САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Козлов Антон Павлович

Портирование операционной системы с модульным HAL в пользовательский режим

Дипломная работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Терехов А. Н.

Научный руководитель:
асп. каф. сист. прогр. Абусалимов Э. Ш.

Рецензент:
к.ф.-м.н. Фоминых Н. Ф.

Санкт-Петербург
2013

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Department

Anton Kozlov

Porting operating system with modular HAL to user mode

Graduation Thesis

Admitted for defence.
Head of the chair:
Professor A. N. Terekhov

Scientific supervisor:
Postgraduate E. S. Abusalimov

Reviewer:
PhD N.F. Fominykh

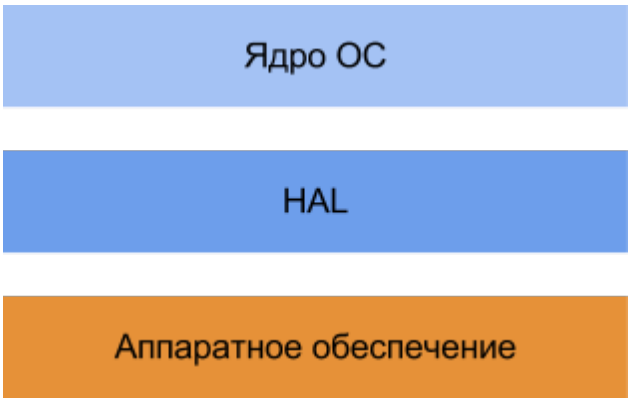
Saint-Petersburg
2013

Оглавление

1. Введение

Операционные системы используются на самых различных видах оборудования: персональные компьютеры, сервера, мобильные телефоны, сетевые устройства (коммутаторы, маршрутизаторы), встроенные системы. Часто, одна ОС с некоторыми изменениями может работать на разном аппаратном обеспечении, при этом её программный интерфейс остаётся неизменным.

Для этого операционные системы строятся из логических уровней. Уровни опираются друг на друга, используя некоторый интерфейс. На самом низком уровне находится HAL (Hardware Abstraction Layer) -=- слой аппаратных абстракций. В его задачи входит абстрагирование аппаратного обеспечения для более высоких уровней ОС.



Если интерфейсы HAL строго определены, высокоуровневая часть ОС пользуется только ими, то для переноса ОС на новую платформу достаточно разработать слой аппаратных абстракций.[8]

Существующие ОС имеют тенденцию рассматривать HAL как один большой компонент. Это влечет за собой трудность разработки и тестирования, так как архитектуры ОС предполагают, что слой аппаратных абстракций полностью реализован и функционирует верно, что скорее всего не выполняется в процессе его разработки. HAL является немалой частью ОС, которую невозможно создать без поэтапного тестирования.

Операционные системы общего назначения предоставляют комплекс мер по защите и изоляции запущенных программ. Эти меры сводят к

минимуму ущерб, который может нанести ошибка в конкретном прикладном ПО, по крайней мере, ошибка не распространится на другие прикладные программы. Также программный интерфейс ОС общего назначения позволяет производить отладку одной прикладной программы другой, что позволило создать большое количество инструментов для отладки, предназначенных для разных целей и с разными возможностями. Классы таких инструментов как отладчики, профилировщики, динамические средства обнаружения ошибок сильно помогают во время разработки и тестирования.

В то же время разработка самой ОС происходит в более суровых условиях. В обычном режиме работы, когда ОС работает непосредственно на оборудовании, проверками на отсутствие ошибок занимается сама операционная система. Такой способ не подходит для этапа активной разработки из-за низкой скорости и возможности вывести из строя средства диагностики. Поэтому используются другие средства, позволяющие смоделировать поведение операционной системы.

Среди различных средств моделирования представляет интерес запуск операционной системы как прикладного приложения в другой ОС. Такой метод называют запуском в пользовательском режиме.

Этот метод обладает достоинствами и недостатками. Он позволяет использовать большой набор инструментов, предназначенных для отладки прикладных программ. С другой стороны, отлаживаемая ОС теряет некоторые свойства, например, свойство реального времени. Также, трудозатратным оказывается процесс подготовки к тестированию — переноса, который осуществляется под конкретную хост-ОС разработчика.

2. Постановка задачи

В данной дипломной работе передо мной стояли следующие задачи:

- Разработать организацию слоя аппаратных абстракций, позволяющей облегчить перенос
- Реализовать слой абстракций для запуска в пользовательском режиме ОС Linux

3. Обзор

3.1. Способы отладки

Существуют несколько способов отладки операционной системы, каждый обладает своими достоинствами и недостатками.

3.1.1. Отладочная печать

Самым простым способом отладки является отладочная печать. Способ заключается в выводе доступной информации об обнаруженных ошибках о состоянии системы или оборудования на специальное устройство ввода/вывода, к которому разработчик имеет доступ. В штатном режиме, когда ошибок не обнаружено, ввод со стороны разработчика не предусматривается, чтобы не вносить существенные задержки и не влиять на ход исполнения. Когда обнаруживается ошибка, разработчику может быть предоставлен интерактивный интерфейс, позволяющий проинспектировать состояние системы более детально.

Данный способ чрезвычайно полезен на ранних стадиях разработки, поскольку опирается только на драйвер любого устройства ввода/вывода. Поскольку он может быть очень простым, это позволяет использовать метод во время разработки на новом типе оборудования.

Недостатками метода является низкая производительность отладки. Если причину сбоя не удается обнаружить после сбоя, то приходится модифицировать систему для того, чтобы больше отладочной информации было доступно во время штатной работы. Это означает, что всякий раз при недостатке информации приходится проделывать цикл анализа данных, изменения кода ОС, компиляции, загрузки, воспроизведения ошибки.

3.1.2. Встроенные средства самодиагностики

Развитием метода отладочной печати являются встроенные средства самодиагностики. Они наследуют сильные и слабые стороны отладочной печати и призваны автоматизировать анализ состояния си-

стемы. Средства самодиагностики являются комплексным ПО, разработка которого несет накладные расходы и которые могут стать причиной снижения производительности системы. Примером такой системы является kmemcheck — подсистема ядра ОС Linux, в задачи которой входит анализ использования оперативной памяти и выявление доступа к неинициализированным участкам.

Методы самодиагностики хорошо применяется на поздних стадиях тестирования, таких как бета-тестирование. Это объясняется отсутствием необходимости разворачивать окружение для тестирования, возможностью отправить отчет разработчику.

С другой стороны, при использовании метода отладка производится из самой разрабатываемой системы. Сбой в ОС может затронуть средства диагностики и повлиять на корректность предоставляемых данных.

3.1.3. Полная виртуализация

Другим способом отладки является использование средств виртуализации, таких как виртуальные машины. Многие виртуальные машины предоставляют удаленный интерфейс отладки, позволяющий анализировать и изменять состояние виртуальной машины внешним инструментом.

Некоторые виртуальные машины гарантируют идентичное поведение реальному оборудованию, поэтому они могут использоваться во время разработки под аппаратное обеспечение, которое недоступно разработчику.

Интерфейс отладки предоставляет базовый набор функций, который принято ассоциировать с отладкой: установка точек останова, анализ памяти и т.д. Виртуальная машина и интерфейс отладки изолированы от исполняемой системы, т.е. ошибка ОС не исказит отладочную информацию.

С другой стороны, функциональность отладочного средства ограничена функциями, предоставляемыми интерфейсом виртуальной машины. В этот интерфейс не входят некоторые функции, такие как провер-

ка на исполнение инструкций в сегменте данных или разыменование нулевого указателя. Это является следствием того, что виртуальная машина не имеет модели исполняемой программы, а лишь имитирует аппаратную модель вычислителя.

Также отладочные инструменты должны явно поддерживать использование интерфейса удаленной отладки. Это существенно ограничивает выбор доступных инструментов.

3.1.4. Перенос в пользовательский режим

Еще одним способом отладки является перенос операционной системы в пользовательский режим. Он заключается в том, что в целевой операционной системе заменяется слой аппаратных абстракций на новый, который реализует операции взаимодействия с оборудованием через вызовы функций хост-ОС. В результате операционная система запускается и работает как обычное прикладное приложение с точки зрения хост-системы.

Этот способ является частными случаем паравиртуализации. Сама паравиртуализация редко используется для отладки ядра, поскольку часто подразумевает запуск на низкоуровневом гипервизоре, интерфейс отладки которого не отличается от интерфейса виртуальной машины.

Такой способ позволяет использовать инструменты для отладки прикладных программ в процессе разработки ОС. Также хост-ОС предоставляет обычные методы защиты, которые обнаруживают обращения по нулевому адресу, исполнение кода из секции данных, использование инструкций ввода/вывода (in и out на платформе x86).

В то же время, метод предполагает перенос на новую условно аппаратную платформу. Степень трудности переноса целиком зависит от архитектуры слоя аппаратных абстракций.

Помимо этого, производится отладка ядра ОС и HAL для пользовательского режима в целом. Для того, чтобы поведение ОС в пользовательском режим не отличалось от поведения на других платформах, требуется четко определить интерфейс слоя абстракций.

Также этим методом невозможно проверить некоторые характеристики, например, временные, поскольку время выполнения делится между отлаживаемой ОС и другими программами. Это нарушает свойства реального времени, т.к. время реакции на событие обуславливается еще и алгоритмами в хост-ОС.

Несмотря на все недостатки, данный метод отладки может эффективно применяться для поиска ошибок и проверки корректности работы ядра ОС.

3.2. HAL различных ОС

Рассмотрим HAL популярных операционных систем. Обзор будет проводиться свободных или открытых ОС. Основным моментом, на который будет обращаться внимание, будет архитектурная организация слоя аппаратных абстракций для архитектуры Intel x86 и его взаимодействия с кодом ядра.

3.2.1. Linux

Linux является свободной ОС (распространяется по условиям GPLv2). Данная система обладает монолитно-модульной архитектурой. ОС разрабатывается длительное время и используется на широком спектре оборудования, от серверов до встраиваемых решений. Рассматривалась версия системы 3.7.9[4].

Архитектурно-зависимые части ОС, относящиеся к x86, находятся в каталоге arch/x86. В его составе находятся следующие подкаталоги:

boot	Содержит части, относящиеся к загрузке. Входной точкой в архитектурно-зависимую загрузку является низкоуровневые функции, отвечающие за загрузку образа ядра, а затем передающие управление функции из состава файла main.c. В нем определяется следующий порядок инициализации: • Инициализировать раннее устройство ввода/вывода • Инициализировать кучу для динамической памяти • Проверить возможности процессора, установить его режим и карту памяти • Инициализировать клавиатуру и экран
include	Заголовочные файлы, относящиеся к слою абстракций, и разнородные реализации интерфейса HAL inline-функциями и макро-определениями, среди них, например, функции преобразования порядка байт в слове
kernel	Основная реализация HAL, содержит, помимо прочего, реализацию целой функции печати — аналога printf, интерфейса удаленной отладки; драйвера для систем управления питанием acpi и apm, нескольких системных таймеров; примитивные драйвера VGA и последовательного порта, шины PCI и т.д.
lib	Оптимизированные библиотечные функции общего назначения
net	Оптимизированные библиотечные функции сетевой подсистемы.

Остальные подкаталоги содержат драйвера архитектурно-специфичных подсистем и функций.

Архитектурно-независимая часть инициализации ОС, расположенная в init/main.c, в начале работы последовательно инициализирует HAL: системный таймер, MMU, контроллер прерываний. [5][1]

В рамках проекта uClinux было создана операционная система на базе GNU/Linux, включающая, в том числе, модифицированное ядро ОС Linux, позволяющие работу на процессорах без MMU. Такая поддержка осуществляется специальным драйвером, не управляющим аппаратным MMU, а лишь поддерживающим собственное состояние и адекватно реагирующим на запросы высокоуровневых частей ядра ОС. Начиная с версии ядра 2.6 uClinux был объединен с основной веткой разработки ядра Linux.

Таким образом, можно утверждать, что архитектура ОС Linux предполагает наличие полностью реализованного и функционирующего слоя аппаратных абстракций.

3.2.2. NetBSD

Операционная система NetBSD произошла из семейства BSD. ОС поддерживает не менее 57 аппаратных платформ. Одной из её отличительных особенностей является слой аппаратных абстракций, который скрывает механизмы общения с аппаратным обеспечением по различным шинам и позволяет драйверам устройств состоять из машинно-зависимой и машинно-независимой частей. Перенос системы на другую архитектуру с похожим аппаратным окружением облегчается тем, что высокоуровневая часть драйвера остается неизменной.

Далее на предмет архитектурных особенностей слоя аппаратных абстракций рассматривалась NetBSD версии 6.0. [6]

Архитектурно-зависимые части расположены в подкаталоге arch. Подкаталог arch/i386 содержит исходные коды, относящиеся к типу процессоров Intel x86. Среди них таблица прерываний, загрузочный код, относящийся к разным способам загрузки, драйверы шин PCI и ISA.

В подкаталоге arch/x86 содержатся драйвера типичных устройств платформы Intel x86, среди них драйвера системного таймера, контроллера прерываний, прочих устройств, расположенных на PCI и ISA шинах.

Как видно, NetBSD имеет четкое разделение программного кода на

поддержку процессорной архитектуры и драйвера устройств, которые обязательно присутствуют на платформе.

В тоже время, в отличие от рассмотренной ранее ОС Linux, NetBSD не может быть собрана без поддержки MMU.

В отличие от ОС Linux, NetBSD имеет более явное разделение исходного кода на платформо-зависимые и платформо-независимые части. В то же время, основу слоя аппаратных абстракций NetBSD составляет набор исходных кодов, обязательно входящих в сборку и демонстрирующих лишь базовое разделение по функциям и интерфейсам, которая используется только внутри HAL, но не предоставляет эту информацию более высокоуровневой части ядра этой ОС.

Таким образом, слой аппаратных абстракций рассмотренных ОС представляет собой один модуль с четко специфицированным интерфейсом, покрывающим большое число функций, начиная от определения типа процессора и заканчивая драйверами некоторых платформоспецифичных устройств.

3.3. Проекты по переносу ОС в пользовательский режим

Операционные системы Linux и NetBSD имеют возможность запуска в пользовательском режиме, проекты по переносу называются User-mode Linux и NetBSD/usermode соответственно.

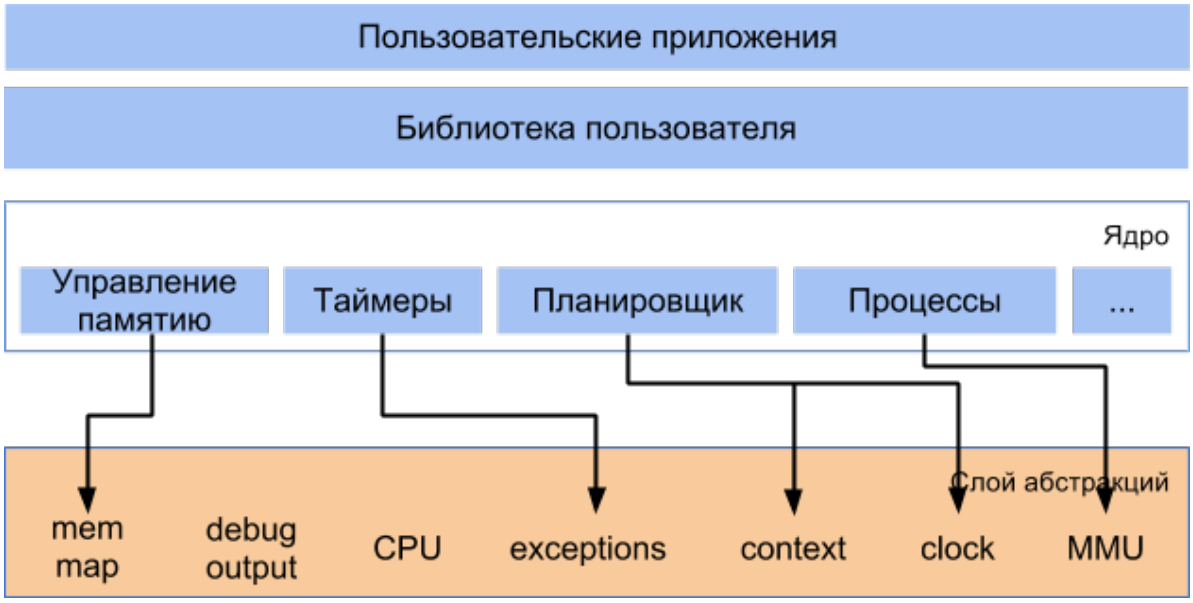
Оба проекта преследуют несколько целей, начиная от паравиртуализации и заканчивая отладкой и профилированием.

Оба проекта используют похожую структуру, при которой ядро целевой ОС линкуется с libc хост-ОС. При этом целевые ОС используют не реальное оборудование, а файловый интерфейс к устройствам, предоставляемый хост-ОС. Прерывание от устройства имитируются POSIX сигналом SIGIO, который генерируется хост-ОС. Поддержка виртуальной памяти также реализована через вызов хост-ОС. В качестве операции смены контекста процессора NetBSD/usermode использует вызовы setcontext/switchcontext из состава libc, в то время, как в UML исполь-

зуется собственный механизм.[2][7]

В общем, проекты по переносу ОС Linux и NetBSD в пользовательский режим похожи по структуре, за исключением способа взаимодействия слоя аппаратных абстракций с ядром ОС.

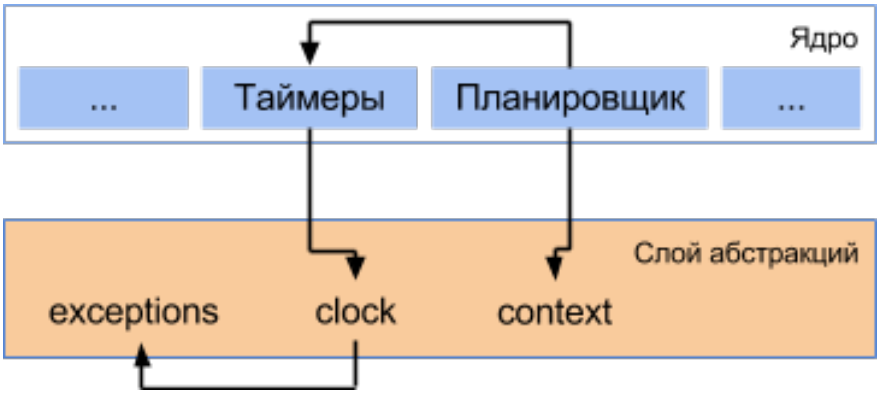
4. Архитектура HAL



Как уже упоминалось, ядро ОС состоит из различных подсистем, которые работают одинаково, независимо от оборудования.

Для этого они опираются на слой аппаратных абстракций, который предоставляет на разных платформах одинаковое программное окружение, на котором строятся остальные высокоуровневые части ОС.

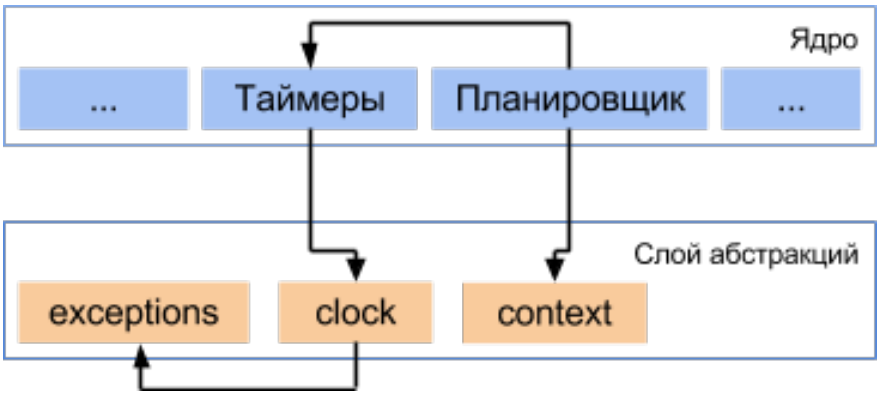
Очень мало подсистем ядра опирается на все интерфейсы слоя аппаратных абстракций сразу. Большинство использует только несколько групп интерфейсов. При этом некоторые группы функций используют другие интерфейсы слоя абстракций.



Исходя из этого, для обеспечения работоспособности некоторых подсистем достаточно неполной реализации слоя аппаратных абстракций.

В ходе дипломной были определены зависимости подсистем ядра от интерфейсов слоя абстракций. В результате, интерфейсы были разби-

ты по модулям, каждый из которых реализует несколько близких по смыслу функций, а в сумме они реализуют полный интерфейс.

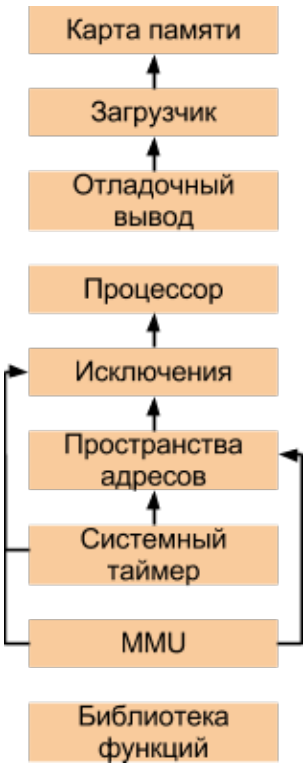


При этом каждый слой аппаратных абстракций для различных платформ структурно так же состоит из модулей.

Такая архитектура обладает преимуществом по сравнению с классической.

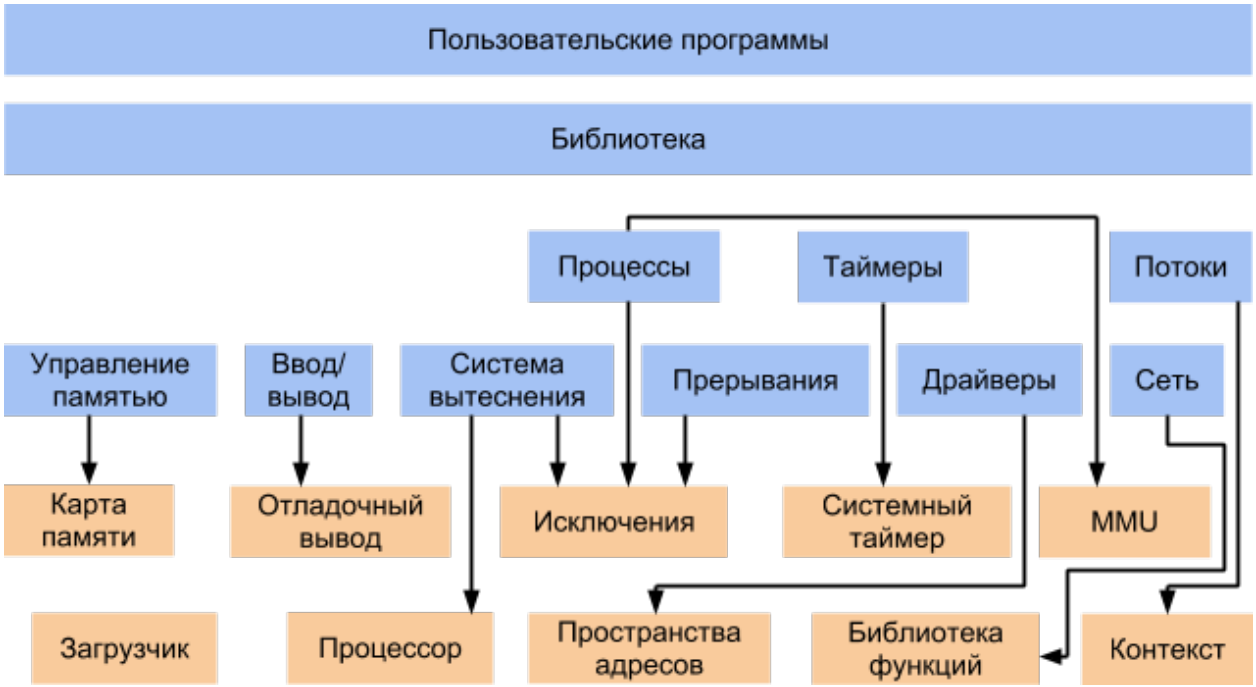
Во-первых, новая структура позволяет неполную реализацию слоя абстракций. Это достигается с помощью модифицированной процедуры сборки и инициализации с модульным слоем абстракций.

Во-вторых, на основе этой архитектуры можно получить порядок реализации интерфейсов слоя абстракций, который опирается на зависимости между модулями слоя абстракций.



В целом, эти свойства существенно помогают во время переноса операционной системы на новую платформу.

Таким образом, структура взаимодействия ядра и модульного слоя аппаратных абстракций выглядит таким образом.



4.1. Модули

В представленной архитектуре непосредственно слой аппаратных абстракций состоит из необходимых модулей, т.е. включает минимум драйверов устройств. Помимо уменьшения программного кода HAL это позволит переиспользовать высокоуровневую часть драйверов на разных платформах.

В результате работы были выделены следующие модули слоя аппаратных абстракций. Они перечислены согласно зависимостям, т.е. их порядок соответствует порядку реализации во время переноса на новую платформу. Описание модуля содержит описание интерфейсов, которые должны реализоваться модулем.

4.1.1. Карта памяти

Первым модулем из состава слоя аппаратных абстракций, который необходимо реализовать, является карта памяти. Она определяет положение секции данных, программных инструкций, стека, кучи ядра, вектора прерываний и возможных областей ввода/вывода. Использование одной карты памяти для всех перечисленных регионов памяти позволяет следить за отсутствием их пересечений, проверять ограничения на размер, как в меньшую, так и в большую сторону.

Карта памяти используется не только во время работы ОС, но и во время компоновки финального образа ядра ОС.

4.1.2. Загрузчик

Обычно в начальный момент загрузки ОС процессор, в регистре счетчика команд содержит заранее известный адрес первой инструкции, стартовый адрес. Содержание других регистров не определено.

Поэтому слой аппаратных абстракций содержит модуль загрузчика, отвечающий за первичную инициализацию, подготовку к запуску инициализации ОС.

Загрузчик размещается таким образом, чтобы его первая инструкция располагалась по стартовому адресу. Так, во время загрузки систе-

мы управление передается загрузчику. В его задачи входит, во-первых, инициализация регистров. Среди них указатель стека, который инициализируется в конец области, отведенной под стек. Затем следует инициализация режима процессора, выключение прерываний и другие функции обязательной подготовки, например, выключение сторожевого таймера.

После этого загрузчик размещает секции данных и кода ядра ОС, обнуляет область памяти неинициализированных переменных и передает управление функции инициализации ядра ОС.

4.1.3. Отладочный вывод

Следующим модулем, который должен быть реализован во время переноса является драйвер устройства отладочного вывода. Его функции состоят в возможности вывести один символ на устройство вывода, которое доступно разработчику.

Драйвер отладочного вывода создается как можно более простым, не использующим какие-либо приемы оптимизации. Например, отладочным устройством может быть COM-порт, тогда процесс вывода символа состоит в том, чтобы проверить, что в аппаратном буфере есть место для нового символа, если так, то записать его в буфер, иначе ждать, пока место не появится. Ожидание в таком случае должно быть реализовано постоянной проверкой свободного места и не может использовать ни прерывания от COM-порта, ни таймеры (поскольку соответствующие модули HAL могут быть не загружены или еще не реализованы).

Как уже отмечалось, отладочный вывод является важным средством отладки, поэтому во время загрузки ядра ОС он инициализируется одним из первых.

4.1.4. Процессор

Следующим этапом переноса является реализация модуля процессора. Он содержит описание порядка байтов в слове процессора, а также

определяет доступ к слову состояния процессора.

В слове состояния находится флаг маскирования прерываний, который позволяет включать или отключать обработку прерываний на процессоре. Отключение прерываний является способом реализаций критических секций при использовании одного вычислительного ядра, что и происходит на ранних стадиях загрузки.

В этом же модуле находятся функции перевода процессора в режим пониженного энергопотребления.

4.1.5. Пространства адресов

Задачей драйвера устройства является взаимодействие с аппаратным обеспечением, т.е. получение информации о текущем статусе устройства и его изменение. Это происходит чтением и записью регистров устройства — специальных переменных в аппаратном обеспечении, которые определяют режим работы.

Таким образом, для разных пространств адресов должен быть определен способ доступа, позволяющий получить и записать значение из требуемого регистра, расположенного по известному адресу.

Для платформы Intel x86 должны быть реализованы способы доступа к двум адресным пространствам.

Шина ввода/вывода подключается напрямую к процессору и используется для программирования устройств, характерных для платформы. Для доступа к шине используются специальные инструкции процессора: in — для чтения данных из адреса шины и out — для записи, поэтому, требуется определить операции на шине через эти инструкции.

Доступ к отображенным в память регистрам можно обеспечить чтением и записью по определенными адресам в оперативной памяти, при этом используются механизмы установки барьеров памяти, что гарантирует корректность. [3]

4.1.6. Исключения

Следующим этапом переноса является реализация модуля поддержки исключений процессора, прерываний и пользовательских исключений. Модуль состоит из нескольких частей.

Во-первых, модуль должен содержать таблицу прерываний и исключений. Эта таблица размещается в определенном месте и состоит из адресов процедур обработки исключительных ситуаций и прерываний. Когда происходит исключение, процессор сохраняет минимальную часть контекста исполнения, по типу исключения вычисляет смещение в этой таблице и передает управление процедуре обработки, адрес которой находится в таблице по смещению.

Также в модуле должны находиться процедуры начала обработки исключений трех типов, которые подготавливают переход к архитектурно-независимым функциям обработки.

Драйвер контроллера прерываний является необходимым компонентом для поддержки прерываний. В его функции входит управление типами аппаратных прерываний, позволяющее разрешить или запретить аппаратному прерыванию с определенным номером по возникновению обрабатываться на вычислительном ядре.

Также интерфейс может включать функции принудительного выставления и очистки ожидающих прерываний, но их реализация может отсутствовать из-за аппаратных особенностей конкретного контроллера прерываний.

4.1.7. Системный таймер

Драйвер системного таймера является следующим этапом в переносе операционной системы. Устройство системного таймера позволяет генерировать прерывания через заданный промежуток времени и используется ОС для отсчета времени в программных таймерах, оценки времени и т.д.

Модуль системного таймера должен позволять инициализировать устройство, что подразумевает установку обработчика прерываний, на-

стройку разрешения временных отсчетов в зависимости от системной частоты и т.п. Также интерфейс драйвера должен поддерживать установку временного интервала, после которого однократно или многократно будет возникать прерывание, означающее истечение интервала.

4.1.8. Переключение контекстов

Для обеспечения возможности иметь несколько потоков исполнения ОС опирается на модуль переключения контекстов вычислительного ядра. В его состав входит определение контекста и функция переключения.

Контекст — это архитектурно-зависимая структура, описывающая состояние процессорного ядра. Структура контекста содержит все доступные регистры, в том числе регистры счетчика команд и указателя стека.

Задачей функция переключения контекста является сохранение текущего контекста, а затем загрузка сохраненного ранее. Из-за того, что ни текущий, ни загружаемый контекст не должны быть изменены, функция переключения эксплуатирует низкоуровневые возможности архитектуры процессора и поэтому находится в слое аппаратных абстракций.

4.1.9. MMU

Поддержка изолированных адресных пространств процессов в операционных системах опирается на устройство MMU (Memory Managment Unit). Модуль слоя аппаратных абстракций взаимодействия с MMU должен содержать несколько частей.

Во-первых, модуль должен содержать управление кэшем TLB (Translation Lookahead Buffer), который служит для ускорения работы MMU. Интерфейс определяет возможность сбросить кэш отображения данных или инструкций.

Во-вторых, модуль должен содержать определение контекста памяти, содержащего описание адресного пространства одного процесса.

Также в модуле должны находиться операции работы с контекстом, позволяющие модифицировать его согласно общей модели: установить в контексте новое отображение физической памяти, получить отображение из контекста, назначить разрешения для отображения на исполнение, запись и т.п.

4.1.10. Библиотека функций

Библиотечный модуль предоставляет набор утилитарных функций, оптимизированных для конкретной платформы. Примером может быть аппаратная реализация функции test-and-set.

В случае, если какая-либо операция не реализована на конкретной аппаратной платформе, то ядро использует функцию из собственной библиотеки, не зависящей от платформы.

5. Реализация

Предложенная модульная архитектура была апробирована в операционной системе Embox. Выбор был обоснован возможностью описывать модули сборки произвольной природы, в которых удалось достаточно просто выразить модули слоя аппаратных абстракций.

Данная ОС имеет возможность запуска на платформах Intel x86, SPARC, ARMv7, MIPS, Microblaze, PowerPC. В ходе работы существующие слои аппаратных абстракций для платформ были переорганизованы под модульную структуру.

Для апробации реализации под новую платформу был выбран пользовательский режим ОС GNU/Linux на платформе Intel x86, который еще не был реализован. Перенос в пользовательский режим позволит использовать некоторые средства отладки, недоступные прежде.

5.1. Модульная структура слоя абстракций

Для того, чтобы позволить неполную реализацию, модули представлены как отдельные единицы компиляции, при этом сборка операционной системы является успешной и при отсутствии некоторых модулей.

Система инициализации явным образом проверяет наличие каждого следующего модуля HAL. В случае, если он присутствует, система инициализирует его, запускает встроенные средства тестирования данного модуля и все подсистемы ядра, которые опираются на него и уже инициализированные модули. В случае, если следующий модуль слоя абстракций отсутствует, ядро ОС заканчивает свою процедуру инициализации. Так подсистемы ядра, которые уже были инициализированы, продолжают работу.

Для предоставления информации о доступных и недоступных модулях слоя абстракции используется объектно-ориентированный подход. Система инициализации ядра содержит слабые указатели на структуры, каждая из которых соответствует одному из всевозможных модулей слоя. Каждая единица сборки содержит неизменную инициализированную структуру, содержащую набор операций с данным модулем.

Во время компоновки, слабые указатели разрешаются на структуры включенных модулей, указатели на не включенные модули получают определенное значение NULL. Перед каждым обращением к операциям модуля ядро ОС сначала проверяет значение слабого указателя, в случае NULL значения модуль считается отсутствующим, иначе, ядро из структуры получает указатель на нужную операцию.

Для увеличения производительности, некоторые частые операции, такие как взаимодействие через пространства адресов, определяются не через косвенную адресацию, а через макроопределения. Ядро ОС содержит набор макроопределений, реализующих минимальный интерфейс для отсутствующих модулей. Каждая единица сборки, в свою очередь, содержит макроопределения, перекрывающие определения ядра и реализующих требуемый интерфейс слоя абстракций данного модуля.

5.2. HAL для пользовательского режима

Как уже упоминалось, слой аппаратных абстракций для пользовательского режима должен позволять запускать целевую операционную систему как приложение в хост-ОС, которой в данном случае является ОС GNU/Linux.

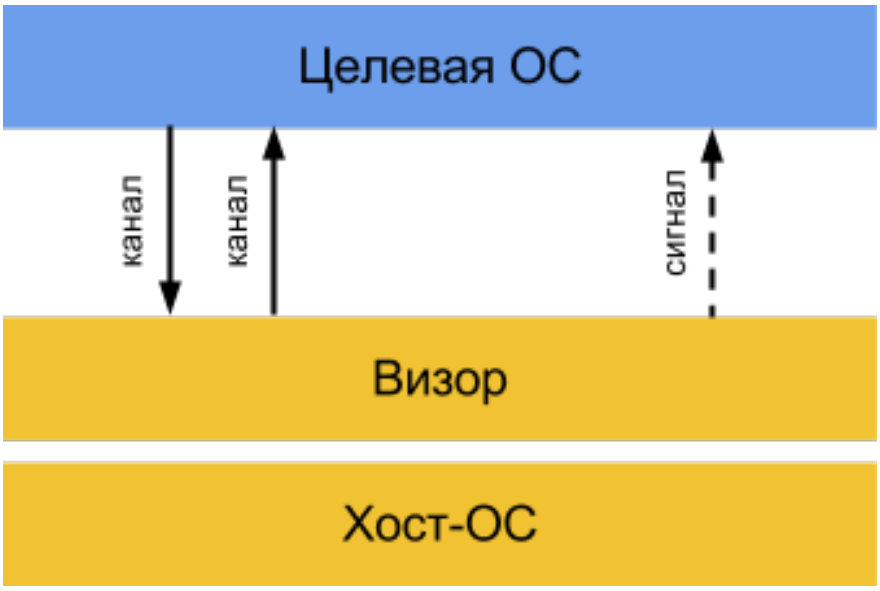
Одной из характеристик платформы пользовательского режима является то, что некоторые функции слоя абстракций и ядра уже предоставлены хост-системой. Например, загрузка различных секций в память происходит без участия целевой ОС, также предоставляются выделение памяти или терминал ввода/вывода. Естественно, что некоторые из них перекрывают функции целевой системы. Задачей переноса является сохранение как можно большего количества подсистем ядра и модулей слоя абстракций целевой ОС во время запуска в пользовательском режиме, чтобы иметь возможность их протестировать.

Рассмотренные ранее ОС в пользовательском режиме для взаимодействия с хост-системой используют специфический вариант компоновки со стандартной библиотекой. Получившийся образ ОС представляет собой приложение, загружающееся стандартным образом.

Во время реализации слоя для пользовательского режима я избрал другой способ взаимодействия. Основной характеристикой является модель работы, которая гораздо ближе к работе виртуальной машины, чем к пользовательскому приложению.

Для взаимодействия с хост-системой используется выделенная сущность — визор. Это приложение хост-системы, которое преобразует сообщения-запросы от целевой ОС в вызовы функций хост-ОС и возвращает результат в сообщениях, сходных с исключениями на аппаратных платформах.

Для взаимодействия визора и целевой системы могут использоваться любые стандартные средства межпроцессного взаимодействия, позволяющие передавать данные произвольной природы и сгенерировать исключение в целевой ОС из другого процесса. Например, для ОС Linux такими средствами являются именованные каналы и POSIX сигналы.



На основе этих средств в слое абстракции построен механизм передачи сообщений, что позволяет потенциально переносить решение по запуску в пользовательском режиме на другие ОС, например, Microsoft Windows. В таком случае модификации затронут реализацию межпроцессного взаимодействия в слое абстракций, где они составляют малую часть, и визор, который является простым приложением хост-ОС, что не должно составить трудностей.

Такая организация характеризуется наиболее близким соответствием с аппаратной платформой, что позволяет задействовать максимальное количество программного кода, который будет использоваться на любой аппаратной платформе. Использование двух процессов для работы позволяет отлаживать именно ядро ОС, оставляя взаимодействие с хост-ОС вне фокуса. С другой стороны, использование нескольких процессов и межпроцессного взаимодействия влияет на производительность.

5.3. Модули слоя абстракций для пользовательского режима

5.3.1. Карта памяти

Карта памяти определяется для пользовательского режима на основе обычной карты памяти прикладных приложений. Для ОС Linux на Intel x86 определяются следующие адреса: программный код и неизменяемые данные располагаются с адреса 0x10000, для конкретной сборки ОС достаточно 512 kb зарезервированного размера; изменяемые данные, инициализированные и неинициализированные, а так же программный стек располагаются в секции, начинающейся с адреса 0x8000000 размером 2Mb.

5.3.2. Загрузчик

До передачи управления на загрузчик хост операционная система размещает секции образа целевой ОС по отведенным для них адресам и устанавливает стек в верхних адресах адресного пространства.

Задачей загрузчика является инициализация межпроцессного взаимодействия для общения с визором и установка указателя стека на конец отведенного для него пространства.

Инициализация межпроцессного взаимодействия заключается в открытии нисходящего именованного канала на чтение, восходящего на запись и посылке сообщения визору, в котором содержится идентифи-

катор процесса целевой системы.

Установка стека происходит после инициализации взаимодействия с визором, поскольку этот процесс может использовать стек большего размера, чем отведенный размер под системный стек в целевой ОС.

5.3.3. Отладочный вывод

Отладочный вывод реализуется через сообщение визору, в котором содержится текстовая информация. Визор при обработке такого сообщения печатает его содержимое.

5.3.4. Процессор

В модуле поддержки процессора определяется little-endian порядок байт, который используется на архитектуре Intel x86.

Поскольку исключения реализованы посредством сигналов, аналогом маскирования прерываний является активация и деактивация обработчика сигналов.

Перевод процессора в спящий режим реализуется через системную функцию pause, которая приостанавливает текущий процесс до получения сигнала.

5.3.5. Пространства адресов

В данной архитектуре определено только одно адресное пространство, в котором находятся регистры устройств, предоставляемых визором.

Установка значения в адресном пространстве происходит посредством посылки сообщения и его запоминания. При обращении на чтение возвращается ранее установленное значение либо ноль.

5.3.6. Исключения

Исключения реализуются по-разному для различных типов.

Исключения ошибок и программные исключения происходят в целевой ОС, поэтому для единообразия сообщение генерируется локально, а функция обработки сообщений вызывается напрямую.

Прерывание посылается визором в виде сообщения, которое содержит номер прерывания. Функция обработки целевой ОС передает такие сообщения на обработку драйверу контроллера прерываний, который содержит таблицу маскированных и немаскированных прерываний.

5.3.7. Системный таймер

Системный таймер реализуется через систему timerfd операционной системы Linux.

По сообщению-запросу в визоре происходит настройка timerfd на генерирование событий через заданные промежутки времени. Эти события затем преобразуются в прерывания целевой ОС.

5.3.8. Переключение контекстов

Переключение контекстов реализовано с помощью функций makecontext и swapcontext, предоставляемых хост-ОС.

5.3.9. MMU

Блок управления MMU на платформе не реализован. Благодаря новой архитектуре слоя абстракций стало возможно функционирование и при отсутствии этого модуля.

5.4. Запуск ОС в пользовательском режиме

Операционная система Embox была запущена под отладчиком gdb. Во время запуска хост-системой были обнаружены обращения к адресам памяти, не определенных в карте памяти, а также обращения по нулевому адресу.

Использование анализатора обращения к памяти valgrind обнаружило использование неинициализированного значения. Также были реа-

лизованы функции взаимодействия с ядром valgrind, которые позволило встроить функции обнаружения утечек памяти в системные механизмы выделения памяти ОС.

Стало возможно использовать инструмент perf для профилирования работы функций ядра, который, помимо прочего, позволяет замерить количество промахов процессорного кеша, а также количество успешно предсказанных переходов относительно общего количества переходов.

Получившееся решение из-за выбранной архитектуры обладает меньшей производительностью по сравнению с использованием виртуальной машины. Для примера использовалась отправка с хост-ОС сетевого запроса ICMP и ожидание подтверждения. Среднее время отклика виртуальной машины составило 0.038 миллисекунд, среднее время отклика ОС в пользовательском режиме - 0.160 миллисекунд.

6. Результаты

В ходе работы была разработана архитектура слоя аппаратных абстракций, облегчающая процесс переноса на новую платформу указанием порядка реализации модулей слоя аппаратных абстракций.

Архитектура слоя абстракций апробирована во время переноса ОС в пользовательский режим. Помимо этого, добавлена возможность использовать инструментальные средства, такие как gdb, valgrind и perf для отладки ядра. Их применение обнаружило несколько ошибок в программном коде ядра ОС Embox.

Список литературы

[1] Daniel P. Bovet Marco Cesati Ph.D. Understanding the Linux Kernel. — O'Reilly Media, 2005.

[2] Dike Jeff. User Mode Linux. — Prentice Hall, 2006.

[3] Jonathan Corbet Alessandro Rubini Greg Kroah-Hartman. Linux Device Drivers. — O'Reilly Media, 2005.

[4] Linux Kernel Organization Inc. Linux source 3.7.9 // http://kernel.org. — 2012. — URL: `https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.7.9.tar.xz` (online; accessed: 30.05.2013).

[5] Love Robert. Linux Kernel Development. — Addison-Wesley Professional.

[6] The NetBSD Foundation Inc. NetBSD kernel source 3.7.9 // http://netbsd.org. — 2012. — URL: `ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-6.0/source/sets/syssrc.tgz` (online; accessed: 30.05.2013).

[7] Zandijk Reinoud. NetBSD/usermode. — EuroBSD, 2012.

[8] Таненбаум Э. Современные операционные системы. — СПб. : Питер, 2010.