Канд.техн.наук С.Г. Семенцов, инж. А.В. Аксенов, инж. А.Н. Кобзарев

ОБЩИЙ ПОДХОД К ПОСТРОЕНИЮ ОПЕРАЦИОННЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ В СИСТЕМАХ УПРАВЛЕНИЯ

Введение

При решении задачи создания высокопроизводительных систем управления и обработки больших массивов в реальном времени, в частности при создании систем управления и защиты атомных станций, возникает необходимость использования так называемых операционных систем реального времени (ОСРВ). Благодаря разветвлённой архитектуре и иерархической системе выполнения процессов, становится возможным своевременное выполнение важнейших системных процессов и отстранение на второй план остальных вспомогательных процессов. В качестве примера рассмотрим схему построения обычного ядра операционной системы (Unix).

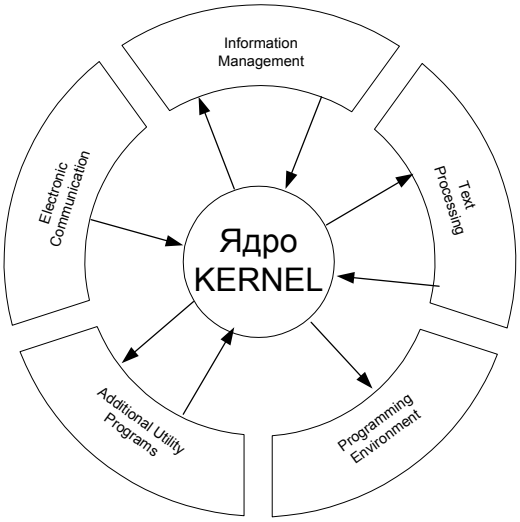

Рис.1. Структура ядра Unix-подобной ОС

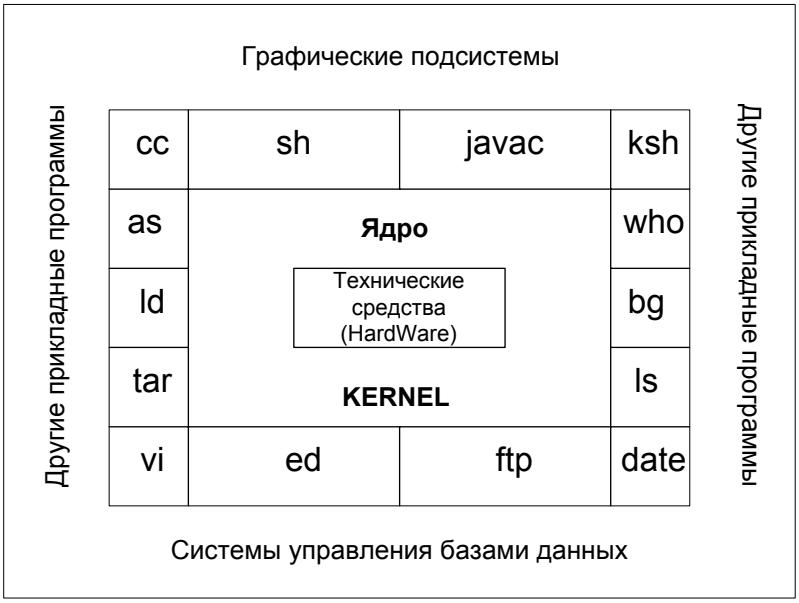

Рис.2. Структура ОС

На рис. 1 представлен набор сфер, каждая из которых представляет собой специфический набор программ Unix, включающих в себя такие компоненты (рис. 2), как:

- ядро;
- shell (командный интерпретатор);
- системные утилиты;
- программные средства, которые используются в режиме выполнения команд;
- некоторые подсистемы верхнего уровня.

На рис. 3 показана пирамида приоритетов выполнения процессов. Как видно из рис. 3 все пользовательские и системные утилиты имеют маленький приоритет, из-за чего наблюдается простой системы и, соответственно, снижение производительности [1]. Особенно этот фактор сказывается на повторяющихся операциях, таких как обработка и анализ сетевых пакетов. Также, из-за невозможности множественного запуска системных серверов (демонов), трудно реализуема синхронная обработка нескольких потоков. В случае их последовательной обработки время выполнения задач значительно превысит допустимый для систем реального времени временной лимит.

Рис.3. Распределение приоритетов в ОС

Учитывая вышеприведенные ограничения ОС общего применения для решения подобных задач необходимо применение ОС со специальной архитектурой – ОС реального времени (ОСРВ). Разберём основные особенности её построения, архитектуры и требования к аппаратному обеспечению.

Все компоненты ОСРВ функционально не отличаются от ОС общего применения, так как базовые отличия заключены в ядре системы, а именно, в порядке обработки процессов и раздачи приоритетов на право первичной обработки. Следовательно, поменялась и схема приоритетов (рис.4).

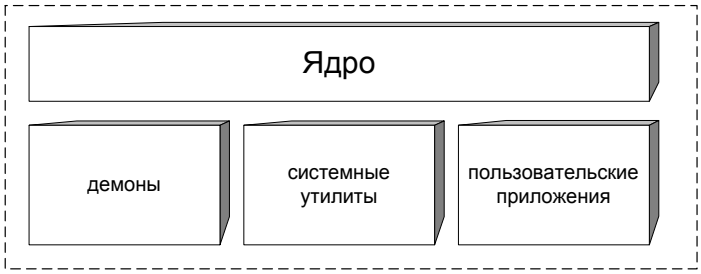

Рис.4. Распределение приоритетов в ОСРВ

Как видно из рисунка, все составляющие находятся на одной высоте, так как приоритеты выдаются вручную и никак не зависят

от поведения системы. Тем самым могут выполняться сразу несколько процессов, у них будут одинаковые идентификаторы, следовательно, на них будет выделено равное процессорное время. Благодаря такому механизму можно выполнять сразу несколько процессов, в нашем случае это одновременная обработка сразу нескольких потоков данных независимо друг от друга (рис.5).

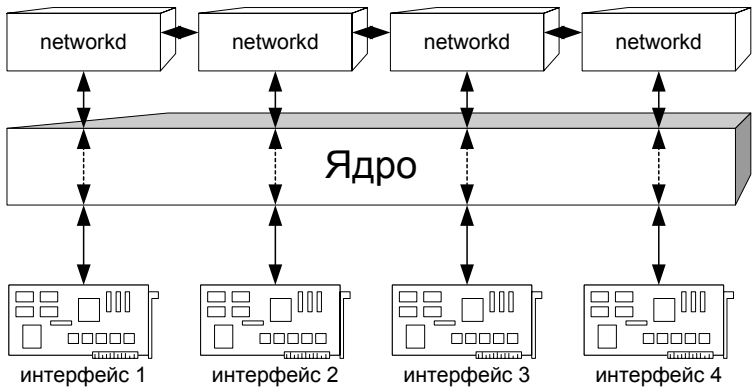

Рис.5. Управление интерфейсами

В данной реализации ОС возможно запустить сразу несколько обработчиков интерфейсов, что позволит независимо друг от друга работать с различными потоками данных. Это решение предоставляет большую свободу выбора при реализации механизмов управления системой защиты атомных станций.

Методы решения типовых задач ОСРВ

К операционным системам реального времени предъявляется ряд требований. Одним из важнейших является минимальный размер системы [2]. Размер кода по сравнению с ОС общего применения уменьшается за счет исключения из него некоторых прикладных программ, отсутствие которых не сказывается на работе всей системы и не нарушает ее гибкости. Упрощение также возможно путем сокращения дополнительных функций в прикладном программном обеспечении [3].

Что касается многофункциональности, то дело здесь обстоит несколько сложнее, поскольку обычные системы общего применения обладают значительно более широким набором функций. Это один из недостатков такого подхода при создании ОСРВ. Однако

при создании систем для промышленных приложений очевидно, что необходимо жертвовать функциональностью для уменьшения размеров ядра.

Для обеспечения многозадачности в ОСРВ требуется значительный объем оперативной и сверхоперативной памяти. Связано это с тем, что для обеспечения обработки данных в реальном времени необходимо обеспечить отсутствие задержек. Следовательно, существует необходимость использования быстрых видов памяти. Кэширование данных в ПЗУ в этом случае неприемлемо, так как скорости обмена информацией будут на несколько порядков ниже, по сравнению с ОЗУ. Возможно, с созданием нового поколения ПЗУ, скорость обмена информацией с которыми будет достаточна для выполнения данной задачи, эта проблема будет решена. Если в управление ОС включаются алгоритмы обработки данных в реальном времени, то задачи, выполняемые вне этих алгоритмов, автоматически имеют пониженный приоритет выполнения, а значит и доступ к памяти. Обойти эти ограничения довольно сложно. Из вышесказанного очевидно что, ОС РВ имеют множество недостатков, как, впрочем, и достоинств. В последнее время предпринимаются попытки объединения преимуществ ОСРВ и ОС общего применения, т.е. наметилась тенденция к интеграции обычных ОС и ОСРВ.

Существуют два основных направления развития технологий реального времени: разработка многофункциональных ОСРВ и написание встраиваемых в обычные ОС модулей для поддержки режима РВ. Второе направление очень бурно развивается в последнее время. Это связано с тем, что модификация используемых ОС экономит конкретному заказчику деньги на переобучение персонала, а разработчику время на их создание. ОСРВ все чаще интегрируются в системы с открытым кодом (FreeBSD, Linux и др.). Преимущество этих систем перед остальными очевидно: отсутствие затрат на приобретение исходного кода, возможность создания собственного программного обеспечения, не обращаясь за описанием необходимых функций к разработчикам. Кроме того, стабильность, свойственная Unix-системам, выделяет их из числа прочих ОС. Стоит отметить, что многие фирмы, занимающиеся созданием ОСРВ, в пакеты с ОС включают специальные языки программирования с поддержкой функций реального времени. Иногда данные функции внедряются в популярные среды разработки ПО, такие как C++. Причем зачастую компиляторы пишутся специально для данной ОС. Еще одним достоинством такой интеграции является полная совместимость программных продуктов для обеих версий ОС (см. таблицу).

Преимущество использования дополнительного ядра-модуля

(рис 6.) состоит в том, что оно начинает работу лишь тогда, когда запускается программа-клиент, требующая для правильной работы поддержки РВ [4]. Также возможен вариант альтернативного выбора режима пользователем. В любом случае система способна выполнять обычные программы, используя при этом стандартное ядро, что существенно увеличивает гибкость ОС.

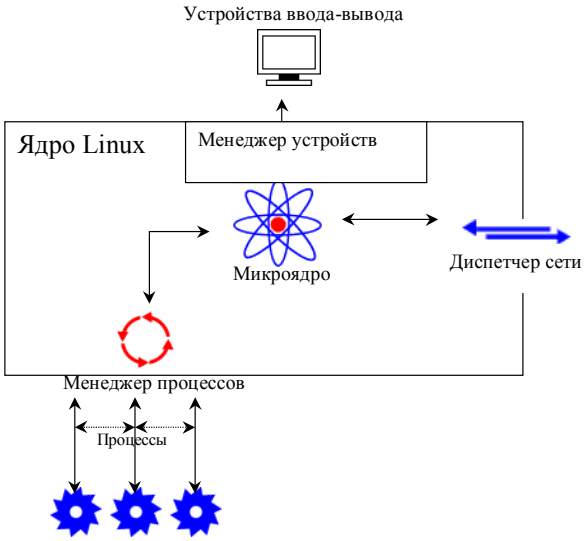

Рис.6. Структурная схема работы ОС Matrix RT

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

В многозадачных ОС общего назначения используются, как правило, различные модификации алгоритма круговой диспетчеризации, основанные на понятии непрерывного кванта времени (time slice), предоставляемого процессу для работы. Планировщик по истечении каждого кванта времени просматривает очередь активных процессов и принимает решение, кому передать управление, основываясь на приоритетах процессов (присвоенных им численных значениях). Приоритеты могут быть фиксированными или меняться со временем - это зависит от алгоритмов планирования в данной ОС, но рано или поздно процессорное время получат все процессы в системе.

Сравнение основных параметров ОСРВ

Параметр	QNX	pSOS+	LynxOS	Matrix RT
Время реакции на перерывание[1] (interrupt latency) при максимальной загрузке процессора (мкс)	20	3	8	~50
Время переключения контекста[2]	100	130	170	150
Размеры системы[3]	10 кбайт	?	?	1,5 Мбайт
Возможность исполнения системы из ПЗУ (ROM)	Да	Да	Да	Да
Алгоритмы диспетчеризации	Приоритетный с вытеснением	Адаптивная диспетчеризация	FIFO, квантование времени, Round-Robin, адаптивные	Адаптивная диспетчеризация, FIFO
Механизмы межзадачного взаимодействия	Семафоры, события, сигналы, каналы данных (pipes), очереди сообщений	Семафоры, события, сигналы, каналы данных, средства для работы с разделяемой памятью, очереди сообщений	Семафоры, события, мьютексы, сигналы, каналы данных, очереди сообщений	Семафоры, события, сигналы, каналы данных, очереди сообщений

[1] Интервал времени - от момента возникновения события на объекте до выполнения первой инструкции в программе обработки этого события - и является временем реакции системы на события (время выполнения цепочки действий - от события на объекте до генерации прерывания - никак не зависит от ОСРВ и целиком определяется аппаратурой, а вот интервал времени от возникновения запроса на прерывание и до выполнения первой инструкции

его обработчика определяется целиком свойствами операционной системы и архитектурой компьютера).

[2] Время, которое система затрачивает на передачу управления от процесса к процессу (от задачи к задаче, от нити к нити) - время переключения контекста.

[3] Размер системы исполнения, а именно суммарный размер минимально необходимого для работы приложения системного набора (ядро, системные модули, драйверы и т. д.).

Алгоритмы круговой диспетчеризации в чистом виде в ОСРВ неприменимы. Основной недостаток - непрерывный квант времени, в течение которого процессором владеет только один процесс. Планировщики же ОСРВ имеют возможность сменить процесс до истечения временного кванта, если в этом возникла необходимость. Один из возможных алгоритмов планирования при этом - "приоритетный с вытеснением". ОСРВ отличаются богатством различных алгоритмов планирования: динамические, приоритетные, монотонные, адаптивные и пр., цель же всегда преследуется одна - предоставить инструмент, позволяющий в нужный момент времени исполнять именно тот процесс, который необходим [5].

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

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

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

Здесь приведены только базовые, обязательные механизмы, использующиеся в ОСРВ. Кроме того, почти в каждой ОСРВ имеется целый набор дополнительных, специфических только для нее, механизмов, касающихся системы ввода-вывода, управления преры-

ваниями, работы с памятью. Каждая система содержит также ряд средств, обеспечивающих ее надежность: встроенные механизмы контроля целостности кодов, инструменты для работы со сторожевыми (Watch-Dog) таймерами.

Для полноты картины укажем некоторые, наиболее часто встречающиеся, недостатки ОС РВ. Во-первых, далеко не для всех устройств, требующих поддержки дополнительного ядра, существуют драйверы, обеспечивающие правильную работу в этом режиме. Но этот недостаток не относится к техническим и не понижает достоинства системы, так как создание драйверов является делом времени и осведомленности фирмы производителя. Как правило, такие фирмы регулярно увеличивают базы драйверов, а так же выпускают драйверы по требованию заказчика. Второй недостаток более существен и связан с большим временем реакции на событие. Очень сложно разработать систему, удовлетворяющую этому требованию, так как нет единого толкования этого параметра и разработчики коммерческих систем часто приводят значения, которые не соответствуют истине [6,7].

Верификация ОСРВ

Процесс верификации и аттестации используется не только для тестирования созданной системы, но и на ранних стадиях проектирования, для сбора информации по системам-прототипам. Целью процесса верификации и аттестации программного обеспечения является оценка уровня безопасности, отказоустойчивости и надежности системы.

При вводе в эксплуатацию создаваемой системы необходимо следовать жёстким рекомендациям, устанавливаемым стандартом, который предписывает общие подходы и основные принципы при разработке и испытаниях ПО. В настоящее время существует проект дополнения [7] стандарта МЭК 60880 по проблемам программного обеспечения для систем контроля и управления классов 2 и 3 по МЭК 61513 [8,9], т.е. для систем, выполняющих важные функции поддержки безопасности, и систем, выполняющих информационные и вспомогательные функции.

Наряду с использованием синтаксических анализаторов компилятора, статических анализаторов (lint), необходимо использовать специальные программные средства для анализа синтаксически правильных конструкций, несущих в себе скрытые ошибки, а также средства для выявления потенциальных ошибок безопасности (security scanners), далее – сканер.

Сканер просматривает исходный текст программы, пытаясь об-

наружить библиотечные вызовы, которые могут показывать на уязвимость с точки зрения безопасности. При этом, обнаружив такой вызов и используя специальную базу данных уязвимостей, которая содержит информацию об опасных функциях, сканер может представить конкретные сведения о том, какие проблемы данная функция может нести, степень уязвимости и риска, которые она несет, и общее решение в данном случае. Данный инструмент имеет большое количество настроек, для обеспечения должного уровня безопасности и скорости обработки больших объемов исходных текстов, а также возможности по расширению и обновлению базы данных выявленных уязвимых функций и созданию специфической узконаправленной базы.

В зависимости от используемого ПО, база данных сканера может содержать от сотни типовых опасных функций, которые разделены по степени риска от их использования, на 6 основных категорий: от 0 – безопасное использование, до 5 – наиболее небезопасное.

Типовой сканер обеспечивает выделение функций, имеющих следующие уязвимости:

1. Ошибки переполнения буфера:
   - функции, имеющие 3 степени риска (1, 3, 4);
   - в циклических конструкциях;
   - в зависимости от библиотечной реализации.
2. Создание условий гонки при взаимодействии процессов / файлов (4 возможных случая).
3. Использование функций на основе реализации rand().
4. Использование функций семейства exec().
5. Отсутствие проверки полученной информации функциями ввода на ее соответствие ожидаемой (ill effect).
6. Отсутствие проверки в функциях ввода-вывода на подмену файлов символическими ссылками.
7. Использование в функциях строк переменного формата (non-constant).

Заключение

На основании приведенных выше выводов можно сделать вывод, что Unix-подобные системы безусловно могут быть использованы в качестве основы для ОСРВ систем управления различного назначения. Процесс разработки такой ОС можно представить в следующем виде.

На первом этапе разрабатывается микроядро новой ОС, чему предшествует процесс верификации операционной системы – прототипа.

Основной сложностью при создании микроядра является разработка перехватчика прерываний и диспетчера процессов. Основной упор необходимо сделать на оптимизацию кода микроядра, что позволит заранее исключить нежелательные временные задержки. Следующая проблема – выбор эффективной схемы диспетчеризации, так как она в конечном итоге определит основной параметр системы – время реакции на прерывание. Большая часть времени разработки ложится на отладку планировщика.

Следующим этапом в создании ОСРВ является написание менеджеров устройств, работающих по механизму реального времени.

Написание драйверов устройств, обслуживаемых микроядром, – достаточно трудоемкий процесс, так как от правильно написанного драйвера зависит правильное функционирование устройства и в конечном итоге время реакции на событие, которое складывается из времени реакции на прерывание и времени передачи сигнала на устройство, а так же его реакции на сигнал. В системе шифрации драйвер пишется для сетевой платы, модема или другого сетевого устройства.

Наконец отладка работы системы в целом и верификация системы. Этот этап включает в себя контроль основных параметров ОС и получения полного списка используемых функций, пакетов и т.п. В частности такой подход применен при разработке ОС «мягкого» реального времени LiNEM в НПП ВНИИЭМ. Данная ОС предназначена для работы в составе программно-технического комплекса информационно-диагностической сети системы управления и защиты реакторов ВВЭР-1000, применяемых на АЭС «Тяньвань», «Бушер» и др. Проведенные квалификационные испытания этой системы показали высокую устойчивость и стабильность системы в широком диапазоне внешних воздействий.

ЛИТЕРАТУРА

1. Жданов А., Латыев А. Замечания о выборе операционных систем при построении систем реального времени. ЗАО "РТСофт". М.: "PCWeek". N 1. 2001.

2. Фрейдман А.В. Разработка встроенных систем с комфортом Nucleus EDE - интегрированная среда разработки встроенных систем. "Науцилус". 2000.

3. Жданов А.А. Современный взгляд на операционные системы реального времени. ЗАО "РТСофт". М.: 2001.

4. Мороз А.А., Михненко А.Е. Операционная платформа реального времени – MATRIX REALTIME // Молодежная научно-техническая конференция «Наукоемкие технологии и интеллектуальные системы», 17-18 марта 2001 г. М.: МГТУ им. Н.Э. Баумана. С.150-153.

5. Власов А.И., Лыткин С.Г., Яковлев В.Л. Краткое практическое руково-

дство разработчика по языку PL/SQL. М.: Машиностроение. 2000. (Библиотечка журнала "Информационные технологии").

6. Власов А.И., Колосков С.В., Пакилев А.Е. Нейросетевые методы и средства обнаружения атак на сетевом уровне// 2-я Всероссийская конференция Нейроинформатика-2000. Сб. научных трудов. Ч.1. М.: МИФИ. 2000. С. 30-40.

7. Шахнов В.А., Мороз А.А., Михненко А.Е., Власов А.И. Операционная система реального времени Matrix RealTime как основа для построения экспериментальных систем обработки сигналов в реальном времени // 2-я Международная конференция СНГ "Молодые ученые - науке, технологиям и профобразованию для устойчивого развития: проблемы и новые решения". М.: Октябрь. 2000. Ч.2,3. С.100-103.

8. IEC. Draft. 1999. Computer-based systems important for safety. Software aspects for I&C systems of class 2 and 3.

9. IEC 61513. 1998. Nuclear power plants – Instrumentation and control for systems important to safety – General requirements for systems.