

Создание отказоустойчивых, высокоскоростных и адаптивных систем регистрации, физических измерений и управления исследовательскими установками.

Курякин А.В., ФГУП «РФЯЦ-ВНИИЭФ», г. Саров.

Данная работа посвящена актуальному вопросу создания программного обеспечения для отказоустойчивых, высокоскоростных и адаптивных систем регистрации данных физических экспериментов и управления исследовательскими физическими установками. Этот вопрос особенно важен при создании программно-аппаратных комплексов для радиационно-опасных физических установок, например, тритиевых газовых комплексов [1], тритиевых мишеней для экспериментов на ускорителях [2] и т.д.

Ясно, что программное и аппаратное обеспечение этих установок должно быть отказоустойчивым. Вместе с тем многие ядерно-физические установки требуют регистрации быстрых сигналов, что также должно быть обеспечено программно и аппаратно. Кроме того, для исследовательских установок весьма актуальным является вопрос адаптивности программного обеспечения, так как обычно исследовательские установки постоянно модифицируются (на то они и исследовательские). Это приводит к тому, что у разработчиков программного обеспечения постоянно ощущается сильнейший дефицит времени на отладку прикладного кода, что неизбежно приводит к наличию в этом коде ошибок различной степени тяжести. Поэтому программное обеспечение должно быть гибким, допускающим достаточно быстрое изменение и отладку прикладного кода, но без потери его высокой надежности.

Сложность решения данной задачи связана с тем, что требования высокой отказоустойчивости, скорости и адаптивности являются в значительной степени взаимно противоречивыми. Программное обеспечение можно сделать адаптивным, то есть «гибким» и настраиваемым, но тогда оно будет по необходимости сложным, и сделать его отказоустойчивым будет непросто. Можно сделать программное обеспечение отказоустойчивым, но далеко не просто сделать его одновременно высокоскоростным.

Данная работа не претендует на полное рассмотрение этого сложного вопроса, что потребовало бы, вероятно, написания отдельной книги. Здесь лишь приводится опыт автора по построению подобного рода систем и описание возникающих при этом проблем и решений, позволяющих эти проблемы устранить или хотя бы ослабить.

Чтобы исключить разночтения, следует уточнить терминологию – что именно понимается под отказоустойчивыми, высокоскоростными и адаптивными системами.

Отказоустойчивость – это свойство системы регистрации и управления, которое заключается в том, что система должна сохранять возможность безаварийной работы, даже при отказе отдельных её компонентов. Отказоустойчивость не следует путать с надежностью, то есть низкой вероятностью отказа отдельных компонентов системы. Она обеспечивается не столько надежностью (качеством) своих компонентов, сколько структурой системы и рядом специальных мер. Отказоустойчивой является система, которая полностью или частично сохраняет свою функциональность даже в случае, если отдельные компоненты системы все же отказали, например, под воздействием внешних неблагоприятных факторов. Примером высоконадежной, но не отказоустойчивой системы являются швейцарские часы, великолепно работающие – до первой песчинки в часовом механизме. Примером отказоустойчивой системы является подводная лодка, разделенная на герметичные переборки, которая сохраняет, частично или полностью, ходовые качества и боеспособность даже в случае крупной пробоины в корпусе.

Можно перечислить структурные свойства и меры, которые в совокупности обеспечивают отказоустойчивость. В первую очередь, это модульность, параллелизм, автономия и изоляция сбоев. Программное обеспечение отказоустойчивой системы должно состоять из четко оформленных модулей, которые работают параллельно (одновременно) и автономно (независимо друг от друга), причем модули изолированы, то есть сбой в одном модуле не влечет за собой «каскад» сбоев в других модулях. Перечисленные выше свойства обеспечиваются различными способами. Важнейшими среди них являются автономные контроллеры, процессы операционной системы (OS) и виртуальные машины (VM). При использовании контроллеров модульность, автономия, параллелизм и защита обеспечиваются на аппаратном уровне, но это довольно дорогое во всех смыслах решение, к тому же не всегда совместимое с высокой скоростью регистрации. Защита на уровне процессов в современных процессорах и операционных системах поддерживается аппаратно и программно. Модульность и автономность процессов обеспечивается средствами OS, параллелизм обеспечивается системой разделения времени (приоритетная вытесняющая многозадачность), а изоляция – разделением адресного пространства процессов и защитой памяти на аппаратном уровне MMU (memory management unit). Такая защита реализована, например, в системах Unix, Linux, QNX, Windows на процессорах x86. Подробное рассмотрение этих вопросов можно найти в литературе по теории построения операционных систем [3]. Другим способом обеспечения тех же самых свойств являются виртуальные машины, которые также обеспечивают защиту логических адресных пространств у прикладных программ и изоляцию ошибок, но только не на аппаратном машинном уровне, а на уровне языка

программирования и исполнительной (runtime) системы, роль которой выполняет интерпретатор байт-кода данной виртуальной машины. Примерами таких систем являются виртуальные машины Java, Erlang [4], а также система Daq Pascal [5] в пакете CRW-DAQ [6].

Если сравнивать концепции процессов OS и VM, то защита процессов, безусловно, дает более высокую производительность программ, но имеет ограниченный характер и требует аппаратной поддержки (например, MMU). Виртуальная машина, при правильной реализации интерпретатора байт-кода, может обеспечить гораздо более высокую степень защиты модулей, причем аппаратно независимой, но при этом вносит существенное замедление в работу программы, достигающее 2-10 раз, в зависимости от качества интерпретатора VM. Некоторые свойства VM (например, защита объектов и массивов, предотвращение «утечки» памяти и других используемых ресурсов) недостижимы для процессов OS, поэтому VM не следует просто так отвергать из-за более низкой производительности, если вопрос отказоустойчивости в данной системе имеет приоритет над другими свойствами. Наилучшее сочетание, конечно, дает использование обеих защит - как процессов OS, так и виртуальных машин, каждая защита работает в своей области.

Другими, не менее важными мерами обеспечения отказоустойчивости являются резервирование, иерархия управления, диагностика и рефлексия. Резервирование – это введение в систему аппаратной, программной и информационной избыточности. Например, дублирование регистрирующих каналов или исполнительных механизмов, дающее возможность продолжения работы на резервном оборудовании при критическом сбое основного оборудования. Диагностика и рефлексия – это наличие средств наблюдения системы за состоянием своих компонентов, а также критериев «правильной» их работы. Иерархия управления предполагает разделение функций и четкую систему подчиненности, при которой «ведущие» модули высокого уровня предписывают «ведомым» низкоуровневым автономным модулям исполнять определенную задачу и следят за её исполнением. Например, некоторый «супервизор» может запускать ряд автономных процессов OS для исполнения отдельных заданий, следить за их работой и перезапускать эти процессы, если они перестали подавать признаки жизни, например, «сбрасывать» аппаратный или программный сторожевой таймер (watchdog). В иерархической системе управления «ведущие» модули выполняют задачи стратегического планирования, а «ведомые» - задачи тактики и исполнения. При этом автономность модулей гарантирует, что «ведомые» модули будут продолжать исполнение своих функций при временном разрыве связи с «ведущими» модулями.

Здесь можно поспорить с авторами, которые указывают на особую устойчивость лишенных иерархии «сетевых» систем, подобных «ризоме» или плесени, где каждый автономный модуль равен другому. С одной стороны, такие системы действительно обладают высокой стойкостью к внешним воздействиям, но с другой они не способны к решению каких-либо задач управления, кроме одной – поддержания своего существования. Типичным примером «сетевой» системы, лишенной иерархии, является компьютерный вирус, умеющий только одно – заражать все большее число компьютеров. Вот только вряд ли можно считать его хорошим примером системы управления. Ведь все-таки системы управления создаются для решения определенных задач, и должны иметь «центральный пульт управления», которого лишены «сетевые» системы без иерархии.

Следует подчеркнуть, что отказоустойчивость системы обеспечивается, только если все перечисленные свойства обеспечиваются именно в комплексе, одновременно, иначе их использование в значительной мере теряет смысл. Так, резервирование лишено смысла, если одновременно не обеспечена диагностика и рефлексия, без которых система не сможет увидеть, что какой-то компонент изменил свои параметры и понять, что это измерение означает его сбой и необходимость его замены на резервный компонент.

Под высокоскоростными системами здесь понимаются такие системы регистрации, пропускная способность которых близка к максимально достижимой на данной аппаратной платформе. Понятно, что конкретное численное значение этой пропускной способности зависит от аппаратуры и может отличаться на порядки. Здесь важно то, что высокоскоростная система регистрации подвергает данные на пути от «источника» к «приемнику» (например, от высокоскоростного АЦП к файлу в базе данных) минимальному количеству операций копирования, буферизации, ожидания при синхронизации, интерпретации и т.д.

Высокоскоростные системы по необходимости должны опираться на аппаратные возможности данной платформы. Среди таких возможностей можно упомянуть прямой доступ к памяти DMA (direct memory access), использование общей памяти (shared memory), специализированных сигнальных процессоров DSP (digital signal processor).

Под адаптивностью здесь понимается возможность достаточно быстрого приспособления программы к изменяющимся условиям эксплуатации. Адаптивность предполагает модульность и возможность быстрого, буквально в режиме online, изменения алгоритмов работы отдельных модулей, без необходимости полной «сборки» и перезапуска всего проекта, который может быть большим и не допускающим даже кратковременных остановок работы, как, например, на радиационно-опасных установках.

Адаптивность также может обеспечиваться многими методами. Наверное, самым

распространенным является метод параметризации алгоритмов управления, когда в них закладывается ряд параметров, которые можно изменять извне. Этот способ хорош тем, что он вообще не требует перекомпиляции прикладного ПО. Но в то же время он имеет ограниченные возможности адаптации и порой сильно усложняет сам алгоритм управления, делая его отладку сложным делом из-за необходимости проверки всех возможных «ветвлений» параметризованного алгоритма управления.

Другим способом обеспечения адаптивности является динамическая компоновка, например, с помощью загружаемых библиотек DLL (dynamic link library). В этом случае возможности адаптации определяются в основном качеством API (application program interface), с помощью которого загружаемый модуль взаимодействует ядром системы.

Виртуальные машины также могут обеспечить высокую степень адаптивности, особенно если они снабжены механизмом компиляции «на лету» или JIT-компиляции (just in time compilation). Примером может служить система DAQ Pascal пакета CRW-DAQ, в которой каждый модуль с прикладным кодом DAQ Pascal исполняется в отдельной виртуальной машине, которая может быть в любой момент за доли секунды приостановлена, перекомпилирована и запущена вновь, причем как система в целом, так и все остальные модули продолжают нормальную работу.

Одновременное достижение отказоустойчивости, высокой скорости и адаптивности является трудной задачей и требует принятия компромиссных решений и усложнения структуры программы. Это связано с тем, что меры для обеспечения указанных свойств часто противоречат друг другу и требуют разумного подхода. Например, изоляция сбоев, как в форме защиты памяти процессов OS, так и в форме VM, противоречит высокой скорости, так как созданные «барьеры» защиты адресных пространств приходится преодолевать при передаче данных (например, с помощью сообщений или каналов), теряя на этом время. Или, например, динамическая компоновка с загрузкой DLL библиотек повышает адаптивность, но снижает отказоустойчивость за счет введения в адресное пространство основной программы потенциально «опасного» и плохо проверенного прикладного кода DLL, ошибка в котором «убивает» всю систему. Можно сказать, что каждая правильно спроектированная система неизбежно будет компромиссом между отказоустойчивостью, скоростью и адаптивностью, при котором каждое из этих свойств ограничено приемлемым для данной задачи уровнем.

Рассмотрим в качестве примера (Рис.1), иллюстрирующего эти тезисы, программу драйвера для сбора данных с помощью измерительного преобразователя напряжения E14-140 фирмы L CARD [7], созданную в пакете CRW-DAQ. Этот, подключаемый по шине USB, 16 (32) – каналный преобразователь содержит 14-битный

АЦП с частотой опроса до 100 и даже 400 кГц, в зависимости от модификации. В отличие от цифровых осциллографов, которые зачастую имеют и более высокую частоту АЦП, он позволяет вести сбор данных в непрерывном режиме, а не только в виде «снимков» конечной длины. Так или иначе, E140 можно рассматривать как хороший пример достаточно быстрого «блочного» устройства. При этом программа сбора данных E140 является лишь малой частью «большой» системы, работающей под управлением пакета CRW-DAQ, которую здесь нет смысла рассматривать, а потому добавление нового драйвера E140 не должно приводить к деградации отказоустойчивости этой «большой» системы.

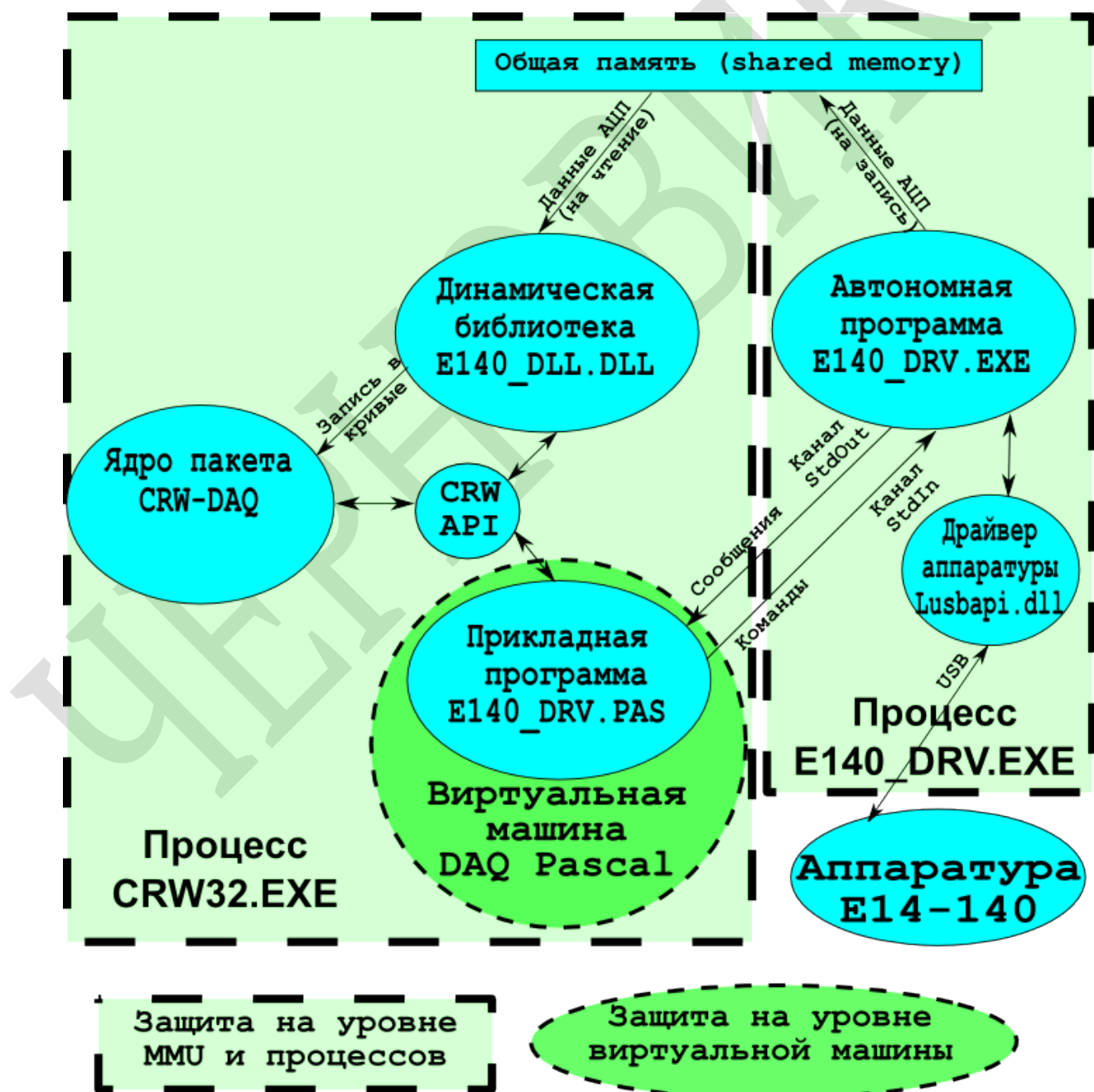


Рис.1. Структура программы сбора данных для преобразователя E14-140.

Преобразователь E140 поставляется с «фирменным» драйвером, доступ к которому дается через динамическую библиотеку `Lusbapi.dll`. В конкретной измерительной системе кроме этой библиотеки требуется также код для настройки АЦП, преобразования измеренных данных в требуемые единицы измерения, фильтрации и т.д. При создании отказоустойчивой системы этот «добавочный» по отношению к «основной» системе исполняемый код должен считаться потенциально опасным источником ошибок, не из-за недоверия к конкретной фирме, разумеется, а из общего принципа изоляции ошибок при построения отказоустойчивых систем. Поэтому, несмотря на наличие в пакете CRW-DAQ механизма (API) для подключения загружаемых библиотек DLL, идея реализации драйвера в виде простой загружаемой DLL библиотеки, работающей в адресном пространстве основной измерительной программы, была отвергнута.

Для обеспечения приемлемого уровня отказоустойчивости было решено выделить драйвер в отдельный процесс `E140_DRV.EXE`, изолированный от процесса основной программы `CRW32.EXE` средствами OS и MMU. Однако, обеспечивая защиту, изоляция процесса создает проблему передачи измеренных данных, особенно при высокой скорости обмена. Из имеющихся средств IPC (inter process communication) на платформе x86 наиболее быстрым способом является, конечно, общая память, поскольку, в отличие от сообщений или каналов связи, она позволяет вообще исключить промежуточное копирование данных, при сохранении, в то же время, защиты на уровне страниц памяти MMU. Однако использование общей памяти затрудняется тем, что она не имеет готового метода синхронизации потоков, позволяющего передавать данные без потерь. Поэтому «общение» процессов через общую память всегда должно сопровождаться введением того или иного механизма синхронизации приема и передачи данных.

Наиболее удобным и часто используемым механизмом такой синхронизации в пакете CRW-DAQ являются анонимные каналы (anonymous pipe), позволяя передавать и принимать данные дочернего консольного процесса через переназначение потоков стандартного ввода-вывода `StdIn` и `StdOut`. Язык DAQ Pascal в пакете CRW-DAQ хорошо защищен механизмом виртуальной машины, а также имеет развитую библиотеку для работы с процессами OS. Поэтому все функции по запуску драйверного процесса, его остановки, перезапуска и при критическом сбое, передачи команд в канал консоли `StdIn` и приема ответных сообщений через канал `StdOut`, интерфейса пользователя для настройки АЦП и ряд других функций возложен на прикладную драйверную программу `E140_DRV.PAS`, написанную на языке DAQ Pascal. Перечисленные функции не требуют

высокой скорости работы, и поэтому их реализация в рамках виртуальной машины не создает препятствий для высокой скорости сбора данных, обеспечивая, в то же время, хорошую отказоустойчивость. Использование средств языка DAQ Pascal, с его механизмом компиляции «на лету», также дает высокую адаптивность прикладной программы, которую достаточно легко можно менять «под задачу». Однако проблему быстрого приема данных АЦП из общей памяти довольно «медленный» язык DAQ Pascal решить не может. Поэтому, кроме запуска процесса E140_DRV.EXE, программа E140_DRV.PAS также загружает динамическую библиотеку E140_DLL.DLL, которая вызывается при получении от процесса E140_DRV.EXE сообщения о готовности очередного блока данных АЦП, для быстрого чтения и обработки этого блока данных.

Система работает следующим образом. Пакет CRW-DAQ (процесс CRW32.EXE) запускает программу E140_DRV.PAS, работающую в защищенной виртуальной машине. Эта программа запускает драйверный процесс E140_DRV.EXE, переназначив его стандартный ввод-вывод в анонимные каналы, и загружает динамическую библиотеку E140_DLL.DLL. Она также обеспечивает интерфейс пользователя и сервисные функции по сохранению и отображению данных, заданию режимов АЦП и другие. При пуске и остановке измерений или изменении параметров программа E140_DRV.PAS посылает в консоль StdIn драйверного процесса соответствующие команды. После этого драйверный процесс с помощью фирменной библиотеки Lusbapi.dll начинает процесс сбора данных в буфер общей памяти, разделенный на несколько блоков. При заполнении очередного блока общей памяти драйверная программа посылает в StdOut сообщение с описанием этого блока, содержащее имя буфера общей памяти и смещение (индекс) блока данных в нем. Получив это сообщение из анонимного канала связи, программа E140_DRV.PAS вызывает код динамической библиотеки E140_DLL.DLL для чтения данных АЦП. Код библиотеки подключает буфер общей памяти к адресному пространству процесса CRW32.EXE, если он еще не подключен, а затем читает из буфера указанный в сообщении блок данных, преобразует коды АЦП в требуемые единицы измерения (милливольты, например), и записывает их в базу данных, то есть в кривые, по терминологии пакета CRW-DAQ. Связь между ядром пакета, программой DAQ Pascal и кодом DLL осуществляется через тщательно проработанный программный интерфейс API. При этом запись данных в кривые идет прямо через API, минуя код DAQ Pascal, для обеспечения высокой производительности. Заметим, что за счет использования общей памяти исключаются лишние операции копирования данных, так как код DLL библиотеки берет данные АЦП прямо из той же памяти, куда они были записаны драйвером Lusbapi.dll. В то же время канал связи между процессами используется только для передачи коротких

сообщений, обеспечивающих синхронизацию данных. Это общий принцип при работе с быстрыми каналами связи с общей памятью – основной поток высокоскоростных данных и поток синхронизирующих команд и сообщений должны быть разделены, причем поток команд и сообщений должен и иметь четкую дисциплину доступа.

Следует заметить, что загружаемая библиотека E140_DLL.DLL, обеспечивая высокую скорость обмена данных, по-прежнему остается, с точки зрения защиты, потенциальным источником критических сбоев системы CRW-DAQ, так как работает в адресном пространстве основной программы CRW32.EXE, но при этом не защищена виртуальной машиной. Однако здесь надо заметить, что код этой библиотеки сведен до минимально достижимого объема (около 200 строк), за счет вынесения всех функций, кроме приема данных из общей памяти, их преобразования и записи в базу данных пакета CRW-DAQ, в защищенные компоненты (процесс E140_DRV.EXE и программу E140_DRV.PAS). Минимизация объема небезопасного кода - это пример компромисса между обеспечением отказоустойчивости и высокой скорости работы программы.

Следует также отметить, что адаптивность созданной программы повышается также тем, что для редактирования и компиляции всех компонентов драйверной программы в пакете CRW-DAQ имеются встроенные средства разработки, редакции и отладки. Программа E140_DRV.PAS редактируется и компилируется встроенными средствами языка DAQ Pascal. Код загружаемой библиотеки E140_DLL.DLL и автономной программы драйвера E140_DRV.EXE написан на языке Object Pascal. Его можно редактировать встроенным в пакет редактором DPR проектов и компилировать интегрированным в пакет компилятором командной строки DCC (Delphi Command line Compiler). Таким образом, пакет дает широкие возможности адаптации кода драйвера для конкретной прикладной задачи.

Опыт опытной эксплуатации созданного программного обеспечения показал его высокую отказоустойчивость по отношению к сбоям драйвера. Если драйверный процесс E140_DRV.EXE «падает» (например, при физическом разрыве связи USB), то основной процесс CRW32.EXE продолжает нормальную работу, а программа E140_DRV.PAS периодически делает попытки перезапуска драйвера. Если восстановить USB соединение, драйвер перезагружается успешно и продолжает работу в прежнем режиме.

Данный пример показывает, как можно добиться приемлемого уровня скорости, отказоустойчивости и адаптивности, сочетая лучшие стороны различных методов и технологий и соблюдая изложенные выше базовые принципы построения таких систем.

Список использованных источников:

1. Ю.И. Виноградов, В.С. Арюткин, **А.В. Курякин** и др. Автоматизированная система контроля и управления комплексом подготовки газовой смеси для экспериментального исследования мюонного катализа ядерных реакций синтеза // Приборы и техника эксперимента, 2004, № 3. С.29–41.
2. Ю.И. Виноградов, В.С. Арюткин, **А.В. Курякин** и др. Система контроля и управления комплекса тритиевой мишени для исследования экзотических нейтронно-избыточных ядер. // ВАНТ, Серия: Физика ядерных реакторов, вып. ½, 2002, С.197-200.
3. Й. Хердер, Х. Бос, Э.Таненбаум. Построение надежных операционных систем, допускающих наличие ненадежных драйверов устройств. // http://citforum.ru/operating_systems/reliable_os/ , <http://www.minix3.org/doc/reliable-os.pdf>.
4. Joe Armstrong. Making reliable distributed systems in the presence of software errors. // A Dissertation of Doctor of Technology, The Royal Institute of Technology Stockholm, Sweden, December 2003. www.sics.se/~joe/thesis/.
5. **А.В. Курякин**, Ю.И. Виноградов. Программное обеспечение автоматизированных измерительных систем в области тритиевых технологий. // ВАНТ, серия «Термоядерный синтез», 2008 г., выпуск 2, стр. 80-90.
6. **А.В. Курякин**, Ю.И. Виноградов. Программа для автоматизации физических измерений и экспериментальных установок (CRW-DAQ). // Свидетельство РФ об официальной регистрации программы для ЭВМ № 2006612848 от 10.08.2006 г. Домашний сайт www.crw-daq.ru.
7. <http://www.lcard.ru/products/external/e-140m>.