Отладка приложений и настройка параметров конфигурации операционной системы реального времени RTEMS

Николай Баландин, Александр Крапивный (Москва)

В статье рассматривается настройка параметров конфигурации операционной системы реального времени RTEMS и способы отладки приложений при помощи компонентов, предоставляемых пользователю операционной системой, а также возможность использования графического интерфейса.

Введение

Одним из этапов в создании любого программного продукта является отладка и анализ алгоритма работы приложения. Этот касается и приложений, работающих в операционных системах реального времени (ОСРВ), в частности RTEMS. Здесь, помимо правильности обработки данных, возникает вопрос о правильном планировании задач и своевременной реакции задач на события.

Можно проводить анализ приложения в режиме реального времени с целью сбора информации о том, какое количество ресурсов потребляет каждая задача. Таким образом, для приложений реального времени возникает необходимость в дополнительном анализе планирования и быстродействия задач. Операционная система реального времени RTEMS предоставляет разработчику полный набор компонентов для создания точного и удобного механизма анализа и отладки приложения. Об этих компонентах пойдёт речь в данной статье.

Отладчик GDB

В ОСРВ RTEMS нет резидентного отладчика, поэтому стандартный отладчик GDB (GNU Project debugger) может быть сконфигурирован при сборке системы как кросс-отладчик. Отладчик GDB построен по принципу host-target и состоит из двух модулей – отладочного сервера, работающего на целевой платформе, и отладчика на инструментальной платформе; соединение между ними осуществляется через UDP/TCP. Недостатком такого способа отладки является «потеря» режима реального времени при использовании точек останова, поэтому для сохранения режима реального времени приходится использовать другие методы отладки, в первую очередь, это отладка встроенными средствами RTEMS.

Отладка приложений средствами командного интерпретатора RTEMS

ОСРВ RTEMS имеет в своей командной оболочке набор команд для отладки приложений и сбора информации о текущем состоянии системы. Рассмотрим основные из них:

- *mdump* адрес длина – выводит на экран дамп памяти за данной длины, начиная с заданного адреса;
- *wdump* работает так же, как и *mdump*, но выводит дамп по словам;
- *medit* адрес байты – модификация памяти, начиная с заданного адреса, замена на указанную последовательность байт;
- *mmove* назначение источник длина – копирование участка памяти;
- *malloc info|stats* – вывод информации или статистики об используемой памяти.

ОСРВ RTEMS имеет следующие команды оболочки для сбора системной информации:

- *cpuuse* – вывод информации обо всех потоках в системе, времени их выполнения и времени, прошедшего с момента последней загрузки системы;
- *stackuse* – информация об использовании стека каждым потоком;
- *wkspace* – информация об использовании рабочего пространства;
- *config* – вывод конфигурации системы;
- *itask* – список всех задач при инициализации в системе;
- *task* – список всех задач в системе;
- *queue* – вывод информации об очередях сообщений;
- *sema* – вывод информации о семафорах;
- *region* – вывод информации о регионах памяти;
- *part* – вывод информации о разделах в памяти;
- *object* – вывод информации об объектах RTEMS;
- *driver* – отображение таблицы драйверов устройств;
- *netstat* [tcp] [udp] [ip] [icmp] [mbuf] [route] [if] – вывод состояния и статистики стека сетевых протоколов TCP/IP.

Данные команды могут быть использованы только при включении командной оболочки и пригодны в основном для получения информации о системе, для автоматизации анализа или отладки.

Менеджер расширений пользователя

Помимо средств командной оболочки, RTEMS позволяет разработчику использовать набор системных вызовов для сбора информации о состоянии системы и добавлять свои функции, вызываемые ядром RTEMS при переключении контекста в планировщике задач или возникновении неисправимой ошибки (fatal error). В ОСРВ RTEMS такие функции предоставляются разработчику менеджером расширений пользователя. Они служат для обработки системных событий в целях отладки, мониторинга или расширения функциональности системы в целом; RTEMS позволяет вызывать функции пользователя в следующих ситуациях:

- создание задачи (*task creation*);
- запуск задачи (*task start*);
- начало задачи (*task begin*);
- перезапуск задачи (*task restart*);
- завершение задачи (*task exit*);
- удаление задачи (*task delete*);
- переключение контекста задачи (*task context switch*);
- возникновение неисправимой ошибки (*fatal error detection*).

Все вместе, эти функции задаются в RTEMS структурой, содержащей указатели на вызываемые функции пользователя при обработке этих событий. Такая структура называется набором расширений (*extension set*). Структура набора расширений пользователя может быть задана при компиляции приложения и таким образом статически включена в таблицу конфигурации RTEMS, которая содержит все ключевые данные о функционировании системы (объём RAM, таблицу драйверов, максимальное количество ресурсов и т.д.). Такие обработчики будут активны в течение всей работы операционной системы.

Помимо статического способа задания таблицы расширений, RTEMS позволяет динамически устанавливать и удалять свои обработчики переключения контекста. Обработчики пользователя могут быть использованы при мониторинге активности задач (профилирование), фиксировании моментов наступления неисправимых ошибок или при контроле состояния стека при переключении задач. Преимущество динамически подключаемых обработчиков состоит в том, что они могут быть удалены «на лету», если, например, понадобится увеличить производительность системы.

Недостатком динамического подключения набора расширений является то, что зарегистрироваться в системе он сможет только после начального этапа инициализации RTEMS (функция *initialize_executive*); таким образом, динамически подключаемыми наборами расширений не удастся обрабатывать ошибки, возникающие на этапе инициализации системы. Это может происходить при вызове функции *rtems_fatal_error_occurred*, которая сигнализирует о наступлении неисправимой ошибки и может вызываться как из кода ядра RTEMS, так и из приложения. Отслеживать ошибки при инициализации RTEMS можно только при помощи статического набора расширений, который с самого начала инициализации RTEMS будет перехватывать вызов функции *rtems_fatal_error_occurred*.

ОСРВ RTEMS позволяет нескольким наборам расширений быть одновременно активными. Это обеспечивает одновременную загрузку разных наборов расширений, каждый из которых будет выполнять определённую задачу. При вызове функция пользователя будет получать в качестве параметров указатель на системную структуру TCB (Thread Control Block – блок управления потоком), которая содержит всю служебную информацию о потоке. Таким образом, пользователь может работать с этой структурой в целях отладки или расширения функциональности операционной системы.

Динамический набор расширений создаётся функцией *rtems_extension_create*, принимающей в качестве параметров имя расширения, указатель на таблицу обработчиков и указатель на идентификатор расширения, который будет выдан в случае его успешной регистрации в системе. Удаляется динамически подключаемый набор расширений в любой момент при помощи вызова *rtems_extension_delete*.

Кроме установки своих обработчиков при переключении контекста задач или при возникновении ошибок, программист может указать при компиляции, какая задача будет выполняться в системе при отсутствии других готовых к выполнению задач (*idle task* – задача бездействия системы). По умолчанию задача бездействия системы представляет собой бесконечную петлю (указатель на функцию, которая выполняется при отсутствии других активных задач, содержится в таблице конфигурации системы).

Для предотвращения возможных ошибок переполнения стека внутри потока, RTEMS предоставляет режим отладки со слежением за состоянием стека потока (модуль проверки границ стека – *Stack Bounds Checker*). Необходимость контроля состояния стека указывается определением имени *STACK_CHECKER_ON* при компиляции, которое включается в таблицу конфигурации системы. При создании новой задачи в RTEMS одним из параметров является размер стека, который будет использовать эта задача. Если в системе активен модуль контроля состояния стека, то при создании новой задачи её область стека заполняется контрольными значениями для индикации того, что данный адрес в стеке ещё не был использован. При выполнении задачи, по мере использования стека эти значения удаляются. При переключении контекста задачи модуль проверки границ стека проверяет, во-первых, не повреждены ли последние *N* байт контрольных значений и, во-вторых, попадает ли указатель стека задачи в заданный интервал. Если нарушено одно из этих условий, значит, поток превысил максимально допустимую границу, и при помощи команды *printk* будет сделана попытка вывода сообщения об ошибке переполнения стека.

Число *N* последних байт, которые будут проверены, зависит от модели процессора. Также внутри самого потока можно вызывать системную функцию для проверки собственного стека потока (*rtems_stack_checker_is_blown*). Также есть функция, докладывающая информацию об использовании стека каждой из задач (*rtems_stack_checker_report_usage*). Платой за подключение этого модуля является увеличение времени создания задачи и переключения контекста, т.к. данный модуль не может быть отключен «на лету». Если необходимо динамически включать или выключать контроль стека, то осуществить это можно при помощи динамически подключаемого набора расширений. Перечисленные выше компоненты ядра RTEMS дают возможность разработчику осуществлять анализ работы приложения, не выходя из режима реального времени.

Таблица конфигурации системы

Операционная система RTEMS должна быть правильно сконфигурирована под конкретное приложение. Информация для настройки системы включает все основные данные о ресурсах – период переключения контекстов задач, используемые драйверы, максимальное число дескрипторов в системе и т.д. Вся эта информация из таблицы конфигурации в заголовочном файле *rtems/confdefs.h* помещается RTEMS в системные структуры данных на этапе инициализации. Не указанные в программе пользователя параметры задаются по умолчанию. Сама таблица состоит из следующих разделов:

- поддержка библиотек файловой системы и ввода-вывода (*filesystem and IO library support*) – параметры сбора статистики по вызовам *malloc*, максимальное количество файловых дескрипторов, выбор корневой файловой системы (*IMFS/mini-IMFS*), указание собственной таблицы для монтирования разделов, включение модуля проверки границ стека;
- основная системная информация (*basic system information*) – количество микросекунд за один системный такт (*clock tick*), количество тактов за один квант времени (*timeslice quantum*) для каждой задачи, максимальный приоритет любой задачи в системе, минимальный размер стека для каждой задачи и стека прерываний, указатели на функции пользователя по размещению и удалению стека;
- конфигурация задачи бездействия системы (*idle task*) – указатель на задачу бездействия системы, предоставленную пользователем, размер стека для задачи бездействия системы;
- таблица драйверов устройств (*device driver table*) – максимальное количество зарегистрированных драйверов в системе, параметры использования драйвера консоли, часов, таймера, часов реального времени, сторожевого таймера, драйвера */dev/null*, драйверов для приложения;
- таблица конфигурации *RTEMS API* – максимальное число задач, таймеров, семафоров, наборов расширения и т.д.;
- таблица конфигурации *POSIX API* – максимальное число ресурсов для POSIX;
- таблица конфигурации *ITRON API* – максимальное число ресурсов для ITRON.

Пример с использованием модуля расширений

В качестве примера использования функций *Extension Manager* рассмотрим небольшую программу, регистрирующую в системе динамически подключаемый набор расширений. Полный текст программы и *Makefile* к ней находятся на сайте журнала. Программа будет регистрировать набор расширений, затем создаст две новые задачи и уничтожит себя.

Работа наших расширений будет заключаться в выводе информации о задачах – их имён, идентификаторов, приоритета, времени выполнения. Для создания набора расширений необходимо объявить свои функции – обработчики и структуру, состоящую из указателей на эти функции. Последняя состоит из восьми обработчиков для следующих ситуаций: создание задачи, старт задачи, перезапуск задачи, уничтожение задачи, переключение, начало, выход, неисправимая ошибка.

```
// функции расширений Extension
Manager
bool my_task_create (rtems_tcb *
curr, rtems_tcb * next);
rtems_extension my_task_start
(rtems_tcb * curr, rtems_tcb *
next);
rtems_extension my_task_restart
(rtems_tcb * curr, rtems_tcb *
next);
rtems_extension my_task_delete
(rtems_tcb * curr, rtems_tcb *
deleted);
rtems_extension my_task_switch
(rtems_tcb * curr, rtems_tcb *
```

```
next);
rtems_extension my_task_begin
(rtems_tcb * curr);
rtems_extension my_task_exit
(rtems_tcb * curr);
rtems_extension my_fatal_error
(Internal_errors_Source src, bool
internal, uint32_t error_code);
// набор функций-обработчиков
системных событий
rtems_extensions_table MyExtTable
= {
my_task_create,
my_task_start,
my_task_restart,
my_task_delete,
my_task_switch,
my_task_begin,
my_task_exit,
my_fatal_error };
```

Объявленная нами структура *MyExtTable* и будет использована при регистрации набора расширений. Как и всякий объект в RTEMS, наш набор расширений должен иметь своё 4-байтовое имя и идентификатор:

```
// регистрируем расширения
ext_name = rtems_build_name('E',
'X', 'T', '1');
status = rtems_extension_create
(ext_name, &MyExtTable, &ext_id);
if (status!=RTEMS_SUCCESSFUL)
printf ("Error: cannot create ex-
tension with %d\n", status);
else printf («Extension
created\n");
```

Если при регистрации произошла ошибка с кодом *RTEMS_TOO_MANY* – слишком много объектов, – это означает, что в таблице конфигурации RTEMS было указано недостаточное количество ресурсов данного типа или мы вообще забыли его указать. Для RTEMS это является типичной ошибкой разработчика по невнимательности – для каждого типа ресурсов следует указывать их максимальное количество; по умолчанию RTEMS может и вовсе не выделить для них места, как, например, с набором расширений, поэтому указываем максимальное число ресурсов:

```
#define
CONFIGURE_MAXIMUM_USER_EXTENSIONS 1
```

Теперь займёмся самими обработчиками. Вызываться они будут при смене контекста у каждой из задач или при возникновении неисправимой ошибки. Пусть наш обработчик переключения задач *my_task_switch* выводит на экран идентификатор задачи, с которой происходит переключение, и время, которое она выполнялась. Указатель на структуру ядра RTEMS с информацией о задаче, с которой происходит переключение контекста, передаёт аргумент *rtems_tcb * curr*.

```
printk("*** current task: id
[%d], time elapsed [%d] \n",
curr->Object.id,
curr->cpu_time_used );
```

Поле *Object* в структуре *tcb* содержит информацию об объекте (имя, идентификатор), а поле *cpu_time_used* содержит время выполнения данной задачи. Полное описание этой структуры находится в заголовочном файле *rtems/score/thread.h*, который содержит всю необходимую информацию о потоке как, например, приоритет при

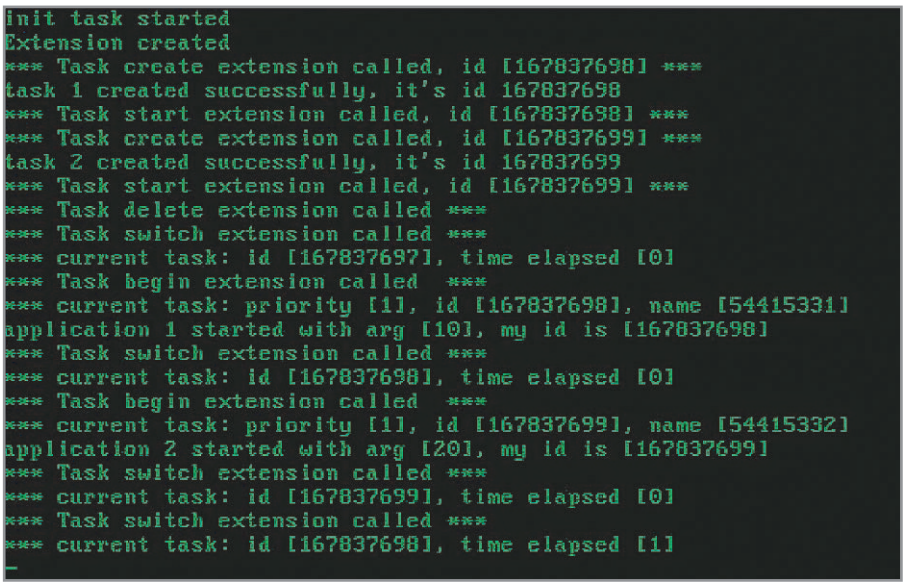

Рис. 1. Пример работы набора расширений

создании потока и его текущий приоритет, количество ресурсов, контекст регистров.

Теперь осталось решить проблему порядка переключения задач и частоты их переключения. При быстром переключении задач мы не будем успевать считывать с экрана сообщения от обработчиков, поэтому сделаем так, чтобы они переключались один раз в секунду. Для этого мы будем создавать задачи с атрибутом *RTEMS_TIMESLICE*, который будет для каждой задачи выделять определённый квант времени, по истечении которого она переключится на другую задачу с таким же приоритетом:

```
name = rtems_build_name('T', 'A',
'S', '1'); // формируем имя потока
status = rtems_task_create (name,
1, RTEMS_MINIMUM_STACK_SIZE,
RTEMS_TIMESLICE,
RTEMS_FLOATING_POINT, &id);
```

Теперь необходимо задать число микросекунд на системный такт и число тактов на квант времени для задачи:

```
#define
CONFIGURE_MICROSECONDS_PER_TICK
1000
#define
CONFIGURE_TICKS_PER_TIMESLICE
1000
```

Один квант будет занимать 1000 × × 1000 = 1 000 000 мкс, или 1 с, т.е. планировщик RTEMS будет переключать задачи 1 раз в секунду. Пример работы приложения показан на рисунке 1.

Статистика использования центрального процессора

ОСРВ RTEMS имеет компонент, отвечающий за сбор информации об использовании центрального процессора – менеджер статистики использования ЦП. Во время анализа и отладки приложений реального времени важно точно знать, какое количество времени центрального процессора потребляет каждая из задач. RTEMS получает эту информацию, запоминая при каждом переключении контекста, сколько системных тактов выполнялась задача. Если задача выполнялась меньше одного системного такта, то предполагается, что она заняла один такт. Приложение может динамически «докладывать» об использовании центрального процессора при помощи вызова функции *rtems_cpu_usage_report*. Также статистика может быть сброшена в любой момент времени вызовом функции *rtems_cpu_usage_reset*.

Графический пакет PicoTK

В базовом виде RTEMS не включает в себя поддержку графического режима, однако для платформ i386 имеется несколько дополнений для RTEMS, позволяющих работать с графическим режимом. В этой статье речь пойдёт о графическом пакете *PicoTK* как о наиболее простом в настройке прикладном интерфейсе.

Как следует из названия, *PicoTK* является небольшим пакетом и служит для простого представления информации, в отличие от других пакетов поддержки графики, таких как *Qt/embedded* или *NanoX*, ориентированных на более сложный оконный интерфейс.

Пакет *PicoTK* располагает следующими возможностями:

- поддерживает глубину цвета 1, 4 или 8 бит, соответственно 2, 16 или 256 отображаемых цветов;
- позволяет рисовать точки, линии, текст, битовые изображения;
- поддерживает разрешения 320×200, 640×480, 800×600, 1280×1024;
- имеет эмулятор для Linux/X11, использующий механизм разделяемой памяти.

Использование эмулятора значительно ускоряет этап разработки части приложения для RTEMS, отвечающей за графическое представление информации, за счёт кросс-компиляции и запуска графического модуля приложения для RTEMS в ОС Linux. Эмулятор поддерживает разрешение 640 × 480.

Эмулятор *fbe* (frame buffer emulator – эмулятор буфера кадра) работает по принципу последовательного чтения кадра из участка разделяемой памяти между собой и графическим приложением. Поступающее изображение из участка разделяемой памяти отображается через *X-Windows*.

Для установки *PicoTK* на ПК, где ведётся разработка приложений для RTEMS, необходимо наличие X-сервера и библиотек *Xlib*. RTEMS поддерживает *PicoTK*, начиная с бета-версии 4.5.0. Рассмотрим особенности установки и настройки пакета *PicoTK* для работы с RTEMS.

Помещаем архив с *PicoTK* в домашнюю папку и распаковываем:

```
$ tar -xzf picotk-xx.tgz
```

Собираем *PicoTK* (компилируем из X, при сборке из текстовой консоли будет выдана ошибка):

```
$ cd picotk
$ make
```

Запускаем пример приложения на *PicoTK* в эмуляторе:

```
$ emulators/fbe & demo/demo
```

Далее создаём статическую библиотеку пакета *PicoTK* для RTEMS *libPTKrtems.a*:

```
$ export PATH=$PATH:/opt/rtems-
4.9/bin
```

```
$ export
RTEMS_MAKEFILE_PATH=/{rtems in-
stall point}/i386-rtems/pc386/
$ make rtems
```

В результате сборки в папке *demo/o-optimize* появится файл *demo.exe* – приложение в RTEMS с использованием *PicoTK*; его работа показана на рисунке 2.

Заключение

Написание программы для микроконтроллера, содержащей одну-две выполняемые задачи, обычно не представляет большой сложности. Однако с увеличением количества задач и учётом приоритета их выполнения, написание программы сильно усложняется. Поэтому в настоящее время всё большую популярность приобретают операционные системы реального времени. Главным преимуществом ОСРВ является освобождение разработчика от написания планировщика задач, межзадачного взаимодействия, синхронизации между задачами, процедур обслуживания прерываний и многих других, нетривиальных функций.

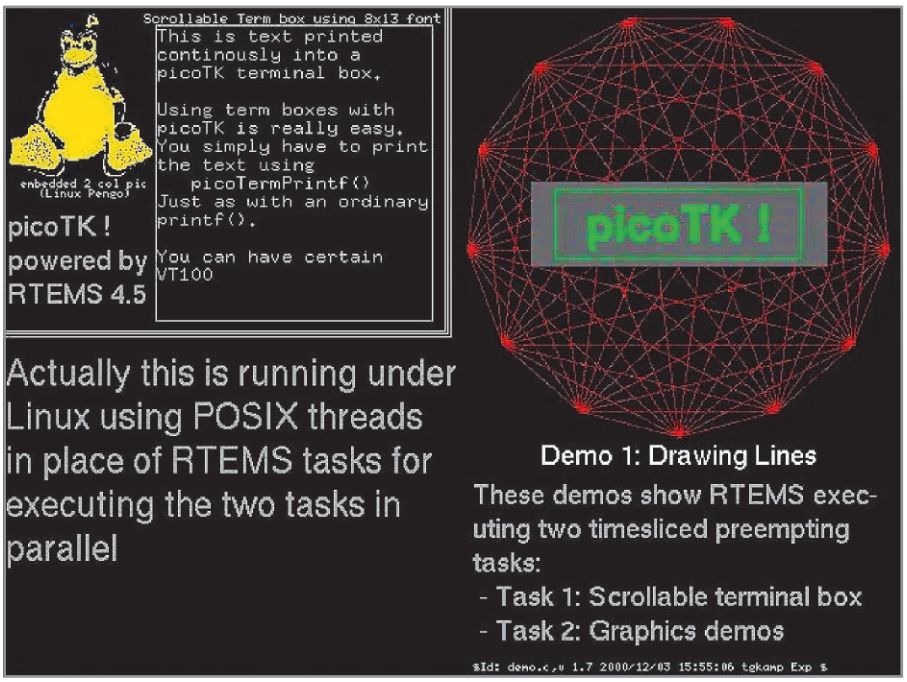

Рис. 2. Приложение для RTEMS с использованием *PicoTK*

ОСРВ RTEMS является современной операционной системой «жёсткого» реального времени, поддерживающей возможность интеграции графического интерфейса, в которой, благодаря открытому исходному коду, разработчик может оптимально настроить работу своего приложения под конкретную задачу.

Литература

1. RTEMS Applications C User's Guide.
2. RTEMS POSIX API User's Guide.
3. RTEMS CPU Supplement.