INFORMATION TECHNOLOGY

Предотвращение взаимоблокировок в ОС для программируемых систем на кристалле

К. С. Гайдук*, О. Г. Шевченко

Донецкий национальный технический университет, Покровск, Украина
*Corresponding author. E-mail: ks.gayduk@gmail.com

Paper received 28.06.18; Accepted for publication 04.07.18.

https://doi.org/10.31174/SEND-NT2018-172VI20-06

Аннотация. В работе представлен подход к предотвращению взаимоблокировок во встраиваемых операционных системах, основанный на объединении событийно-управляемой и сервис-ориентированной архитектур, асинхронного ввода-вывода, а также других приемов, использованных при разработке авторской операционной системы. С целью сокращения накладных расходов процессорного времени, обусловленных потребностью в динамическом распределении памяти, а также сокращения дисперсии времени выделения блоков памяти, операционная система использует ресурс программируемой логики.

Ключевые слова: *предотвращение взаимоблокировок, автоматное программирование, встраиваемые операционные системы, программируемые системы на кристалле, Predicate OS.*

Введение. Основная часть работ в направлении исследований проблемы взаимоблокировок (ВБ) была завершена еще до 1980 г., однако для ряда систем данная проблема остается актуальной. К числу таковых можно отнести: системы, характеризующиеся большим количеством элементов и связей (например, современная локальная вычислительная сеть автомобиля содержит порядка 80 микроконтроллеров, а общий объем исходного кода встраиваемого программного обеспечения может достигать 100 MLOC); системы с динамически изменяемым составом, топологией и алгоритмом работы; системы с децентрализованным управлением.

Большинство методов уклонения от взаимоблокировок и их предотвращения предполагает существенные накладные расходы процессорного времени и ОЗУ, либо являются непрактичными (алгоритм банкира, порядковая нумерация ресурсов, принудительное отнятие ресурсов и пр.). Применение парадигмы автоматного программирования позволяет достичь предотвращения взаимоблокировок, однако может потребовать динамического распределения памяти, что ограничивает применение указанной парадигмы при разработке встраиваемых операционных систем.

Частичная либо полная аппаратная реализация распределителя памяти в значительной мере нивелирует вышеуказанное ограничение.

Краткий обзор публикаций по теме. Предотвращение взаимоблокировок посредством атак на условия Коффмана, а также ряд алгоритмов уклонения от ВБ, описаны в [1]. В [2, 3] упоминается о языках формального описания программного обеспечения (Funclet+, xGiotto, АФС), а также методе Model Checking, позволяющих достичь предотвращения ВБ путем верификации. Важным инструментом в борьбе с взаимоблокировками являются сети Петри [4, 5], позволяющие выполнять проверку системы на предмет свободы от тупиков формализованными математическими методами, а также добиваться уклонения от ВБ либо их предотвращения за счет организации супервизорного контроля.

В [6] представлен распределенный алгоритм уклонения от ВБ в системах автоматизированных транспортных тележек, в соответствии с которым роботы многократно останавливаются на пути своего движения и обмениваются сообщениями с соседями, координируя тем самым совместную деятельность. В работе [7] описан коэволюционный генетический алгоритм разрешения взаимоблокировок в сенсорных сетях автомобилей, позволяющий выбрать процесс либо группу процессов, удаление которых избавляет от кругового ожидания, и влечет минимальные отклонения в работе системы от штатного режима.

Теоретическая эффективность автоматного программирования в решении проблемы взаимоблокировок обсуждается в работах [3, 8]. Алгоритм динамического распределения памяти, пригодный для применения в операционных системах реального времени, описан в [9].

В работе [10] дано определение операционных систем для программируемых (реконфигурируемых) систем на кристалле, отличительной чертой которых является возможность задействования ресурса программируемой логики в качестве сопроцессоров либо аппаратных задач.

Цель. Целью настоящей статьи является рассмотрение возможности практического применения событийно-управляемой и сервис-ориентированной архитектур как инструмента борьбы с взаимоблокировками при разработке встраиваемых операционных систем.

Материалы и методы. Предотвращения ресурсных взаимоблокировок можно достичь посредством атаки на условие взаимного исключения, в случае которой с каждым ресурсом в системе сопоставляется программный модуль (сервис), буферизирующий запросы к данному ресурсу, и обрабатывающий их [1]. Организация программного обеспечения (ПО) в виде множества сервисов, обменивающихся запросами и ответами, соответствует сервис-ориентированной архитектуре (Service-Oriented Architecture, SOA), хотя буферизация сообщений не является ее неотъемлемым атрибутом.

Распространенным является совместное применение SOA и событийно-управляемой архитектуры (Event-Driven Architecture, EDA), согласно которой ПО организуется в виде диспетчера и множества обработчиков, вызываемых диспетчером при наступлении соответствующих событий. Если логика работы системы достаточно сложна, и реакция на события зависит от предыстории, то программное обеспечение может быть организовано в виде системы конечных автоматов, взаимодей-

ствующих посредством обмена сообщениями, и вызывающих на переходах необходимые обработчики.

Использование SOA и EDA предполагает также применение асинхронного ввода-вывода (Asynchronous I/O, AIO), в случае которого единица ПО, инициировавшая операцию ввода-вывода (ВВ), не блокируется в ожидании ее завершения, но продолжает выполнение работы. О завершении операции ВВ, инициировавшая ее единица ПО впоследствии уведомляется посредством сообщения, что интерпретируется как событие [3, 8]. Применение исключительно асинхронного ввода-вывода, позволяет выполнить атаку на условие удержания и ожидания.

Стоит отметить, что указанные выше архитектурные решения всё же не исключают возможности ресурсных взаимоблокировок, в силу конечности размера очередей сообщений. Для решения данной проблемы, могут быть использованы различные приемы, такие как:

1) динамическое изменение размера очередей;

2) сопоставление с сообщениями крайних сроков их обработки, и последующим периодическим прореживанием очередей от неактуальных сообщений.

Система конечных автоматов, взаимодействующих по акторной модели путем обмена сообщениями, не застрахована от коммуникационных взаимоблокировок, что проиллюстрировано рисунком 1, на котором изображено три автомата A1-A3, а текущие состояния отмечены цветом. Автомат A1 может изменить текущее состояние лишь при условии получения сообщения e1, однако соответствующее событие может быть сгенерировано лишь автоматом A3 на переходе из состояния 1 в состояние 2 и т.д. Очевидно, что представленная система находится в тупиковом состоянии.

В целях предотвращения взаимоблокировок указанного типа, с текущими состояниями автоматов системы могут быть сопоставлены соответствующие таймауты. По истечении таймаута, супервизор может переводить автомат в начальное либо некоторое иное состояние, тем самым делая невозможным круговые ожидания неограниченной продолжительности.

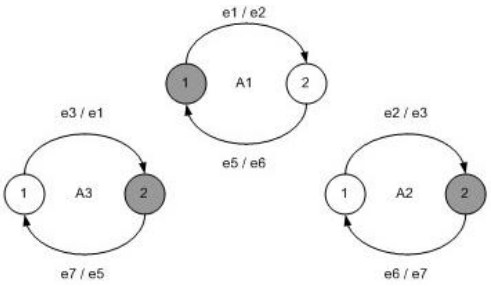

Рис. 1. Состояние коммуникационной ВБ

Взаимодействие автоматов посредством обмена сообщениями, может требовать динамического распределения памяти (ДРП), которое редко используется во встраиваемых системах по причине низкого быстродействия и недетерминированности времени работы большинства алгоритмов ДРП. Однако, в работе [9] представлен алгоритм TLSF (Two-Level Segregated Fit), характеризующийся достаточно высоким быстродействием и константным временем выделения блока памяти.

Алгоритм TLSF использует два уровня битовых карт и неотсортированные списки свободных блоков. Узким местом алгоритма являются операции поиска старшего и младшего значащих бит в слове, однако в ряде машинных архитектур имеются команды, реализующие аппаратный поиск указанных бит за константное время (например, команды BSR и BSF в архитектуре x86). Ряд широко используемых во встраиваемых системах аппаратных архитектур (ARM, AVR, PIC и др.) не имеет в своем составе аналогичных команд, однако это можно компенсировать путем использования программируемой логики в качестве сопроцессора.

Результаты и их обсуждение. Была разработана встраиваемая операционная система (ОС) Predicate OS, характеризующаяся микроядерной архитектурой (рис. 2). Микроядро обеспечивает динамическое распределение памяти, управление задачами (создание, удаление, планирование и т.д.), межзадачное взаимодействие посредством обмена сообщениями, а также счет локального системного времени. Прочие функции операционной системы, такие как поддержка асинхронного ввода-вывода, прореживание очередей от неактуальных сообщений и контроль таймаутов пребывания задач в текущих состояниях, реализованы в виде множества системных сервисов.

На момент написания данной работы, целевой аппаратной платформой для Predicate OS является программируемая система на кристалле (ПСНК) PSoC 5LP фирмы Cypress, однако за счет наличия в составе ОС уровня аппаратных абстракций (УАА), код микроядра и системных сервисов является аппаратно-независимым.

Взаимодействие между отдельными компонентами микроядра, системными сервисами и задачами реализовано за счет интерфейса системных вызовов и обмена сообщениями. Исключение составляют сервисы сборки мусора (неактуальных сообщений) и контроля таймаутов, имеющие прямой доступ к переменной-счетчику хранения локального времени в системе, которая принадлежит подсистеме времени.

Базовой структурой данных разработанной ОС являются динамические двусвязные списки, на основе которых реализованы очереди с приоритетом. Почтовый ящик задачи – это очередь с приоритетом, в которой сообщения отсортированы по убыванию приоритета и возрастанию крайнего срока обработки. Задача считается готовой к выполнению, если ее входная очередь сообщений не пуста.

Каждая задача (включая системные сервисы), в зависимости от логики ее работы, описывается моделью конечного автомата Мили, Мура, либо смешанного автомата. Входным алфавитом H автомата является декартово произведения вида $H = X \times E$, где X – множество условий, проверяемых автоматом, E – множество событий, обрабатываемых автоматом. Выходной алфавит автомата представлен множеством обработчиков, которые он может вызывать на своих переходах. Код текущего состояния автомата хранится в блоке управления задачей.

С точки зрения программной реализации, задача – это совокупность Си-функции, реализующей функции переходов и выходов автомата, а также множества функций-обработчиков, вызываемых автоматом на переходе. Получив управление, задача-автомат извлекает из своего почтового ящика сообщение, выполняет его обработку, и переходит в новое состояние, после чего возвращает управление планировщику. Переход является атомарным, и не может прерываться другими задачами. Планирование выполнения задач осуществляется на основании приоритетов.

Использование функций задержки в системе не предусмотрено, однако задачи могут создавать отложенные задания: об истечении требуемого интервала времени задача будет уведомлена специальным сообщением. По причине событийной ориентированности системы, циклы ожидания при разработке ПО также не применяются.

Сервисы сборки мусора и контроля таймаутов являются аналогами периодических задач: по итогам выполнения одного такта работы, сервис создает отложенное задание, в соответствии с которым планировщик в положенное время помещает в почтовый ящик задачи-сервиса сообщение активации, переводя тем самым задачу в состояние готовности.

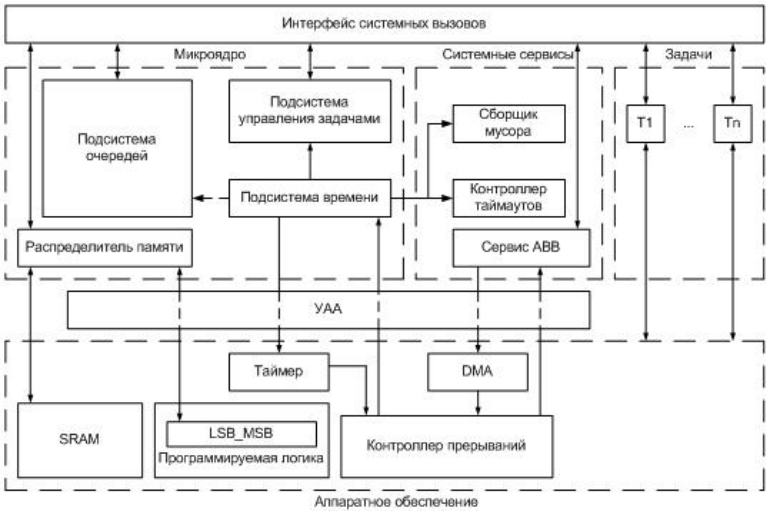

Рис. 2. Архитектура операционной системы Predicate OS

За один такт работы, сервис сборки мусора осуществляет фильтрацию одной из существующих в системе очередей, удаляя из нее неактуальные сообщения (при наличии таковых). Обработка очередей осуществляется в соответствии с алгоритмом round-robin.

Периодическое прореживание очередей от неактуальных сообщений является актуальным не только в контексте предотвращения взаимоблокировок, но и в контексте обеспечения безопасности устройств интернета вещей, препятствуя отказу в обслуживании в случае DoS-атаки.

Сервис контроля таймаутов за один такт своей работы выполняет проверку всех существующих в системе задач на предмет превышения ими допустимого времени пребывания в текущем состоянии. В случае превышения таймаута, задача принудительно переводится в состояние с кодом ноль, что должно учитываться программистом при разработке прикладного ПО.

В текущей версии операционной системы (0.1), асинхронный ввод-вывод (ABB) поддерживается только в отношении операций с памятью SRAM. В целях сокращения времени чтения и записи, сервис ABB использует прямой доступ к памяти.

Табл. 1. Сравнение временных характеристик распределителей памяти

№ п/п	Время выделения блока, тактов ЦП			СКО	V, %	Аппаратная поддержка	Алгоритм	Источник
	ср	мин	макс					
1	1293	183	3451	514	39,8	-	Best-Fit	FreeRTOS, heap2.c
2	323	241	398	24	7,4	-	First-Fit	FreeRTOS, heap4.c
3	273	111	875	128	46,9	-	?	malloc()
4	381	236	529	64	16,8	+	TLSF	Predicate OS
5	625	363	1007	129	20,6	-	TLSF	Predicate OS

Аппаратная поддержка распределителя памяти TLSF реализована за счет имеющейся в составе ПСНК программируемой логики типа CPLD, на базе которой выполнены сопроцессоры определения номеров старшего и младшего значащих бит в слове (на рис. 2 показаны как единое устройство LSB_MSB).

Результаты сравнения быстродействия и дисперсии времени выделения блока для полученной реализации TLSF, а также некоторых распределителей, входящих в состав популярной встраиваемой ОС FreeRTOS (v. 9.0), и стандартным библиотечным распределителем (СБР) malloc(), представлены в табл. 1. Из таблицы видно, что СБР имеет малое среднее время выделения блока, однако достаточно большое значение коэффициента вариации V–порядка 47%. Также видно, что аппаратная поддержка алгоритма TLSF позволяет сократить среднее время выделения блока на 39% и коэффициент вариации на 4%.

Коэффициент вариации времени выделения блока для алгоритма First-Fit почти вдвое меньше, чем для TLSF с аппаратной поддержкой, однако стоит обратить внимание на соответствующие гистограммы распределений (рис. 3, а-б).

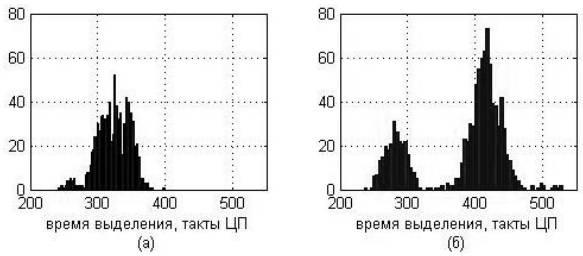

Рис. 3. Гистограммы распределения времени выделения блока для алгоритмов First-Fit (а) и TLSF с аппаратной поддержкой (б)

Из рис. 3 видно, что алгоритм TLSF характеризуется выраженным двумодальным распределением. Если учесть, что при проектировании систем реального времени интерес обычно представляет верхняя граница времени выполнения функции, и устранить из выборки значения ниже 320 (что примерно соответствует центру между двумя модами), то коэффициент вариации составит примерно 5,7%.

Верхние границы системных требований Predicate OS v. 0.1 составляют: 45 Кб FLASH-памяти, 34 Кб SRAM (32 Кб из которых отводится под системную кучу), 42 макроячейки (21,9% от общего количества) и 60 P-термов (15,6% от общего количества).

Выводы. Совместное использование архитектур SOA и EDA, а также асинхронного ввода-вывода, позволяет выполнить атаки на условия взаимного исключения, а также удержания и ожидания, однако это не гарантирует свободы системы от ресурсных ВБ по причине ограниченности размера очередей. Организация программного обеспечения в виде системы конечных автоматов, взаимодействующих посредством обмена сообщениями, а также использование таких приемов как периодическое прореживание очередей от неактуальных сообщений, контроль таймаутов пребывания автоматов в текущих состояниях, и организация очередей на базе динамических списков, позволяют достичь предотвращения ресурсных и коммуникационных ВБ в системе. Потребность в динамическом распределении памяти может ограничивать использование предложенного подхода при разработке встраиваемых ОС, однако использование алгоритма TLSF с аппаратной поддержкой в значительной мере нивелирует данное ограничение.

Дальнейшие исследования целесообразно вести в направлении полной аппаратной реализации распределителя памяти, исследования реактивных свойств Predicate OS, а также возможности организации работы в режиме реального времени.

ЛИТЕРАТУРА

1. Таненбаум Э. Современные операционные системы / Э. Таненбаум. – СПб: Питер, 2010. – 1120 с.
2. N. Akiyama, A. Ikeda, and T. Miyazaki, "Deadlock-free Behavior Definition for Wireless Sensor Nodes Using Formal Verification", Commun. ACM, Vol. 21, No. 8, P. 666–677, 2017.
3. Gaiduk K., Shevchenko O. The Deadlock Problem & Approaches to Its Solution. Computer and informational systems and technologies: a collection of abstracts of the 2nd International Science and Technology Conference (Kharkiv, April 18-19, 2018), P. 59.
4. J.-P. Lopez-Grao, J.-M. Colom, and F. Tricas, "The deadlock problem in the control of Flexible Manufacturing Systems: An overview of the Petri net approach", Proc. 2014 IEEE Emerg. Technol. Fact. Autom., P. 1–12, 2014.
5. Deadlock Prevention Based on Structure Reuse of Petri Net Supervisors for Flexible Manufacturing Systems / Z.Li, G. Liu, H. Hanisch, M. Zhou. // IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans. – 2012. – №42. – P. 178 – 191.
6. Collision and Deadlock Avoidance in Multirobot Systems A Distributed Approach / Y.Zhou, H. Hu, Y. Liu, Z. Ding. // IEEE Transactions on systems, man, and cybernetics: systems. – 2017.
7. J. Xu, Z. Zheng, and M. R. Lyu, "CGA-based deadlock solving strategies towards vehicle sensing systems", Eurasip J. Wirel. Commun. Netw., P. 1–11, 2014.
8. Гайдук К. С., Шевченко О. Г. Аналітичний огляд парадигм подійно-орієнтованого та автоматного програмування. Інформаційна безпека та комп'ютерні технології: збірник тез доповідей III міжнародної наук.-практ. конф. (м. Кропивницький, 19-20 квітня 2018 р.), С. 192-195.
9. TLSF: новая система распределения динамической памяти для систем реального времени / M.Masmano, I. Ripoll, A. Crespo, J. Real.
10. Мельник В. А. Операційні системи реконфігуровних комп’ютерів: будова і організація функціонування / В. А. Мельник, А. Ю. Кіт. // Вісник Нац. ун-ту "Львівська Політехніка". – 2014. – №806. – С. 162–167.

REFERENCES

1. Tanenbaum A. Modern Operating Systems / A. Tanenbaum. – SPb: Piter, 2010. – 1120 p.
2. N. Akiyama, A. Ikeda, and T. Miyazaki, "Deadlock-free Behavior Definition for Wireless Sensor Nodes Using Formal Verification", Commun. ACM, Vol. 21, No. 8, P. 666–677, 2017.
3. Gaiduk K., Shevchenko O. The Deadlock Problem & Approaches to Its Solution. Computer and informational systems and technologies: a collection of abstracts of the 2nd International Science and Technology Conference (Kharkiv, April 18-19, 2018), P. 59.
4. J.-P. Lopez-Grao, J.-M. Colom, and F. Tricas, "The deadlock problem in the control of Flexible Manufacturing Systems: An overview of the Petri net approach", Proc. 2014 IEEE Emerg. Technol. Fact. Autom., P. 1–12, 2014.
5. Deadlock Prevention Based on Structure Reuse of Petri Net Supervisors for Flexible Manufacturing Systems / Z.Li, G. Liu, H. Hanisch, M. Zhou. // IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans. – 2012. – №42. – P. 178 – 191.
6. Collision and Deadlock Avoidance in Multirobot Systems A Distributed Approach / Y.Zhou, H. Hu, Y. Liu, Z. Ding. // IEEE Transactions on systems, man, and cybernetics: systems. – 2017.
7. J. Xu, Z. Zheng, and M. R. Lyu, "CGA-based deadlock solving strategies towards vehicle sensing systems", Eurasip J. Wirel. Commun. Netw., P. 1–11, 2014.
8. Gaiduk K. S., Shevchenko O. G. An analytical overview of event-oriented and automata-based programming paradigms. Information Security and Computer Technologies: a collection of abstracts of the 3nd International Science and Practical Conference (Kropivnitsky, April 19-20, 2018), P. 192-195.
9. TLSF: a New Dynamic Memory Allocator for Real-Time Systems / M.Masmano, I. Ripoll, A. Crespo, J. Real.
10. Melnik V. A. Operational systems of reconfigurable computers: the structure and organization of functioning / V. A. Melnik, A. Yu. Kit. // Bulletin of Lviv Polytechnic National University. – 2014. – №806. – P. 162–167.

Deadlock Prevention in OS for Programmable Systems on a Chip

K. S. Gaiduk, O. G. Shevchenko

Abstract. The paper presents an approach to preventing deadlocks in embedded operating systems based on combining event-driven and service-oriented architectures, asynchronous I/O and other techniques which used to develop the author's operating system. In order to reduce the overhead of CPU time due to the need for dynamic memory allocation, as well as to reduce the dispersion of allocation time for memory blocks, the operating system is using a programmable logic resource.

Keywords: *deadlock prevention, automata-based programming, embedded operating systems, programmable system on a chip, Predicate OS.*