Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Разработка системы тестирования программно-аппаратных комплексов

Курсовая работа студента 445 группы
Батюкова Александра Михайловича

Научный руководитель ……. А. Бондарев

Санкт-Петербург
2010

Оглавление

Введение

Предметная область

Вопросы, затрагиваемые в курсовой работе, относятся к области разработки встраиваемых программно-аппаратных систем.

Встраимаемая система (встроенная система, англ. Embedded system) — это специализированная компьютерная система, предназначенная для выполнения определенных функций, часто в реальном времени. Термин «встраиваемая» подразумевает, что такие системы обычно встроены в состав конкретного устройства/оборудования и выполняют некоторые управляющие/контролирующие функции, связанные с функционированием устройства.

Физически встраиваемая система может быть как простейшим портативным устройством (например, расположнным на одной микросхеме микроконтроллера), так и сложной системой с множеством выполняемых функции и нетривиальной внутренней логикой.

Основной особенностью встраиваемых систем является высокий уровень оптимизации под конктретную задачу, причем как на уровне программного обеспечения, так и на аппаратном уровне. Заточенность под конкретную задачу, заложенная в самом определении встраиваемой системы, позволяет при совместной работе инженеров и программистов создавать эффективные и надежные программно-аппаратные решения.

Важно осознавать, что разработка встраивамых систем, таким образом, возможна только при тесном взаимодействии между инженерами (которые разрабатывают аппаратуру) и программистами (которые пишут ПО).

Еще одной особенностью предметной области, важной для рассмотрения, является достаточно высокий уровень вхождения для программиста. В отличие от некоторых других IT-областей кроме базовых знаний в области разработки программного обеспечения зачастую требуются фундаментальные знания, на первый взгляд мало связанные с программированием.

Цель курсовой работы

При разработке встраиваемых систем работа инженеров и программистов ведется параллельно. Естественно, что в процессе разработки и с той и с другой стороны возникают ошибки. Со стороны инженеров это могут быть ошибки в железе, программисты могут ошибаться в программной части.

Цена каждой такой ошибки очень высока, так как зачастую с ней приходится сталкиваться людям, далеким от той области, где она возникла. Так, например, инженер не обязан разбираться в коде программы, а для программист — в причинах некорректной работы процессора.

Таким образом возникает естественная идея — создать некое инструментальное средство для удобного поиса и устранения ошибок, возникающих в процессе разработки встраиваемых систем. Будем называть это инструментальное средство в дальнейшем «Тестовая система».

Сформулируем некоторые требования, которым должно удовлетворять тестовая система:

- всеобъемлимость — тесты, представленные в системе должны полностью тестировать и аппаратуру, и программный код
- универсальность — система не должна быть привязана ни к какому железу или программному коду
- удобство — несмотря на всеобъемлимость, система не должна быть громоздкой, ей должно быть удобно пользоваться
- гибкость — система должна быть приспособлена к тестам разного характера, как к тестам железа, так и к тестам ПО

Целью курсовой работы будет разработка системы тестирования, удовлетворяющей указанным выше требованиям и ее программная реализация.

Обзор существующих решений

В данный момент в области разработки встраиваемых систем можно выделить следующие Тестовые системы, подходящие под данное выше определение:

- тестовая система OS eCos
- тестовая система OS RTEMs
- система самотестирования Linux

Остальные систем тестирования либо похожи на вышеперечисленные, либо не достойны рассмотрения.

Опишем основные проблемы, возникающие при работе с каждой из представленных выше тестовых подсистем, что позволит четче сформулировать требования, выдвигаемые к разрабатываемому инструментальному средству.

тестовая система OS eCos

- недостаточно всеобъемлюща, нет удобного интерфейса для тестов аппаратуры и архитектурно-зависимых частей системы
- мало приспособлена для организации автоматического тестирования
- достаточно громоздка (файлы конфигурации /home/ecos/config/ecos.ecc порядка 400 килобайт)

тестовая система OS RTEMs

- достаточно сложна в настройке, требует вмешательства в исходные коды системы для включения новых тестов
- мало приспособлена для разработки, лишь информирует о неполадке, не давая возможности ее исследования
- запуск тестов возможен только после инициализации всего оборудования (то есть тест по стуи не отличается от обычной функции)

система самотестирования Linux

- мало приспособлена для разработки по тем же причинам

- громоздка

Кроме того Linux — слишком сложная система для того, чтобы постоянно контролировать ее поведение, как тестовая система он малопригоден. Linux не ориентирован на встраиваемые системы.

Обзор существующих решений позволяет выделить дополнительные требования, которым, по моему мнению, должна удовлетворять приемлемая для разработки встраиваемого ПО тестовая система:

- простота в настройке, добавление теста должно быть достаточно просто для не слишком опытного человека
- должна предоставлять удобный интерфейс для тестирования аппаратуры и архитектурно-зависимых частей системы
- должна быть приспособлена для организации автоматического тестирования
- должна предоставлять возможность детального тестирования системы на разных уровнях инициализации оборудования
- кроме того должна быть в достаточной степени конфигурируема и по возможности не громоздка

Описание решения

Реализация описанной системы тестирования будет проводиться в рамках open-source проекта embox. Название проекта расшифровывается как «Essential toolbox for embedded development», из чего можно понять цели создающих его разработчиков. Проект позиционируется как высококонфигурируемая инструментальная система для применения на всех этапах разработки встраиваемых систем. В одном из возможных вариантов конфигурации embox выступает как полноценная тестовая система.

Исходный код проекта можно найти по адресу *http://code.google.com/p/embox/.*

Единая система тестирования

Зачастую проект разработки встраиваемой системы длится достаточно долго. За время разработки накапливается достаточно много разнообразных тестов, таких как

- тесты аппаратуры (например, тесты памяти)
- тесты на функциональность (при добавлении новой функциональности пишут тесты на нее)
- тесты, направленные на выявление и устранение ошибок разного рода

Все это множество тестов необходимо хранить на протяжении всего проекта. Более того, это множество тестов следует хранить и после окончания проекта с целью переиспользования кода в дальнейших разработках. Все это приводит к мысли организовать единую базу основных тестов, которые могут пригодиться при разработке встраиваемой системы. В исходные коды embox включен набор основных тестов, таких как тесты памяти, тесты на прерывания, тесты работы процессора, тесты работоспособности таймера и некоторые другие. Код тестов можно располагается в */src/tests/*.

В процессе разработки при появлении нового теста можно включить его в базу. Процесс добавления теста описан в Приложении 1. В соответствие со сформулированными к тестовой системе требованиями процесс добавления нового теста обладает следующими особенностями:

- не требует вмешательства в исходные коды системы, не относящиеся к реализации теста. От программиста требуется только реализовать функциональность теста и оформить функции в соответствие с стандартизированным, описанным в Приложении 1 интерфейсом.

- позволяет указать уровень выполнения теста (то есть момент инициализации системы, в который будет выполнен конкретно этот тест). Например, тесты памяти можно выполнять при старте системы,еще до инициализации драйверов устройств.

Модульность

Каждый тест в системе представлен как отдельно подключаемый и конфигурируемый модуль. Это означает, что любой тест можно включить в текущую сборку системы независимо от других тестов. Такой подход позволяет создавать как тестовые образы, направленные на тестирование одной конкретной части системы (например, тесты памяти), так и полноценные тестирующие все части системы и взаимодействие между ними.

При создании тестового образа часто критичен размер получаемого для исполнения файла. Это может быть важно, например, при тестировании оперативной памяти, когда нет возможности записать код программы в оперативную память, а остальные ресурсы памяти сильно ограничены.

Кроме того для запуска тестов на потактовом симуляторе, где скорость выполнения инструкции крайне невелика, важно чтобы требуемые тесты начинали выполнение как можно быстрее после старта системы. Например, тесты памяти можно проводить практически сразу после начальной инициализации процессора и основных драйверов.

Легко представить себе ситуацию, когда для выполнения теста требуется наличие в системе определенного драйвера. Для этого все остальные части системы, например, драйвера устройств, тоже представлены в системе как модули. Можно указать в Makefile зависимость теста от требуемого для его выполнения модуля (как это сделать описано в Приложении 1) и система гарантирует, что нужный модуль будет проинициализирован до начала выполнения теста. Это реализовано благодаря возможностям системы конфигурации embox, спроектированной Антоном Бондаревым и Эльдаром Абусалимовым.

Интерфейс тестирования архитектурно-зависимых частей

При разработке встраиваемых систем возникают проблемы тестирования архитектурно-зависимых частей системы, например, таких как mmu или исключительные ситуации процессора (traps, трепы). Такие части, как правило, присутствуют всегда. По понятным причинам чать функциональности таких тестов должна быть реализована на ассемблере.

Кроме того возникают дополнительные сложности, связанные с необходимостью предоставить возможность детального тестирования системы. По вполне понятным при-

чинам после завершения теста система должна вернуться в состояние, полностью идентичное тому, которое было до проведения теста (ведь тест по определению не может никак влиять на систему, он ее только проверяет). Но, например, тестирование трепов немыслимо без установки тестовых обработчиков прерываний, что означает необходимость изменения таблицы прерываний на время теста. Похожие рассуждения верны и для других архитектурно-зависимых частей, например, mmu. Как итог, нужен некоторый механизм, обеспечивающий возврат системы из теста в рабочее состояние.

Ассемблерный код и механизм восстановления рабочего окружеия скрыты в интерфейсе тестирования архитектурно-зависимых частей системы.

Интерфейс описан в файлах /src/include/hal/tests и /src/include/hal/env. Функции имеют общую сигнатуру для всех архитектур. Архитектурно зависисмая реализация функций располагается в /src/arch/.

Механизм сохранения/восстановления рабочего окружения реализован с помощью функций, работающих со структурой типа _<prefix>_env_t. Для каждой архитектуры эта структура переопределена по-своему и содержит всю необходимую информацию для восстановления рабочего окружения.

Организация регрессионного тестирования

Регрессио́нное тести́рование (англ. regression testing, от лат. regressio — движение назад) — собирательное название для всех видов тестирования программного обеспечения, направленных на обнаружение ошибок в уже протестированных участках исходного кода.

В нашем случае под регрессионным тестированием мы будем понимать повторные прогоны тестов, созданных после исправления той или иной ошибки, и хранящихся в базе тестов.

Повторное появление одних и тех же ошибок — случай достаточно частый. Иногда это происходит из-за человеческой ошибки при работе с системой управления версиями. Может возникнуть ситуация, когда после следующего изменения в программе что-то из предыдущего перестаёт работать. И наконец, при переписывании какой-либо части кода часто всплывают те же ошибки, что были в предыдущей реализации. Поэтому считается хорошей практикой при исправлении ошибки создать тест на неё и регулярно прогонять его при последующих изменениях программы. Естественно пожелать делать прогоны тестов автоматически.

Для удобной организации регрессионного тестирования от тестовой системы в первую очередь требуется возможность автоматической генерации набора тестовых загрузочных образов. Это позволяет с помошью скриптов поочередно загружать их на симулятор программно-аппаратного комплекса и проверять работоспособность всех частей системы без участия человека.

Описание результата

В рамках курсовой работы реализована тестовая система проекта embox.

Создана единая база тестов, включающая в себя основные тесты как программных, так и аппаратных частей системы. Механизм добавления в систему нового теста описан в Приложении 1 и достаточно прост. В рамках организованного на базе проекта embox студенческого проекта компании «Lanit-Tercom» с добавлением нового теста в систему справилось подавляющее большинство студентов 2 курса, не обладающих опытом разработки в предметной области.

Система тестирования в достаточной степени модульна и конфигурируема. Предоставляется возможность добавления в систему произвольной выборки тестов из реализованных. При тестировании памяти за счет использования модульности удается ограничить размер исполняемого файла 40 килобайтами.

Указав уровень выполнения теста «сразу после начальной инициализации» удается дождаться запуска тестов памяти не более чем через 20 минут после старта симуляции на потактовом симуляторе. Для сравнения в аналогичных системах это время составляет порядка часа, а загрузки Linux дождаться не представляется возможным.

Разработан общий для всех представленных в проекте процессорных архитектур (sparc v8 и Microblaze) интерфейс тестирования mmu и traps. Реализованы механизмы сохранения/восстановления контекста, благодаря чему тестирование происходит абсолютно прозрачно для системы.

В рамках проекта embox реализована возможность множественной генерации образов для загрузки, что позволяет использовать систему тестов для регрессионного тестирования.

Заключение

В итоге разработана система тестирования в целом удовлетворяющая поставленным требованиям. Система применяется в коммерческих проектах, ей пользуются как программисты, так и инженеры, разрабатывающие встраиваемые системы.

Приложение 1. Описание процесса добавления нового теста.

Добавление теста

Для добавления нового теста в систему в одной из этих папок нужно создать свою папку с тестом (см. шаблон src/tests/_test_template). В ней обязательно должны быть как минимум 2 файла следующего содержания:

- Файл с исходным кодом теста
- makefile для сборки этого теста в системе

Требования к исходному коду теста

Тесты представлены в системе как модули пакета `embox.test` или `<platform>.test`. Для того что бы система могла распознать тест, его нужно определить следующим образом:

- Реализовать функцию выполнения теста.
- Объявить тест в системе с помощью соответствующего макроса.

Функция выполнения

Каждый тест должен иметь функцию выполнения, которая запускается системой при вызове данного теста, со следующей сигнатурой:

```
int run(void);
```

Функция возвращает 0 в случае прохождения теста и любое другое значение, если тест не пройден. Если для теста задана функция подробного описания, то это значение будут передано ей в качестве аргумента.

Функция подробного описания

Каждый тест может иметь функцию подробного описания, определить ее нужно следующим образом:

```
void (*info)(int);
```

Эта функция вызывается по требованию пользователя в случае провала теста и должна напечатать причину неудачи. В качестве аргумента передается последнее ненулевое значение, возвращенное функцией выполнения теста.

Макросы для объявления тестов

Для объявления теста в системе необходимо указать некоторую информацию, в первую очередь, функция выполнения теста. Это делается с помощью следующих макросов объявленных в файле <embox/test.h>:

- `EMBOX_TEST(run);` - объявляет тест в системе. В качестве имени теста по умолчанию используется имя модуля, описанного в makefile'e.
- `EMBOX_TEST_DETAILS(run, "name", info);` - позволяет задать для теста произвольное имя и функцию детальной информации.

Макросы для экспортирования и импортирования тестов

Экспорт и импорт тестов нужен, когда из одного теста хочется вызвать другой, не прибегая к стандартному вызову тестов через фреймворк. Для этого применяются макросы:

- `EMBOX_TEST_EXPORT(<symbol_name>);`
- `EMBOX_TEST_IMPORT(<symbol_name>);`

В вызываемом тесте указываем макрос EMBOX_TEST_EXPORT, в вызывающем - EMBOX_TEST_IMPORT, с одинаковыми аргументами. Тест вызывается по имени, заданному в аргументе.

Пример кода минимального теста

```
#include <embox/test.h>

EMBOX_TEST(run);

static int run(void) {
            int result = 0;

            /*
            * The test itself.
            */

            return result;
}
```

Требования к содержимому makefile

- Makefile должен быть описан так:

```
$_MODS += <test_name>
$_SRCS-<test_name> += *.c
```

В Makefile могут быть описаны зависимости от других модулей системы, например

```
$_DEPS-<test_name> += <dep>
```

Конфигурирование теста

Необходимо добавить в файл конфигурации mods-tests.conf запись о новом тесте:

```
test(<test_name>, <runlevel>)
```

или

```
test_platform(<test_name>, <runlevel>)
```

для тестов платформо-зависимых частей системы. Где:

- `<test_name>` - имя модуля теста, заданное в makefile'е.
- `<runlevel>` - (необязательно) уровень выполнения системы, на котором следует запускать тест

Тесты могут выполняться в автоматическом режиме; любой тест можно запустить вручную из консоли командой test.

Список литературы

1. Таненбаум Эндрю, «Архитектура компьютера», 5-е издание, - СПб.:Питер, 2009. - 844 с.: ил.
2. Таненбаум Эндрю, «Современные операционные системы», 2-е издание, - СПб.:Питер, 2009. - 758 с.: ил.
3. Одинцов Игорь. «Профессиональное программирование. Системный подход.» - БХВ-Петербург, 2006. - 624 стр.:ил.
4. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. — Пер. с англ. — СПб.: Символ-Плюс, 2001. — 304 стр.: ил.
5. http://www.ecos.sourceware.org
6. http://www.rtems.com
7. http://www.denx.de/wiki/u_boot
8. http://code.google.com/p/embox/w/list
9. http://en.wikipedia.org