УДК 004.451
Д. Н. Ашневиц – студент кафедры информационных систем
С. В. Осмоловский – научный руководитель

ОСОБЕННОСТИ РЕАЛИЗАЦИИ СТАНДАРТНОЙ БИБЛИОТЕКИ ЯЗЫКА СИ ВО ВСТРАИВАЕМОЙ ОПЕРАЦИОННОЙ СИСТЕМЕ

В настоящее время огромное распространение получает такая область электроники, как встраиваемые системы. Сфера их применения колоссальна и расширяется с каждым днем все больше и больше. Встраиваемые системы можно встретить от банкомата до сложных систем автоматического управления ответственными техническими процессами, таких как системы контроля доступа и бортовые авиационные системы. Сферы применения встраиваемых систем накладывают на них определённые ограничения по размеру, весу, уровню отказоустойчивости, энергопотреблению и тепловыделению. Применение встраиваемых систем в военно-космических областях вносят требования по радиационной и электромагнитной стойкости, работоспособности в вакууме, гарантированному времени наработки, сроку доступности решения на рынке и другим качествам. В связи с этим встраиваемые системы имеют сравнительно малый объём встроенной памяти и относительно низкие вычислительные мощности. Всё это требует от встраиваемых операционных систем (ОС) и операционных систем реального времени (ОСРВ), полностью встроенных в устройства и решающих вполне конкретный круг задач, компактности и высокой скорости реакции на различные события.

В данной статье мы рассмотрим один из способов увеличения производительности и снижения размеров проектов, работающих на платформах встраиваемых ОС, а именно отказ от существующих реализаций стандартной библиотеки языка Си (libc) и реализация собственной библиотеки libc, удовлетворяющей требованиям по размерам и производительности, на примере встраиваемой ОС mcOS, используемой в коммутаторах и концентраторах SpaceWire. В сравнении с аналогами mcOS является компактной встраиваемой операционной системой с высокой производительностью. Основными характеристиками mcOS являются:[1]

- микроядерная архитектура;
- вытесняющая многозадачность;
- количество потоков и уровней приоритетов ограничивается доступной памятью;
- синхронизация одновременно выполняющихся потоков посредством мьютексов;
- протокол наследования приоритетов;
- передача сообщений;
- обработка аппаратных прерываний через сообщения;
- POSIX-совместимость (неполная);
- драйверы таймера, UART, SpaceWire;
- механизм работы с динамической областью памяти;
- понятный и масштабируемый код на языке Си;
- использование свободного компилятора GCC, поддерживающего архитектуру MIPS32

Исторически сложилось так, что язык Си до стандартизации не обеспечивал встроенной функциональности, такой как операции ввода – вывода. Позже для поддержки этой функциональности, сообществом программистов было реализовано то, что мы сейчас называем стандартной библиотекой Си. И это стало частью принятого в 1989 году стандарта ANSI Си. Стандартная библиотека ANSI Си состоит из 24 заголовочных файлов, в которых содержаться функции ввода – вывода, функции работы со строками, вычисления основных математических функций, выделение памяти и многое другое. Каждый заголовочный файл содержит объявления одной или более функций, определения типов данных и макросов. Текущая же реализация функций описана отдельно в библиотечном файле. [2, 3]

Практически все проекты различных размеров и уровней сложности, разрабатываемые на языке Си, используют функционал, предоставляемый libc. Часто реализации libc уже идут в комплекте с используемыми в системах разработки компиляторами, например с набором компиляторов gcc.

С точки зрения встраиваемых систем библиотека языка Си должна быть компактной. Также, из-за того, что во встраиваемых системах применяются не самые высокопроизводительные процессоры,

библиотечные функции должны выполняться быстро. Немаловажным фактом также является реентерабельность функций (свойство функции, позволяющее нескольким вызывающим потокам независимо друг от друга вызывать один и тот же экземпляр данной функции), позволяющая использовать функции в обработчиках прерываний и в такой многозадачной системе, как mcOS.

Обычная GNU-реализация стандартной библиотеки C – glibc – спроектирована с расчетом на совместимость и интернационализацию, и итоговый размер разработчиков не столь волнует. Существует несколько альтернатив, создаваемых с прицелом на экономию занимаемой памяти:

- uClibc: этот проект начинался как реализация библиотеки C для процессоров без модуля управления памятью (memory management unit, MMU-less). С самого начала библиотека была миниатюрной, но в то же время предоставляла функциональность, сходную с glibs. Был отброшен код, отвечающий за многоплатформенность, поддержку многобайтовых кодировок и бинарную совместимость. Более того, утилита конфигурации uClibc дает пользователю свободу выбора, какой код следует включать в библиотеку, а какой можно исключить, тем самым сэкономив на размере.
- Newlib: Newlib выросла при выходе Red Hat на рынок встраиваемых устройств. Newlib содержит очень полную реализацию математической библиотеки, и поэтому рекомендуется для использования в измерительных устройствах и устройствах управления.
- dietlibc: из перечисленных реализаций эта самая миниатюрная - занимает всего 70 Кб, за счет исключения поддержки множества функций, к примеру, поддержки динамически связываемых библиотек. Имеет замечательную поддержку архитектур ARM и MIPS.

У данных реализаций есть как свои преимущества, так и недостатки. В процессе реализации было проведено тестирование библиотеки glibc с использованием компилятора gcc 4.1.3. Полученные результаты не удовлетворяли требованиям компактности mcOS (размеры проектов составляли порядка 80кб). Именно поэтому было решено использовать собственную реализацию стандартной библиотеки Си. Уже на момент реализации большей части самых часто используемых функций libc было произведено сравнительное тестирование новейшей библиотеки NewLib 1.20.0 и библиотеки Си mcOS. Тестовые проекты собирались с использованием последней стабильной версии набора компиляторов gcc 4.6.3, поддерживающих микропроцессорную архитектуру MIPS32. У проектов, собранных с помощью NewLib, размер на 5% – 7% больше, а скорость выполнения функций наравне, а в некоторых случаях и выше, чем у проектов, собранных с помощью стандартной библиотеки Си в mcOS (рис 1.), но используя NewLib мы не можем использовать динамическое выделение памяти и использовать форматированный вывод на консоль через UART (функция debug_printf()). Таким образом, на данный момент реализация стандартной библиотеки Си в mcOS по компактности и быстродействию находится на уровне, а в некоторых случаях и опережает последнюю на данный момент версию стандартной библиотеки Си NewLib.

В mcOS у разработчика есть возможность выбора между компактной и быстрой реализацией стандартной библиотеки языка Си. Для этого при сборке целевого проекта в файле параметров компиляции makedefs проекта с помощью переменной условной компиляции LIBC_OPT задаётся режим оптимизации функций: по размеру – OPTIMAL_SIZE или по быстродействию – OPTIMAL_SPEED. Это позволяет пользователю выбирать тот параметр, который для него является более важным в данном проекте: высокая производительность или же малый размер проекта. Для примера возьмём функцию libc заполнения буфера указанным символом memset(). В реализации данной функции в mcOS при выборе директивы компиляции OPTIMAL_SIZE тело функции занимает всего 4 строки кода и при этом в ней используется простой механизм побайтового заполнения указанной области памяти. Если же выбрана директива компиляции OPTIMAL_SPEED, то тело функции занимает 31 строку кода и она использует специальные алгоритмы, анализирующие размер заполняемой области и, при возможности, заполняющих указанную область одновременно пачкой байт (для микропроцессорной архитектуры MIPS32 алгоритм пытается за итерацию заполнить область 4 машинными словами (16 байт)). Это позволяет получить значительную прибавку в скорости на больших блоках памяти (больше 16 байт) по сравнению с компактной, но медленной реализацией функции. Как видно из графика сравнения производительности некоторых стандартных функций заголовочного файла string.h (рис. 2.) при использовании директивы компиляции OPTIMAL_SPEED мы не получаем выигрыша и даже проигрываем в скорости при малых размерах заполняемого блока памяти, но получаем выигрыш в скорости в 4 и более раз при размерах

блока больше чем 16 байт. Стоит также обратить внимание на преимущество директивы компилятора OPTIMAL_SIZE, а именно позволяют экономить около 60 – 100 байт ПЗУ и 6 – 10 байт ОЗУ для каждой из функций libc. Например, размер тестового проекта для сравнения временных характеристик при выбранном OPTIMAL_SPEED составил 22548 байт, а при выбранном OPTIMAL_SIZE – 21876 байт. Тестирование проходило на платах MC-24EM SpW-GigaSpW(Virtex5) с частотой RISC-ядра 100 МГц. Проект собирался компилятором GCC 4.4.1 с опцией оптимизации по размеру (-Os). Секция данных располагалась во внутренней памяти RISC-ядра (cram), секция кода – в кэшируемой SDRAM. Тестируемая область памяти находилась в SDRAM.

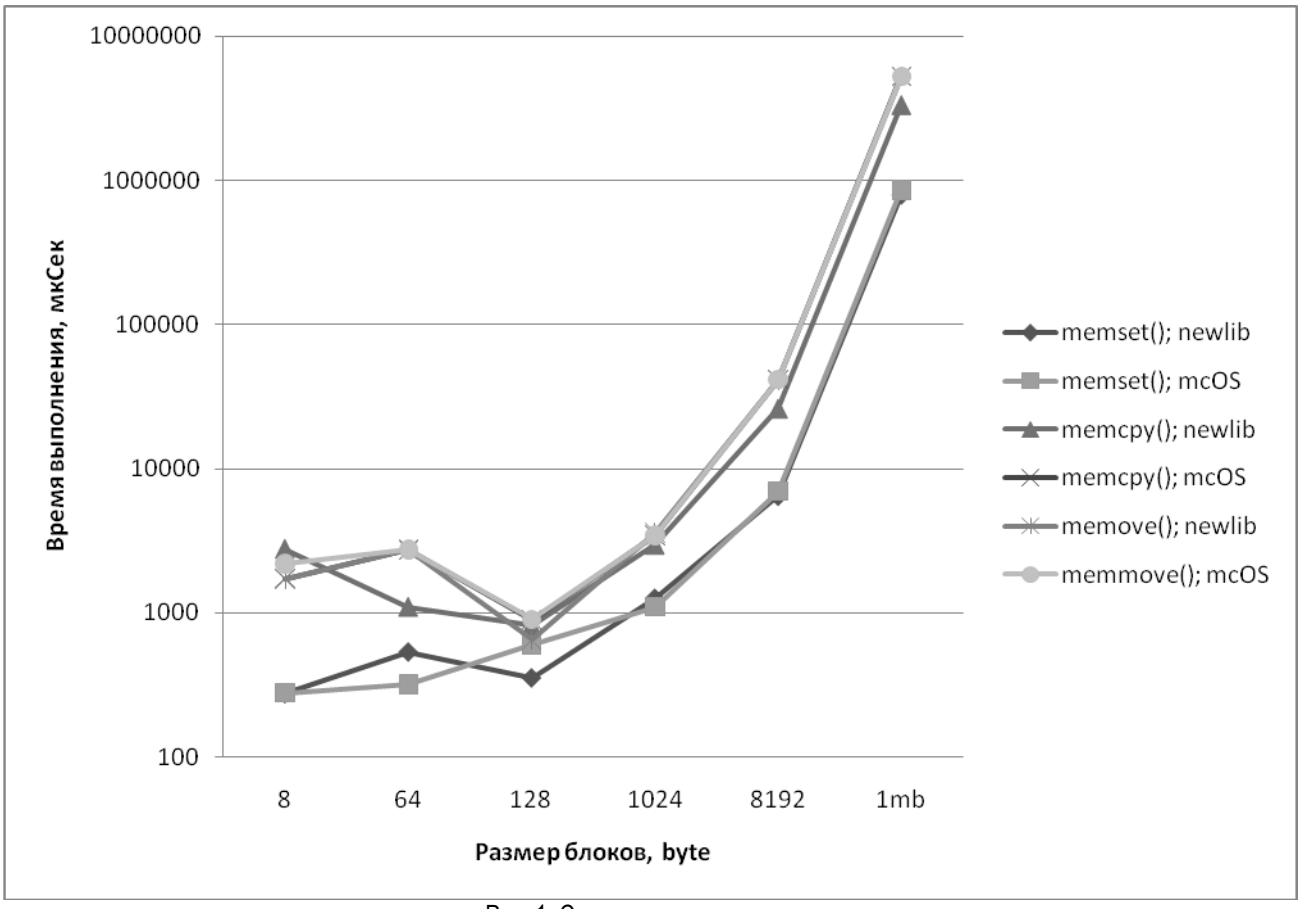

Рис. 1. Сравнение с аналогами

Реализация данных функций на ассемблере с сохранением алгоритмов, используемых в их машинно-независимых версиях, написанных на Си, дал небольшой прирост – около 5% – 7% по скорости (в тестировании и сравнении принимали участие функции memset () и memcpy ()).

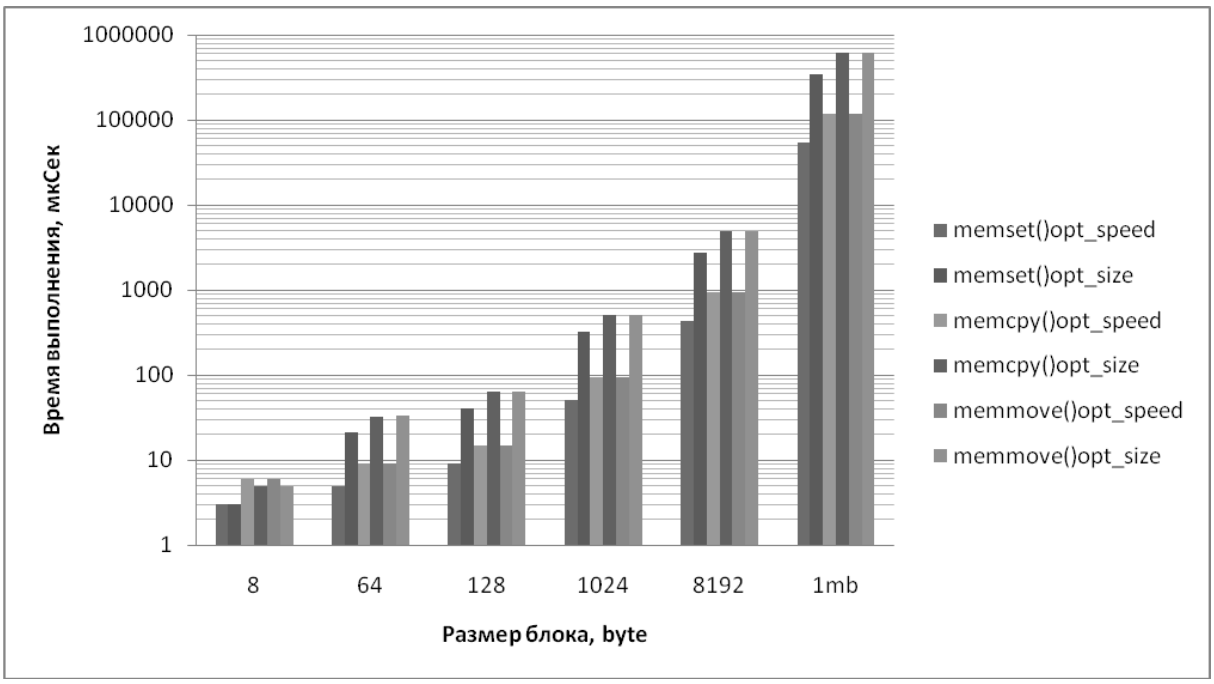

Рис. 2. Сравнение производительности

Так что за сложностью реализации и необходимостью полностью переписывать код таких функций в случае перехода на другую процессорную архитектуру было принято решение отказаться от их дальнейшей реализации.

Стандартная библиотека Си является частью требований, выдвигаемых стандартом POSIX, подстандартом которого является профиль минимальных (встроенных) систем реального времени (Minimal (Embedded) Realtime System Profile) – PSE51, который описывает API-окружение систем реального времени с минимальными функциональными возможностями. mcOS проектировалась и реализовывалась в соответствии именно с требованиями этого профиля. mcOS предоставляет набор API, в котором присутствуют функции для управления потоками, их состоянием и параметрами, функции для синхронизации и взаимодействия потоков между собой, для управления динамической памятью. На данный момент mcOS имеет неполную поддержку стандарта POSIX.1003.13: планируется реализация части стандарта POSIX Threads, определяющего API для создания и управления потоками. Также mcOS частично соответствует стандартам POSIX 1003.1a и POSIX 1003.1b. Из стандарта 1003.1b (Realtime Extensions) mcOS поддерживает планирование выполнения (с учетом приоритетов, циклическое планирование находится на стадии доработки), таймеры, передачу сообщений. mcOS удовлетворяет требованию о реализации не менее 32 уровней приоритетов. Согласно стандарту 1003.1c (Threads) mcOS обладает набором функций для управления потоками, планированием с учетом приоритетов, мьютексами и приоритетным наследованием в мьютексах. [1]

В итоге, благодаря собственной реализации функциональности стандартной библиотеки Си мы получили большую производительность по сравнению с вышеописанными реализациями libc. Также мы имеем возможность при создании проектов на платформе mcOS, в зависимости от требований, предъявляемых к ним, выбирать оптимальное соотношение размер/производительность функциональности libc, а следовательно и проектов. В дальнейшем планируется вести работы по расширению в mcOS поддержки стандартов POSIX.1003.1a, POSIX. 1003.1b и POSIX. 1003.1c. согласно профилю PSE51.

Библиографический список

1. *Суворова Е.А., Осмоловский С.В.* Встраиваемая ОС для микропроцессоров с архитектурой MIPS32 // Шестьдесят третья студенческая научно-техническая конференция ГУАП: Сб. докл.: Ч.1 Технические науки, СПб.: - ГУАП, 2010, с. 323-327.
2. Си (язык программирования) http://ru.wikipedia.org/wiki/Язык_программирования_Си
3. Стандартная библиотека языка Си http://ru.wikipedia.org/wiki/Стандартная_библиотека_языка_Си