|  |  |
| --- | --- |
| osThreadTerminate(NULL);  osThreadTerminate(myTask03Handle); | Позволяет **уничтожить задачу**, которая выполняется (c помощью параметра NULL) или ту на которую указывает хэндл . |
| //глобальное объявление  osThreadId myTask03Handle;  void StartTask03(void const \* argument);  //создание задачи  osThreadDef(myTask03, StartTask03, osPriorityIdle, 0, 128);  myTask03Handle = osThreadCreate(osThread(myTask03), NULL);  //описание задачи void StartTask03(void const \* argument) {  …  } | **Создает задачу:**   1. Объявляем   #define osThreadDef(name, thread, priority, instances, stacksz)  Name – имя задачи  Thread – указатель на адрес функции в памяти (название ф-ции)  Priority – начальный приоритет  Instances – количество возможных экземпляров потока (0 – не ограничено)  Stacksz – Размер стека в байтах   1. Создаем экземпляр задачи   osThreadId osThreadCreate (const osThreadDef\_t \*thread\_def, void \*argument)  osThreadDef\_t – имя потока  argument – аргумент |
| osThreadList((unsigned char \*)str\_buf); | Вернет в массив str\_buf **информацию о текущих процессах** Работает только при включенных режимах отладки операционной системы, конкретно  GENERATE\_RUN\_TIME\_STATS  USE\_TRACE\_FACILITY  USE\_STATS\_FORMATTING\_FUNCTIONS  Описание данных: (B : Blocked, R : Ready, D : Deleted, S : Suspended), далее приоритет, далее стек, далее ID задачи |
| uint32\_t osKernelSysTick(void); | Возвращает **количество системных квантов** операционной системы в формате uint32\_t. |
| osMessageQId myQueue01Handle; //глобальное объявление ID очереди  osMessageQDef (myQueue01, 16, uint32\_t); // создание очереди  myQueue01Handle = osMessageCreate(osMessageQ(myQueue01), NULL); // Создание очереди  osMessagePut(myQueue01Handle, VARIABLE, 50000); // положить что-то в очередь  //взять из очереди  osEvent event; //создаем структуру данных которые нужно принять  event = osMessageGet(myQueue01Handle, 100); //забираем данные из структуры  if (event.status == osEventMessage) { //проверяем, не пустая ли структура  do {  printf(" %i", event.value.v); //данные хранятся в event.value.v  event = osMessageGet(myQueue01Handle, 100); // проверяем, есть ли что еще в очереди  } while (event.status == osEventMessage); // проверяем, есть ли что еще в очереди  } | Добавляет **очередь из переменных** |

|  |  |
| --- | --- |
| typedef struct {  …  } struct\_typedef\_name; //объявление структуры  osMailQId struct\_Queue\_ID; //объявляем ID структуры  osMailQDef(stroutqueue, 3, struct\_typedef\_name); //объявляем имя структуры и размер  // Создаем структуру для работы с нашей структурой  struct\_Queue\_ID = osMailCreate(osMailQ(stroutqueue), NULL);  // Отправка данных  struct\_typedef\_name \*qstruct;  do {  qstruct = osMailAlloc(struct\_Queue\_ID, osWaitForever); // выделяем память  if (qstruct == 0) osDelay(1); //проверка на корректность указателя  else break; // пытаться выделить пока не получится это сделать  } while (1);  sprintf(qstruct->text,"hello! "); //можем пользоваться структурой  osMailPut(struct\_Queue\_ID, qstruct); //отправляем структуру в очередь  // Прием данных  osEvent event1; //создаем структуру для работы с принимаемой структурой  struct\_typedef\_name \*qstruct; // создаем указатель на принимаемую структуру  // заполняем структуру для работы с нашей структурой  event1 = osMailGet(struct\_Queue\_ID, osWaitForever);  if (event1.status == osEventMail) { //если очередь не пуста  do {  qstruct = event1.value.p; // присваиваем нужный указатель нашей структуре  printf("\n%s%i",qstruct->text,qstruct->value1); //работаем со структурой  osMailFree (struct\_Queue\_ID, event1.value.p); // освобождаем память  event1 = osMailGet(struct\_Queue\_ID, 0); // проверяем очередь  } while (event1.status == osEventMail); // повторять пока очередь есть  } | Добавляет **очередь из структур** |
|  |  |
|  |  |
|  |  |