ITMO

Встроенные системы

Отчёт по лабораторной работе №3

Реализовать игру “Тетрис”

Группа P3331

Вариант №2

Выполнили:

Чураков Александр Алексеевич

Дата сдачи отчета: 09.12.2025

Содержание

1 Введение **2**

2 Архитектура системы **2**

3 Используемые примитивы FreeRTOS **2**
3.1 Задачи (Tasks) . . . 2
3.1.1 Создание задач . . . 2
3.1.2 Реализация задачи KeyboardTask . . . 3
3.1.3 Реализация задачи GameTask . . . 4
3.1.4 Реализация задачи DisplayTask . . . 5
3.2 Очереди (Queues) . . . 6
3.2.1 Создание очереди . . . 6
3.2.2 Отправка данных в очередь . . . 7
3.2.3 Получение данных из очереди . . . 7
3.3 Мьютексы (Mutexes) . . . 7
3.3.1 Создание мьютексов . . . 8
3.3.2 Использование мьютекса game_mutex . . . 8
3.3.3 Использование мьютекса display_mutex . . . 8

4 Логика синхронизации **8**
4.1 Принципы синхронизации . . . 8
4.2 Сценарии синхронизации . . . 9
4.2.1 Сценарий 1: Обработка нажатия клавиши . . . 9
4.2.2 Сценарий 2: Обновление игровой логики . . . 9
4.2.3 Сценарий 3: Отрисовка на экране . . . 10
4.3 Предотвращение взаимных блокировок (Deadlock) . . . 10
4.4 Приоритеты задач и их влияние на синхронизацию . . . 11
4.5 Таймауты и их роль . . . 11

5 Временные характеристики **12**
5.1 Частоты обновления . . . 12
5.2 Время реакции . . . 12
5.3 Автоматическое падение фигур . . . 12

6 Прерывания **12**
6.1 SysTick прерывание . . . 12
6.2 Отсутствие других прерываний . . . 13

7 Диаграмма последовательностей **13**

8 Резюме использования FreeRTOS примитивов **15**

9 Выводы **15**

1 Введение

В данной лабораторной работе реализована игра “Тетрис” на микроконтроллере STM32F427VITx с использованием операционной системы реального времени FreeRTOS. Игра отображается на OLED дисплее и управляется через матричную клавиатуру 4x3.

Основная цель работы — демонстрация использования примитивов синхронизации FreeRTOS для организации многозадачной системы с разделением функций между параллельными задачами.

2 Архитектура системы

Система построена на основе FreeRTOS и использует архитектуру с тремя основными задачами:

- **KeyboardTask** — опрос клавиатуры и отправка команд
- **GameTask** — обработка игровой логики
- **DisplayTask** — отрисовка на OLED дисплее

Задачи взаимодействуют через примитивы синхронизации FreeRTOS: очереди и мьютексы.

3 Используемые примитивы FreeRTOS

3.1 Задачи (Tasks)

В проекте создано **3 задачи** с различными приоритетами:

Задача	Приоритет	Период	Назначение
KeyboardTask	2	20 мс	Опрос клавиатуры, отправка кодов клавиш в очередь
GameTask	3 (высший)	10 мс	Обработка команд клавиатуры, обновление игровой логики
DisplayTask	1 (низший)	50 мс	Отрисовка игрового состояния на OLED дисплее

3.1.1 Создание задач

Задачи создаются в функции `main()` перед запуском планировщика:

```
//                                FreeRTOS
xTaskCreate(vKeyboardTask, "Keyboard",
            configMINIMAL_STACK_SIZE * 2, NULL, 2, NULL);
xTaskCreate(vGameTask, "Game",
            configMINIMAL_STACK_SIZE * 4, NULL, 3, NULL);
xTaskCreate(vDisplayTask, "Display",
            configMINIMAL_STACK_SIZE * 4, NULL, 1, NULL);

//                                          FreeRTOS
vTaskStartScheduler();
```

Листинг 1: Создание задач FreeRTOS

Параметры функции `xTaskCreate()`:

- `vKeyboardTask` — указатель на функцию задачи
- `"Keyboard"` — имя задачи (для отладки)
- `configMINIMAL`$_S TACK_S IZE * 2 - - -$
- `NULL -- параметры задачи`
- `2 -- приоритет задачи (чем выше число, тем выше приоритет)`
- `NULL -- handle задачи (не сохраняется)`

3.1.2 Реализация задачи KeyboardTask

```
void vKeyboardTask(void *pvParameters)
{
  uint8_t old_code = 0xFF;
  uint8_t code;
  uint32_t last_send_time = 0;

  (void)pvParameters;

  while (1) {
    code = Get_KeyCode();

    if (code == 0xFF) {
      old_code = 0xFF;
      //

      if (xSemaphoreTake(game_mutex, pdMS_TO_TICKS(10)) == pdTRUE) {
        tetris_game.fast_drop = false;
        xSemaphoreGive(game_mutex);
      }
    } else {
      uint32_t current_time = HAL_GetTick();

      if (code != old_code) {
        //                                        -

        xQueueSend(key_queue, &code, 0);
        old_code = code;
        last_send_time = current_time;
      } else {
        //
        if (code == 7 && (current_time - last_send_time) >= 100) {
          xQueueSend(key_queue, &code, 0);
          last_send_time = current_time;
          //

          if (xSemaphoreTake(game_mutex, pdMS_TO_TICKS(10)) == pdTRUE)
             {
            tetris_game.fast_drop = true;
            xSemaphoreGive(game_mutex);
```

```
36            }
37          }
38        }
39      }
40
41      vTaskDelay(pdMS_TO_TICKS(20)); //                    20
42    }
43  }
```

Листинг 2: Реализация задачи KeyboardTask

Используемые функции FreeRTOS:

- `xQueueSend()` — отправка данных в очередь (строки 252, 259)
- `xSemaphoreTake()` — захват мьютекса (строки 243, 262)
- `xSemaphoreGive()` — освобождение мьютекса (строки 245, 264)
- `vTaskDelay()` — задержка задачи на 20 мс (строка 270)

3.1.3 Реализация задачи GameTask

```
1  void vGameTask(void *pvParameters)
2  {
3    uint8_t key_code;
4
5    (void)pvParameters;
6
7    while (1) {
8      //
9      if (xQueueReceive(key_queue, &key_code, pdMS_TO_TICKS(10)) ==
         pdTRUE) {
10       //
11
12       if (xSemaphoreTake(game_mutex, portMAX_DELAY) == pdTRUE) {
13
14         if (!tetris_game.game_over) {
15           switch (key_code) {
16             case 3: //               4 (          )
17               Tetris_MovePiece(&tetris_game, -1, 0);
18               break;
19             case 5: //               6 (            )
20               Tetris_MovePiece(&tetris_game, 1, 0);
21               break;
22             case 7: //               2 (        )
23               Tetris_DropPiece(&tetris_game);
24               break;
25             case 4: //               5 (
                                    )
26               Tetris_RotatePiece(&tetris_game);
27               break;
28             case 6: //               1 (
                                    )
29               Tetris_RotatePieceCounterClockwise(&tetris_game);
```

```
29                break;
30              case 10: //              0 (          )
31                tetris_game.paused = !tetris_game.paused;
32                break;
33            }
34          } else {
35            Tetris_Init(&tetris_game);
36          }
37
38          xSemaphoreGive(game_mutex);
39        }
40      }
41
42      //
43      if (xSemaphoreTake(game_mutex, pdMS_TO_TICKS(10)) == pdTRUE) {
44        uint32_t lines_cleared_before = tetris_game.lines_cleared;
45        Tetris_Update(&tetris_game);
46
47        if (tetris_game.lines_cleared > lines_cleared_before) {
48          sound_off_time = HAL_GetTick() + 150;
49        }
50
51        xSemaphoreGive(game_mutex);
52      }
53
54      //

55      uint32_t current_time = HAL_GetTick();
56      if (sound_off_time > 0 && current_time >= sound_off_time) {
57        Buzzer_Set_Volume(BUZZER_VOLUME_MUTE);
58        sound_off_time = 0;
59      }
60
61      vTaskDelay(pdMS_TO_TICKS(10)); //                  10
62    }
63  }
```

Листинг 3: Реализация задачи GameTask

Используемые функции FreeRTOS:

- `xQueueReceive()` — получение данных из очереди с таймаутом 10 мс (строка 283)
- `xSemaphoreTake(game`$_m utex, portMAX_DELAY) - - -$ (285)
- `xSemaphoreTake(game`$_m utex, pdMS_TO_TICKS(10)) - - -$ (321)
- `xSemaphoreGive()` -- освобождение мьютекса (строки 316, 331)
- `vTaskDelay()` -- задержка задачи на 10 мс (строка 341)

3.1.4 Реализация задачи DisplayTask

```
1  void vDisplayTask(void *pvParameters)
2  {
```

```
  (void)pvParameters;

  while (1) {
    //

    if (xSemaphoreTake(game_mutex, portMAX_DELAY) == pdTRUE) {
      if (xSemaphoreTake(display_mutex, portMAX_DELAY) == pdTRUE) {
        //
        Tetris_Draw(&tetris_game);
        oled_UpdateScreen();

        //
        xSemaphoreGive(display_mutex);
      }
      xSemaphoreGive(game_mutex);
    }

    vTaskDelay(pdMS_TO_TICKS(50)); //
                                   50
  }
}
```

Листинг 4: Реализация задачи DisplayTask

Используемые функции FreeRTOS:

- `xSemaphoreTake(game`$_m utex, portMAX_DELAY) - - -$ (352)
- `xSemaphoreTake(display`$_m utex, portMAX_DELAY) - - -$ (353)
- `xSemaphoreGive() -- освобождение мьютексов (строки 359, 361)`
- `vTaskDelay() -- задержка задачи на 50 мс (строка 364)`

3.2 Очереди (Queues)

Очереди используются для асинхронной передачи данных между задачами без блокировки.

3.2.1 Создание очереди

```
//
key_queue = xQueueCreate(10, sizeof(uint8_t));
if (key_queue == NULL) {
  Error_Handler();
}
```

Листинг 5: Создание очереди для клавиатуры

Параметры:

- `10` — максимальное количество элементов в очереди
- `sizeof(uint8`$_t) - - -$ (1)

Тип очереди: FIFO (First In First Out) — первый пришедший элемент первым извлекается.

3.2.2 Отправка данных в очередь

```
//    KeyboardTask
xQueueSend(key_queue, &code, 0);
```

Листинг 6: Отправка кода клавиши в очередь

Параметры:

- `key`$_{queue}$ $---$ *handle*
- `&code -- указатель на данные для отправки`
- `0 -- таймаут (0 = неблокирующая операция)`

Поведение: Если очередь полна, функция возвращает ошибку без ожидания.

3.2.3 Получение данных из очереди

```
//    GameTask
if (xQueueReceive(key_queue, &key_code, pdMS_TO_TICKS(10)) == pdTRUE) {
  //
}
```

Листинг 7: Получение кода клавиши из очереди

Параметры:

- `key`$_{queue}$ $---$ *handle*
- `&key`$_{code}$ $---$
- `pdMS`$_{T}O_{T}ICKS(10)$ $---$ 10

Поведение:

- Если очередь не пуста — сразу возвращает данные, возвращает `pdTRUE`
- Если очередь пуста — ждет максимум 10 мс
- Если за 10 мс данные не поступили — возвращает `pdFALSE`

3.3 Мьютексы (Mutexes)

Мьютексы используются для защиты общих ресурсов от одновременного доступа несколькими задачами.

3.3.1 Создание мьютексов

```
//
display_mutex = xSemaphoreCreateMutex();
if (display_mutex == NULL) {
  Error_Handler();
}

//
game_mutex = xSemaphoreCreateMutex();
if (game_mutex == NULL) {
  Error_Handler();
}
```

Листинг 8: Создание мьютексов для синхронизации

Особенности мьютексов FreeRTOS:

- Мьютекс может быть захвачен только одной задачей в каждый момент времени
- Задача, захватившая мьютекс, должна его освободить
- Мьютекс имеет приоритет наследования — если задача с высоким приоритетом ждет мьютекс, приоритет задачи, удерживающей мьютекс, временно повышается

3.3.2 Использование мьютекса game_mutex

Мьютекс $\texttt{game}_mutex$

3.3.3 Использование мьютекса display_mutex

Мьютекс $\texttt{display}_mutexOLED$.

```
//    DisplayTask -
if (xSemaphoreTake(game_mutex, portMAX_DELAY) == pdTRUE) {
  if (xSemaphoreTake(display_mutex, portMAX_DELAY) == pdTRUE) {
    Tetris_Draw(&tetris_game);      //

    oled_UpdateScreen();             //

    xSemaphoreGive(display_mutex);   //

  }
  xSemaphoreGive(game_mutex);
}
```

Листинг 9: Использование display_mutex

Важно: Порядок захвата мьютексов всегда одинаковый ($\texttt{game}_mutex$ß

4 Логика синхронизации

4.1 Принципы синхронизации

Система использует двухуровневую синхронизацию:

1. Асинхронная передача данных через очередь:
    - KeyboardTask отправляет коды клавиш в очередь
    - GameTask получает коды из очереди
    - Разделение производителя и потребителя данных
    - Буферизация до 10 нажатий клавиш
2. Синхронизация доступа к общим ресурсам через мьютексы:
    - game-mutex защищает структуру tetris-game
    - display-mutex защищает OLED дисплей
    - Гарантирует консистентность данных

4.2 Сценарии синхронизации

4.2.1 Сценарий 1: Обработка нажатия клавиши

1. KeyboardTask опрашивает клавиатуру через $\texttt{Get}_{K}eyCode()$

1. Обнаружена новая клавиша
2. KeyboardTask отправляет код в $\texttt{key}_{q}ueue\texttt{xQueueSend}()()$
2. KeyboardTask засыпает на 20 мс через vTaskDelay()
3. Планировщик переключается на GameTask (приоритет 3)
4. GameTask получает код из $\texttt{key}_{q}ueue\texttt{xQueueReceive}()(10)$
4. GameTask захватывает $\texttt{game}_{m}utex\texttt{xSemaphoreTake}()()$
4. GameTask обрабатывает команду (изменяет $\texttt{tetris}_{g}ame$)
4. GameTask освобождает $\texttt{game}_{m}utex\texttt{xSemaphoreGive}()$
4. GameTask засыпает на 10 мс

Защита от гонок:

- Мьютекс $\texttt{game}_{m}utex,$
- Очередь $\texttt{key}_{q}ueue$

4.2.2 Сценарий 2: Обновление игровой логики

1. GameTask просыпается через 10 мс
2. GameTask пытается захватить $\texttt{game}_{m}utex10$
2. Если мьютекс свободен:
    - Захватывает мьютекс
    - Вызывает Tetris_Update() (автоматическое падение фигуры)
    - Проверяет удаление линий

- Освобождает мьютекс

3. Если мьютекс занят (например, DisplayTask читает состояние):
   - Ждет максимум 10 мс
   - Если не получил мьютекс -- пропускает цикл обновления
   - Это предотвращает блокировку задачи

Защита от блокировок:

- Использование таймаута предотвращает бесконечное ожидание
- Если мьютекс занят, задача пропускает один цикл обновления, но не блокируется

4.2.3 Сценарий 3: Отрисовка на экране

1. DisplayTask просыпается через 50 мс
2. DisplayTask захватывает $\texttt{game}_m utex()$
3. Гарантирует консистентность данных игры

DisplayTask захватывает $\texttt{display}_m utex()$
Гарантирует эксклюзивный доступ к дисплею

DisplayTask вызывает Tetris_Draw() (читает $\texttt{tetris}_g ame$)

DisplayTask вызывает oled_UpdateScreen() (пишет в дисплей)

DisplayTask освобождает $\texttt{display}_m utex$

DisplayTask освобождает $\texttt{game}_m utex$

DisplayTask засыпает на 50 мс

Защита от гонок:

- Двойной захват мьютексов гарантирует:
  - Данные игры не изменяются во время чтения
  - Дисплей не используется другими задачами во время записи
- Порядок захвата важен: сначала $\texttt{game}_m utex$,
- Порядок освобождения обратный: сначала $\texttt{display}_m utex$,

4.3 Предотвращение взаимных блокировок (Deadlock)

Потенциальная проблема: Если две задачи захватывают мьютексы в разном порядке, возможна взаимная блокировка.

Решение в проекте:

- Всегда используется один и тот же порядок захвата:
  1. $\texttt{game}_m utex$
  1. $\texttt{display}_m utex$

- Порядок освобождения обратный:

  1. `display`$_{m}utex$

  1. `game`$_{m}utex$

Пример безопасной последовательности:

```
// DisplayTask
                  :
Take(game_mutex)      Take(display_mutex)      ...
Give(display_mutex)     Give(game_mutex)
```

4.4 Приоритеты задач и их влияние на синхронизацию

Иерархия приоритетов:

1. **GameTask (приоритет 3)** — высший приоритет
   - Обрабатывает пользовательский ввод
   - Обновляет игровую логику
   - Должна реагировать быстро
2. **KeyboardTask (приоритет 2)** — средний приоритет
   - Опрашивает клавиатуру
   - Может быть временно вытеснена GameTask
3. **DisplayTask (приоритет 1)** — низший приоритет
   - Отрисовка не критична для игрового процесса
   - Может быть вытеснена обеими другими задачами

Влияние на синхронизацию:

- `GameTask` может вытеснить `KeyboardTask` или `DisplayTask` в любой момент
- `DisplayTask` не может вытеснить другие задачи
- Это гарантирует быструю реакцию на пользовательский ввод

4.5 Таймауты и их роль

Таймауты используются для:

1. **Предотвращения блокировок:**
   - `xQueueReceive(..., pdMS_TO_TICKS(10))` — `GameTask` не блокируется надолго
   - `xSemaphoreTake(game_mutex, pdMS_TO_TICKS(10))` — задачи не ждут бесконечно
2. **Обеспечения отзывчивости:**
   - Если мьютекс занят, задача пропускает цикл, но продолжает работу

- Это особенно важно для `GameTask`, которая должна обновляться регулярно

3. **Бесконечное ожидание используется только когда необходимо:**
   - `xSemaphoreTake(game_mutex, portMAX_DELAY)` в `GameTask` при обработке команд
   - `xSemaphoreTake(..., portMAX_DELAY)` в `DisplayTask` при отрисовке
   - Гарантирует завершение критических операций

5 Временные характеристики

5.1 Частоты обновления

Задача	Частота	Период
KeyboardTask	50 Гц	20 мс
GameTask	100 Гц	10 мс
DisplayTask	20 Гц	50 мс
SysTick	1000 Гц	1 мс

Таблица 2: Частоты обновления задач

5.2 Время реакции

- **Реакция на нажатие клавиши:** максимум 30 мс (20 мс опрос + 10 мс обработка)
- **Обновление игровой логики:** каждые 10 мс
- **Обновление экрана:** каждые 50 мс (20 FPS)

5.3 Автоматическое падение фигур

Скорость падения фигур рассчитывается по формуле:

$$drop\_interval = \max(1000 - (level \times 50), 100) \text{ мс}$$

При удержании клавиши “вниз” скорость увеличивается до 50 мс.

6 Прерывания

6.1 SysTick прерывание

Единственное используемое прерывание в системе — это системный таймер SysTick.

```
void SysTick_Handler(void)
{
  HAL_IncTick();
#if (INCLUDE_xTaskGetSchedulerState == 1 )
  if (xTaskGetSchedulerState() != taskSCHEDULER_NOT_STARTED)
  {
#endif
    xPortSysTickHandler();  //
       FreeRTOS
```

```
#if (INCLUDE_xTaskGetSchedulerState == 1 )
  }
#endif
}
```

Листинг 10: Обработчик прерывания SysTick

Функции:

1. `HAL_IncTick()` — инкремент счетчика времени HAL
2. `xPortSysTickHandler()` — вызов планировщика FreeRTOS для переключения задач

Частота: 1000 Гц (каждую миллисекунду)
Роль в синхронизации:

- Обеспечивает кванты времени для планировщика
- Позволяет переключение задач по времени
- Используется для измерения времени в `pdMS_TO_TICKS()`

6.2 Отсутствие других прерываний

Клавиатура опрашивается программно (polling), а не через прерывания. Это упрощает синхронизацию, но увеличивает задержку реакции.

7 Диаграмма последовательностей

На рисунке 1 представлена диаграмма последовательностей взаимодействия задач, очередей и мьютексов в системе.

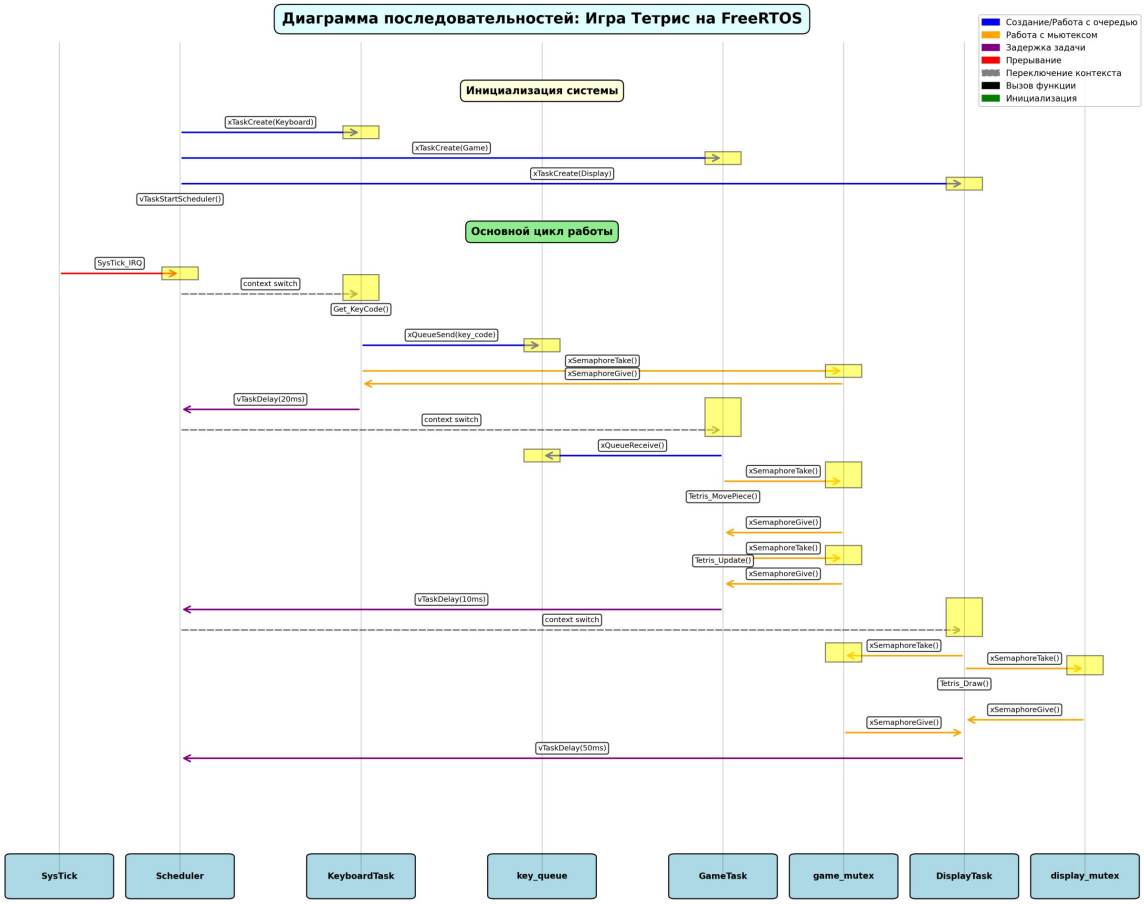

Рис. 1: Диаграмма последовательностей взаимодействия задач FreeRTOS

Диаграмма показывает:

- Инициализацию системы и создание задач
- Основной цикл работы с переключением задач
- Взаимодействие через очередь `key`$_{queue}$
- `Использование мьютексов для синхронизации`
- `Задержки задач через vTaskDelay()`

8 Резюме использования FreeRTOS примитивов

Примитив	**Количество**	**Назначение**
Задачи (Tasks)	3	Разделение функциональности на параллельные потоки
Очереди (Queues)	1	Асинхронная передача кодов клавиш между задачами
Мьютексы (Mutexes)	2	Защита общих ресурсов (структура игры, OLED дисплей)
Планировщик	1	Управление выполнением задач с учетом приоритетов
Прерывания	1 (SysTick)	Системный таймер для планировщика

Таблица 3: Используемые примитивы FreeRTOS

Всего использовано: 8 примитивов FreeRTOS для реализации многозадачной игры Тетрис.

9 Выводы

В ходе выполнения лабораторной работы была реализована игра “Тетрис” с использованием операционной системы реального времени FreeRTOS. Система демонстрирует эффективное использование примитивов синхронизации:

- **Задачи** обеспечивают параллельное выполнение различных функций системы
- **Очереди** позволяют асинхронную передачу данных без блокировок
- **Мьютексы** защищают общие ресурсы от гонок данных
- **Планировщик** обеспечивает своевременное выполнение критических задач

Применение правильных приоритетов задач, использование таймаутов и мьютексов гарантируют отсутствие взаимных блокировок и обеспечивают отзывчивость системы.