НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЕ УЧРЕЖДЕНИЕ
ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ
им. Г.И. Будкера СО РАН

Т.В. Саликова

ПОРТИРОВАНИЕ EPICS НА ПЛАТФОРМУ ОПЕРАЦИОННОЙ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ LynxOS

ИЯФ 2004-10

Новосибирск
2004

Портирование EPICS на платформу операционной системы реального времени LynxOS

Т.В. Саликова

Институт ядерной физики им. Г.И. Будкера
630090, Новосибирск, Россия

Аннотация

В статье описаны концепции и механизмы использованные для портрования EPICS на платформу новой операционной системы UNIX. При полной сохранность архитектуры EPICS новые дополнительные функциональные возможности обеспечивают поддержку операционной системы жесткого реального времени LynxOS/x86 и оборудования, разработанного в нашем институте. Применение портированного EPICS позволяет снизить стоимость аппаратных и программных средств, используемых для автоматизации ЛСЭ.

Porting EPICS to real time operating system LynxOS

T.V. Salikova

Budker Institute of Nuclear Physics
630090, Novosibirsk, Russia

Abstract

This article describes concepts and mechanisms used in porting of EPICS codes to platform of operating system UNIX. Without destruction of EPICS architecture, new features of EPICS provides the support for real time operating system LynxOS/x86 and equipment produced by institute. Application of ported EPICS reduces the cost of software and hardware is used for automation of FEL.

Введение

EPICS (**E**xperimental **P**hysics and **I**ndustrial **C**ontrol **S**ystem) [1] – комплекс инструментальных программных средств, предназначенных для создания модульных масштабируемых **распределенных систем управления и сбора данных.**

Создателями и ведущими разработчиками EPICS являются Аргоннская и Лос-Аламоская национальные лаборатории, проект появился в начале 90-х годов. В настоящее время сообщество EPICS объединяет более ста национальных лабораторий, университетов, ускорительных центров и промышленных корпораций в США и Канаде, Европе и Азии. Поскольку EPICS развивается коллективами, работающими на экспериментальных установках, разработчики заложили в архитектуру EPICS ряд свойств, базирующихся на принципах открытых систем, что обеспечивает возможность количественной и функциональной модернизации системы управления в процессе длительной эксплуатации экспериментальной установки. EPICS позволяет создавать расширяемые системы (scalable system), построенные по модульному принципу (modularity). Ориентация на открытые стандарты позволяет обеспечить независимость от производителей программного и аппаратного обеспечения, интегрировать разнородные программные и аппаратные объекты, решает технический аспект организации обмена данных между объектами. При выборе программных и аппаратных средств автоматизации эксперимента в условиях финансовых ограничений привлекательным свойством EPICS является, то что – это **свободное программное обеспечение (free software[1])** для членов сообщества**.**

Сообщество предоставляет исходные тексты и документацию, содержащую полное описание EPICS, а также обеспечивает техническую поддержку. Все члены сообщества участвуют в развитии EPICS и безвозмездно передают свой программный продукт в общее пользование. Сообщество EPICS имеет несколько web-site в Internet и обеспечивает телеконференции (на базе e-mail) для консультаций и обсуждения

[1] “**Free software**” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech”, not as in “free beer”.
http://www.gnu.org/phylosophy/free-sw.html

технических вопросов. Каждый год сообщество проводит рабочие совещания, которые включают тренинг. Такая стратегия сообщества обеспечивает постоянную эволюцию EPICS.

1. Архитектура EPICS

На базе EPICS строится двухуровневая распределенная система управления, структура которой приведена на рисунке 1 [9]. Такая структура распределенной системы управления позволяет оптимально использовать компьютерные ресурсы. Каждый уровень решает свою специфическую задачу, которая предъявляет определенные требования к аппаратным и программным ресурсам.

Верхний уровень – обеспечивает интерактивный интерфейс для оператора, инструментальную среду как для разработки программного обеспечения, а так же для обработки полученной информации и визуализации. На этом уровне необходимо использовать мощности графических станций. В терминах EPICS этот уровень обозначается **OPI** (**OP**erator **I**nterface).

Нижний уровень – обеспечивает управление оборудованием в режиме жесткого реального времени. Обычно для этих целей используются **встроенные компьютеры** (embedded computer), управляемые операционной системой реального времени со специализированными магистралями для сопряжения с аппаратурой. Это может быть аппаратура, выполненная на основе магистрально-модульных стандартов, или распределенная аппаратная сеть (FieldBus – полевая сеть), а также возможно и подключение аппаратуры через традиционные интерфейсы: RS232, ISA, PCI. В терминах EPICS этот уровень обозначается **IOC** (**I**nput/**O**utput **C**ontroller).

CA (Channel **A**ccess) [7] предоставляет необходимый комплекс сетевого сервиса для обеспечения надежной и корректной работы распределенной системы управления, на базе протокола TCP/IP.

В IOC [9] загружается **база данных** (**IOC database**) и стартует ряд процессов для управления ресурсами IOC, структура IOC приведена на рисунке 3. База данных [4] состоит из рекордов со сложной структурой, каждый рекорд имеет уникальное имя. А его структура позволяет описать выделенную часть оборудования и указать алгоритм управления этим технологическим узлом. Создается процесс управления объектом – определены пределы изменения управляемых параметров, осуществляется слежение за изменением параметров, контроль состояния объекта, преобразования считанных с датчика данных в инженерную единицу измерения (и обратное преобразование), и ряд других функций. Процесс

можно контролировать через механизм Channel Access. Это функциональное средство называется **process variable (PV** - переменная процесса, или технологический параметр).

Именно работа с PV как с объектом в распределенной системе управления или системе сбора данных делает EPICS современным инструментальным средством для создания систем управления.

В ранних версиях EPICS в качестве IOC использовался только VME крейт со встроенным компьютером на базе процессора Motorola 68K: MVME167, MVME162 под управлением операционной системы жесткого реального времени VxWorks [17], как показано на рисунке 1. А на OPI уровне под управлением операционных систем UNIX работали графические станции фирм: SUN, HP, DEC, Silicon Graphics Inc.

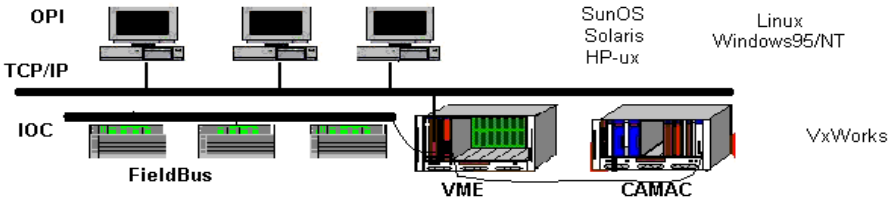


Рис. 1. Распределенная система управления на базе EPICS.

В июне 1998 года сообщество EPICS выпустило стабильную рабочую версию R3.13.0Beta12 с расширенными функциональными возможностями.

На OPI уровне можно было использовать компьютеры под управлением популярных операционных систем Windows95/NT и Linux, появилась возможность для портирования программного обеспечения OPI уровня на платформы новых типов операционных систем. Появилась новая функциональная возможность **Channel Access Portable Server**, которая поддерживает штатные операции с PV, которые не являются частью IOC. Что позволило использовать в качестве IOC персональный компьютер под управлением Windows95/NT или рабочую станцию под управлением UNIX. Был расширен список типов встроенных компьютеров, используемых на уровне IOC, практически включена поддержка всех типов процессоров, на которых работает VxWorks: PowerPC, Intel x86, Motorola 68k.

А в мае 2000 года была выпущена версия R3.14.0alpha1, поддерживающая **OSI** интерфейс (**Operating System Independent**). OSI позволяет в качестве IOC использовать не только компьютеры под управлением VxWorks, но и создавать функциональные возможности для нормальной работы IOC служб на платформе UNIX операционных систем и Windows95/NT.

1.1 Channel Access

Channel Access обеспечивает взаимодействие между программными объектами распределенной системы управления, и занимает прикладной уровень в иерархии сетевых протоколов TCP/IP. Для корректного совместного использования PV CA интерфейс должен обеспечивать **прозрачность размещения** (location transparency), что скрывает место физического нахождения ресурсов, **прозрачность доступа** (transparency access), что делает незаметным различие в представлении данных и способах доступа к ресурсам, **параллельный доступ** (concurrency transparency), что координирует многозадачный режим работы с ресурсом [15].

CA построен на базе модели клиент/сервер с использованием программного интерфейса BSD socket. На каждом IOC работает сервер, обеспечивающий неограниченное количество соединений с различными клиентами, а также и клиент может иметь неограниченное количество соединений с различными серверами. CA сервер IOC обеспечивает доступ клиентов, работающих на OPI уровне к базам данных и осуществляет синхронизацию между рекордами, локализованными в разных IOC. На рисунке 2 приведена принципиальная схема взаимодействия CA клиента и CA сервера. Для обнаружения рекорда CA клиент посылает широковещательный пакет всем станциям сети (broadcasting frame) или персональные пакеты (unicast) и ожидания подтверждения от сервера, обслуживающего рекорд с указанным именем. Если рекорд найден, тогда создается виртуальный канал, далее все операции обмена используют идентификатор этого

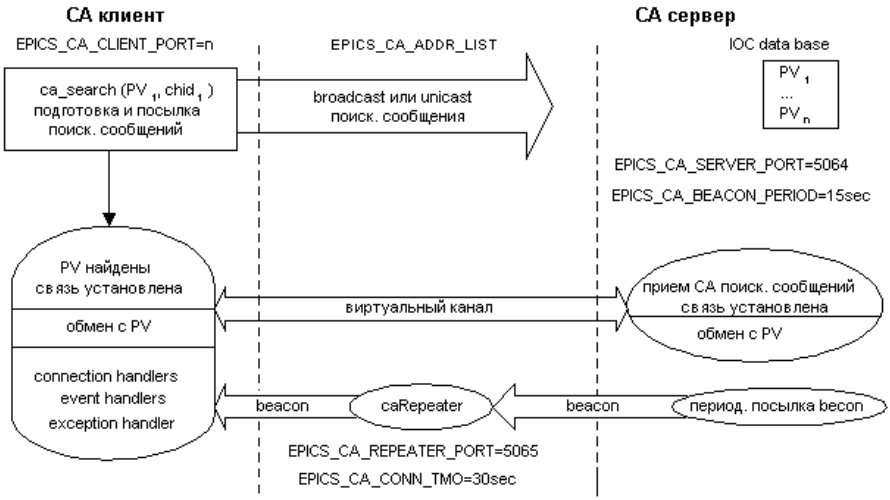


Рис. 2. Принципиальная схема взаимодействия CA клиента и CA сервера.

канала. Подобным способом CA клиенты определяют, какие сервера имеют связь или потеряли связь, периодически получая сигнальные пакеты (beacon) от серверов. Для нормальной работы CA требуется сетевое оборудование, поддерживающие широковещательный режим передачи пакетов в локальной сети [16], для этих целей на канальном уровне обычно используется Ethernet (IEEE 802.3), а также специальная конфигурация среды CA.

Необходимо отметить, что CA предоставляет удобный и простой интерфейс для создания программ OPI уровня, от программиста не требуется знаний списка IP адресов и глубоких знаний протоколов TCP/IP, вся эта сложная работа выполняется интерфейсом CA. CA контролирует право доступа к рекорду, текущие состояние канала связи, преобразования данных в зависимости от типа передаваемых данных и предоставляет следующие возможности для работы с рекордами:

ca_search – операция поиска рекорд с уникальным именем в IOC базах данных.

ca_get (ca_array_get; ca_get_callback) – операция чтения данных из рекорда.

ca_put (ca_array_put; ca_put_callback) – операция записи данных в рекорд.

«event/connection/exception handler» – возможность создавать подпрограммы для обработки значащих событий. Значащим событием является изменение значения поля или состояния рекорда, что является результатом обработки рекорда или реакцией на внешнее событие. Значащие события помещаются в очередь и обрабатываются в порядке их поступления. Специальный обработчик событий (event handler) осуществляет текущий контроль состояния рекорда, запускается всякий раз, когда происходит «замаскированное» событие. Для мониторинга состояния виртуального канала целесообразно использовать connection handler, который обрабатывает каждое изменение состояния канала. А по умолчанию всегда работает exception handler, который отслеживает состояние связь и результат выполнения операции сервером.

CA клиент может вести обмен данными с сервером следующими способами:

Периодичный опрос (polling). Клиент посылает серверу вызов ca_get/ca_put и далее в течение указанного интервала времени ожидает исполнения вызова.

Асинхронная передача. CA клиент инициирует вызов ca_put_callback/ca_get_callback, в списке параметров указывает подпрограмму callback, и далее продолжает работу. И только когда сервер завершит выполнение указанной операции, управление будет передано указанной подпрограмме (callback).

На основе механизма обработки значащих событий построен мониторинг PV, который обслуживает три типа «замаскированных» событий:

– Изменение значения PV на величину большую, чем значение «мертвой зоны» мониторинга (deadband), заданной в поле рекорда MDEL. Если разница между текущим значением PV и предыдущим превышает значение «мертвой зоны», тогда инициируется событие.
– Изменение значения PV на величину большую, чем значение «мертвой зоны» архиватора, заданной в поле рекорда ADEL.
– Изменение статуса аварийной сигнализации (alarm status), фиксируемое в поле рекорда STAT.
– В операционных системах VxWorks, LynxOS мониторинг PV реализован на основе механизма приоритетного прерывания[2] (preemption). Но ряд операционных систем не поддерживает приоритетное прерывание, поэтому дополнительно CA использует механизм мультиплексирования ввода/вывода [14] для контроля за поступлением значащих событий, этот способ усложняет структуру клиентской программы. Необходимо заметить, что использование мониторинга PV значительно снижает трафик в сети.

Конфигурация среды CA позволяет настроить оптимально работу CA с учетом топологии сети и особенностей работы управляющей системы, далее поясняется роль наиболее важных переменных CA среды. При инициализации CA формируется список из IP адресов, используемых для нахождения имен рекордов и рассылки beacon серверами. CA сервер периодично с интервалом, определенным значением **EPICS_CA_BEACON_PERIOD** посылает сигнальные пакеты beacon для контроля состояния связей с CA клиентами. Список IP адресов учитывает все сетевые интерфейсы, подключенные к компьютеру, т.к. каждый интерфейс обслуживает свою подсеть, и на их основе формируется список. Следующие переменные CA среды позволяют определить способ составления списка IP адресов:

EPICS_CA_AUTO_ADDR_LIST – выбирает механизм определения списка IP адресов. Если значение этой переменной установить равным «YES», тогда операций search/beacon будут использовать широковещательные передачи (broadcast). А если значение равно «NO», тогда операций search/beacon выполняют персональные передачи (unicast) по адресам из списка **EPICS_CA_ADDR_LIST**, который содержит список IP адресов[3].

---

[2] Процесс с более высоким приоритетом захватывает центральный процессор, и вытеснят процесс с низким приоритетом.

[3] В версии R3.14 в списке EPICS_CA_ADDR_LIST могут использоваться не только IP адреса, но и имена узлов. А также возможно и определение номера порта CA сервера, работающего на указанном узле, например: EPICS_CA_ADDR_LIST "193.168.3.100 193.168.3.101:5564".

EPICS_CA_SERVER_PORT – номер порта CA сервера, по умолчанию 5064 порт. Обычно CA клиент получает при инициализации «случайный» номер порта в интервале от 1024 до 5000.

EPICS_CA_REPEATER_PORT – номер порта **caRepeater**, по умолчанию 5065 порт. **caRepeater** прослушивает beacon пакеты в локальной сети и «повторяет» полученные UDP сообщения всем CA клиентам, работающим на том же самом узле.

В Release 3.13 поддерживается новая функциональная возможность – **Portable Channel Access Server** (PCAS). В предыдущих версиях CA позволял создавать виртуальный канал только между CA клиентом и CA сервером, работающем в IOC и обеспечивающим доступ к выбранному рекорду базы данных. А PCAS обеспечивает доступ к данным, не принадлежащим базе данных IOC, данные могут принадлежать управляющим программам, работающим в среде UNIX или в Windows96/NT. Например, PCAS используется в качестве интерфейса между клиентом и управляющей программой, созданной на основе LabView. Причиной создания PCAS явилась необходимость интеграции в системы управления новых относительно недорогих но «закрытых», патентованных аппаратных и программных продуктов.

Семейство протоколов TCP/IP постоянно развивается [16], а ранние версии CA были ориентированы на работу с 4.3BSD системой сокетов, что привело к ошибкам при работе со структурами *ifconf* и *ifreq*. Автором в тексты CA были внесены коррективы для поддержки 4.4BSD, и была введена опции – **DBSD44** для условной компиляции. Сообщение об внесенных коррективах было направлено в сообщество EPICS, и в конце 1999 года вышла новая R3.13.2 версия с поддержкой 4.4BSD.

Необходимо сделать замечания относительно механизмов реализации прозрачности CA, которые увеличивают трафик в сети. Протоколы TCP/IP обеспечивают прозрачность доступа (access transparency), что делает незаметным различие в представлении данных и способах доступа к ресурсам. Поэтому лишней операцией является конверсия передаваемых данных, которую выполняет CA сервер. Согласно описанию вызова:

*int ca_search(char *Name_PV, chid *Channel_ID) ,*

который устанавливает виртуальный канал между клиентом и сервером для обслуживания PV с указанным именем. Интерфейс CA определяет значения элементов идентификатора канала в соответствии с параметрами рекорда: тип и размер передаваемых данных. Если в вызове

*int ca_put(chtype TYPE, chid Channel_ID, void *PV_Address) ,*

указанный тип не соответствует реальному типу PV, попытка записи строки «off» в поле типа представления числа с плавающей запятой будет воспринята как ошибка, а запись строки «3.3» пройдет без сбоя. Когда сервер обнаруживает сбой, обработчик особых событий (event handler) передаст

сообщение клиенту. Аналогично будет выполнена и передача значения PV, выходящего за пределы указанные в рекорде. Этот пример демонстрирует излишнюю загруженность CA сервера и канала связи.

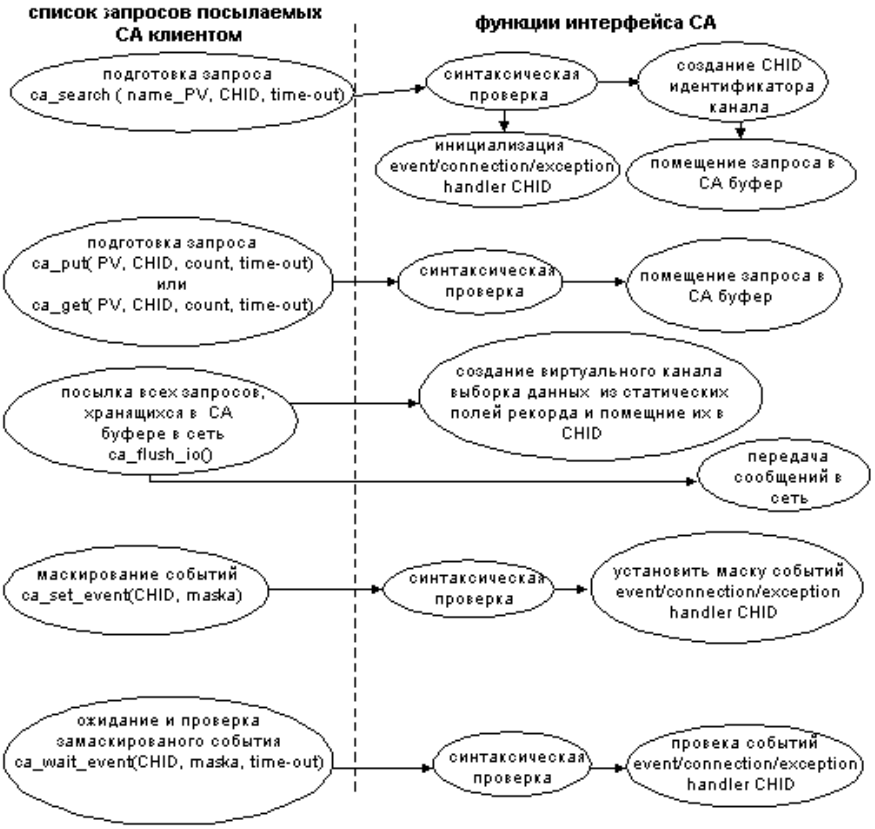


Рис. 3. Схема модифицированного CA.

Рационально передать интерфейсу CA на уровне клиента функцию проверки параметров вызова и включить часть полей рекорда в структуру идентификатора канала, тогда проверку соответствия параметров вызова можно выполнять до посылки вызова. Небольшая модификации значительно оптимизирует работу CA, схема модификации приведена на рисунке 3, а детальное описание находится в «Приложении Б". Рационально выполнять следующий алгоритм: вызов *ca_search()* создает виртуальный канал и расширенную структуру идентификатора канала, дополнительные элементы которой включали бы часть полей рекорда. В настоящих версиях CA некоторую часть полей рекорда можно прочитать, используя особые

аргументы вызова *ca_get()*. Обычно рекорд содержит небольшое число полей от 40 до 130, большая часть из них необходима только для работы внутренних служб IOC, в унифицированную структуру идентификатора канала необходимо дополнительно было бы включить 15 полей общих для различных типов рекордов. Тогда при формировании вызовов можно провести проверку соответствия указанных аргументов и параметров до посылки сообщения в сеть. Если поверка прошла успешно и есть соединение на линии связи, только тогда можно было бы выполнить передачу.

Выделение асинхронных вызовов *ca_put_callback()* и *ca_get_callback()* в большей степени отражает внутренние механизмы обработки рекордов, время исполнения посланного вызова всегда неопределенно. Логично было бы немного изменить формат вызовов, в список аргументов добавить таймаут, который бы отсчитывал время между посылкой вызова в сеть и получением ответа. При формировании вызова поиска PV достаточно инициировать один обработчик событий для всех событий, необходимо было сделать обработчик событий внутренним частью интерфейса CA, а в программе клиента предоставить возможность для маскирования событий.

Модификация CA была выполнена, и успешно прошли тесты, но проблема совместимости со старыми версиями интерфейса CA полностью не решена.

1.2 Операторский интерфейс

OPI (OPerator Interface) – это графические станции под управлением операционных систем типа UNIX или персональные компьютеры с Windows95/NT. Основными функциями OPI уровня является обеспечение удобного человеко-машинного интерфейса (HMI/MMI - human/man machine interface), предоставление инструментальной среды для создания статических баз данных и подготовки управляющих программ, а также визуализация, хранение и обработка собранной информации. Сообществом EPICS создано множество программных продуктов, работающих на OPI уровне, из которых необходимо выделить удобные в использовании графические редакторы: **MEDM** и **EDD/DM**, позволяющие оперативно создавать и настраивать панели управляющих программ, а также редакторы статических баз данных **dct** (Data base Configuration Tool). Предлагаемый набор графических редакторов создает интегральную инструментальную среду, которая позволяет избавить пользователей от сложностей в применении базовых технологий при создании приложений. Пользователи используют операции конфигурирования и сборки, а не профессиональные знания в области программирования.

В операционных системах типа UNIX в качестве графической системы используется **X11-Window,** свободное программное обеспечение ассоциации X-Consortium, в состав которой входит большинство ведущих

производителей аппаратного и программного обеспечения, что гарантирует поддержку современных моделей и видео карт и мониторов.

Необходимо отметить, что система **X11-Window** поддерживает сетевые технологии, что позволяет на удаленных узлах запускать клиенты, которые реализуют графические панели (или окна - «xterm» window). Для платформы Windows95/NT существует ряд программных пакетов, которые обеспечивают поддержку функциональных возможностей X11 сервера. На базе X11-Window создано и создается множество программных пакетов для визуализации и обработки информации, которые могут быть использованы как дополнительные инструментальные средства на OPI уровне. Также богатый выбор графических средств имеется и для платформы Windows95/NT.

1.3 Контроллер ввода/вывода

IOC (Input/Output Controller) – это VME/VXI крейт со встроенным компьютером (embedded computer) на базе процессоров типа: Motorola 68K, PowerPC, Intel x86 под управлением операционной системы жесткого реального времени VxWorks. В IOC загружается **база данных (IOC database**) и стартует ряд процессов для управления ресурсами IOC, принципиальная схема структуры IOC приведена на рисунке 4.

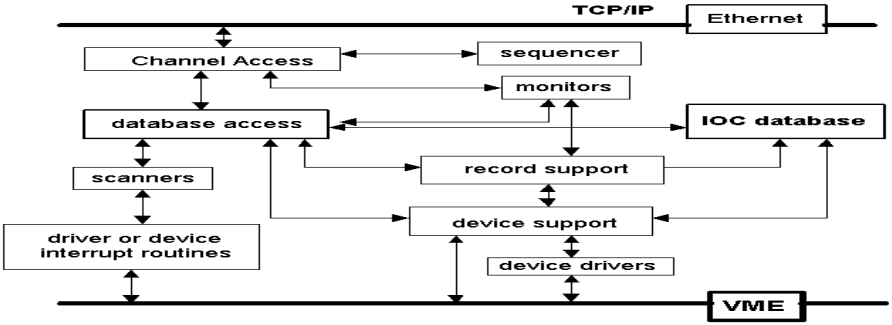


Рис. 4. Принципиальная схема структуры ОС.

Поскольку EPICS развивался на базе VxWorks [17], то некоторые концепции VxWorks были перенесены в архитектуру EPICS. Так для работы с VxWorks требуется комплекс **host–target**. **Target** (целевой компьютер) – это встроенный компьютер под управлением операционной системы жесткого реального времени VxWorks. Целевой компьютер контролирует работу оборудования, но использует выделенную периферию главного компьютера посредством сетевых служб. **Host** (главный компьютер) – это рабочая станция под управлением UNIX или персональный компьютер под управлением Window95/NT, где установлена инструментальная среда

Tornado. VxWorks имеет уникальную функциональную возможность – динамический загрузчик **ld** (dynamic linker) [18]. Динамический загрузчик позволяет в интерактивном режиме загружать объектные модули и устанавливать связи через **system symbol table**, т.е. исполняет функции компоновщика или редактора связей. Возможна и обратная операция удаление модуля из оперативной памяти.

На платформе VxWorks часть модулей IOC динамически загружается и выгружается по мере необходимости. На платформе LynxOS это функциональная возможность не реализована, поэтому в портированном варианте EPICS полностью собирается загрузочный файл **iocCore**, в котором содержаться все модули, реализующие службы IOC, алгоритм программы **iocCore** представлен на рисунке 5.

Рис. 5. Принципиальная схема алгоритма ***iocCore.***

Существуют два вида файлов, определяющих структуру различных типов рекордов и содержащие набор рабочих экземпляров рекордов:

database definition file – файл с расширением **dbd** содержит определения структур различных типов рекордов и вспомогательных структур, используемых в базе данных. Определения структур базы данных, которые используются в системе управления ЛСЭ, находятся в файле **FEL.dbd**.

record instance file – файл с расширением **db** содержит экземпляры рекордов с заданными именами и определенными рабочими полями, например, **CAMAC.db**.

При загрузке **iocCore** в качестве входных параметров определяется файлы базы данных, например:

iocCore ./dbd/FEL.dbd ./dbd/CAMAC.db.

Если при запуске **iocCore** имена файлов пропущены, по умолчанию загружаются файлы **FEL.dbd** и **CAMAC.db**.

Основой концепцией в EPICS является PV – технологический параметр, его структура и поведение описывается рекордом, на рисунке 6 указаны основные поля рекорда, которые являются общими для всех типов рекордов [13]. Существует более 40 типов рекордов, в дополнение к ним можно создать новый тип рекорда. Механизм доступа к базе данных (**database access**) обеспечивает ее нормальную работу, во время обработки рекорда осуществляет его блокировку, отслеживает таймауты, обслуживает приоритетность, выполняет преобразование данных в инженерные единицы измерения и ряд других функций.

```
char            name[29];       Record Name
char            desc[29];       Descriptor
unsigned short  scan;           Scan Mechanism
short           phas;           Scan Phase
short           evnt;           Event Number
unsigned short  dtyp;           Device Type   (канал ввода/вывода)
unsigned short  stat;           Alarm Status
unsigned short  sevr;           Alarm Severity
unsigned char   pact;           Record active
struct rset     *rset;          Address of RSET
struct dset     *dset;          DSET address
void            *dpvt           Device Private  (struct callback *)
unsigned short  prio;           Scheduling Priority
unsigned char   udf;            Undefined
TS_STAMP        time;           Time
DBLINK          flnk;           Forward Process Link
double          val;            Current EGU Value
unsigned short  linr;           Linearization
float           eguf;           Engineer Units Full
float           egul;           Engineer Units Low
char            egu[16];        Engineering Units
float           aoff;           Adjustment Offset
float           aslo;           Adjustment Slope
float           high;           High Alarm Limit
float           low;            Low Alarm Limit
unsigned short  hsv;            High Severity
unsigned short  lsv;            Low Severity
double          mdel;           Monitor Deadband
void            *pbrk;          Ptrto brkTable
long            rval;           Current Raw Value

DBLINK          inp;   (ai) Input Specification     DBLINK      out;   (ao) Output Specification
```

Рис. 6. Основные общие поля рекордов.

Рекорд имеет уникальное имя, указанное в поле рекорда **NAME.**

Поля **STAT**, **SERV**, **UDF**, **HIGH**, **LOW** (всего 16 полей) обслуживают механизм аварийной сигнализации, что обеспечивает слежение за пределами изменения технологического параметра и за корректностью процесса передачи данных.

Поле **LINR** определяет способ преобразования данных, полученных с датчика ADC в инженерную единицу измерения, указанную в поле **EGU.** Возможно передача исходных данных и без преобразования. Преобразование

может быть выполнено посредством таблицы, адрес таблицы указан в поле **BKPT.** Для линейного преобразования используются значения полей **EGUF** и **EGUL**:

$$f[engineering\_unit] = \frac{measured\_ADC}{full\_scale\_ADC}(EGUF - EGUL) + EGUL\ .$$

Аналогично выполняется и обратное преобразование из физической величины в код.

Поле **MDEL** определяет максимально допустимое значение (deadband) на которое может изменяться величина технологического параметра, чтобы запустился механизм уведомления OPI клиентов о том, что значение PV изменилось (monitor).

Каждый рекорд обслуживается своим собственным наборам подпрограмм поддержки рекорда (**record support routines**), адреса которых перечислены в структуре **RSET** (record support entry table). А взаимодействие с аппаратурой осуществляется посредством набора подпрограмм поддержки устройств (**device support routines**), адреса которых перечислены в структуре **DSET** (**device support entry table**). Подпрограмм поддержки устройств позволяют реализовать алгоритм управления устройством с учетом специфических требований, предъявляемых к работе этой аппаратурой. Набор подпрограмм поддержки устройств можно разделить на два класса: **синхронный и асинхронный**, что определяется только временем, затраченным на выполнение подпрограммы. Основное назначение подпрограмм поддержки устройств - скрыть особенности работы устройства.

По направлению передачи данных рекорды разделены на входные и выходные, например: аналоговый входной тип рекорда - **ai** где **i** (input) обозначает ввод данных в программу. Аналоговый выходной тип рекорда - **ao** где **o** (output) обозначает вывод данных из программы.

Существует **пять режимов обработки рекорда,** один из которых можно указать в поле **SCAN**:

1. Периодическое сканирование – интервал времени сканирования рекомендуется выбирать в пределах от 0.1 до 10 секунд.
2. Программное событие – функциональная возможность синхронизации процессов на базе механизма событий (event).
3. Внешнее прерывание – периферийное устройство выставляет вызов на обработку прерывания (interrupt), работает только в среде VxWorks. На платформе LynxOS/x86 эта функция эмулируется с помощью асинхронного вызова драйвера.
4. Пассивное сканирование – это результат исполнения некого рекорда имеющего связь, указанную в поле **FLINK** с текущим рекордом, или вызов, поступивший из CA сервера.
5. Одноразовое сканирование – рекорд будет исполнен на стадии инициации базы данных в IOC.

Очередность обработки рекордов определяется значением поля рекорда **PHAS,** так в очереди с одинаковым периодом сканирования, указанным в поле **SCAN** вначале обслуживаются рекорды с наименьшем значением phase (phase = {0; … ; 255}). Это возможность позволяет упорядочить работу связанных рекордов.

2. Портирование EPICS на платформу LynxOS

IOC состоит из VME/VXI крейта со встроенным компьютером под управлением операционной системы реального времени VxWorks, стоимость необходимого аппаратного и программного обеспечения для работы такого комплекса весьма высока. С целью снижения финансовых расходов на приобретение аппаратного и программного обеспечения было проведено портировния EPICS на платформу персональных компьютеров с операционными системами типа UNIX. Внесенные дополнения в исходные тексты дистрибутива EPICS были выполнены с учетом требований стандартов **POSIX** (**Portable Operating System Interface**) [19], на базе которых строятся все современные операционные системы UNIX. Эти дополнения не нарушают архитектуру EPICS, они обеспечивают поддержку EPICS на платформе операционной системы реального времени LynxOS.

Так как в задачу входило снижение финансовых затрат, то в данной работе использованы персональные компьютеры на базе процессоров типа Intel x86, которые лидируют по основному маркетинговому показателю - отношение производительности процессора к цене. Была включена поддержка для разработанной в ИЯФ аппаратуры для сопряжения с объектом управления. Были использованы свободно распространяемые программные продукты фонда GNU и графическая система **X11-Window**, развиваемая консорциумом XFree86. Необходимо отметить, что для дальнейшего развития EPICS целесообразно использовать свободное программное обеспечение, поскольку оно постоянно модернизируется с целью поддержка новых аппаратных и программных технологий. Основным достоинством принципов свободного программного обеспечения является открытость продукта и свободный доступ неограниченного числа пользователей, которые тестируют продукт, обсуждают, вносят дополнения, что обеспечивает прогрессивное развитие программного продукта и отсеивание неудачных идей.

Возможны два пути включения поддержки UNIX операционных систем на уровне IOC:

1. Создание унифицированного интерфейса для служб IOC, который был бы промежуточным звеном в формировании системных вызовов (call) с учетом специфических особенностей поддерживаемых операционных систем. Это большой объем работы, который включает существенную модификацию исходных текстов EPICS, такую работу может выполнить

только сообщество под контролем ведущих разработчиков сообщества EPICS. В настоящие время сообщество EPICS выполняет этот проекта, выпущена версия 3.14, в которой заложены функциональные возможности поддержки UNIX систем на уровне IOC. Для этой цели был разработан **Operating System Independent** (**OSI**) интерфейс.

2. Второй путь портирования EPICS описан в данной работе. Был составлен список всех функциональных возможностей VxWorks, которые используются службами IOC. И был создан механизм для их реализации на базе LynxOS – виртуальный интерфейс.

В начале работ в 1998 года был использован дистрибутив EPICS версии Release 3.13.0.12.beta12, но в 2000 году был выполнен переход на стабильную версию Release 3.13.3.

Выбор операционной системы жесткого реального времени LynxOS 3.0.0 был определен ее ведущей позицией на рынке операционных систем реального времени. Также важное значение имеет факт широкого применения LynxOS в системах управления экспериментальных комплексов CERN.

LynxOS является сертифицированной операционной системой жесткого реального времени, которая обеспечивает совместимость со стандартами UNIX и POSIX. LynxOS обеспечивает многозадачность (multitasking) и POSIX multithreading, различные механизмы планирования, механизмы межзадачной синхронизации и коммуникации (Inter-Process Communication), поддержку сетевых служб на основе протоколов TCP/IP, графический интерфейс пользователя построенный на базе X11-Window и Motif, а также ряд других функциональных возможностей.

Поскольку стояла задача портирования EPICS в гомогенную среду операционной системы реального времени LynxOS/x86, где OPI и IOC являются персональными компьютерами с процессорами типа Intel x86. То первым шагом было включение поддержки для LynxOS/x86 в качестве новой платформы. Для выполнения инсталляции были подготовлены конфигурационные файлы, где определены параметры настройки окружения (environment’s variables) для OPI (host) и IOC (target), указан тип процессора и тип операционной системы, определены опции для компиляторов, компоновщиков и ряд других параметров для учета функциональных особенностей LynxOS, что подробно описано в «Приложении А”.

Способ создания поддержки для UNIX операционных систем на уровне IOC.

Был составлен список необходимых системных вызовов (call) и функций (function) VxWorks, используемые службами IOC. Был создан виртуальный интерфейс, включающий библиотеку подпрограмм **libIOC.** Подпрограммами **libIOC** имеют внешние спецификации соответствующие системным вызовам VxWorks.

А также была создана вспомогательная библиотека подпрограмм **libRoutine** для нестандартных C/C++ функций, используемых только в VxWorks.

В VxWorks работает интерпретатор команд (shell), позволяющий вызывать некоторые подпрограммы IOC, которые информируют о состоянии объектов IOC. Аналогичные функции исполняет специально написанная подпрограмма **shell-thread**, которая является отдельным *thread*. Подпрограмма **shell-thread** предоставляет меню, в котором перечислен ряд подпрограмм, которые выводят информацию о состоянии компонентов базы данных.

Структура IOC, работающего на платформе LynxOS/x86 показана на рисунке 7. Если сравнить со стандартной структурой IOC видно, что отсутствует поддержка sequencer, driver/device interrupt routine, весь обмен с аппаратурой идет только через драйвер.

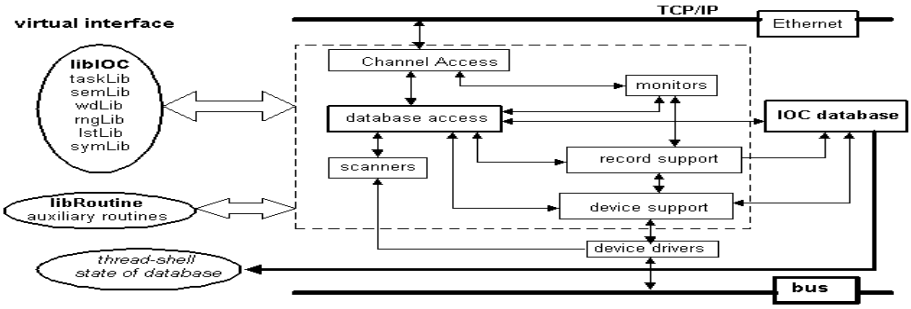


Рис. 7. Структура модифицированного IOC.

2.1 Поддержка аппаратуры, подключенной к IOC

Часть подпрограмм для работы со статистической базой данных являются общими как для **database access** служб, так и для редакторов баз данных **dct** (database configuration tool).

Необходимо отметить монолитность программного блока, реализующего **database access** службы, в блок включена поддержка ограниченного набора типов каналов ввода/вывода (Рис. 8), часть из них предназначены для обслуживания устройств, выполненных в популярных стандартах сопряжения с оборудованием. В ряде случаев большинство каналов ввода/вывода, указанных типов в работе IOC не используются, а включение нового типа канала весьма затруднительно. Было бы рационально организовать модульную структуру для поддержки набора типов каналов ввода/вывода, создать единую структуру для всех типов каналов ввода/вывода, и включить поддержку для конфигурации системы

сопряжения с оборудованием. Что позволило бы создавать новые типы каналов ввода/вывода для поддержки дополнительных стандартов сопряжения с оборудованием.

```
/* link types */
#define CONSTANT                0
#define PV_LINK                 1
#define VME_IO                  2
#define CAMAC_IO                3
#define AB_IO                   4
#define GPIB_IO                 5
#define BITBUS_IO               6
#define MACRO_LINK              7
#define DB_LINK                 10
#define CA_LINK                 11
#define INST_IO                 12              /* instrument */
#define BBGPIB_IO               13              /* bitbus -> gpib */
#define RF_IO                   14
#define VXI_IO                  15
#define CAMAC_PPI               20      /* support PPI_CAMAC for LynxOS IOC */

#define LINK_NTYPES 15
```

Рис. 8. Список типов каналов ввода/вывода.

Для работы с абстрактным типом каналов ввода/вывода был подготовлен программный модуля для обработки структуры каналов ввода/вывода. Модуль был включен в программный блок для обслуживания статистической базы данных, и на его основе была создана поддержка канала ввода/вывода для оборудования **CamacPPI**.

По аналогии с поддержкой рекордов, обслуживающих оборудование типа VME/VXI, GPIB и т.д. была создана поддержка и для оборудования **CamacPPI**. В файле *link.h* определена структура канала ввода/вывода для оборудования **CamacPPI**. А в файл *dbStaticLib.c* внесена поддержка для обработки рекордов, обслуживающих оборудование **CamacPPI**.

Внесенные дополнения позволяют редакторам **dct,** которые используют библиотеку подпрограмм статической базы данных обслуживать оборудование типа **CamacPPI** и обрабатывать форму для заполнения последовательности CNAF. Файл базы данных является обычным текстовым файлом, с который может работать любой текстовый редактор, но рационально использовать редакторы **dct**, так как они предоставляют форму для заполнения полей рекордов, выполняют синтаксический анализ структуры рекорда и устанавливают по умолчанию значения некоторых полей рекорда.

В поле **DTYP** (device type) рекорда определяется тип оборудования **CamacPPI** и подпрограмма поддержки устройства. В поле **INP/OUT** (input/output link) рекорда находятся параметры для канала ввода/вывода - цепочка NAF. Структура канала ввода/вывода для оборудования **CamacPPI** определена как *struct camacPPI*, представленная на рисунке 9.

Набор подпрограмм поддержки устройств позволяет скрыть специфические детали работы с устройством и обеспечить унифицированный интерфейс для служб IOC, более подробное описание

приведено в «Приложении В». Использование функциональных возможностей виртуального интерфейса позволяет в качестве подпрограммы поддержки устройства порождать *thread* – постоянно активный процесс, контролирующий работу технологического объекта.

```
struct camacPPI {
short b;      short c;      short w16_24;     short allNAFs;
short n1;     short a1;     short f1;             u_int data1;      int ticks1;     short Qwait1;
short n2;     short a2;     short f2;             u_int data2;      int ticks2;     short Qwait2;
short n3;     short a3;     short f3;             u_int data3;      int ticks3;     short Qwait3;
short n4;     short a4;     short f4;             u_int data4;      int ticks4;     short Qwait4;
short n5;     short a5;     short f5;             u_int data5;      int ticks5;     short Qwait5;
short n6;     short a6;     short f6;             u_int data6;      int ticks6;     short Qwait6;
short n7;     short a7;     short f7;             u_int data7;      int ticks7;     short Qwait7;
short n8;     short a8;     short f8;             u_int data8;      int ticks8;     short Qwait8;
short n9;     short a9;     short f9;             u_int data9;      int ticks9;     short Qwait9;
char      *parm; };

short b            - branch, определяет номер ППИ-6 интерфейса: 0 - PPI0, 1 - PPI1.
short c             -  crate, номер крейта в пределах от 0 до 5.
short w16_24   - определение размер передаваемого слова в байтах.
short allNAFs   - число исполняемых NAF при обработке рекорда.
short n_i          - номер CAMAC модуля в крейте изменяется от 1 до 23.
short a_i          - субадрес меняет значение от 0 до 15.
short f_i          - код функции в пределах от 0 до 31. Если код функции положительный,
                     данные читаются из модуля, если отрицательный, данные записываются.
u_int data_i      - передаваемые данные.
int ticks_i       - задает время ожидания Q. квант равен  10μsec.
short Qwait_i   - определяет счетчик цикла ожидания Q   .

char *parm     - последовательность NAF, исполняемая при инициализации рекорда.
Имеет следующий формат:
                    @i n1 a1 f1 0xData1 ... ni ai fi 0xDatai
i - определяет число NAF
```

Рис. 9. Структура канала ввода/вывода для оборудования ***CamacPPI.***

2.2 Виртуальный интерфейс системы управления задачами

VxWorks операционная система жесткого реального времени, поддерживающая многозадачность (multitasking) на базе следующих функциональных возможностей:

а) планировщик обеспечивает приоритетное прерывание обслуживания (preemptive priority scheduling),
б) механизмы межзадачной синхронизации и коммуникации (Inter-Process Communication),
в) обработку аппаратных и программных прерываний (interrupt handling).
г) сторожевой таймер (watchdog timer),
д) управление памятью (memory management).

Задача может находиться всего в четырех состояния, указанных на рисунке 10. Механизм многозадачности VxWorks можно реализовать с помощью модели **POSIX multithread** [6; 10; 12]**,** которая позволяет создать внутри программы или процесса несколько конкурирующих автономных частей – *thread*, обладающими теми же функциональными возможностями, что и процессы.

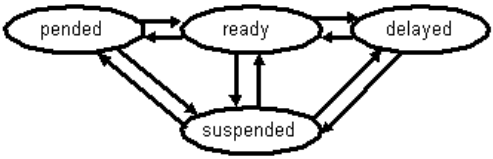
Thread выполняет роль VxWorks *задачи*, каждая *задача* имеет **TCB** (Task Control Block), где хранится контекст *задачи*. Структура *struct TaskName_ThreadID* является аналогом **TCB**, где хранится имя *задачи* (*thread*), идентификатор *thread*, атрибуты *thread.*

```
struct TaskName_ThreadID    {
char              name[Length_NameThread];
pthread_t         tid;
pthread_attr_t  t_attr;     };
```

Принципиальная схема структуры виртуального интерфейса управления задачами приведена на рисунке 11. Имеется ядро интерфейса **taskLib,** контролирующее работу создаваемых *thread*. И библиотека подпрограмм выполняющих идентичные действия, что и стандартные VxWorks вызовы (call) и функции (function). Ядро интерфейса **taskLib** содержит массив:

struct TaskName_ThreadID TaskName_TID[LynxOS_SYS_NTHREADS+1],

куда помещается **TCB** вновь созданного *thread*, а при удалении *thread* все элементы соответствующего **TCB** обнуляются. В виртуальном интерфейсе используются следующие обозначения: *TID* – это идентификатор *задачи* (task identification), *tid* – идентификатор *thread* (thread identification). Часть вызовов интерфейса управления задачами предназначены для изменения состояния *задачи,* список которых приведен на рисунке 10, другая часть передает параметры состояния задачи в пользовательскую программу.



состояния задач	
ready	задача готова, ожидает CPU
pended	задача заблокирована, нет доступа к неким ресурсам
delayed	задача в режиме ожидания на определенное время
suspended	задача не может выполнаться, но может переити в другое состояние

операции переходов	
ready -> pended	semTake() msgQReceive()
ready -> delayed ready -> suspended	taskDelay() tasksuspend()
pended -> ready pended -> suspended	semGive() msgQSend() taskSuspend()
delayed -> ready delayed -> suspended	expired delay taskSuspend()
suspended -> ready suspended -> pended suspended -> delayed -> suspended	taskResume() taskActivate() taskResume() taskResume() taskInit()

Рис. 10. Граф программно управляемых состояний VxWorks задачи.

$$TaskState(TCB, currentState) \xrightarrow{call} TaskState(TCB, newState)$$

$$TaskState(TCB, currentState) \xrightarrow{call} TaskState(TCB, currentState); parameter_K(TCB)$$

Ниже приведено описания базового набора подпрограмм интерфейса управления задачами.

*int TID = taskSpawn(char *name, int priority, int options, int stackSize,*

FUNCPTR entryPt, int arg1, … int arg10)

Подпрограмма создает и активизирует новую задачу с указанными параметрами. В виртуальном интерфейсе этот вызов обслуживает подпрограмма, которая находит свободный **TCB** в буфере *TaskName_TID, с*оздает атрибуты *thread* посредством функции *pthread_attr_create(),* устанавливает приоритет и стек *thread,* используя функции *pthread_attr_setprio()* и *pthread_attr_setstacksize().* Создает и активизирует *thread tid=pthread_create(),* заносит параметры *thread* в **TCB**, как показано на рисунке 11.

$$TaskState(0,0) \xrightarrow{\text{taskSpawn}} TaskState(TCB, ready)$$

STATUS taskDelay (int ticks).

Задача, выполнившая этот вызов, освобождает процессор на указанное время в тиках.

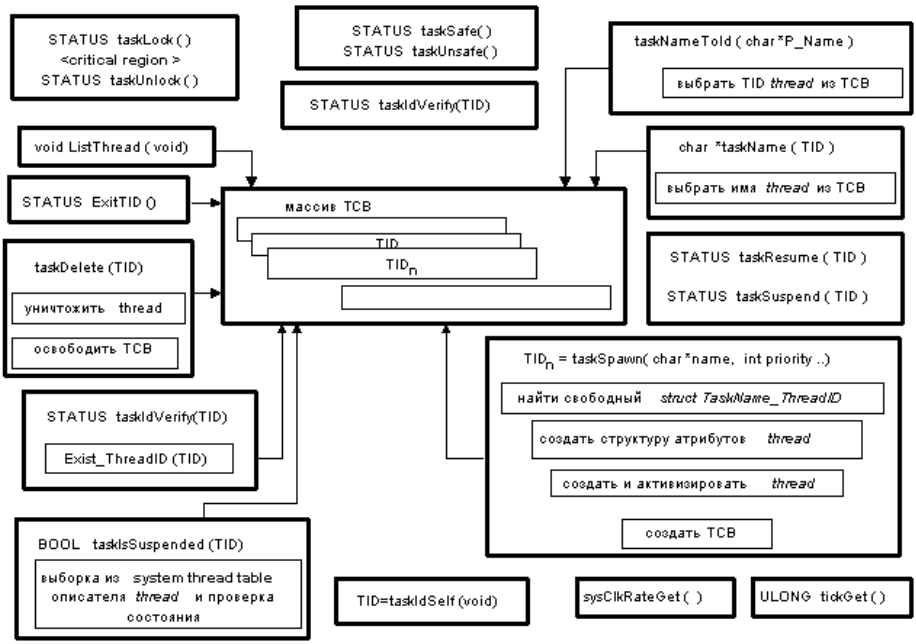


Рис. 11. Интерфейс управления задачами.

В виртуальном интерфейсе эту операцию выполняет вызов *susleep(seconds, useconds)*.

void taskDelayMicroSec (long MicroSec)

Действие вызова аналогично действию вызова *taskDelay()*, только интервал времени указан в микросекундах.

$$TaskState(TCB, ready) \xrightarrow{\text{taskDelay}} TaskState(TCB, delayed)$$

STATUS taskDelete (int TID)

Задача с указанным идентификатором *TID* уничтожается.

Виртуальный интерфейс проверяет, установлен ли «*taskSafe*» флаг защиты в **TCB**, если флаг не установлен, тогда вызов *pthread_kill(tid, SIGTERM)* уничтожает *thread* с идентификатором *tid*, а процедура виртуального интерфейса обнуляет соответствующий **TCB**. Если флаг установлен, уничтожение *thread* блокируется, пока флаг защиты не будет снят.

$$TaskState(TCB, currentState) \xrightarrow{\text{taskDelete}} TaskState(0,0)$$

int TID = taskIdSelf (void)

Получить идентификатор текущей *задачи*.

В виртуальном интерфейсе эту операцию выполняет вызов *pthread_self()*.

STATUS taskIdVerify (int TID)

Вызов для проверки существования *задачи* с указанным идентификатором. Подпрограмма виртуального интерфейса выполняет проверку существования *thread* через системный вызов *pthread_getprio()*.

BOOL taskIsSuspended (int TID)

Вызов проверяет состояние *задачи* с указанным идентификатором, находится ли *задача* в состоянии **suspended.** Подпрограмма виртуального интерфейса проверяет, есть ли в буфере **TCB** *thread* с указанным идентификатором. Если идентификатор найден, далее выбирает из **system thread table** описатель *thread* и проверяет его состояние.

*char *taskName (int TID)*

Получить имя *задачи* с указанным идентификатором. Подпрограмма виртуального интерфейса выполняет проверку существования *thread* через системный вызов *pthread_getprio()*, и проводит выборку имени *thread* из **TCB**.

*int TID = taskNameToId (char *P_Name)*

Получить идентификатором для *задачи* с указанным именем. Подпрограмма виртуального интерфейса выполняет выборку идентификатора *thread* из **TCB** с указанным именем и проверяет существования *thread*.

STATUS taskLock (void)

Вызов запрещает переключение контекста, *задача* переходит в состояние **ready** и захватывает процессор.

STATUS taskUnlock (void)

Вызов разрешает переключение контекста.

Вызовы LynxOS *sdisable(ps)* и *srestore(ps)* функционально идентичны VxWorks вызовам *taskLock ()* и *taskUnlock()* но исполняются только в режиме ядра. Чтобы избежать вытеснения *thread* (preemptive priority scheduling) при его работе в критической области (critical region), виртуальный интерфейс максимально повышает приоритет *thread*.

$$TaskState(TCB, currentState) \xrightarrow{\text{taskLock}} TaskState(TCB, ready + Lock \,\text{Re}\, scheduling)$$

$$TaskState(TCB, ready) \xrightarrow{\text{taskUnlock}} TaskState(TCB, newState + Unock \,\text{Re}\, scheduling)$$

STATUS taskSafe (void)

Подпрограмма виртуального интерфейса устанавливает флаг защиты в **TCB**, который блокирует операцию уничтожения *thread*.

STATUS taskUnsafe (void)

Подпрограмма виртуального интерфейса снимает флаг защиты в **TCB**.

$$TaskState(TCB, ready) \xrightarrow{\text{taskSafe}} TaskState(TCB + \text{Safe}, ready)$$

$$TaskState(TCB + \text{Safe}, ready) \xrightarrow{\text{taskUnsafe}} TaskState(TCB, ready)$$

STATUS taskSuspend (int TID)

Перевести *задачу* в состояние **suspended**.

Виртуальный интерфейс выполняет системный вызов *pthread_ kill (tid, SIGSTOP)*, и *thread* переходит в состояние **suspended**.

STATUS taskResume (int TID)

Задача переходит в состояние **ready**. Виртуальный интерфейс выполняет системный вызов *pthread_kill(tid, SIGCONT)*, и *thread* переходит в активное состояние.

$$TaskState(TCB, currentState) \xrightarrow{\text{taskSuspend}} TaskState(TCB, suspened)$$

$$TaskState(TCB, suspended) \xrightarrow{\text{taskResume}} TaskState(TCB, ready)$$

STATUS ExitTID (void)

Подпрограмма виртуального интерфейса, предварительно проверив **TCB,** завешает работу текущего *thread* с помощью вызова *pthread_cancel(pthread_self())* освобождает используемые *thread* ресурсы.

int Exist_ThreadID (int TID)

Подпрограмма виртуального интерфейса проверяет, есть ли в буфере **TCB** *thread* с указанным идентификатором. Если идентификатор найден, далее выбирает из **system thread table** описатель *thread* и проверяет его состояние.

void ListThread(void)

Подпрограмма виртуального интерфейса выводит информацию о состоянии всех существующих *thread*, список которых представлен на рисунке 12.

int sysClkRateGet (void)

Вызов для определения системной характеристика - число тиков в секунде. Виртуальный интерфейс использует стандартный POSIX вызов *sysconf(_SC_CLK_TCK).*

ULONG tickGet (void)

Вызов для выборки текущего значения “счетчика тиков”, при загрузки VxWorks счетчик обнуляется. В виртуальном интерфейсе “счетчик тиков” начинает отсчет с 00:00:00 текущего дня.

```
tid=55____name=taskwd
tid=41____name=cbLow
tid=40____name=cbMedium
tid=61____name=cbHigh
tid=33____name=dbCaLink
tid=37____name=scanOnce
tid=43____name=scan1000
tid=42____name=scan500
tid=45____name=scan300
tid=46____name=scan200
tid=47____name=scan100
tid=48____name=scan50
tid=49____name=scan20
tid=50____name=scan10
tid=51____name=EV dbCaLink
tid=52____name=CA_TCP
tid=53____name=CA_UDP
tid=54____name=CA_online
tid=60____name=RD_dbCaLink
tid=56____name=CA_client
tid=64____name=CA_event
```

Рис. 12. Список активных задач IOC.

*void remCurIdGet (char *user, char *passwd)*

Подпрограмма из библиотеки удаленных командных функций *remLib.h*. Подпрограмма получает имя и пароль пользователя, который в настоящий момент имеет привилегированный доступ к удаленному главному компьютеру (host). В виртуальном интерфейсе эти данные можно получить с помощью вызова *char *cuserid(char *s),* реализованного в стандарте POSIX.1-1988 или вызова *char *getlogin(void),* реализованного в POSIX.1-2001.

VxWorks поддерживает кроме стандартного ANSI C механизма контроля статуса завершения вызовов и подпрограмм – **errno**, еще и **errnoLib** библиотеку контроля статус ошибок в текущей *задаче* или в указанной *задаче*. В виртуальном интерфейсе для текущей *задачи* статус ошибки всегда равен «OK». А для *задачи* с указанном TID проверяется существование *thread*, если проверка прошла успешно в статусе ошибки устанавливается код «OK», иначе статус ошибки равен «ERROR».

2.3 Семафоры

VxWorks обеспечивает мощный механизм межзадачных коммуникационных возможностей, необходимый для координирования взаимодействия *задач*. Одним их этих механизмов являются семафоры, которые выполняют два типа операций:

взаимное исключение (mutual exclusion) - исключает одновременный доступ двух или более процессов к разделяемым ресурсам.

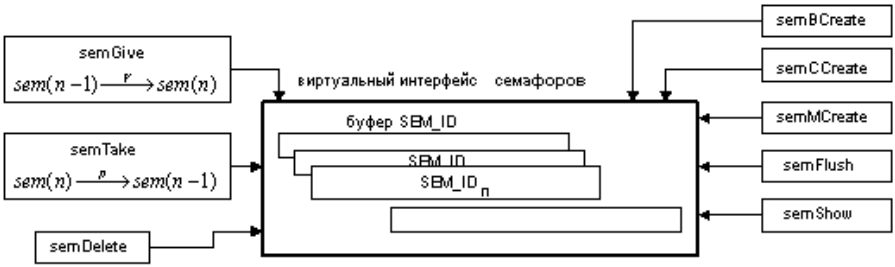
синхронизация (synchronization) - координируют выполнение задачи с внешними событиями.

VxWorks работает с четырьмя типами семафоров: POSIX именованные (named) и неименованные (unnamed) семафоры, бинарные (binary), взаимоисключающие семафоры (mutual exclusion), семафоры со счетчиком (counting). Службы IOC используют все четыре типа семафоров. Интерфейс POSIX именованных и неименованных семафоров как для VxWorks, так и для LynxOS идентичны Совместимость тестовых программ, использующих POSIX семафоры, была проверена на платформах обеих операционных систем. А семафорные интерфейсы LynxOS и VxWorks для бинарных семафоров, взаимоисключающих семафоров и семафоров со счетчиком существенно отличается. VxWorks имеет единый интерфейс для управления всеми типами семафоров, каждый созданный семафор имеет идентификатор, где указан тип семафора. LynxOS для каждого типа семафора резервирует отдельные ресурсы, различны и механизмы управления разными типами семафоров. VxWorks для проверки и декрементирования значение семафора в вызове *semTake (SEM_ID semId, int timeout)* использует тайм-аут, что

позволяет ограничивать время ожидания семафора и избежать тупиковых ситуаций, которые возможны при обращении к разделяемым ресурсам. На платформе LynxOS на базе семафоров со счетчиком можно реализовать подобную модель семафоров, удовлетворяющую спецификациям VxWorks интерфейса семафоров.

Каждый VxWorks семафор имеет идентификатор **SEM_ID**, который в виртуальном интерфейсе заменен аналогичной структурой *struct SEMAPHORE*. Ядро виртуального интерфейса семафоров содержит массив идентификаторов **SEM_ID** и контролирует операции изменения состояния семафоров (Рис 13). Виртуальный интерфейс помещает *thread* в очередь к семафору в соответствии с приоритетом *thread*. В виртуальном интерфейсе семафоров используется вызов *csem_t csem_create_val(int initial_value)* для создания пользовательских семафоров со счетчиком. При создании семафора

```
typedef struct SEMAPHORE
{
csem_t          sem_id;         csem_t
unsigned short  semType;        semaphore type
unsigned short  priority;       semaphore priority (default/FIFO)
unsigned long   QueueHead;      blocked task queue head
unsigned int    count;          counter is equal to csem_t->w_count
                                OR semaphore state (SEM_FULL/SEM_EMPTY)
unsigned int    tid_owner;      TID of thread that is spawned this semaphore
unsigned int    ticks_timeout;  timeout in ticks
struct timeval  sec_timeout;    timeout in sec & μsec
};
```



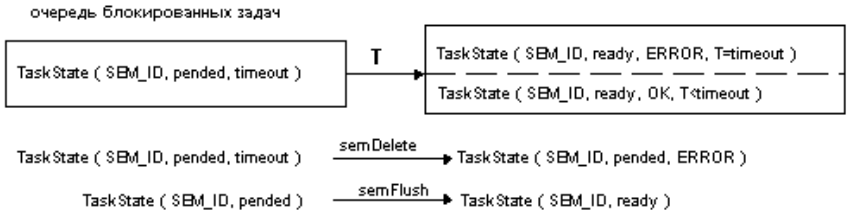


Рис. 13. Виртуальный интерфейс семафоров.

определяется свободная структура *struct SEMAPHORE* в блоке идентификаторов семафоров **SEM_ID**, проводиться инициирование элементы в соответствии с типом создаваемого семафора и указанными опциями.

SEM_ID semBCreate (int options, SEM_B_STATE initialState)

Вызов создает и инициирует бинарный семафор, который имеет два состояния: **SEM_EMPTY** соответствует 0 (семафор недоступен), **SEM_FULL** соответствует 1 (семафор доступен).

SEM_ID semMCreate (int options)

Вызов создает и инициирует взаимоисключающий семафор, который является специализированным вариантом бинарного семафора. При создании значение семафора устанавливается равным **SEM_FULL**.

SEM_ID semCCreate (int options, int initialCount)

Вызов создает семафор со счетчиком и устанавливает начальное значение семафора.

STATUS semGive (SEM_ID semId)

Операция инкрементирует значение семафора, операция ***V*** (verbogen) по определению Дейкстра.

$$sem(n-1) \xrightarrow{V} sem(n+1)$$

В LynxOS аналогичную функцию выполняет вызов *csem_signal(csem_t sd)*. Для бинарных и взаимоисключающих семафоров вызов снимает блокировку с первого в очереди ожидающего *thread*, счетчик семафора присваивается значение **SEM_FULL**.

$$TaskState(TCB_{queue1}, (sem = 0), pended) \xrightarrow{\text{semGive}} TaskState(TCB, (sem = 1), ready)$$

STATUS semTake (SEM_ID semId, int timeout)

Операция декрементирует значение семафора, операция ***P*** (proberen) по определению Дейкстра.

$$sem(n) \xrightarrow{P} sem(n-1)$$

В виртуальном интерфейсе семафоров эту функцию выполняет вызов *csem_wait(csem_t sd, struct timeval *timeout)*. Для бинарных и взаимоисключающих семафоров вызов устанавливает значение семафора **SEM_EMPTY**, и блокирует *thread*, если значение семафора было равно **SEM_EMPTY.**

$$TaskState(TCB, (sem = 1), ready) \xrightarrow{\text{semTake}} TaskState(TCB, (sem = 0), ready)$$

$$TaskState(TCB, (sem = 0), ready) \xrightarrow{\text{semTake}} TaskState(TCB, (sem = 0), pended, timeout)$$

Блокированные *thread* в очереди ожидают перехода семафора в состояние **SEM_FULL**, если в течение таймаута *thread* не дождался семафора, *thread* будет разблокирован, а в программу будет передана ошибка завершения вызова по таймауту.

Для семафоров со счетчиком запрос декрементирует счетчик, и если значение счетчика больше нуля, *thread* не блокируется.

$$TaskState(TCB,(sem=n),ready) \xrightarrow{\text{semTake}} TaskState(TCB,(sem\le 0),pended,timeout)$$

$$TaskState(TCB,(sem=n),ready) \xrightarrow{\text{semTake}} TaskState(TCB,(sem\ge 0),ready)$$

STATUS semDelete (SEM_ID semId)

В виртуальном интерфейсе запрос *csem_delete(csem_t sd)* удаляет указанный семафор из списка активных и освобождает ресурсы. Все *thread,* ожидающие семафор будет разблокированы.

STATUS semFlush (SEM_ID semId)

Состояние семафора не изменяется, но все *thread,* ожидающие этот семафор будет разблокированы.

STATUS semShow (SEM_ID semId, int level)

Запрос позволяет вывести информацию о состоянии указанного семафора.

void semShow_ALL (void)

Вывод информации о состоянии всех семафоров, с которыми работает виртуальный интерфейс.

2.4 Сторожевой таймер

Ряд служб IOC используют сторожевой таймер (watchdog timer). Если большинство встроенных компьютеров имеют аппаратную реализацию сторожевого таймера, то на платформе персональных компьютеров возможна только программная реализация этого механизма на базе часов реального времени (Real Time Clock) [8]. POSIX.4 поддерживает модель внутренних таймеров, в виртуальном интерфейсе на основе этой модели и реализована библиотека вызовов для сторожевого таймера **wdLib.**

WDOG_ID wdCreate (void)

Вызов создает сторожевой таймер и инициирует идентификатор **WDOG**, который в виртуальном интерфейсе заменен структурой *struct wdog.*

typedef struct wdog {
int Delay_Tick;
FUNCPTR wdRoutine;
int wdParameter;
tid_t ST_TID_wdog; } WDOG;

STATUS wdStart (WDOG_ID wdId, int delay, FUNCPTR pRoutine, int parameter)

Вызов выполняет запуск сторожевого таймера *wdId*. После истечение интервала времени *delay*, заданного в тиках будет запущена подпрограмма *pRoutine*. Виртуальный интерфейс использует *timer_create(clock_id, evp, timerid)* функцию для асинхронного уведомления об истечении указанного интервала времени, а *pRoutine* объявляется в качестве подпрограммы для обработки сигналов. Функция *timer_settime(wdId, flags, value, NULL)* разрешает работу таймера и определяет интервал времени *value.*

STATUS wdCancel(WDOG_ID wdId)

Вызов аннулирует все посылки асинхронных уведомлений об истечении указанного интервала времени, находящиеся в очереди к указанному таймеру. Виртуальный интерфейс использует функцию *timer_settime(wdId, flags, NULL, NULL)* для остановки таймера.

STATUS wdDelete(WDOG_ID wdId)

Вызов завершает работу сторожевого таймера и освобождает ресурсы. Аналогичные действия выполняет соответствующая подпрограмма виртуального интерфейса, функция *timer_delete(wdId)* удаляет внутренний таймер.

2.5 Циклические списки и связанные списки

Циклические списки и связанные списки – это стандартные модели [2], описывающие структуру данных в качестве логически связанной последовательности элементов, но на платформе VxWorks существуют специальные библиотеки подпрограмм для работы со списками.

Подпрограммы из библиотеки **rngLib,** предназначенные для работы с циклическими списками перечислены в таблице 1, а их структура представлена на рисунке 14. Запись и чтение данных из циклического буфера производится в порядке поступления – FIFO.

В VxWorks используются подпрограммы из библиотеки **lstLib** для работы с двойными связанными списками, которые перечислены в таблице 2, а структура связанных списков представлена на рисунке 14.

Таблица 1. Процедуры для работы с циклическими списками

Процедуры	Описание процедур
RING_ID rngCreate (int SizeRingBuffer)	Подпрограмма создает список, размером *SizeRingBuffer* байт. И возвращает адрес идентификатора *RING_ID.*
void rngDelete (RING_ID ringId)	Удалить циклический список
int rngNBytes(RING_ID ringId)	Определить размер списка в байтах
*int rngBufGet (RING_ID rngId, char *User_Buffer, int N_Bytes)*	Скопировать *N_Bytes* байт из списка *ringId* в список *User_Buffer*
*int rngBufPut (RING_ID rngId, char *User_Buffer, int N_Bytes)*	Скопировать *N_Bytes* байт из списка *User_Buffer* в список *ringId*

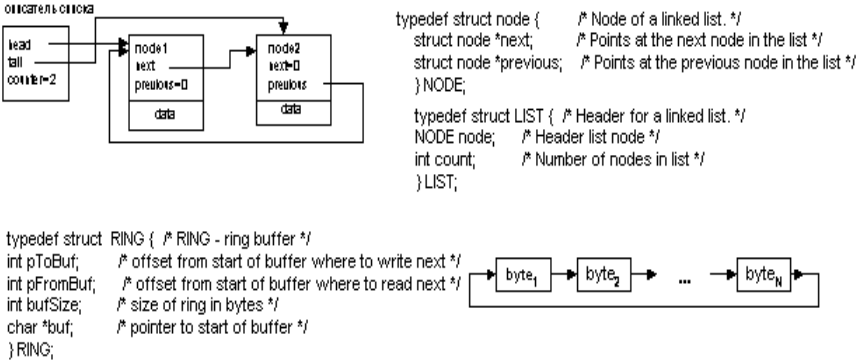


Рис. 14. Структура связанных и циклических списков.

Таблица 2. Процедуры для работы со связанными списками

Процедуры	Описание процедур
*void lstInit(LIST *pList)*	Создать описатель связанных списков - **pList*
*void lstAdd (LIST *pList, NODE *pNode)*	Добавить указатель адреса узла **pNode* в конец связанного списка
*int lstCount (LIST *pList)*	Выбрать счетчик числа узлов списка
*void lstDelete (LIST *pList, NODE *pNode)*	Удалить указатель адреса узла **pNode* из связанного списка
*void lstFree (LIST *pList)*	Освободить описатель связанных списков
*NODE *lstFirst (LIST *pList)*	Выбрать указатель адреса первого узла из списка
*void lstInsert (LIST *pList, NODE *pPrev, NODE *pNode)*	Поместить указатель адреса узла **pNode* в связанный список после указателя **pPrev*
*NODE *lstNext (NODE *pNode)*	Выбрать указатель адреса следующего за **pNode* узла
*NODE *lstLast (LIST *pList)*	Выбрать указатель адреса последнего узла из списка

При работе со списками VxWorks не обеспечивает синхронизацию доступа на уровне ядра, задачи должны самостоятельно контролировать доступ к разделяемому ресурсу.

2.6 System symbol table

VxWorks предоставляет уникальную функциональную возможность – динамически загрузить или выгрузить объектный модуль, в процессе загрузки система выделяет оперативную память и выполняет функции компоновщика (linker), определяет физические адреса для внешних ссылок посредством **system symbol table**. Этот механизм активно используется рядом служб IOC, подпрограммы из библиотеки **symLib** (symbol table subroutine library) используются для динамической загрузки модулей из библиотек: подпрограмм поддержки рекордов, подпрограмм поддержки устройств, подпрограмм поддержки драйверов. Операционные системы типа UNIX не имеют такой функциональной возможности, компоновщик строит загрузочный файл из объектных модулей и библиотек, полученный загрузочный файл нельзя модифицировать. С целью локализации адресов для внешних ссылок были написаны макрокоманды (macro substitution), которые резервируют ресурсы и создают таблицу внешних ссылок на **RSET** (Record Support Entry Table) и **DSET** (Device Support Entry Table).

Каждый тип рекорда имеет индивидуальный пакет подпрограмм поддержки рекорда и набор подпрограммы поддержки устройств, которые инициализируются при старте **iocCore**. Алгоритм инициализации служб IOC содержится в файле *src/db/iocInit.c*. Процедура инициализации поддержки рекордов *initRecSup()* размещает в памяти структуру рекорда (например, *struct aiRecord*), создает таблицу внешних ссылок на **RSET** и выполняет инициализацию рекорда, дополнительно содержит макрокоманды *pdbRecordTypeRSET("имя рекорда")*, определяющие тип каждого рекорда, включенного в базу данных IOC. Процедура *initDevSup()* размещает в памяти таблицу внешних ссылок на **DSET** и выполняет инициализацию устройства, дополнительно содержит макрокоманды *pdevSup ("подпрограмма поддержки устройства")*, перечисляющие все подпрограммы поддержки устройств. Использование библиотек подпрограмм поддержки драйверов возможно только в VxWorks, в UNIX системах работа с контроллером периферийного оборудования идет в режиме ядра через стандартный механизм - драйвер. Поэтому реализация поддержки драйверов в программе **iocCore**, которая работает в режиме пользователя, невозможна. При инициализации IOC подпрограмма *initDrvSup()* обнуляет указатель адреса списка поддерживаемых драйверов и завершает работу кодом OK.

Поскольку реализация функциональных возможностей предоставляемых библиотекой **symLib** на платформе UNIX операционных систем невозможна, для обнаружения некорректного использования вызовов **symLib** в

виртуальном интерфейсе имеются макрокоманды, подменяющие вызовы оператором:

printf("<symNameCall> %s %d \n",__FILE__, __LINE__)

Который посылает на консоль сообщение о том, в какой строке, какого файла используется вызов с указанным символьным именем.

2.7 Тестирование портированого EPICS

На базе EPICS Release 3.13.3 были выполнены тесты с целью определения производительности, предельных нагрузок и работоспособности портированого EPICS.

Программа **iocCore** в активном состоянии занимает 3.5 Мбайта, когда работают все службы IOC. На обслуживание рекорда требуется дополнительно от 0.6 до 4 Кбайт в зависимости от типа рекорда и выбранного набора подпрограмм поддержки устройств.

Для определения минимального времени выполнения операций обмена информацией между CA клиентом и базой данных IOC была подготовлена база данных, которая содержала 1000 аналоговых входных рекордов (тип **ai**), которые обслуживались синхронной подпрограммой поддержки устройств. Все рекорды имели программный канал (DTYP = “Soft Channel”), интервал сканирования 1сек. С целью определения порога предельных нагрузок для служб обработки рекордов был организован счет количества обработанных рекордов в течение 10 сек. Все рекорды обрабатывались посредством синхронной подпрограммы поддержки устройства, которая инкрементировала счетчик количества всех обработанных рекордов и текущие значение рекорда.

В качестве канала был использован приватный сегмент Ethernet, работающий на скорости V=10 Мбит/сек, во время теста к сегменту было подключено три компьютера. Один из компьютеров, не участвовал в тестировании, использовался в качестве *snooper* (инструмента слежения за сетевой магистралью).

При выбранной конфигурации канал полностью предоставлен для обмена между тестовой программой и базой данных IOC, минимальное время исполнения запроса определяется прохождением запроса через службы: интерфейсы CA клиент и CA сервера, передача данных по каналу и выборка значения PV посредством механизма database access, которые представлены на рисунке 15:

$$t_{CAclient\_IOC} = t_{CAclient} + t_{network} + t_{CAservr} + t_{DBaccess\_CA}$$

Измерения были выполнены для запросов: ca_search() – поиск PV, ca_get() – чтение данных из PV. Для проверки в качестве IOC были выбраны компьютеры с различными процессорами, средние время $T_{CAclient\_IOC}$ слабо зависит от типа процессоров (Таблица 3).

Таблица 3. Время исполнения CA операций для различных типов компьютеров.

iocCore	**Motorola 68040 33 МГц** (60Гц RTC)	
	Pentium III 550МГц	UltraSPARC I 143МГц
ca_search()	5.05мсек	7.2мсек
ca_get()	2.1мсек	2.4мсек
iocCore	**Pentium III 550 МГц** (100Гц RTC)	
	Pentium III 550МГц	UltraSPARC I 143МГц
ca_search()	2.9мсек	7.02мсек
ca_get()	0.8мсек	1.6мсек

Максимальное время обслуживания запроса каналом $t_{network}$ равно сумме времени передачи пакета λ/V и времени ожидания канала $\tau\omega=(\sigma/V)(1-1/n)^{1-n}$ [5] (n = 2 два узла, σ = 512 бит). Размер передаваемого сообщения в среднем равен 128 бит (передается значение PV – это либо число с плавающей точкой, либо длинное целое).

$$t_{network} = \lambda/V + (\sigma/V(1-1/n)^{1-n} = (128 + 512\cdot 2)/10^7 \approx 0.1 \text{ мсек}.$$

Более 95% времени обработки запроса приходится на службы CA и database access.

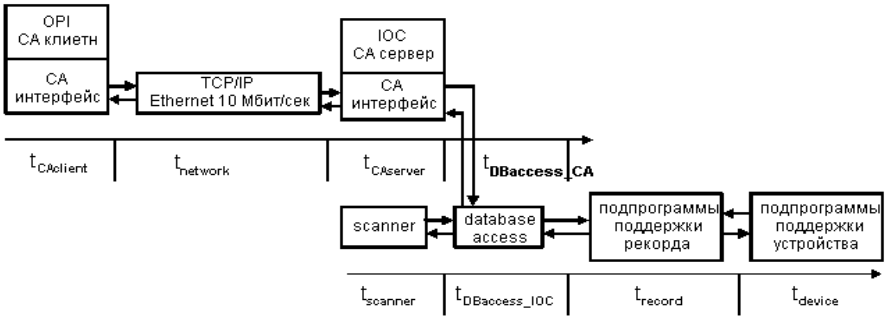


Рис. 15. Временная диаграмма обработки запросов ca_search() и ca_get().

Измерения времени обработки рекорда t_{DB_IOC} (Рис. 15) службами: *database access, record support routine, device support routine, scanner*. Изменяя количество рекордов в базе данных и интервал сканирования можно эмпирически определить производительность[4] служб базы данных IOC и фактор утилизации процессора U.

[4] По правилам “урезанной” оценки времени выполнения алгоритма O(), количество операций пропорционально числу рекордов.

$$t_{DB\_IOC} = t_{scaner} + t_{DBaccess\_IOC} + t_{record} + t_{device} \ .$$

По теореме Liu-LayLand система реального времени планируема при условии U≤1 [11]. Система обрабатывает m периодических событий, время обработки i-го события – C_i, T_i – период поступления.

$$U = \sum_i^n \frac{C_i}{T_i} \approx m \frac{\langle C \rangle}{\langle T \rangle} \ .$$

Максимальное число обработанных рекордов составляет 10200 за <T> = 1сек. IOC тратит примерно <C> = $t_{DB\_IOC} \approx 98$ мксек на один рекорд, фактор утилизации процессора U = 98·10200 мксек./1 сек ≈ 0.9996 без учета аппаратных расходов. В таблице 4 приведены зависимость количества обработанных рекордов от времени сканирования, измерения выполнены для базы данных, содержащей 1000 рекордов и работающей на платформе Pentium-III 550 МГц.

Таблица 4. Зависимость количества обработанных рекордов от периода сканирования

SCAN	1 сек	0.5 сек	0.2 сек	0.1 сек	0.05 сек
Число обработанных рекордов	1000	2000	5000	9427	9845

Невысокая скорость обработки рекордов определяется обслуживанием “временных очередей”[5], которые порождаются механизмами доступа к разделяемым ресурсам (семафоры с тайм-аутом, программные таймеры, сторожевые таймеры и т.д.). При высокой произвольности компьютера (тактовая частота процессора f_{CPU} = 550 МГц, f_{RAM} = 133 МГц) время простоя процессора составляло более 80%.

По правилам “урезанной” оценки времени выполнения алгоритма O(), количество операций пропорционально числу рекордов. Тогда время выполнения алгоритма

$O(N_{comand\_per\_record\_C/C++})$ $T = (1/f_{CPU} + 1/f_{RAM})\ (N_{comand\_per\_record\_C/C++})$
$\approx 910^{-9}(500 \div 2000) \approx 5 \div 20$ мксек .

[5] Проблема *scheduling latency*. По определению webster.com, *scheduling latency* – это временной интервал между поступлением события началом работы процесса/программы (*tread)* запускаемым этим событием. *Scheduling latency* определяется механизмами реализации что подробно описано в статье Vinayak Hegde.

Приложение А Конфигурационные файлы

Сообщество EPICS предоставляет необходимые для инсталляции исходные тексты и документацию. Инсталляция легко выполняется, указав тип процессора и тип операционной системы для платформ OPI и IOC в конфигурационных файлах можно запускать процесс инсталляции. Первое тривиальное изменение - внесение дополнений в конфигурационные файлы директории *epics/startup*, приведенные в таблице А.1, посредством которых в начальной стадии инсталляции EPICS на главном (host) компьютере будут определены новый тип операционной системы и параметры интерпретатора команд. В качестве командного процессора использован **bash-2.02**, который поддерживает стандарт POSIX-2, и является наиболее удобным и современным из подобного класса оболочек. Для выполнения сценариев (scripts) необходимо использовать программные пакеты **make-3.77** и **perl-5.003** или более поздние версий. При установке этих программных пакетов на платформу LynxOS/x86 был внесен ряд коррекцией в их дистрибутивы, ориентированных на функциональные особенности LynxOS/x86 version 3.0.0.

Таблица А.1. Параметры интерпретатора команд.

Функциональное назначение	**shell variables**
Сетевое имя OPI компьютера	HOSTNAME=fel
тип процессора OPI компьютера	HOSTTYPE=pc486
тип операционной системы на OPI компьютере	HOST_ARCH=LynxOS
тип процессора IOC компьютера	HOST_CLASS=pc486

Следующий шаг – создание в директории *epics/base/config* конфигурационных файлов, перечисленных в таблице А.2, в файлах описаны параметры настройки окружения (environment's variables) для OPI (host) и IOC (target) платформ.

Для включения коррективов в OPI программное обеспечение, учитывающих специфику LynxOS, в командной строке **gcc/g++** компилятора[6] должна быть указана опция **–DLynxOS**. Для учета структурных особенностей LynxOS при компиляции в файле **CONFIG.Host.LynxOS** нужно указать опции, зависящие от архитектуры платформы OPI:

ARCH_DEP_CFLAGS = –DBSD44 –DLynxOS –D_X86_

ARCH_DEP_LDFLAGS = –lc –lbsd –lnsl –lm –llynx

[6] Для инсталляции EPICS необходимо использовать компилятор **gcc-2.8.1** или более поздние его версии. Но дистрибутив **gcc**, который можно найти на сайте фонда GNU, предназначен для LynxOS v2.4, а для более поздних версий необходимо внести коррективы в тексты **gcc**, описывающие функции и структуру исполняющей системы. Мной был адаптирован **gcc-2.95** на платформе LynxOS/x86 v3.0.0

Таблица А.2. Конфигурационные файлы

Имена файлов	Функциональное назначение
CONFIG.Host.LynxOS	Определяет утилиты, необходимые для построения OPI. Задает опции компилятора C/C++. Определяет используемые библиотеки.
CONFIG.LynxOS	На host компьютере описывает кросс-компилятор.
CONFIG_HOST_ARCH.LynxOS	Задает дополнительную информацию для кросс-компилятора.

Для обеспечения работы всех IOC служб в режиме реального времени включены опции компилятора ***–mposix –mthreads***. Опция **–DLynxOS_pc486** необходима для компиляции коррективов, сделанных в текстах IOC программ. Опция определяет еще и тип процессора. В файле **CONFIG.lynxos** должны быть указаны следующие опции:

OP_SYS_CFLAGS =

–DLynxOS –mposix –mthreads –DLynxOS_pc486 –D_X86_ –DBSD44

OP_SYS_LDLIBS = –mposix –mthreads –lc –lbsd –lnsl –lm –lposix –llynx

Для определения порядка расположения байтов в машинном слове необходима опция **–D_X86_** (little endian – адресация байтов в порядке следования от младшего к старшему, используемая в архитектуре процессоров Intel x86), в текстах EPICS используется именно эта переменная условия компиляции. Обычно **gcc/g++** по умолчанию определяет тип процессора и его архитектурные особенности, штатной опцией является **__x86__** для процессоров Intel x86, что было не учтено разработчиками EPICS. Эта опция включает соответствующие архитектуре процессора программные сегменты, работающие с сетевыми пакетами. Формат TCP/IP заголовков пакетов имеет представление big endian (метод представление данных, когда старший значащий бит или байт стоит первым), это так называемый порядок сетевых байтов (network byte order), который описан в RFC 791 .

Опции **–DBSD44** включает компиляцию коррективов, выполненных для поддержки сокетной системе 4.4BSD в подпрограммах CA, работающие со структурами *ifconf* и *ifreq*. Что было необходимо сделать, поскольку, когда LynxOS v2.5 и последующие ее версии поддерживают 4.4BSD.

Обо всех внесенных автором дополнениях в базовое программное обеспечение EPICS было направлено ответственному разработчику CA Jeff Hill, и также были переданы и конфигурационные файлы для поддержки LynxOS. И в конце 1999 года вышла новая R3.13.2 версия EPICS с поддержкой LynxOS/x86 и 4.4BSD.

Приложение Б. Модификация CA

Внести детерминированность в операции синхронные передачи данных по виртуальному каналу можно посредством посылки отдельного сообщения для каждого отдельного PV в сеть с указанным тайм-аутом. Если тайм-аут равен “бесконечности”, запрос переходит в разряд асинхронных и обслуживается указанной подпрограммой *asyn_subroutine*. Предложенный вариант является надстройкой над существующим интерфейсом CA. Реально все запросы *ca_search, ca_get* и т.д. помещаются в буфер CA, и только по запросу *ca_flush_io ()* (или *ca_test_io (), ca_pend_io ()* и т.д.) все сообщения из CA буфера передаются в сеть.

В структуру идентификатора канала необходимо дополнительно было бы включить 15 полей общих для различных типов рекордов, которые определяются при установлении связи.

```
struct CHID_record {
   struct CHID *chid;
   char          desc[29];          /*Descriptor*/
   unsigned short      scan;      /*Scan Mechanism*/
   TS_STAMP time;      /*Time*/
    char          egu[16];/*Engineering Units*/
    float         hopr;      /*High Operating Range*/
   float         lopr;      /*Low Operating Range*/
   float         eguf;      /*Engineer Units Full*/
   float         egul;      /*Engineer Units Low*/
   float         drvh;      /*Drive High Limit*/
   float         drvl;      /*Drive Low Limit*/
   float         hihi;      /*Hihi Alarm Limit*/
   float         lolo;      /*Lolo Alarm Limit*/
   float         high;      /*High Alarm Limit*/
   float         low;       /*Low Alarm Limit*/
   double        mdel;      /*Monitor Deadband*/  }
```

Запрос поиска PV устанавливает виртуальный канал и выбирает дополнительную информацию о рекорде.

ca_search_PV (name_PV, CHID_record, time-out, asyn_subroutine)

Запросы обмена данными выполняют анализ передаваемых параметров, если параметры соответствуют параметрам рекорда, запрос выполняется.

ca_put_PV (PV, CHID_record, count, time-out, asyn_subroutine)

ca_get_PV (PV, CHID_record, count, time-out, asyn_subroutine)

Запросы управления событиями:

ca_set_event_PV (CHID_record, mask)

ca_wait_event_PV (CHID_record, mask, time-out, asyn_subroutine)

mask – маска с набором битов для каждого триггера событий, определяет тип событий. Маска триггера событий является “логическим или” одной или более констант.

DBE_VALUE – значение канала превышает зону нечувствительности (dead band) для мониторинга

DBE_LOG – значение канала превышает зону нечувствительности (dead band) для архиватора

DBE_ALARM – состояния «сигнала ошибки» изменилось.

Для каждого CHID_*record* по умолчанию создается одна программа реакции на особые ситуации (event/connection/exception handler), которая контролирует состояние линии связи и внешние события.

Приложение В. Поддержка для канала ввода/вывода CAMAC

CAMAC драйвер предназначен для работы на платформе операционной системы LynxOS, обслуживающий персональный компьютер с процессором типа Intel x86. ISA плата - последовательно-параллельный интерфейсом ППИ-6 обслуживает шесть каналов для подключения крейт контролеров К0607 [3]. Аппаратно на плате перемычками можно устанавливать два значения адреса портов ввода/вывода, а также номера вызовов прерываний (IRQ). Что позволяет одновременно в компьютере установить два интерфейса: первый с базовым адресом 0x250 и IRQ-5, второй с 0x240 и IRQ-7 и работать с двумя **символьными специальными файлами** (character special file): **/dev/PPI0, /dev/PPI1**. Стандартный вызов к CAMAC модулю определяется параметрами *BCNAF*:

B (branch) – соответствует символьному специальному файлу **/dev/PPI0** или **/dev/PPI1**, определяется описателем файла – **fd**, например:

(int) fd = open (“/dev/PPI0”, (int) oflags, (mode_t) mode)

C (crate) – номер крейта.
N (number of module) – номер позиции модуля в крейте.
A (sub-address) – субадрес.
F (function) – функция.

Через вызов *ioctl* в драйвер передается значения *CNAF*:

*(int) ioctl ((int) fd, (int) command, (struct CNAF_ioctl *) Argument)*

(int) command определяет операцию, которую выполнит драйвер

CNAF – передать CNAF.
Z_operation – выполнить операцию Z.
C_operation – выполнить операцию C.
I_operation – выполнить операцию I.
ISR_operation – разрешить прерывание.
NoISR_operation – запретить прерывание.
DebugStatus – режим отладки, после завершения операции в программу будут возвращены состояния всех регистров ППИ-6.

В структуре *CNAF_ioctl* можно определить значения *CNAF* и максимально допустимое время исполнения операции.

```
struct CNAF_ioctl
  {     char      Crate;              /* C - crate */
        char      N_module;           /* N - module */
        char      A_subaddress;       /* A - subaddress */
        char      Function;           /* F - function. */
        char      Q_LAM_Word;         /* descpition of operation */
        long      TimeOut_sec;        /* timeout in seconds */
        long      TimeOut_usec;       /* timeout in microseconds */     };
```

Q_LAM_Word определяет размер передаваемого слова, завершение операции с ожиданием Q по умолчанию, или завершение операции по приходу LAM. *Q_LAM_Word* является логическим ***И*** следующих значений операндов:

LAM_CAMAC – ожидание LAM.
Word16 – обмен двухбайтовыми словами.
Word24 – обмен трехбайтовыми словами.

Для асинхронного обмена с CAMAC модулем необходимо создать системный *thread* один для группы LAM, что можно выполнить посредством вызова с кодом операции *ISR_operation*. Будет установлен соответствующий бит в маске регистра масок крейт контроллера, зарегистрирован вызов в очереди на обслуживание данной группы LAM вызовов. Если очередь пустая при поступлении вызова с кодом операции *NoISR_operation*, системный процесс (thread), обслуживающий данную группу LAM вызовов удаляется, прерывания от данной группы запрещаются.

Стандартные синхронные вызовы обмена данными:

*(int) read ((int) fd, (char *) Buffer, (int) Counter)*

*(int) write((int) fd, (char *) Buffer, (int) Counter)*

Стандартные асинхронные вызовы обмена данными:

*(int) aread ((int) fd, (char *) Buffer, (int) Counter, (struct aiocb *) cb)*

*(int) awrite((int) fd, (char *) Buffer, (int) Counter, (struct aiocb *) cb)*

*char *Buffer* – адрес массива передаваемых данных.
int Counter – число передаваемых байтов.
*struct aiocb *cb* – структура асинхронного контрольного блока ввода/вывода.

И обязательная стандартная процедура *close,* которая «правильно завершает» незавершенные вызовы. Что важно для удаления системных *thread*, которые обслуживают незавершенные вызовы, ожидающие прихода LAM:

(int) close ((int) fd)

Экспериментально измеренное минимальное время исполнения одного *CNAF* равно 16 микросекундам, включая CAMAC цикл, определяется следующей последовательностью операций:

а) проверка готовности ППИ-6,
б) посылка CNA в крейт контроллер,
в) посылка F в крейт контроллер,
г) чтение или запись шестнадцати битового слова и старшего байта,
д) выборка статусных регистров для определения кода завершения передачи данных.

Это сумма времен – программное время формирования вызова $t_{CPU} \approx$ 1нсек, определяемое тактовой частотой процессора >500МГц, время одного обращения к порту ППИ-6 $t_{ISA} \approx 0.125$ мксек, определяется тактовой частотой ISA шины 8МГц, всего восемь обращений $\sum t_{ISA} \approx$ мксек, время CAMAC цикла $t_{CAMAC} =$ 1мксек. Следовательно остальное время ≈14 мксек тратится на работу ППИ-6 и крейт-контроллера К607, что составляет 87% времени.

Драйвер является частью IOC, с CAMAC оборудованием работает только один процесс ***iocCore***, доступ к CAMAC ресурсам для других процессов запрещен. Условия разделения ресурсов решаются внутри ***iocCore***, когда рекорд обрабатывается, в поле рекорда **PACT** устанавливается флажок – «процесс активный», что запрещает запуск других механизмов обработки данного рекорда. А при работе в критичной области программы (critical code region, где не допустимо прерывание со стороны других процессов), устанавливается блокировка доступа. Блокировка доступа необходима при работе с разделяемыми ресурсами, что обеспечивает корректность работы в многозадачные системах с приоритетным прерыванием обслуживания (preemptive scheduling), механизм которого поддерживают операционные системы жесткого реального времени: LynxOS и VxWorks.

На рисунке 9 представлена структура канала ввода/вывода для оборудования **CamacPPI**. Структура *struct camacPPI* содержит следующие параметры: номер интерфейса ППИ-6, номер крейта и размер передаваемого слова в байтах, эти параметры являются общими для всех NAF в данном

рекорде. Рекорд может содержать максимум девять NAF, которые выполняются при обработке рекорда, и до шести NAF для инициализации рекорда, что определяется предоставляемыми ресурсами **database access** службы.

Чтобы учесть время реакции CAMAC модуля или организовать цикл (Рис. B.1), в структуру кроме параметров NAF включены два дополнительных параметра для создания цикла. Qwait – это счетчик цикла, ticks – временная задержка, для которой квант времени равен 10мксек, что временем таймаута ППИ-6.

Рис. B.1. Схема цикла обслуживания NAF.

Таблица B.1. Универсальные подпрограмм поддержки устройств для канала ввода/вывода CamacPPI**.**

Тип рекорда	Синхронные	Асинхронные
ai	devAaiCamacPPI.c devAiCamacPPI.c	DevAaiCamacPPI_Asyn.c devAiCamacPPI_Asyn.c
ao	devAaoCamacPPI.c devAoCamacPPI.c	devAaoCamacPPI_Asyn.c devAoCamacPPI_Asyn.c
bi	devBiCamacPPI.c	devBiCamacPPI_Asyn.c
bo	devBoCamacPPI.c	devBoCamacPPI_Asyn.c
longin	devLiCamacPPI.c	devLiCamacPPI_Asyn.c
longout	devLoCamacPPI.c	devLoCamacPPI_Asyn.c
mbbiDirect	devMbbiDirectCamacPPI.c	devMbbiDirectCamacPPI_Asyn.c
mbboDirect	devMbboDirectCamacPPI.c	devMbboDirectCamacPPI_Asyn.c

И был написан пакет подпрограмм поддержки устройств, обеспечивающих синхронный и асинхронный доступ к аппаратуре для основного набора типов рекордов, которые перечислены в таблице B.1. Все подпрограммы работают с каналом ввода/вывода CamacPPI, подробно структура канала представлена на рисунке 9. В поле рекорда **INP/OUT** (input/output link) находятся цепочка NAF, во всех подпрограммах поддержки устройств последний NAF выполняет передачу данных.

$$NAF_{LAST} \xrightarrow{data} VAL$$

$$NAF_{LAST} \xrightarrow{data} DEVICE$$

А также были специально написаны отдельные подпрограмм поддержки устройств, учитывающие индивидуальные особенности работы с устройствами.

Выбранный вариант передачи параметров в канал ввода/вывода очень удобен для работы *thread* в качестве подпрограммы поддержки устройства. Например, создается рекорд типа **ai** (аналоговый ввод) для контроля работы технологического объекта, обслуживаемый *thread*. *thread* постоянно активный и посредством одного из NAF периодически с заданным интервалом выбирает данные и помещает их в поле рекорда **VAL**, что является стандартным алгоритмом для обслуживания рекордов типа **ai**. И дополнительно в подпрограмме анализируется полученное значение, и в случае выхода за пределы производит отключение. При этом используется интерфейс базы данных IOC, и достигается оперативность, которая невозможна при использовании механизма sequencer[7].

Литература

1. Базовый URL сообщества EPICS. http://www.aps.anl.gov/epics.
2. Кнут Д. *Искусство программирование для ЭВМ. Сортировка и поиск*. Т.3. - Москва: Мир, 1978.
3. Нифонтов В.И., Орешков А.Д., Путьмаков А.И. и др. *К607 – контроллер и драйвер для организации связи в последовательном виде между ЭВМ «Электроника-60» и крейтами КАМАК* // Препринт ИЯФ 82-90, Новосибирск, 1990.
4. Anderson J.N., Kraimer M.R. *EPICS IOC Record Reference Manual. Draft Octoder 1992* // Argonne National Laboratory http://www.aps.anl.gov/epics
5. Boggs D.R., Mogul J.C., Kent Ch.A. *Measured Capacity of an Ethernet: Myths and Reality*. WRL Research Report 88/4, DEC. // Western Research Laboratory, September 1988. http://www.research.digital.com/wrl/techreports/index.html.
6. Butenhof D.R. *Programming with POSIX Threads*. – Addison-Wesley Longman Inc., 1997
7. Hill Jeffrey, *EPICS R3.12 Channel Access. Reference manual*. 11 December, 1996. // Los Alamos National Laboratory Los Alamos. http://www.aps.anl.gov/epics

[7] Механизм для подпрограмм, выполняющих роль автомата конечных состояний.

8. Gallmeister B.O. POSIX.4: *Programming for the Real World*. – O'Reilly & Associates, Inc., 1995.
9. Kraimer M.R. *EPICS IOC Application Developer's Guide. Release 3.13.0beta12*. June 1998 // Argonne National Laboratory. http://www.aps.anl.gov/epics
10. Lewis B., Berg D.J. Multithreaded Programming with Pthreads. – Sun Microsystems Press. 1998.
11. Liu C.L., Layland J.W. *Scheduling algorithm for multiprogramming in hard real time environment.* // Journal. ACM. Vol. 20, No 1. January 1973. P. 46-61
12. Nichols B., Buttlar D., Farrell Ja. P. *Pthreads Programming*. – O'Reilly & Associates, Inc., 1996.
13. Stanley Ph., Anderson J.N., *Kraimer M.R. EPICS Record Reference Manual. Release 3.13*. 1995. // Argonne National Laboratory http://www.aps.anl.gov/epics
14. Stevens W.R. *Advanced Programming in the UNIX Environment*. –Addison-Wesley Pub.Company. 1992.
15. Stevens W.R. *TCP/IP Illustrated, Volume 1. The Protocols*. – Addison-Wesley Logman, Inc. 1994.
16. Stevens W.R., Wright G.R. *TCP/IP Illustrated, Volume 2. The Implementation*. – Addison-Wesley Logman, Inc. 1995.
17. *VxWorks 5.3,Programmer's Guide*. – Wind River Systems Inc. 1995. – p.538.
18. *VxWorks 5.3, Reference Manual*. – Wind River Systems Inc. 1995.
19. *IEEE P1003.1, POSIX Draft 7*. June 2001 // Open Group Technical Standard, IEEE. http://www.ieee.org/

Т.В. Саликова

Портирование EPICS на платформу операционной системы реального времени LynxOS

T.V. Salikova

Porting EPICS to real time operating system LynxOS

ИЯФ 2004-10

Ответственный за выпуск А.М. Кудрявцев
Работа поступила 24.02.2004

Сдано в набор 25.02.2004
Подписано в печать 26.02.2004
Формат 60x90 1/16 Объем 2.8 печ.л., 2.3 уч.-изд.л.
Тираж 85 экз. Бесплатно. Заказ № 10

Обработано на IBM PC и отпечатано
на ротапринте ИЯФ им. Г.И. Будкера СО РАН
Новосибирск, 630090, пр. Академика Лаврентьева, 11