Лекция 7 Архитектура программного обеспечения

План курса «Встраиваемые микропроцессорные системы»:

Лекция 1: Введение. Язык программирования С

Лекция 2: Язык программирования С, применение для встраиваемых систем

Лекция 3: Стандартная библиотека языка С

Лекция 4: Ядро ARM Cortex-M3. Микроконтроллер Миландр К1986ВЕ92QI

Лекция 5: Этапы разработки микропроцессорных систем

Лекция 6: Разработка и отладка программ для встраиваемых систем

Лекция 7: Архитектура программного обеспечения

Лекция 8: Периферийные модули: Timer, DMA, ADC, DAC

Лекция 9: Периферийные модули: CAN, USB, Ethernet, SDIO

Структура программы для встраиваемой системы

1. Без использования операционной системы (bare-metal);

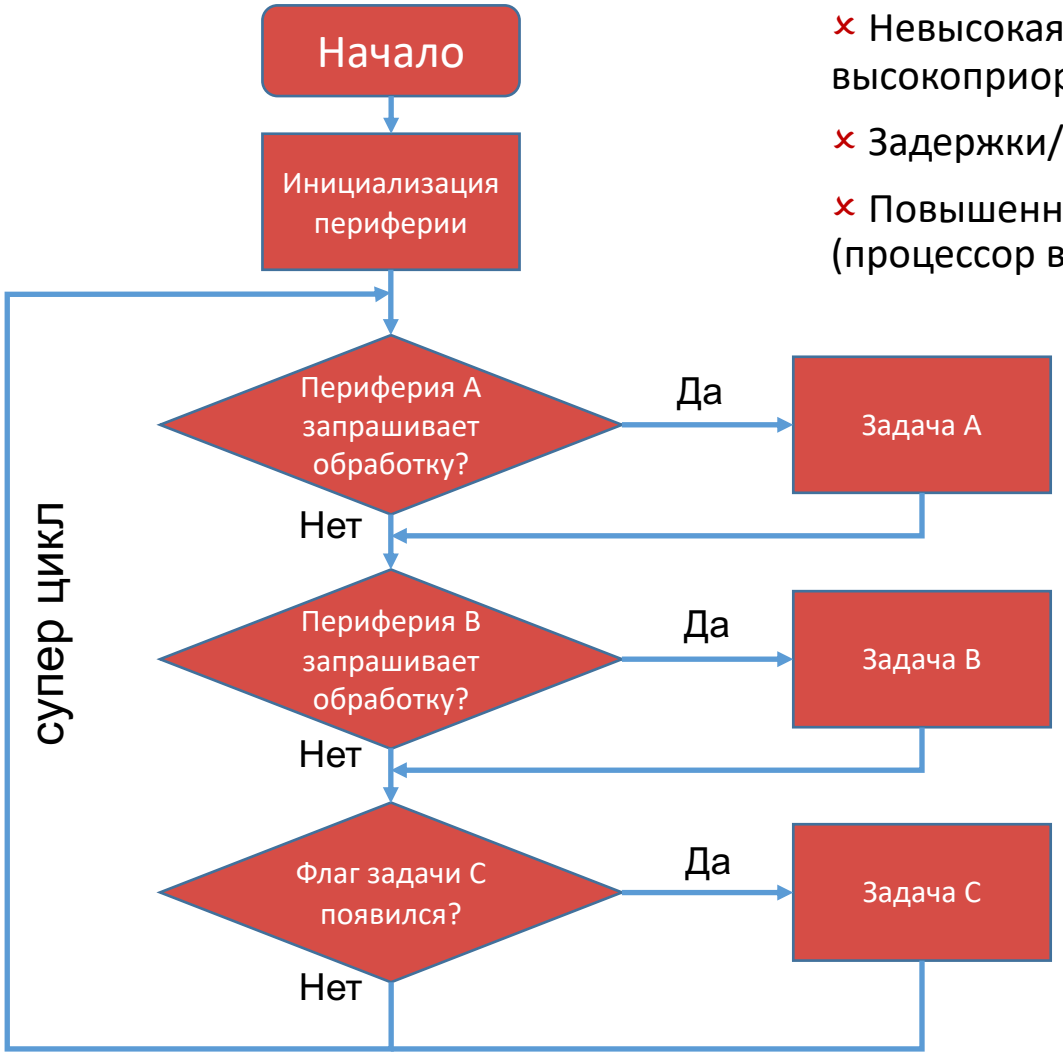
2. Операционная система реального времени (Real Time Operating System-RTOS):

   FreeRTOS, Contiki OS, VxWorks, In-House OS (своя ОС);

3. Встраиваемая операционная система общего назначения:

   Linux, Android, Windows CE.

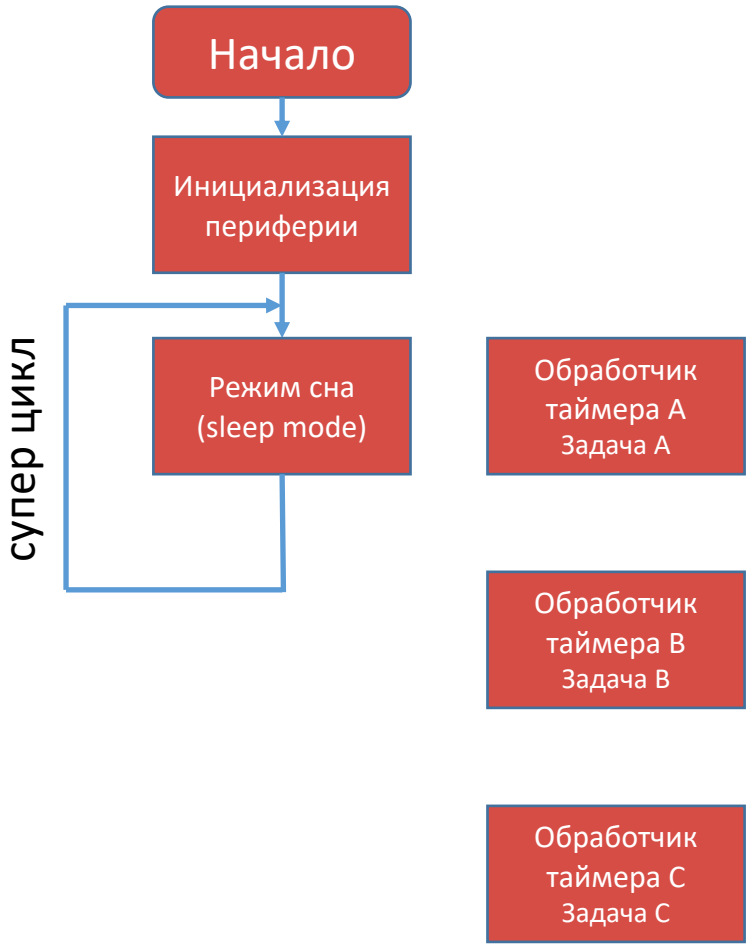
Опрос периферийных модулей и флагов (polling + super loop)



- ✓ Простая архитектура
- × Невысокая скорость реакции на высокоприоритетные события
- × Задержки/джиттер обработки событий
- × Повышенное энергопотребление (процессор всегда в работе)

```
void main()
{
  init_periph();
  while(1)
  {
    regulator();
    protection();
    keyboard();
  }
}
```

Управление по таймеру (timer-based interrupts)

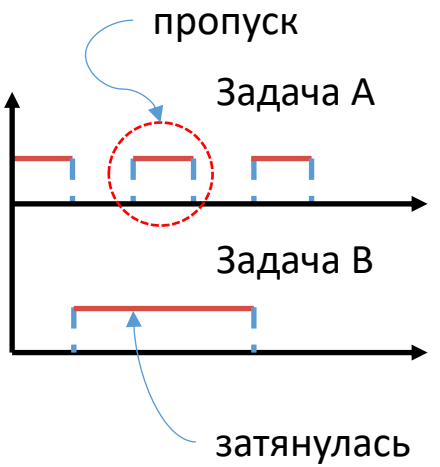


✓ Низкое энергопотребление, возможен сон в супер цикле

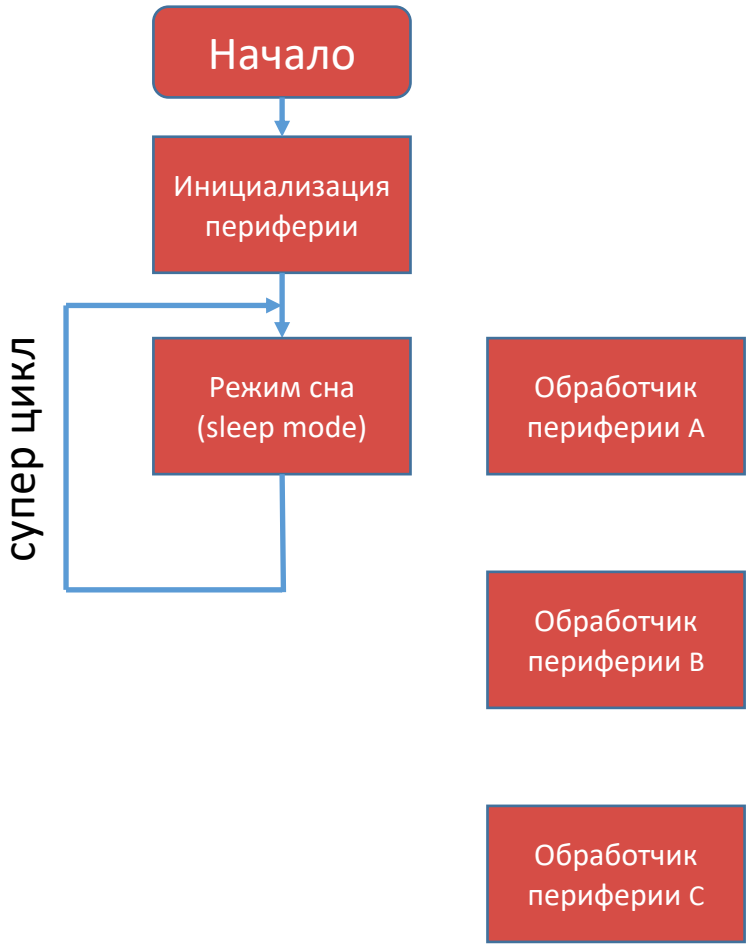
✗ Сложная обработка в прерывании снижает скорость реакции на событие (запрет на вложенные прерывания вынуждает ожидать завершения обработки)

✗ Низкоприоритетные прерывания могут «голодать» при высокой активности высокоприоритетных прерываний

```
void main()
{
  init_periph();
  while(1);
}
void timerA_isr()
{
  regulator();
}
void timerB_isr()
{
  protection();
}
```



Управление по прерываниям (interrupt driven)

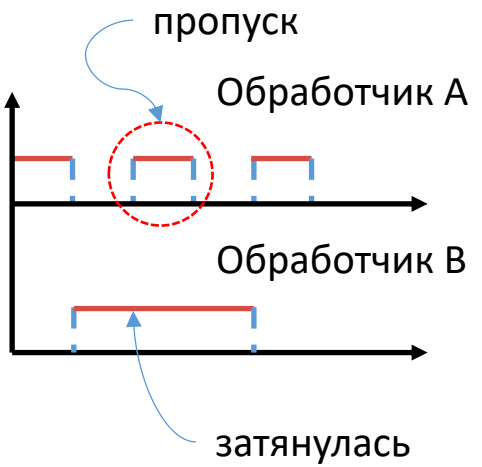


✓ Низкое энергопотребление

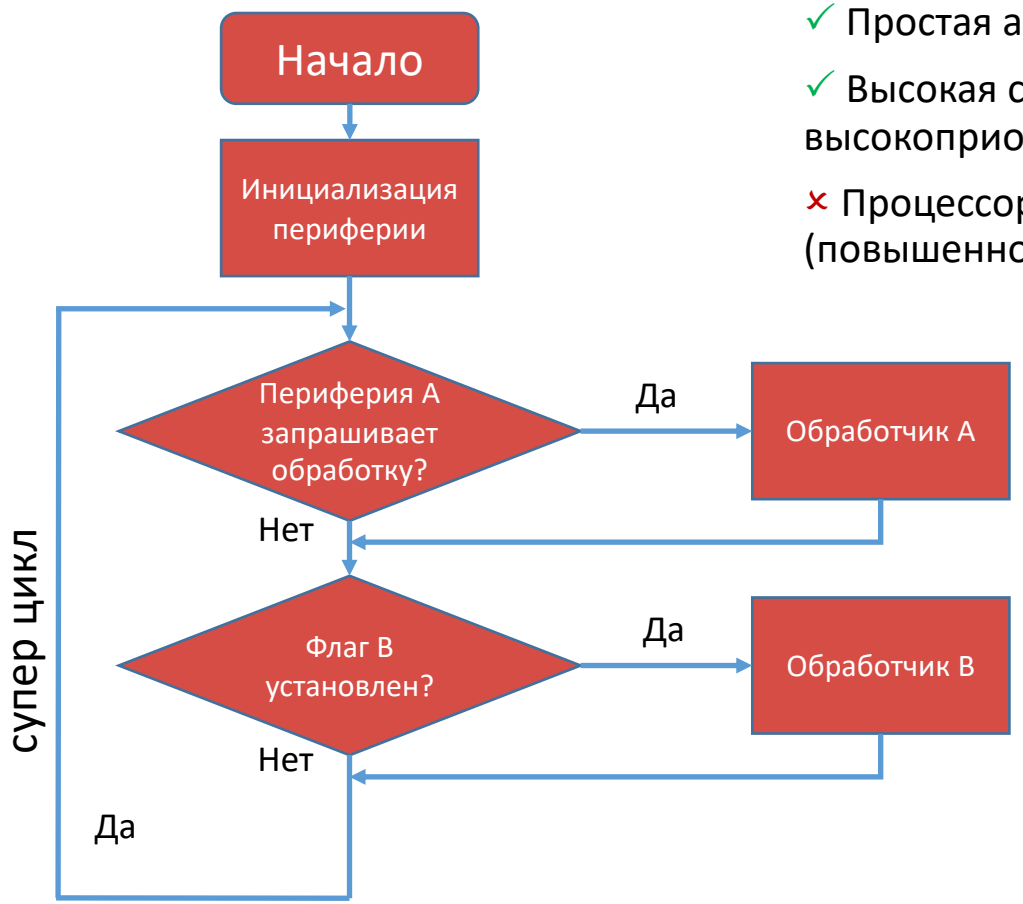
✗ Сложная обработка в прерывании снижает скорость реакции на событие (запрет на вложенные прерывания вынуждает ожидать завершения обработки)

✗ Низкоприоритетные прерывания могут «голодать» при высокой активности высокоприоритетных прерываний

```
void main()
{
  init_periph();
  while(1);
}
void adc_isr()
{
  regulator();
}
void gpio_isr()
{
  protection();
}
```

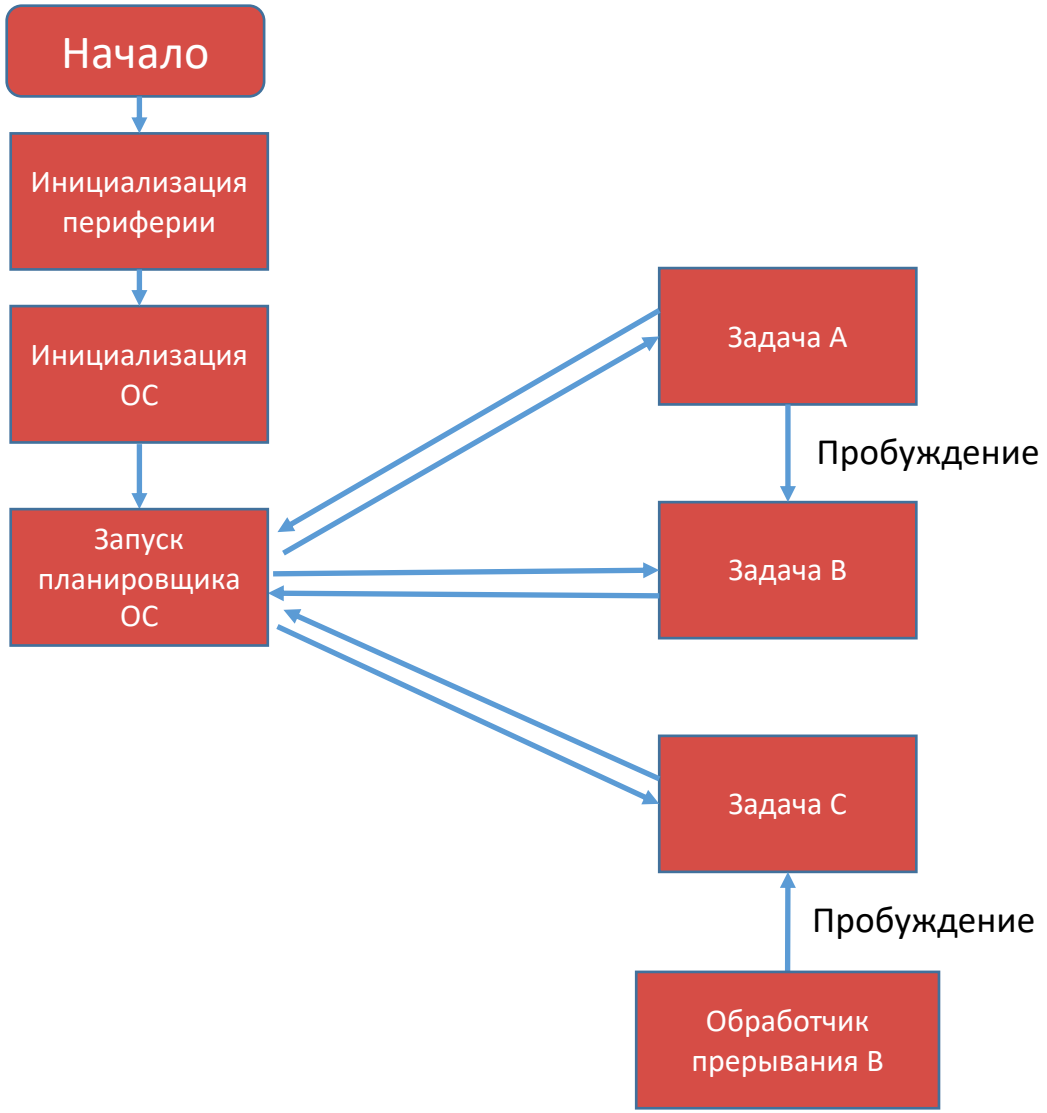


Гибридный способ (polling + interrupt driven)



- ✓ Простая архитектура
- ✓ Высокая скорость реакции на высокоприоритетные события
- ✗ Процессор всегда в работе (повышенное энергопотребление)

Операционная систем (OS)



✓ Возможность построения сложных систем

✗ Требуется изучение интерфейса (API) операционной системы

✗ Операционная система требует ресурсов (память данных, память программ, процессорное время)

```
void main()
{
  init_periph();
  os_start();
}
void regulator_task()
{
}
void protection_task()
{
}
```

Операционные системы

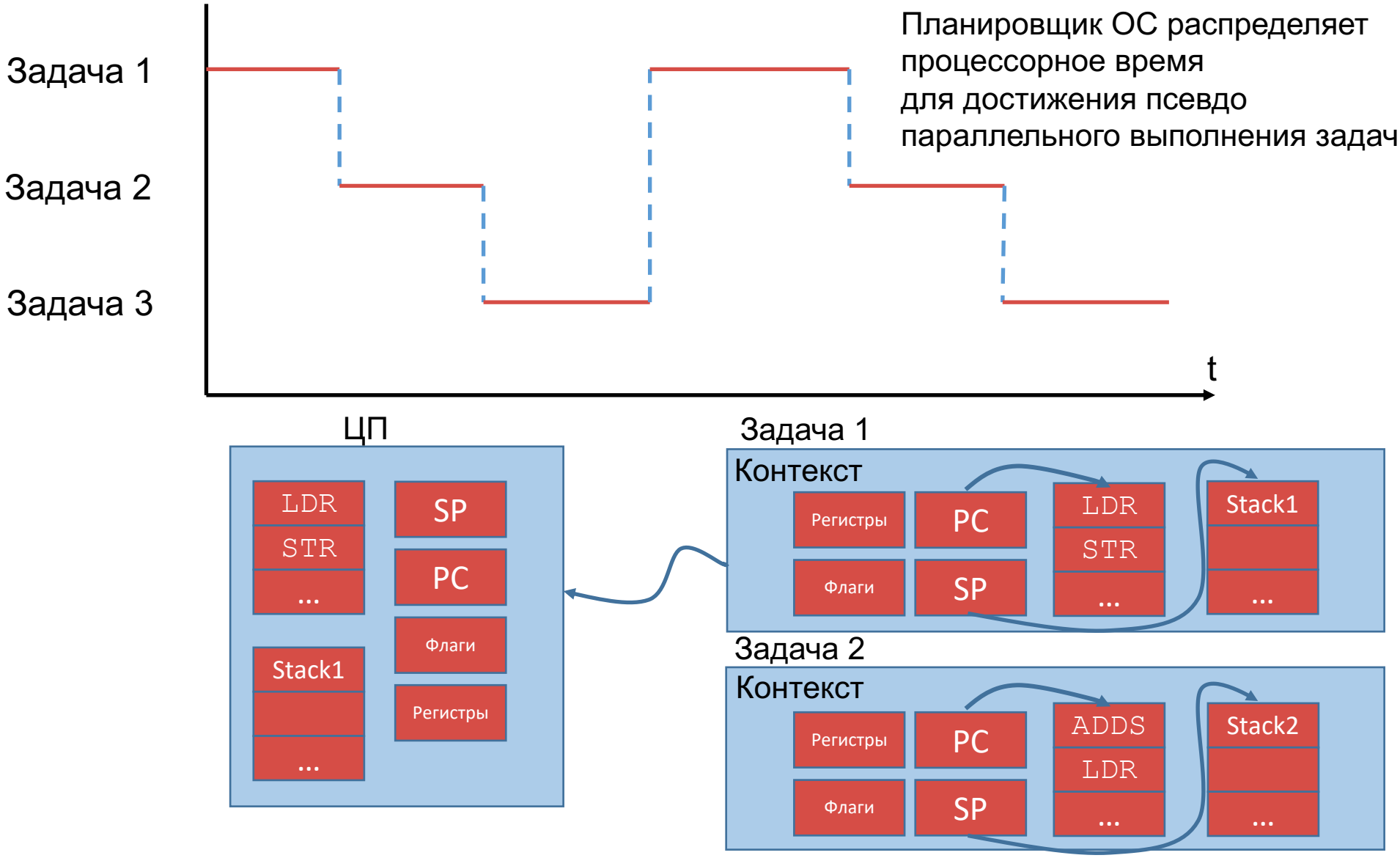
Операционная система общего назначения - комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

Операционная система реального времени (ОСРВ) – система предоставляющая набор функций для организации многозадачности, взаимодействия между задачами и доступа задач к общим ресурсам в системах реального времени и предназначенная для обработки различных событий.

Операционная система «жесткого» реального времени – обеспечивает фиксированное время реакции на событие в любых условиях.

Операционная система «мягкого» реального времени – обеспечивает в среднем постоянное время реакции на события в любых условиях.

Распределение процессорного времени

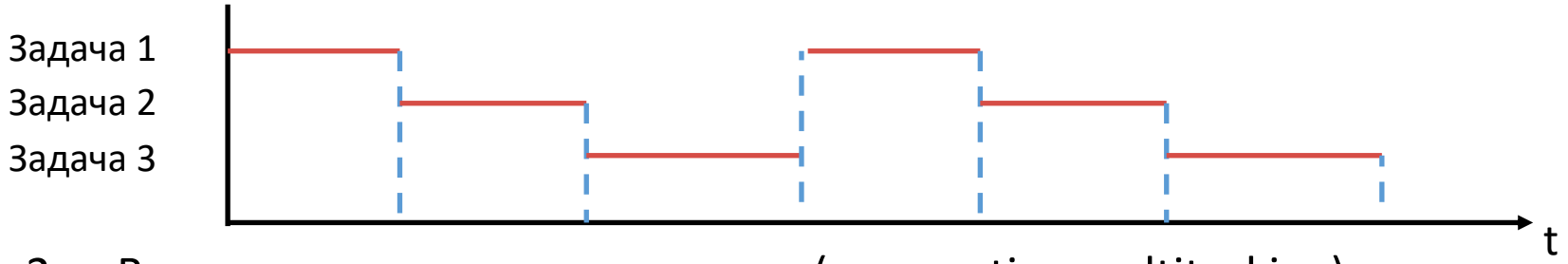


Функции операционных систем

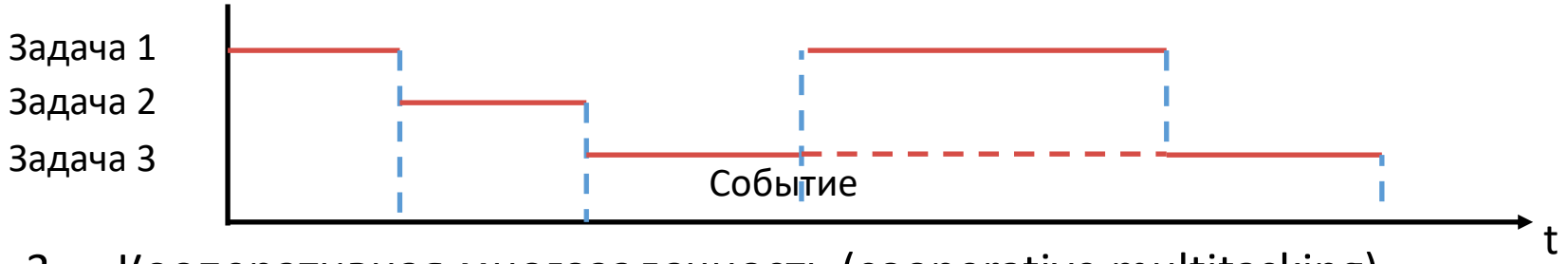
1. **Многозадачность**
   Отлаженный механизм выделения времени (time-slices) задачам и переключения между задачами.
2. **Взаимодействие между задачами**
   Очереди для передачи данных и семафоры для синхронизации.
3. **Временная база**
   Интерфейс для отсчета временных интервалов.
4. **Доступ к ресурсам**
   Мьютексы (блокировки) для доступ к одному ресурсу разными задачами.
5. **Распределение памяти**
   Механизм динамического выделения памяти в куче.

Виды многозадачности

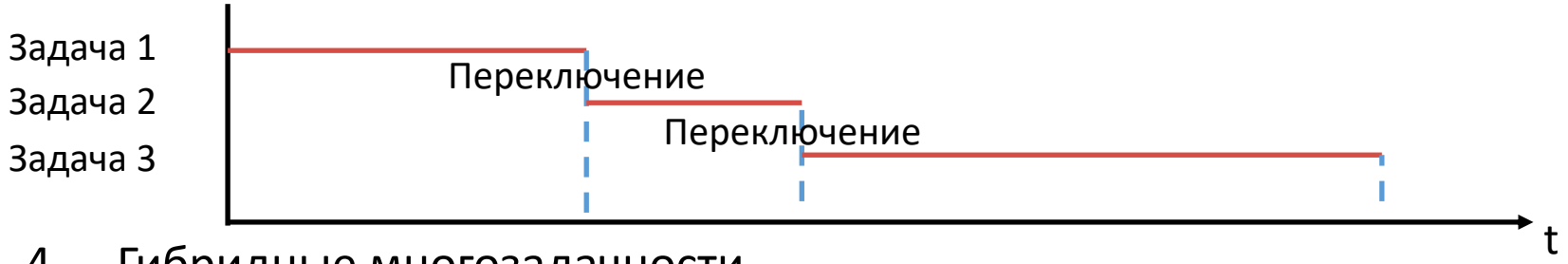
1. «Циклическая» многозадачности (round-robin multitasking)



2. Вытесняющая многозадачность (preemptive multitasking)

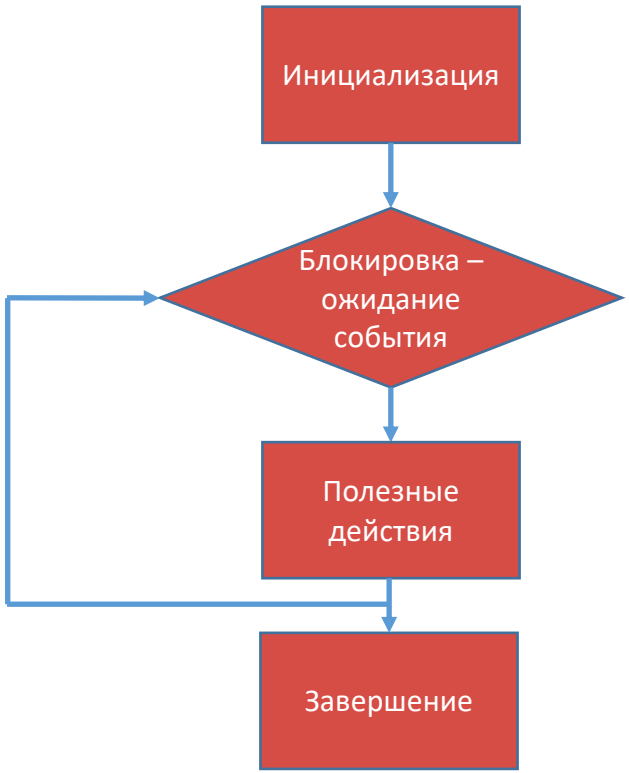


3. Кооперативная многозадачность (cooperative multitasking)



4. Гибридные многозадачности
   Циклическая + вытесняющая + кооперативная

Цикл жизни задачи



Задача (thread, task) – это функция которая работает в цикле `while(1)` и содержит блокирующий вызов, ожидающий какое-либо событие.

Задача может быть в одном из состоянии: работа (**Run**), блокировка (**Blocked**), приостановлена (**Suspend**), уничтожена (**Destroyed**).

Когда задача в состоянии блокировки (например, ожидает события «буфер готов») низкоприоритетные задачи могут исполнятся.

Другая задача или прерывание может сгенерировать событие для разблокировки задачи.

Код с полезными действия исполняется и задача вновь блокируется до появления нового события.

Каждая задача работает в своем «приватном» стеке – стеке задачи.

Задача (task/thread) и функция (подпрограмма)

Функция (подпрограмма) – блок команд который имеет свои внутренние (локальные) переменные и возвращает результат работы.

Задача (thread, task) – функция которая запускается в определенном контексте (приоритет, свой стек, состояние (работа, простой) и т.д.)

```
int func(int a, int b)
{
  return a + b;
}
```

```
void task(...)
{
  /* Пролог */
  ...
  while (1) {
    semaphore_wait();
    /* Обработка */
    ...
  }
  /* Эпилог */
  ...
}
```

Инициализация (запускается один раз)

Цикл

Ожидание ресурсов

Полезная работа

Выход из задачи (запускается один раз)

Доступ к общим ресурсам

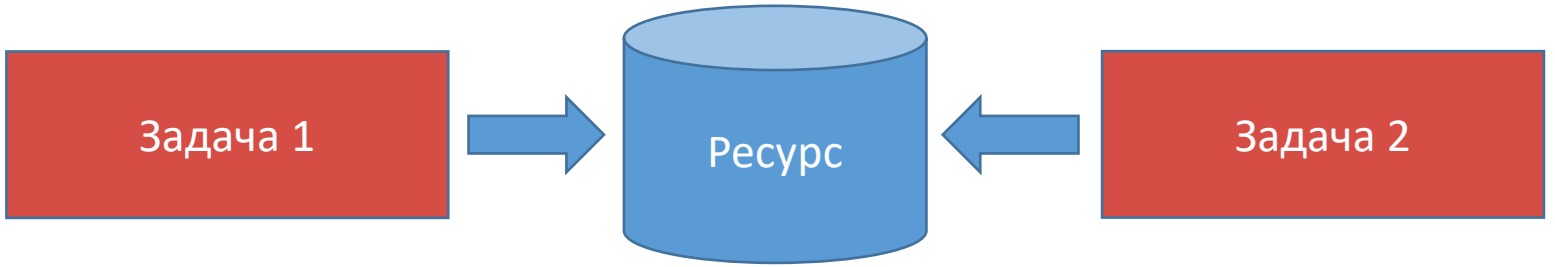
Модель «Производитель – потребитель»

Задача 1 генерирует данные или сообщения и кладет в контейнер (как правило очередь).

Задача 2 блокируется пока не появится данные в контейнере.

Контейнер реализуется средствами операционной системы

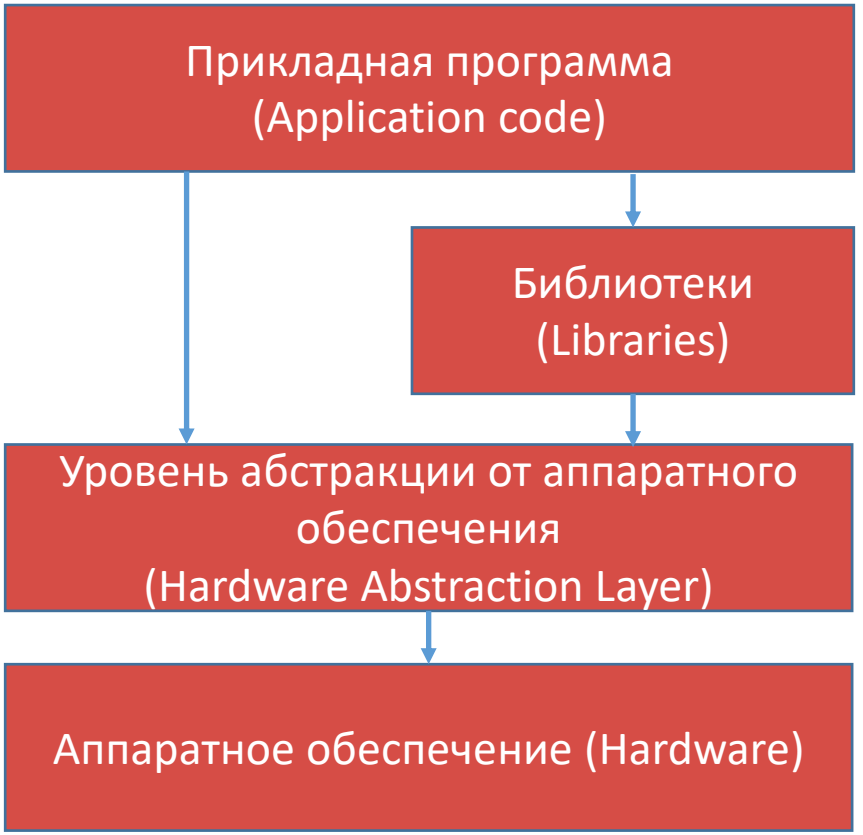
Модель «Конкурентный доступ»



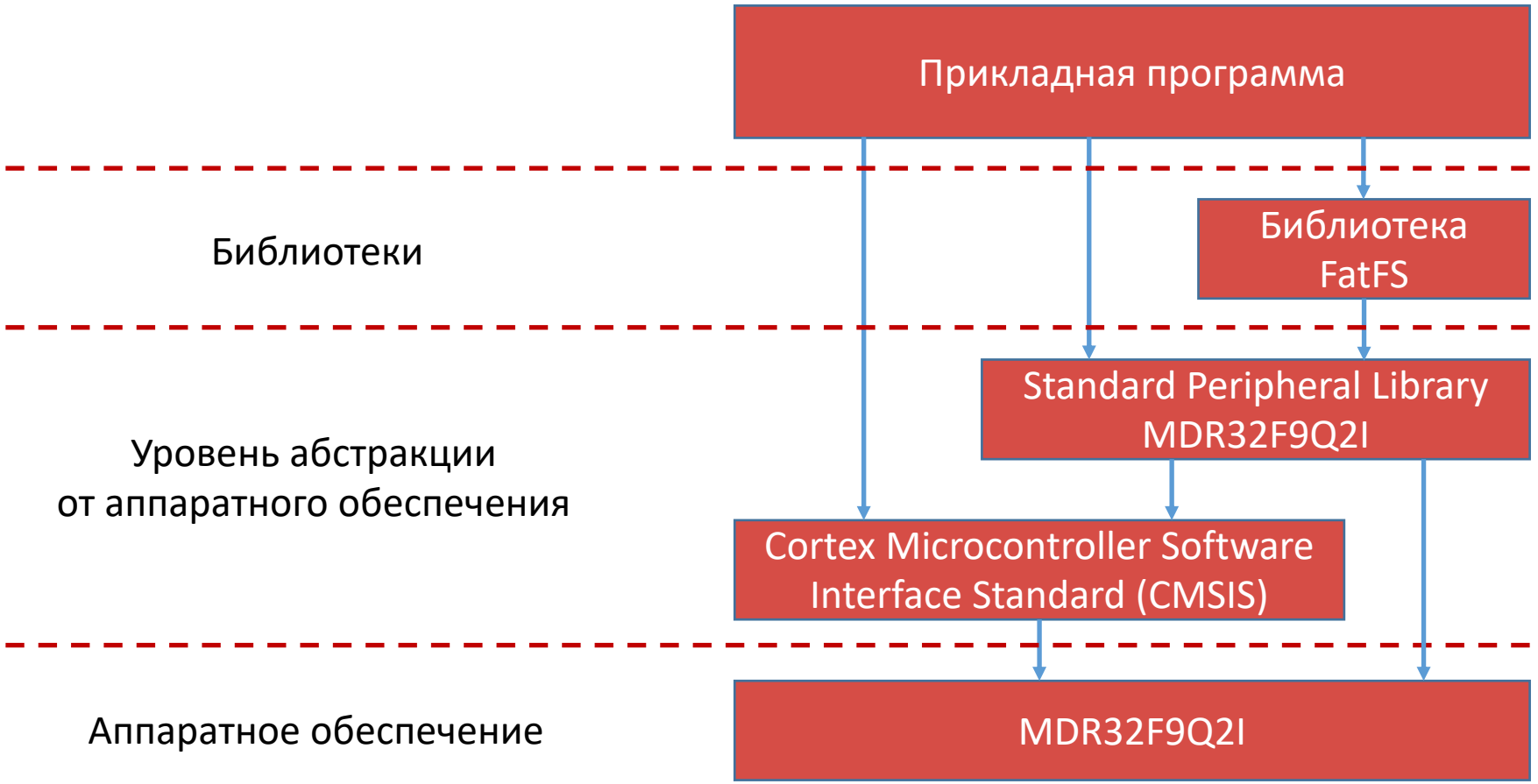
Любая задача в любой момент времени может попытаться получить доступ к ресурсу (общая память (shared memory), интерфейс UART, АЦП и т.д.). Доступность ресурсов определяет мьютекс или семафор. Если он захвачен, то задача может использовать ресурс. Если нет, то задача блокируется до его освобождения. Мьютексы и семафоры реализуются средствами операционной системы.

Вытеснение одной задачи другой может вызвать инверсию приоритетов и «deadlock».

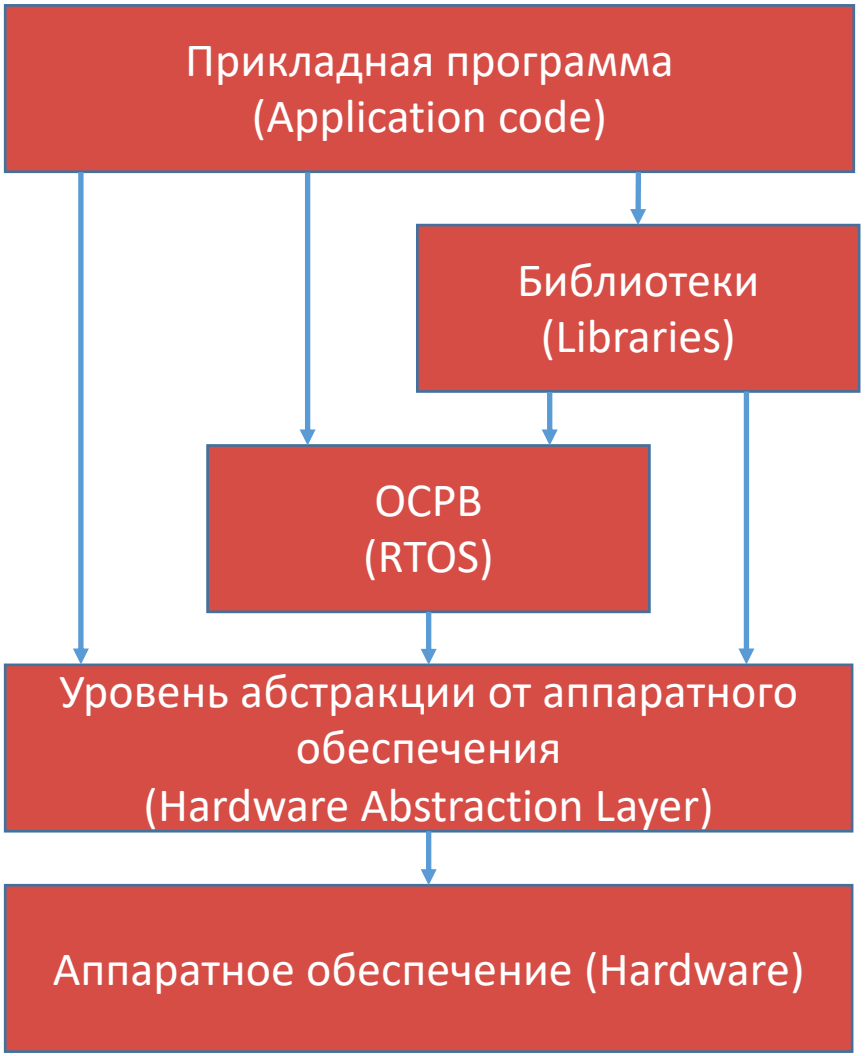
Архитектура программы: без ОС



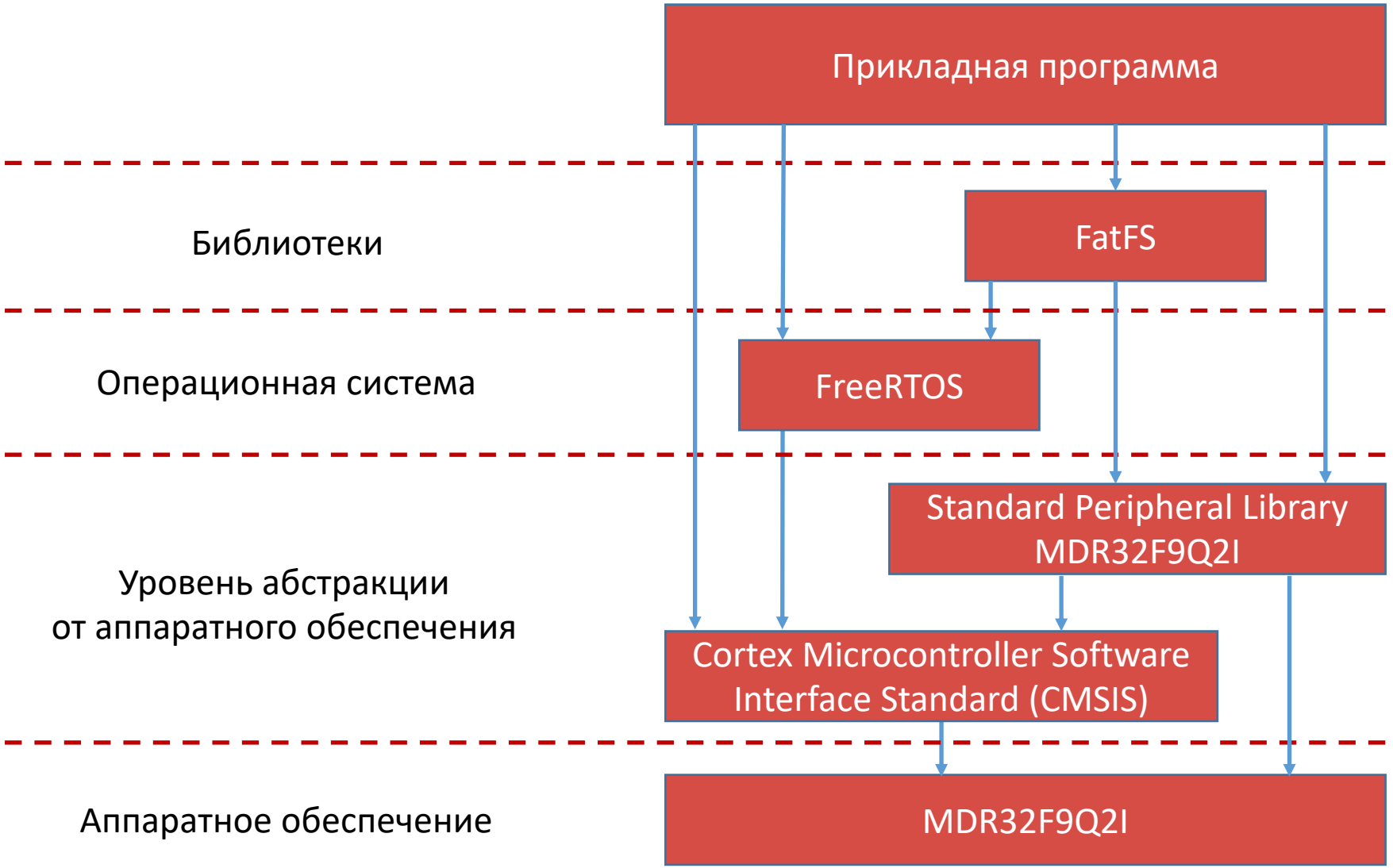
Архитектура программы: пример без ОС



Архитектура программы: ОСРВ



Архитектура программы: пример с ОСРВ



Заключение

1. Выбор способа организации (без ОС, ОСРВ, ОС общего назначения) зависит от конкретных задач. Например, в задачах управления преобразователем следует использовать ОСРВ или не использовать ОС вовсе, а в задачах передачи данных по различных протоколам допустимо применение ОС общего назначения.
2. Важным является абстракция программного обеспечения от аппаратуры. Абстракцию следует закладывать на самых ранних этапах разработки программного обеспечения. Хорошая абстракция позволит проводить тестирование программного обеспечения без аппаратуры, повысит переносимость программы, а также позволит повторно использовать наработки в будущих проектах.