Архитектура и принципы построения операционной среды «мини-ОС»

Ю.Н. Фонин, С. Грассман

Аннотация. В статье представлены архитектура и принципы построения операционной среды «мини-ОС», предназначенной для использования в так называемых встроенных системах. Особое внимание в статье уделено вопросам портирования мини-ОС и вопросам минимизации усилий на адаптацию ОС под требования аппаратуры и приложений пользователя.

1. Введение

Для обеспечения функционирования сложных систем цифровой обработки сигналов (ЦОС), таких как системы, реализующие протоколы беспроводной передачи данных в режиме реального времени, требуются специальные, адаптированные под конкретные приложения, аппаратные конфигурации. Такие конфигурации могут включать в себя несколько процессоров, а также специальные аппаратные ускорители для быстрого выполнения тех или иных функций. Для портирования алгоритмов на такие аппаратные системы необходим набор сервисов, поддерживающих коммуникации и синхронизацию между различными потоками в приложении. Обычно такие сервисы предоставляются операционной средой.

Для портирования стандартной операционной системы реального времени требуется от 3 до 20 человеко-месяцев, в зависимости от сложности системы. С другой стороны, для реализации задач ЦОС требуется только некоторое подмножество модулей ОС, и, соответственно, для портирования этих модулей нужно меньше времени, чем для портирования полноценной ОС.

В ходе работы были исследованы различные операционные среды – от больших, многофункциональных систем, таких как Linux и Symbian, до небольших, обладающих минимальным набором функций, таких как VsWorks, RTEMS и EUROS [1, 2, 4]. Особенностью всех этих систем является то, что они основываются на неразделяемом ядре с базовым набором модулей, таких как планировщик задач, менеджер управления прерываниями, менеджер драйверов ввода-вывода, набор примитивов синхронизации, системный таймер и т.д. При этом ни один из перечисленных модулей не может быть исключен из системы, даже если он не используется непосредственно приложением. По этой причине для портирования такой системы требуется заново реализовать все платформо-

зависимые части ОС, что существенно увеличивает время портирования системы.

В настоящее время существуют несколько способов ускорения процесса портирования операционной системы. Например, в системе Choice [5] вся платформеннозависимая часть ОС логически выделяется в так называемое «наноядро». При таком подходе портирование ОС на новую платформу сводится к реализации наноядра. Другой метод [6] заключается в документировании методологии портирования посредством паттернов. Для создания и формализации шаблонов разработан специальный «язык паттернов». Такой подход позволяет формализовать процесс портирования ОС.

Нами была разработана операционная система «мини-ОС», отличительными особенностями которой является модульность и конфигурируемость. Любой из базовых модулей может быть исключен из системы без необходимости модификации остальной части ОС. Данное свойство мини-ОС позволяет значительно ускорить процесс портирования системы на новую платформу, а также обеспечивает возможность оптимизации ОС под конкретный набор приложений.

Статья организована следующим образом. В разделе 2 обсуждаются концепция построения системы мини-ОС и основные требования, выдвигавшиеся к системе. Раздел 3 описывает структуру системы, её основные модули. В разделе 4 рассматриваются платформеннозависимые части ОС в соответствии с использующими их модулями. В разделе 5 приводится пример портирования системы на многопроцессорную систему ARM_MUSIC.

2. Концепция разработки мини-ОС

При разработке минимальной операционной системы (далее мини-ОС) к ней выдвигались следующие требования:

1) Модульность – мини-ОС представляет собой базовый набор ортогональных модулей, реализованных таким образом, что конфигурация из любого набора модулей может быть скомпилирована без внесения изменений в код системы.
2) Портируемость – время на адаптацию системы к новой архитектуре должно быть минимальным. Количество платформеннозависимых частей мини-ОС должно быть минимально, и их наличие должно обуславливаться только невозможностью реализации в платформенно-независимом виде.
3) Программный интерфейс мини-ОС должен быть совместим с интерфейсом ОС Windows.
4) Мини-ОС должна поддерживать выполнение приложений на многопроцессорной системе с общей памятью, т.е. должна обеспечиваться возможность синхронизации между потоками, работающими на разных процессорах.

Первое требование было удовлетворено за счет разработки различных реализаций одних и тех же модулей в зависимости от конфигурации ядра. В процессе компиляции мини-ОС выбирается та реализация, которая соответствует конфигурации системы. Такая гибкость была достигнута за счет использования директив условной компиляции в языке Си.

Для уменьшения времени портирования мини-ОС использовались следующие методы:

- Разделение платформеннозависимых и платформеннонезависимых частей ОС. Реализация платформеннозависимых частей в виде макросов. Использование инлайн-ассемблера.
- Формализация функциональности платформеннозависимых частей ядра.
- Поддержка многопроцессорной системы с общей памятью.

Совместимость с ОС Windows достигнута за счет использования интерфейсов функций, а также имен типов данных Windows при декларации объектов ОС, таких как поток, синхронизационный примитив и т.д.

Операционная система мини-ОС, построенная в соответствии с описанными выше принципами, позволяет реализовать следующий сценарий разработки и портирования приложений:

- Разработка и отладка многопоточного приложения выполняется под управлением ОС Windows с использованием стандартного интерфейса для управления потоками и объектами синхронизации и стандартного набора функции для организации ввода-вывода данных.
- Далее производится портирование операционной среды на аппаратную платформу, адаптированную под данное приложение. Тестирование мини-ОС выполняется с помощью специального набора тестовых приложений.
- Производится компиляция приложения совместно с мини-ОС под целевую платформу и выполняется оценка производительности полученной системы.
- Наконец, осуществляется оптимизация приложения и операционной системы. При необходимости производится модификация аппаратной платформы и соответствующих платформеннозависимых частей ОС.

Подобный подход позволяет, во-первых, отделить процесс разработки приложения от портирования на специальную аппаратную систему. Во-вторых, сама операционная система может быть оптимизирована в соответствии с требованиями приложения.

3. Структура мини-ОС

Ядро мини-ОС состоит из следующих модулей:

- *Модуль управления потоками (Task manager)* – включает в себя планировщик потоков, а также набор функций для управления

состоянием потока. Планировщик задач реализован в виде функции, осуществляющей выбор активного потока и переключение контекста.
- *Модуль динамического распределения памяти (Memory manager)* – содержит функции для динамического распределения памяти. Пользователю предоставляются две функции: *alloc()* для выделения памяти данного размера и *free()* для освобождения выделенной памяти.
- *Модуль синхронизационных примитивов (Synchronization primitives)* – включает в себя функции для создания синхронизационных объектов и управления синхронизационными примитивами: семафоры, мьютексы, бинарные события.
- *Модуль управления прерываниями (Interrupt controller)* – представляет собой набор функций для управления состоянием системы прерываний и функцию для инсталляции обработчиков прерываний.
- *Модуль ввода-вывода (IO manager)* – предоставляет унифицированный интерфейс для доступа к устройствам ввода-вывода.
- *Модуль начальной загрузки* – инициализирует регистры процессора, а также структуры и модули операционной системы. Он инициализирует главную задачу (функция main) и запускает её.

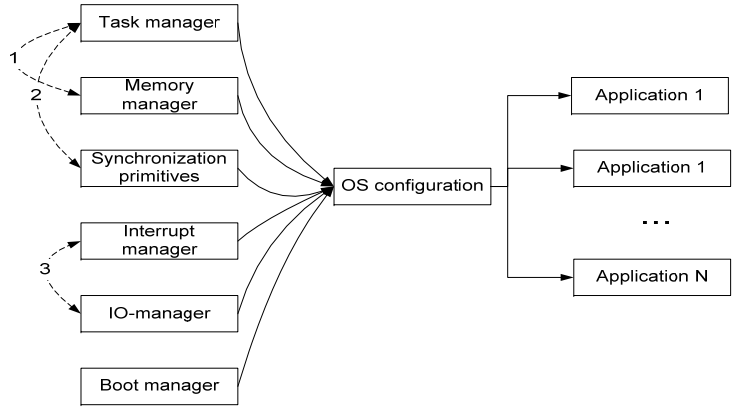

Рис. 1. Базовая структура мини-ОС

Для каждого из модулей имеются несколько реализаций в зависимости от конфигурации операционной системы. На Рис. 1 представлена базовая структура мини-ОС. Прерывистыми линиями обозначены так называемые «относительные зависимости», то есть, если один модуль включен в систему, то другой использует функции зависимого модуля. В противном случае используется реализация модуля без использования зависимого модуля.

В качестве примера рассмотрим зависимость модуля синхронизационного примитива от модуля управления потоками. Если ОС сконфигурирована как однозадачная, то применяется простой опрос объекта синхронизации. В том

случае, когда используется многозадачность, при переходе в режим ожидания сигнала вызывается функция Sleep(), которая переводит текущую задачу в так называемый «спящий» режим и вызывает планировщик задач. При соответствующем изменении состояния объекта синхронизации для «спящей» задачи вызывается функция Resume(), которая переводит ожидающую задачу в активный режим.

В минимальной конфигурации мини-ОС может состоять только из блока начальной загрузки, которая инициализирует процессор и передает управление main-функции приложения. Все остальные модули подключаются к системе в соответствии с заданной конфигурацией.

4. Портирование мини-ОС

Процесс портирования мини-ОС осуществляется модуль за модулем, то есть для каждого модуля реализуется его платформеннозависимая часть, затем модуль тестируется с помощью соответствующей тестовой программы, и после этого портируется следующий модуль. Такой подход позволяет значительно сократить время локализации ошибок, возникающих при портировании операционной системы.

Далее мы рассмотрим набор платформеннозависимых макросов для каждого из модулей, перечисленных в разд. 3. Заметим, что реализация любого макроса требуется только в том случае, если модуль, использующий данный макрос, применяется в требуемой конфигурации.

Платформеннозависимая часть модуля управления потоками включает в себя следующие макросы:

- Сохранение контекста;
- Восстановление контекста;
- Сохранение указателя контекста;
- Загрузка указателя контекста.

Для реализации данных макросов необходима следующая информация:

- Структура контекста, т.е. список регистров, которые отражают состояние процесса в момент прерывания потока.
- Указатель контекста – регистр, содержащий адрес, по которому сохраняется контекст.

Очевидно, что указанные параметры уникальны для каждого конкретного процессора. Поэтому упомянутые макросы должны быть реализованы для каждого нового процессора, на который портируется ОС.

Для портирования модуля динамического распределения памяти требуется определение следующих констант:

- *MMU_START_ADDR* – начальный адрес динамической памяти;
- *MMU_WORD_SIZE* – размер ячейки памяти, соответствующей единичному адресу в байтах;

- *MMU_PAGE_SIZE* – размер страницы памяти в байтах;
- *MMU_NUM_PAGES* – число страниц памяти, изначально доступных для динамического выделения.

Данные константы определяют адресное пространство блока динамически распределяемой памяти. Начальный адрес и размер блока определяется с учетом доступной памяти процессора, не занятой под хранение глобальных данных ОС или приложений, или памяти программ.

Для портирования модуля управления прерываниями на новую платформу необходимо реализовать следующие макросы:

- *DISABLE_ALL_IRQS()* – запрещает обработку всех маскируемых прерываний;
- *ENABLE_ALL_IRQS()* – разрешает обработку маскируемых прерываний;
- *ENABLE_IRQ_N()/DISABLE_IRQ_N()* – запрещает/разрешает обработку прерывания *N*;
- *SET_PRIOR_IRQ_N(prior)* – устанавливает приоритет прерывания *N*, если имеется программируемая система приоритетов.

Для реализации этих макросов требуется следующая информация:

- флаг «запретить все прерывания», запрещающий все прерывания в системе;
- таблица векторов прерываний;
- таблица приоритетов прерываний;
- для каждого прерывания:
  - флаг, запрещающий прерывание;
  - приоритет прерывания или адрес ячейки, содержащий приоритет прерывания.

Для портирования модуля синхронизационных примитивов для многопроцессорной системы требуется реализация макроса SYNC_SWAP(addr, read_val, send_val), который считывает из ячейки памяти по адресу addr значение переменной read_val и записывает значение send_val в ту же ячейку. Ключевым моментом, зависящим от особенностей целевой аппаратной платформы, является то, что в течение всего цикла чтения/записи никакой другой процессор или периферийное устройство не должны получить доступ к данной ячейке. Кроме того, требуется определение следующих констант:

1. *START_SYNC_RAM* – стартовый адрес синхронизационной памяти.
2. *SIZE_SYNC_RAM* – размер синхронизационной памяти (в байтах).

Заметим, что реализация макроса модуля синхронизации требуется только тогда, когда требуется поддержка межпросцессорных взаимодействий через синхронизационную память.

5. Портирование мини-ОС на платформу ARM7TDMI

В качестве аппаратной архитектуры для отладки и тестирования были использованы две системы:

1. Однопроцессорная система на базе процессора ARM7TDMI.
2. Архитектура ARM_MUSIC, состоящая из 32-х процессоров ARM7, общей памяти и специальной памяти синхронизации.

В качестве приложения использовалась многопоточная реализация протокола WLAN 802.11 [7], а также набор специальных тестов для проверки корректной работы каждого отдельного модуля ОС.

Для портирования мини-ОС был использован следующий сценарий:

1. Реализация блока начальной загрузки (за основу был взят стандартный boot, предоставленный разработчиками процессора).
2. Портирование модуля динамической памяти.
3. Портирование планировщика задач с программно управляемым вызовом планировщика (вызов планировщика осуществлялся с помощью функции *Sleep()*).
4. Портирование синхронизационных примитивов (однопроцессорная система).
5. Переход на многопроцессорную систему. Реализация макроса *SYN_SWAP()*.
6. Реализация драйвера таймера. Вызов планировщика по прерыванию таймера.
7. Портирование приложения WLAN, совместная компиляция WLAN и мини-ОС.

Суммарное время портирования операционной среды составило 6 человеко-дней. Для сравнения, портирование операционной среды EUROS в минимальной конфигурации требует минимум один человеко-месяц.

6. Заключение

В настоящей статье описаны принципы построения и структура операционной системы мини-ОС. Мини-ОС представляет собой набор модулей, каждый из которых может быть исключен из системы без изменения других модулей. При портировании мини-ОС требуется реализация только тех платформо-зависимых частей программы, которые используются приложением. Это позволяет осуществить пошаговый процесс портирования ОС на новую платформу, что существенно сокращает время, необходимое для портирования и отладки ОС.

Литература

1. Wind River Systems, Inc., “VxWorks 5.4”, available at http://www.wrs.com/products/html/vxwks54.html
2. RTEMS available at http://www.rtems.com
3. Eonic Systems, Inc., “Virtuoso v.4.1”, available at http://www.eonic.com/
4. EUROS documentation and manuals are available at http://www.kaneff.de/
5. A Case for Nano-Kernels. See-Mong Tan, David K. Raila, Roy H. Campbell Department of Computer Science University of Illinois at Urbana Champaign 1304 W. Springfiel Urbana, IL 61801.
6. A Pattern Language for Porting Micro-kernels. Michel de Champlain Departament of Computer Science, University of Canterbury, Christchurch, New Seland.
7. WLAN 802.11 available at http://grouper.ieee.org/groups/802/11/