ARINC 653 и ЯП

Метод адаптации ARINC 653 - совместимой ОСРВ для ПО, написанного на разных языках программирования

Pavel Isachenko

MIPT

26 марта 2024 г.

План

Что такое ОСРВ

Операционная система реального времени (ОСРВ) — операционная система, способная обеспечить требуемый уровень сервиса в определённый промежуток времени. Бортовые ОСРВ используются для управления бортовым оборудованием.

Когда кто-то говорит: «Я хочу язык программирования, который может делать все, что ему скажу», то я даю этому человеку леденец.
— Alan J. Perlis

Мотивация

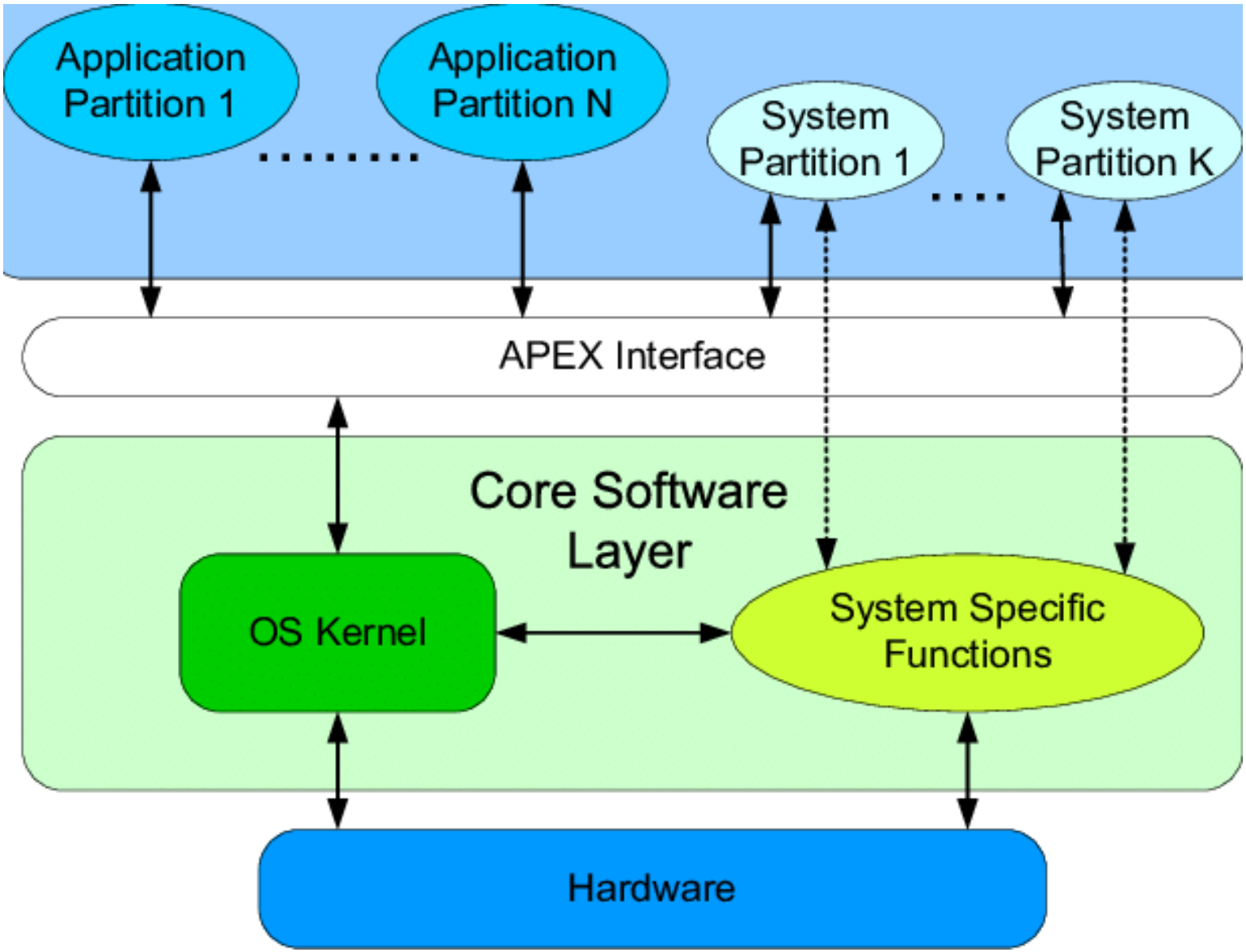
На данный момент существует огромное множество ЯП. Растет количество систем реального времени. Возникает вопрос об интеграции языковых средств (библиотеки, компиляторы, интерпретаторы и т. д.) в ОСРВ.
Современные операционные системы, используемые в аэрокосмической отрасли, должны обеспечивать высокую надежность и разрабатываются в соотествии со специальными стандартами. Одним из стандартов на программное обеспечение, используемое в бортовом оборудовании, является ARINC 653.

Цель

Разработать метод адаптации компилируемого языка программирования для разработки функционального ПО в ARINC 653 совместимых ОСРВ

Что дает нам стандарт

Спецификация ARINC 653[1] дает определение APplication EXecutive (APEX), который поддерживает пространственное и временное разделение приложений. Системы ARINC 653 состоят из разделов программного обеспечения. Каждый раздел представляет собой отдельное приложение, и для каждого раздела имеется выделенное пространство памяти, что обеспечивает разделение пространства. Аналогично, APEX предоставляет выделенный интервал времени для каждого раздела для поддержки разделения по времени. ARINC 653 поддерживает многозадачную среду в каждом разделе посредством четко определенного набора механизмов взаимодействия процессов и планирования на основе упреждающего приоритета.


Application Partition 1
Application Partition N
System Partition 1
System Partition K
APEX Interface
Core Software Layer
OS Kernel
System Specific Functions
Hardware

APEX interface

Основная цель ARINC 653 — определить универсальный Интерфейс APEX interface (API = Application Program Interface) между основным программным обеспечением CSW компьютера авионики и прикладным программным обеспечением.

Задачи общие для любого ЯП

1. Описать и реализовать APEX interface ЯП
2. Создать специфичный тулчейн для ЯП
3. Адаптировать особенности ЯП к APEX окружению
4. Адаптировать runtime ЯП

Пример

ARINC 653 P2 дает нам определение: The types FILE_ID_TYPE and DIRECTORY_ID_TYPE are used to define identifiers for open files and directories. Identifiers for open files and directories remain open until closed or a partition restart occurs.... Мы же в свою очередь должны реализовать это определение на выбранном нами ЯП.

```
type FILE_MODE_TYPE is (READ, READ_WRITE);
type MESSAGE_SIZE_TYPE is a numeric type; -- number of bytes
type FILE_ERRNO_TYPE is numeric type;
type FILE_NAME_TYPE is a n-character string;
type FILE_ID_TYPE is numeric type;
type DIRECTORY_ID_TYPE is numeric type;
```

Пример

В соответствии с ARINC 653 ресурсы выделяются только на стадии инициализации, после не освобождаются

Почему Ada

Некторые из причин:

1. Одним из требований при разработке языка была максимально лёгкая читаемость текстов программ, даже в ущерб лёгкости написания
2. Язык построен таким образом, чтобы как можно большее количество ошибок обнаруживалось на этапе компиляции.
3. Детали синтаксиса разработаны так, чтобы снизить вероятность случайных ошибок. Например, в идентификаторах запрещено использовать несколько знаков подчёркивания подряд.
4. ГЛАВНОЕ: APEX interface описан в спецификации ARINC 653 P1, остается его реализовать

Болтовня ничего не стоит. Покажите мне код. — Linus Torvalds

```
with Ada.Text_IO;

procedure Hello is
  use Ada.Text_IO;
begin
  Put_Line("Hello, world!");
end Hello;
```

Текущее состояние

1. Собран тулчейн с поддержкой Ada на базе GNAT.
2. Реализована базовая Ada Runtime Library для powerpc.

О тулчейне

В простом случае это компилятор и линковщик, приспособленные для создания исполняемых файлов под отличную архитектуру. Для Ada есть множество копмиляторов, я остановился на GNAT. GNAT — свободный компилятор языка Ада, является частью GNU Compiler Collection. GCC собирается с флагами –disable-libada –enable-languages=ada

О libada

Библиотека времени выполнения Ada отвечает за реализацию стандартной библиотеки. Собирается для каждой платформы и соответственно архитектуры отдельно, содержит различные настройки к примеру: разрешены ли исключения, рекурсия, нужно ли проводить проверки стека. Компилируемая программа статически связывается с библиотекой времени выполнения для создания окончательного исполняемого файла.

Будущее

1. Написать libada для других архитектур.
2. Реализовать APEX interface для Ada.
3. Адаптировать особенности языка к APEX окружению
4. Суммаризировать полученные знания.