УДК 681.3.068

РЕАЛИЗАЦИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ ДЛЯ МИКРОКОНТРОЛЛЕРОВ AVR ATmega32

М.Г. Логутенко, А.Н. Осокин, С.А. Щербаков*

Томский политехнический университет
*ООО НИИ «ЭлеСи», г. Томск
E-mail: logutenko_maria@mail.ru

Реализована операционная система для микроконтроллеров AVR ATmega на языке Си. В качестве примера использования ОС при разработке ПО и декомпозиции конкретной задачи на отдельные процессы реализован драйвер (программный модуль) для подчиненного устройства, взаимодействующего с главным по протоколу Modbus. Произведено тестирование разработанного программного обеспечения. Проведен сравнительный анализ разработанной операционной системы и свободно распространяемой операционной системой реального времени FreeRTOS.

Ключевые слова:
Микроконтроллер, процесс, диспетчер, критическая секция, операционная система реального времени (ОСРВ).

Key words:
Microcontroller, process, scheduler, critical section, real time operating system (RTOS).

Логутенко Мария Геннадьевна, магистрант кафедры вычислительной техники Института кибернетики ТПУ.
E-mail: logutenko_maria@mail. ru
Область научных интересов: применение микропроцессоров, цифровых сигнальных процессоров.

Осокин Александр Николаевич, канд. техн. наук, доцент кафедры вычислительной техники Института киберне-тики ТПУ.
E-mail: osokin@vt.tpu.ru
Область научных интересов: помехоустойчивое кодирова-ние, сжатие информации, применение микропроцессоров, цифровых сигнальных процессоров.

Щербаков Сергей Андреевич, руководитель сектора ООО НИИ «ЭлеСи», г. Томск.
E-mail: sergey.sherbakov@elesy.ru
Область научных интересов: применение встроенного программирования в области автоматизированного управления.

Микроконтроллеры AVR фирмы «Atmel» представляют собой мощный инструмент, прекрасную основу для создания современных высокопроизводительных и экономичных встраиваемых контроллеров многоцелевого назначения.

Популярность микроконтроллеров AVR постоянно увеличивается. Не последнюю роль в этом играет соотношение показателей «цена/быстродействие/энергопотребление», являющееся одним из лучших на рынке 8-ми разрядных микроконтроллеров. Кроме того, постоянно растет число выпускаемых сторонними производителями разнообразных программных и аппаратных средств поддержки разработок устройств на их основе. Все это позволяет говорить о микроконтроллерах AVR как о новом индустриальном стандарте среди 8-ми разрядных микроконтроллеров общего применения.

В рамках единой базовой архитектуры микроконтроллеры AVR подразделяются на три семейства: Classic AVR, Mega AVR, Tiny AVR.

Микроконтроллеры семейства Mega имеют наиболее развитую периферию и наибольшие среди всех микроконтроллеров AVR объемы памяти программ и данных. Они предназначены для использования в мобильных телефонах, контроллерах различного периферийного оборудования (принтеры, сканеры, современные дисковые накопители, приводы CD-ROM/DVD-ROM и т. п.), сложной офисной технике и т. д. [1–3].

В компании «ЭлеСи» в 2010 г. были разработаны модули ввода-вывода серии ТМ с использованием микроконтроллеров AVR ATmega. Модули серии ТМ представляют собой универсальные модули ввода-

вывода. Основная область их применения – системы автоматического и автоматизированного управления технологическими процессами [4].

При решении любой задачи с использованием микроконтроллера (8-ми , 16-ти или 32-х разрядных) пользователь часто сталкивается с набором следующих трудностей:

- невысокая производительность процессора и ограниченная емкость ОЗУ;
- большое количество обработчиков прерывания (ISR – Interrupt Service Routine) как источников событий, которые необходимо успеть обслужить;
- отсутствие компактной операционной системы (ОС), которая помогает в решении поставленных задачи и содержит малые накладные расходы на выполнения функций ОС;
- отсутствие соглашений по эффективной обработке критических секций и критических участков, обеспечивающих обработку их без закрытия прерываний;
- отсутствие методики соглашений по написанию обработчиков прерывания, удовлетворяющих соглашениям реального времени.

Решить данные проблемы позволяет системное программное обеспечение – операционная система.

В контексте текущего рассмотрения, операционная система – совокупность программного обеспечения (ПО), дающего возможность разбить поток выполнения программы на несколько независимых, асинхронных по отношению друг к другу, процессов и организовать взаимодействие между ними.

Исходя из того, что основная функция ОС – поддержка параллельного асинхронного исполнения разных процессов и взаимодействия между ними, встает вопрос о планировании процессов, т. е. когда какой процесс должен получить управление, когда отдать управление другому процессу и т. п. Эта задача возлагается (хоть и не полностью) на часть ядра ОС, называемой планировщиком или диспетчером. По способу организации работы планировщики бывают:

- с приоритетным вытеснением (preemptive);
- с вытеснением без приоритетов (round-robin или «карусельного» типа);
- без вытеснения (cooperative).

На практике встречаются различные комбинации из упомянутых вариантов, что вносит значительное разнообразие в алгоритмы планирования процессов.

Очевидно, что способность ОС реагировать на события определяется, в первую очередь, типом планировщика. Из упомянутых выше наиболее «быстрыми» в смысле реакции на события являются ОС с приоритетными вытесняющими планировщиками – у них время реакции (при прочих равных условиях) минимально. Остальные типы планировщиков уступают им по скорости реакции.

Однако вытесняющие ОС имеют и недостаток – они значительно более требовательны к ОЗУ, чем невытесняющие. Это принципиальный аспект: в силу того, что любой процесс может быть прерван в любой момент времени, его (процесса) контекст (содержимое регистров процессора, состояние стека) должен быть сохранен соответствующим образом, чтобы при следующем получении управления этим процессом, он смог продолжить свою работу [5].

Данная реализация ядра ОС сочетает возможности и кооперативных ОС, и ОС с вытесняющим планированием. В 2007–2008 гг. в компании «ЭлеСи» данная концепция была реализована на ассемблере для микроконтроллеров Renesas SH2, SH2A.

Следующим этапом развития ядра ОС стала его реализация на Си для микроконтроллера AVR ATmega32.

Ядро ОС построено на ряде соглашений. Ядро не контролирует процессорное время исполнения конкретного прерывания, требует только соблюдения соглашения их написания. Та часть прерывания, которая не уместилась во временные рамки, должна быть исполнена на fork-процессе – приоритетном системном процессе (ПСП). Время нахождения в стандартной программе обслуживания прерываний ISR регламентируется пользователем и, как правило, должно составлять несколько микросекунд (в зависимости от задачи). Все fork-процессы выполняются целостно в порядке их поступления (гарантируется работой ядра), прерывания должны быть разрешены всем источникам. Fork-процесс будет поставлен в очередь ядра, если на данный момент исполняется

системная часть ядра (другой fork-процесс, директива). Выполнения фонового процессов приостанавливается и будет продолжено по завершению выполнения всей очереди fork-процессов. Для постановки в очередь пользователь должен предоставить блок PDB (Process Description Block).

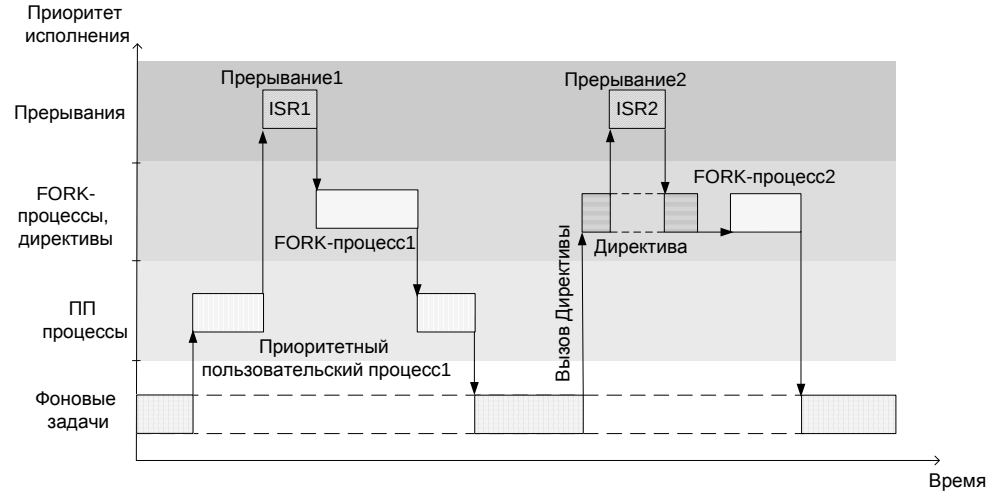

Рис. 1. Временная диаграмма выполнения процессов

Рассмотрим работу обработчика прерывания.

Обработчик прерывания должен быть объявлен с использованием зарезервированного слова __task, что говорит о том, что компилятор не будет сохранять содержимое рабочих регистров при входе в данную функцию. Обработчик прерывания не может иметь параметров и должен быть объявлен как void. Регистр статуса SREG записывается в стек (при этом используется рабочий регистр R16, который перед этим сохраняется в стеке). Ответственность за сохранение контекста при выполнении обработчика прерывания возлагается на разработчика. Если в обработчике прерывания выполняется вызов функции постановки fork-процесса в очередь, то на выходе из обработчика должна осуществляться проверка, не была ли прервана директива. Если директива была прервана, то прерывание должно завершиться макросом EXIT. В противном случае, нужно осуществить переход в диспетчер с помощью функции EndISR(). Если же в обработчике прерывания не вызывается функция постановки fork-процесса в очередь, то он всегда завершается макросом EXIT.

Директивный уровень используется для разграничения доступа к критическим участкам процессов. Для упрощения работы с директивами могут быть использованы макросы BEGIN_DIR и END_DIR.

Кроме того, пользователю предоставляется возможность создания приоритетных пользовательских процессов (ППП). Блок PDB для ППП не отличается от PDB для fork-процесса. Постановка в очередь ППП осуществляется с помощью функции call_pp, входными параметрами которой является указатель на блок PDB ППП и приоритет процесса. Уровень приоритета ППП ниже, чем fork-процессов.

Для создания пользовательской фоновой задачи пользователь должен предоставить блок TDB (Task Description Block). Для постановки пользовательской задачи в очередь на исполнение пользователь должен вызвать функцию task_create. После того, как все необходимые фоновые задачи созданы, для их запуска вызывается функция start_tasks(). В ней сохраняется контекст холостой фоновой задачи, осуществляется инициализация указателей CSTACK, настраивается таймер 0 для отсчета интервалов переключения пользовательских фоновых задач, загружается контекст задачи, стоящей первой в очереди фоновых задач и передается управление на ее исполнение.

На рис. 2 приведено разбиение процессов по уровню приоритета.

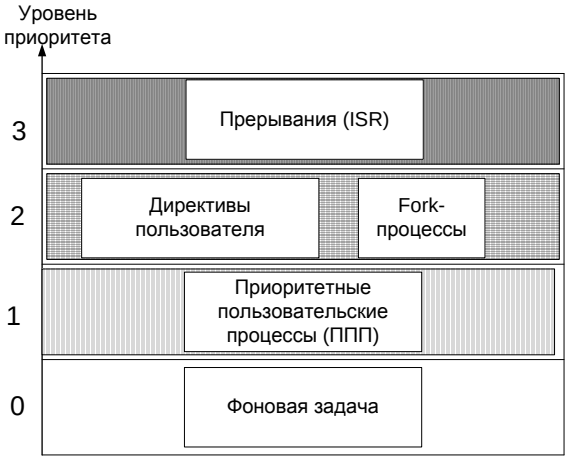

Рис. 2. Уровни приоритета

Функция управления процессами возложена на диспетчер. Диспетчер имеет 3 основных блока: обработка fork-очереди, обработка очереди ППП и изменение фоновой задачи. В блоках обработки fork-очереди и очереди ППП осуществляется проверка на наличие процессов в соответствующей очереди. Если очередь не пуста, то происходит удаление очередного процесса из очереди и его выполнение. При обработке очереди фоновых задач осуществляется проверка, истекло ли время работы текущей программы. Если это произошло, то контекст текущей задачи сохраняется, а сама задача помещается в конец очереди. При этом из очереди выбирается новая задача и загружается его контекст, а задача передается на исполнение.

Обобщая структуру ядра, можно выделить следующие выполняемые функции:

- инициализация ядра void init_kernel() (должна быть вызвана на этапе инициализации пользовательской задачи, до первого выполнения ISR). Функция выполняет инициализацию очереди fork-процессов, очереди ППП, очереди фоновых задач, обнуляет переменные ядра;
- диспетчеризация процессов _task void dispatcher() выполняет проверку очереди fork-процессов. Если очередь не пуста, запускает отложенный fork-процесс, иначе осуществляется проверка очереди ППП. Если очередь не пуста, выполняются ППП один за другим. В противном случае, производится проверка, не истекло ли время выполнения текущей фоновой задачи. Если истекло, то контекст текущей задачи сохраняется в ее TDB, а из очереди фоновых задач выбирается новая задача. Переход в программу диспетчера процессов осуществляется либо по завершению fork-процесса, либо по завершению выполнения директивы или ISR;
- формирование очереди fork-процессов call_fork (PDB *). Вызов данной функции производится в тех случаях, когда исполнение ISR не укладывается в определенные временные рамки или когда необходимо произвести разграничение к критическим ресурсам на уровнях секций процессов;
- формирование очереди ППП call_pp (PDB *p_newPDB, unsigned char prior). Постановка ППП в очередь осуществляется с помощью директивы;
- формирования очереди фоновых задач task_create (TDB *p_newTDB) и запуска их выполнения start_tasks ();
- вспомогательные функции – функции обработки очереди, функция перехода в диспетчер из прерывания EndISR(), макросы BEGIN_DIR, END_DIR.

В качестве примера использования ОС при разработке ПО и декомпозиции конкретной задачи на отдельные процессы был реализован драйвер (программный модуль) для подчиненного устройства, взаимодействующего с главным по протоколу Modbus RTU.

Разработанная ОС успешно прошла все тесты (табл. 1).

Таблица 1. Результаты тестирования

Содержание теста	Ожидаемый результат
Проверка регистра статуса SREG	Выполнение процессов из ППП-очереди и работа диспетчера не должны изменять регистра статуса SREG текущей фоновой задачи.
Проверка указателя стека SP	Выполнение процессов из ППП-очереди и работа диспетчера не должны изменять указатель стека SP
Проверка регистров	Выполнение процессов из ППП-очереди и работа диспетчера не должны изменять регистры текущей фоновой задачи.
Выполнение трех фоновых задач	Фоновые задачи должны осуществляться последовательно друг за другом.
Проверка сохранности регистров при выполнении трех фоновых задач	Фоновые задачи не должны изменять контекст друг другу

Таблица 2. Сравнение функциональных возможностей разработанной ОС и FreeRTOS

Характеристика	Операционная система	
	FreeRTOS	Разработанная ОС
Вытесняющая и кооперативная многозадачность	+	+
Отсутствие ограничений на количество задач, которые может создать пользователь	+	+
Возможность периодического запуска задач	+	+
Директивный уровень разграничения критических секций	-	+
Двоичные, счетные, рекурсивные семафоры, мьютексы	+	-
Очередь сообщений – для обеспечения межзадачной коммуникации	+	-
Функции управления памятью	+	-
Функции статистики времени выполнения задач	+	-

Проведем сравнение разработанной ОС и свободно распространяемой ОСРВ FreeRTOS по функциональным возможностям и времени выполнения некоторых функций (таблицы 2 и 3, соответственно). FreeRTOS – легко портируемая свободная ОС реального времени. Весь системно-независимый код из состава FreeRTOS написан на языке Си, что позволило разработчикам создать большое количество портов на различные архитектуры и средства разработки [6].
В качестве количественного критерия сравнения FreeRTOS и разработанной ОС выступило среднее время выполнения следующих операций:

- постановка задачи на выполнение, инициированная из обработчика прерывания;
- выполнение критической секции.

Таблица 3. Сравнение разработанной ОС с FreeRTOS по времени выполнения

Время выполнения операций, мкс	Операционная система	
	FreeRTOS	Разработанная ОС
Постановка задачи на выполнение из обработчика прерывания	11	4
Вход в критическую секцию	0,6	0,3
Выход из критической секции	0,6	0,7

Как видно из полученных данных, FreeRTOS предоставляет больше средств для синхронизации процессов и обмена информацией между ними, чем разработанная ОС. С одной

стороны, это позволяет использовать ОС для решения более широкого круга проблем, с другой – увеличивает накладные расходы и время, необходимое разработчику для освоения функциональных возможностей ОС. Таким образом, хотя FreeRTOS и обладает большей функциональностью, но требует больших накладных расходов, что в условиях ограничения ресурсов (времени выполнения, памяти) является крайне нежелательным.

Выводы

Разработана работоспособная операционная система, предоставляющая формализованный набор средств для распределения задач по процессам и по организации потока управления, что качественно меняет процесс разработки ПО в сторону упрощения, формализации логики работы программы как в пределах одного процесса, так и в пределах всей программы в целом. Возникает тенденция к появлению типовых решений, что упрощает повторное использование кода в других проектах, а также упрощает переносимость между платформами хотя бы в рамках одной ОС.
Кроме того показано, что от конкретного решения при построении ОС напрямую зависит производительность ПО. Вследствие этого разработчику рекомендуется перед выбором ОС для проекта изучить документацию и исходный код ОС для того, чтобы выявить ее слабые и сильные места и определить применимость к решению поставленной задачи.

СПИСОК ЛИТЕРАТУРЫ

1. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «Atmel». – М.: Додэка-XXI, 2004. – 560 с.
2. Баранов В. Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Додэка-XXI, 2004. – 288 с.
3. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному. – М.: Солон-Пресс, 2003. – 288 с.
4. Официальный сайт компании ЭлеСи. // Сайт компании ЭлеСи. 2005. URL: http://www.elesy.ru (дата обращения: 18.01.2011).
5. scmRTOS: Операционная система реального времени для однокристальных микроконтроллеров/ Руководство пользователя. – Новосибирск, 2006. – 109 с.
6. Щербаков К.С., Щербаков С.А. Особенности работы операционной системы реального времени FreeRTOS // Itech. Журнал интеллектуальных технологий. – 2011. – № 18. – С. 71–77.

Поступила: 21.09.2011.