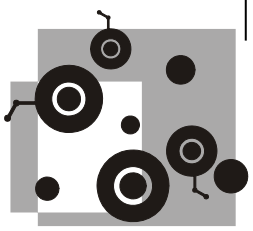
ЗАНЯТИЕ 6



Платформа QNX

Это занятие предназначено для тех, кто не знаком с платформой QNX6 — если такое возможно ☺. На нем будут рассмотрены следующие вопросы:

- что такое платформы QNX4 и QNX6, чем они отличаются;
- что представляет собой инструментальный комплект QNX Momentics;
- что представляет собой операционная система реального времени QNX Neutrino.

6.1. QNX4 vs. QNX6

Что же мы имеем в виду, произнося слово "QNX"? Понятие "QNX" неоднократно трансформировалось в течение почти 30 лет своего существования. Обычно под термином QNX понимают семейство операционных систем жесткого реального времени. Однако правильнее рассматривать QNX как собирательное название программных платформ для построения компьютерных систем реального времени. В настоящее время для коммерческого использования доступны программные платформы QNX двух поколений: QNX4 и QNX6.

6.1.1. Платформа QNX4

QNX4 — технологии, появившиеся в начале 90-х годов прошлого века и с некоторыми усовершенствованиями дошедшие до сего

дня. Основу платформы QNX4 составляет ОСРВ QNX 4.25, графическая оболочка Photon microGUI 1.14, Socket 4.25/5.0 (средства поддержки стека протоколов TCP/IPv4) и система программирования Watcom C/C++ 10.6.

Важный момент: QNX4 работает исключительно на архитектурной платформе *x*86 начиная с i386. ОСРВ QNX 4.25 — первая ОСРВ, прошедшая сертификацию POSIX в 1993 году — упорное (хотелось написать — фанатичное) следование различным международным и промышленным стандартам всегда было отличительным свойством компании QNX Software Systems.

Сопровождение ОСРВ QNX 4.25 продолжается в основном в направлении поддержки новых аппаратных устройств, хотя и не ограничивается этим. Последняя редакция дистрибутива выпущена в ноябре 2009 года. Среди новинок последних двух-трех лет — поддержка USB и приводов SATA. Прекращение сопровождения QNX 4.25 даже не обсуждается — это обусловлено длительными сроками эксплуатации компьютерных систем реального времени, построенных на основе QNX4.

В 2003 году на базе ОСРВ QNX 4.25 была создана защищенная операционная система реального времени (ЗОСРВ) "QNX" (десятичный номер КПДА.00002-01), которая в 2004 году успешно прошла сертификацию в *ФСТЭК* (Федеральная служба по техническому и экспортному контролю — **http://www.fstec.ru/**) России по 3-му классу защиты от несанкционированного доступа (НСД) и 2-му уровню контроля отсутствия недекларированных возможностей (НДВ) — это позволяет предприятиям, разработавшим автоматизированную систему (АС, на базе QNX 4.25), с минимальными усилиями аттестовать свои изделия до класса 1Б включительно (об информационной безопасности см. материалы *занятия 3*).

Конечно, несмотря на свою удивительную эффективность и доказанную надежность, продукты QNX4 родом из 1990-х годов — выпускникам вузов третьего тысячелетия, привыкшим к интегрированным средам разработки, не комфортно работать в текстовом редакторе vedit и использовать командную строку для сборки проектов. Однако если все-таки им придется сопровождать созданные лет десять тому назад (и, надо сказать, прекрасно рабо-

тающие) системы на базе QNX4, то они могут использовать кроссплатформенный подход к разработке (т. е. разработку в среде Windows или Linux) в интегрированной среде системы программирования *OpenWatcom* (**www.openwatcom.org**) — на момент написания этой книги актуальной версией была 1.8. Одно предостережение: использовать язык C++ при разработке с помощью OpenWatcom, увы, не удастся. Однако о каких-либо проблемах с "чистым" языком C мне не известно.

Как бы то ни было, QNX4 продолжает занимать существенную нишу на рынке промышленной автоматизации. Мало того, платформу QNX4 используют даже в новых проектах. Ничего удивительного: судьба этой технологии прекрасно иллюстрирует важность концепции Proven-in-Use *(см. занятие 3)*.

Для использования QNX4 также есть причины, которые называют программисты, применявшие DOS. Поясню — в России (думаю, что не только в ней) DOS по факту является самой широко используемой программной платформой на рынке промышленной автоматизации. Так вот, во-первых, инструменты QNX4 гораздо комфортнее и мощнее инструментов DOS. Не говоря уже о полной защите памяти, многозадачности и встроенной поддержке распределенных вычислений. Во-вторых, в QNX4 (кстати, в QNX6 тоже) для доступа к регистрам используются хорошо известные функции `in`/`out`. И в-третьих, многие графические приложения для DOS разрабатывались с использованием программного интерфейса VESA BIOS Extensions (VBE), поддерживаемого в QNX. Все эти факторы делают перенос ПО систем реального времени из DOS в QNX4 достаточно комфортным занятием.

6.1.2. Платформа QNX6

Платформа QNX6 включает ряд технологий, основными среди которых являются:

- операционная система реального времени *QNX Neutrino*. Ее часто для краткости называют просто QNX6 — так что всегда обращайте внимание, в каком контексте используется термин "QNX6";

- комплект инструментальных средств *QNX Momentics*;
- пакеты программного обеспечения так называемого "промежуточного слоя" (англ. *middleware*) *QNX Aviage*.

Программное обеспечение QNX6 обычно делится на две категории: средства разработки и средства исполнения. Вспомним эти два понятия.

Инструментальные средства — это программное обеспечение, предназначенное для создания и/или конфигурирования средств исполнения. Средства разработки не требуются во время эксплуатации целевых систем.

Примечание

Не требуются по идее, конечно. Безусловно, можно создать такую целевую систему, которая будет использовать и компилятор, и средства отладки, но это будет использованием средств разработки, так сказать, не совсем по назначению.

Средства исполнения — это программные компоненты, используемые в ЭВМ, предназначенной для эксплуатации конечными потребителями, т. е. программное обеспечение целевых систем. По-английски этап эксплуатации называют *runtime* — отсюда средства исполнения (англ. *runtime modules*, дословно *компоненты для этапа эксплуатации*) часто называют жаргонизмом *рантаймы*.

QNX Momentics и QNX Neutrino поставляются в виде продукта, который называется QNX Software Development Platform (SDP). Текущая версия на момент написания книги — 6.4.2.

Существует три равнозначных (ну ладно, *почти* равнозначных) дистрибутива QNX SDP — Windows-host (для кроссплатформенной разработки приложений QNX Neutrino в среде Windows), Linux-host (для кроссплатформенной разработки приложений QNX Neutrino в среде Linux) и Neutrino-host (для кроссплатформенной разработки приложений QNX Neutrino в среде самой QNX Neutrino, т. е. для *резидентной* разработки).

Выбор комплекта зависит исключительно от ваших личных предпочтений. Я предпочитаю Windows-host, но, если честно, в среде известных мне пользователей QNX я в меньшинстве.

Одно меня утешает: говорят, что в Северной Америке пользователей дистрибутива Windows-host большинство — уж очень много разработчиков перешли на QNX6 с операционных систем реального времени, не имеющих резидентных средств разработки, поэтому привыкших к кроссплатформенному подходу. Что касается Neutrino-host, то этот дистрибутив популярен среди программистов, работавших ранее с QNX4 — они привыкли к резидентной разработке.

Впрочем, если вести разработку для целевых аппаратных платформ, отличных от Intel, то разработка даже с использованием Neutrino-host, естественно, будет кроссплатформенной.

Кроме QNX Momentics в состав каждого из дистрибутивов QNX SDP так же входят одинаковые (вот тут уже честно одинаковые) средства исполнения ОСРВ QNX Neutrino, из которых можно "лепить" собственные конфигурации этой операционной системы для всех поддерживаемых аппаратных архитектур. Дистрибутив Neutrino-host дополнительно содержит, разумеется, инсталлятор ОСРВ QNX Neutrino для платформы Intel *x*86.

Имеются также дистрибутивы QNX SDP Lite. В них включены средства исполнения QNX Neutrino только для платформы Intel *x*86. Это облегчает скачивание дистрибутива из Интернета для ознакомительных целей.

6.2. QNX Momentics

Комплект инструментальных средств QNX Momentics включает:

- инструменты разработки и отладки приложений на C/C++ с генерацией кода для всех аппаратных платформ, на которых работает ОСРВ QNX Neutrino;
- визуальный построитель графических интерфейсов Photon Application Builder (PhAB);
- инструменты для анализа и оптимизации программ — системный профилировщик, прикладной профилировщик, анализатор покрытия кода тестами;

- инструменты для построения целевых систем — формирования загружаемых образов, образов встраиваемой файловой системы, создания файловых систем Power Safe и QNX4, конвертирования образов в форматы программаторов ППЗУ и т. д.;
- инструменты для мониторинга целевых систем QNX Neutrino (выполняющихся процессов, каналов и соединений, состояния потоков, сигнальных масок, файловых дескрипторов и т. д.), анализа использования ОЗУ процессами;
- программы удаленного доступа к рабочему столу Photon целевой системы QNX Neutrino из среды Windows и UNIX — Phindows и Phinx.

В состав QNX Momentics входит большой комплект весьма качественной и обстоятельной документации в формате HTML на английском языке. Эта документация относится как к QNX Momentics, так и к QNX Neutrino.

В Windows-host и Linux-host документация доступна через элемент меню **Help** интегрированной среды разработки, в Neutrino-host она доступна через запускаемую из панели инструментов или с помощью меню **Launch** программу Helpviewer. Ключевые документы переведены на русский язык и доступны в книжных магазинах. Так что у вас есть, что почитать на досуге, уважаемый читатель ☺.

Но если вас интересует мое мнение — я бы начал с "Jumpstart Guide". Русский перевод этого руководства доступен на сайте **www.swd.ru**, а видеопрезентацию авторов инструкции на английском и немецком языках можно посмотреть на сайте **www.youtube.com**. Затем я прочел бы концептуальный документ "QNX Neutrino Realtime Operating System. System Architecture" (он есть в штатной электронной документации (Help), а также издан на русском языке). Дальнейшее чтение зависит от вашей специализации.

6.3. QNX Neutrino

Операционная система жесткого реального времени QNX Neutrino разрабатывается сообществом Foundry27 (**www.foundry27.com**)

в рамках проектов "QNX Operating System", "Networking", "Filesystems", "Graphics", "Multimedia", "BSP and Drivers" и др.

QNX Neutrino очень хорошо документирована. Базовые книги, необходимые для ее использования ("ОСРВ QNX Neutrino. Системная архитектура", " ОСРВ QNX Neutrino. Руководство пользователя" и учебник Роберта Кртена "Введение в QNX Neutrino: Руководство для разработчиков приложений реального времени"), переведены на русский язык.

Нет смысла пересказывать документацию, поэтому я буду говорить не столько о функциональных возможностях этой знаменитой ОСРВ, сколько о том, *зачем* они нужны.

6.3.1. Поддержка аппаратуры

Поддержка процессорных архитектур

Рынок встраиваемых систем реального времени на самом деле не является каким-то единым понятием. Он состоит из совершенно разных отраслевых рынков — рынка автомобильных систем, рынка авиационных систем, рынка систем телекоммуникации и т. д. В каждой из отраслей свои требования, устои, а также свои предпочтения по отношению к аппаратуре. Причем речь идет не только о процессорных архитектурах, но и о коммуникационных шинах, видеоадаптерах и т. д.

QNX Neutrino используется в разных отраслях, поэтому поддерживает несколько процессорных архитектур:

- *Intel x86*. Еще лет пять назад российский рынок систем реального времени был, по сути дела, рынком CISC-архитектуры *x*86. Однако в последние годы существенно возрос интерес к другим платформам, например, ARM, MIPS и PowerPC. Но это вовсе не значит, что платформа *x*86 стала менее популярной — просто российские предприятия приступили к решению задач, специально для которых создавались другие платформы. Поддержка процессорных плат на базе *x*86, в частности на базе процессора Intel Atom, остается важным направлением для QNX Neutrino. Кстати, корпорация Intel предлагает оптимизированный компилятор Intel C++ для QNX Neutrino (**http://software.intel.com/en-us/intel-compilers/**).

- *MIPS*. Эта RISC-архитектура разрабатывается компанией MIPS Technologies и лицензируется несколькими производителями процессоров. Основными достоинствами MIPS-архитектуры являются низкое энергопотребление и тепловыделение процессоров, а также возможность размещения на одном кристалле большого количества процессорных ядер (я слышал о 32-ядерных процессорах MIPS). В настоящее время наибольшее распространение процессоры MIPS получили в телекоммуникации и мультимедийных потребительских устройствах, хотя существуют рабочие станции и даже суперкомпьютеры с процессорами MIPS.
- *ARM*. Эта относительно простая RISC-архитектура разрабатывается компанией Advanced RISC Machine и лицензируется более чем двумя десятками производителей процессоров. По факту — это самая распространенная в мире процессорная архитектура. Дело в том, что процессоры ARM ввиду их малых размеров и низкого энергопотребления используются в массовых портативных устройствах, например, в мобильных телефонах. Процессоры с архитектурой ARM выпускала также корпорация Intel, но в 2007 году продала этот бизнес компании Marvell и затем создала собственный CISC-процессор Atom, совместимый с *x*86 по системе команд.

Примечание

В 2007 году 98% мобильных телефонов использовали процессор ARM. Считается, что в 2009 году процессоры ARM занимали 90% рынка встраиваемых процессоров.

- SuperH 4-го поколения (*SH-4*). RISC-архитектура, разработанная корпорацией Hitachi специально для автомобильных систем, которым требовался высокопроизводительный процессор с низким энергопотреблением и тепловыделением. С 2004 года технология SH-4 была передана компании Renesas Technology — совместному предприятию Hitachi и Mitsubishi Electric по производству чипов. Среди направлений деятельности Renesas Technology — поставка комплексных аппаратных решений для автомобильных компьютерных систем. В конце 2009 года корпорации Hitachi, Mitsubishi и NEC заявили о соз-

дании на базе компаний NEC Electronics (подразделения корпорации NEC — производства электронных компонентов) и Renesas Technology единой компании по производству чипов с названием Renesas Electronics.

- *PowerPC*. Популярная RICS-архитектура, предлагаемая корпорациями IBM и Freescale. Процессоры PowerPC широко распространены на рынках автомобильной электроники и телекоммуникаций.

Поддержка процессорной архитектуры операционной системой означает, что:

- ядро и остальные программные компоненты, предназначенные для данной процессорной архитектуры, могут выполняться на процессорах с совместимой системой команд;
- в состав поставки системы программирования входит кодогенератор компилятора для данной аппаратной архитектуры, позволяющий транслировать исходный код, написанный на соответствующем языке программирования, в объектный код, способный исполняться на данных процессорах.

При этом поддержка процессорной архитектуры операционной системой не означает, что вы можете запустить операционную систему на какой-то конкретной процессорной плате с поддерживаемым процессором. Для решения этой задачи необходим соответствующий пакет *BSP* (Board Support Package — пакет поддержки процессорной платы).

Пользователи, имевшие дело только с ПК на базе процессоров Intel, обычно не сразу понимают суть проблемы — ведь им вроде как не нужен никакой BSP. Нет, на самом деле нужен. Дело в том, что на процессорных платах ПК используется весьма стандартизованная (можно сказать, ожидаемая) аппаратура, поэтому в большинстве случаев достаточно стандартизованного BSP. Впрочем, мы об этом еще поговорим.

Поддержка многоядерных процессоров

В линейках продуктов многих производителей процессоров центральное место уже заняли многоядерные процессоры. Основными достоинствами многоядерных процессоров принято считать

значительно бóльшую, по сравнению с одноядерными кристаллами, вычислительную мощность благодаря параллельной обработке, высокую компактность вычислительной системы, а также возможность работы на меньших тактовых частотах, что снижает тепловыделение и энергопотребление системы.

ОСРВ QNX Neutrino обеспечивает три способа поддержки многоядерных процессоров:

- *асимметричная многопроцессорность* (*Asymmetric Multiprocessing*, AMP) — на каждом процессорном ядре работает отдельная ОС или отдельные копии одной и той же ОС;
- *симметричная многопроцессорность* (*Symmetric Multiprocessing*, SMP) — все процессорные ядра одновременно управляются единственной копией ОС, а приложения могут перемещаться между ядрами;
- *"исключительная" многопроцессорность* или многопроцессорность "с привязкой" (*Bound Multiprocessing*, BMP) — все процессорные ядра одновременно управляются единственной копией ОС, но приложение или его потоки могут быть привязаны к конкретному ядру.

Подробности вы найдете в штатной документации QNX Neutrino, но некоторые замечания, пожалуй, не будут лишними.

Наиболее трудоемким в использовании является AMP. Вообще, поддержка этого режима была реализована в QNX Neutrino в последнюю очередь и явно для решения частной задачи одного из крупных заказчиков. Дело в том, что AMP обычно использовалась тогда, когда какая-либо ОСРВ имела необходимые характеристики для работы с объектом управления, но не имела приемлемых функциональных возможностей для построения графических интерфейсов пользователя и/или для организации сетевого взаимодействия. Тогда для сетевых задач и задач отображения использовали ОС общего назначения, работающую на отдельном процессоре (процессорном ядре) на одной процессорной плате.

Почему я говорю в прошедшем времени? Потому что AMP, хотя и используется в некоторых системах и, вероятно, будет использоваться в силу наличия на некоторых предприятиях обученных

специалистов и проверенных хорошо работающих методик, но, тем не менее, вряд ли может быть названа перспективным подходом. Обеспечение совместного использования аппаратных ресурсов не всегда просто для нескольких процессов в одной ОС, что же говорить об AMP, когда ресурсы нужно разделять между ОС, а затем внутри ОС между процессами и потоками. Разумеется, ресурсы необходимо распределить на этапе разработки системы — этого нельзя в полной мере сделать на этапе исполнения.

Разумеется, режим SMP намного проще в использовании — ни системному инженеру, ни программисту не надо думать о выделении ресурсов — это сделает ядро самостоятельно на этапе исполнения, т. е. динамически. Ну ладно, сильно не ругайтесь, иногда об этом надо думать на этапе разработки — особенно когда не выделение ресурсов программе из-за их нехватки может привести к невыполнению компьютерной системой какой-либо из своих функций (помните, как стандарт IEC 61508-6 называет невыполнение системой функции? Отказ (Failure)). Но думать надо будет гораздо менее напряженно, чем в случае AMP ☺.

У SMP есть один важный "недостаток". Дело в том, что еще несколько лет назад на процессорных платах компьютерных систем реального времени обычно стоял один процессор. При этом операционные системы могли использоваться многозадачные. Но разработчики-то знали, что процессор один и что задачи выполняются не параллельно, а квазипараллельно, т. е. на единственном процессоре с разделением времени. Поэтому в ряде случаев для упрощения кода механизмы синхронизации при разделении задачами ресурсов не использовались. Синхронизация достигалась, например, запуском работающих с ресурсом (файлом, устройством, областью памяти) задач в порядке работы с этим ресурсом и установкой для каждой из них дисциплины диспетчеризации FIFO и одинакового значения приоритета — это гарантировало, что работа с ресурсом будет выполняться в нужной последовательности. Но если такое приложение перенести в SMP-систему, то оно будет работать некорректно!

Для борьбы с этим "недостатком" SMP-систем и предназначен режим BMP. По сути дела это тот же SMP, но имеется возмож-

ность "привязать" те или иные потоки тех или иных процессов к тем или иным процессорам (процессорным ядрам). Для каждого "привязанного" потока остальные процессоры будут как бы не существовать. Таким образом, можно сделать так, что некоторые приложения будут выполняться квазипараллельно на одном процессорном ядре, в то время как остальные процессы ядро QNX Neutrino будет динамически распределять по всем доступным процессорным ресурсам.

Поддержка процессорных плат

В широком смысле под BSP-пакетом поддержки процессорной платы могут пониматься все программные компоненты, необходимые для старта QNX Neutrino на данной процессорной плате и для работы со всеми периферийными устройствами как интегрированными в плату, так и подключаемыми к ней через разъемы. Однако чаще всего речь идет о базовом BSP, компоненты которого решают следующие задачи:

- первичную инициализацию аппаратуры процессорной платы (таймера, ОЗУ, контроллера прерываний) и переключение процессора в защищенный режим;
- поиск загружаемого образа QNX Neutrino в энергонезависимой памяти ЭВМ (жесткий диск, ПЗУ, бортовая или съемная flash-память, DVD-диск и т. п.) или на внешней ЭВМ, подключенной через сетевой адаптер, последовательный канал или иной поддерживаемый интерфейс с внешним миром;
- копирование образа (а также — при необходимости — декомпрессия) в ОЗУ и передача управления ядру.

Кроме того, в базовый BSP часто включают драйвер последовательного порта и так называемый *"сервер PCI"* — программу, обеспечивающую поддержку PCI BIOS. Все остальные драйверы устройств рассматриваются как опциональные и включаются в состав BSP по желанию заказчика.

Имеется возможность скачать свыше ста готовых BSP в исходных кодах с портала Foundry27. Их можно модифицировать и включать в состав коммерческих компьютерных систем реально-

го времени, т. к. они распространяются на очень демократичных условиях лицензии Apache 2. При необходимости разработку BSP можно заказать как у компании QNX Software Systems, так и у ее партнеров, в том числе российских.

6.3.2. Базовые функциональные возможности

Практически все доклады и статьи об ОСРВ QNX Neutrino начинаются с кратких сведений о ее микроядерной архитектуре. Дело в том, что эта информация очень многое объясняет в понимании сути всех остальных механизмов и возможностей. Поэтому несколько слов об этом должны сказать и мы. Так же в рамках этого вопроса мы поговорим об интересных сетевых и графических возможностях QNX Neutrino, о которых, пожалуй, вам следует знать.

Микроядерная архитектура

Итак, операционная система QNX Neutrino имеет микроядерную архитектуру. Микроядро является всего лишь коммутирующим элементом, позволяющим процессам общаться друг с другом. При этом с точки зрения микроядра программы, реализующие системные сервисы (даже те, которые в других ОС реализуются непосредственно ядром — сетевая подсистема, файловая подсистема и т. п.), являются обычными прикладными процессами. Это обстоятельство имеет определенные последствия, самым важным из которых является то, что добавление или удаление драйверов устройств, файловых систем, сетевых стеков и т. п. может осуществляться прямо во время работы QNX Neutrino. Второе важное обстоятельство заключается в том, что поддержка новых устройств не требует перекомпиляции ядра — это чрезвычайно важно для сертификации по требованиям функциональной и информационной безопасности *(см. занятие 3)*.

Микроядерная архитектура QNX Neutrino позволяет разработчику целевой системы самому решить, какие системные сервисы достаточны для решения его прикладной задачи, и создать свою собственную конфигурацию операционной системы весьма небольшого размера. По сути дела разработчик может написать

(и при необходимости сертифицировать) собственный драйвер или системный сервис и включить его в систему без помощи и ведома разработчика ОС — при этом не ухудшив характеристик жесткого реального времени ОС.

Микроядро отвечает за реализацию всех следующих механизмов поддержки жесткого реального времени ОСРВ QNX Neutrino:

- фиксированные приоритеты потоков (256 уровней) и ISR — пожалуйста, не путайте приоритеты с nice-числами, используемыми в ОС общего назначения;
- мгновенное вытеснение задачи с меньшим приоритетом;
- вытисняемые системные вызовы (!) и даже ISR (!!);
- защита от инверсии приоритетов на базе протокола наследования приоритетов (Priority Inheritance Protocol);
- отсутствие непредвиденных расходов ресурсов (например, на свопинг);
- механизм трассировки ядра, позволяющий узнать точное время каждой операции.

Так же микроядро выполняет некоторые другие функции, например, автоматически распределяет задачи по всем доступным процессорным ядрам в режиме SMP или BMP. Важно отметить, что механизмы микроядра никоим образом не могут быть нарушены процессами. Повлиять на работу ядра вы можете только в том случае, если пишете собственную процедуру обработки того или иного аппаратного прерывания — но это тема отдельного разговора, которая хорошо изложена в документации.

Примечание

Более подробную информацию вы можете найти в штатном справочнике Help, в документе "QNX Neutrino Realtime Operating System → System Architecture" (напомню, он издан на русском языке). Кроме того, весьма полезно почитать, например, электронный документ "QNX Neutrino System Analysis Toolkit → User's Guide".

Сетевая подсистема

Сетевая подсистема QNX Neutrino обеспечивает широкую поддержку современных коммуникационных технологий — стека протоколов TCP/IP (включая IP версий 4 и 6), IPSec, межсетевого экрана PF и Wi-Fi.

Кроме того, сетевая подсистема QNX Neutrino поддерживает уникальную технологию формирования распределенной вычислительной среды — QNX Transparent Distributed Networking (TDN). О ней я скажу чуть подробнее.

QNX TDN является логическим следствием микроядерной архитектуры. Это — традиционный и чрезвычайно эффективный механизм всех ОСРВ семейства QNX. В QNX Neutrino он основан на протоколе четвертого уровня модели ISO OSI, получившем название *Qnet*.

Примечание

В русскоязычной литературе в качестве замены "ISO OSI" встречается аббревиатура "ЭМВОС" — "Эталонная модель взаимодействия открытых систем".

Уникальность этого механизма заключается в том, что Qnet, по сути дела, связывает в единую сетевую инфраструктуру непосредственно экземпляры ядер QNX Neutrino, выполняющихся на разных компьютерах сети (в терминах TDN — на разных узлах). Это означает, что с точки зрения системных и прикладных программ, выполняющихся на разных узлах, все они работают на одной виртуальной ЭВМ.

Конечно, ОЗУ каждого узла сохраняет независимость, т. е. оперативная память узлов является некогерентной. Но важно то, что как только мы запускаем поддержку Qnet на каких-то узлах сети, все программные компоненты этих узлов автоматически становятся сетевыми приложениями без какой-либо модификации.

Возможности TDN хорошо иллюстрирует такой пример — мы можем, работая за терминалом узла А, запустить на ЦПУ узла Б программу, которая хранится в файловой системе узла В, при этом указать этой программе, чтобы она в качестве своей "род-

ной" файловой системы использовала файловую систему узла Г, исходные данные считывала из устройства ввода-вывода на узле Д, в качестве управляющей консоли использовала одну из виртуальных консолей узла Е. Все это указывается в качестве параметров при запуске программы — от программиста не требуется писать ни одной строчки сетевого кода.

Кстати, применяемая в QNX Neutrino графическая оболочка Photon microGUI — это набор взаимодействующих программ, предоставляющих различные сервисы (управление видеоадаптером, управление устройствами ввода, управление сервером шрифтов, управление рабочим столом и т. д.). При использовании Qnet можно создавать весьма интересные сетевые конфигурации графических средств.

В сети Qnet существует несколько способов защиты информации. К ним относятся: отображение (mapping) идентификаторов пользователей, защита сетевых сообщений от несанкционированной модификации, использование различных методов взаимной идентификации узлов и т. д.

Отказоустойчивость Qnet обеспечивается с помощью резервирования физических линий связи. При установлении логических соединений между парами процессов можно задавать один из трех вариантов выбора физического канала:

- *С автоматической балансировкой нагрузки* (loadbalance). В этом случае Qnet самостоятельно принимает решение, по какой из доступных физических линий передавать пакеты. Для выбора адаптера Qnet анализирует время отклика удаленного узла на запросы, посланные по разным физическим линиям, и использует ту линию, которая является наиболее быстрой в данный момент времени. Этот способ используется Qnet по умолчанию.
- *С предпочтением* (preferred). В этом случае задается, какой из сетевых адаптеров предпочтителен для передачи информации. Если предпочтительный адаптер недоступен, то Qnet будет использовать остальные адаптеры, автоматически балансируя нагрузку между ними.

- *Эксклюзивный* (exclusive). Позволяет задавать передачу данных строго через определенный адаптер. Если адаптер недоступен, Qnet не будет передавать данные вообще, даже если с удаленным узлом можно связаться через другие адаптеры.

Изменение варианта не требует перекомпиляции программ, что особенно важно для сертифицированного программного обеспечения.

Механизмы Qnet позволяют элегантно решать проблему нехватки вычислительных ресурсов в случаях, когда в процессе эксплуатации целевой системы реальная нагрузка превышает пределы, предусмотренные при проектировании компьютерной системы реального времени.

В таких случаях в сетевую компьютерную систему, узлы которой объединены Qnet, могут быть добавлены дополнительные ЭВМ, на которые перенесено выполнение тех или иных вычислительных задач. Для этого QNX Neutrino поддерживает механизм, который позволяет распределять запросы клиентов между процессами-серверами, выполняющимися на разных узлах сети Qnet. Решение о направлении запроса на конкретный сервер может задаваться исходя из различных соображений. Например, с учетом загрузки каналов связи или текущей загрузки серверов. Кроме того, этот механизм может быть использован в интересах информационной безопасности.

Примечание

Более подробно о QNX TDN вы можете прочесть в штатных электронных документах:

- "QNX Neutrino Realtime Operating System → User's Guide" (издан на русском языке);
- "QNX Neutrino Core Networking Stack → User's Guide".

Графическая подсистема

В QNX Neutrino для построения графических интерфейсов пользователя обычно используют следующие графические средства:

- графическая оболочка QNX Photon microGUI;

- технология QNX CoreGraphics.

Конечно, поскольку QNX Neutrino является POSIX-совместимой ОС, в нее могут быть перенесены — и перенесены — различные свободно распространяемые средства. Примерами таких переносов (портирований) могут служить графическая среда X Window System и консольная библиотека ncurses.

Примечание

Например, такую популярную реализацию X Window System, как X.Org, перенесли в QNX Neutrino энтузиасты из сообщества **qnx.org.ru**. Что касается библиотеки ncurses, то она позволяет создавать платформонезависимые, быстрые псевдографические интерфейсы, не требующие графической оболочки. Одно из самых известных ncurses-приложений — Midnight Commander.

Однако мы не будем обсуждать это ПО, поскольку оно не входит в официальную поставку QNX Neutrino и не сопровождается компанией QNX Software Systems. Разумеется, это не значит, что вы не можете пользоваться в своих проектах тем, чем хотите.

Photon microGUI

Основное назначение Photon microGUI — построение ЧМИ АРМ операторов АСУ ТП. Эта оболочка по своей функциональности похожа на традиционную для *NIX-подобных ОС графическую оболочку X Window System — обеспечивает работу с несколькими окнами, виртуальными экранами, рабочим столом и т. п. Рабочий стол, например, содержит стартовое меню, панель быстрого запуска, панель задач, хранители экрана, меню и т. д.

Достоинства:

- Photon microGUI — полноценная многооконная графическая среда;
- имеет модульную архитектуру, позволяющую использовать его во встроенных системах;
- чрезвычайно развитые сетевые возможности;
- простота разработки GUI-приложений.

Недостаток:

- плохо справляется с перерисовкой быстро изменяющегося графического потока (например, воспроизведение видеороликов).

Этот недостаток является обратной стороной высокой модульности и гибкости конфигурирования Photon microGUI.

Разработка приложений для Photon microGUI обычно ведется с помощью достаточно простого в использовании визуального построителя PhAB (Photon Application Builder). Кроме того, в Photon microGUI с помощью технологии XPhoton можно запускать некоторые приложения X Window System, после перекомпиляции под QNX Neutrino, разумеется.

Технология QNX CoreGraphics

Предложена в 2006 году. Первоначальное название — QNX Advanced Graphics. Предназначена для построения сложных графических интерфейсов, требующих быстрой перерисовки 2- и 3-мерных изображений. В основе QNX CoreGraphics лежит реализация стандарта OpenGL® ES, представляющего собой подмножество OpenGL для встроенных систем. Реализация QNX поддерживает профили Common и EGL.

Достоинства:

- высокое быстродействие;
- простота переноса программ Open GL ES из других ОС на уровне исходных текстов;
- не требуется графическая оболочка;
- может использоваться одновременно с Photon microGUI.

Недостаток:

- необходимость знания технологии Open GL ES разработчиками.

Суть технологии CoreGraphics заключается в том, чтобы предоставить программам доступ непосредственно к видеодрайверам без передачи сообщений и переключений контекста при рисовании. Это существенно повышает скорость графических приложений при компактности, необходимой для встроенных систем.

Физически QNX CoreGraphics представляет собой набор компонентов разработки (библиотеки и заголовочные файлы) и компонентов времени исполнения (библиотеки времени исполнения, OpenKODE-совместимый менеджер композиции и др.).

Примечание

Менеджер композиции выполняет для приложений OpenGL и OpenVG функцию, чем-то напоминающую функцию оконного менеджера в X Window System или в Photon microGUI. До появления стандарта OpenKODE разработчики должны были вручную кодировать расположение и размеры окна. Теперь, если приложение OpenGL или OpenVG написано с использованием OpenKODE API, расположение и размер "окна" можно задавать с помощью настроек менеджера композиции.

Вообще, в графической подсистеме QNX немало интересных особенностей. Например, помимо обычных веб-браузеров (Mozilla, Bon Echo) существует технология *QNX Web Browser Engine*. Она основана на открытом движке веб-браузеров WebKit (**http://webkit.org**) и заменила технологию *Voyager*.

Пожалуй, я скажу кое-что о Voyager. Во-первых, чтобы защитить эту технологию от несправедливых упреков, во-вторых, чтобы было понятно, для чего предназначена Web Browser Engine (см. штатный электронный документ "QNX Neutrino Web Browser Engine. Developer's Guide").

Технология QNX Voyager (впрочем и QNX Web Browser Engine тоже) предназначена для того, чтобы вставлять отображение веб-страничек прямо в некоторую часть окна приложения. Voyager состоит из двух частей — сервера и клиента. Сервер Voyager — это программа vserver или netfront, взаимодействующая через протокол HTTP с каким-либо веб-сервером и рисующая предоставленной ему области окна приложения Photon. Область в окне приложения для рисования предоставляет клиент Voyager — Photon-виджет PtWebClient, доступный в PhAB. Клиент Voyager не отличается интеллектом — он поддерживает ровно те "фишки" и версии HTTP, которые поддерживает используемый в паре с клиентом сервер.

Что касается поставлявшегося ранее "браузера" Voyager, то это был, по сути дела, просто пример реализации приложения с использованием PtWebClient. Поэтому не надо требовать от него функциональности "нормальных" браузеров.

6.3.3. Обеспечение отказоустойчивости

Для начала вспомним определение термина *"отказоустойчивость"* (fault tolerance), предлагаемое IEC 61508:

"Способность функциональной единицы продолжать выполнять требуемую функцию при наличии неисправностей и ошибок".

Из материала *занятия 3* следует вспомнить, что в качественном программном обеспечении содержится 290 дефектов на 1 млн строк кода. Это значит, что теоретически любой программный компонент в произвольный момент времени может выйти из строя — т. е. "зависнуть" или "упасть". Это, конечно, не совсем так — вспомните о подходе "Proven-in-Use". Но указанный подход применим только к штатным компонентам QNX Neutrino, миллионы экземпляров которой годами эксплуатируются в самых разных приложениях по всему миру. Как же обеспечить отказоустойчивость наших с вами компонентов, которые имеют всего лишь десятки или сотни экземпляров?

Для этого QNX Neutrino содержит ряд механизмов, которые можно условно разделить по назначению: локализация сбоя, идентификация сбоя, восстановление после сбоя.

Локализация сбоев

Локализация сбоев позволяет ограничить, так сказать, "масштабы бедствия" — т. е. сократить перечень функций компьютерной системы, которые могут отказать или ухудшиться в случае сбоя.

Локализация сбоя связана с такой характеристикой компьютерных систем, как *SPoF* (Single Point of Failure — точка отказа). SPoF — эта часть системы, сбой в которой может привести к отказу всей системы. В QNX Neutrino точкой SPoF является модуль *procnto* (микроядро с менеджером процессов), размер которого составляет около 0,1 млн строк кода. Для сравнения, размеры

ядер других популярных ОС в зависимости от конфигурации могут колебаться от 3 до 40 млн строк кода. Модуль же procnto является постоянным и никак не зависит от конфигурации данного экземпляра QNX Neutrino — т. е. является тем самым модулем procnto, который верифицирован фирмой-разработчиком и работает в миллионах других систем. Я заостряю ваше внимание на этом потому, что при любом изменении конфигурации даже проверенного ядра вы фактически получаете уже другое, никем не верифицированное, ядро!

Итак, как вы уже поняли, в QNX локализация сбоя в первую очередь обеспечивается микроядерной архитектурой. Таким образом, фатальная ошибка в вашем графическом приложении в обработчике нажатия кнопки не повлияет ни на ядро, ни на сетевую подсистему, ни даже на графическую систему — "рухнет" только ваше приложение. Если вы написали драйвер, скажем, сетевого адаптера, то ошибка может привести к отказу сетевой подсистемы, но не повлияет на остальные части системы. При этом и ваше графическое приложение, и сетевая система могут быть восстановлены простым запуском соответствующих программ. Но о восстановлении мы еще поговорим особо.

Другим важным средством локализации сбоев в QNX Neutrino является механизм адаптивного квотирования ресурсов процессора. Его наличие является основным требованием авиационного стандарта ARINC 653.

Необходимость квотирования ресурсов в первую очередь диктуется возможными логическими ошибками в программе или настройках системы, не приводящими к аварийному завершению программы. Например, если высокоприоритетный поток выполняет холостой бесконечный цикл, то низкоприоритетные потоки никогда не смогут получить доступ к ресурсам ЭВМ — ведь холостой поток будет иметь право на максимальный доступ к ресурсам. Вторая решаемая проблема заключается в том, что в современных системах на ЭВМ могут выполняться сотни и даже тысячи конкурирующих за процессорные ресурсы потоков — чрезвычайно сложно корректно настроить такую систему даже располагая мощными инструментами анализа.

Суть механизма адаптивного квотирования изображена на рис. 6.1. В некоторой вычислительной системе выполняется шесть потоков — A, B, C, D, E и F. Каждый из потоков имеет свое значение приоритета: максимальный приоритет имеет поток A, минимальный — поток F. Для квотирования ресурсов процессора созданы три логических раздела — по 30%, 30% и 40% соответственно. Поток A запускается в разделе 1, потоки B, C и D — в разделе 2, остальные потоки — в разделе 3. На рисунке показано, что потоки ведут себя стандартным образом для системы реального времени, но только в пределах своих разделов. Если же какие-то потоки не полностью используют свой раздел (в нашем примере — раздел 3), то "лишнее" процессорное время распределяется между готовыми к исполнению потоками всех разделов в соответствии с их приоритетами. Кстати, именно поэтому механизм квотирования называется *адаптивным*.

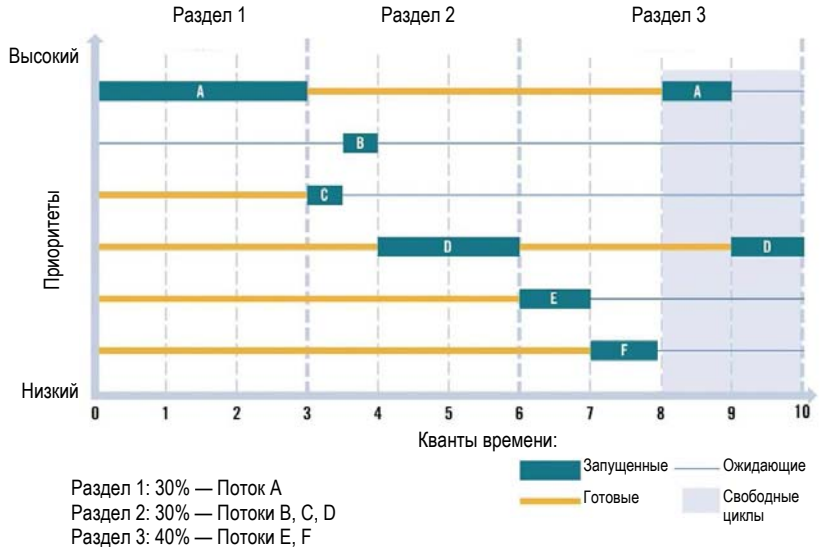


Рис. 6.1. Адаптивное квотирование

Что же дает механизм адаптивного квотирования процессорных ресурсов:

- повышение защищенности и коэффициента готовности целевой системы за счет невозможности монополизации ресурсов какой-либо программой (при DoS-атаках и некорректном коде);
- сокращение трудозатрат на сопровождение целевой системы на 25—30% (по оценкам экспертов компании QNX Software Systems) за счет локализации аномального поведения.

Следует сказать, что защита от монополизации ресурсов позволяет запускать в критичной системе даже потенциально ненадежные программы — за счет квотирования они, даже имея высокий приоритет, не смогут воспрепятствовать выполнению важных задач.

Подробнее о механизме адаптивного квотирования процессорных ресурсов можно узнать из электронного документа "QNX Neutrino Adaptive Partitioning → User's Guide".

Идентификация сбоев

Средства идентификации сбоя разработаны из предположения, что следующим шагом будет выполняться анализ причин сбоя и/или восстановление.

Для того что была понятна суть подходов QNX Neutrino к идентификации сбоев, наверное, не лишним будет пояснить на уровне, так сказать, понимания физического смысла, что вообще собой представляет *сбой* или *крах программы* в QNX Neutrino.

Напомню, что такое выполняющаяся программа, т. е. *процесс* (рис. 6.2):

- сегменты кода и данных исполняемого файла, загруженные в ОЗУ и изолированные от других процессов;
- контейнер потоков и динамически выделенных ресурсов;
- информация, зарегистрированная в таблицах ядра и менеджера процессов (pid, ppid, таймеры, префиксы, код завершения и др.).

Завершение процесса инициируется либо им самим, либо из внешнего мира. Завершается процесс в две фазы:

- *Фаза 1*

  Происходит собственно уничтожение процесса, т. е. освобождение физически занятых ресурсов (ОЗУ, ОСВ-блоки, таймеры).

- *Фаза 2*

  Происходит внутри менеджера процессов. Выполняется очистка таблицы процессов менеджера процессов.

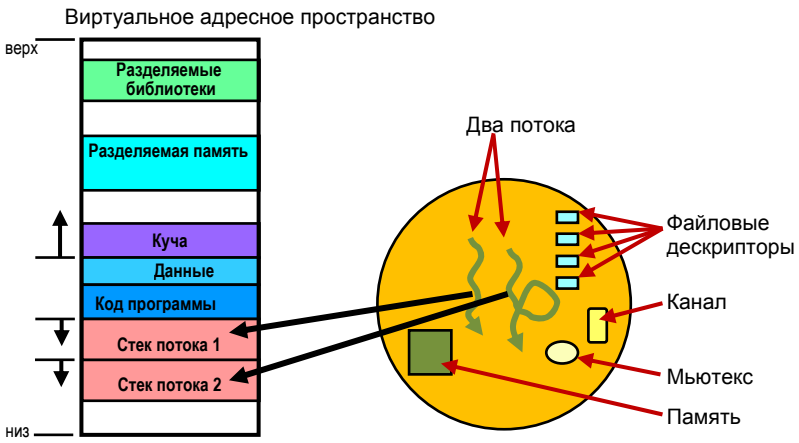


Рис. 6.2. Схема процесса QNX Neutrino

Завершение процесса из внешнего мира инициируется путем посылки процессу одного из сигналов POSIX. Обычно для завершения процесса используют сигнал `SIGTERM` или `SIGKILL` — для этого процесс, посылающий сигнал, должен иметь соответствующие полномочия (иметь тот же эффективный идентификатор пользователя или принадлежать суперпользователю root). Но соответствующий сигнал может сгенерировать и ядро, если процесс нарушит принятые в операционной системе "правила игры", например, попытается "залезть" в чужую область памяти или выполнить деление на нуль (в данных случаях будут сгенерированы, соответственно, сигналы `SIGSEGV` и `SIGFPE`, но есть и другие сигна-

лы — см. документацию). В результате инициируется завершение процесса.

Коротко говоря, необходимо понимать, что:

- "крах" программы — это завершение процесса ядром ОС при нарушении процессом "правил игры";
- "крах" программы — это штатная ситуация с точки зрения ОС;
- "крах" программы — это, по сути дела, положительное явление, сужающее SPoF и обеспечивающее отказоустойчивость системы в целом.

Процедура завершения процесса и в штатной ситуации, и при ее "крахе" осуществляется только ядром. В QNX Neutrino есть возможность "попросить" ядро в случаях, когда процесс "нашкодит" и будет уничтожаться, сообщать об этом определенной программе и/или отдавать информацию, необходимую для анализа причин "краха". В штатной поставке QNX Neutrino есть две программы, которые могут передать ядру вашу "просьбу":

- Программа *dumper*. Она "просит" ядро отдавать ей информацию о сбоившей программе и сохраняет эту информацию в виде так называемого core-файла. Полученный core-файл можно проанализировать в графическом режиме с помощью интегрированной среды разработки — на какой строке программы произошел сбой, каково было значение переменных и т. д.
- Монитор ключевых процессов *hum*. Первое его назначение — максимально быстрое определение факта сбоя или "зависания" процесса и принятие мер для восстановления нормальной работы компьютерной системы. Подробнее о мониторе ключевых процессов мы поговорим в рамках обсуждения средств восстановления.

Давайте подытожим:

- При сбое процесса ядро может немедленно извещать об этом монитор ключевых процессов и отдавать информацию, необходимую для анализа, программе *dumper*.
- При "зависании" процесса монитор может это обнаружить (у него есть специальный механизм *heart-beat* — "сердцебиения") и сделать "снимок" состояния процесса с помощью программы *dumper*.

Кроме того, в QNX Neutrino есть механизмы:

- автоматического извещения процесса-сервера о завершении процесса-клиента (сервер должен "подписаться" на получение соответствующих извещений от ядра);
- автоматического перенаправления клиентских запросов на резервный ("теневой") сервер на основе перегрузки путевых имен;
- перенаправления ("редиректа") клиентских запросов между несколькими серверами (возможно — в сети Qnet).

Подробнее о первых двух механизмах можно почитать в упомянутом учебнике Кртена, переведенном на русский язык. Эта книга доступна в оригинале в штатной электронной документации ("QNX Neutrino Realtime Operating System → Getting Started with QNX Neutrino: A Guide for Realtime Programmers").

Восстановление после сбоев

Рассказ о средствах восстановления после сбоя следует начать с технологий быстрого старта операционной системы. Их две:

- *QNX FastBoot*. Реализуется в BSP и основана на том обстоятельстве, что можно написать такой код инициализации процессорной платы (вспомните о базовом BSP), который мог бы стартовать без BIOS и выполнять минимально достаточные действия. Такой подход позволяет, например, на платах с процессором Intel Atom обеспечивать запуск полноценной конфигурации QNX Neutrino в течение 1—2 секунд после подачи питания.

- *QNX Instant Device Activation*. Реализуется в BSP и заключается во "внедрении", если можно так выразиться, кода драйвера устройства в код BSP. "Внедренный" код (его называют *"мини-драйвером"*) выполняется до старта ядра и обеспечивает обработку внешних событий (например, сигналов, передаваемых по шине CAN) через десятки миллисекунд после подачи питания на процессорную плату.

Кстати, "мини-драйверы" после окончания загрузки ОС можно без задержек времени или потерь данных "преобразовать" в полноценный драйвер. Впрочем, можно так и оставить работать "мини-драйвер", если его функционала достаточно. Подробнее эта технология описана в штатном электронном документе "QNX Neutrino Instant Device Activation → User's Guide".

Что касается отказа отдельных программ, то, конечно, важными средствами их восстановления являются упомянутые ранее механизмы перенаправления. Но я хотел бы продолжить разговор о мониторе ключевых процессов.

Чтобы обеспечить собственную отказоустойчивость, монитор для начала "клонирует" самого себя, т. е. порождает дублирующий процесс, которому, используя механизм разделяемой памяти, предоставляет полную информацию о мониторинге целевой системы. Монитор и его дублер строго следят друг за другом, и при сбое одного из них второй процесс немедленно порождает новый процесс-дублер.

В качестве объекта мониторинга могут быть заданы любые (серверные или клиентские) процессы, выполняющиеся как на той же ЭВМ, что и монитор, так и на других узлах Qnet. Для каждого из объектов мониторинга задается порядок действий при их "крахе" или "зависании". Действия могут задаваться в широком диапазоне — от простого перезапуска до сложных сценариев многоступенчатого восстановления достаточно сложных распределенных систем.

На одной ЭВМ может быть запущен один тандем монитора ключевых процессов и его дублера. Мониторы могут быть запущены на любом количестве узлов Qnet. И при этом каждый из монито-

ров сети Qnet может контролировать объекты, работающие на любом из доступных узлов Qnet.

Помимо восстановления процессов в распределенных целевых системах существует еще проблема восстановления логических соединений. Ее суть заключается в следующем. При отсутствии физического соединения сетевая подсистема пытается в течение определенного времени (тайм-аута) "достучаться" до удаленной ЭВМ. Если по истечении тайм-аута соединение не восстановилось, функция в приложении, инициировавшая передачу данных, завершится с ошибкой `EINTR`. Следовательно, для обеспечения восстановления работоспособности компьютерной системы при сбоях физических линий связи программист должен предусмотреть в клиентской программе обработку ошибок `EINTR`, возникающих при разрывах соединений, и принимать меры к восстановлению соединений. Это усложнит программу.

Для того чтобы разделить логику прикладной задачи от логики обработки отказов линий связи, а также для автоматизации идентификации таких отказов в приложении существует технология восстановления логических соединений, основанная на использовании библиотеки восстановления клиента (Client Recovery Library). Подробнее этот вопрос можно прочесть в штатном электронном документе "QNX Neutrino High Availability Framework → Developer's Guide".

6.4. Выводы

QNX — это высокотехнологичная платформа с обширным современным функционалом. Технологии QNX обстоятельно описаны в документации, что в сочетании с доступностью исходных кодов дает разработчикам компьютерных систем реального времени достаточно богатые возможности для творчества.

Для полноты картины перечислю сертификаты, полученные на ОСРВ QNX Neutrino компанией QNX Software Systems.

- POSIX PSE52 Realtime Controller 1003.13-2003. Этот сертификат подтверждает то, что у QNX Neutrino состав и функцио-

нальность API (интерфейса прикладного программирования), утилит и командного интерпретатора соответствует требованиям профиля PSE52 (контроллер реального времени) стандарта IEEE 1003.13-2003.

Примечание

Стандарт IEEE 1003.13-2003 (он же ISO/IEC ISP 15287-2, он же POSIX.13) определяет так называемые *профили прикладных контекстов реального времени* — т. е. стандартные подмножества POSIX-функциональности IEEE 1003.1, необходимые для систем реального времени.

- Common Criteria Evaluation Assurance Level (EAL) 4+, т. е. сертификат соответствия требованиям "Общих критериев" по ОУД4+.
- IEC 61508 Safety Integrity Level 3 (SIL3).

Что такое "Общие критерии" и IEC 61508, а также что значат уровни ОУД и SIL, мы обсуждали на *занятии 3*.