массового обслуживания / А.И. Мартышкин, Р.А. Бикташев, Н.Г. Востоков // Фундаментальные исследования. – 2014. – № 11-10. – С. 2155-2159.

5. Журнал Радиоэлектроники №12, 2002. Сервисы GRID, как объекты стандартизации.

6. Журнал PCWeek №5(371), Компьютерная неделя, М.: 2003.

7. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. The Data GRID: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. J. Network and Computer Applications, 2001.

8. Childers, L., Disz, T., Olson, R., Papka, M.E., Stevens, R. and Udeshi, T. Access GRID: Immersive Group-to-Group Collaborative Visualization. In Proc. 4th International Immersive Projection Technology Workshop, 2000.

9. Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C. GRID Information Services for Distributed Resource Sharing, 2001.

10. Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J. and Welch, V. Design and Deployment of a National-Scale Authentication Infrastructure. IEEE Computer, 33(12):60-66. 2000.

УДК 004.451

СОВРЕМЕННЫЕ НАПРАВЛЕНИЯ РАЗВИТИЯ ВСТРАИВАЕМЫХ ОПЕРАЦИОННЫХ СИСТЕМ

А.И. Мартышкин

кандидат технических наук, доцент кафедры вычислительных машин и систем, ФГБОУ ВО «Пензенский государственный технологический университет», г. Пенза, Россия, e-mail: Alexey314@yandex.ru

Аннотация В статье рассматриваются современные концепции и направления развития операционных систем для встроенного применения.

Ключевые слова: встраиваемая система, операционная система, аппаратная часть, программная часть, вычислительная система.

MODERN TRENDS IN THE DEVELOPMENT OF EMBEDDED OPERATING SYSTEM

A.I. Martyshkin

Ph.D., Associate Professor of the Department of Computers and Systems, FGBOU VO "Penza State Technological University", Penza, Russia, e-mail: Alexey314@yandex.ru

Abstract. The article deals with modern concepts and trends in the development of operating systems for embedded applications.

Keywords: embedded system, operating system, hardware, software, computer system.

Введение. Стремительное развитие аппаратной части встроенных систем и расширение их функциональных возможностей повлекло за собой кардинальные изменения в подходе к построению их программного обеспечения. С другой стороны, применение специальных встроенных операционных систем позволило облегчить и ускорить разработку программного обеспечения за счет возможности проектирования их программного обеспечения на языках высокого уровня, обеспечить переносимость программ между различными аппаратными платформами на уровне переносимости исходного кода.

Цель работы. Рассмотреть современные направления развития встраиваемых операционных систем.

Материал и результаты исследований. Современные встроенные операционные системы строятся в соответствии с концепцией микроядр, что позволяет использовать компонентный подход при проектировании программного обеспечения [9]. Архитектура типичной операционной системы для встроенного применения представлена на рисунке 1 [10].

Рисунок 1 – Архитектура типичной ОС для встроенного применения

Такие операционные системы содержат две основные составные части – микроядро и компоненты ядра. Микроядро представляет собой часть операционной системы, осуществляющую непосредственное взаимодействие

с аппаратными средствами встроенной системы и предоставляет программно-абстрактный интерфейс для доступа к аппаратной части системы компонентам ядра. Компоненты ядра определяют общую функциональность встроенной операционной системы, представляют прикладным программам интерфейс операционной системы на уровень прикладных процессов [1-5].

Простейший анализ компонент ядра, изображенных на рисунке 1, показал, что почти все компоненты, исключая планировщик, являются исторически применяемыми и стандартизированными механизмами. Например, стандарт POSIX 1003.1-1988 (Portable Operating System Interface for Computing Environment) предоставляет механизмы межпроцессного взаимодействия, механизмы распределения памяти и др., а конкретный состав драйверов устройств регламентируется спецификацией аппаратного обеспечения [11].

Хотя механизм планирования процессов в вычислительных системах хорошо изучен, однако существует множество различных методик планирования, каждая из которых имеет свои плюсы и минусы, определяемые организацией аппаратного обеспечения вычислительной системы, конкретным составом математического обеспечения и характеристиками потоков обрабатываемых в системе процессов. Обычно в операционной системе реализуется одна из возможных схем планирования исполнения процессов, причем, жестко заданная. В результате такого подхода эффективность работы операционной системы во многом определяется параметрами исполняющихся в системе процессов и составом ее аппаратной части. Это приводит к тому, что при изменении вида нагрузки или состава аппаратной части эффективность работы операционной системы изменяется спорадически [9].

Перспективной задачей сегодня является создание адаптивных планировщиков, способных динамически изменять стратегию планирования в зависимости от состава исполняемых процессов и состава аппаратной части системы. Причем, такой планировщик должен выполнять следующие основные функции (рисунок 2):

– динамически отслеживать изменение аппаратных ресурсов вычислительной системы;

– производить классификацию готовых процессов на основе анализа их требованиях к аппаратной части системы и к компонентам программного обеспечения системы;

– динамически формировать правила распределения ресурсов системы между готовыми процессами в зависимости от состояния системных ресурсов.

Для эффективной реализации перечисленных функций адаптивный планировщик должен получать данные о составе аппаратуры и исполняющихся в данный момент в системе процессах. Данные об аппаратной части, предоставляемые планировщику, могут содержать информацию о количестве и типах аппаратных устройств доступных процессам, их пропускной способности и иных характеристиках, влияющих на выбор стратегии планирования. С другой стороны, планировщик должен иметь возможность выполнить оценку также и параметров исполняющихся, готовых к исполнению, временно заблокированных процессов и прогнозируемых к исполнению процессов. В этом случае представляется возможным сформулировать правила усовершенствовании стратегии планирования или ее кардинальном изменении системным планировщиком.

Создание подобного адаптивного планировщика и реализация обсужденной схемы формирования стратегии планирования требует решения целого комплекса сложных вопросов, к которым в первую очередь следует отнести:

– каким образом производить анализ состава аппаратуры?

– каким образом определять потребности процессов в аппаратных средствах?

– какую методику следует выбрать для анализа входных данных планировщика и адаптивного формирования стратегии планирования?

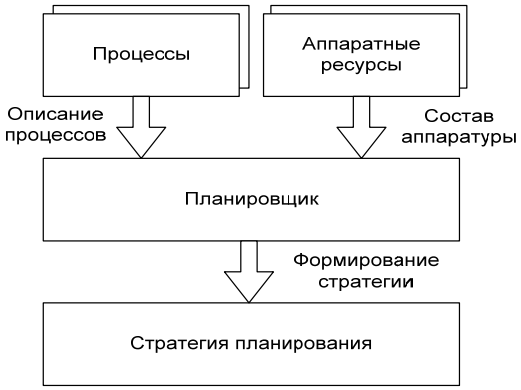

Рисунок 2 – Формирование стратегии планирования адаптивным планировщиком

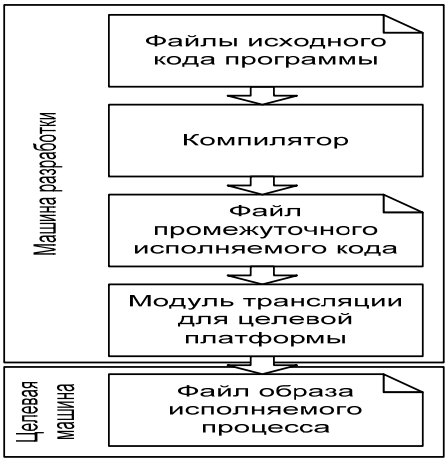

Рисунок 3 – Цикл формирования образа исполняемого процесса во время компиляции

Решение вопроса об анализе аппаратуры можно решить введением в состав операционной системы специального программного компонента для динамического анализа состава аппаратных средств. Этот компонент вводится на уровень компонент операционной системы и должен обеспечивать зондирование аппаратных ресурсов вычислительной системы в определенные временные кванты или по инициативе системных ресурсов. Причем, при изменении конфигурации компонент системы должен обеспечиваться динамический анализ системных ресурсов и ввод данных о новой конфигурации в адаптивный планировщик.

Следует отметить, что такой компонент особенно полезен для вычислительных систем, в которых возможно динамическое изменение аппаратной части в режиме горячей замены. Если аппаратная часть вычислительной системы статична, то в качестве такого компонента может быть использован механизм однократного определения конфигурации аппаратных средств, срабатывающий при первичном запуске операционной системы.

Более сложная ситуация складывается с компонентой адаптивного планировщика, обрабатывающей информацию о требованиях исполняемых процессов к ресурсам системы. Для принятия решения в этом случае необходимо, чтобы находящиеся в памяти системы образы исполняемых процессов несли дополнительную информацию для оперативного и стратегического планирования. Формирование этой дополнительной информации может осуществляться двумя способами:

– формирования образа исполняемого процесса во время компиляции;

– использования промежуточной среды исполнения на целевой машине.

Оба способа подразумевают создание специального компилятора способного по исходному коду программы генерировать некоторый промежуточный код. Дальнейшая обработка промежуточного кода зависит от выбранного способа реализации процесса, порядка использования активных ресурсов системы. Схема формирование образа исполняемого процесса во время компиляции приведена на рисунке 3.

При использовании такой схемы компилятор формирует промежуточный код, который должен немедленно обрабатываться специальным модулем трансляции для его адаптации под ресурсы системы. Модуль трансляции должен обеспечивать анализ промежуточного кода и формировать файл образа исполняемого процесса, который дополнительно к коду программы должен также содержать и информацию для адаптивного планировщика. Распространение программы производится в виде файла образа, который при запуске на целевой машине загружается непосредственно в память вычислительной системы и далее выполняется планирование конкретной схемы его исполнения. Преимуществом данного подхода является высокая скорость загрузки и работы программы на целевой машине. Недостаток заключается в том, что переносимость программ между различными платформами ограничивается на уровне исходного кода [8].

Второй приемлемый в данной ситуации подход может быть основан на организации промежуточной среды исполнения программ. Этот подход применяется, например, фирмами Sun и Microsoft при проектировании программ на языке Java. В этом случае, ввиду изначальной ориентации программ на переносимость и мультиплатформенность на уровне исполняемого кода, можно как нельзя лучше осуществить анализ запросов к аппаратным ресурсам практически во время выполнения программы. Схема исполнения программы с использованием промежуточной среды исполнения представлена на рисунке 4.

В соответствии с этим подходом исходный код разработанной программы оперативно компилируется компилятором и в результате создается промежуточный исполняемый код программы, пригодный для исполнения на любой вычислительной системе, имеющей соответствующую промежуточную среду исполнения.

При запуске получившегося промежуточного кода на исполнение операционная система автоматически вызывает промежуточную среду исполнения, которая обеспечивает интерпретацию промежуточного кода и формирование образа исполняемого процесса по ходу выполнения промежуточного кода. Применение техники промежуточного кода и среды исполне-

ния позволяет расширить функциональность промежуточной среды исполнения механизмами анализа обращений процесса к аппаратным ресурсам вычислительной системы.

Рисунок 4 - Динамическое формирование образа процесса промежуточной средой исполнения.

Таким образом, модификация широко применяемого подхода использования промежуточной среды исполнения позволит обеспечить адаптивный планировщик практически всей необходимой ему информацией об исполняющихся в системе процессах. Причем, промежуточная среда исполнения должна быть реализована в виде компонента операционной системы.

Преимуществом этого подхода является переносимость исполняемого кода, а к недостатку - некоторая потеря во времени при исполнении процессов ввиду необходимости выполнения процедуры интерпретации промежуточного кода [7].

С целью компенсации временных затрат, связанных с интерпретацией промежуточного кода, следует рассмотреть новые методы теории компиляции, направленные на увеличение производительности анализаторов кода, в частности, аппарат рекуррентных нейронных сетей, используемый при анализе «программных» грамматик.

С учетом свойства масштабируемости, которое присуще микроядерным встроенным операционным системам, возможно спроектировать операционную систему, реализующую оба вышеописанных подхода. В этом случае на уровне компонент операционной системы размещается дополнительный компонент промежуточной среды исполнения. При запуске программы на выполнение, представленной в виде промежуточного кода, системный загрузчик задействует дополнительный компонент промежуточной среды исполнения. При запуске программы, представленной в виде образа исполняемого процесса, на конкретной целевой машины загрузчик в

этом случае выполнит загрузку процесса в оперативную память и оповестит планировщик о новом процессе и его характеристиках.

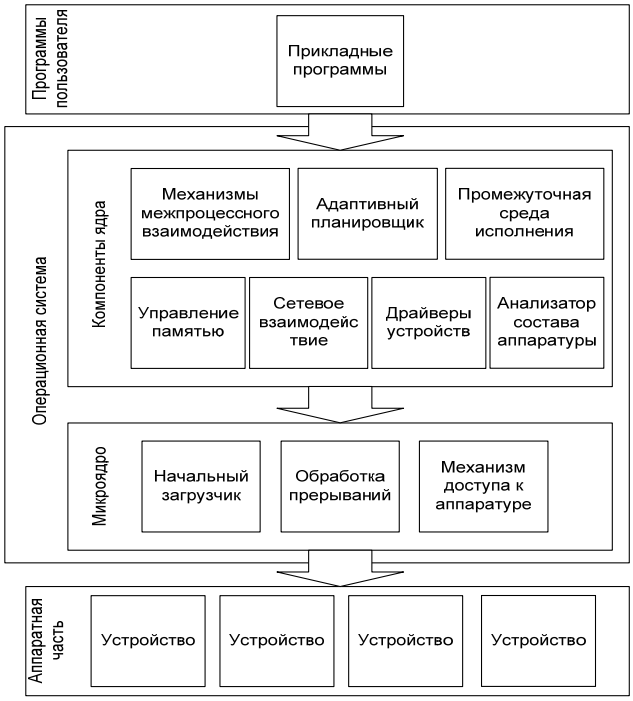

Рисунок 5 – Предлагаемая архитектура встраиваемой ОС.

Методика анализа входных данных адаптивного планировщика и адаптивного формирователя стратегии планирования исполнения процессов может быть решена методами теории искусственных нейронный сетей. В соответствии с требованиями, предъявляемыми к механизму планировщика, для формирования адаптивной стратегии планирования целесообразно использовать искусственную нейронную сеть, обладающую свойствами кластеризации и самообучения.

Наиболее подходящей моделью такой сети является самоорганизующаяся карта признаков или SOFM (Self-Organizing Feature Map) предложенная Кохоненом в начале 80-х годов [6]. Сеть SOFM обладает свойствами сравнения и представления характерных признаков прототипов, что должно позволить планировщику произвести классификацию исполняемых процессов и принять решение о степени важности их исполнения.

Интегрируя все вышеперечисленные решения проблем, возникших при проектировании встроенной операционной системы с адаптивным планировщиком, можно предложить новую модификацию архитектуру системы, представленную на рисунке 5. На наш взгляд дополнительно введенные компоненты гармонично вписываются в архитектуру операционной системы и не противоречат принципам модульности и масштабируемости.

Вывод. Многие специалисты считают, что Grid произведет такую же революцию в области вычислительной обработки, какую сеть Интернет произвела в сфере коммуникации. Кроме того, в ближайшем будущем потребуется большое количество программистов, которые будут адаптировать различные приложения в среду Grid, следовательно, опыт проводимых исследований в данной области пригодится при подготовке новых специалистов.

Использование данной технологии позволит получить за очень короткий промежуток времени больше аналитических данных по построенным моделям, даст возможность использовать вычислительные ресурсы для оптимизации этих моделей.

ЛИТЕРАТУРА

1. Бикташев Р.А., Мартышкин А.И. Комплекс программ для измерения производительности функций операционных систем // XXI век: итоги прошлого и проблемы настоящего плюс. 2013. № 10 (14). С. 190-197.
2. Карасева Е.А., Мартышкин А.И. Обзор средств управления процессами и ресурсами многопроцессорных операционных систем // Международный студенческий научный вестник. 2016. № 3-1. С. 80-81.
3. Мартышкин А.И. К вопросу оценки времени обслуживания заявок при выполнении операций обмена в многопроцессорных системах на кристалле с разделяемой памятью // Приоритеты мировой науки: эксперимент и научная дискуссия: Материалы X международной научной конференции. 2016. С. 81-87.
4. Мартышкин А.И. Основные функции ядра современных операционных систем реального времени // Синтез науки и общества в решении глобальных проблем современности: сборник статей Международной научно-практической конференции: в 2-х частях. Ответственный редактор: Сукиасян Асатур Альбертович. 2016. С. 82-84.
5. Мартышкин А.И., Короткова Н.Н. Устройство аппаратной поддержки диспетчеризации с пространственным разделением задач для многопроцессорной операционной системы //Инновационные механизмы решения проблем научного развития: сборник статей Международной научно-практической конференции. 2016. С. 53-56.
6. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стен. – СПб.: Питер, 2003. – 877 с.: ил. – (Серия «Классика computer science»).
7. Роберт Каллан. Основные концепции нейронных сетей: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 287 с.
8. Робин Хантер. Основные концепции компиляторов: Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 256 с.
9. Сигаев А. Операционные системы для встраиваемых применений. Компоненты и технологии, 2002.
10. Столлингс В. Операционные системы, 4-е издание: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 848 с.: ил.
11. ELC platform specification. Version 1.0: The Embedded Linux Consortium, 2002.