ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ РАДИОТЕХНИКИ,
ЭЛЕКТРОНИКИ И АВТОМАТИКИ (ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)»

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ДОКЛАД

«ОПЕРАЦИОННЫЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ»

Студента факультета ВМС
группы ВВ-2-06
Краснякова Андрея Михайловича

МОСКВА 2008

ОГЛАВЛЕНИЕ

Отличия ОСРВ и ОС общего назначения

Основой любого аппаратно-программного комплекса, в том числе работающего в режиме реального времени, является операционная система (ОС). ОС - это 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.

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

Система работает в реальном времени, если ее быстродействие адекватно скорости протекания физических процессов на объектах контроля или управления.

Применение ОСРВ всегда связано с аппаратурой, с объектом, с событиями, происходящими на объекте. СРВ как аппаратно-программный комплекс, включает в себя датчики, регистрирующие события на объекте, модули ввода-вывода, преобразующие показания датчиков в цифровой вид, пригодный для обработки этих показаний на компьютере, и, наконец, компьютер с программой, реагирующей на события, происходящие на объекте. ОСРВ ориентирована на обработку внешних событий. Именно это приводит к коренным отличиям (по сравнению с ОС общего назначения) в структуре системы, в функциях ядра, в построении системы ввода-вывода.

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

Назовем системой реального времени (в дальнейшем СРВ) аппаратно-программный комплекс, реагирующий в предсказуемые времена на непредсказуемый поток внешних событий.

Это означает, что:

- Система должна отреагировать на событие, произошедшее на объекте, своевременно, то есть в течение времени, критического для этого события (meet deadline). Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается ошибкой для систем реального времени;

- Система должна успевать реагировать на одновременно происходящие события. Даже если два или большее число внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение временных интервалов, критических для этих событий.

Системы жесткого и мягкого реального времени

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

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

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

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

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

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

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

Системы исполнения и системы разработки ОСРВ

Одно из коренных внешних отличий систем реального времени от систем общего назначения - четкое разграничение систем разработки и систем исполнения.

Система исполнения ОСРВ - это набор инструментов (ядро, драйверы, исполняемые модули), обеспечивающих функционирование приложения реального времени.

Большинство современных ведущих ОСРВ поддерживают целый спектр аппаратных архитектур, на которых работают системы исполнения (Intel, Motorola, MIPS, PowerPC и другие). Это объясняется тем, что набор аппаратных средств - часть комплекса реального времени и аппаратура должна быть также адекватна решаемой задаче, поэтому ведущие ОСРВ перекрывают целый ряд наиболее популярных архитектур, чтобы удовлетворить самым разным требованиям по части аппаратуры.

Система разработки - набор средств, обеспечивающих создание и отладку приложения реального времени. Системы разработки (компиляторы, отладчики и всевозможный вспомогательный инструментарий) работают, как правило, в популярных и распространенных ОС, таких как UNIX и Windows. Кроме того, многие ОСРВ имеют и так называемые резидентные средства разработки, исполняющиеся в среде самой операционной системы реального времени, - особенно это относится к ОСРВ класса "ядра".

Функционально средства разработки ОСРВ отличаются от привычных систем разработки, так как часто они содержат средства удаленной отладки, средства профилирования (измерение времен выполнения отдельных участков кода), средства эмуляции целевого процессора, специальные средства отладки взаимодействующих задач, а иногда и средства моделирования.

Параметры ОСРВ

1. Время реакции системы

Почти все производители систем реального времени приводят такой параметр, как время реакции системы на прерывание (interrupt latency).

В самом деле, если главным для системы реального времени является ее способность вовремя отреагировать на внешние события, то такой параметр, как время реакции системы является ключевым. Поймем, какие времена мы должны знать для того, чтобы предсказать время реакции системы.

События, происходящие на объекте, регистрируются датчиками, информация с датчиков передается в модули ввода-вывода (интерфейсы) системы. Модули ввода-вывода, получив информацию от датчиков и преобразовав ее, генерируют запрос на прерывание в управляющем компьютере, подавая ему тем самым сигнал о том, что на объекте произошло соответствующее событие. Получив сигнал от модуля ввода-вывода, система должна запустить программу обработки этого события.

Интервал времени - от момента возникновения события на объекте до выполнения первой инструкции в программе обработки этого события - и является временем реакции системы на события, и, проектируя систему реального времени, разработчики должны уметь вычислять этот интервал. Из чего он складывается?

Время выполнения цепочки действий - от события на объекте до генерации прерывания - никак не зависит от ОСРВ и целиком определяется аппаратурой, а вот интервал времени от возникновения запроса на прерывание и до выполнения первой инструкции его обработчика определяется целиком свойствами операционной системы и архитектурой компьютера. Причем это время нужно уметь оценивать в худшей для системы ситуации, то есть в предположении, что процессор загружен, что в это время могут происходить другие прерывания, что система может выполнять какие-то действия, блокирующие прерывания.

2. Время переключения контекста

В операционные системы реального времени заложен параллелизм, возможность одновременной обработки нескольких событий, поэтому все ОСРВ являются многозадачными (многопроцессными, многонитиевыми). Для того чтобы уметь оценивать накладные расходы системы при обработке параллельных событий, необходимо знать время, которое система затрачивает на передачу управления от процесса к процессу (от задачи к задаче, от нити к нити), то есть время переключения контекста.

3. Размеры системы

Для систем реального времени важным параметром является размер системы исполнения, а именно суммарный размер минимально необходимого для работы приложения системного набора (ядро, системные модули, драйверы и т. д.). Хотя надо признать, что с течением времени значение этого параметра уменьшается, тем не менее, он остается важным, и производители систем реального времени стремятся к тому, чтобы размеры ядра и обслуживающих модулей системы были невелики. Например, размер ядра ОСРВ OS9 на микропроцессорах MC68xxx - 22 кбайт, VxWorks - 16 кбайт.

4. Возможность исполнения системы из ПЗУ (ROM)

Это свойство ОСРВ - одно из базовых. Оно позволяет создавать компактные встроенные СРВ повышенной надежности, с ограниченным энергопотреблением, без внешних накопителей.

Механизмы реального времени

Процесс проектирования конкретной системы реального времени начинается с тщательного изучения объекта. Разработчики проекта исследуют объект, изучают возможные события на нем, определяют критические сроки реакции системы на каждое событие и разрабатывают алгоритмы обработки этих событий. Затем следует процесс проектирования и разработки программных приложений.

Идеалом является ОСРВ, в которой приложения реального времени разрабатываются на языке событий объекта. Называется она "система, управляемая критическими сроками". Разработка приложений реального времени в этой системе сводится к описанию возможных событий на объекте. В каждом описателе события указывается два параметра: временной интервал - критическое время обслуживания данного события и адрес подпрограммы его обработки. Всю дальнейшую заботу о том, чтобы подпрограмма обработки события стартовала до истечения критического интервала времени, берет на себя операционная система.

Но это - абстракция. В реальности же разработчик должен перевести язык событий объекта в сценарий многозадачной работы приложений ОСРВ, стараясь оптимально использовать предоставленные ему специальные механизмы и оценить времена реакций системы на внешние события при этом сценарии.

Система приоритетов и алгоритмы диспетчеризации

Базовыми инструментами разработки сценария работы системы являются система приоритетов процессов (задач) и алгоритмы планирования (диспетчеризации) ОСРВ.

В многозадачных ОС общего назначения используются, как правило, различные модификации алгоритма круговой диспетчеризации (Round Robin), основанные на понятии непрерывного кванта времени (time slice), предоставляемого процессу для работы. Алгоритмы круговой диспетчеризации в чистом виде в ОСРВ неприменимы. Основной недостаток - непрерывный квант времени, в течение которого процессором владеет только один процесс.

Для ОСРВ выдвигаются следующие требования:

1. ОС должна быть *многонитевой* по принципу абсолютного приоритета (прерываемой). Планировщик должен иметь возможность прервать любую нить и предоставить ресурс той нити, которой он более необходим. ОС (и аппаратура) должны также обеспечивать прерывания на уровне обработки прерываний.

2. Должно существовать понятие *приоритета нити*. Проблема в том, чтобы определить, какой задаче требуется ресурс. В идеальной ситуации ОСРВ отдает ресурс нити или драйверу с ближайшим крайним сроком. Чтобы реализовать это, ОС должна знать время, требуемое каждой из выполняющихся нитей для завершения, однако это крайне сложно для реализации, поэтому разработчики ОС принимают иную точку зрения: вводится понятие уровня приоритета задачи, и временные ограничения сводят к приоритетам. На сегодняшний день не имеется иного решения, поэтому понятие приоритета нити необходимо.

3. Должна существовать *система наследования приоритетов*.

Именно этот механизм синхронизации и тот факт, что различные нити используют одно и то же пространство памяти, отличают нити от процессов. Процессы не разделяют одно и то же пространство памяти.

Комбинация приоритета нити и разделение ресурсов между ними приводит к классической проблеме инверсии приоритетов. Это можно проиллюстрировать на примере, где есть, как минимум, три нити. Когда нить низшего приоритета заняла ресурс, разделяемый с нитью высшего приоритета, а сначала выполняется нить среднего приоритета, выполнение нити высшего приоритета будет приостановлено, пока не освободится ресурс и не отработает нить среднего приоритета. В этой ситуации время, необходимое для завершения нити 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 вызывающей нити. Наследование означает, что блокирующая ресурс нить наследует приоритет блокируемой нити (справедливо лишь в том случае, если блокируемая нить 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 нити вручную прежде, чем ресурс окажется заблокированным – это возможно в случае, когда две нити разных приоритетов претендуют на один ресурс. В общем случае решения не существует.

Механизмы межзадачного взаимодействия

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

Средства для работы с таймерами

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

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

Здесь описаны только базовые, обязательные механизмы, использующиеся в ОСРВ. Кроме того, почти в каждой ОСРВ вы найдете целый набор дополнительных, специфических только для нее механизмов, касающийся системы ввода-вывода, управления прерываниями, работы с памятью. Каждая система содержит также ряд средств, обеспечивающих ее надежность: встроенные механизмы контроля целостности кодов, инструменты для работы со сторожевыми таймерами.

Классы систем реального времени

Количество операционных систем реального времени, несмотря на их специфику, очень велико (на данный момент ~100). Наверное, этих систем еще больше, если иметь в виду некоммерческие ОСРВ. Однако сама специфика применения ОСРВ требует гарантий надежности, в том числе и юридических - этим, видимо, можно объяснить тот факт, что среди некоммерческих систем реального времени нет сколько-нибудь популярных.

Среди коммерческих систем реального времени можно выделить группу ведущих систем - по объемам продаж и по популярности. Это системы: VxWorks, OS9, pSOS, LynxOS, QNX, VRTX.

Исполнительные системы реального времени

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

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

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

Кроме того, системы разработки для продуктов этого класса традиционно дороги (порядка $20000). Хотя, надо отметить, что качество и функциональность систем разработки в этом классе традиционно хороши, так как они были изначально кроссовыми.

Наиболее ярким представителем систем этого класса является операционная система VxWorks.

Область применения - компактные системы реального времени с хорошими временами реакций.

Ядра реального времени

В этот класс входят системы с монолитным и микроядерным ядром, где и содержится реализация всех механизмов реального времени этих операционных систем. Исторически системы этого типа были хорошо спроектированы.

Системы этого класса, как правило, модульные, хорошо структурированные, имеют наиболее развитый набор специфических механизмов реального времени, компактные и предсказуемые. Наиболее популярные системы этого класса: OS9, QNX.

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

UNIX'ы реального времени

Исторически системы реального времени создавались в эпоху расцвета и бума UNIX а и поэтому многие из них содержат те или иные заимствования из этой концепции операционный системы (пользовательский интерфейс, концепция процессов и т.д.).

Часть разработчиков ОСРВ попыталась просто переписать ядро UNIX, сохранив при этом интерфейс пользовательских процессов с системой, насколько это было возможно. Реализация этой идеи не была слишком сложной, поскольку не было препятствия в доступе к исходным текстам ядра, а результат оказался замечательным. Получили и реальное время, и сразу весь набор пользовательских приложений - компиляторы, пакеты, различные инструментальные системы.

Однако Unix'ы реального времени не избавлены от следующих недостатков: системы реального времени получаются достаточно большими и реактивность их ниже, чем реактивность систем первых двух классов.

Наиболее популярным представителем систем этого класса является операционная система реального времени Lynx OS.

ОСРВ QNX

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

ОС QNX представляет собой гибрид 16/32-битовой операционной системы, которую пользователь может конфигурировать по своему усмотрению. Наиболее часто 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, означающую 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 Windows NT);

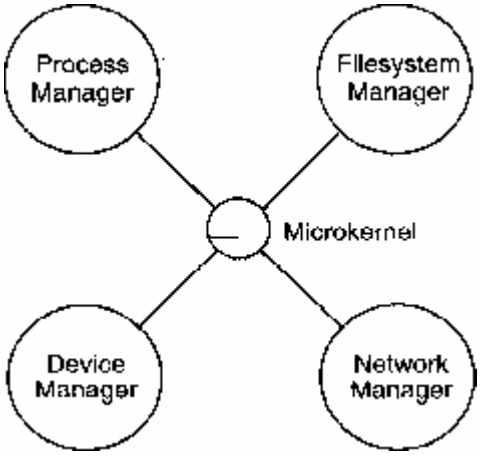
- масштабируемость и эффективность, достигаемые оптимальным использованием ресурсов и означающие ее применимость для встроенных (embedded) систем. В каталоге dev присутствуют только необходимые для поставленных задач файлы, соответствующие нужным драйверам. Драйверы и менеджеры можно запускать и удалять (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 выражается в следующем. 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.

Архитектура микроядра

ОС QNX состоит из небольшого ядра, представляющего собой группу взаимодействующих процессов. Причем, как видно из иллюстрации, структура ядра не иерархична, а, скорее, похожа на команду из нескольких игроков одного уровня, взаимодействующих между собой и их центром.



Микроядро ОС QNX - очень мало в размерах, оно содержит только две основные функции:

- Управление прохождением сообщений - микроядро управляет прохождением всех сообщений между процессами;
- Диспетчеризация процессов - диспетчер - это часть самого микроядра, и он запускается всякий раз, когда какой-либо процесс изменил свое состояние в результате получения сообщения или в результате выполнения аппаратного прерывания.

При обычных условиях ядро недоступно - оно защищено несколькими слоями защиты. Оно становится доступно только по прямому вызову из какого-либо системного процесса или аппаратного прерывания.

Системные процессы практически ничем не отличаются от процессов пользователя, но они недоступны для прямого вызова из пользовательских процессов или прикладных программ.

При различных вариантах запуска исполняемой программы в ОС QNX, она может быть представлена как системный процесс или как внешний процесс пользователя.

Драйверы устройств.

Драйверы устройств в QNX - это один из видов системных процессов, или, точнее было бы сказать, служб, которые избавляют операционную систему от необходимости работать со всеми теми особенностями, которые необходимы для устойчивой работы специфического оборудования.

Запуск драйвера в QNX сродни обычному запуску исполняемой программы, т.е. первый запуск драйвера ничем не отличается от запуска любой другой программы. Но после первого запуска драйвер присоединяется к системе, т.е. ассоциируется с наиболее близким системным процессом в виде его расширения или дополнения и больше не требует никакой настройки. При добавлении нового драйвера в системе не может произойти никаких побочных эффектов в виде, например, конфликта с другими устройствами.

Однажды подключившись к системе, драйвер может принимать следующий вид:

- Исчезнуть, как процесс, но сохранить все расширения, которые им были проведены на системном процессе, с которым он был ассоциирован;
- Сохранить свою индивидуальность как процесса.

Межпроцессорный обмен информацией

QNX была первой коммерческой операционной системой, в основе которой лежал обмен сообщениями как единственный способ информационного обмена между взаимодействующими процессами. ОС QNX обязана своей мощью, простотой и элегантностью полному внедрению этого метода в работу самой ОС.

Сообщения в QNX представляют собой пакет байт, посланный от одного процесса к другому. В QNX придается большое значение содержанию сообщения, оно имеет смысл только для процесса-источника и процесса получателя, и ни для какого бы то ни было другого процесса.

Обмен сообщениями обеспечивает не только передачу информации, но является способом синхронизации параллельной работы нескольких процессов. При передаче, приеме и ответе на сообщение процессы подвергаются разнообразным изменениям состояния, которые влияют на то, когда и как долго данный процесс может работать. Имея информацию о приоритетах и состояниях процессов, микроядро может управлять работой процессов как можно эффективней с точки зрения распределения ресурсов системы. Этот единственный согласующий метод - обмен сообщениями - действует внутри всей системы.

QNX в сети

В своем первоначальном виде локальная сеть предоставляет механизм для обмена файлами и совместное использование периферийных устройств среди нескольких соединенных компьютеров. QNX же продвинулась гораздо дальше этой примитивной концепции и объединила всю сеть в единичном однородном наборе ресурсов.

Любой процесс на любом компьютере в сети может быть использован с другого компьютера. С точки зрения исполняемых в ОС QNX программ нет различий между локальными и удаленными ресурсами, т.е. нет необходимости встраивать в программу дополнительные средства для доступа к удаленным ресурсам. Фактически программа

должна иметь только средства для определения того, где находится ресурс типа файла или устройства, - на локальном компьютере или каком-либо другом узле сети.

Пользователи могут получать доступ к файлам в любой точке в сети, использовать любое периферийное устройство и запускать любые программы на компьютерах, находящихся в сети, если, конечно, пользователь имеет соответствующие права доступа. QNX предоставляет широкие возможности по созданию приложений, состоящих из нескольких процессов и выполняющихся на нескольких распределенных компьютерах.

Сети QNX имеют возможность одновременной поддержки всех типов приложений, как реального времени, так внутреннего системного времени компьютера, причем одновременно. В сети QNX могут одновременно соединяться компьютеры с различной аппаратной сетевой поддержкой и различными стандартами протоколов.

Примеры применения QNX за рубежом

- Наиболее ярким примером применения QNX является работа с кредитными карточками VISA во всех региональных офисах Северной Америки.
- Управление дорожным движением. В городе Оттава-Карлетон (Канада) на базе QNX разработана система управления движением городского транспорта муниципалитета города. Эта система объединяет около 700 светофоров и 3000 придорожных датчиков на протяжении 1100 километров шоссе. Пропускная способность этих шоссе — 5,4 миллиарда автомобилей в год. Кроме времени и продолжительности переключения сигналов светофоров на каждом перекрёстке города данная система управления должна фиксировать происходящие события, анализировать работоспособность оборудования через придорожные датчики.
- Управление ядерным реактором. Одно из отделений канадской компании Atomic Energy of Canada Ltd., которая известна как разработчик, производитель и продавец ядерных реакторов, специализируется на разработке программных продуктов по управлению и мониторингу. На основе QNX этим отделением разработана система управления ядерным реактором, которая называется «Распределённая Система Управления с Открытой Архитектурой».
- Cisco Systems использует оптимизированную версию микроядра QNX Neutrino в программном обеспечении IOS XR. Программный пакет IOS XR предназначен для управления коммутаторами Cisco CRS-1, обеспечивает непрерывный режим работы и поддерживает развитые функции управления терабитными коммутаторами с распределённой архитектурой.

Примеры применения QNX в России

- Наиболее известным применением QNX в России является система автоматизированного контроля и управления разводкой Троицкого моста через Неву в Санкт-Петербурге, реализованная ЗАО НПП «Промтрансавтоматика». Эта система эксплуатируется с апреля 2002 года. После реконструкции мост ни разу не выбился из графика разводки.
- Система управления северными магистральными нефтепроводами, расположенная в г. Ухта. Система включает в себя шесть операторских мест с горячим резервированием, которые выполняют управление одновременно по четырём направлениям магистрального нефтепровода на участке Ярославль-Ухта (протяжённость 1500 км)
- Системы управления движением судов в портах (Санкт-Петербург, Мурманск, Владивосток, Мариуполь) и по Керченскому проливу (порт Кавказ).

Литература

1. *А.А. Зарубин* “Микропроцессорное программное управление. Архитектура IXA.”

2. *А.А. Жданов* “Современный взгляд на ОС реального времени”, http://www.asutp.ru/

3. http://ru.wikipedia.org/wiki/QNX