Лекция 7 Архитектура программного обеспечения

План курса «Встраиваемые микропроцессорные системы»:

Лекция 1: Введение. Язык программирования С

Лекция 2: Язык программирования С. Стандартная библиотека языка С

Лекция 3: Применение языка С для встраиваемых систем

Лекция 4: Микроконтроллер

Лекция 5: Этапы разработки микропроцессорных систем

Лекция 6: Разработка и отладка программ для встраиваемых систем

Лекция 7: Архитектура программного обеспечения для встраиваемых систем

Лекция 8: Периферийные модули: DMA, USB, Ethernet

Структура программы для встраиваемой системы

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

- Отсутствуют накладные расходы;
- Полный контроль аппаратного обеспечения;
- Простые системы, выполняющие одну или несколько функций;
- Жесткое соблюдение временных интервалов (например для управления транзисторами).

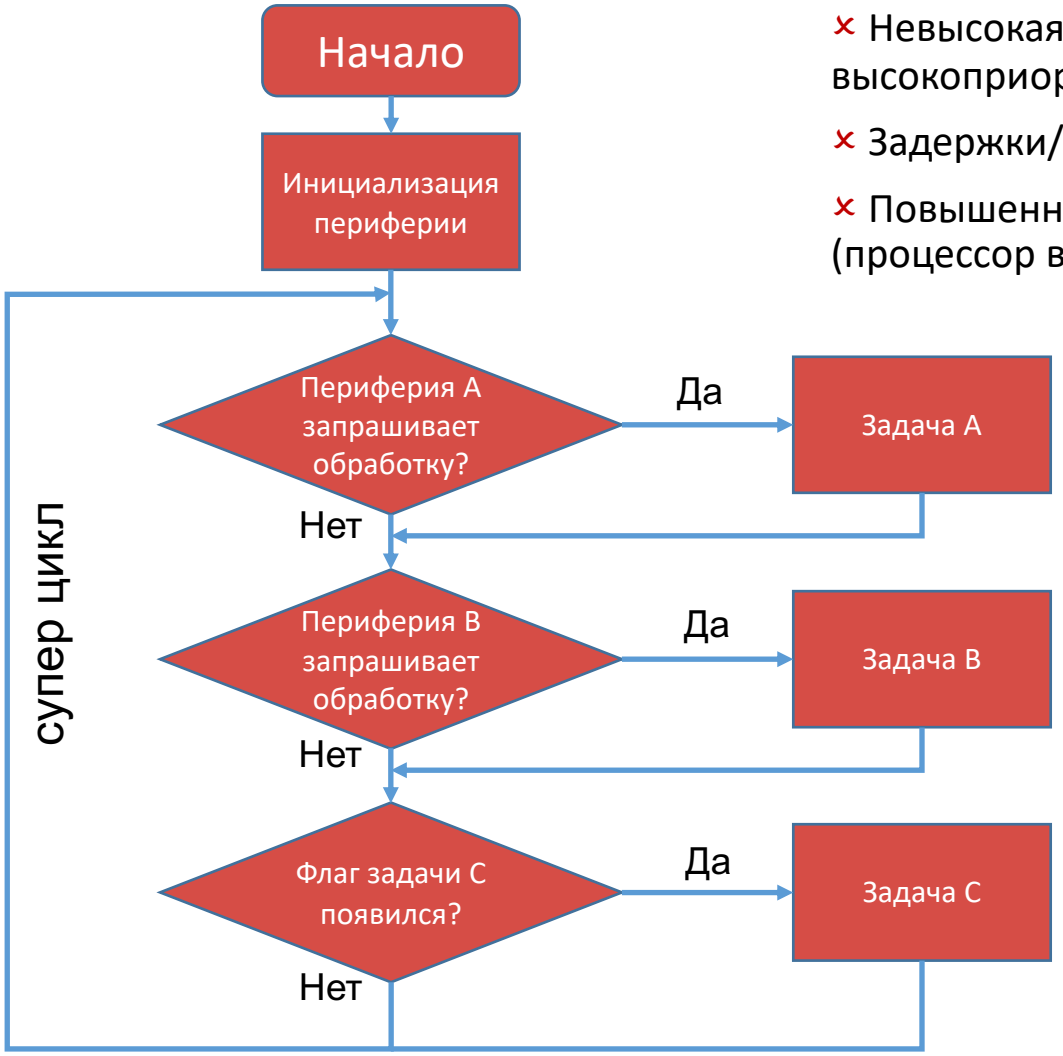
2. Операционная система реального времени (RTOS – Real-Time Operating System):

- Накладные расходы (планировщик), требуется более производительный МК;
- Полный контроль аппаратного обеспечения;
- Многозадачность;
- Различные библиотеки для сетевого взаимодействия, файловой системы и т.д.;
- Примеры: FreeRTOS, Zephyr, Azure RTOS (ThreadX), Mynewt, Contiki, VxWorks, In-House OS (своя ОС). https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

3. Встраиваемая операционная система общего назначения (GPOS – General Purpose Operation System):

- Большие накладные расходы (планировщик, управление память (MMU), фоновые процессы);
- Требуется микропроцессор с внешней ОЗУ и ПЗУ;
- Сложный контроль аппаратного обеспечения (написание драйверов);
- Огромное количество готовых библиотек для выполнения сложных задач (GUI, Networking, CV и т.д.);
- Можно писать на любом языке программирования (C/C++, Python, JS, и т.д.);
- Примеры: Linux, Android, macOS, Windows 11.

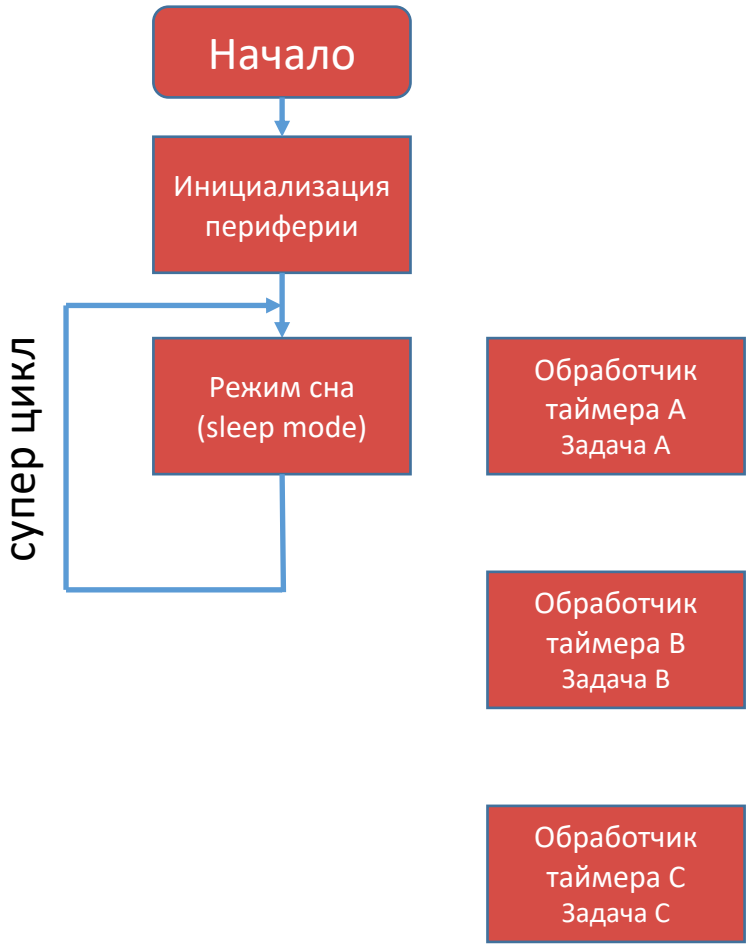
Опрос периферийных модулей и флагов (polling + super loop)



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

```
void main()
{
  init_periph();
  while(1)
  {
    if(event1)
      regulator();
    if(event2)
      protection();
    if(event3)
      keyboard();
  }
}
```

Управление по таймеру (timer-based interrupts)

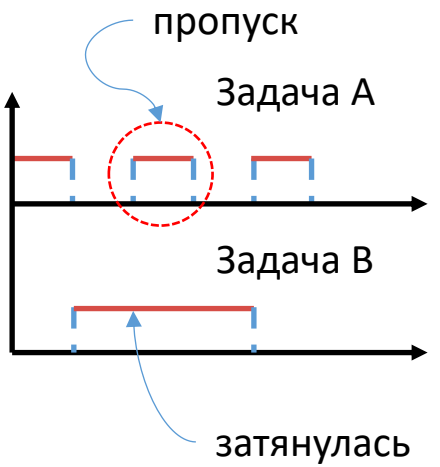


✓ Низкое энергопотребление, возможен сон в супер цикле

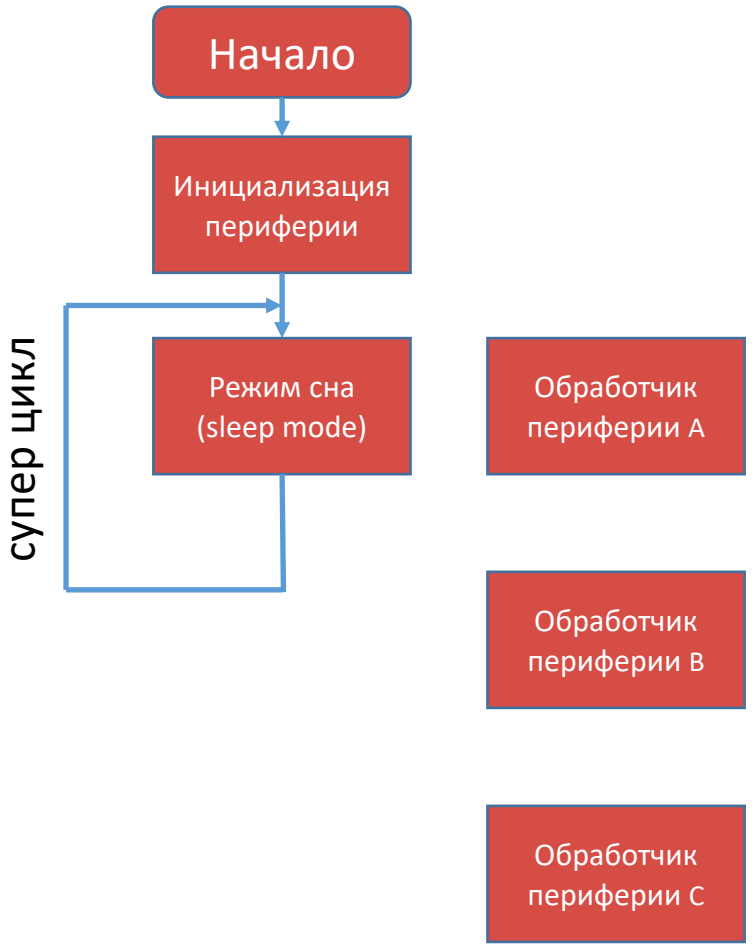
✗ Сложная обработка в прерывании снижает скорость реакции на событие (запрет на вложенные прерывания вынуждает ожидать завершения обработки)

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

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



Управление по прерываниям (interrupt driven)

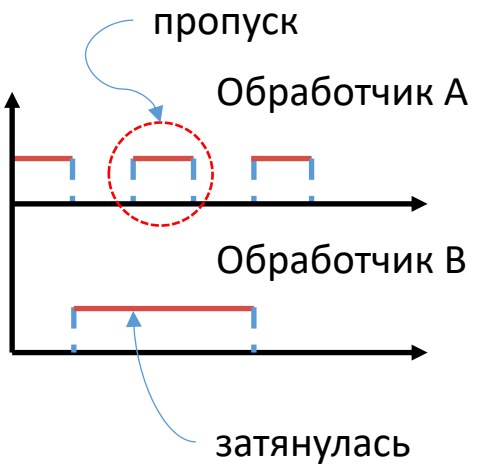


✓ Низкое энергопотребление

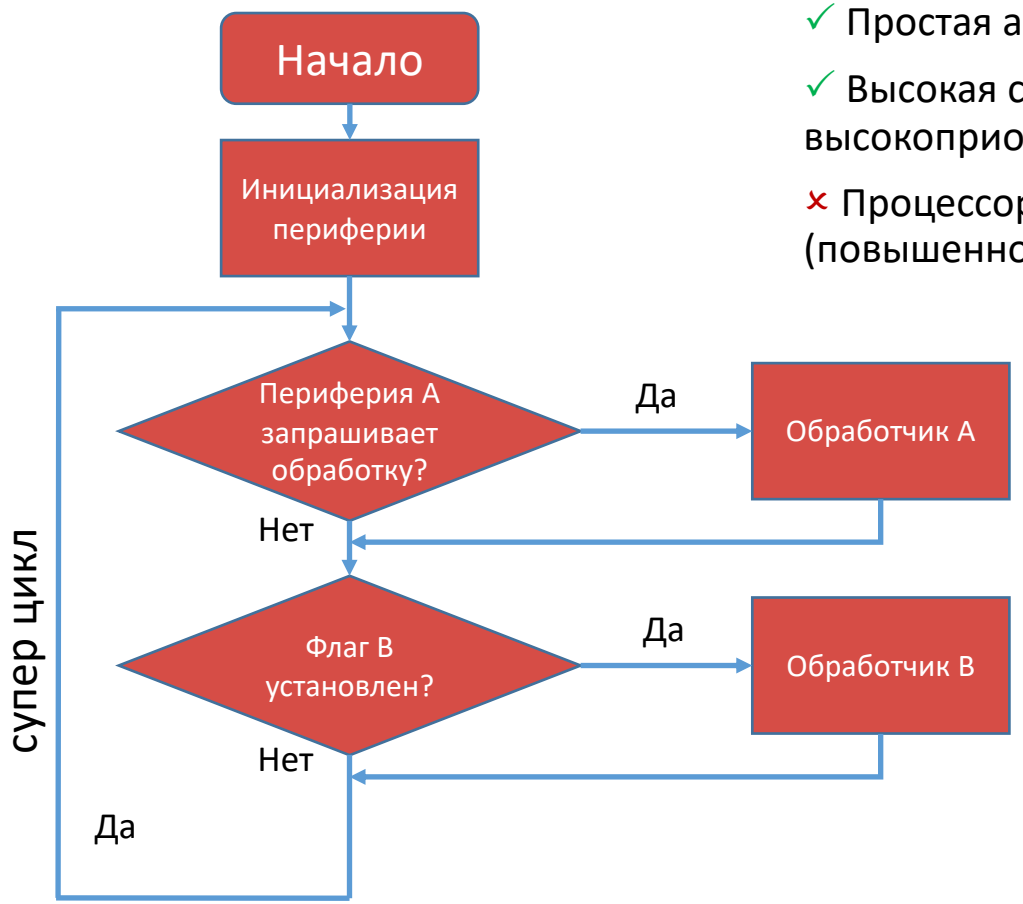
✗ Сложная обработка в прерывании снижает скорость реакции на событие (запрет на вложенные прерывания вынуждает ожидать завершения обработки)

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

```
void main()
{
  init_periph();
  while(1)
    sleep();
}
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)

Операционные системы

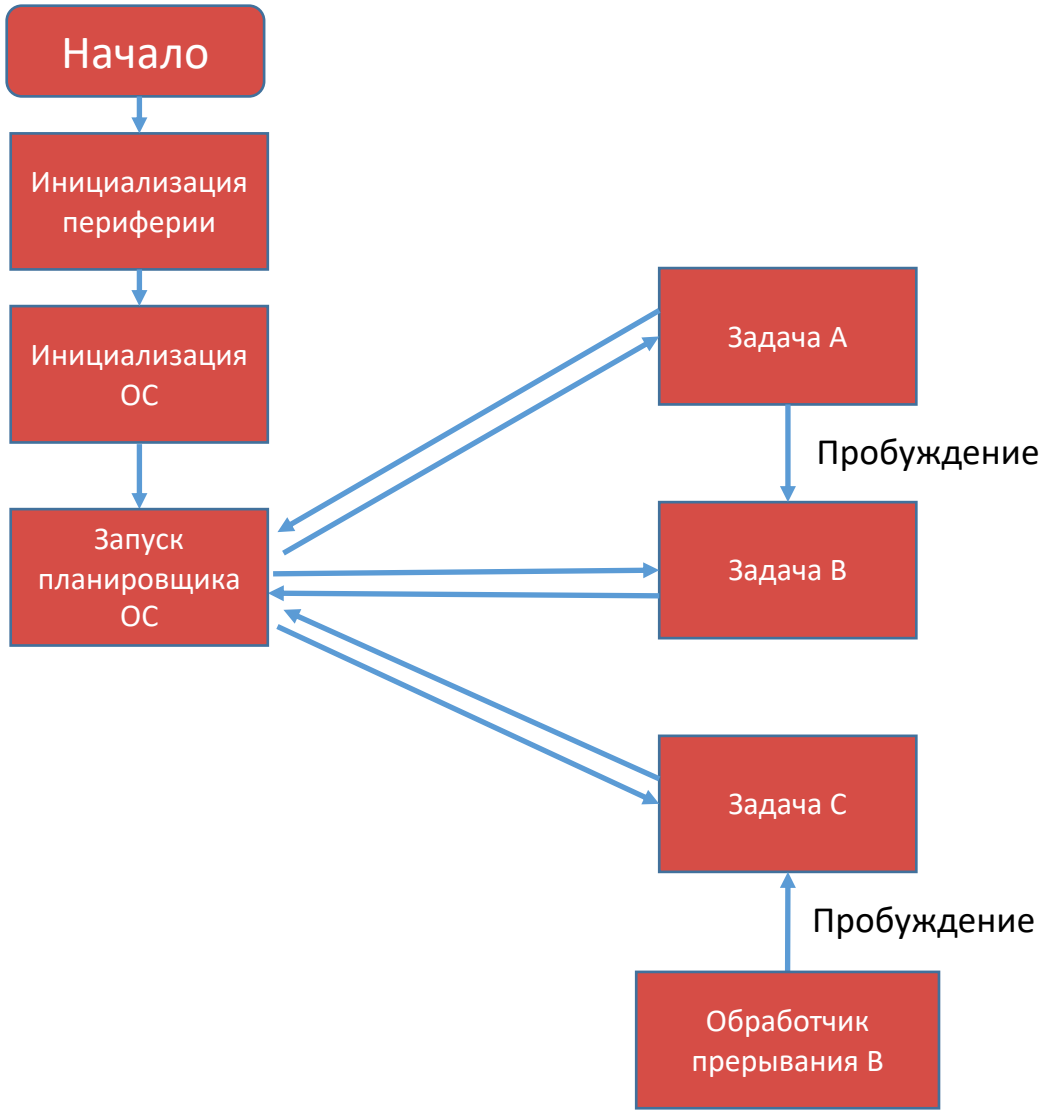
Операционная система общего назначения (ОС, GPOS) - комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.
Примеры: Linux, Android, macOS, Windows 11

Операционная система реального времени (ОСРВ, RTOS) – 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.
Примеры: FreeRTOS, Zephyr, Mynewt, VxWorks, Real Time Linux, Windows IoT

Операционная система «жесткого» реального времени (Hard Real Time) – обеспечивает фиксированное время реакции на событие в любых условиях.
Примеры: FreeRTOS, Zephyr, Azure RTOS (ThreadX), Mynewt, Contiki, VxWorks

Операционная система «мягкого» реального времени (Soft Real Time) – обеспечивает в среднем постоянное время реакции на события в любых условиях.
Примеры: Real Time Linux, Windows IoT

Операционная систем (OS)



✓ Возможность построения сложных систем

✗ Требуется изучение интерфейса (API) операционной системы

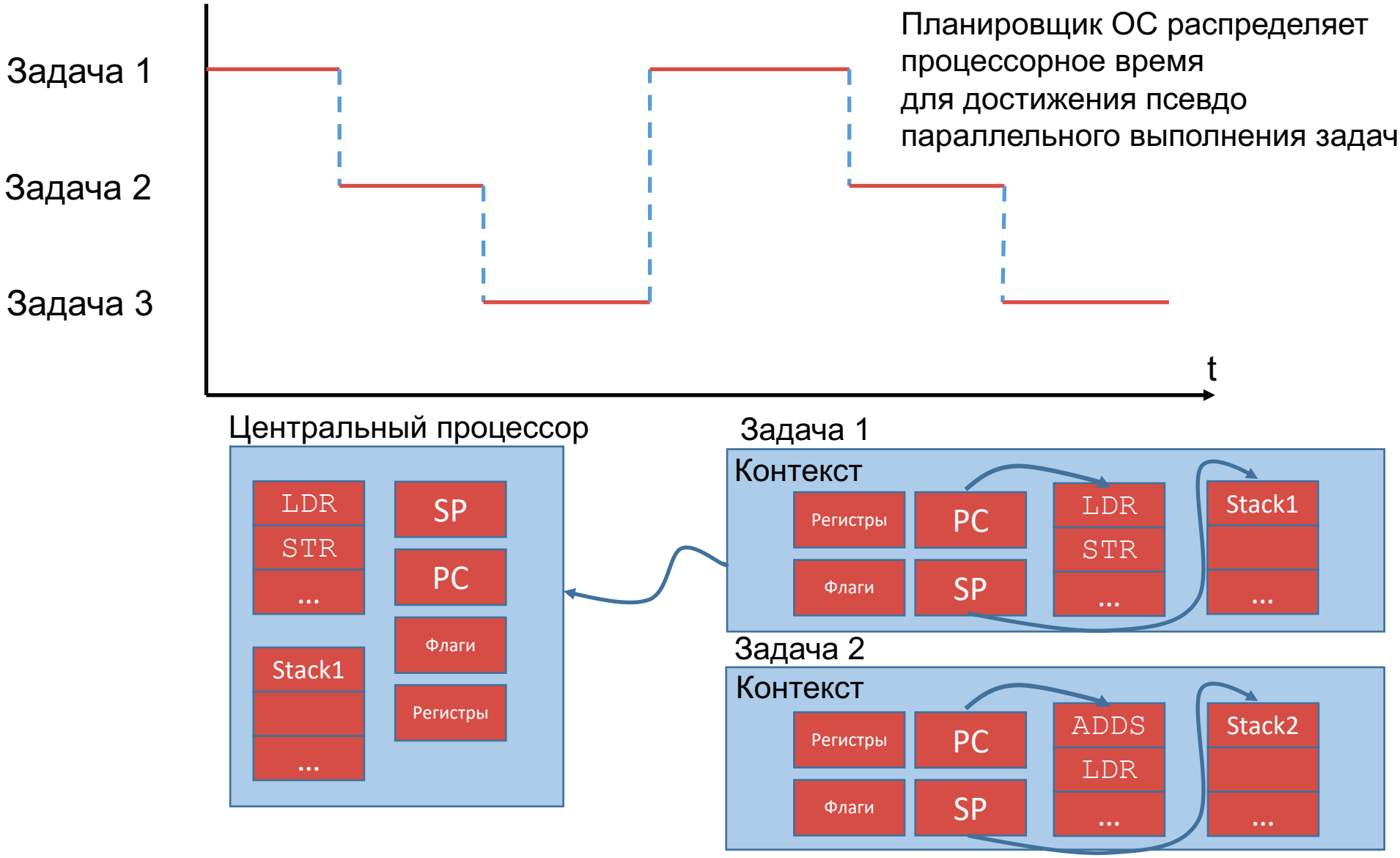
✗ Операционная система требует ресурсов (память данных, память программ, процессорное время)

```
void main()
{
  init_periph();
  os_start();
}
void regulator_task()
{
  while(1)
  {
    func1();
    func2();
  }
}
void protection_task()
{
  while(1)
    semph_wait(protection);
}
void adc_isr()
{
  semph_give(protection);
}
```

Функции операционных систем

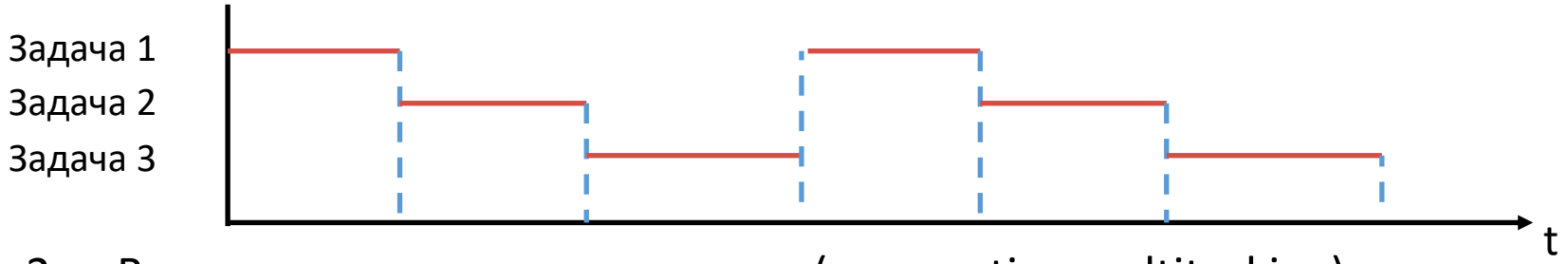
1. **Многозадачность**
   Отлаженный механизм выделения процессорного времени (time-slices) задачам и переключения между задачами.
2. **Межпроцессное взаимодействие (IPC - Inter-Process Communication)**
   Очереди для передачи данных, семафоры для синхронизации и так далее.
3. **Временная база**
   Интерфейс 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.

Распределение процессорного времени



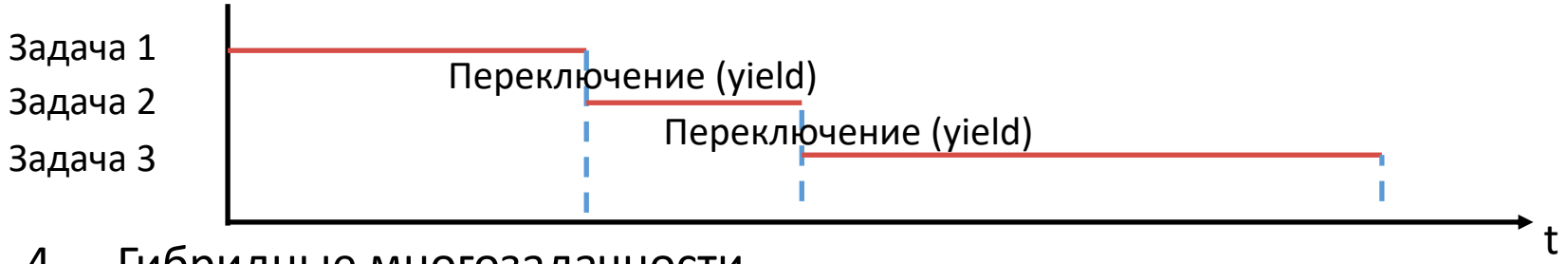
Виды многозадачности

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



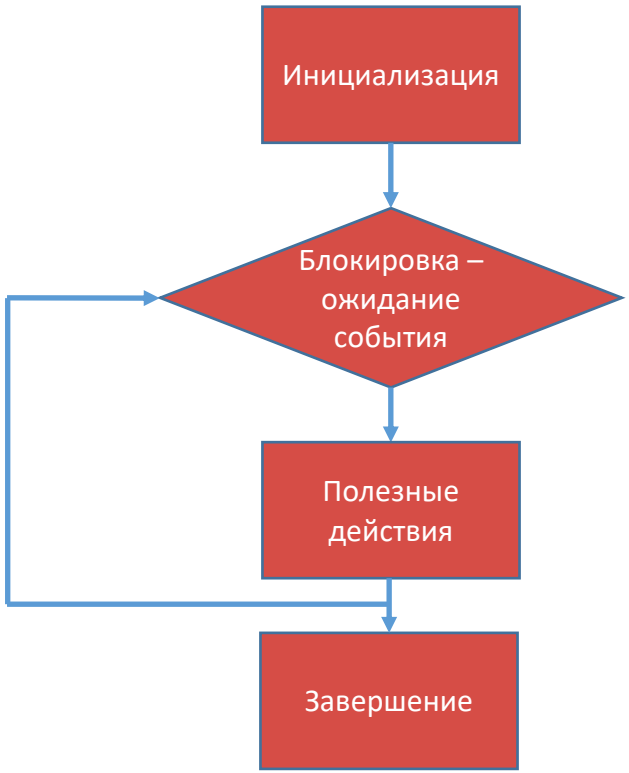
2. Вытесняющая многозадачность (preemptive multitasking)

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/поток (task/thread) и функция (подпрограмма)

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

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

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

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

Инициализация (запускается один раз)

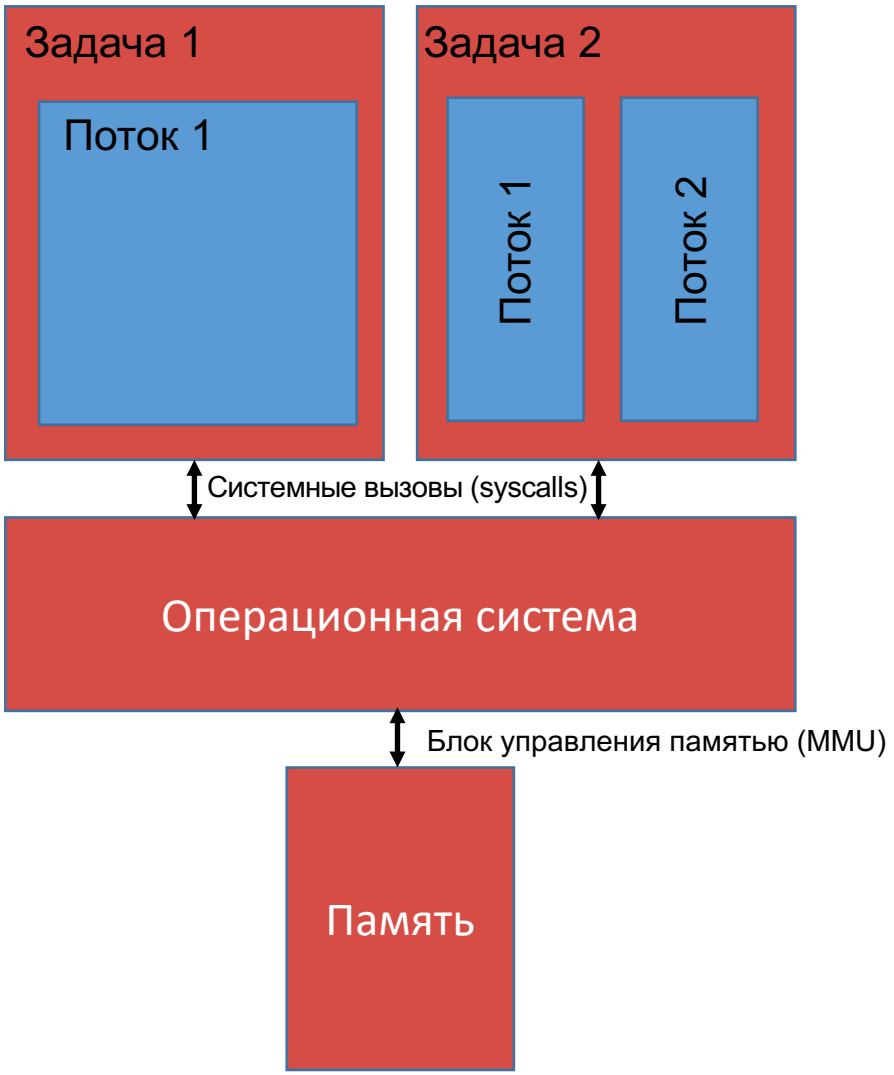
Цикл

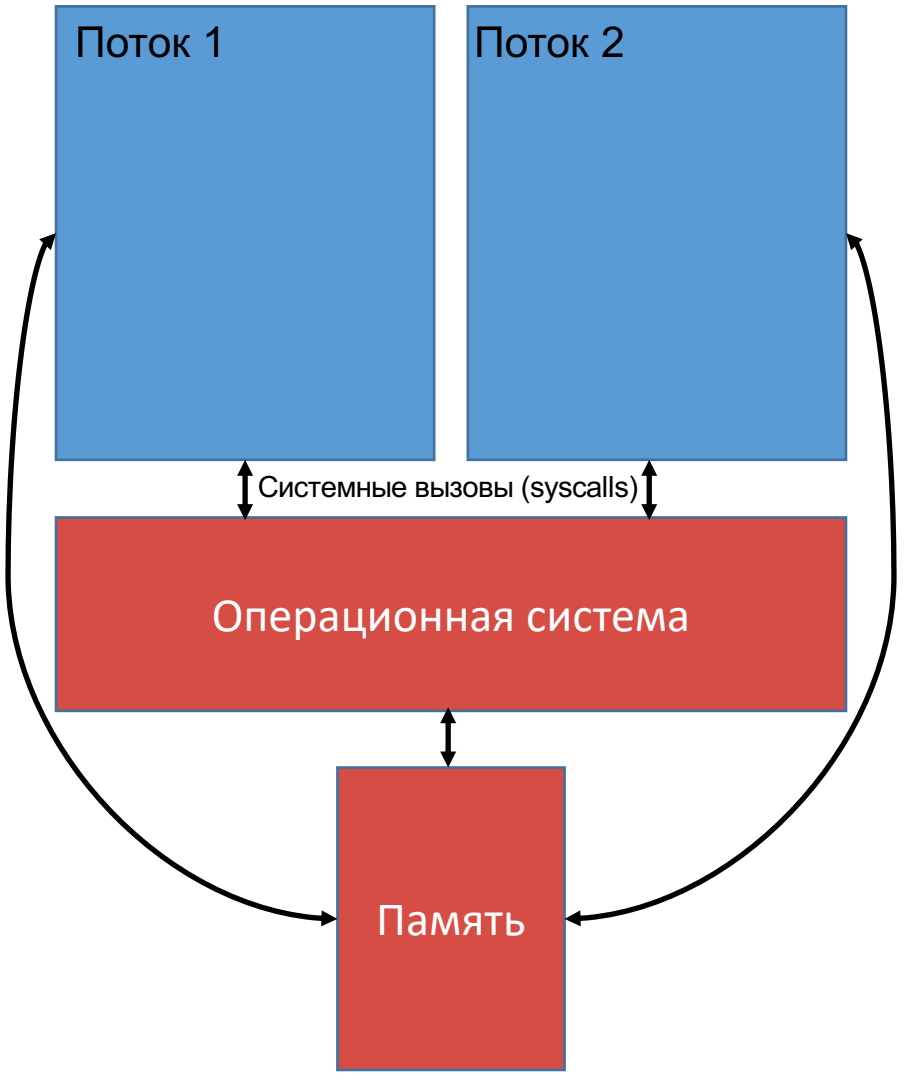
Ожидание ресурсов

Полезная работа

Выход из задачи (запускается один раз)

Задача (task) и поток (thread)





Доступ к общим ресурсам

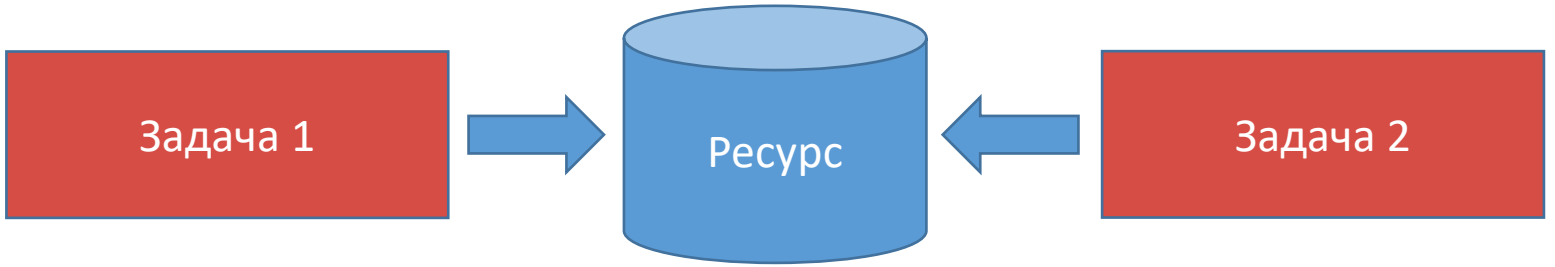
Модель «Производитель – потребитель»

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 программного обеспечения

Общие принципы декомпозиции программного обеспечения:

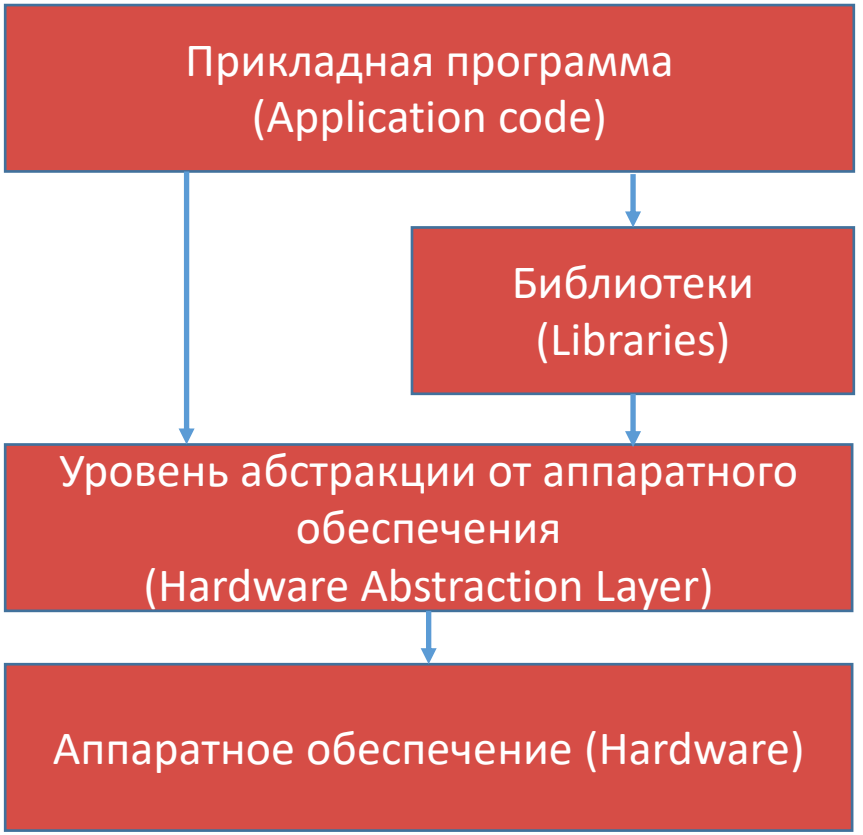
- при проектировании подпрограмм (функций) следует сохранять подпрограмму достаточно короткой и выполняющую только одну задачу. Если участки кода повторяются несколько раз, то их следует выносить в отдельную функцию.
- при проектировании модулей (файлов с исходными кодами) следует ограничивать длину модуля (файла). Если подпрограммы используются в других модулях, то для этих подпрограмм следует создать свой модуль.
- модули повторяющиеся между проектами следует выносить в библиотеки.

Особенностью встраиваемого программного обеспечения является сильная привязка программы к аппаратному обеспечению, а именно к периферийным устройствам и обвязке.

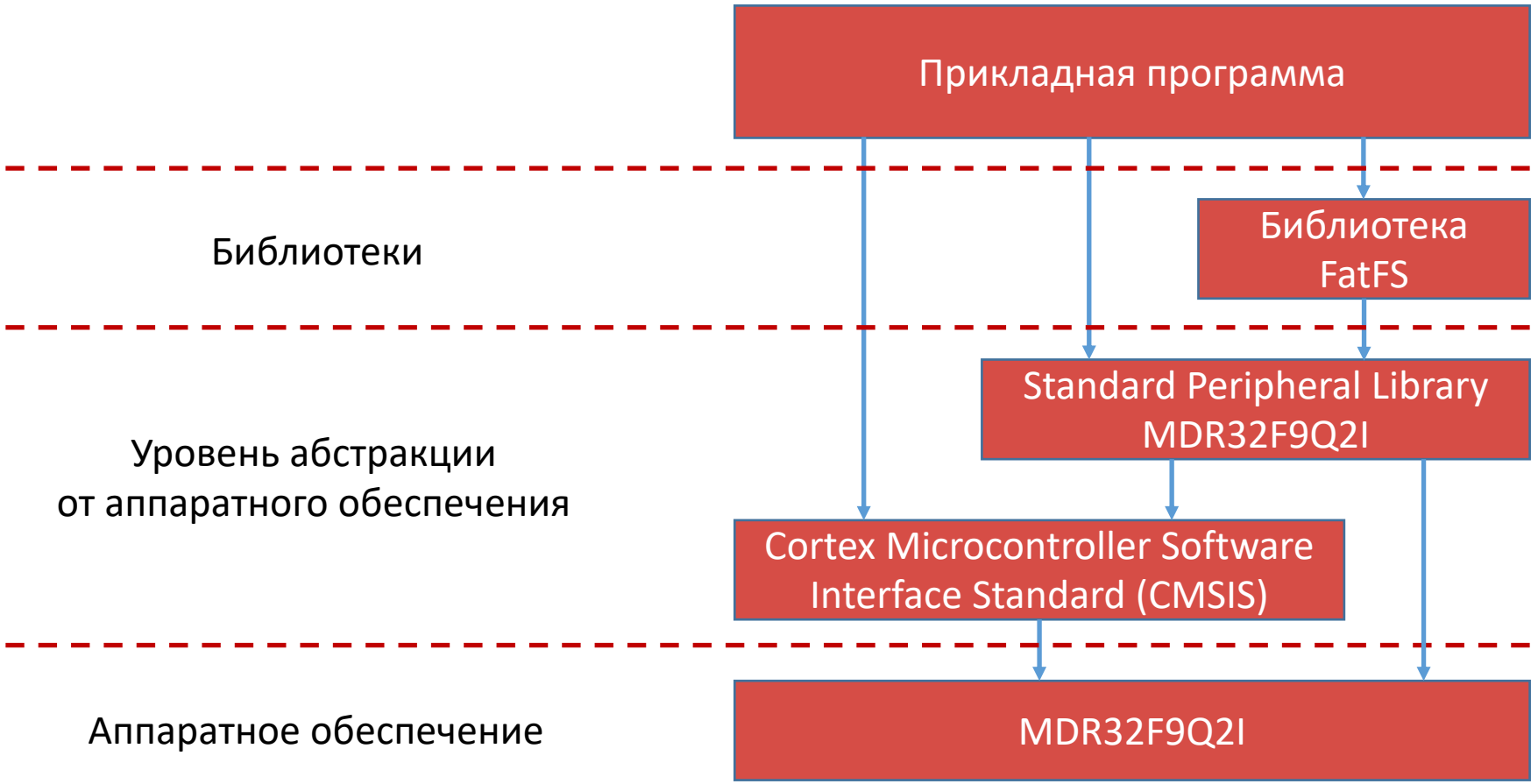
Аппаратное обеспечение следует абстрагировать с помощью слоя HAL - Hardware Abstraction Layer.

Библиотеки также следует абстрагировать с помощью оберток (wrappers) и модулей портирования (ports).

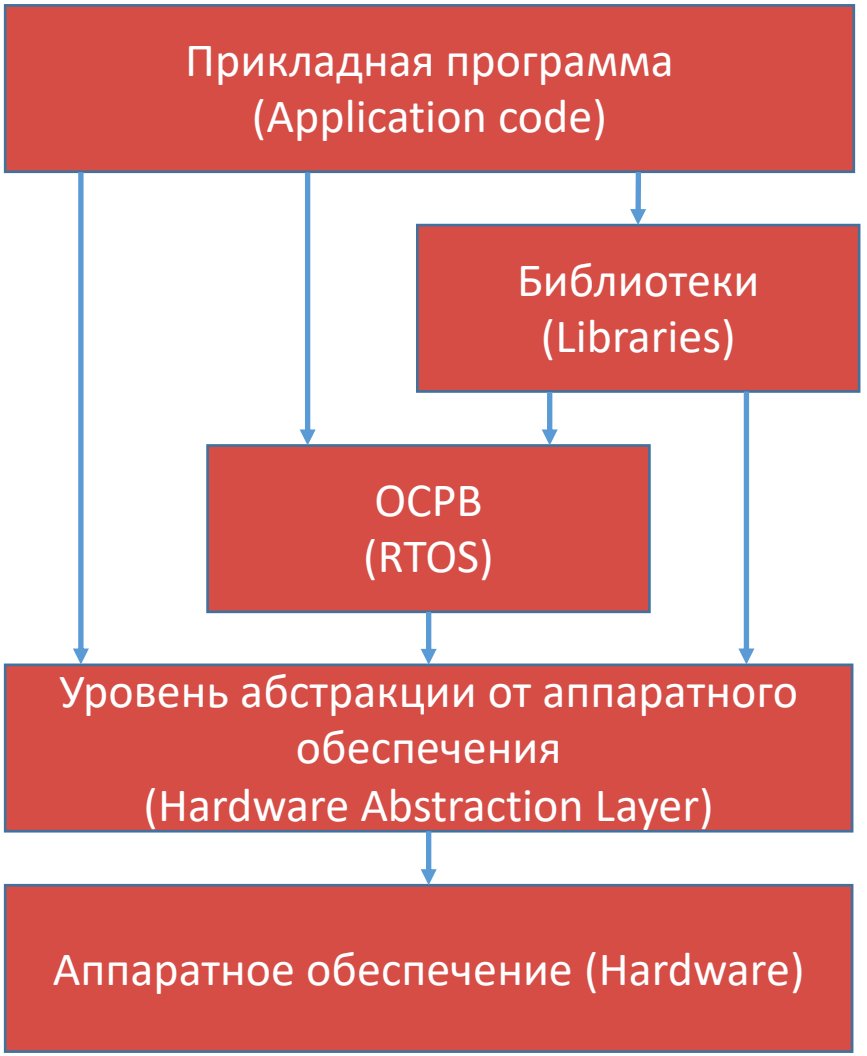
Архитектура программы: без ОС



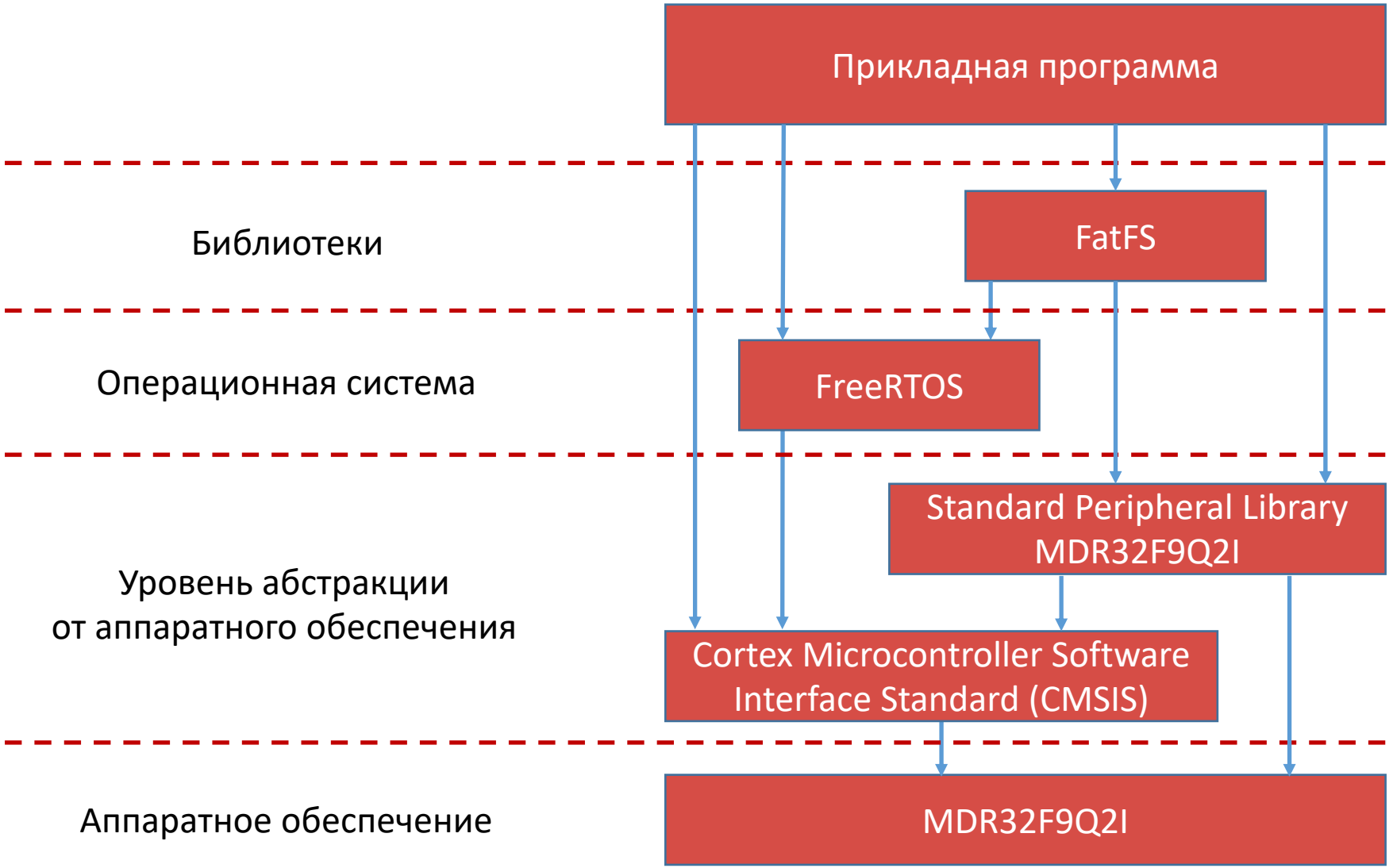
Архитектура программы: пример без ОС



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.