков стали работать по меньшей мере с небольшой многопроцессорной системой. Со временем число ядер будет только расти. На многопроцессорных системах могут работать многие популярные операционные системы, включая Windows и Linux.

Операционные системы персональных компьютеров

К следующей категории относятся операционные системы персональных компьютеров. Все их современные представители поддерживают многозадачный режим. При этом довольно часто уже в процессе загрузки на одновременное выполнение запускаются десятки программ. Задачей операционных систем персональных компьютеров

является качественная поддержка работы отдельного пользователя. Они широко используются для обработки текстов, создания электронных таблиц, игр и доступа к Интернету. Типичными примерами могут служить операционные системы Linux, FreeBSD, Windows 7, Windows 8, Windows 10 и OS X (macOS) компании Apple..

Операционные системы карманных персональных компьютеров, планшетов и смартфонов

Продолжая двигаться по нисходящей ко все более простым системам, мы дошли до планшетов, смартфонов и других карманных компьютеров. Эти компьютеры, изначально известные как КПК, или PDA (Personal Digital Assistant — персональный цифровой секретарь), представляют собой небольшие компьютеры, которые во время работы держат в руке. Самыми известными их представителями являются смартфоны и планшеты. На этом рынке доминируют операционные системы Android от Google и iOS от Apple, но у них имеется множество конкурентов. Большинство таких устройств обладают многоядерными процессорами, GPS, камерами и другими датчиками, достаточным объемом памяти и сложными операционными системами.

Встроенные операционные системы

Встроенные системы работают на компьютерах, которые управляют различными устройствами. Поскольку на этих системах установка пользовательских программ не предусматривается, их обычно компьютерами не считают. Примерами устройств, где устанавливаются встроенные компьютеры, могут послужить микроволновые печи, телевизоры, автомобили, пишущие DVD, обычные телефоны и MP3-плееры. В основном встроенные системы отличаются тем, что на них ни при каких условиях не будет работать стороннее программное обеспечение. В микроволновую печь невозможно загрузить новое приложение, поскольку все ее программы записаны в ПЗУ. Следовательно, отпадает необходимость в защите приложений друг от друга и операционную систему можно упростить. Наиболее популярными в этой области считаются операционные системы Embedded Linux, QNX и VxWorks.

Операционные системы сенсорных узлов

Сети, составленные из миниатюрных сенсорных узлов, связанных друг с другом и с базовой станцией по беспроводным каналам, развертываются для различных целей. Такие сенсорные сети используются для защиты периметров зданий, охраны государственной границы, обнаружения возгораний в лесу, измерения температуры и уровня осадков в целях составления прогнозов погоды, сбора информации о перемещениях противника на поле боя и многого

другого. Узлы такой сети представляют собой миниатюрные компьютеры, питающиеся от батареи и имеющие встроенную радиосистему. Они ограничены по мощности и должны работать длительный период времени в необслуживаемом режиме на открытом воздухе, часто в сложных климатических условиях. Сеть должна быть достаточно надежной и допускать отказы отдельных узлов, что по мере потери емкости батарей питания будет случаться все чаще. Каждый сенсорный узел является настоящим компьютером, оснащенным процессором, оперативной памятью и постоянным запоминающим устройством, а также одним или несколькими датчиками. На нем работает небольшая, но настоящая операционная система, обычно управляемая событиями и откликающаяся на внешние события или периодически производящая измерения по сигналам встроенных часов. Операционная система должна быть небольшой по объему и несложной, поскольку основными проблемами этих узлов являются малая емкость оперативной памяти и ограниченное время работы батарей. Так же как и у встроенных систем, все программы являются предварительно загруженными, и пользователи не могут запустить программу, загруженную из Интернета, что значительно упрощает всю конструкцию. Примером широко известной операционной системы для сенсорных узлов может послужить TinyOS.

Операционные системы реального времени

Еще одна разновидность операционных систем — это системы реального времени. Эти системы характеризуются тем, что время для них является ключевым параметром. Например, в системах управления производственными процессами компьютеры, работающие в режиме реального времени, должны собирать сведения о процессе и использовать их для управления станками на предприятии. Довольно часто они должны отвечать очень жестким временным требованиям. Например, когда автомобиль перемещается по сборочному конвейеру, то в определенные моменты времени должны осуществляться вполне конкретные операции. Если, к примеру, сварочный робот приступит к сварке с опережением или опозданием, машина придет в негодность. Если операция должна быть проведена точно в срок (или в определенный период времени), то мы имеем дело с системой **жесткого реального времени**. Множество подобных систем встречается при управлении производственными процессами, в авиационно-космическом электронном оборудовании, в военной и других подобных областях применения. Эти системы должны давать абсолютные гарантии того, что определенные действия будут осуществляться в конкретный момент времени. Системами жесткого реального времени являются такие системы как QNX, RTOS, VxWorks.

Другой разновидностью подобных систем является система мягкого реального времени, в которой хотя и нежелательно, но вполне допустимо несоблюдение срока какого-нибудь действия, что не наносит непоправимого вреда. К этой категории относятся цифровые аудио- или мультимедийные системы. Смартфоны также являются системами мягкого реального времени.

Поскольку к системам реального времени предъявляются очень жесткие требования, иногда операционные системы представляют собой простую библиотеку, сопряженную с прикладными программами, где все тесно взаимосвязано и между частями системы не существует никакой защиты. Примером такой системы может послужить eCos. Категории операционных систем для КПК, встроенных систем и систем реального времени в значительной степени перекрываются друг с другом по свойственным им признакам. Практически все они имеют по крайней мере некоторые аспекты систем мягкого реального времени. Встроенные системы и системы реального времени работают только с тем программным обеспечением, которое вложили в них разработчики этих систем; пользователи не могут добавить в этот арсенал собственное программное обеспечение, что существенно облегчает решение задач защиты. КПК и встроенные системы предназначены для индивидуальных потребителей, а системы реального времени чаще используются в промышленном производстве. Тем не менее, несмотря на все это, у них есть определенное количество общих черт.

Операционные системы смарт-карт

Самые маленькие операционные системы работают на смарт-картах. Смарт-карта представляет собой устройство размером с кредитную карту, имеющее собственный процессор. На операционные системы для них накладываются очень жесткие ограничения по требуемой вычислительной мощности процессора и объему памяти. Некоторые из смарт-карт получают питание через контакты считывающего устройства, в которое вставляются, другие — бесконтактные смарт-карты — получают питание за счет эффекта индукции, что существенно ограничивает их возможности. Некоторые из них способны справиться с одной-единственной функцией, например с электронными платежами, но существуют и многофункциональные смарт-карты.

Контрольные вопросы по теме

Уровень модуля

Уровень курса

1. Понятие абстрактных ресурсов.
2. Операционная система в качестве менеджера ресурсов.
3. Понятие "процесс" в операционных системах.
4. Понятие "адресное пространство" в операционных системах.
5. Понятие "файл" в операционных системах.
6. Оболочка операционной системы.
7. Классы операционных систем.
8. Операционные системы реального времени.

Лекция №4

Тема: Операционные системы реального времени

Оглавление

Источники:

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил.
2. Столлингс В. Операционные системы: внутренняя структура и принципы проектирования, 9-е изд.: Пер. с англ. – СПб.: ООО “Диалектика”, 2020. – 1264 с.: ил.

Особенности операционных систем реального времени

Вспомним, что операционной системой называют комплекс программ, обеспечивающий управление ресурсами аппаратно-программного комплекса (вычислительной системы) и процессами, использующими эти ресурсы при вычислениях. Ресурсом в данном контексте является любой логический или физический (и в совокупности) компонент вычислительной системы или аппаратно-программного комплекса и предоставляемые им возможности.

Основными ресурсами являются процессор (процессорное время), оперативная память и периферийные устройства.

Управление ресурсами сводится к выполнению следующих задач: упрощение доступа к ресурсам, распределение их между процессами.

Решение первой задачи позволяет "спрятать" аппаратные особенности вычислительной системы, и тем самым предоставить в распоряжение пользователю или программисту виртуальную машину с существенно облегченным управлением. Таким образом, ОС поддерживает следующие интерфейсы: пользовательский (командный язык для управления функционированием системы и набор сервисных услуг); программный (набор услуг, освобождающий программиста от кодирования рутинных операций).Функция распределения ресурсов является одной из наиболее важных задач, решаемых ОС, однако она присуща не всем ОС, а только тем, которые обеспечивают одновременное выполнение нескольких программ (процессов).

Процессом называется последовательность действий, предписанных программой или ее логически законченной частью, а также данные, используемые при вычислениях. Процесс является минимальной единицей работы, для которой выделяются ресурсы.

В настоящее время существует большое разнообразие ОС, которые классифицируются по следующим признакам:

- ✓ количество пользователей, одновременно обслуживаемых системой;
- ✓ число процессов, которые могут одновременно выполняться под управлением ОС;
- ✓ тип доступа пользователя к системе;
- ✓ тип аппаратно-программного комплекса.

В соответствии с первым признаком различаются одно- и многопользовательские ОС.

Второй признак делит ОС на одно- и многозадачные.

В соответствии с третьим признаком ОС делятся на:

- ➢ системы с пакетной обработкой. В этом случае из программ, подлежащих выполнению, формируется пакет, который

предъявляется системе для обработки. В этом случае пользователи непосредственно с ОС не взаимодействуют;

- системы разделения времени, обеспечивающие одновременный интерактивный доступ к вычислительной системе нескольких пользователей через терминалы. При этом ресурсы системы выделяются каждому пользователю "по очереди", в соответствии с той или иной дисциплиной обслуживания;
- системы реального времени, которые должны обеспечивать гарантированное время ответа на внешние события (более подробно см. ниже).

Четвертый признак делит ОС на одно- и многопроцессорные, сетевые и распределенные. Для многопользовательских и многозадачных ОС важным показателем является дисциплина обслуживания. В соответствии с этим различают вытесняющий и согласующий режимы многозадачной работы. При *вытесняющей организации* выделением задачам процессорного времени занимается только ОС (например, для каждой задачи процессор выделяется по очереди, причем на строго фиксированный промежуток времени, но возможно и приоритетное обслуживание). В случае *согласующей организации* каждая задача, получив управление, сама определяет, когда ей "отдать" процессор другой задаче. В общем случае согласование эффективнее и надежнее вытеснения, но определяющим фактором при реализации программ становится тот факт, что данная программа не должна монопольно использовать процессорное время.

Система реального времени (СРВ) – это система, правильность функционирования которой зависит не только от логической корректности вычислений, но и от времени, за которое эти вычисления производятся.

Для событий, происходящих в такой системе, важно время, когда эти события происходят, и их логическая корректность. Система работает в реальном времени, если ее быстродействие адекватно скорости протекания физических процессов на объектах контроля или управления (имеются в виду процессы, непосредственно связанные с функциями, выполняемыми конкретной системой реального времени). Система управления должна собрать данные, произвести их обработку по заданным алгоритмам и выдать управляющее воздействие за такой промежуток времени, который обеспечивает успешное выполнение поставленных задач.

Операционные системы, предназначенные для обеспечения работы систем реального времени, называются **операционными системами реального времени ОСРВ** (англ. RTOS – Real Time Operation system)/

Назначение систем реального времени

В настоящее время задачи управления, требующие взаимодействия управляющей системы со сложными технологическими процессами или техническими объектами решаются исключительно при помощи компьютеров. С точки зрения управляющей системы все события в этих технологических процессах происходят в произвольные моменты времени, образно говоря: когда им захочется. Компьютер же может выполнять лишь конечное число операций в конечное время, поэтому возникает вопрос: а успеет ли компьютер с нужной скоростью обсчитать ситуацию и выдать конкретные управляющие действия, которые необходимы именно в строго определенный момент времени. Проблемы подобного рода появились вследствие наличия быстроменяющихся процессов в современном производстве. Ясно, что сигналы в природе распространяются с конечной скоростью, скорость работы тоже конечна, поэтому мгновенных действий (вызванных неким событием) от компьютера ожидать принципиально невозможно. Ведь каким бы современным (читай: мощным по производительности, т.е. высокой скоростью обработки команд и операций) компьютер бы ни был – ему физически нужны хотя бы доли секунды, чтобы выполнить небольшую простую группу команд, а иногда и эти доли секунды представляют собой недопустимо большой интервал времени – реакция системы управления на какое-то событие может оказаться запоздалой. На практике задачи, безусловно, допускают некоторое запаздывание действий, и, если система имеет время реакции меньшее, чем эта допустимая задержка, то такую систему называют *системой реального времени*. Так как в природе разные процессы протекают с разной скоростью, одна и таже система может укладываться в заданные рамки для одного процесса и не укладываться для другого. Таким образом, о системе реального времени имеет смысл говорить применительно к конкретной задаче. Например, чтобы построить зависимость средней температуры воздуха за день от дня недели в качестве системы реального времени подойдет практически любой компьютер с практически любым ПО. Если же речь идет о посадке самолета, когда роль играют даже миллисекунды, то выбор аппаратного и программного обеспечения, способных обеспечить решение задачи посадки самолета, приобретает критически важное значение.

В общем случае для компьютеризированной системы управления задача наблюдения и управления динамическим процессом сводится к непрерывному обмену сигналами с внешним миром. Компьютер – дискретная система, поэтому обмен сигналами может происходить только в дискретные моменты времени, то есть через заданные промежутки времени, считая, что в эти малые промежутки времени внешний мир остается неизменным. Если наша система

способна обрабатывать информацию и выдавать управляющие сигналы с требуемой частотой, то ее можно назвать системой реального времени. Можно сказать и так: обмен сигналами производится сразу же после наступления события, состоящего в том, что завершился очередной интервал времени и наступил новый. **Время реакции должно быть меньше времени дискретизации процесса**. Это – важнейшее требование, когда речь идет о системах реального времени. Следует отметить, что неудовлетворительная по запаздыванию работа системы при управлении процессами определенного рода может привести к фатальным последствиям. В то время как при управлении процессами другого рода такое же запаздывание не приведет ни к каким внештатным и даже ни к каким нежелательным ситуациям. Например: если система измерения температуры из описанного выше примера случайно опоздает на непозволительное время, то это значит, что мы просто изменили выборку точек съема температуры, но все равно получили правильный результат. Если же при внезапном порыве ветра автомат захода на посадку случайно задержится хотя бы на одну секунду, самолет может не попасть на полосу. Таким образом, следует делить системы на системы жесткого и мягкого реального времени.

Системой жесткого реального времени называется система, где неспособность обеспечить реакцию на какие-либо события в заданное время является отказом и ведет к невозможности решения поставленной задачи. Время реакции в системах жесткого реального времени должно быть минимальным. Большинство систем жесткого реального времени являются системами контроля и управления. Такие СРВ сложны в реализации, так как к ним предъявляются особые требования в вопросах безопасности.

Точного определения **мягкого реального времени** не существует, поэтому можно отнести сюда все СРВ, не подпадающие под категорию жестких. Так, система мягкого реального времени может не успевать все делать в заданное время, поэтому возникает проблема определения критериев успешности (нормальности) ее функционирования.

Системы жесткого реального времени в свою очередь можно разделить на системы специализированные и универсальные.

Специализированная СРВ – система, где конкретные временные требования изначально определены. Такая система должна быть специально спроектирована для удовлетворения этих требований.

Универсальная СРВ должна уметь выполнять произвольные (заранее неопределенные) временные задачи без применения специальной техники. Разработка таких систем является самой сложной задачей, хотя обычно, требования, предъявляемые к таким системам, мягче, чем требования к специализированным системам.

Основные требования к ОСРВ

Системы реального времени в связи с особенностями их применения должны отвечать следующим основным требованиям (в дополнение к тем требованиям, которые предъявляются к обычным операционным системам):

- возможность параллельного выполнения нескольких задач;
- предсказуемость;
- ограниченное и заранее известное максимальное время отклика на событие;
- повышенные требования к надежности и безопасности управления;
- возможность безотказной работы в течение длительного времени.

Условия применения ОСРВ

Условия применения ОСРВ характеризуются следующими основными факторами:

- ✓ технические объекты, подвергаемые управлению, чаще всего представляют собой сложные системы, состоящие из большого числа компонентов;
- ✓ технические объекты, подвергаемые управлению, чаще всего представляют собой распределенные системы;
- ✓ системы управления включают в себя большое количество датчиков, взаимодействие с аппаратурой происходит с использованием разнообразных интерфейсов, а объем передаваемых данных велик;
- ✓ успешное выполнение задач управления зависит от времени реакции системы;
- ✓ сложность тестирования.

СРВ должны реагировать на различные типы внутренних и внешних событий (периодических и непериодических). Необходимо отметить, что принадлежность системы к классу СРВ никак не связана с ее быстродействием. Исходные требования к времени реакции системы и другим временным параметрам определяются или техническим заданием на систему, или просто логикой ее функционирования. Интуитивно понятно, что быстродействие СРВ должно быть тем больше, чем больше скорость протекания процессов на объекте контроля и управления.

Применяемость ОС в качестве ОСРВ

Классические ОС: Win NT, Linux и Unix.

В качестве операционной системы реального времени могут быть использованы классические ОС с расширением RT, которое указывает на его

принадлежность к ОС реального времени – **R**eal **T**ime (Win NT - RTX, RT Linux, RT Unix)

Коммерческие ОСРВ

Существует целый класс операционных систем, разработанных специально для их применения в системах реального времени. Примером могут служить такие системы как QNX, VxWorks, OS9 и другие. Необходимо отметить, что такие системы достаточно дороги. Например, стоимость полного пакета ОС VxWorks (Tornado 1.0) в 2002 году составляла около 15 000 долларов США. Однако с годами подобные системы дешевеют – на сегодня стоимость упомянутой ОС составляет около 10 000 долларов (Tornado версии 2.0 и выше – полная стоимость зависит от выбираемых компонентов).

Для более детального рассмотрения возможностей ОСРВ представлены ориентировочные цифры, дающее представление о порядке времени реакции и подходящих операционных системах. Данная таблица сформирована на основании экспериментальных данных, полученных на базе вычислительных комплексов, построенных на основе процессоров Intel 80486DX. Безусловно, данный процессор на сегодняшний день является устаревшим, но можно сделать выводы об уровне реакции на внешние события различных систем

Требуемое время реакции	***Рекомендуемые к использованию ОС***
Менее 10 мкс	Только ОСРВ, но даже они могут быть бессильны
10 - 100 мкс	Операционные системы реального времени
100 мкс - 1 мс	ОСРВ, RTAI, RT- UNIX и LINUX, расширения реального времени для Windows NT, CE
1 мс	Можно пытаться делать что-то с Linux и Windows NT, но не для систем, где опоздания реакции могут привести к тяжелым последствиям

Из таблицы видно, что временные рамки ОСРВ достаточно жесткие. Среди современных операционных систем есть класс продуктов, разработанных специально для построения систем жесткого реального времени - VxWorks, OS9, QNX, LynxOS, OSE и другие. Эти системы содержат необходимый набор инструментов, и в некоторых случаях являются единственным выбором – на него приходится идти, невзирая на затраты. Однако достаточно часто требования к реальному времени (полная

предсказуемость времени реакции) снижаются, например, необходимо добиться только нужной средней производительности.

Иногда достаточно жестко контролировать только одно из событий, допуская при этом задержки реакций на остальные. В подобных случаях возможности выбора расширяются, и желаемых результатов можно достичь, используя такие широко распространенные операционные системы как LINUX, Windows NT, Windows CE, дополняя их расширениями реального времени (RTAI, RT LINUX, RTX).

Принципы построения ОС для СРВ

1. Многопоточность (multi-threaded) и прерываемость

Как указывалось выше, ОСРВ должна быть предсказуемой. Это означает, что максимальное время выполнения того или иного действия должно быть известно заранее и должно соответствовать требованиям приложения.

Первое требование состоит в том, что ОС должна быть многопоточной по принципу абсолютного приоритета (прерываемой). Планировщик должен иметь возможность прервать любой поток и предоставить ресурс тому потоку, которому он более всего необходим. ОС (и аппаратура) должны также обеспечивать прерывания на уровне обработки прерываний.

Поток выполнения – наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени). В качестве аналогии потоки выполнения процесса можно уподобить нескольким вместе работающим поварам. Все они готовят одно блюдо, читают одну и ту же кулинарную книгу с одним и тем же рецептом и следуют его указаниям, причём не обязательно все они читают на одной и той же странице.

2. Приоритеты потоков

Проблема состоит в том, что необходимо определить, какой задаче требуется конкретный ресурс. В идеальной ситуации ОСРВ отдает ресурс потоку или драйверу с ближайшим сроком исполнения (так называемые ОС, управляемые временным ограничением (deadline driven OS)).

Чтобы реализовать это, ОС должна знать время, требуемое каждому из выполняющихся потоков для их завершения (до сих пор не существует ОС, построенной по этому принципу, так как он слишком сложен для реализации), поэтому разработчики ОС применяют другой подход: вводится понятие уровня приоритета задачи, и временные ограничения сводят к приоритетам. Так как умозрительные решения чреваты ошибками, показатели СРВ при этом снижаются. Чтобы более эффективно осуществить указанное преобразование ограничений, проектировщик может воспользоваться теорией расписаний или имитационным моделированием, хотя и это может оказаться бесполезным. На сегодняшний день не имеется иного решения, поэтому понятие приоритета потока необходимо.

3. Предсказуемые механизмы синхронизации задач

Задачи разделяют данные (ресурсы) и должны сообщаться друг с другом, следовательно, должны существовать механизмы блокирования и коммуникации.

4. Система наследования приоритетов

На самом деле именно этот механизм синхронизации и тот факт, что различные потоки используют одно и то же пространство памяти, отличают потоки от процессов. Процессы не разделяют одно и то же пространство памяти. Так, например, старые версии UNIX не являлись многопоточными. Старый UNIX – многозадачная ОС, где задачами являются процессы, которые сообщаются через потоки (pipes) и разделяемую память. Оба эти механизма используют одну и ту же файловую систему, в результате поведение файловой системы потенциально может стать непредсказуемым.

Комбинация приоритета потока и разделение ресурсов между потоками приводит к другому явлению: классической проблеме инверсии приоритетов. Это можно проиллюстрировать примером, где есть как минимум три потока. Когда поток низшего приоритета занял ресурс, разделяемый с потоком высшего приоритета, а сначала выполняется поток среднего приоритета, выполнение потока высшего приоритета будет приостановлено, пока не освободится ресурс и не отработает поток среднего приоритета. В этой ситуации время, необходимое для завершения потока высшего приоритета, зависит от нижних приоритетных уровней – это и есть инверсия приоритетов. Ясно, что в такой ситуации трудно выдержать ограничение на время исполнения.

Чтобы устранить такие инверсии, ОСРВ должна допускать наследование приоритета, т. е. повышение приоритета до уровня вызывающего потока. Наследование означает, что блокирующий ресурс поток наследует приоритет блокируемого потока (справедливо лишь в том случае, если блокируемый

поток имеет более высокий приоритет). Иногда утверждают, что в грамотно спроектированной системе такая проблема не возникает. В случае сложных систем с этим нельзя согласиться. Единственный способ решения этой проблемы состоит в увеличении приоритета потока вручную прежде, чем ресурс окажется заблокированным – это возможно в случае, когда два потока разных приоритетов претендуют на один ресурс. В общем случае решение этой проблемы еще не найдено.

5. Поведение ОС должно быть известно

Наконец, следует рассмотреть временные ограничения. Время выполнения системных вызовов и временные характеристики поведения системы в различных обстоятельствах должны быть известны разработчику, поэтому производитель ОСРВ должен приводить следующие характеристики:

- ✓ латентную задержку прерывания (т. е. время от момента прерывания до момента запуска задачи): она должна быть предсказуема и согласована с требованиями приложения. Эта величина зависит от числа одновременно "висящих" прерываний;
- ✓ максимальное время выполнения каждого системного вызова (должно быть предсказуемым и независимым от числа объектов в системе);
- ✓ максимальное время маскирования прерываний драйверами и ОС.
- ✓ системные уровни прерываний;
- ✓ уровни прерываний драйверов устройств, их временные характеристики и т. д.

Контрольные вопросы по теме

Уровень модуля

Уровень курса

1. Особенности операционных систем реального времени.
2. Назначение систем реального времени.
3. Основные требования и условия применения операционных систем реального времени.
4. Применяемость операционных систем в качестве операционных систем реального времени.
5. Принципы построения операционных систем для систем реального времени.

Лекция №5

Тема: Операционная система реального времени QNX

Оглавление

Источники:

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. – СПб.: Питер, 2015. – 1120 с.: ил.
2. Столлингс В. Операционные системы: внутренняя структура и принципы проектирования, 9-е изд.: Пер. с англ. – СПб.: ООО “Диалектика”, 2020. – 1264 с.: ил.

Общее описание операционной системы реального времени QNX

Операционная система QNX является разработкой канадской компании QNX Software System Ltd (1981).

Операционная система QNX представляет собой гибрид 16/32-битовой операционной системы, которую пользователь может конфигурировать по своему усмотрению. Она применяется для создания систем, работающих в реальном масштабе времени.

QNX – первая коммерческая ОС, построенная на принципах микроядра и обмена сообщениями. Система реализована в виде совокупности независимых (но взаимодействующих через обмен сообщениями) процессов различного уровня (менеджеры и драйверы), каждый из которых реализует определенный вид сервиса.

Эти идеи позволили добиться нескольких важнейших преимуществ:

- предсказуемость, означающая ее применимость к задачам жесткого реального времени. Ни одна версия UNIX не может достичь подобного качества, поскольку код ядра слишком велик. Любой системный вызов из обработчика прерывания в UNIX может привести к непредсказуемой задержке (как и в Windows NT);
- масштабируемость и эффективность, достигаемые оптимальным использованием ресурсов и означающие ее применимость для встроенных (embedded) систем. В каталоге dev присутствуют только необходимые для поставленных задач файлы, соответствующие нужным драйверам. Драйверы и менеджеры можно запускать и удалять (кроме файловой системы) динамически, просто из командной строки. Возможна также покупка только тех модулей, которые реально необходимы для обеспечения нужных функций;
- расширяемость и надежность одновременно, поскольку написанный драйвер не нужно компилировать в ядро, рискуя вызвать нестабильность системы.

Система построена по технологии **FLEET** [Fault-tolerance (отказоустойчивая), **Load-balancing** (регулирующая нагрузку), **Efficient** (эффективная), **Extensible** (расширяемая), **Transparent** (прозрачная)], которая характеризуется следующим. QNX является ОСРВ на основе микроядра (размером около 10 Кб). В качестве основного средства взаимодействия между процессами система использует передачу сообщений. Благодаря этому в 32-битовой среде возможно взаимодействие процессов с 32 и 16-битовыми кодами, причем сообщения передаются между любыми процессами,

независимо от того, находятся ли процессы на одном компьютере или на разных узлах сети. Пользователь, работая на одном из узлов сети, может иметь доступ к любым ресурсам остальных узлов, включая порты, файловую систему и задачи. Пользователю нет необходимости вникать в сетевой протокол. Сетевой протокол определяет также пакеты, которые применяются и для передачи сообщений. Сетевой администратор распознает эти пакеты и переправляет микроядру, которое, в свою очередь, переправляет их в шину локальных сообщений. QNX распознает не только пакеты сообщений QNX-процессов. Можно также легко обращаться к сетевому администратору для передачи таких пакетных протоколов, как TCP/IP, и др. Возможно обращение к различным сетевым администраторам через один кабель.

Операционная система QNX объединяет всю сеть ПК в единый набор ресурсов с абсолютной прозрачностью доступа к ним. Узлы могут добавляться и исключаться из сети, не влияя на целостность системы. Сетевая обработка данных в QNX является настолько гибкой, что можно объединить в одну сеть любой разнородный набор Intel-совместимых компьютеров, соединенных через Arcnet, Ethernet, Token Ring или через последовательный порт, к которому также может быть подключен модем. Кроме того, возможно участие компьютера одновременно в нескольких сетях, и если одна из них окажется перегруженной или выйдет из строя, то QNX автоматически будет использовать другие доступные сети без потери информации.

Для QNX разработано множество пользовательских программ, например, базы данных, которые по производительности часто превосходят аналоги под управлением других операционных систем.

Операционная система как организованный набор процессов

ОС QNX Neutrino строится на основе компактного микроядра, способного управлять группой взаимодействующих процессов. Как видно па рис. 1, структура операционной системы больше напоминает "слаженную команду", чем иерархию, так как несколько равноправных "игроков" взаимодействуют в ней между собой посредством координирующего ядра.

ОС QNX Neutrino действует как своего рода "программная шина", позволяющая динамически присоединять/отсоединять модули ОС по мере необходимости.

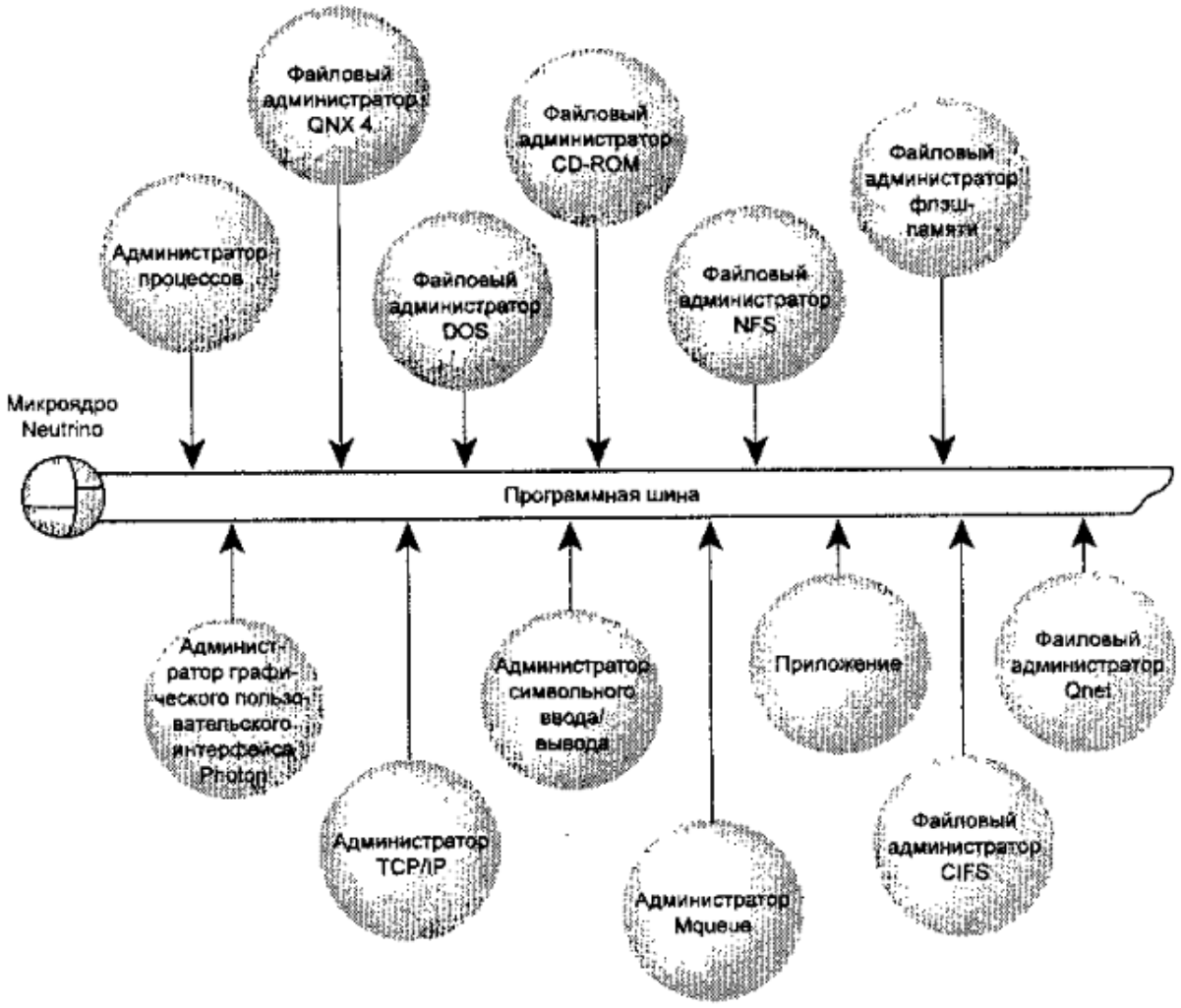


Рис. 1. Архитектура ОС QNX Neutrino

QNX Neutrino как операционная система на основе обмена сообщениями

В QNX применен обмен сообщениями между процессами в качестве основного метода межзадачного взаимодействия. Полная интеграция механизма межзадачного обмена сообщениями в саму архитектуру операционной системы во многом определяет ее высокую производительность, простоту и эффективность.

В ОС QNX Neutrino сообщение представляет собой группу байтов, передаваемых от одного процесса другому. Сообщение не несет никакого абсолютного значения в операционной системе. Его содержание имеет смысл только для отправителя сообщения и его получателя.

Механизм обмена сообщениями позволяет процессам не только обмениваться данными между собой, но и является средством синхронизации выполнения нескольких процессов. Процессы отправляют и получают со-

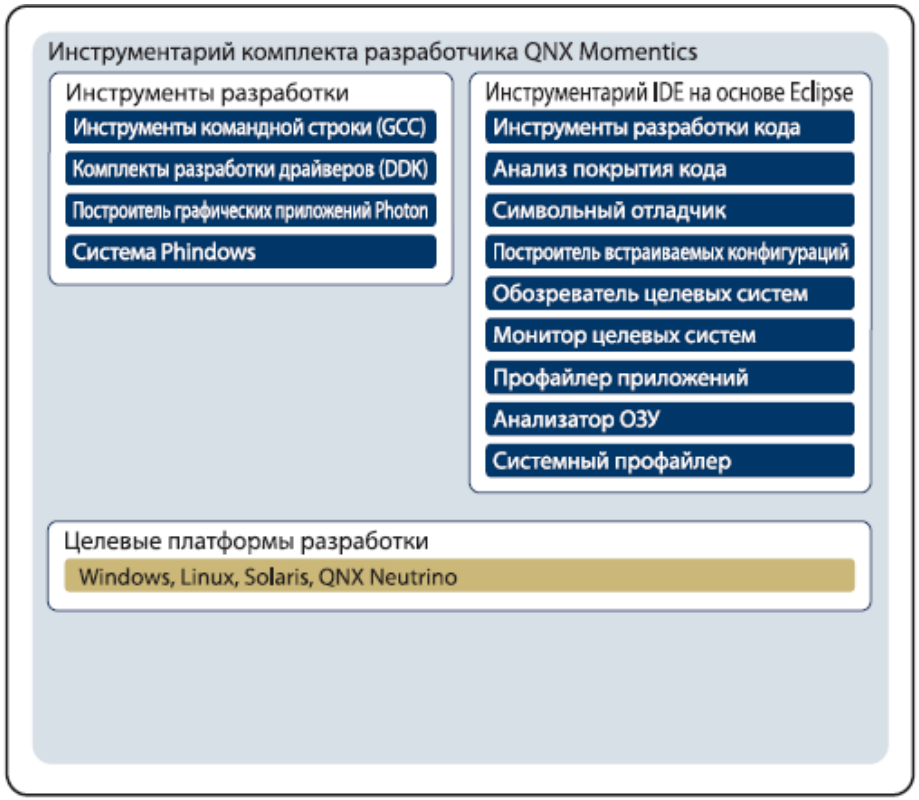
общения, а также отвечают на них. При этом происходят различные "изменения состояния" процессов, от которых зависит, когда и в течение какого времени эти процессы должны выполняться. Определяя состояние и приоритет каждого из процессов, микроядро осуществляет их планирование оптимальным способом и с наиболее эффективным расходованием вычислительных ресурсов. Таким образом, механизм обмена сообщениями является основополагающим и постоянно действует на всех уровнях операционной системы.

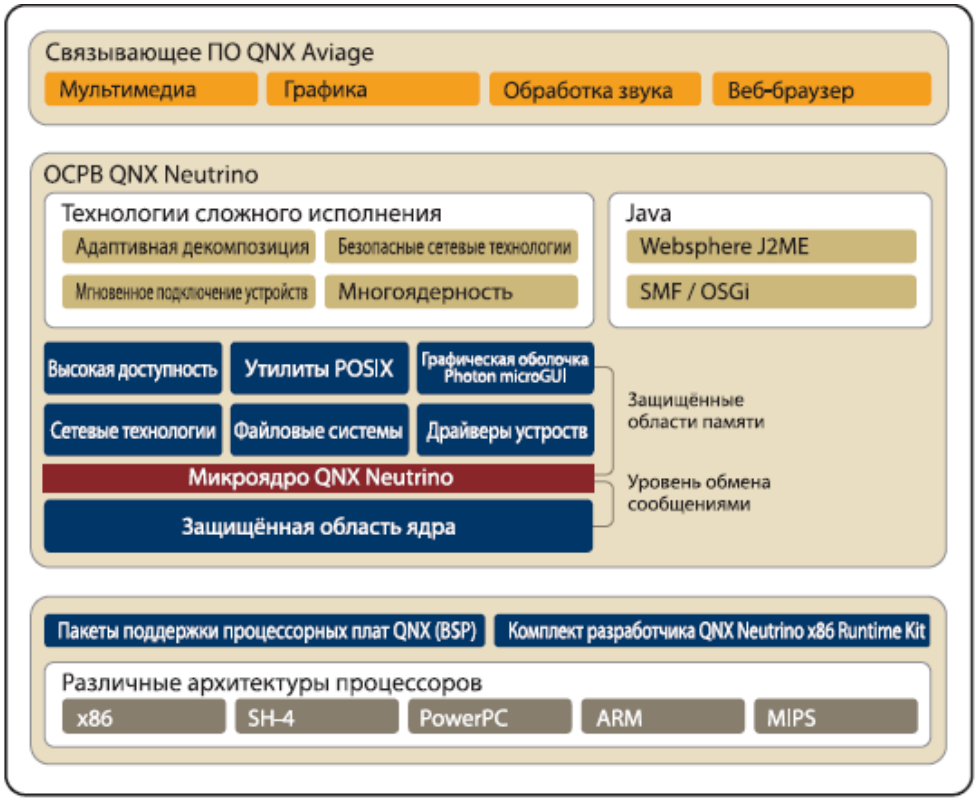
Для приложений, работающих в режиме реального времени, и других приложений критического назначения требуется, чтобы механизм межзадачного взаимодействия имел высокую степень надежности, поскольку процессы, на основе которых работают такие приложения, очень тесно связаны между собой. Метод обмена межзадачными сообщениями как раз и позволяет достичь строгого управления и высокой надежности выполнения приложений в ОС QNX Neutrino.

Профессиональный пакет

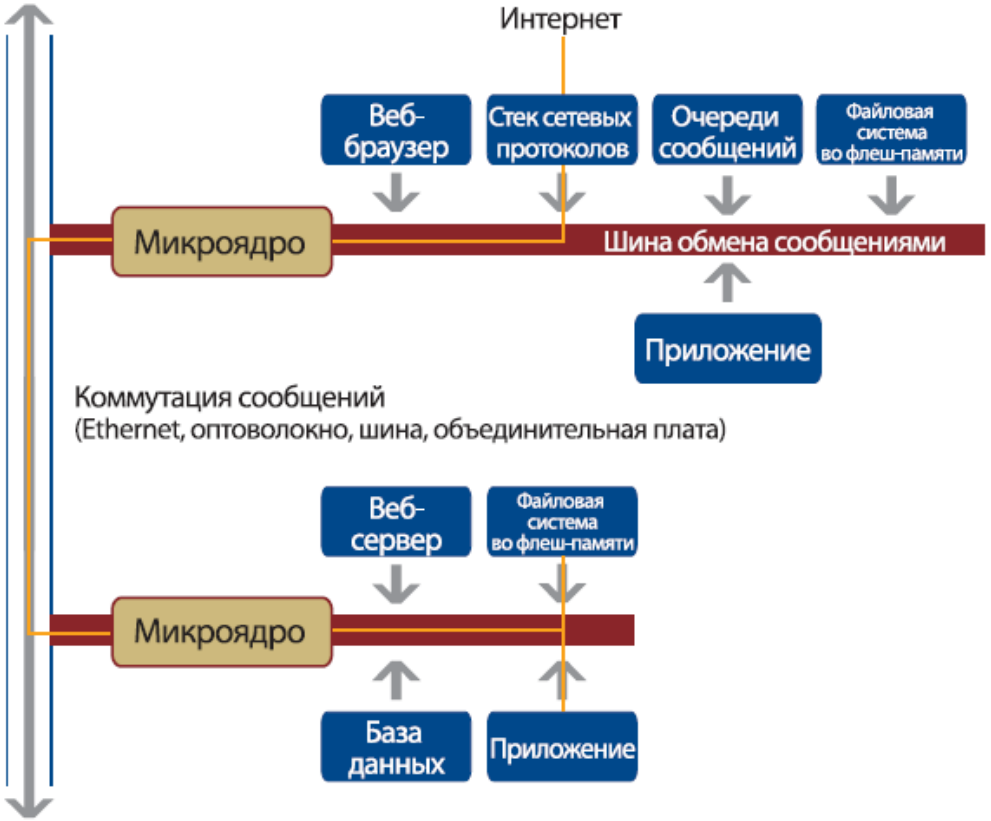
В состав QNX входят графическая оболочка Photon microGUI – полнофункциональная и в то же время встраиваемая оконная среда с расширяемой подсистемой мультимедиа и поддержкой аппаратно-независимого многослойного интерфейса. В её состав также включены масштабируемые шрифты и встроенная поддержка Unicode; широкий спектр файловых систем, включая образную файловую систему, файловую систему в ОЗУ или ППЗУ Flash, файловые системы QNX, Linux, DOS, CD - ROM, DVD, NFS, CIFS, пакетную файловую систему и файловую систему со сжатием; администратор систем высокой готовности поддерживает квитанции работоспособности для ранней диагностики отказов и предоставляет интеллектуальные механизмы перезапуска отказавших компонентов; поддержка Java для среды исполнения, оптимизированные для QNX: WebSphere Embedded Environment (стандарт Java Powered) и WebSphere Custom Environment; POSIX API поддержка POSIX 1003.1-2003, включая многопоточность, расширения реального времени и множество других опций; отказоустойчивый сетевой интерфейс в QNX приложениях могут прозрачно взаимодействовать по резервированным сетевым соединениям. Если одно соединение нарушается, ОС автоматически перенаправит сетевой трафик по одному или нескольким альтернативным маршрутам. Поддерживается также балансировка нагрузки между всеми доступными соединениями для увеличения пропускной способности.

Номенклатура компонентов QNX





Прозрачные распределенные вычисления



Технология прозрачных распределенных вычислений позволяет приложению получить доступ к любому узлу в сети. Приложения и сервисы могут мгновенно стать распределенными по сети без разработки специального кода.

Функции реального времени: QNX обеспечивает быстрое, предсказуемое время реакции за счёт вытесняющего планирования, сверхмалых задержек обработки прерываний, распределённого наследования приоритетов и многих других реализованных в ней современных механизмов. В поставку профессионального пакета QNX Momentics входит всё, что необходимо на каждом этапе разработки системы, от встраивания на процессорную плату до системного анализа. Помимо графической IDE разработчики могут использовать и обычный инструментарий командной строки, получая абсолютно такие же бинарные модули и контекст. QNX Momentics обеспечивает разработчикам свободу выбора инструментария разработки, так как его интегрированная среда основана на Eclipse – открытой платформе, поддерживаемой большим и постоянно расширяющимся сообществом компаний производителей. Eclipse также предоставляет расширяемую архитектуру подключаемых модулей, позволяющую QNX Momentics работать с практически любым типом информационного содержания. Например, в состав QNX Momentics входит множество подключаемых модулей для разработки и анализа встраиваемых образов, исходных текстов на C/C++ и прочих объектов, характерных для встраиваемых систем. Поддержка множества языков программирования (C, C++, встраиваемый C++ или Java), инструментальных ОС (Windows, Solaris или QNX) и целевых процессоров (ARM, MIPS, PowerPC, SH -4, StrongARM, XScale или x86) позволяет существенно сократить время разработки проекта независимо от его масштаба и сложности.

Средства разработки кода QNX Momentics включают в себя:

- «мастера» проектов,
- редакторы кода,
- средства управления исходными текстами,
- средства построения проектов.

QNX Momentics поддерживает большое количество широко распространённых библиотек, включая

- ANSI C,
- POSIX,
- Dinkum C++, полная версия,
- Dinkum C++, встраиваемая версия с сокращенной STL,
- GNU C++ (только для x86),
- сжатие,
- сеть,
- графика,
- виджеты,
- XML.

IDE включает в себя встроенную поддержку протокола управления исходными текстами CVS (управление версиями) включая поддержку удалённого сервера и доступ к защищенным репозитариям посредством SSH. Также поддерживается система управления исходными текстами ClearCase, поставляемая компанией Rational Software в виде подключаемого модуля для Eclipse.

Возможности:

- локальное управление версиями;
- распределённое управление версиями;
- поддержка журнала изменения файлов (кем и какие изменения внесены);
- визуальное сравнение версий;
- интерактивное слияние изменений в ситуации, когда несколько разработчиков изменяют один и тот же файл.

Используя PhAB™, визуальное средство разработки приложений QNX Photon microGUI®, можно создавать полнофункциональные пользовательские интерфейсы с простотой щелчка мыши. Возможности PhAB™ включают в себя:

- готовые шаблоны PhAB;
- обширная палитра доступных элементов управления (виджетов);
- привязка к виджетам диалоговых окон или вызовов функций (предопределенные, по «горячей» клавише или по низкоуровневым событиям);
- полная поддержка со стороны интегрированной среды QNX Momentics;
- минимизация кода;
- многоязычная поддержка.

Инструментальные средства разработки

Операционная система QNX обладает средствами отладки и тестирования и, в частности, в ее состав входит *профайлер* – средство сбора и анализа информации о выполнении программ с целью оптимизации их работы, которое применяется в процессе разработки программного обеспечения. Профайлер позволяет разрешать конфликты синхронизации, определять ситуации взаимных блокировок, выявлять корни семантических ошибок, находить скрытые неполадки в программном и аппаратном обеспечении и оптимизировать производительность приложения, причём как для однопроцессорных, так и для многопроцессорных целевых систем. Характеристики системы могут быть проанализированы в реальном времени, в моменты возникновения событий. Удобный инструмент поиска позволяет

проанализировать детали по каждому событию, включая время возникновения, владельца и тип. Системный профайлер может отображать огромные объёмы информации, включая информацию о вызовах ядра, аппаратных прерываниях, состоянии потоков, обмене сообщениями и событиях планировщика. Сложные комбинации условий могут быть отслежены благодаря развитой системе динамических фильтров, определяемых пользователем. В приложения могут быть встроены средства генерации специализированных сообщений для подсистемы трассировки, оказывающей упреждающее воздействие на процесс записи событий. Профайлер приложений предоставляет информацию об использовании процессорного времени каждым потоком и отображает её одновременно как в виде абсолютных значений, так и в виде процентной доли от общего времени с возможностью сортировки. Профайлер может анализировать динамически загружаемые разделяемые библиотеки, отвечая тем самым на вопрос, где кроется причина снижения производительности в коде приложения или в библиотеке, которую оно вызывает.

ОС QNX предоставляет пользователю библиотеку распределения памяти, содержащую реализацию большинства типовых операций над строками и памятью. Эти функции перед выполнением операции проверяют корректность использования указанной области памяти, позволяя выявлять ошибки типа переполнения, выборки из пустого буфера, некорректного использования памяти и повторного освобождения одной и той же области. «Интеллектуальный» механизм отслеживает ошибки работы с памятью. При возникновении ошибки соответствующий фрагмент исходного текста будет помечен предупреждением, при этом можно:

- продолжить выполнение программы;
- завершить программу и сохранить её образ в дамп файле;
- остановить программу и немедленно переключиться в отладчик для локализации проблемы.

Анализатор ОЗУ помогает визуально представлять использование памяти программами и может быстро выявлять переполнения буферов, некорректные освобождения памяти и множество других типовых ошибочных ситуаций. Анализатор ОЗУ предоставляет:

- информацию на уровне процесса, которая позволяет быстро оценить карту памяти программы;
- специализированную статистику распределения памяти для выявления проблем. Статистика включает в себя суммарное количество свободных, распределённых и используемых байтов и блоков; динамический журнал использования памяти для оценки динамики ситуации. Комплекты разработки драйверов (DDK)

позволяют быстро создавать драйверы для нестандартного оборудования – аудио, графических и сетевых адаптеров, устройств ввода, принтеров, символьных и USB устройств. Комплекты включают в себя исходные тексты, детальную документацию и готовый программный каркас, в котором весь высокоуровневый аппаратно-независимый код уже реализован в виде библиотек.

Поскольку в QNX драйверы выполняются как обычные пользовательские процессы, их можно отлаживать и оптимизировать при помощи того же интегрированного инструментария, который QNX Momentics предоставляет для отладки обычных приложений. Нет никакой необходимости в использовании отладчиков на уровне ядра, которые могут застопорить работу всей целевой системы, в результате скрывая ошибки. Микроядерная архитектура QNX позволяет тестировать изменения в коде драйверов без перезагрузки системы и даже без перезапуска сеанса отладки: необходимо просто перекомпилировать и перезапустить драйвер.

Наличие встроенного эмулятора позволяет тестировать и отлаживать драйверы непосредственно на инструментальном компьютере, не теряя времени на ожидание целевой аппаратуры. QNX Momentics предоставляет полный набор инструментария для начальной загрузки и взаимодействия с целевым оборудованием. В этот инструментарий входят пакеты поддержки (BSP) для широкого спектра процессорных плат, построитель встраиваемых систем, позволяющий быстро формировать и настраивать целевые образы, и уникальный целевой агент, динамически загружающий сервисные модули по мере необходимости.

QNX Momentics также включает в себя навигатор целевых систем, который позволяет привязывать проекты к IP адресам или именам хостов, однозначно определяя программно-аппаратную конфигурацию целевых систем. Эти конфигурации впоследствии могут использоваться при работе с другими инструментальными средствами. Навигатор целевых систем также обеспечивает интерактивное отображение выполняющихся процессов, позволяя просмотреть следующую информацию о целевой системе: использование ресурсов, атрибуты потоков, файловые дескрипторы и т.д. QNX Momentics предоставляет богатый выбор готовых пакетов поддержки процессорных плат (BSP – Board Support Package) на основе процессоров ARM, MIPS, x86, PowerPC, SH-4, StrongARM и XScale. Каждый BSP снабжён детальной документацией и исходными текстами всех бинарных модулей, включая начальный загрузчик, стартовый код и драйверы устройств.

Контрольные вопросы по теме

Уровень модуля

Уровень курса

1. Общее описание операционной системы реального времени QNX.
2. Операционная система QNX как организованный набор процессов на основе обмена сообщениями.
3. Состав профессионального пакета операционной системы реального времени QNX.
4. Инструментальные средства разработки прикладного ПО системы реального времени QNX

Лекции № 6

Тема: Клиент-серверные технологии. OPC-сервер

Оглавление

Источники:

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.: ил.
2. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил.
3. Бабак В.П. Теоретические основы информационно-измерительных систем: Учебник / В.П. Бабак, С.В. Бабак, В.С. Ерёменко и др. – К., 2014. – 832 с.

Введение в клиент-серверные технологии

Как правило компьютеры и программы, входящие в состав информационной системы, не являются равноправными. Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам. Компьютер (или программу), управляющий ресурсом, называют сервером этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер...). Клиент – это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу и получающий от сервера запрошенную информацию. Сервер – аппаратный или программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам. Клиент и сервер какого-либо ресурса могут находится как на одном компьютере, так и на различных компьютерах, связанных сетью.

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в системе, где имеются *серверы*, узлы-поставщики некоторых специфичных функций (сервисов) и *клиенты*, потребители этих функций.

Основная идея архитектуры «клиент-сервер» состоит в разделении сетевого приложения на несколько компонентов, каждый из которых реализует специфический набор сервисов. Компоненты такого приложения могут выполняться на разных компьютерах, выполняя серверные и/или клиентские функции. Это позволяет повысить надежность, безопасность и производительность сетевых приложений и сети в целом.

Практические реализации такой архитектуры называются **клиент-серверными технологиями**. Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются *протоколом обмена (протоколом взаимодействия)*.

Архитектура «клиент-сервер» широко применяется в измерительных и управляющих системах. Можно говорить, что технология «клиент-сервер» является частью компьютерно-интегрированных технологий. Датчики компьютерно-интегрированных систем являются источником измерительной информации, и они отправляют измеренные данные в программы управления и сбора данных. То есть, датчики предоставляют сервис по предоставлению измерительных данных. Они, по существу, являются серверами. А программа управления и сбора данных в данном случае является клиентом. Об этом взаимодействии применительно к компьютерно-интегрированным системам речь пойдет в разделе, посвященном OPC серверам. Первоначально будут рассмотрены основные понятия и принципы клиент-серверного взаимодействия.

Отметим, что программы диспетчерского управления и сбора данных в компьютерно-интегрированных системах называют SCADA-системами (**S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition – диспетчерское управление и сбор данных) или SCADA-пакетами, поскольку они на самом деле представляют собой целый комплекс программ. SCADA-системам будет посвящена одна из лекций.

Виды клиент-серверных технологий

Архитектура клиент-сервер применяется в большом числе сетевых технологий, используемых для доступа к различным сетевым сервисам. Кратко рассмотрим некоторые типы таких сервисов (и серверов). Это лишь несколько типов из всего многообразия клиент-серверных технологий, используемых как в локальных, так и в глобальных сетях.

Web-серверы

Изначально представляли доступ к гипертекстовым документам по протоколу HTTP (Huper Text Transfer Protocol). Сейчас поддерживают расширенные возможности, в частности работу с бинарными файлами (изображения, мультимедиа и т.п.).

Серверы приложений

Предназначены для централизованного решения прикладных задач в некоторой предметной области. Для этого пользователи имеют право запускать серверные программы на исполнение. Использование серверов приложений позволяет снизить требования к конфигурации клиентов и упрощает общее управление сетью.

Серверы баз данных

Серверы баз данных используются для обработки пользовательских запросов на языке SQL. При этом СУБД находится на сервере, к которому и подключаются клиентские приложения.

Файл-серверы

Файл-сервер хранит информацию в виде файлов и представляет пользователям доступ к ней. Как правило файл-сервер обеспечивает и определенный уровень защиты от несакционированного доступа.

Почтовые серверы

Представляют услуги по отправке и получению электронных почтовых сообщений.

OPC-серверы

OPC-серверы (***O**pen **P**latform **C**ommunications server*) предназначены для управления объектами автоматизации и технологическими процессами. Они реализуют интерфейс OPC — набор спецификаций стандартов, разработанный специально для целей автоматизации.

Клиентское приложение

Для доступа к тем или иным сетевым сервисам используются клиенты, возможности которых характеризуются понятием «толщины». Оно определяет конфигурацию оборудования и программное обеспечение, имеющиеся у клиента. Рассмотрим возможные граничные значения:

«Тонкий» клиент

Этот термин определяет клиента, вычислительных ресурсов которого достаточно лишь для запуска необходимого сетевого приложения через web-интерфейс. Пользовательский интерфейс такого приложения формируется средствами статического HTML (выполнение JavaScript не предусматривается), вся прикладная логика выполняется на сервере. Для работы тонкого клиента достаточно лишь обеспечить возможность запуска web-браузера, в окне которого и осуществляются все действия. По этой причине web-браузер часто называют "универсальным клиентом".

«Толстый» клиент

Таковым является рабочая станция или персональный компьютер, работающие под управлением собственной дисковой операционной системы и имеющие необходимый набор программного обеспечения. К сетевым серверам «толстые» клиенты обращаются в основном за дополнительными услугами (например, доступ к web-серверу или корпоративной базе данных). Так же под «толстым» клиентом подразумевается и клиентское сетевое приложение, запущенное под управлением локальной ОС. Такое приложение совмещает компонент представления данных (графический пользовательский интерфейс ОС) и прикладной компонент (вычислительные мощности клиентского компьютера).

Как уже отмечалось выше, отдельным видом клиента является SCADA-система.

Клиент-серверная архитектура

Двухзвенная архитектура

В любой сети, построенной на современных сетевых технологиях, присутствуют элементы клиент-серверного взаимодействия, чаще всего на основе **двухзвенной архитектуры**. Двухзвенной она называется из-за необходимости распределения трех базовых компонентов: представление

данных, прикладной компонент, управление ресурсами – между двумя узлами (клиентом и сервером).

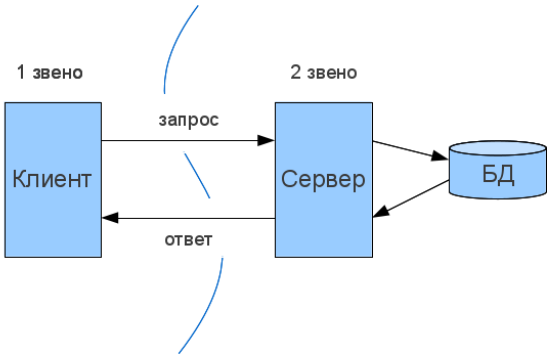


Рис.1. Двухзвенная клиент-серверная архитектура

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса (рис. 1)

Трехзвенная архитектура

В сетевых технологиях все больше и больше начинают использовать распределенные вычисления. Они реализуются на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная клиент-серверная архитектура становится **трехзвенной**.

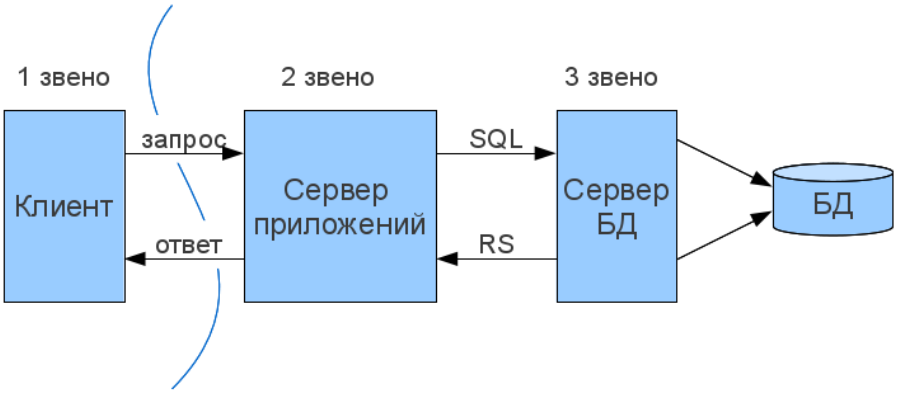


Рис.2. Трехзвенная клиент-серверная архитектура

Как правило, третьим звеном в трехзвенной архитектуре становится сервер приложений, т.е. компоненты распределяются следующим образом (рис. 2):

1. Представление данных – на стороне клиента.
2. Прикладной компонент – на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО).
3. Управление ресурсами – на сервере БД, который и представляет запрашиваемые данные.

Трехзвенная архитектура может быть расширена до многозвенной путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня. Абстрактный пример многозвенной модели приведен на 3.

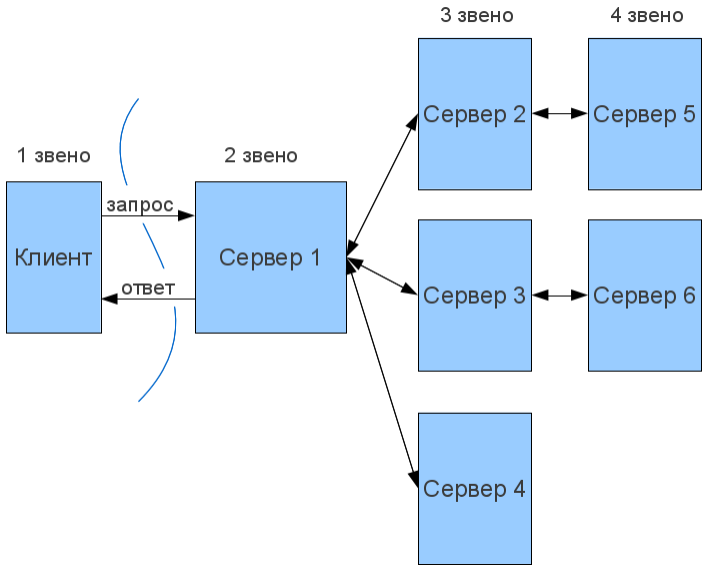


Рис.3. Многозвенная клиент-серверная архитектура

OPC сервер в компьютерно-интегрированных системах

Как уже было указано, OPC-серверы специально предназначены для управления объектами автоматизации и технологическими процессами. Они реализуют так называемый интерфейс OPC (***O**pen **P**latform **C**ommunications*), представленный в соответствующих стандартах. Стандарт OPC разработан международной организацией OPC Foundation, членами которой являются более 400 фирм, работающих в области средств автоматизации и измерительной техники. Первая версия OPC стандарта была выпущена в 1998г.

Обзор стандарта OPC

Главной целью стандарта OPC явилось обеспечение возможности совместной работы (интероперабельности) средств автоматизации, функционирующих на разных аппаратных платформах, в разных промышленных сетях и производимых разными фирмами. До разработки OPC стандарта SCADA пакет нужно было адаптировать к каждому новому

оборудованию индивидуально. Существовали длинные списки "поддерживаемого оборудования", очень сложной была техническая поддержка. При модификации оборудования нужно было вносить изменения во все драйверы, каждый из которых поддерживал протокол обмена только с одной клиентской программой. Число таких драйверов доходило до сотен.

После появления стандарта OPC практически все SCADA-пакеты были перепроектированы как OPC-клиенты, а каждый производитель аппаратного обеспечения стал снабжать свои контроллеры, модули ввода-вывода, интеллектуальные датчики и исполнительные устройства стандартным OPC сервером. Благодаря появлению стандартизации интерфейса стало возможным подключение любого физического устройства к любой SCADA, если они оба соответствовали стандарту OPC. Разработчики получили возможность проектировать только один драйвер для всех SCADA-пакетов, а пользователи получили возможность выбора оборудования и программ без прежних ограничений на их совместимость.

Стандарт OPC относится только к интерфейсам, которые OPC сервер предоставляет клиентским программам. Метод же взаимодействия сервера с аппаратурой (например, с модулями ввода-вывода), стандартом не предусмотрен, и его реализация возлагается полностью на разработчика аппаратуры. Поэтому стандарт OPC может быть использован не только для взаимодействия SCADA с "железом", но и для обмена данными с любым источником данных, например, с базой данных или с GPS приемником.

OPC сервер как средство взаимодействия с техническим устройством может быть использован при разработке заказных программ на C++, Visual Basic, VBA и т. п.

Стандарт OPC состоит из нескольких частей:

- OPC DA (OPC Data Access) - спецификация для обмена данными между клиентом (например SCADA) и аппаратурой (контроллерами, модулями ввода-ввода и др.) в реальном времени;
- OPC Alarms & Events (A&E) - спецификация для уведомления клиента о событиях и сигналах тревоги, которые посылаются клиенту по мере их возникновения. Этот сервер пересылает аварийные сигналы, действия оператора, информационные сообщения, результаты контроля состояния системы;
- OPC HDA (Historical Data Access) - спецификация для доступа к предыстории процесса (к сохраненным в архиве данным). Сервер обеспечивает унифицированный способ доступа с помощью DCOM технологии. Обеспечивает чтение, запись и изменение данных;

- Batch - спецификация для особых физико-химических технологических процессов обработки материалов, которые не являются непрерывными. В таких процессах выполняется загрузка нескольких видов сырья в определенных пропорциях согласно рецепту, устанавливаются режимы обработки, а после выполнения цикла обработки и выгрузки готового материала загружается новая партия сырья. OPC сервер выполняет обмен между клиентом и сервером рецептами, характеристиками технологического оборудования, условиями и результатами обработки;
- OPC Data eXchange - спецификация для обмена данными между двумя OPC DA серверами через сеть Ethernet;
- OPC Security - спецификация, которая определяет методы доступа клиентов к серверу, которые обеспечивают защиту важной информации от несанкционированной модификации;
- OPC XML-DA - набор гибких, согласующихся друг с другом правил и форматов для представления первичных данных с помощью языка XML, веб технологий и сообщений SOAP (см. раздел "Архитектура автоматизированной системы".);
- OPC Complex Data - дополнительные спецификации к OPC DA и XML-DA, которые позволяют серверам работать со сложными типами данных, такими как бинарные структуры и XML-документы;
- OPC Commands - набор программных интерфейсов, который позволяет OPC клиентам и серверам идентифицировать, посылать и контролировать команды, исполняемые в техническом устройстве (в контроллере, модуле ввода-вывода);
- OPC Unified Architecture - принципиально новый набор спецификаций, который уже не базируется на DCOM технологии.

Наиболее широко используется спецификации OPC DA и реже - OPC HDA.

Тег

При использовании сервер OPC используется понятие тег. Тег – это метка, адрес, идентификатор данных (точка, источник или канал поступления данных), переменная, которая приписана этим данным. В простейшем случае один датчик и есть один тег - переменная в которой находится текущее значение. Например, к системе подключен датчик температуры. Показаниям датчика приписан уникальный тег. Эти показания можно найти по этому уникальному тегу.

В более сложном случае одному устройству могут соответствовать несколько тегов. Например, к сети подключен программируемый логический контроллер (ПЛК), а к этому ПЛК подключено несколько датчиков, каждый из которых измеряет какой-то один параметр. Тогда каждому датчику (точнее - показаниям каждого датчика) будет соответствовать свой уникальный тег.

Бывает, что одно и то же устройство измеряет несколько параметров, например: влажность и температуру. Тогда данным измерения влажности будет соответствовать один тег, а данным измерения температуры – другой тег.

OPC DA сервер

Сервер OPC DA является наиболее широко используемым в промышленной автоматизации. Он обеспечивает обмен данными (запись и чтение) между клиентской программой и физическими устройствами. Данные состоят из трех полей: значение, качество и временная метка. Параметр качества данных позволяет передать от устройства клиентской программе информацию о выходе измеряемой величины за границы динамического диапазона, об отсутствии данных, ошибке связи и другие.

Существует четыре стандартных режима чтения данных из OPC сервера:

- *синхронный режим*: клиент посылает запрос серверу и ждет от него ответ;
- *асинхронный режим*: клиент отправляет запрос и сразу же переходит к выполнению других задач. Сервер после выполнения функции запроса посылает клиенту уведомление и тот забирает предоставленные данные;
- *режим подписки*: клиент сообщает серверу список тегов, значения которых сервер должен отправлять клиенту только в случае их изменения. Для того, чтобы шум данных не был принят за их изменение, вводится понятие "мертвой зоны", которая слегка превышает максимально возможный размах помехи;
- *режим обновления данных*: клиент вызывает одновременное чтение всех активных тегов. Активными называются все теги, кроме обозначенных как "пассивные". Такое деление тегов уменьшает загрузку процессора обновлением данных, принимаемых из физического устройства.

В каждом из этих режимов данные могут читаться либо из кэша OPC сервера, либо непосредственно из физического устройства. Чтение из кэша выполняется гораздо быстрее, но данные к моменту чтения могут устареть. Поэтому сервер должен периодически освежать данные с максимально возможной частотой. Для уменьшения загрузки процессора используют

параметр частоты обновления, которая может быть установлена для каждой группы тегов индивидуально. Кроме того, некоторые теги можно сделать пассивными, тогда их значения не будут обновляться данными из физического устройства.

Запись данных в физическое устройство может быть выполнена только двумя методами: синхронным и асинхронным и выполняется сразу в устройство, без промежуточной буферизации. В синхронном режиме функция записи выполняется до тех пор, пока из физического устройства не поступит подтверждение, что запись выполнена. Этот процесс может занимать много времени, в течение которого клиент находится в состоянии ожидания завершения функции и не может продолжать выполнение своей работы. При асинхронной записи клиент отправляет данные серверу и сразу продолжает свою работу. После окончания записи сервер отправляет клиенту соответствующее уведомление.

В соответствии со стандартом, OPC сервер во время инсталляции автоматически регистрируется в реестре Windows. Запуск сервера осуществляется так же, как любой другой программы или автоматически из клиентской программы.

На рис. 4 показано диалоговое окно OPC сервера. Сервер позволяет выполнить поиск физических устройств, подключенных к COM-порту компьютера. На рис. 4 окно сервера слева показывает, что к компьютеру подключены три модуля ввода: NL16HV, NL8TI и NL8AI. Для удобства представления измеряемых величин (тегов) на объекте автоматизации имена тегов могут быть составными и путь к тегу может быть представлен в виде дерева, как показано на рис. 4. Имя выделенного на рисунке тега выглядит как "NL8TI.Laboratory32.Top.Vin4". Все имена и их структура задаются с помощью средств окна OPC сервера.

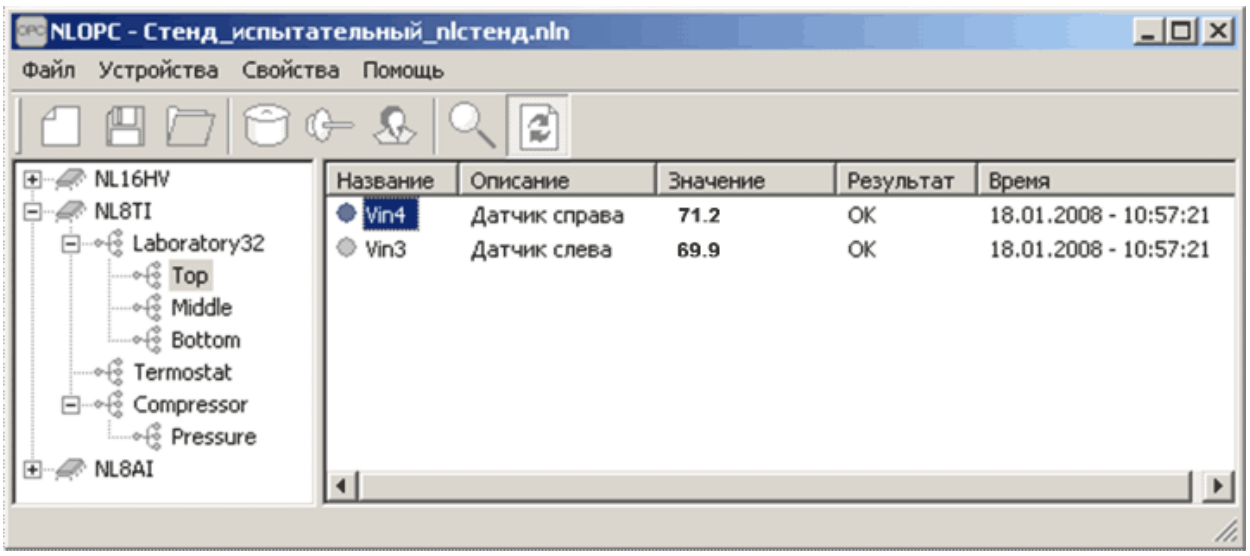


Рис. 4. Пример диалогового окна OPC сервера

При использовании OPC клиента (например, SCADA), имена тегов, доступные через OPC сервер, представляются в аналогичной форме в окне навигатора тегов (рис. 5). Клиент показывает все OPC серверы, установленные на компьютерах, доступных по сети Ethernet, и позволяет использовать все теги этих серверов.

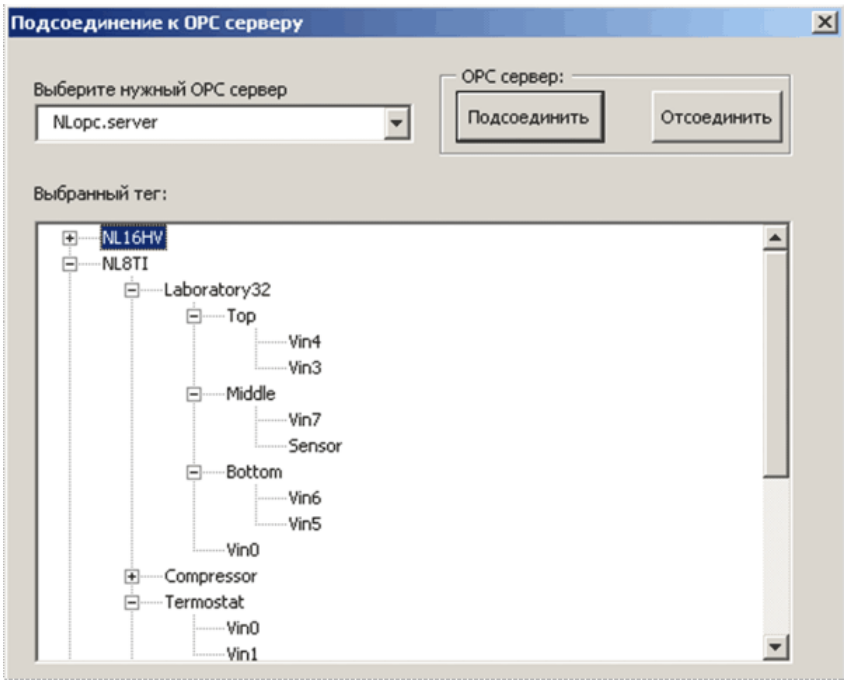


Рис. 5. Пример диалогового окна навигатора тегов OPC клиента

Пример архитектуры систем, включающих OPC серверы и OPC клиенты, показаны на рис. 6 и рис. 7. В качестве OPC клиента может выступать программа на языке C++ (например, SCADA-пакет) или программа на языке Visual Basic, VBA, Delphi или любая другая программа, поддерживающая внедрение COM-объектов (рис. 6). Программа на языке C++ взаимодействует с OPC сервером через интерфейс OPC Custom, а программа на Visual Basic, VBA, Delphi - через интерфейс автоматизации OPC Automation. OPC сервер и OPC клиенты могут работать только на компьютерах и контроллерах с операционными системами, поддерживающими технологию DCOM (например, Windows XP и Windows CE).

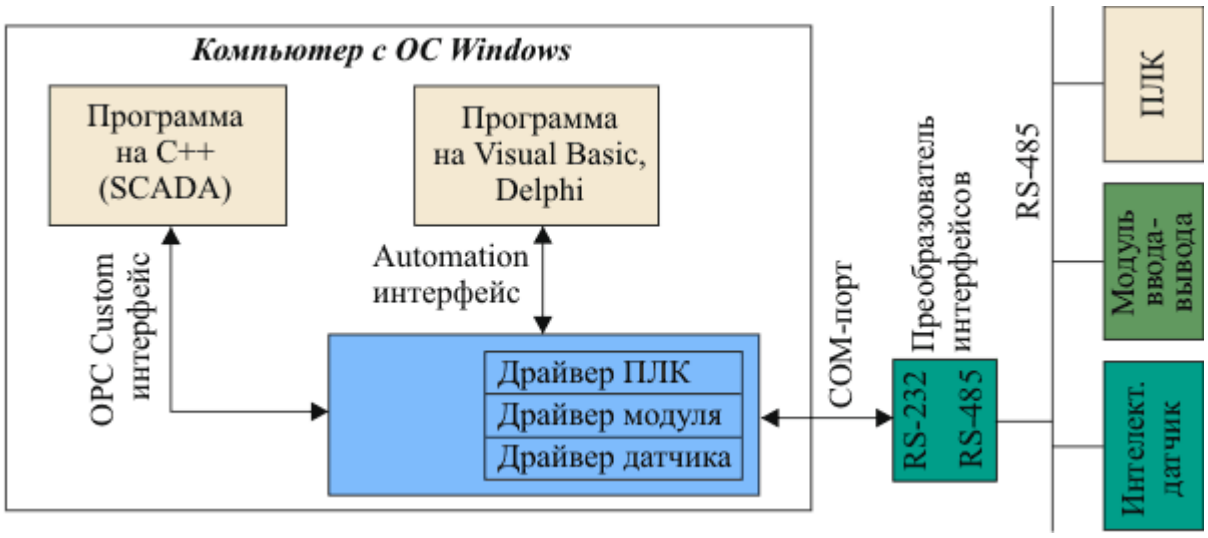


Рис. 6. Простой пример взаимодействия прикладных программ и физических устройств через OPC сервер на одном компьютере.

OPC сервер подключается к физическим устройствам любым способом; эти способы стандартом не предусмотрены. Например, сервер NLopc фирмы НИЛ АП использует для каждого физического устройства свой драйвер (рис. 6).

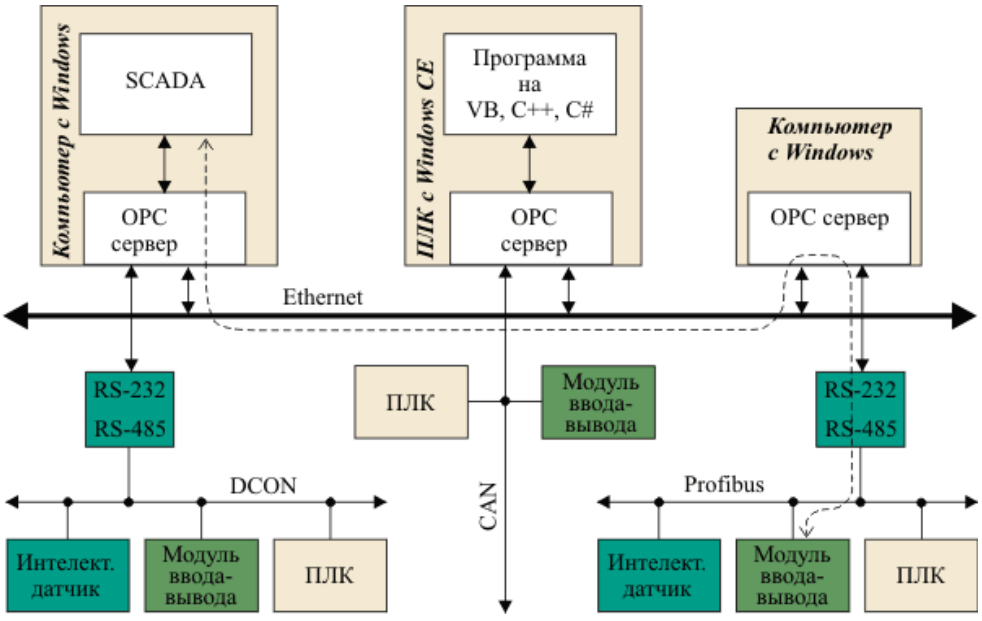


Рис. 7. Пример применения OPC технологии для сетевого доступа к данным в системах автоматизации

Клиентская программа и OPC сервер могут быть установлены на одном и том же компьютере, как показано на рис. 6, или на разных компьютерах сети Ethernet (рис. 7). При наличии нескольких компьютеров каждый из них может содержать OPC сервер и подключенные к нему физические устройства. В такой системе любой OPC клиент с любого компьютера может обращаться к любому OPC серверу, в том числе к расположенному на другом компьютере

сети. Это достигается благодаря технологии DCOM, использующей удаленный вызов процедур (RPC - Remote Procedure Call). Например, SCADA на рис. 7 может обратиться за данными к модулю ввода-вывода по пути, указанному на рис. 7 штриховой линией. Обратим внимание, что компьютеры и контроллеры в такой архитектуре могут работать с разными промышленными сетями. Обмен данными с программируемыми логическими контроллерами (ПЛК), работающими с ОС Windows CE, выполняется точно так, как с компьютерами.

При использовании оборудования разных производителей на компьютере (контроллере) может быть установлено несколько OPC серверов разных производителей, однако OPC сервер монопольно занимает COM-порт компьютера (поскольку непрерывно выполняет обновление данных), поэтому количество портов должно быть равно количеству OPC серверов. Для наращивания количества COM портов можно использовать преобразователи интерфейса USB в RS-232. К разным портам компьютера могут быть подключены разные промышленные сети. В этом случае OPC серверы используются в качестве межсетевых шлюзов.

Контрольные вопросы по теме

Уровень модуля

Уровень курса

1. Введение в клиент-серверные технологии.
2. Клиент-серверная архитектура.
3. OPC сервер в компьютерно-интегрированных системах. Стандарт OPC.
4. Понятие "тег" в компьютерно-интегрированных системах.
5. Режимы чтения данных из OPC сервера.

Лекции № 7 – 8 ***Тема:*** Программирование контроллеров

Оглавление

Источники:

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.: ил. https://rutracker.org/forum/viewtopic.php?t=4956359
2. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил.
3. Бабак В.П. Теоретические основы информационно-измерительных систем: Учебник / В.П. Бабак, С.В. Бабак, В.С. Ерёменко и др. – К., 2014. – 832 с.

Программное обеспечение в компьютерно-интегрированных технологиях

Современные системы промышленной и лабораторной автоматизации позволяют решать широкий круг задач, которые можно разделить на несколько групп, имеющих свои особенности:

- автоматизация управления технологическими процессами (АСУ ТП);
- взаимодействие системы с диспетчером (оператором);
- автоматизированный контроль и измерения (мониторинг);
- обеспечение безопасности;
- дистанционное управление, измерение, сигнализация (задачи телемеханики).

История развития программных средств автоматизации показала, что все особенности отдельных применений можно учесть путем настройки нескольких универсальных программ на выполнение конкретной задачи. К таким универсальным программам относятся:

- OPC сервер (рассмотрен в предыдущей лекции);
- средства МЭК-программирования контроллеров;
- SCADA-пакеты.

Для систем автоматизации, не связанных с АСУ ТП, используются программы LabVIEW, MatLab, HP-VEE и др., ориентированные на автоматизацию эксперимента, измерений или математическую обработку их результатов. Для простых задач или широко тиражируемых приложений бывает экономически эффективно использовать заказное программирование на C++ или Visual Basic с применением покупных ActiveX элементов, снижающих трудоемкость разработки.

Развитие программных средств автоматизации

Для решения перечисленных выше задач первоначально использовались универсальные языки программирования высокого уровня и команда профессиональных программистов. Однако практика показала крайне низкую эффективность такой разработки. Оказалось, что разработка системы должна выполняться не программистами, а специалистами той предметной области, которая нуждается в автоматизации, т. е. технологами, а также системными интеграторами, которые осуществляют комплексное внедрение системы.

Необходимость в разработке средств программирования, предназначенных специально для систем автоматизации и ориентированных на технологов, была вызвана следующими причинами:

- требованием надежности программного обеспечения. Система, написанная целиком на алгоритмическом языке для конкретного заказа,

содержала слишком много программного кода, на тщательную разработку и тестирование которого не хватало времени;

- сжатыми сроками внедрения системы и ограниченной стоимостью работ. Для создания системы в короткий срок при ограниченном бюджете требовалось большое количество готовых универсальных программных компонентов, уже написанных и тщательно оттестированных;
- необходимостью модификации системы в процессе ее эксплуатации. Внести изменения в специализированную программу мог только написавший ее программист, который к этому времени обычно работал уже на другом предприятии. Поэтому вместо того, чтобы модифицировать программное обеспечение, его приходилось переписывать заново;
- требованиями совместимости с другими системами автоматизации, работающими на том же предприятии. Были необходимы стандартные интерфейсы между программами, созданными разными производителями на разных аппаратно-программных платформах;
- высокими требованиями к качеству пользовательского интерфейса. Ограниченный бюджет времени и финансовых ресурсов не позволял разработать достаточно хороший программный интерфейс на универсальных алгоритмических языках.

Разделение труда по созданию программных средств автоматизации

Перечисленные причины привели к следующему разделению труда по созданию программных средств для систем автоматизации: фирмы, специализирующиеся на программном обеспечении, создают универсальные системы программирования задач автоматизации (SCADA-пакеты и средства МЭК-программирования), а инжиниринговые фирмы (системные интеграторы) адаптируют эти средства к нуждам конкретного заказчика. В результате достигается решение всех перечисленных выше проблем. Более того, благодаря существенному упрощению процесса адаптации по сравнению с классическим программированием изменения в алгоритмы управления могут быть внесены, например, технологом эксплуатирующей организации без привлечения системных интеграторов или программистов.

В настоящее время заказные программы естественным путем вытеснены с рынка промышленной автоматизации SCADA-пакетами и аналогичными универсальными средствами автоматизации, а также средствами программирования контроллеров на языках стандарта МЭК 61131-3.

Графическое программирование

Языки визуального программирования появились в начале 90-х годов и содержат большое число стандартных функций и библиотек, а также готовых средств визуализации. Они позволяют создавать очень удобные и эффектные программы, однако достигается это за счет резкого увеличения объема программного кода. Поэтому языки визуального программирования, как и текстовые, по-прежнему не позволяют модифицировать алгоритмы силами технологов без участия профессиональных программистов.

Настоящую революцию в программировании систем автоматизации сделали языки графического программирования. Одним из первых в этом классе был графический язык среды Simulink, входящей в состав Matlab (MathWorks Inc), а также языки LabVIEW (National Instruments) и HP-VEE (Hewlett Packard). Они были предназначены и успешно использовались для сбора данных, моделирования систем автоматизации, автоматического управления, обработки собранных данных и их визуального представления в виде графиков, таблиц, звука, с помощью компьютерной анимации. Графические языки были настолько простыми и естественными, что для их освоения зачастую было достаточно метода проб и ошибок без использования учебников и консультаций. Человек, не знакомый с программированием на алгоритмических языках, пользуясь только логикой и понимая постановку прикладной задачи, мог собрать работающее приложение из готовых компонентов, набрасывая их мышкой на экран монитора и проводя графические связи для указания потоков информации.

Первые языки программирования алгоритмов работы систем автоматизации были нестандартными. Каждая фирма, создававшая контроллер или SCADA-пакет, предлагала свой язык. Это требовало от системных интеграторов дополнительных усилий и затрудняло освоение новых SCADA пакетов и средств программирования контроллеров.

Поэтому появление в 1993 году стандарта на языки программирования контроллеров МЭК 61131-3 стало большим шагом в направлении создания открытых систем автоматизации и обеспечило снижение стоимости разработки, сокращение сроков, повышение качества реализации алгоритмов автоматизации и возможность детального изучения языков программирования, пригодных для любого контроллера. МЭК 61131-3 устанавливал стандарты для пяти языков программирования, рассчитанных на специалистов разных профессий, не связанных с программированием.

Графический интерфейс

Создание графических интерфейсов пользователя на компьютере явилось большим достижением в направлении развития средств

диспетчерского управления. Главным эффектом от применения графического интерфейса является существенное снижение количества ошибок, допускаемых оператором (диспетчером) в стрессовых ситуациях при управлении производственными процессами. Проектирование пользовательского интерфейса основано на следующих принципах:

- *узнаваемость*: назначение элементов экрана должно быть понятно без предварительного обучения, допустимые манипуляции с этими элементами также должны быть понятны интуитивно. Пользовательский интерфейс не должен содержать излишней детализации;
- *логичность*: пользователь, имеющий опыт работы с одной программой, должен быть способен быстро, практически без обучения, адаптироваться к любой аналогичной программе;
- *отсутствие "сюрпризов"*: знакомые из прошлого опыта операции с элементами на экране должны вызывать знакомые реакции системы;
- *восстанавливаемость*: система не должна быть чувствительна к ошибкам оператора. Оператор должен иметь возможность отменить любое свое неправильное действие. Для этого используются многократные подтверждения, отмены, возврат на несколько шагов назад, установка контрольных точек и т. п.;
- *наличие удобной справки*, подсказок, встроенных в пользовательский интерфейс, средств контекстного поиска и замены;
- *адаптация к опыту пользователя*: начинающий пользователь должен иметь более простой интерфейс с большим количеством подсказок. Для опытного пользователя количество подсказок должно быть уменьшено, поскольку они мешают в работе.

Связь с физическими устройствами

Связь программного обеспечения с физическими устройствами в системах автоматизации осуществляется с помощью методов DDE, OLE, COM, DCOM и OPC.

Технология обмена данными между приложениями Windows с аббревиатурой DDE (Dynamical Data Exchange - "динамический обмен данными") - появилась в 1987 г. вместе с Windows 2.0. В промышленной автоматизации DDE использовалась для обмена данными между SCADA в качестве DDE-клиента и физическим устройством, которое поставлялось с DDE сервером.

После появления OLE (Object Linking and Embedding - "связывание и внедрение объектов") фирмы Microsoft, а позже COM (Component Object Model - "модель многокомпонентных объектов") и DCOM(Distributed COM - "COM для распределенных систем") технология DDE была полностью вытеснена этими новыми средствами, которые оказались гораздо более эффективными.

Технология COM предоставляет средства для взаимодействия между разрозненными программными модулями, написанными на разных языках программирования, которые собираются в единую систему во время исполнения. Взаимодействие COM объекта с другими программами или программными модулями выполняется через программные интерфейсы с использованием метода "клиент-сервер".

Одной из составляющих COM является Automation - средства взаимодействия программ, написанных на C++ с программами на языке VBA (Visual Basic for Application) или Delphi, а также с программами на языках сценариев (VBScript, JScript). Благодаря автоматизации COM-объект может быть также размещен и исполняться на веб-странице.

Расширение COM в виде DCOM позволяет программам взаимодействовать между собой, даже если они исполняются на разных компьютерах локальной сети. Поэтому DCOM явилась универсальной программной технологией, которая как нельзя лучше позволяет осуществить взаимодействие между SCADA в качестве клиента и сервером, обеспечивающим интерфейс к аппаратным средствам промышленной автоматизации. Именно благодаря этому свойству DCOM была использована в качестве базы для разработки стандарта OPC - "OLE for Process Control" - "OLE для управления процессами", который лежит в основе всех современных SCADA пакетов, взаимодействующих с аппаратурой через OPC сервер.

Базы данных

Системы автоматизации работают с большими объемами данных, которые необходимо хранить, сортировать, группировать, извлекать и представлять в виде, удобном для пользователя. Данные извлекаются с помощью языка запросов SQL (Structured Query Language - "структурированный язык запросов"), который стал стандартом в системах автоматизации. Наиболее распространенными системами управления базами данных (СУБД) являются Microsoft SQL Server, Wonderware Industrial SQL Server, Microsoft Access и Excel. Основными свойствами СУБД являются:

- наличие пользовательского интерфейса на базе языка запросов SQL;

- возможность одновременного обслуживания нескольких пользователей;
- корректность работы с данными.

Открытые системы используют обращение к СУБД через драйвер ODBC (Open Database Connectivity - "подключение к открытой базе данных"). ODBC используется, когда необходимо обеспечить независимость прикладной программы от типа СУБД или типа операционной системы и требуется подключиться к нескольким различным СУБД (например, одновременно к MS SQL Server, MS Excel, MS Access, Paradox и др.). При использовании нескольких ODBC драйверов ими управляет менеджер драйверов. ODBC драйвер транслирует стандартный SQL запрос в формат запроса для конкретной СУБД. Таким образом, для работы с новой базой данных пользователю достаточно добавить в систему новый ODBC драйвер, не изменяя прикладную программу.

Системы программирования на языках МЭК 61131-3

Стандарт МЭК 61131-3 устанавливает пять языков программирования ПЛК, три графических и два текстовых. Первоначально стандарт назывался IEC 1131-3 и был опубликован в 1993 г. но в 1997 г. МЭК (IEC) перешел на новую систему обозначений и в названии стандарта добавилась цифра "6". Продвижением стандарта занимается организация PLCopen.

Основной целью стандарта было повышение скорости и качества разработки программ для ПЛК, а также создание языков программирования, ориентированных на технологов, обеспечение соответствия ПЛК идеологии открытых систем, исключение этапа дополнительного обучения при смене типа ПЛК.

Системы программирования, основанные на МЭК 61131-3, характеризуются следующими показателями:

- надежностью создаваемого программного обеспечения. Надежность обеспечивается тем, что программы для ПЛК создаются с помощью специально предназначенной для этого среды разработки, которая содержит все необходимые средства для написания, тестирования и отладки программ с помощью эмуляторов и реальных ПЛК, а также множество готовых фрагментов программного кода;
- возможностью простой модификации программы и наращивания ее функциональности;
- переносимостью проекта с одного ПЛК на другой;
- возможностью повторного использования отработанных фрагментов программы;

– простотой языка и ограничением количества его элементов.

Языки МЭК 61131-3 появились не как теоретическая разработка, а как результат анализа множества языков, уже используемых на практике и предлагаемых рынку производителями ПЛК. Стандарт устанавливает пять языков программирования со следующими названиями:

- структурированный текст (ST - Structured Text);
- последовательные функциональные схемы (SFC - "Sequential Function Chart");
- диаграммы функциональных блоков (FBD - Function Block Diagram);
- релейно-контактные схемы, или релейные диаграммы (LD - Ladder Diagram);
- список инструкций (IL - Instruction List).

Графическими языками являются SFC, FBD, LD. Языки IL и ST являются текстовыми.

В стандарт были введены несколько языков (а не один) для того, чтобы каждый пользователь мог применить наиболее понятный ему язык. Программисты чаще выбирают язык IL (похожий на ассемблер) или ST, похожий на язык высокого уровня Паскаль; специалисты, имеющие опыт работы с релейной логикой, выбирают язык LD, специалисты по системам автоматического управления (САУ) и схемотехники выбирают привычный для них язык FBD.

Выбор одного из пяти языков определятся не только предпочтениями пользователя, но и смыслом решаемой задачи. Если исходная задача формулируется в терминах последовательной обработки и передачи сигналов, то для нее проще и нагляднее использовать язык FBD. Если задача описывается как последовательность срабатываний некоторых ключей и реле, то для нее нагляднее всего будет язык LD. Для задач, которые изначально формулируются в виде сложного разветвленного алгоритма, удобнее будет язык ST.

- Языки МЭК 61131-3 базируются на следующих принципах:
- вся программа разбивается на множество функциональных элементов - Program Organization Units (POU), каждый из которых может состоять из функций, функциональных блоков и программ. Любой элемент МЭК-программы может быть сконструирован иерархически из более простых элементов;
- стандарт требует строгой типизации данных. Указание типов данных позволяет легко обнаруживать большинство ошибок в программе до ее исполнения;

- имеются средства для исполнения разных фрагментов программы в разное время, с разной скоростью, а также параллельно. Например, один фрагмент программы может сканировать концевой датчик с частотой 100 раз в секунду, в то время как второй фрагмент будет сканировать датчик температуры с частотой один раз в 10 сек;
- для выполнение операций в определенной последовательности, которая задается моментами времени или событиями, используется специальный язык последовательных функциональных схем (SFC);
- стандарт поддерживает структуры для описания разнородных данных. Например, температуру подшипников насоса, давление и состояние "включено-выключено" можно описать с помощью единой структуры "Pomp" и передавать ее внутри программы как единый элемент данных;
- стандарт обеспечивает совместное использование всех пяти языков, поэтому для каждого фрагмента задачи может быть выбран любой, наиболее удобный, язык;
- программа, написанная для одного контроллера, может быть перенесена на любой контроллер, совместимый со стандартом МЭК 61131-3.

Любой ПЛК работает в циклическом режиме. Цикл начинается со сбора данных с модулей ввода, затем исполняется программа ПЛК и оканчивается цикл выводом данных в устройства вывода. Поэтому величина контроллерного цикла зависит от времени исполнения программы и быстродействия процессорного модуля.

Язык релейно-контактных схем, LD

Графический язык релейной логики впервые появился в виде электрических схем, которые состояли из контактов и обмоток электромагнитных реле. Такие схемы использовались в автоматике конвейеров для сборки автомобилей до эры микропроцессоров. Язык релейной логики был интуитивно понятен людям, слегка знакомым с электротехникой и поэтому оказался наиболее распространенным в промышленной автоматике. Обслуживающий персонал легко находил отказ в оборудовании, прослеживая путь сигнала по релейной диаграмме.

Однако язык LD проблематично использовать для реализации сложных алгоритмов, поскольку он не поддерживает подпрограммы, функции, инкапсуляцию и другие средства структурирования программ с целью повышения качества программирования. Эти недостатки затрудняют

многократное использование программных компонентов, что делает программу длинной и сложной для обслуживания.

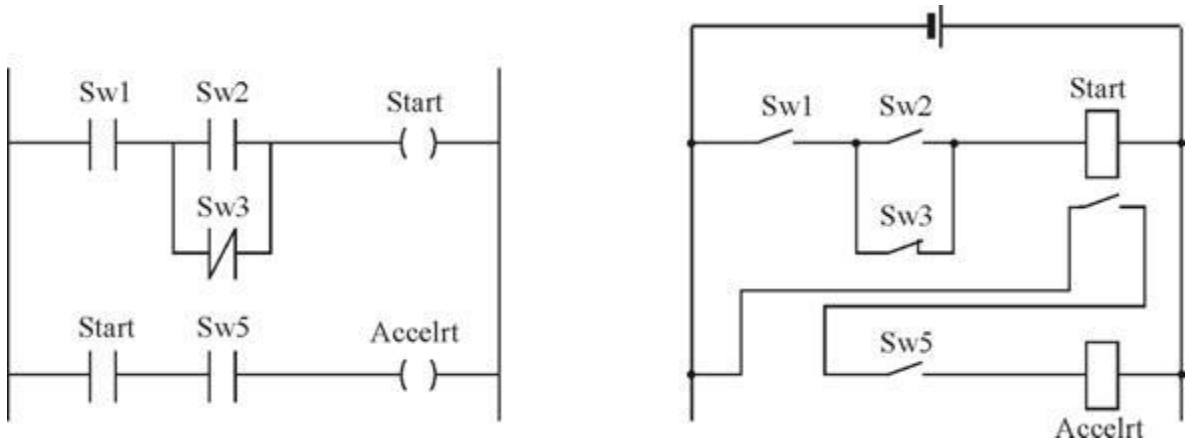


Рис. 1. Пример программы на языке LD (слева) и ее эквивалент в виде электрической цепи с реле и выключателями (справа)

Для выполнения арифметических функций в язык LD были добавлены функциональные блоки, которые выполняли операции сложения, умножения, вычисления среднего и т.д. Сложные вычисления в этом языке невозможны. Недостатком является также то, что только маленькая часть программы умещается на мониторе компьютера или панели оператора при программировании.

Несмотря на указанные недостатки, язык LD относится к наиболее распространенным в мире, хотя используется для программирования только простых задач.

Список инструкций, IL

Язык IL напоминает ассемблер и используется для реализации функций, функциональных блоков и программ, а также шагов и переходов в языке SFC. Основным достоинством языка является простота его изучения. Наиболее часто язык IL используется в случаях, когда требуется получить оптимизированный код для реализации критических секций программы, а также для решения небольших задач с малым количеством разветвлений алгоритма.

Листинг 4. Пример программы на языке IL

Метки	Операторы	Операнды	Комментарии
	LD	Voltage	(*Загрузить Voltage в аккумулятор*)
	GT	220	(*Если >220*)
	JMPCN	M1	(*Перейти к метке, если ">220" не верно*)

```
      LD      Current   (*Загрузить Current в аккумулятор*)
      SUB     10        (*Вычесть из аккумулятора 10 *)
      ST      Current   (*Присвоить Current значен. аккумулятора*)
M1:   LD      0         (*Загрузить в аккумулятор значение "0"*)
      ST      Out       (*Присвоить Out значение аккумулятора*)
```

В основе языка лежит понятие аккумулятора и переходов по меткам. Пример программы на зыке IL с комментариями приведен в листинге 4. Начинается программа с загрузки в аккумулятор значения переменной. Дальнейшие шаги программы состоят в извлечении содержимого аккумулятора и выполнении над ним ограниченного числа допустимых действий (их в языке всего 24).

Структурированный текст, ST

Язык ST является текстовым языком высокого уровня и очень сильно напоминает Паскаль:

Листинг 5. Пример программы на языке ST

```
IF Voltage>220 THEN
   Current:=Current - 10;  (*Если V>220 В, то уменьшить ток на 10*)
ELSE
   Current:=50; Speed:= ON;(*Установить ток 50А и включить мотор*)
END_IF;
```

Язык ST имеет много отличий от языка Паскаль и разработан специально для программирования ПЛК. Он содержит множество конструкций для присвоения значений переменным, для вызова функций и функциональных блоков, для написания выражений условных переходов, выбора операторов, для построения итерационных процессов. Этот язык предназначен в основном для выполнения сложных математических вычислений, описания сложных функций, функциональных блоков и программ.

Диаграммы функциональных блоков, FBD

FBD является графическим языком и наиболее удобен для программирования процессов прохождения сигналов через функциональные

блоки. Язык FBD удобен для схемотехников, которые легко могут составить электрическую схему системы управления на "жесткой логике", но не имеют опыта программирования.

Функциональные блоки представляют собой фрагменты программ, написанных на IL, SFC или других языках, которые могут быть многократно использованы в разных частях программы и которым соответствует графическое изображение, принятое при разработке функциональных схем электронных устройств, см. рис. 2.

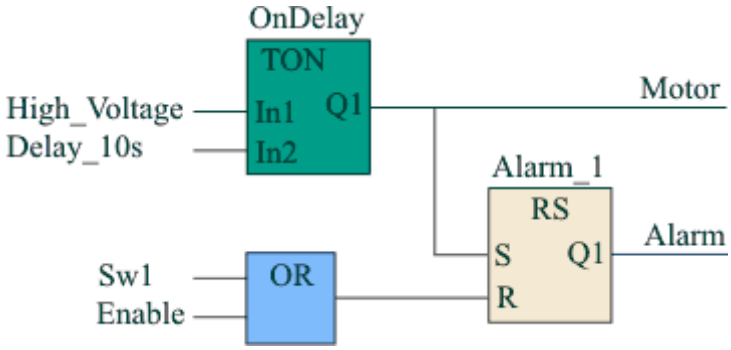


Рис. 2. Пример программы на языке FBD

Язык FBD может быть использован для программирования функций, функциональных блоков и программ, а также для описания шагов и переходов в языке SFC. Функциональные блоки инкапсулируют данные и методы, чем напоминают объектно-ориентированные языки программирования, но не поддерживают наследование и полиморфизм.

Типичным применением языка FBD является описание "жесткой логики" и замкнутых контуров систем управления. Язык функциональных блоков является удобным также для создания и пополнения библиотеки типовых функциональных блоков, которую можно многократно использовать при программирования задач промышленной автоматизации. К типовым блокам относятся блок таймера, ПИД-регулятора, блок секвенсора, триггера, генератора импульсов, фильтра, и т. п.

Последовательные функциональные схемы, SFC

SFC называют языком программирования, хотя по сути это не язык, а вспомогательное средство для структурирования программ. Он предназначен специально для программирования последовательности выполнения действий системой управления, когда эти действия должны быть выполнены в заданные моменты времени или при наступлении некоторых событий. В его основе лежит представление системы управления с помощью понятий состояний и переходов между ними.

Язык SFC предназначен для описания системы управления на самом верхнем уровне абстракции, например, в терминах "Старт", "Наполнение

автоклава", "Выполнение этапа №1", "Выполнение этапа №2", "Выгрузка из автоклава". Язык SFC может быть использован также для программирования отдельных функциональных блоков, если алгоритм их работы естественным образом описывается с помощью понятий состояний и переходов. Например, алгоритм автоматического соединения модема с коммутируемой линией описывается состояниями "Включение", "Обнаружение тона", "Набор номер", "Идентификация сигнала" и переходами "Если длинный - то ждать 20 сек", "Если короткий - перейти в состояние "Набор Номера"" и т.д.

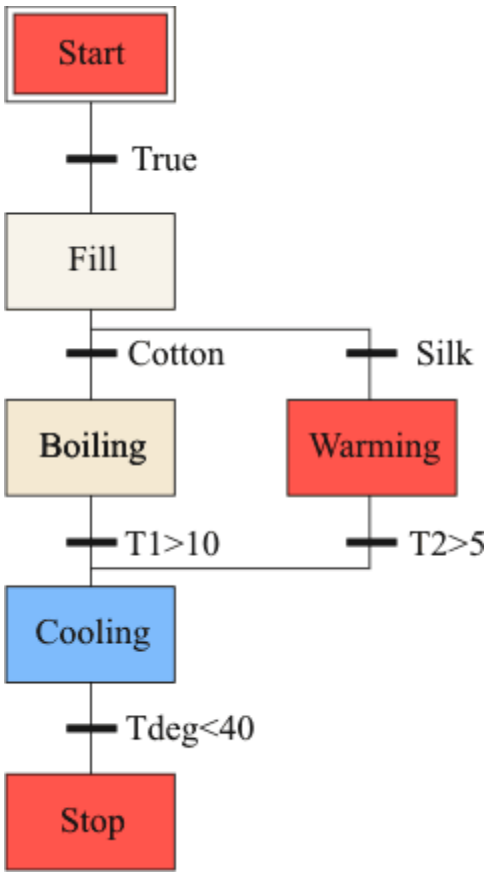


Рис. 4. Пример программы на языке SFC

На рис. 4 показан фрагмент программы на языке SFC. Программа состоит из шагов и условий переходов. Шаги показываются на схеме прямоугольниками, условия переходов - жирной перечеркивающей линией. Программа выполняется сверху вниз. Начальный шаг на схеме показывается в виде двойного прямоугольника. Условия переходов записываются рядом с их обозначениями. Каждый шаг программы может представлять собой реализацию сложного алгоритма, написанного на одном из МЭК-языков.

Программное обеспечение

Программирование ПЛК на описанных выше языках МЭК 61131-3 осуществляется с помощью специализированного программного обеспечения, которое разрабатывается производителями ПЛК или фирмами,

специализирующимися на создании ПО для систем автоматизации. Наиболее известными в мире являются системы CoDeSys фирмы 3S и ISaGRAF фирмы ICS Triplex.

CoDeSys

CoDeSys (Co ntroller Development System) представляет собой комплекс программ для проектирования прикладного программного обеспечения, отладки в режиме эмуляции и загрузки программы в ПЛК. Основными частями системы являются среда разработки программы и среда ее исполнения (CoDeSys SP), которая находится в ПЛК.

В CoDeSys входят графические и текстовые редакторы для всех пяти языков МЭК 61131-3. Этот комплекс полностью реализует требования стандарта и дополнительно вводит ряд оригинальных расширений, самым удобным из которых является объектно-ориентированные программирование. Однако расширениями языка можно не пользоваться, чтобы сохранить требования к совместимости языков, предъявляемое к открытым системам.

В одном проекте может быть использовано несколько контроллеров разных производителей. Каждый из них может программироваться как независимое устройство или с учетом их взаимодействия в промышленной сети. Проект состоит из нескольких приложений, распределенных по нескольким контроллерам. В одном ПЛК может существовать несколько независимых приложений.

Программа, написанная на языках МЭК, компилируется системой CoDeSys в машинный код, оптимизированный для заданной аппаратной платформы. Компилятор выдает диагностические сообщения как на этапе компиляции, так и на этапе ввода операторов языка.

Машинный код, сгенерированный компилятором CoDeSys, загружается в ПЛК, после чего разработчик имеет возможность использовать широкий набор функций для быстрой и эффективной отладки приложения. Текущие значения переменных видны непосредственно в редакторах программ. Программу можно выполнять по шагам или по контроллерным циклам. Можно задавать точки останова программы, просматривать стек вызовов, подготавливать связные наборы значений переменных и загружать их одной командой.

При отсутствии реального контроллера отладку программы можно выполнять с помощью встроенного программного эмулятора.

Система имеет также встроенный многоканальный программный трассировщик (графический самописец) значений переменных. Он позволяет наглядно представить динамически изменяющиеся данные проекта. Данные аккумулируются в памяти ПЛК и могут синхронизироваться с определенными

событиями. Трассировщик полезен не только при отладке, но и при анализе нештатных ситуаций в процессе эксплуатации оборудования.

После изменения программы во время отладки перекомпилируются только измененные части программы. Их можно подгружать в контроллер без остановки выполнения прикладной программы. Эта возможность системы называется "горячим обновлением" кода.

Программируемое устройство соединяется с CoDeSys через вспомогательный программный компонент – шлюз связи, который использует протокол TCP/IP. Шлюз работает на компьютере программиста или удаленно, например, через интернет или сеть Ethernet. Контроллер подключается компьютеру через любой последовательный канал или сеть. Добавив драйвер, изготовитель ПК может поддержать свой оригинальный протокол связи.

Общение ПЛК со SCADA осуществляется с помощью стандартного OPC сервера.

Для того, чтобы ПЛК можно было программировать с помощью CoDeSys, в контроллере должна быть установлена система исполнения. Установку системы выполняет изготовитель контроллера. Изготовитель обеспечивает также поддержку всех модулей ПЛК, поэтому конечный пользователь может сосредоточиться на разработке только прикладной программы.

Среда исполнения CoDeSys может функционировать в ПЛК под управлением различных операционных систем или вообще без них, в том числе на обычном персональном компьютере. Собственное ядро реального времени может устанавливать контроллерный цикл с точностью до нескольких микросекунд. Прикладная программа остается работоспособной даже при зависании ОС.

Помимо средств программирования, CoDeSys имеет встроенную систему визуализации, которая применяется для операторского управления, а также моделирования на этапе разработки. Визуализацию можно запустить на компьютере, графической панели ПЛК или встроенном в контроллер web-сервере.

Пользователь может самостоятельно расширять возможность CoDeSys путем создания библиотек программных модулей. Например, он может реализовать поддержку нестандартных интерфейсов.

Комплекс программирования CoDeSys построен по компонентной технологии Microsoft на базе автоматизации. Поэтому изготовитель ПЛК может включить в комплекс свои собственные компоненты, от конфигуратора оригинальной сети до собственного языка программирования ПЛК.

Для систем, связанных с безопасностью, CoDeSys имеет библиотеку функциональных блоков PLCopen Safety, систему исполнения для

оборудования с дублированием и специализированное расширение среды программирования. При внезапном отключении питания CoDeSys автоматически сохраняет значения переменных во флеш-памяти или в ОЗУ с батарейным питанием.

ISaGRAF

Система ISaGRAF фирмы ICS Triplex также состоит из среды разработки и среды исполнения. Среда исполнения может функционировать практически на любой операционной системе и любой аппаратной платформе, включая персональный компьютер. Среда разработки поддерживает все пять зыков МЭК 61131-3 и функциональные блоки МЭК 61499, имеет средства для редактирования, компиляции, документирования, управления библиотеками, архивирования, моделирования системы при отсутствии реального ПЛК и отладки с подключенным ПЛК.

Комплекс программ ISaGRAF первый на рынке использовал новый стандарт МЭК 61499 для программирования распределенных систем управления.

Связь между SCADA пакетом и контроллером, запрограммированным с помощью ISaGRAF, осуществляется с помощью стандартного OPC сервера.

Среда исполнения создается и загружается в контроллер производителем ПЛК и является независимой от исполняемой в ней программы пользователя.

Среда разработки имеет знакомый по Windows-приложениям интерфейс с подсказками, панелями инструментов, окнами, с функциями вставки и замены и т. п. Код, полученный на выходе среды разработки, может исполняться на любой аппаратно-программной платформе без изменений, если на ней предварительно установлена среда исполнения. Среда разработки может также транслировать пользовательскую программу, написанную на МЭК-языках, в текст на языке Си.

Контрольные вопросы по теме

Уровень курса

1. Системы программирования на языках МЭК 61131-3.
2. Языки МЭК 61131-3: язык релейно-контактных схем, LD.
3. Языки МЭК 61131-3: список инструкций IL и структурированный текст ST.
4. Языки МЭК 61131-3: диаграммы функциональных блоков FBD.
5. Языки МЭК 61131-3: последовательные функциональные схемы SFC.

Лекции № 9 ***Тема:*** SCADA-системы

Оглавление

Источники:

1. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.: ил.

   https://rutracker.org/forum/viewtopic.php?t=4956359

2. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил.
3. Бабак В.П. Теоретические основы информационно-измерительных систем: Учебник / В.П. Бабак, С.В. Бабак, В.С. Ерёменко и др. – К., 2014. – 832 с.

Пользовательский интерфейс, SCADA-пакеты

Большинство систем автоматизации функционирует с участием человека (оператора, диспетчера). Интерфейс между человеком и системой называют человеко-машинным интерфейсом (ЧМИ), в зарубежной литературе - HMI (Human-Machinery Interface) или MMI (Man-Machinery Interface). В частном случае, когда ЧМИ предназначен для взаимодействия человека с автоматизированным технологическим процессом, его называют SCADA-системой (Supervisory Control And Data Acquisition). Этот термин переводится буквально как "диспетчерское управление и сбор данных", но на практике его трактуют гораздо шире, а современные SCADA-пакеты включают в себя широчайший набор функциональных возможностей, далеко выходящий за рамки сбора данных и диспетчерского управления.

Рис. 1 Пульт оператора технологического процесса

Функции SCADA

Существующие в настоящее время SCADA-пакеты выполняют множество функций, которые можно разделить на несколько групп:

- настройка SCADA на конкретную задачу (т. е. разработка программной части системы автоматизации);
- диспетчерское управление;
- автоматическое управление;
- хранение истории процессов;
- выполнение функций безопасности;
- выполнение общесистемных функций.

Несмотря на множество функций, выполняемых SCADA, основным ее отличительным признаком является наличие интерфейса с пользователем. При отсутствии такого интерфейса перечисленные выше функции совпадают с функциями средств программирования контроллеров, а управление является автоматическим, в противоположность диспетчерскому.

Качество решений, принятых оператором (диспетчером), часто влияет не только на качество производимой продукции, но и на жизнь людей. Поэтому комфорт рабочего места, понятность интерфейса, наличие подсказок и блокировка явных ошибок оператора являются наиболее важными свойствами SCADA, а дальнейшее их развитие осуществляется в направлении улучшения эргономики и создания экспертных подсистем.

Иногда SCADA комплектуются средствами для программирования контроллеров, однако эта функция вызвана коммерческими соображениями и слабо связана с основным назначением SCADA.

В SCADA-пакетах используют понятие *аларма* и *события*. Событие - это изменение некоторых состояний в системе. Примерами событий могут быть включение перевалки зерна в элеваторе, завершение цикла периодического процесса обработки детали, окончание загрузки бункера, регистрация нового оператора и т. п. События не требуют срочного вмешательства оператора, а просто информируют его о состоянии системы.

В отличие от события, аларм (от английского "alarm" - "сигнал тревоги") представляет собой предупреждение о важном событии, в ответ на которое нужно срочно предпринять некоторые действия. У английского слова "аларм" имеется точный русский перевод - "сигнал тревоги" или "аварийный сигнал", однако термин "аларм" уже прочно вошел в лексикон промышленной автоматизации.

Примерами алармов может быть достижение критической температуры хранения зерна в элеваторе, после которого начинается его возгорание, достижение критического значение давления в автоклаве, после которого возможен разрыв оболочки, срабатывание датчика открытия охраняемой двери, превышение допустимого уровня загазованности в котельной и т.п.

В связи с тем, что алармы требует принятия решения, их делят на подтвержденные и неподтвержденные. Подтвержденным называется аларм, в ответ на который оператор ввел команду подтверждения. До этого момента аларм считается неподтвержденным.

Алармы делятся на дискретные и аналоговые. Дискретные сигнализируют об изменении дискретной переменной, аналоговые алармы появляются, когда непрерывная переменная $y(t)$ входит в заранее заданный интервал своих значений. В качестве примера на рис. 5 показано деление всего интервала изменения переменной $y(t)$ на интервалы "Норма", "Внимание" (предаварийное состояние) и "Авария":

- аларм "Внимание" возникает при $y(a) < y(t) < y(b)$ во время нарастания наблюдаемой переменной и при $y(d) < y(t) < y(c)$ во время ее уменьшения;

- аларм "Авария" возникает при $y(b) < y(t)$.

Каждая критическая граница на рис. 5 имеет зону нечувствительности (мертвую зону), которая нужна для того, чтобы после снятия состояния аларма переменная не могла вернуться в него вследствие случайных выбросов в системе (шумов). Границы зон на рис. 5 могут изменяться с течением времени.

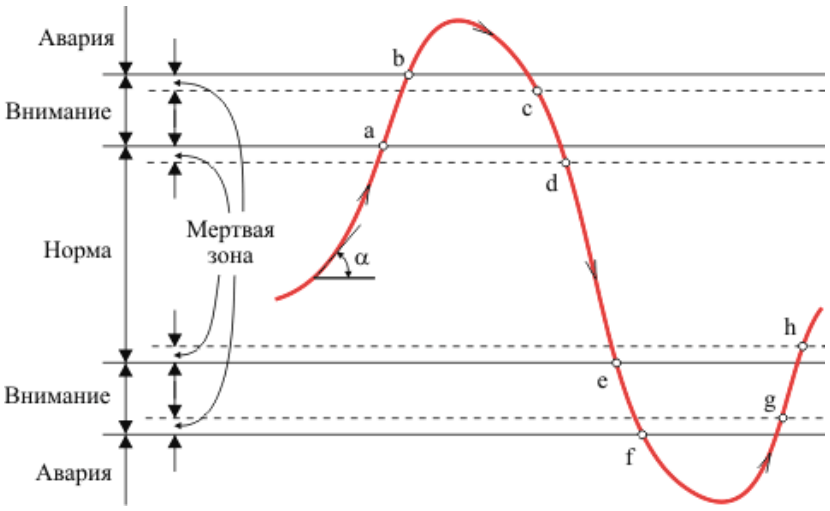


Рис. 2. Пример назначения интервалов аналоговым алармам

Аналогичные границы могут быть назначены для скорости изменения переменной (для производной функции $y(t)$), которая определяется как угол α наклона касательной к кривой $y(t)$.

Методика выдачи алармов должна быть надежной. В частности, всплывающие окна с сообщениями алармов должны быть всегда поверх остальных окон, алармы могут дублироваться звуком и светом. Поскольку алармов в системе может быть много, им назначают разные приоритеты, разные громкости и тоны звукового сигнала и т. п.

Разработка человеко-машинного интерфейса

Одной из основных функций SCADA является разработка человеко-машинного интерфейса, т.е. SCADA одновременно является и ЧМИ, и инструментом для его создания. Быстрота разработки существенно влияет на рентабельность фирмы, выполняющей работу по внедрению системы автоматизации, поэтому скорость разработки является основным показателем качества SCADA с точки зрения системного интегратора. В процесс разработки входят следующие операции:

- создание графического интерфейса (мнемосхем, графиков, таблиц, всплывающих окон, элементов для ввода команд оператора и т д.);

- программирование и отладка алгоритмов работы системы автоматизации. Многие SCADA позволяют выполнять отладку системы как в режиме эмуляции оборудования, так и с подключенным оборудованием;
- настройка системы коммуникации (сетей, модемов, коммуникационные контроллеров и т п.);
- создание баз данных и подключение к ним SCADA.

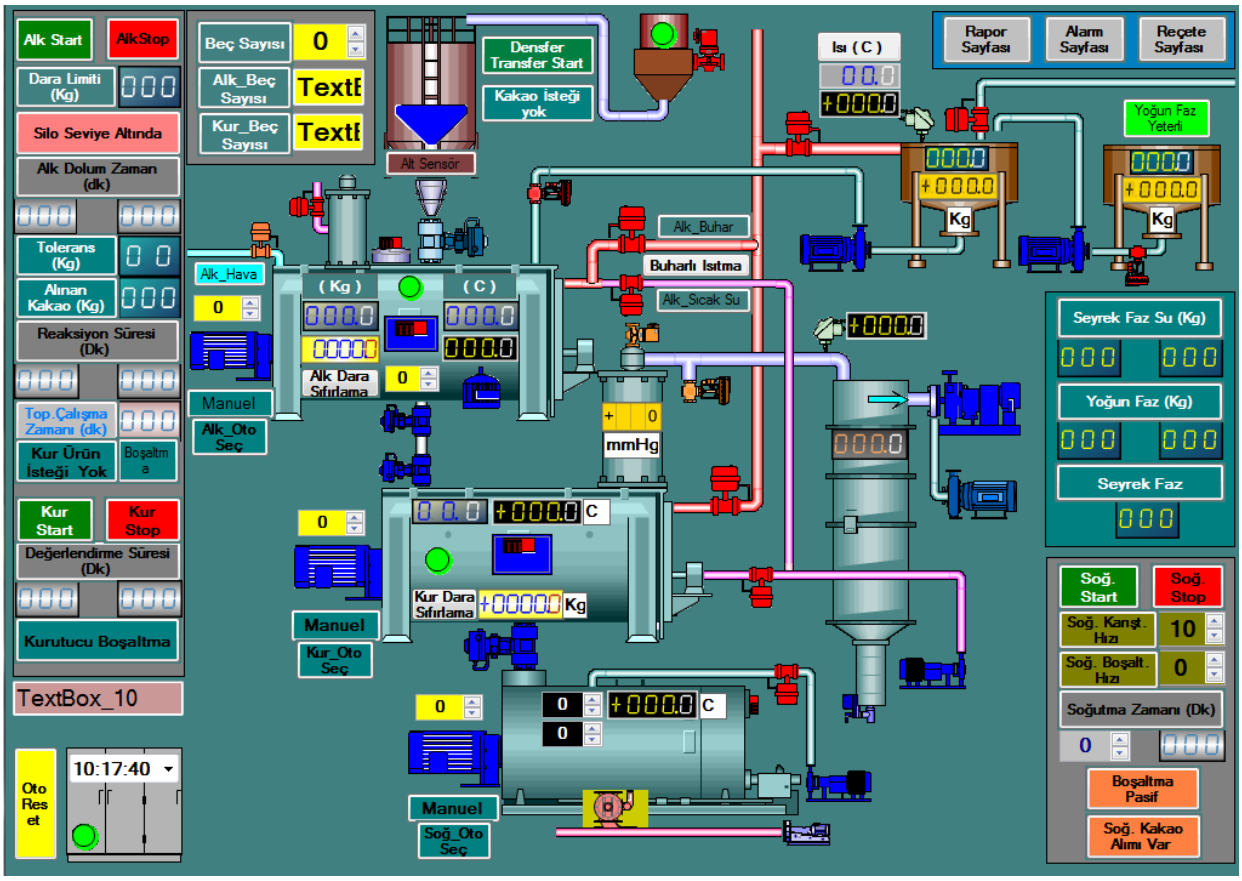


Рис. 3 Пример экрана пульта оператора в SCADA-системе

SCADA как система диспетчерского управления

Как система диспетчерского управления SCADA может выполнять следующие задачи:

- взаимодействие с оператором (выдача визуальной и слуховой информации, передача в систему команд оператора);
- помощь оператору в принятии решений (функции экспертной системы);
- автоматическая сигнализация об авариях и критических ситуациях;
- выдача информационных сообщений на пульт оператора;
- ведение журнала событий в системе;

– извлечение информации из архива и представление ее оператору в удобном для восприятия виде;
– подготовка отчетов (например, распечатка таблицы температур, графиков смены операторов, перечня действий оператора);
– учет наработки технологического оборудования.

SCADA как часть системы автоматического управления

Основная часть задач автоматического управления выполняется, как правило, с помощью ПЛК, однако часть задач может возлагаться на SCADA. Кроме того, во многих небольших системах управления ПЛК могут вообще отсутствовать и тогда компьютер с установленной SCADA является единственным средством управления. SCADA обычно выполняет следующие задачи автоматического управления:

- автоматическое регулирование;
- управление последовательностью операций в системе автоматизации;
- адаптация к изменению условий протекания технологического процесса;
- автоматическая блокировка исполнительных устройств при выполнении заранее заданных условий.

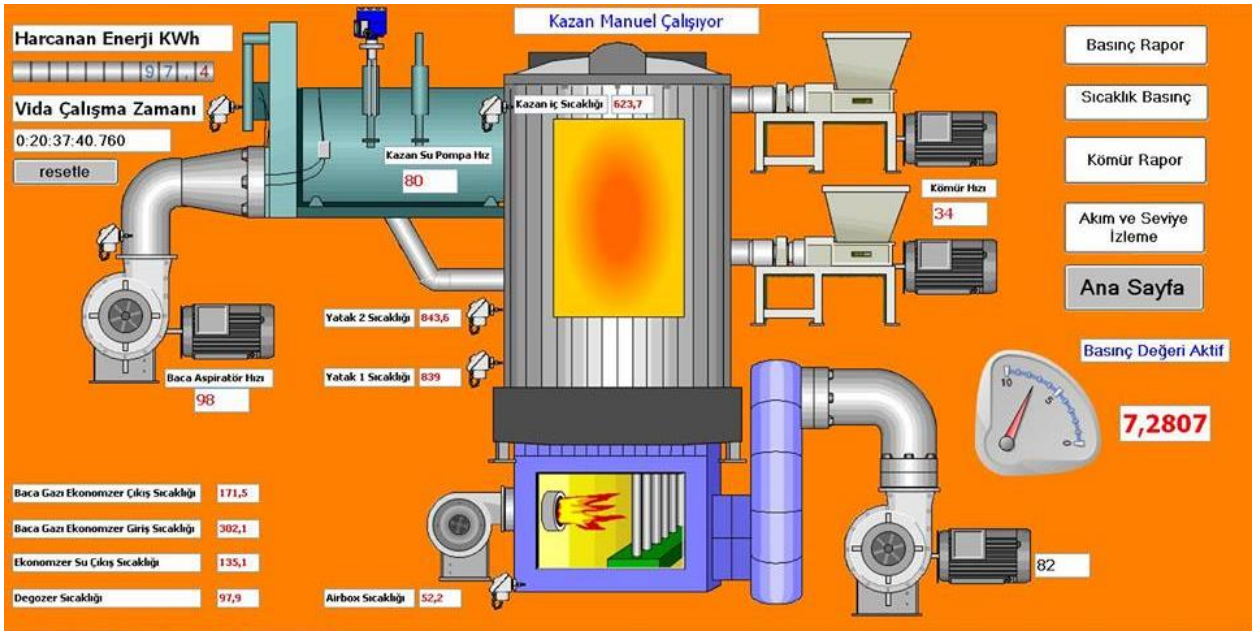


Рис. 4 Пример экрана пульта оператора в SCADA-системе

Хранение истории процесса

Знание предыстории управляемого процесса позволяет улучшить будущее поведение системы, проанализировать причины возникновения опасных ситуаций или брака продукции, выявить ошибки оператора. Для создания истории система выполняет следующие операции:

- сбор данных и их обработка (цифровая фильтрация, интерполяция, сжатие, нормализация, масштабирование и т. д.);
- архивирование данных (действий оператора, собранных и обработанных данных, событий, алармов, графиков, экранных форм, файлов конфигурации, отчетов и т. п.);
- управление базами данных (реального времени и архивных).

Безопасность SCADA

Применение SCADA в системах удаленного доступа через интернет резко повысило уязвимость SCADA к действиям враждебных лиц. Пренебрежение этой проблемой может приводить, например, к отказу в работе сетей электроснабжения, жизнеобеспечения, связи, отказу морских маяков, дорожных светофоров, к заражению воды неочищенными стоками и т.п. Возможны и более тяжелые последствия с человеческими жертвами или большим экономическим ущербом. Для повышения безопасности SCADA используют следующие методы:

- разграничение доступа к системе между разными категориями пользователей (у сменного оператора, технолога, программиста и директора должны быть разные права доступа к информации и к модификации настроек системы);
- защиту информации (путем шифрования информации и обеспечения секретности протоколов связи);
- обеспечение безопасности оператора благодаря его отдалению от опасного управляемого процесса (дистанционное управление). Дистанционный контроль и дистанционное управление являются типовыми требованиями и выполняются по проводной сети, радиоканалу (через GSM- или радиомодем), через интернет и т.д.;
- специальные методы защиты от кибератак;
- применение межсетевых экранов.

Общесистемные функции

Поскольку SCADA обычно является единственной программой для управления системой автоматизации, на нее могут возлагаться также некоторые общесистемные функции:

- осуществление взаимодействий между несколькими SCADA, между SCADA и другими программами (MS Office, базой данных, MATLAB и т.п.);
- диагностика аппаратуры, каналов связи и программного обеспечения.

Свойства SCADA

Анализ свойств различных SCADA позволяет выбирать систему, оптимальную для решения поставленной задачи. Все многообразие свойств SCADA-пакетов можно разбить на следующие группы:

- инструментальные свойства;
- эксплуатационные свойства;
- свойства открытости;
- экономическая эффективность.

Инструментальные свойства

К инструментальным относятся свойства SCADA, влияющие на эффективность работы системных интеграторов:

- быстрота разработки проекта;
- легкость освоения;
- поддерживаемые средства коммуникации;
- наличие функций для сложной обработки данных;
- наличие языков МЭК 61131-3 и универсального алгоритмического языка типа Visual Basic;
- степень открытости для разработчика (поддержка COM и ActiveX для подключения программных модулей пользователя, а также OPC, ODBC, OLE DB;
- качество технической документации (полнота, ясность изложения, количество ошибок);
- наличие режима эмуляции оборудования для отладки;
- наличие внутренних графических редакторов, позволяющих отказаться от применения внешних редакторов типа CorelDraw или Photoshop; поддержка типовых графических форматов файлов;
- качество технической поддержки (время реакции на вопросы пользователей, наличие "горячей линии" технической поддержки.

SCADA используют языки программирования МЭК 61131-3, ориентированные на технологов, которые дополняются функциями, специфическими для SCADA. Большинство SCADA имеют встроенный редактор и интерпретатор языка Visual Basic фирмы Microsoft.

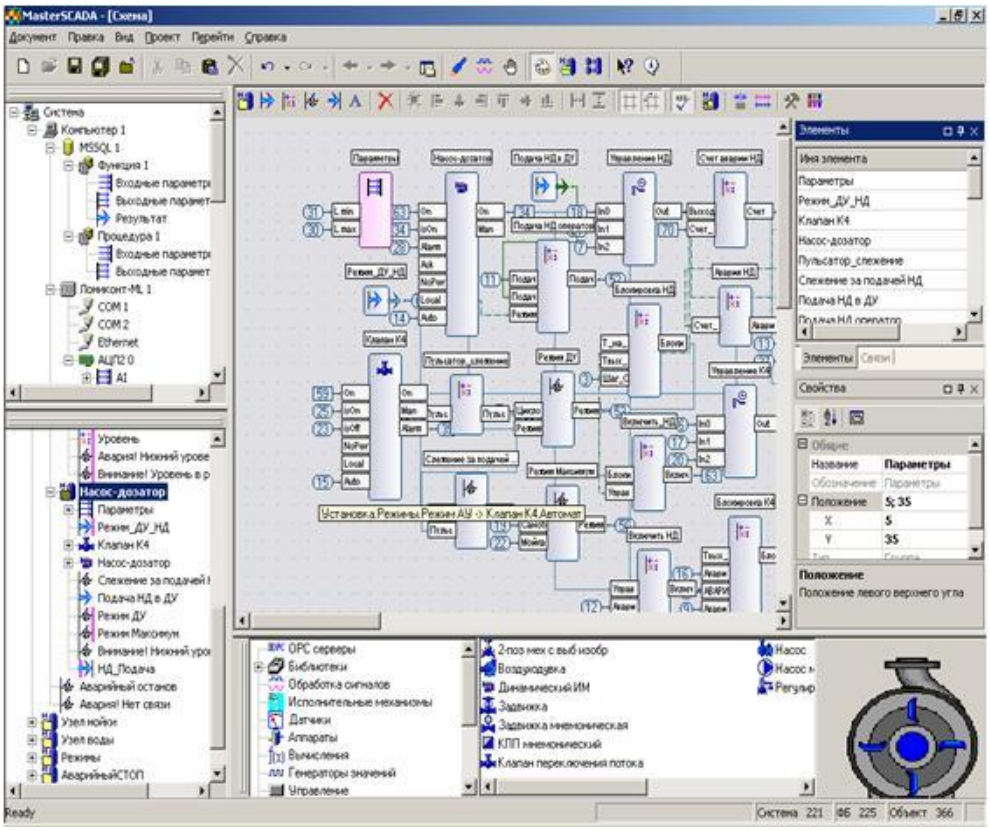


Рис. 5 Среда разработки в SCADA-системе

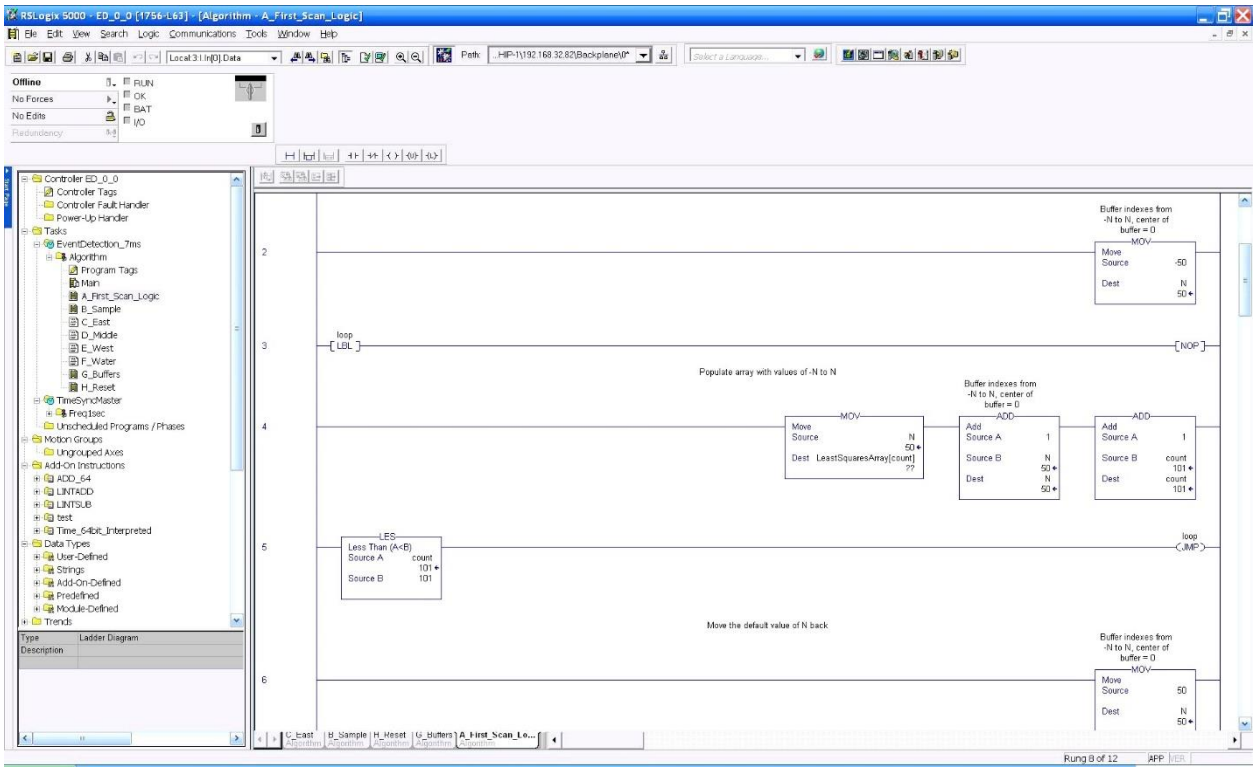


Рис. 6 Программирование ПЛК в SCSDA-системе

Эксплуатационные свойства

Качество SCADA в процессе эксплуатации оценивается конечными пользователями и характеризуется следующим набором свойств:

- робастность (нечувствительность к ошибкам пользователя, защищенность от вандалов и враждебных элементов, устойчивость к ошибкам в исходных данных);
- надежность;
- информационная защищенность;
- наличие средств сохранения данных при нештатных ситуациях, отключениях питания и сбоях;
- наличие автомата перезапуска системы при ее зависании или после прерывания питания;
- поддержка резервирования SCADA (операторской станции, сетевых серверов, клиентских рабочих станций, резервное копирование данных);
- поддержка переключения экранов с разной детализацией изображений; поддержка нескольких мониторов.

Степень открытости

Степень открытости очень сильно влияет на экономическую эффективность системы, однако это влияние носит случайный характер, поскольку зависит от степени использования свойств открытости в конкретном проекте.

Открытость для программирования пользователем SCADA обеспечивается возможностью подключения программных модулей, написанных пользователем или другими производителями. Это обычно достигается тем, что SCADA разрабатывается как контейнер для COM-объектов и ActiveX элементов. Совместимость с аппаратурой и базами данных других производителей достигается с помощью стандарта OPC, применением интерфейса ODBC или OLE DB. Открытость системы программирования достигается поддержкой языков МЭК 61131-3.

Особенно интересно с точки зрения открытости применение веб-интерфейса, поскольку он обеспечивает доступ к SCADA с любого компьютера из любой точки мира, независимо от аппаратной платформы, типа канала связи, операционной системы и используемого веб-навигатора.

Экономическая эффективность

Экономическую эффективность SCADA можно определить как отношение экономического эффекта от ее внедрения к общей сумме затрат на внедрение и поддержание системы в работоспособном состоянии. На экономическую эффективность в конечном счете влияют практически все свойства SCADA, однако в первую очередь можно выделить следующие:

- масштабируемость (возможность применения как для больших, так и для малых систем);
- модульность. Модульность позволяет сделать заказную комплектацию системы в зависимости от поставленной задачи. Типовыми модулями могут быть, например, модуль ввода-вывода, модуль визуализации, модуль алармов, модуль трендов, модуль отчетов, модуль коммерческого учета энергоресурсов и др.;
- стоимость обслуживания;
- условия обновления версий;
- надежность поставщика, наличие опыта практического применения;
- стоимость обучения;
- стоимость технической поддержки;
- методы ценообразования.

Общим недостатком универсальных SCADA является их низкая экономическая эффективность при использовании для решения простых задач. Несмотря на то, что цена SCADA-пакетов существенно снижается при уменьшении количества доступных пользователю тегов и набора модулей, остается высокой цена технической поддержки. Также дорогой (трудоемкой) остается адаптация универсальной SCADA к конкретной задаче. Поэтому ряд фирм предлагают более узкоспециализированные, но достаточно простые в настройке микро-SCADA с сокращенной функциональностью, например, пакет RLDataView.

Программное обеспечение

Ниже мы рассмотрим отличительные особенности двух известных пакетов: MasterSCADA и Trace Mode.

MasterSCADA

Система MasterSCADA предназначена для создания полномасштабных систем автоматизации в различных отраслях промышленности. Основной ее особенностью является объектный подход, использованный на уровне описания системы при ее настройке на конкретный объект автоматизации. Например, цех, участок, технологический блок и физическое устройство при

создании проекта с помощью MasterSCADA рассматриваются как отдельные объекты. Для каждого объекта создается свое описание на технологическом языке программирования. Описание включает в себя свойства объекта и документы объекта. Свойствами могут быть период опроса, способ линеаризации датчика, диапазон входных сигналов. Документами объекта являются его изображение, мнемосхема, график изменения переменных и т. п. Любой документ в системе относится к некоторому объекту. Такой подход позволяет легко размножать один раз созданные объекты, что повышает скорость настройки SCADA на задачу пользователя.

К признакам объектного подхода относится также возможность наследования всех настроек от "родительских" объектов. Это означает, что в MasterSCADA нет необходимости вводить настройки для каждого типа объектов "с нуля". Можно использовать наследование этих настроек от родительского объекта, изменив в них только те параметры, которые отличают родителя от потомка.

Созданные объекты можно копировать с целью многократного использования. При копировании объекта сохраняются все связанные с ним документы и свойства. Связи с внешними источниками и приемниками данных восстанавливаются после копирования, если в системе имеются такие источники или свободные приемники данных (физические устройства). Это позволяет пополнять библиотеку объектов вновь созданными экземплярами и использовать объекты, созданные другими разработчиками.

Trace Mode

SCADA-система Trace Mode 6 фирмы AdAstrA состоит из инструментальной системы и набора исполнительных модулей. В состав Trace Mode 6 входят также средства управления бизнес-процессами производственного предприятия.

Для увеличения скорости разработки проекта пользователя применяется оригинальная технология автопостроения. Автоматически в SCADA могут быть построены:

- ✓ источники данных ПЛК и модулей ввода-вывода по известной конфигурации;
- ✓ каналы по источникам данных;
- ✓ связи каналов из редактора аргументов;
- ✓ связи контроллер-сервер и сервер-сервер;
- ✓ SQL-запросы;
- ✓ связи с OPC-сервером;
- ✓ связь с ODBC.

Автопостроение позволяет снизить количество ошибок, допускаемых пользователем при ручном создании проекта.

В пятой версии Trace Mode инструментальная система представлена в виде отдельных компонентов, в 6-ой использована интегрированная среда разработки.

В систему Trace M ode 6 включены пять языков программирования – Techno SFC, Techno LD, Techno FBD, Techno ST, и Techno IL, которые являются расширениями соответствующих языков стандарта МЭК 61131-3.

Заключение к главе "Программное обеспечение"

Основными тенденциями развития программного обеспечения для средств автоматизации являются максимальное упрощение процесса программирования и обеспечение открытости инструментальных средств. Конечной целью является предоставление потребителю возможности построения качественной системы автоматизации в максимально короткий срок.

Долгий период неопределенности в средствах программирования ПЛК и SCADA пакетов завершился принятием общепризнанного стандарта МЭК 61131-3 и созданием на его основе инструментальных средств программирования, которые поддерживаются фирмами, специализирующимися на программном обеспечении.

Существенный вклад в открытость систем автоматизации внес стандарт OPC, обеспечивший системным интеграторам широчайший выбор аппаратного обеспечения, совместимого с любыми стандартными SCADA пакетами, а разработчикам контроллерного оборудования - расширение рынков сбыта.

Контрольные вопросы по теме

Уровень курса

1. Функции SCADA.
2. Разработка человеко-машинного интерфейса.
3. SCADA как система диспетчерского управления.
4. SCADA как часть системы автоматического управления.
5. Свойства SCADA.