ПРОГРАММИРОВАНИЕ, 2019, № 6, с. 23–29

ОПЕРАЦИОННЫЕ СИСТЕМЫ

УДК 004.451.3

АРХИТЕКТУРА, РЕАЛИЗАЦИЯ И ИНТЕРФЕЙС ЯДРА ПОДСИСТЕМЫ ВВОДА-ВЫВОДА ЧЕРЕЗ ПОРТЫ ДЛЯ ИСТИННЫХ МИКРОЯДЕР НА ПРОЦЕССОРАХ IA-32

© 2019 г. Е. И. Клименков*,**

Белорусский государственный университет информатики и радиоэлектроники
220013 Минск, ул. П.Бровки, 6, Белоруссия
**E-mail: klimenkov@bsuir.by*
***E-mail: Evgeny.Klimenkov@gmail.com*

Поступила в редакцию 20.06.2018 г.
После доработки 25.12.2018 г.
Принята к публикации 05.03.2019 г.

Управление доступом к средствам ввода-вывода компьютерной системы является одной из основных функций ядра операционной системы. В данной статье предлагается новая архитектура подсистемы управления вводом-выводом через порты, подходящая для применения в истинных микроядрах (микроядрах второго поколения). В частности, предлагаемая архитектура рассматривается на примере процессоров с архитектурой набора команд IA-32, как широко распространенной архитектуры набора команд с поддержкой концепции портов ввода-вывода. Также, в данной работе представлен и описан протокол доступа к средствам ввода-вывода через порты реализованные в ядре ОС, и использующийся драйверами пользовательского режима, а также соответствующий набор оптимизаций, реализованных на стороне ядра.

DOI: 10.1134/S0132347419060037

1. ВВЕДЕНИЕ

Ввод-вывод через порты (ВВчПорты) в компьютерных системах является одним из основных способов организации ввода-вывода. ВВчПорты является неотъемлемой частью архитектуры IBM PC в целом и процессоров с архитектурой набора команд IA-32 [1] в частности, и, следовательно, должен поддерживаться операционной системой.

В отличие от ввода/вывода через память (ВВчПамять), механизм ВВчПорты не является дружественным по отношению к архитектуре микроядра. Блок управления памятью процессора предоставляет удобные средства для реализации детального контроля доступа к ВВчПамять, которые позволяют эффективно осуществлять разделение и изоляцию доступа к устройствам ввода-вывода между процессами драйверов пользовательского режима. В связи с этим, подсистема управления памятью ядра может быть также использована и для организации управления ВВчПамять, а операции ВВчПамять могут выполняться драйверами пользовательского режима с минимальными накладными расходами. К сожалению, архитектура набора команд IA-32 не предоставляет подобных мощных средств для управления ВВчПорты. В результате, разработчики микроядер вынуждены реализовывать специальную поддержку ВВчПорты, ища компромис между эффективностью осуществления ВВчПорты, обеспечением детального контроля доступа к портам, эффективностью использования ресурсов компьютерной системы и простотой/минималистичностью реализации. В данной статье описывается новый подход к управлению ВВчПорты, основанный на ВВчПорты опосредованном ядром операционной системы и ориентированный на производительность. Данный подход опирается на расширение набора операций ввода-вывода и их пакетную обработку.

Данная статья начинается с обсуждения существующих подходов к организации ВВчПорты в микроядрах, после которого предлагается новое решение к данной проблеме.

2. СУЩЕСТВУЮЩИЕ ПОДХОДЫ К УПРАВЛЕНИЮ ВВОДОМ-ВЫВОДОМ ЧЕРЕЗ ПОРТЫ

Концепция портов ввода-вывода является старой, но немаловажной частью подсистемы ввода-вывода платформы IA-32. Несмотря на ряд недостатков и тенденцию постепенного отказа от ВВчПорты в пользу ВВчПамять, порты ввода-вывода продолжают играть важную роль в современ-

ных компьютерных системах. Прежде всего, они являются частью фактического стандарта IBM PC. Во-вторых, архитектура стандартной системной шины современных компьютеров PCI/PCI Express также полагается на ВВчПорты. В связи с этим, операционные системы, предназначенные для платформы IA-32, вынуждены обеспечивать управление доступом к портам ввода-вывода.

2.1. Архитектура ВВчПорты на платформе IA-32

Подсистема ВВчПорты на платформе IA-32 состоит из следующих компонентов:

• Отдельное адресное пространство ввода-вывода.

• Семейство специальных инструкций ВВчПорты: *in** и *out**.

• Поле уровня привилегий ввода-вывода (ПУПВВ) в регистре EFLAGS.

• Битовая карта разрешенных портов ввода-вывода (БКРПВВ) в сегменте состояния задачи.

IA-32 реализует отдельное 16-битное адресное пространство ввода-вывода, к которому можно получить доступ только с помощью специально предназначенного семейства инструкций: *in** и *out**. ПУПВВ обеспечивает глобальный контроль доступа процесса к адресному пространству ввода-вывода, а также, что немаловажно, к управлению приемом прерываний (*cli* и *sti*). БКРПВВ же, в свою очередь, обеспечивает детальный контроль доступа к заданным портам ввода-вывода.

2.2. Управление ВВчПорты через привилегированные домены

Операционные системы на базе монолитного ядра, такие как Linux [2], разделяют все программное обеспечение, выполняющееся в системе на два домена: привилегированный и непривилегированный. Привилегированное программное обеспечение считается надежной и доверенной, и имеет неограниченный доступ ко всей системе и всем функциям центрального процессора, включая неограниченный доступ к ВВчПорты. Такие ОС предполагают, что приложение предполагающее использование ВВчПорты, должно быть реализовано как привилегированный модуль ядра. В результате, эти системы не используют механизмы ПУПВВ и БКРПВВ для управления ВВчПорты.

Несмотря на свою простоту и эффективность, данный подход является неприемлемым для систем на основе микроядра, построенных с учетом принципа минимализма Лидтке [9], и, следовательно, реализующих драйвера в виде процессов пользовательского режима.

2.3. Доверенные потоки ввода-вывода пользовательского режима

Концепция доверенных потоков ВВчПорты опирается на использование ПУПВВ. Согласно этому подходу, операционная система поддерживает специальный тип потоков пользовательского режима, которым разрешено выполнять операции ВВчПорты напрямую. Ядро выставляет разрешающее значение в поле ПУПВВ регистра EFLAGS в контексте таких потоков.

Данный подход является одновременно простым и эффективным, но в то же время, он обеспечивает слабую защищенность системы от вредоносных драйверов. Поскольку доверенные потоки имеют неограниченный доступ ко всему адресному пространству ввода-вывода, они способны вмешиваться в работу друг друга, и, что даже более опасно, полностью останавливать всю систему за счет отключения приема прерываний. Тем не менее, некоторые ОС следуют этому подходу в управлении ВВчПорты [3].

2.4. Управляемый прямой ВВчПорты в пользовательском пространстве

Данный подход распространен в микроядрах первого поколения. Согласно ему ядро контролирует и ограничивает доступ к адресному пространству ввода-вывода на уровне потоков с помощью БКРПВВ. Преимущество данного подхода заключается в том, что операции ВВчПорты могут выполняться драйвером в пользовательском пространстве непосредственно без участия ядра, а значит с минимальными накладными расходами в терминах процессорного времени. Недостатком же является то, что он усложняет архитектуру и реализацию ядра, приводит к потреблению значительных ресурсов адресного пространства ядра и физической памяти. В дополнение к этому, данный подход приводит к деградации производительности переключений контекстов потоков и/или замедляет операции поиска блока управления потоком. Тем не менее, данный подход к управлению ВВчПорты используется в таких операционных системах, как Debian GNU/Hurd [4] и HelenOS [5].

2.5. ВВчПорты опосредованный ядром операционной системы

Второе поколение микроядер использует иную стратегию для управления ВВчПорты. Данная стратегия основана на отказе от использований аппаратных возможностей процессора для обеспечения контроля доступа при ВВчПорты и обеспечивает доступ к операциям ВВчПорты исключительно через системные вызовы. Согласно данному подходу интерфейс к каждой элементарной операции ВВчПорты экспортируется ядром в

драйверы пользовательского режима. Ядро выполняет управление доступом к ВВчПорты на программном уровне и непосредственно выполняет разрешенные запросы на ввод-вывод. Хотя ВВчПорты опосредованный ядром операционной системы прост в реализации и эффективен с точки зрения использования физической памяти и адресного пространства ядра, данная стратегия сопряжена со значительными накладными расходами процессорного времени за счет необходимости в дополнительных ресурсах для осуществления системных вызовов, поскольку каждый акт элементарного доступа к портам ввода-вывода (чтение значения из порта и запись значения в порт) требует выполнения отдельного системного вызова. Наиболее известные исследовательские микроядра, такие как Barrelfish [6], Minix 3 [7] и seL4 [8], следуют данному подходу.

3. ПРЕДЛАГАЕМЫЙ ПОДХОД К ОРГАНИЗАЦИИ УПРАВЛЕНИЯ ВВОДОМ-ВЫВОДОМ ЧЕРЕЗ ПОРТЫ

Предлагаемый подход к управлению ВВчПорты рассматривает второе поколение микроядер в качестве целевой архитектуры. В связи с этим, в качестве его основы был выбран ВВчПорты опосредованный ядром операционной системы. Основным достоинством предлагаемого подхода является сокращение накладных расходов на операции ВВчПорты, присущих оригинальному подходу, с сохранением всех его преимуществ. В качестве основного метода сокращения накладных расходов было выбрано сокращение числа системных вызовов, связанных с выполнением операций ВВчПорты. Такое сокращение может быть достигнуто путем мультиплексирования нескольких запросов ВВчПорты в одном системном вызове, в тех случаях когда это возможно с точки зрения реализации драйвера устройства.

В следующих разделах сначала будет обсуждаться набор основных операций ВВчПорты и его расширение. Затем последует обсуждение упаковки переменного количества запросов ВВчПорты в один системный вызов. И наконец, будет описана полная архитектура портового ввода-вывода основанного на предлагаемом подходе.

3.1. Расширенный набор операций ВВчПорты

IA-32 обеспечивает две основные операции ВВчПорты: чтение значения и запись значения. Принимая во внимание тот факт, что существует три вида портов ввода-вывода: 8-разрядные, 16-разрядные и 32-разрядные, и что для работы с портами разных видов используются различные процесорные инструкции, мы получаем шесть основных операций в целом:

1. Чтение одного байта из порта ввода-вывода.
2. Чтение двух байт из порта ввода-вывода.
3. Чтение четырех байт из порта ввода-вывода.
4. Запись одного байта в порт ввода-вывода.
5. Запись двух байт в порт ввода-вывода.
6. Запись четырех байт в порт ввода-вывода.

Это стандартный набор операций, который экспортируется микроядрами второго поколения в пользовательский режим в виде одного мультиплексированного или нескольких отдельных системных вызовов.

Один из распространенных паттернов доступа к адресному пространству ввода-вывода является чтение-изменение-запись. Во многих случаях порт ввода-вывода отображается на регистр устройства, который содержит набор управляющих флагов, значения которых влияют на его поведение. То есть, операциями часто выполняемыми драйвером является установка и сброс одного или нескольких флагов в таком регистре. При этом значения остальных флагов сохраняются неизменными. Например, контроллер прерываний (PIC) позволяет маскировать линию IRQ, устанавливая бит в порту ввода-вывода. В случае операций ВВчПорты опосредованного ядром, маскирование линии IRQ требует выполнения двух системных вызовов, в то время как в случае набора расширенных операций PMIO потребуется выполнение только одного системного вызова.

С учетом данного паттерна использования, набор основных операций ВВчПорты, поддерживаемых микроядрами, может быть расширен шестью дополнительными операциями:

1. Побитовое И с заданной маской для значения в однобайтовом порту ввода-вывода.
2. Побитовое И с заданной маской для значения в двухбайтовом порту ввода-вывода.
3. Побитовое И с заданной маской для значения в четырехбайтовом порту ввода-вывода.
4. Побитовое ИЛИ с заданной маской для значения в однобайтовом порту ввода-вывода.
5. Побитовое ИЛИ с заданной маской для значения в двухбайтовом порту ввода-вывода.
6. Побитовое ИЛИ с заданной маской для значения в четырехбайтовом порту ввода-вывода.

Последние три операции реализуют установку заданных флагов в регистре устройства отображаемом на порт ввода-вывода. Первые же три, аналогичным образом можно описать как сброс указанных флагов.

Добавленная сложность кода, связанная с поддержкой шести дополнительных операций ВВчПорты является незначительной. В то же время, их наличие сокращает накладные расходы связанные с осуществлением системных вызовов почти вдвое для часто встречающегося паттерна чтение-изменение-запись.

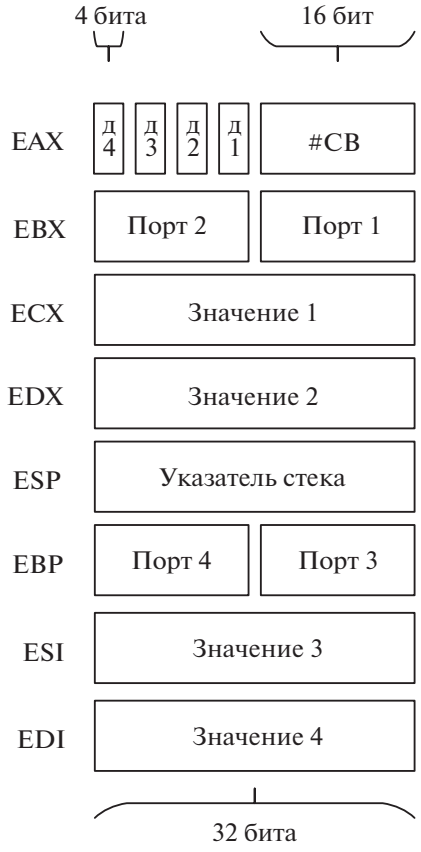

Рис. 1. Схема упаковки аргументов в соглашении о системном вызове ВВчПорты на архитектуре IA-32.

3.2. Упаковка запросов ВВчПорты

Другим часто встречающимся примером использования операций ВВчПорты является последовательный доступ к нескольким портам ввода-вывода без зависимостей по управлению и по данным между ними. Наиболее распространенным случаем этого паттерна является осуществление коммуникации с устройством, которое реализует свое внутреннее адресное пространство ввода-вывода, интерфейс к которому предоставляется через два внешних порта ввода-вывода. Такое устройство предоставляет один из внешних портов для адресации внутреннего адресного пространства, и второй порт – для передачи данных. Коммуникация с такими устройствами состоит из пар операций ВВчПорты: [запись в адресный порт + чтение из порта данных] и [запись в адресный порт + запись в порт данных]. Наиболее заметным примером устройства с внутренним пространством ввода-вывода является контроллер шины PCI, который реализует внутреннее конфигурационное адресное пространство PCI. Другим примером могут служить часы реального времени, которые реализуют внутреннее адресное пространство CMOS-памяти.

В общем случае количество данных, передаваемых в качестве параметров в запрос ВВчПорты, состоит из 16-битного адреса порта и 8-, 16-, 32-битного значения, которые в сумме дают 48 бит данных. Принимая во внимание наличие двенадцати основных операций мультиплексируемых в единый интерфейс системного вызова ВВчПорты необходимо добавить дополнительные 4 бита для кодирования операции. Следовательно, в результате получается 52-битный пакет данных, который необходим для осуществления одного запроса ВВчПорты. Платформа IA-32 предоставляет восемь 32-разрядных регистров общего назначения: EAX, EBX, ECX, EDX, ESP, EBP, ESI и EDI. Регистр ESP зарезервирован и используется процессором для управления стеком. Аналогичным образом, по меньшей мере 8 бит регистра EAX операционная система вынуждена зарезервировать для идентификатора системного вызова. (Предполагается, что восьми бит достаточно для минималистичных микроядер второго поколения, которые поддерживают всего несколько системных вызовов. Например, в нашем случае исследовательское микроядро в настоящее время поддерживает всего 16 системных вызовов, в то время как монолитные ядра обычно перегружены сотнями системных вызовов). В результате получается что для передачи параметров в и из системного вызова может быть использовано максимум $8 \cdot 32 - 32 - 8 = 216$ бит регистрового пространства. $52 \cdot 4 = 208$ бит достаточно для пакетной обработки четырех запросов ВВчПорты в одном системном вызове, для передачи параметров в который используются только регистры ЦПУ без использования стека и памяти. Пример такого пакетного запроса можно найти на Рисунке 1. Проиллюстрированная схема упаковки запросов может рассматриваться в качестве протокола.

Используя обе оптимизации можно упаковать до восьми базовых операций ВВчПорты в один системный вызов.

Как и в случае с предыдущей оптимизацией, добавочная сложность кода, связанная с введением пакетной обработки, является незначительной. Единственным недостатком пакетирования является необходимость специального соглашения о системном вызове для операций ВВчПорты, приводящее к нестандартной упаковке запросов на стороне пользователя. В примере, представленном на Рисунке 1, поля *Значение* в зависимости от значения поля *д* в слоте запроса ВВчПорты могут использоваться либо для передачи аргументов в системный вызов, либо для возврата результатов обратно из системного вызова. Если в *д* задана операция чтения из порта ввода-вывода, то для возврата прочитанного значения используется соответствующий регистр *Значение*. В противном

случае тот же самый регистр *Значение* используется для передачи параметра в системный вызов.

В соответствии с соглашением приложение может запаковывать в системный вызов переменное количество запросов ВВчПорты: от одного и до четырех. При этом драйвер заполняет слоты последовательно и подсистема ввода-вывода ядра будет обрабатывать запросы ввода-вывода в порядке заданном драйвером.

3.3. Архитектура подсистемы портового ввода-вывода

Многослойная архитектура предлагаемой подсистемы управления ВВчПорты представлена на рис. 2.

Вышеописанная модель ВВчПорты включает 12 операций закодированных в 4-разрядные дескрипторы, что позволяет применить на стороне ядра диспетчеризацию запросов на ВВчПорты на основе таблицы функций. Для использования этой возможности необходим унифицированный интерфейс для обработчиков ВВчПорты запросов. Каждая операция ВВчПорты требует двух аргументов: неизменяемый 16-разрядный номер порта и значение с размером не более 32 бит. Значение должно передаваться по ссылке для поддержки всех типов операций: чтение, запись и чтение-изменение-запись.

Номер порта может быть передан в обработчик либо как первый, либо как второй аргумент. Однако, инструкции ВВчПорты IA-32 используют значение в регистре DX для идентификации используемого порта. В то же время, в соглашении о вызове fastcall, EDX используется для передачи второго аргумента во время вызова функции. Таким образом, передача номера порта в качестве второго аргумента в случае использования соглашения о вызове fastcall ставит его на место, где его и ожидает инструкция ВВчПорты. В результате код и процессорные такты, необходимые для манипулирования регистрами, могут быть опущены.

Процедура обработки запросов ВВчПорты включает следующие шаги:

1. Идентификация количества запросов ВВчПорты в пакете.

2. Распаковка кортежей аргументов для каждого запроса в пакете.

3. Проверка прав доступа к портам для каждого из запросов в пакете.

4. Отправка каждого запроса через таблицу функций соответствующему обработчику.

5. Непосредственно выполнение операций ВВчПорты для каждого запроса в пакете.

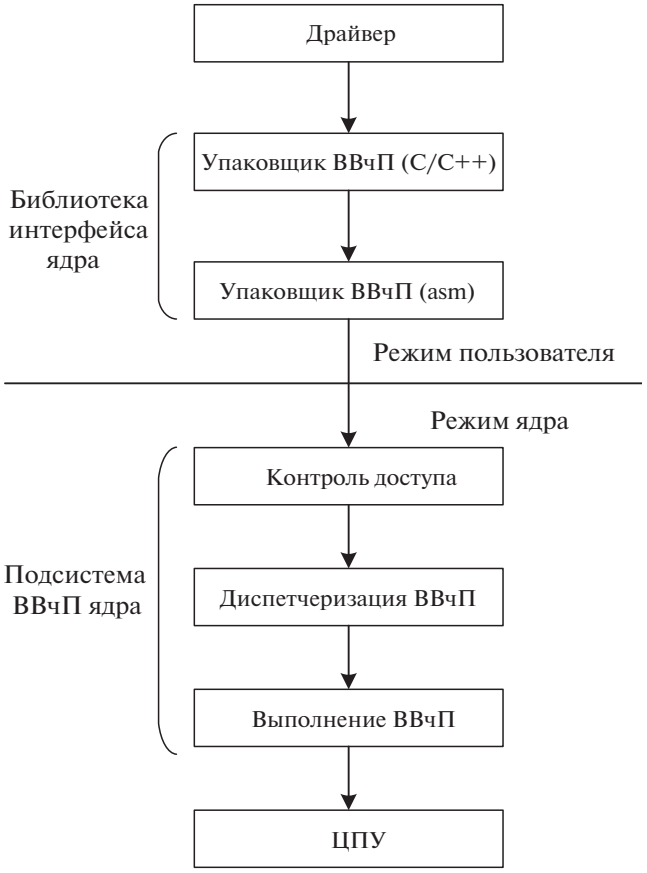

Рис. 2. Многослойная архитектура подсистемы ВВчПорты.

При этом диспетчер системных вызовов ядра, реализуемый на языке ассемблера, выполняет первые два шага.

Соглашение о системном вызове ВВчПорты легко реализуется на стороне ядра, однако, его поддержка на стороне пользовательского режима создает проблему связанную с обеспечением удобства его использования для разрабочиков реализующих драйвера на языках высокого уровня.

Существует 22620 ($12^1 + 12^2 + 12^3 + 12^4$) комбинаций операций ВВчПорты, которые могут быть упакованы в один системный вызов PMIO. Кроме того, эти комбинации содержат переменное количество аргументов. Проблема удобного использования специального интерфейса ВВчПорты может быть решена тремя различными способами.

Первый способ – использование специальной поддержки ВВчПорты на стороне компилятора.

Второй способ – автоматическая предварительная обработка исходного кода драйверов перед компиляцией. Специальный препроцессор может создавать соответствующие оболочки для каждого вызова ВВчПорты, найденного в исход-

```
inline PmioC   (u08  v1,                                  u16 p1                          );
inline PmioWS  (u16  v1, u32 v2,                          u16 p1, u16 p2                  );
inline PmioRWS (u08& v1, u16 v2, u32 v3,                  u16 p1, u16 p2, u16 p3          );
inline PmioRSWR(u08& v1, u16 v2, u32 v3, u08& v4, u16 p1, u16 p2, u16 p3, u16 p4);
```

Шифр операции — Параметры операции

Рис. 3. Примеры функций-оберток верхнего уровня. Шифр операции: C – сброс флагов, R – чтение значения, S – установка флагов, W – запись значения. Размер аргумента *v* определяет размер порта ввода-вывода в соответствующей операции.

ном коде драйвера, и вставлять эту оболочку в исходный код, прежде чем передавать его в стандартный компилятор C/C++.

И, наконец, третий способ – использование специальной библиотеки на основе заголовочных файлов для ВВчПорты. Данный подход используется нами в настоящее время в связи с простотой его реализации.

Библиотека ВВчПорты состоит из двух слоев.

Нижележащий слой состоит из четырех упаковщиков запросов ВВчПорты реализованных на языке ассемблера, каждый из которых упаковывает в системный вызов разное количество запросов. Их функция заключается в размещении запросов ВВчПорты в соответствующих регистрах процессора. Однако, базовые упаковщики обладают двумя существенными недостатками: необходимость помещения аргументов Значение в переменных расположенных в памяти и потеря контроля над соответствием между действительным размером аргумента и его размером заявленным в дескрипторе пакета.

Обе проблемы могут быть исправлены путем введения второго слоя оберток, реализованного в виде встраиваемых функций. Каждая функция при этом реализует упаковку одного конкретного набора запросов ВВчПорты как продемонстрировано на рис. 3. Данное решение подразумевает использование большого заголовочного файла с более чем двадцатью тысячами таких небольших встраиваемых функций. Использование большого заголовочного файла может привести к замедлению процесса компиляции, однако, такое замедление скажется на сборке только драйверов. Также стоит отметить, что использование такого заголовочного файла не скажется на размере исполнимых файлов драйверов. Поскольку функции-обертки объявлены как встраиваемые, компилятор будет инстанцировать только те функции, которые фактически используются драйвером.

4. ВЫВОДЫ

В данной статье была предложена архитектура новой подсистемы управления ВВчПорты, созданная для использования в истинных микроядрах. Предложенная архитектура обеспечивает большую производительность ВВчПорты, достигаемую путем уменьшения среднего количества системных вызовов приходящихся на одну элементарную операцию доступа к портам ввода-вывода. Вместе с этим, предлагаемая архитектура сохраняет полный контроль над правами доступа к портам ввода-вывода со стороны ядра и приверженность принципу минимализма Лидтке [9], однако, требует специальной поддержки в пользовательском пространстве, для обеспечения удобства ее использования разработчиками реализующими драйверы на языках высокого уровня.

Ключевые особенности предлагаемой архитектуры можно сформулировать следующим образом:

• Подход с операциями ВВчПорты опосредованными ядром лежит в основе предлагаемой архитектуры. Однако, в отличие от оригинальной методики, предлагаемый в статье подход приводит к сокращению накладных расходов на операции ВВчПорты за счет мультиплексирования нескольких запросов ВВчПорты в одном системном вызове.

• Стандартный набор операций ВВчПорты, экспортируемых ядром (чтение/запись) расширяется двумя дополнительными операциями – побитовыми И и ИЛИ.

• Предложен механизм для упаковки до четырех запросов ВВчПорты в набор регистров общего назначения IA-32 и, следовательно, в один системный вызов.

• Упаковка переменного количества запросов ВВчПорты предполагает использование специального соглашения о системном вызове для операций ВВчПорты экспортируемых ядром в пользовательское пространство.

• Соглашение о системном вызове требует специальной поддержки при использовании с высокоуровневыми языками программирования. В данной работе предложено три варианта реализации такой поддержки.

СПИСОК ЛИТЕРАТУРЫ

1. Intel Architecture Software Developer's Manual Volume 3: System Programming Guide [Текст]. Number 245472-012. Intel Corporation, 2003. 798 с.
2. *Corbet J.* Linux Device Drivers, 3rd Edition [Текст] / J. Corbet, A. Rubini, G. Kroah-Hartman. Oreilly Media, Inc, 2005. 630 с. ISBN 0-596-00590-3.
3. *Schlussel M.* Ghost Kernel [Электронный ресурс] Режим доступа: https://ghostkernel.org, 23.10.2017.
4. SPI, Debian – Debian GNU/Hurd [Электронный ресурс] Режим доступа: https://www.debian.org/ports/hurd/, 23.10.2017.
5. *HelenOS* [Электронный ресурс] Режим доступа: http://www.helenos.org, 23.10.2017.
6. ETH Zurich, The Barrelfish Operating System [Электронный ресурс] Режим доступа: http://www.barrelfish.org, 23.10.2017.
7. Minix3 [Электронный ресурс] Режим доступа: http://www.minix3.org, 23.10.2017.
8. Data61/CSIRO, seL4 [Электронный ресурс] Режим доступа: https://sel4.systems, 23.10.2017.
9. *Liedtke J.* On Microkernel Construction [Текст]. Proceedings of SOSP-15, Copper Mountain Resort, CO 1995.