УДК 004.451.8

Л.И. Ананьев[1,2], С.В. Семенихин[1,2]
[1]АО «МЦСТ»
[2]ПАО «ИНЭУМ им. И.С. Брука»

L. Ananiev, S. Semenihin

МОДЕЛЬ И ХАРАКТЕРИСТИКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ РЕАЛЬНОГО ВРЕМЕНИ

MODEL AND CHARACTERISTICS OF REAL TIME SOFTWARE

Описана модель работы в режиме реального времени функционального программного обеспечения, аппаратуры и ОС, построенной на базе Linux. Представлены методы и результаты измерения характеристик системы реального времени, а также предложены пути улучшения этих характеристик с использованием и без использования rt_patch для Linux с учетом большого количества процессоров в современных вычислительных системах.

We describe the real time model of the functional software, the hardware, and the operating system that is constructed on the basis of Linux. Measuring methods for the real time system characteristics are described, and the results of the measurements are presented. Different ways to improve upon these characteristics both with and without the use of Linux rt_patch are suggested in view of a large number of processors in modern computing systems.

Ключевые слова: операционная система, реальное время, модель системы реального времени, многопроцессорная вычислительная система, время задержки прерывания, время пробуждения потока, ГЛОНАСС/GPS, POSIX потоки.

Keywords: operating system, real time, real time system model, the multiprocessor computing system, interruption latency time, thread wake up latency time, GLONASS/GPS, POSIX thread,.

Введение

Одним из основных применений операционной системы (ОС) «Эльбрус» является работа в системах реального времени [1]. ОС «Эльбрус» построена на базе Linux,

модифицированного для реального времени с помощью rt_patch, который описан в [2]. С целью измерения временны**ы**х характеристик и поиска путей модернизации ОС «Эльбрус» для удовлетворения требований жесткого реального времени в АО «МЦСТ» используется тестовая программа rt_model (в дальнейшем «модель»), которая моделирует взаимодействие операционной системы с функциональным программным обеспечением реального времени (ФПОРВ) [1]. Тестовая программа rt_model входит в состав системы тестирования ОС «Эльбрус».

Модель работает следующим образом: реальное внешнее прерывание пробуждает первичный поток, который имитирует обработку данных, затем пробуждает второй, фоновый поток, который после имитации обработки данных пробуждает третий фоновый поток, и так далее. Каждый поток, выполнив заданный в параметре теста объём имитации обработки информации, создает заявку, добавляет ее в очередь на обработку следующим потоком и пробуждает его.

1. Особенности модели

Большинство известных программ (например, cyclictest [3]) измеряет характеристики операционной системы реального времени, используя периодическое таймирование для имитации внешних прерываний. Недостатком такого метода, как и при использовании аппаратных таймеров, которые одновременно применяются для нужд самой ОС, является неизбежная зависимость моментов возникновения имитируемого внешнего прерывания от процессов обработки таймерных прерываний в ОС. Например, очередное таймерное прерывание не будет генерироваться, пока ОС занята обработкой предыдущего. Поэтому в модели используется независимое от процессов и занятости ОС аппаратное внешнее прерывание, которое генерирует PCI-модуль приема времени (МПВ), разработанный в АО «МЦСТ». При этом прерывание не вырабатывается в самом процессоре или близком к нему контроллере прерываний APIC, а передается через шины компьютера по тому же пути от МПВ до процессора, как и реальное внешнее прерывание, поступившее на вход

МПВ. В этом заключается принципиальная особенность описываемой модели.

В первичном потоке модели при ожидании следующего прерывания после завершения операций с предыдущим и имитации обработки данных выполняется операция read() для файлового дескриптора одного из входов МПВ. В буфере чтения операции read() можно найти время, прошедшее от момента поступления импульса на вход МПВ до момента запуска драйвера МПВ. Это выполняется следующим образом: при появлении импульса на входе (перед посылкой прерывания процессору) МПВ запускает свой счетчик времени. Драйвер считывает этот счетчик, что позволяет узнать и передать пользователю продолжительность интервала времени, прошедшего от прихода импульса на вход МПВ до начала работы драйвера – задержку прерывания – важнейшую характеристику ОС реального времени. В данной статье применяется термин «импульс», чтобы отличить электрический импульс на входе МПВ от программных сигналов. При поступлении импульса на вход МПВ он посылает процессору прерывание, по которому запускается драйвер МПВ.

Счетчик времени МПВ называется корректирующим, потому что, вычитая его значение из текущего времени начала обработки прерывания драйвером, ОС узнаёт истинное время прихода импульса на вход компьютера. Если, например, внешний сигнал это ежесекундный импульс (PPS – pulse per second) от приемника ГЛОНАСС/GPS, можно точно определить отклонение времени ОС от истинного, причем эта точность не зависит от быстродействия и загруженности процессора и шин. Благодаря тому, что МПВ может сам генерировать прерывания заданной периодичности и фиксировать время, прошедшее после прихода импульса, не требуется дополнительное внешнее оборудование (внешний генератор импульсов, осциллограф) для измерения характеристик ОС реального времени.

В описываемой модели до начала основной работы выполняется блокировка (резидентация) всех данных и кодов программы в оперативной памяти (mlockall). Для оперативного заказа и возврата массивов памяти в многопоточном режиме тест вызывает

функции get_area() и free_area() из библиотеки mcst_rt, используя заранее заказанный резидентный пул памяти. В частности, массивы для заявок на обработку следующим потоком берутся в модели из такого пула. В ОС «Эльбрус» реализован режим RTS_PGFLT_WRN для контроля того, что все массивы в памяти заблокированы. Иначе распечатывается цепочка вызовов процедур, приводящая к прерыванию по отсутствию страницы в памяти.

Синхронизация потоков в модели построена на базе стандартных функций библиотеки libpthread – операции над мьютексами и переменными состояния, но не на семафорах или программных сигналах. Указанные функции рекомендуется использовать для управления потоками в ФПОРВ потому, что они:

- имеют необходимый, без избыточности, интерфейс;
- позволяют реализовать наследование приоритетов;
- экономно расходуют время, т.к. без необходимости не обращаются к ядру;
- имеют особую поддержку в общем программном обеспечении «Эльбрус» для ускорения работы.

В модели (рисунок 1) можно запустить любое количество потоков, пробуждаемых по прерыванию (первичных потоков), и любое количество потоков, пробуждаемых из первичного потока, указав приоритет, привязку к процессору и степень загрузки процессора каждым потоком. В параметрах теста уровень загруженности процессора задается в процентах, а внутри теста это значение пересчитывается в количество повторений небольшого вычислительного цикла, имитирующего обработку данных. Для этого перед запуском собственно модели выполняется калибровочный запуск – определяется количество повторений вычислительного цикла, которые можно выполнить за период между двумя внешними прерываниями заданной периодичности.

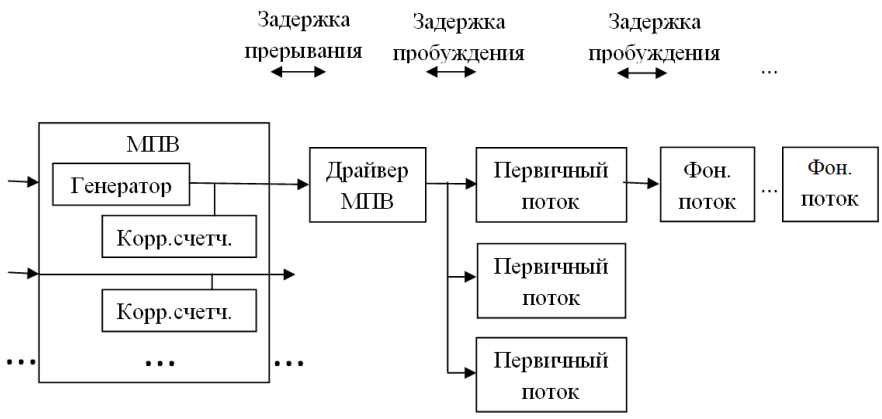

Рис. 1. Аппаратно-программная модель системы реального времени

Варьируя интервал между прерываниями и загрузку процессора, можно экспериментально определить, при какой частоте внешних импульсов и загрузке начинают теряться прерывания, т.е. система не справляется с режимом реального времени. В колонке misd_ints результатов теста указывается количество необслуженных прерываний за все время работы модели. Оно определяется как разность значений регистра количества импульсов на входе МПВ и количества срабатываний драйвера МПВ. Оба эти значения пользователь получает от драйвера в буфере операции read(). Из колонок inq_avrg и inq_max можно узнать среднюю и максимальную длины очереди заявок к данному потоку. Если средняя длина больше 1, то это значение будет бесконечно увеличиваться при увеличении продолжительности работы теста, поэтому оно трактуется как неуспех теста.

В завершение описания модели следует отметить, что она используется не только для предусмотренных разработчиками ОС замеров временны***х*** характеристик, но и для исследования способов уменьшения задержек, для трассировок ядра, а также служит для выработки рекомендаций разработчикам прикладных программ.

Для поиска путей сокращения времени задержки прерывания необходимо знать его составляющие: время закрытых внешних прерываний (prmpt_dis), задержки в аппаратуре и программные задержки от входа в ОС до драйвера МПВ – время входа в прерывание (*_ent). Для измерения времени входа в прерывание при входе в ОС запоминается

значение счетчика команд процессора (тик), чтобы определить количество тиков до входа в драйвер. Время входа в прерывание также передается пользовательской программе в буфере обмена операции read(). Из-за специфики архитектуры «Эльбрус» время входа в прерывание могло оказаться относительно большим по сравнению с другими архитектурами, но опасения не подтвердились, как будет видно из таблиц с результатами.

Второй после времени входа в прерывание важнейший параметр ОС реального времени – это время активизации (пробуждения) потока (wup). Это время от начала работы функции ОС wake_up_process() в драйвере МПВ до переключения на поток, который ждет завершения операции read() для МПВ. Другой вариант времени активизации потока – это время от операции pthread_cond_broadcast() до завершения переключения процессора на поток, который ждет завершения операции pthread_cond_wait ().

2. Применение модели

Результаты работы модели в сокращенном виде выдаются на экран, а полные результаты (около 80 колонок) и параметры окружения работы теста записываются в файл res_rt_mod.csv в формате Comma Separated Values, удобном для дальнейшего анализа в программах Calc или Exel. Каждый новый запуск теста добавляет строку в файле res_rt_mod.csv. Если задана опция -v, то на экран выдаются полные результаты тестирования, такие же, как в файл res_rt_mod.csv. Такое большое количество результатов нужно потому, что модель используется для поиска причин больших задержек и путей их устранения как разработчиками ОС, так и разработчиками ФПОРВ. Как только получен ключевой результат (например, время пробуждения потока), сразу возникают дополнительные вопросы о составляющих элементах задержки и особенностях окружения при работе модели. Для статистического анализа накопленных результатов в программе Exel необходимо каждый тестовый запуск сопровождать всеми характеристиками окружения (частота и количество процессоров, версия Linux, режимы MCST_RT).

Поскольку рассматривается именно модель системы реального времени, а не

измеритель отдельных функций ОС (как например, тестовые пакеты lmbench или hackbench), в результатах присутствует другая группа важных параметров. Она отвечает на вопрос о том, успевает ли система отреагировать на все прерывания: приводится количество пропущенных (не обработанных драйвером) прерываний misd_ints, количество посланных и обработанных потоком заявок (*rqsts). Если средний размер очереди INQ_AVRG больше 1, он будет беспредельно расти с увеличением продолжительности работы теста, что говорит о том, что система не работоспособна. Поскольку детерминированность временны***х*** характеристик – это основное требование к ОС реального времени, третья группа результатов показывает вариации времени имитации обработки (dur_*).

Пример полной выдачи результатов модели (получены на «Эльбрус-4С», 800 МГц) и пояснения к ним представлены в таблице 1.

Таблица 1. Полная выдача результатов модели

`intr THR#=0,`	Следующие результаты – для потока номер 0 (первичного потока, пробуждаемого прерыванием)
`wup_min(mcs)=13.6,`	Минимальное время пробуждения потока 0 (потока пользователя из драйвера), мкс
`wup_avg(mcs)=14.1,`	Среднее время пробуждения потока 0 (потока пользователя из драйвера), мкс
`wup_max(mcs)=17.6,`	Максимальное время пробуждения потока 0 (потока пользователя из драйвера), мкс
`prmpt_dis(mcs)=9,`	Максимальное время запрещения снятия потоков с процессора, на котором работает поток 0, мкс
`min_ent(mcs)=2.6,`	Минимальное время входа в прерывание, мкс
`avg_ent(mcs)=2.7,`	Среднее время входа в прерывание, мкс
`max_ent(mcs)=3.3,`	Максимальное время входа в прерывание, мкс
`min_lat(mcs)=5.1,`	Минимальное время задержки прерывания (от прихода импульса до вызова драйвера), мкс
`avg_lat(mcs)=5.1,`	Среднее время задержки прерывания (от прихода импульса до вызова драйвера), мкс
`max_lat(mcs)=7.0,`	Среднее время задержки прерывания (от прихода импульса до вызова драйвера), мкс
`misd_by_drv=0,`	Суммарное количество прерываний, пропущенных драйвером
`misd_ints=0,`	Суммарное количество прерываний, пропущенных пользователем
`max_misd_ints=0,`	Максимальное количество прерываний, пропущенных пользователем в серии прерываний
`num_wups=100000,`	Суммарное количество пробуждений первичного

	потока за время работы теста
done_rqsts=100000,	Суммарное количество обработанных прерываний
num_rqsts=100000,	Количество заявок в очереди
dur_avg(ms)=0.177,	Среднее время имитации обработки данных после прерывания, мс
dur_min(ms)=0.176,	Минимальное время имитации обработки данных после прерывания, мс
dur_max(ms)=0.181,	Максимальное время имитации обработки данных после прерывания, мс
dur_var(ms)=0.005,	Вариация времени имитации обработки данных после прерывания (max-min), мс
work THR#=1,	Следующие результаты – для потока номер 1 (фонового потока, пробуждаемого первичным потоком)
wup_min(mcs)=14.0,	Минимальное время пробуждения потока 1 от pthread_cond_broadcast() до pthread_cond_wait(), мкс
wup_avg(mcs)=14.6,	Среднее время пробуждения потока 1, мкс
wup_max(mcs)=23.0,	Максимальное время пробуждения потока 1, мкс
prmpt_dis(mcs)=10,	Максимальное время запрещения снятия потоков с процессора, на котором работает поток 0, мкс
INQ_AVRG=1.00,	Среднее количество заявок к данному потоку
num_wups=100000,	Количество пробуждений данного потока
done_rqsts=100000,	Количество заявок, выполненных данным потоком
num_rqsts=100000,	Количество заявок, поставленных в очередь к данному потоку
dur_avg(ms)=0.117,	Среднее время имитации обработки данных в этом потоке, мс
dur_min(ms)=0.116,	Минимальное время имитации обработки данных в этом потоке, мс
dur_max(ms)=0.120,	Максимальное время имитации обработки данных в этом потоке, мс
dur_var(ms)=0.004,	Вариация времени имитации обработки данных (max-min), мс
args_THR#=0,	Ниже перечислены входные параметры потока 0 (первичного, пробуждаемого прерыванием)
cpu_mask=2,	Маска процессоров для потока (процессор номер 1)
polc=FIFO,	Политика планирования потока (приоритет реального времени, с процессора не снимается)
prio=98,	Приоритет потока (возможные значения 0–99)
real_workload %=15,	Заданный процент процессорного времени для имитации обработки данных
work100perc(loops)=9804,	Количество циклов имитации для 100% загрузки процессора (можно задать параметром модели)
args_THR#=1,	Ниже перечислены входные параметры потока 1 (фонового, пробуждаемого первичным)
cpu_mask=4,	Маска процессоров для потока (процессор номер 1)
polc=FIFO,	Политика планирования потока (приоритет реального времени, с процессора не снимается)
prio=50,	Приоритет потока (возможные значения 0–99)
real_workload %=10,	Заданный процент процессорного времени для имитации обработки данных

work100perc(loops)=9709,	Количество циклов имитации для 100% загрузки процессора (можно задать параметром модели)
intrpt_intrv_usec=1123,	Интервал между прерываниями МПВ
release=3.14.53-rt54,	Базовая версия Linux
machine=e2k,	Тип процессора
cpu_freq_mhz=799.999,	Частота процессора
date=Jan 27 20:50:17 2016,	Дата запуска модели
cpu_nr=4,	Количество процессоров
comment=nb,	Комментарий – в параметрах запуска модели для дополнительной характеристики окружения
loop_num=100000,	Заданное количество повторений основного цикла модели
run_time(s)=115.1,	Продолжительность работы модели, с
irq_mask=4,	Маска привязки прерывания МПВ к процессорам
rts_active=0x16,	Маска режимов MCST_RT. Включены POSTP_TICK, HZ_RT, NO_CPU_BLNC (см. команду mod_rts_mask)

Ниже приведены результаты работы модели, сконфигурированной для одной из задач пользователей ОС «Эльбрус». В тестируемой системе на каждом процессоре заданы два потока. Первый поток, который имеет максимально высокий приоритет, пробуждается по внешнему прерыванию каждые 5 мс и занимает 90% процессорного времени. Приоритет второго, фонового потока ниже, чем у первого, но выше, чем у любого из потоков ядра. Он пробуждается в сто раз реже первого, а длительность его работы на процессоре в три раза дольше, чем интервал прерываний. Следует отметить, что разработчики ОС «Эльбрус» по результатам предварительного моделирования считают, что такая высокая загрузка процессора находится в противоречии с требованием минимального времени пробуждения.

Строка запуска на каждом процессоре имеет вид:

rt_model -F98 -l90 -10 -i0 -k100000 -t5000 -F97 -T100 -l300 -b0

Для имитации фоновой загрузки процессоров запускается двухпотоковая задача архивации 7zip_tuned b -mmt2. Команда pidstat показывает, что 7zip_tuned получает 94% процессорного времени до запуска модели и 4,3% при работающей модели. Результаты тестирования при разной занятости процессора потоком реакции на прерывание представлены в таблице 2. Из таблицы следует, что при заданной загрузке работа

выполняется успешно по всем параметрам. В частности, вариация продолжительности работы первого потока не выше 1,3%. При длительности первичной обработки первичным потоком 92% от интервала прерываний и выше система теряет устойчивость. Время пробуждения потока первоначально возросло именно в переходной момент, потому что на процессоре столкнулись работы по пробуждению и снятию потока. В этих условиях процессор лучше не покидать. Такой режим работы был введен в ОС «Эльбрус» позже и описывается ниже.

Таблица 2. Влияние загруженности процессора первичным потоком на работоспособность ФПОРВ

Загруженность процессора, %	wup max, мкс	missed ints	duration min, мс	duration max, мс	duration variation, %
85	33,1	0	4,22	4,27	1,3
87	41,7	0	4,32	4,37	1,1
90	39,8	0	4,46	4,51	1,1
91	32,5	0	4,51	4,56	1,2
92	982,2	13	4,58	4,62	1,0
93	45,1	50	4,62	24,65	433,9
94	46,6	77	4,68	34,71	642,5
95	46,5	100	4,73	44,76	847,3

Таблица 3 показывает результаты тестирования, проведенного при условии, что фоновый поток пробуждается чаще, чем на каждом сотом прерывании (опция -T100) первого потока (и, соответственно, потока пользователя, поскольку его приоритет выше и он не дает возможности работать потокам ядра ОС).

Таблица 3. Влияние загруженности процессора фоновым потоком на работоспособность ФПОРВ

Количество пропускаемых прерываний	wup min, мкс	wup max, мкс	missed ints	duration min, мс	duration max, мс
100	16	32,4	0	4,5	4,5
90	15,9	37,4	0	4,5	4,5
80	15,8	35	2	4,5	14,5
70	15,8	949	2	4,5	4,5
60	15,6	656	1	4,5	4,5
50	15,6	40	19	4,5	14,5

В данном случае резкое увеличение времени пробуждения фонового потока при пропуске 60 и 70 прерываний связано не с задержками в ОС, а с тем, что часто пользовательский поток еще не закончил обработку предыдущей заявки к моменту создания следующей заявки и пробуждения потока для ее обработки. При еще более частом запуске фонового потока время пробуждения нормализовалось, потому что вся работа велась только через очередь заявок. Режим rt_cpu, описанный ниже, мог бы значительно улучшить ситуацию, но в данной конфигурации был доступен только один процессор Е3М (второй – горячий резерв).

В этом эксперименте также видно, что запас загруженности системы фоновым потоком составляет менее 1%. Реально для управления системой остается 2,4% процессорного времени (90% занял первый поток, 3,3% – это 300% загрузки вторым потоком через каждые 90 прерываний и 4,3% – вариация продолжительности работы пользователя).

3. Изменения в ОС «Эльбрус», поддерживающие режим жесткого реального времени

Важное назначение модели состояло в том, чтобы найти и, по возможности, устранить выбросы (пики) задержки ОС «Эльбрус» в режиме реального времени. Хотя rt-patch [2] для Linux значительно сокращал время закрытых внешних прерываний и время запрета снятия потока с процессора (nonpreemption time), время пробуждения и задержки прерывания многократно превышали предъявляемые к ОС требования. В частности, при использовании версии Linux-2.6.14-rt максимальное время пробуждения потока на процессоре «Эльбрус-3М1» с частотой 300 МГц превышало 1 мс, а надо было уложиться в 50 мкс.

При поиске причин выбросов использовалась трассировка ключевых функций операционной системы, в результате анализа которой потребовалось добавить следующие режимы в ОС «Эльбрус» на базе ядра Linux-2.6.14-rt:

RTS_INRPT_SCHED – по планировочному межпроцессорному прерыванию

выполнять переключение на более приоритетный поток, не дожидаясь выхода из ОС;

RTS_NO_DYN_PRIO – не выполнять динамический пересчет приоритетов при каждом снятии с процессора потока с нормальным приоритетом;

RTS_NO_CPU_BLNC – не выполнять динамическое выравнивание загруженности процессоров при каждом вызове планировщика;

RTS_NO_PTIMERS – отключить поддержку медлительных posix-таймеров, предложив вместо них таймер-заменитель на базе таймера локального APICa (tla timer).

В процессе развития ядра Linux многие элементы из rt-patch интегрируются в основную ветвь. В связи с этим при переходе к версии Linux-2.6.33-rt необходимость в режиме RTS_NO_PTIMERS для ОС «Эльбрус» отпала. Это произошло из-за введения таймеров высокого разрешения и появления быстрых интерфейсных функций семейства timerfd. Режимы RTS_INRPT_SCHED и RTS_NO_DYN_PRIO оказались неприменимыми в Linux-2.6.33 вследствие кардинальной перестройки схемы планирования. Но одновременно возникли выбросы из-за того, что при смещении с процессора потока класса реального времени более приоритетным потоком планировщик подбирает для смещаемого потока другой процессор. Соответственно, в режиме RTS_NO_CPU_BLNC был запрещен поиск процессора, чтобы предотвратить задержку активизации потока.

С переходом на ядро Linux-2.6.33-rt появились и другие проблемные зоны быстрой активизации потока. Как показала трассировка функций ОС при работе модели, выбросы случаются из-за использования динамических таймерных прерываний. С одной стороны, непериодические (динамические, dyntick) таймерные прерывания – это сокращение расхода процессорного времени, с другой стороны, они приводят к большим выбросам времени активизации потока, т.к. оказались акцентированными на экономию потребления энергии процессором, а не на потребности систем реального времени. В результате, как показала трассировка, активизация потока на пустующем процессоре (после потока idle) происходила только после исполнения функций, отложенных на время простоя

процессора. Поэтому в ОС «Эльбрус» был введен режим RTS_HZ_RT – включение работы периодических таймеров, как это работало в Linux-2.6.14.

Тем не менее, если операции, вызванные срабатыванием таймера, случались непосредственно перед активизацией потока, они провоцировали выбросы. Усугубляло проблему более активное использование RCU (read copy update – метода отложенной синхронизации данных) в Linux-2.6.33-rt, поэтому в ОС «Эльбрус» были добавлены такие режимы:

RTS_FAST_TIMER – не выполнять таймерные операции, когда на процессоре работает поток системы реального времени;

RTS_NO_RCU_IN_TIMER – не выполнять RCU-работы по таймерному прерыванию.

Но и эти режимы не могли устранить выбросы при активизации потока, т.к. в некоторых случаях таймерные работы начинали выполняться непосредственно перед активизацией потока. Проблема возникала, когда таймерное прерывание приходило непосредственно перед прерыванием от МПВ. Никакие основанные на приоритетах методы в данном случае не могли помочь. Выброс давали даже не сами работы, а разбор списка и активизация потоков этих работ.

Удачным оказалось следующее решение: фиксировать приход таймерных прерываний и откладывать связанную с ними работу до того момента, когда она не помешает основной работе, причем момент продолжения отложенной работы определяет ФПОРВ. Как отмечалось в [1], «в любой системе реального времени всегда есть один или несколько основных циклов ожидания события, обработки полученных данных и выдачи управляющей информации для объекта управления». Для такого цикла нужен быстрый старт, а по завершении обработки внешних данных и выдачи команд функциональное программное обеспечение выходит на ожидание прерывания. Перед этим, используя интерфейсную функцию el_user_tick(), ФПОРВ может разрешить ОС выполнить отложенные таймерные работы. В результате вызова функции el_user_tick() ОС выполняет

разбор и запуск таймерных работ при наличии отметки о необработанном таймерном прерывании. По завершении работы отметка снимается. Пользователь может вызывать функцию el_user_tick() не периодически, но не реже, чем через двойной период таймера ОС. Ситуация, когда в этом режиме приходит таймерное прерывание, а отметка о предыдущем прерывании еще не снята, означает, что пользователь не хочет или не может диктовать ОС, когда можно выполнять отложенные таймерные работы. С этого момента ОС реагирует на таймерные прерывания как обычно, пока пользователь не вызовет el_user_tick().

Для дальнейшего существенного сокращения времени реакции на прерывание в ОС «Эльбрус» реализован режим ожидания прерывания без покидания процессора. Если поток пользователя после получения данного прерывания еще достаточно долго выполняет обработку данных на процессоре и остается мало времени до поступления следующего прерывания, то уже теряет смысл уход данного потока с процессора. Более того, как показало моделирование, при таком встречном выполнении снятия-постановки на процессор одного и того же потока выбросы значительно увеличиваются. Известен адаптивный способ ожидания открытия семафора/мьютекса, при котором поток не покидает процессор при определенных условиях. Аналогичный режим работы есть и в драйвере МПВ – ожидание прерывания без покидания процессора – WAIT_ON_CPU. Если в системе жесткого реального времени действительно требуется быстрая реакция на прерывание, то для этого не жалко потратить часть аппаратных ресурсов в современном многопроцессорном компьютере.

В таблице 4 показаны сравнительные результаты модели в режиме WAIT_ON_CPU.

Таблица 4. Характеристики Linux, rt_patch и дополнений mcst_rt на разных процессорах

rt_patcn	Дополнения mcst_rt	Процессор	Частота процессора, МГц	wup min, мкс	wup max, мкс	wup, max/min	IRQlat min, мкс	IRQlat max, мкс	IRQlat max/min
применен	включены	i686	2666	1,4	18	13	7	28	4

применен	включены	«Эльбрус-E3M1»	300	23	44	2	30	51	2
применен	включены	«Эльбрус-2C+»	496	13	17	1	14	40	3
применен	включены	R1000	820	7	22	3	10	42	4
применен	не включены	«Эльбрус-4C»	800	14	59	4	5	41	8
не применен	включены	«Эльбрус-4C»	800	13	16	1	4	6	2
применен	включены	Эльбрус-4C	800	6	10	2	3	5	2
применен	включены +wait_on_cpu	«Эльбрус-4C»	800	0,7	1,1	1,6	0,7	2,2	3

Несколько более мягким решением является режим rt_cpu. При исследовании работы модели замечено, что выброс задержки возникает вследствие того, что на процессоре побывал поток, не выполняющий строгие требования к потокам реального времени. После него могут быть отложенные работы (упомянутые ранее RCU и др.), межпроцессорные прерывания для чистки кэша и т.п. Функциональное программное обеспечение может обратиться к ОС «Эльбрус» и внести текущий процессор в список процессоров, на которых запрещено работать потокам, которым не обязательно использовать именно этот процессор. Перспективность такого подхода подтвердилась появлением параметра isolcpus= для ядра Linux [4].

Обобщая приведенные выше результаты, следует отметить, что правильная привязка прерываний и потоков к процессорам, использование режима RTS_NO_CPU_BLNC и функции el_user_tick() стали необходимым и достаточным условием того, чтобы время задержки прерывания укладывалось в заданный предел (25 мкс при тактовой частоте процессора 800 МГц).

Большое время пробуждения на процессоре i686 объясняется тем, что долго выполняется переключение процессора с потока idle из-за возможной экономии

электроэнергии процессором, даже после применения стандартного патча rt-patch и при формально отключенном режиме экономии. Большое отставание частоты памяти от процессорной частоты также сказалось на результате для R1000 и i686, выраженном в тактах процессора.

Максимальное время от поступления внешнего импульса на вход устройства МПВ до начала работы прикладной программы пользователя в ответ на внешнее прерывание равно сумме wake_up_max и IRQ_lat_max. Часто требуемый от ОС реального времени быстрый вызов пользовательской программы в ответ на прерывание является следствием консервативного представления о работе ОС и аппаратуры. Если быстрая реакция на прерывание нужна только для того, чтобы точно зафиксировать приход прерывания и/или интервал времени, оставшийся до предельно допустимого времени ответной команды, то пользователю надо иметь в виду, что эту информацию регистрирует аппаратура МПВ независимо от программных задержек скорости и загруженности процессора и задержек в работе памяти. С учетом цикличности работы становится важной не сама величина задержки, а ее детерминированность и доступность для пользователя, что как раз обеспечивают ОС «Эльбрус» и разработанный модуль приема времени.

Сравнивать полученные результаты с другими операционными системами сложно, потому что там используются другие инструменты и измеряются несколько другие параметры. В таблице 5 приведены результаты измерений для QNX и Linux-2.6.33-rt из [5]. Мы считаем, что «задержка перехода из обработчика прерывания в пользовательский поток» в статье «Результаты тестов производительности QNX Neutrino» [5] эквивалентна времени пробуждения потока пользователя из драйвера МПВ (wup). Найти эквивалент для задержки прерывания (IRQlat) ОС «Эльбрус» сложнее. Используемая в [5] «задержка обработки прерывания» скорее эквивалентна «времени входа в прерывание» (*_ent) в ОС «Эльбрус», поскольку она не учитывает аппаратные задержки и не учитывает того, что перед внешним прерыванием могла начаться обработка таймерного прерывания. По

крайней мере, для Linux обработка таймера прибавляется определенно. По-видимому, IRQlat эквивалентно сумме «задержка обработки прерывания» и «время обработки таймера».

Таблица 5. РТ характеристики QNX и Linux-rt

	Частота процессора, МГц	Задержка перехода из обработчика прерывания в пользовательский поток			Задержка обработки прерывания			Время обработки таймера, мкс
		max, мкс	min, мкс	max/min	max, мкс	min, мкс	max/min	
Linux-rt Pentium	200	76,4	21,6		32	8	4,0	55
QNX Pentium	200	15	2,6	5,8	5,8	1,7	3,4	11
QNX Intel Atom	1660	14,7	1,7	8,6	2,8	1,7	1,6	10
QNX ARM A8	1000	6	0,9	6,7	2,6	0,5	5,2	6,5

Заключение

В статье описана программно-аппаратная модель системы реального времени, работа которой опирается на аппаратный имитатор внешних прерываний в МПВ и аппаратную фиксацию интервала времени от момента поступления импульса до начала обработки прерывания.

В результате моделирования было обнаружено, что ОС Linux, включающая rt-patch, не может удовлетворить требованиям работы в жестком реальном времени, особенно если частота процессора недостаточно высока (меньше 800 МГц). С помощью трассировки ключевых функций ядра, выполняемых в модели, был найден и ликвидирован ряд узких мест в ОС. Наряду с этим были введены способы повышения детерминированности работы ОС, которые перечислены ниже в порядке, соответствующем возрастанию требований к жесткости режима реального времени:

- отключение динамической балансировки загруженности процессоров и динамический пересчет приоритетов при каждом переключении потоков;

- выполнение отложенных работ не по таймерным прерываниям, а по назначению пользователя (el_user_tick);
- выделение процессоров, на которых разрешено выполнять только потоки жесткого реального времени (rt_cpu);
- выполнение ожидания внешнего события, не покидая процессор (wait_on_cpu).

Таким образом, значительное увеличение количества процессоров в вычислительных системах открывает возможности, альтернативные применению rt-patch. В результате характеристики реального времени ОС «Эльбрус» на базе стандартного Linux не уступают специализированным операционным системам реального времени.

ЛИТЕРАТУРА

1. Семенихин С.В., Ревякин В.А., Ананьев Л.И. и др. ОС Linux и режим реального времени // Вопросы радиоэлектроники. – 2009. – Сер. ЭВТ. – Вып. 1.

2. McKenney P. A realtime preemption overview. [Электронный ресурс]. – Режим доступа: https://lwn.net/Articles/146861. Дата обращения: 20.01.2016.

3. Rowand F. Using and Understanding the Real-Time Cyclictest Benchmark. [Электронный ресурс]. – Режим доступа: http://eLinux.org/images/0/01/Elc2013_rowand.pdf. Дата обращения: 20.01.2016.

4. Dahl O. Linux Core Isolation and Tickless Operation. [Электронный ресурс]. – Режим доступа: http://www.svenskelektronik.se/ECS/ECS14/Presentations/OSL1.pdf. Дата обращения: 20.01.2016.

5. Махилёв В. Результаты тестов производительности QNX Neutrino // Современные технологии автоматизации. – 2012. – № 2. – С. 82-88.

ИНФОРМАЦИЯ ОБ АВТОРАХ

***Ананьев Леонид Иванович**, к.т.н., ведущий научный сотрудник, АО «МЦСТ», ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д. 24, (499)135-21-58,*

leoan@mcst.ru.

***Семенихин Сергей Владимирович**, д.т.н., проф., зам. генерального директора по науке, АО «МЦСТ», ПАО «ИНЭУМ им. И.С. Брука», 119334, Москва, ул. Вавилова, д. 24, (499)135-20-61, svs@mcst.ru.*