
NobleProg

Wprowadzenie do FreeRTOS

Grzegorz Mazur

NobleProg

Co to jest RTOS?

- Real-Time Operating System - oprogramowanie umożliwiające quasirównoległą pracę wielu programów (zadań/wątków) na mikrokontrolerze
- Zapewnia przełączanie zadań/wątków i synchronizację pomiędzy nimi
- Nie przeszkadza w uzyskaniu determinizmu czasowego odpowiedzi

Struktury oprogramowania µC (1)

- Inicjowanie + pętla zdarzeń
  - Trywialna budowa, tylko do demonstracji
  - przy >1 zdarzeniu grozi gubieniem zdarzeń
  - powoduje zbędne zużycie energii podczas sprawdzania, czy wystąpiło zdarzenie
- Pętla zdarzeń + przerwania
  - Trudności w oszczędzaniu energii, krótszy czas reakcji na zdarzenia, wymaga umiejętnej dekompozycji czynności pomiędzy procedury obsługi przerwań i pętlę zdarzeń

Struktury oprogramowania µC (2)

- Inicjowanie + procedury obsługi zdarzeń (przerwań), bez pętli zdarzeń
  - Minimalizacja poboru energii, najszybsza reakcja na zdarzenia, wymaga dużych umiejętności programisty i panowania nad synchronizacją zdarzeń
- RTOS → wiele pętli zdarzeń
  - Łatwy koncepcyjnie projekt złożonego oprogramowania z wieloma pętlami zdarzeń, kosztem narzutu czasu wnoszonego przez RTOS

Program, zadanie, wątek (1)

- Program – statyczny zapis algorytmu i związanych z nim danych
  - W systemach jednozadaniowych/jednowątkowych pojęcia programu, zadania i wątku są równoważne
- Zadanie/proces (task/process) – instancja programu w systemie wielozadaniowym
  - Proces ma własną, prywatną pamięć, zawierającą jego kod i dane
  - W systemie może działać wiele procesów, w tym również wiele instancji tego samego programu, z których każda ma oddzielną pamięć
  - W systemach wielozadaniowych jednowątkowych proces=wątek

Program, zadanie, wątek (2)

- Wątek (thread) – instancja wykonania w obrębie procesu/zadania
  - Wątki zadania mają wspólną pamięć, nie chronioną przed innymi wątkami
    - kod, dane statyczne, sterta
  - każdy wątek ma własny stos
  - wątek może mieć również własne dane statyczne
  - W prostszych komputerach bez sprzętowego zarządzania pamięcią nie ma zadań (rozdzielności pamięci), a są wątki (np. w systemie FreeRTOS)

RTOS dla µC vs. „pełny” OS

- Brak izolacji zadań – jeden program/zadanie, wiele wątków
  - Brak ochrony pamięci
  - Dane statyczne wspólne dla wszystkich wątków
  - Wszystkie wątki mogą używać wszystkich procedur programu
- Funkcjonalność ograniczona do przełączania wątków i synchronizacji pomiędzy nimi
  - Podstawowy RTOS nie zawiera obsługi urządzeń, systemu plików, interakcji z użytkownikiem

RTOS i „czas rzeczywisty”

- „system czasu rzeczywistego” (ang. real-time system, pol. system nadążny) – to oprogramowanie gwarantujące odpowiedź w czasie wymaganym przez obiekt/urządzenie
- Nie oznacza to „natychmiastowej odpowiedzi”, a jedynie „odpowiedź w wymaganym czasie”

RTOS i „czas rzeczywisty”

- Zastosowanie RTOS wydłuża czas odpowiedzi oprogramowania na zdarzenia
- „tam, gdzie używa się RTOS, nie może być mowy o czasie rzeczywistym” ;)
- użycie RTOS spowalnia działanie oprogramowania, ale upraszcza jego projekt!

FreeRTOS

- Najpopularniejszy mały, darmowy RTOS
  - aktualnie pod marką Amazon
  - Istnieją dziesiątki podobnych systemów o zbliżonej funkcjonalności
- Dostępny dla >30 architektur
  - Łatwa przenośność na nowe architektury
- Wersje komercyjne (wsparcie, certyfikacja)
  - OpenRTOS
  - SafeRTOS

Dokumentacja

- Dostępna online – freertos.org
  - Mastering the FreeRTOS Real Time Kernel - a Hands On Tutorial Guide
  - Reference Manual

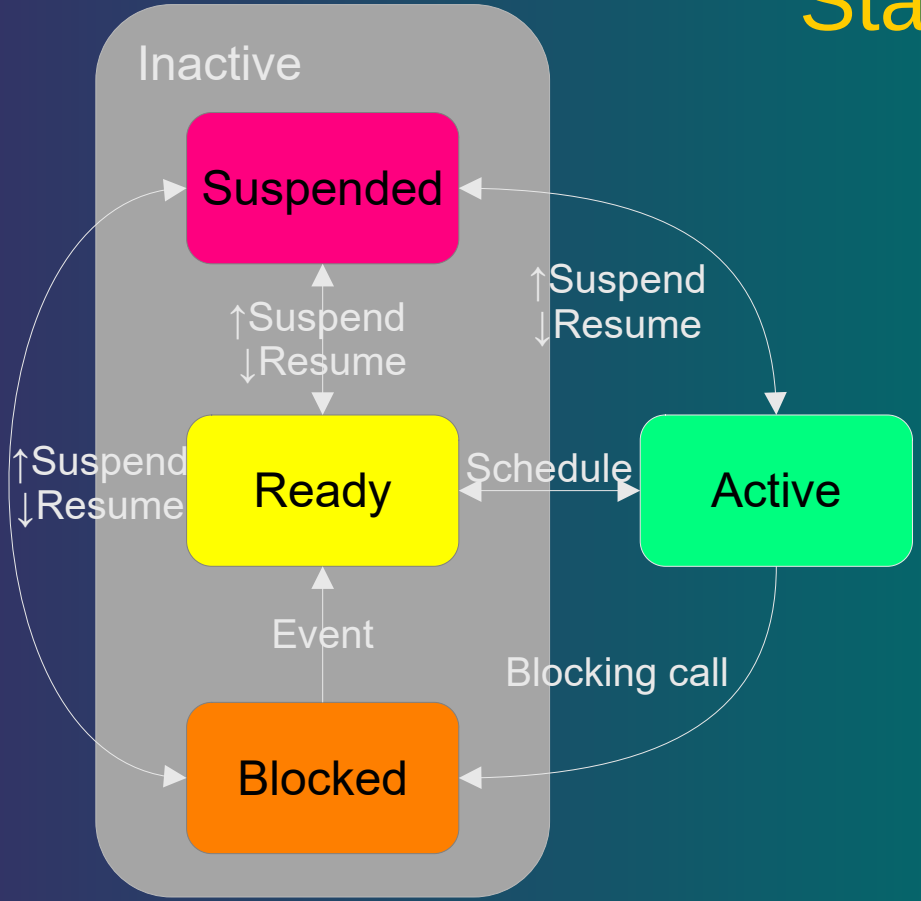
Wątek w systemie FreeRTOS

- Fragment kodu, zwykle w postaci procedury z pętlą nieskończoną, który może być wykonywany quasirównolegle z innymi wątkami
- Każdy wątek ma prywatny stos, zawierający dane dynamiczne automatyczne
- Pozostałe dane – statyczne i sterta – są współdzielone przez wątki
- Wątki o wspólnym kodzie mogą mieć unikatowe dane w postaci struktury
  - procedura wątku ma jeden argument; najczęściej jest to wskaźnik na strukturę zawierającą parametry wątku

Timer systemowy i podział czasu

- Zazwyczaj FreeRTOS pracuje z podziałem czasu
  - Jest to opcja konfiguracji, która może być wyłączona
- Praca z podziałem czasu wymaga użycia timera systemowego, zgłaszającego przerwania ze stałą częstotliwością
  - Typowo 1 kHz (można zmienić)
  - Częstotliwość ta określa okres przełączania zadań

Stany wątku



- Aktywny – aktualnie wykonywany przez procesor
- Nieaktywny – aktualnie niewykonywany
  - Gotowy – może stać się aktywny (na nic nie czeka; stan początkowy)
  - Zawieszony – aktualnie wyłączony
  - Zablokowany – oczekujący na zdarzenie

Priorytety wątków

- FreeRTOS umożliwia różnicowanie priorytetów wątków
- Liczba poziomów priorytetów zależy od konfiguracji systemu
- Od liczby poziomów zależy zajętość pamięci przez system
- Istnieją przynajmniej dwa poziomy priorytetów – w systemie istnieje specjalne zadanie bezczynności o najniższym priorytecie

Szeregowanie wątków

- Wątek może być aktywowany jeśli:
  - Jest gotowy
    i
  - Nie istnieje wątek aktywny ani gotowy o wyższym priorytecie
    i
  - System pracuje z podziałem czasu lub aktualnie aktywny wątek o tym samym priorytecie sam zrezygnuje z aktywności – zmieni swój stan z aktywnego na gotowy – osThreadYield()

Szeregowanie wątków

- Wątek NIE może być aktywowany jeśli:
  - Nie jest gotowy
    lub
  - Istnieje wątek aktywny lub gotowy o wyższym priorytecie
    lub
  - System pracuje bez podziału czasu i aktualnie jest aktywny wątek o tym samym priorytecie

Budowa FreeRTOS

- FreeRTOS składa się (zaledwie) z kilku modułów w języku C odpowiedzianych za różne aspekty funkcjonalności systemu
- Obowiązkowy moduł zarządzania wątkami
- (prawie) obowiązkowy moduł obsługi kolejek
- Opcjonalne:
  - Timer programowy, notyfikacje, zdarzenia, współprogramy

Konfiguracja FreeRTOS

- Wszystkie opcje i parametry są ustawiane przez programistę w pliku FreeRTOSConfig.h
- Opcje decydują o udostępnieniu danej funkcjonalności
- Wartości opcji i parametrów ilościowych mają wpływ na zajętość pamięci programu i danych

FreeRTOS a CMSIS

- CMSIS – zestandaryzowane przez ARM fragmenty oprogramowania mikrokontrolerów Cortex, niezależne od producenta mikrokontrolera
- Istnieją dziesiątki systemów o funkcjonalności b. podobnej do FreeRTOS
  - … ale każdy ma nieco inne wywołania funkcji systemowych
- Rozwiązanie – jednolity interfejs CMSIS dla wszystkich RTOS – CMSIS-RTOS
  - aktualna wersja: CMSIS-RTOS2 (istotne różnice w stosunku do CMSIS-OS/CMSIS-RTOS)

RTOS w CMSIS

- Procedury RTOS są wywoływane za pośrednictwem procedur CMSIS o nazwach rozpoczynających się od „os”
- Nazwy procedur nie zależą od użytego RTOS – ten sam użytkowy program źródłowy może zostać skompilowany z różnymi RTOS
- Można zamiennie używać wywołań CMSIS i czystego FreeRTOS, powoduje to jednak zmniejszenie czytelności kodu
- CMSIS-RTOS udostępnia takie same wywołania funkcji z wątków i z procedur obsługi przerwań

Wątek we FreeRTOS

- Wątek jest opisany przez:
  - Nazwę (tylko dla ułatwienia debugowania)
  - Priorytet
  - Adres procedury rozpoczynającej wykonanie wątku
  - Rozmiar stosu (wyrażony w słowach)
- Po utworzeniu wątek jest identyfikowany w systemie przez uchwyt (handle)

Stos wątku

- Alokowany przez FreeRTOS przy tworzeniu wątku
- Zawiera wszystkie obiekty lokalne (argumenty, zmienne lokalne procedur zadania), informację związaną z wywoływaniem i powrotami z procedur
- Przechowuje stan wątku (wartości rejestrów procesora) gdy wątek jest nieaktywny (*)

Rozmiar stosu wątku

- Deklarowany przez programistę (w słowach, nie bajtach!)
- Musi zapewnić przechowywanie danych lokalnych przy maksymalnym możliwym zagnieżdżeniu procedur, składowanie rejestrów procesora dla każdego poziomu zagnieżdżenia i składowanie stanu podczas nieaktywności (*)

Procedura wątku

- Procedura wątku ma opcjonalny parametr
  - Typ dowolny, rzutowalny na void*
  - W praktyce może to być dana skalarna lub wskaźnik na strukturę
  - W ten sposób można użyć jednej procedury dla wielu podobnych zadań
- Powrót z procedury zadania jest niedozwolony
  - Typowo procedura kończy się pętlą nieskończoną for (;;)

Odmierzanie czasu

- Opóźnienia i odmierzanie czasu można uzyskać na dwa sposoby:
  - Wywołanie funkcji oczekiwania w wątku – osDelay(), osDelayUntil()
    - Opóźnienie może być większe od zadanego przy każdym wywołaniu
  - Programowe timery systemowe – osTimerNew(), osTimerStart()
    - Jednokrotny – wywołuje procedurę po określonym czasie
    - Periodyczny – wywołuje procedurę co określony okres – bez kumulacji opóźnień

Komunikacja i synchronizacja wątków

- Podstawową funkcją RTOS jest zarządzanie aktywnością wątków
  - aktywowanie wtedy, gdy są one gotowe
  - Usypianie wtedy, gdy nie mają co robić
- Do przekazywania informacji o potrzebie aktywowania wątku służą systemowe mechanizmy synchronizacji
- Pierwotnym mechanizmem komunikacji i synchronizacji FreeRTOS jest kolejka

Kolejka w RTOS

- Kolejka – bufor danych o określonej pojemności i typie danych, służący do przekazywania danych („wiadomości”) pomiędzy producentem i konsumentem
- podstawowe wywołania: osMessageQueueGet(), osMessageQueuePut()
- Przy wywołaniu osMessageQueuePut() producent czeka, gdy kolejka jest pełna
- Przy wywołaniu osMessageQueueGet() konsument czeka, gdy kolejka jest pusta

Inne mechanizmy synchronizacji FreeRTOS

- Semafor wielowartościowy – służy do alokacji zasobu, który może być jednocześnie używany przez n użytkowników
- Semafor binarny – zezwala na wykonanie akcji
- Mutex – semafor binarny służący do alokacji pojedynczego zasobu
- znaczniki
  - wątku
  - zdarzeń

Semafor wielowartościowy

- Służy do alokacji zasobu, który może być jednocześnie używany przez nie większą od określonej liczbę użytkowników
  - Zrealizowany jako kolejka pustych elementów
  - Alokacja zasobu – pobranie elementu z kolejki
  - Zwolnienie zasobu – wstawienie elementu do kolejki
- Zajęcie zasobu: osSemaphoreAcquire()
- Zwolnienie zasobu: osSemaphoreRelease()

Semafor binarny

- Służy do zezwalania na wykonanie akcji przez wątek
- Zrealizowany jako jednoelementowa kolejka pustych elementów
- Zezwolenie na akcję/”podniesienie semafora” – osSemaphoreRelease()
- Wykonanie akcji z ew. oczekiwaniem na zezwolenie/”opuszczenie semafora” – osSemaphoreAcquire()

Mutex

- Semafor binarny służący wyłącznie do alokacji pojedynczego zasobu
  - zasób jest zajmowany przez wątek, a następnie zwalniany przez ten sam wątek
- Musi zostać podniesiony przez tego, kto go opuścił
- Priorytet wątku, który zajął mutex, jest podwyższany przy próbie zajęcia mutexa przez wątek o wyższym priorytecie
- Alokacja zasobu: osMutexAcquire()
- Zwolnienie zasobu: osMutexRelease()

Operacje blokujące

- Operacje, które mogą zmienić stan wątku na zablokowany/oczekujący:
  - osThreadJoin
  - osThreadFlagsWait, osEventFlagsWait
  - osDelay, osDelayUntil
  - osMutexAcquire, osSemaphoreAcquire
  - osMessageQueueGet, osMessageQueuePut
- Przy wywołaniu z wątku operacje blokujące mogą mieć podany limit czasu oczekiwania
  - jeśli jest to zbędne, używamy wartości osWaitForever
  - przy wywołaniach z przerwań parametr ten musi mieć wartość 0

Sterta systemowa

- Obszar pamięci służący systemowi do alokacji wszelkich obiektów (list zadań o każdym priorytecie, stosów zadań, kolejek, zbiorów kolejek, listy timerów programowych itp.)
- Zarządzanie stertą – moduł heap_x.c
  - Kilka do wyboru
- System musi dysponować stertą o odpowiedniej pojemności
- Ograniczenie dynamicznej alokacji pamięci ułatwia eliminację błędów w oprogramowaniu
  - należy stosować statyczną alokację opisów obiektów FreeRTOS – wątków, timerów, kolejek

Start systemu

- Uruchomienie systemu następuje poprzez serię wywołań procedur w języku C
- Po zainicjowaniu zasobów sprzętowych mikrokontrolera należy utworzyć przy użyciu wywołań procedur FreeRTOS potrzebne obiekty systemowe – zadania, kolejki, timery
- FreeRTOS jest uruchamiany przez wywołanie procedury vTaskStartScheduler()/osKernelStart()

Czas na praktykę...

- Narzędzia do ćwiczeń
- FreeRTOS w STM32CubeIDE

Środowisko do ćwiczeń

- Płytka uruchomieniowa serii Nucleo-144
- STM32CubeIDE
  - Darmowe środowisko ST dla STM32, bez ograniczeń zajętości pamięci
  - Zbudowane na bazie dawnego CubeMX (generator aplikacji) oraz Atollic TrueSTUDIO (Eclipse + GNU ARM CC)
  - Umożliwia kompilację i debugowanie programów

STM32CubeIDE

- Generator szkieletów kodu dla STM32
- Zawiera konfigurator peryferiów µC
- Zawiera szereg złożonych komponentów oprogramowania, w tym m.in.
  - Stos USB
  - FatFS
  - FreeRTOS
- Funkcje FreeRTOS wywoływane przez interfejs CMSIS – używamy wersji v2 → CMSIS-RTOS2

FreeRTOS w CubeIDE

- Interaktywna konfiguracja systemu i tworzenie obiektów:
  - Zadań
  - Kolejek i semaforów
  - timerów

Tworzenie zadań

- Zadania są na ogół tworzone przed uruchomieniem systemu – funkcja xTaskCreate()/osThreadCreate()

Odmierzanie czasu – oczekiwanie wątków

- FreeRTOS zlicza okresy timera systemowego i używa programowego licznika czasu do odmierzania odcinków czasu dla zadań
- Wywołanie osDelay() blokuje zadanie na określony czas (w ms)
- osDelayUntil() blokuje zadanie do nadejścia zadanego czasu
  - osKernelGetTickCount() pobiera czas systemowy, który może służyć do obliczenia argumentu dla osDelayUntil()

Timery programowe

- Opcjonalne, włączane przez opcję konfiguracji FreeRTOS (również w CubeIDE).
- Włączenie powoduje uruchomienie ukrytego wątku systemowego o priorytecie określonym przez programistę
- Wątek ten wywołuje procedury zdefiniowane przez programistę w zadanych momentach – jednokrotnie lub periodycznie
- Procedura nie powinna zawierać operacji blokujących, gdyż blokowałaby ona wątek timera, a tym samym inne procedury timerów

Operacje na timerach

- Domyślnie timery są zatrzymane
- Dostępne operacje:
  - Start - osTimerStart()
  - Zatrzymanie
  - Restart
- W praktyce dla timerów cyklicznych potrzebny jest tylko jednorazowy start

Wątek bezczynności

- Gdy żaden wątek użytkownika ani timera programowego nie jest aktywny, system aktywuje wątek bezczynności
- Programista może zdefiniować procedurę wywoływaną z zadania bezczynności
  void vApplicationIdleHook(void)
- Może ona być użyta np. do usypiania procesora

Mechanizmy synchronizacji i komunikacji wątków

- Kolejki, pełniące również rolę semaforów
- Zwykle tworzone przed uruchomieniem systemu
- Kolejka jest charakteryzowana przez liczbę i rozmiar elementów

Operacje na kolejkach

- xQueueSendToBack()/osMessageQueuePut(), xQueueSendToFront() – wstawienie na koniec i początek kolejki
- xQueueReceive()/osMessageQueueGet() – pobranie elementu z początku kolejki
- Ostatni argument określa dozwolony czas oczekiwania na wykonanie operacji (blokowania zadania) – po tym czasie operacja kończy się niepowodzeniem
  - osWaitForever → bez lmitu czasu

Semafory

- Semafory są kolejkami pustych elementów
- Operacje semaforowe zaimplementowane we FreeRTOS jako makra rozwijane w operacje na kolejkach
- xSemaphoreGive()/osSemaphoreRelease() - „podniesienie” semafora
- xSemaphoreTake()/osSemaphoreAcquire() – wejście pod semafor („opuszczenie”)

Zbiory kolejek

- Zbiór kolejek umożliwia oczekiwanie zadania na jedno z kilku możliwych zdarzeń
- Semafory też są kolejkami, więc mogą być elementami zbiorów
- Zadanie zostaje odblokowane przy wystąpieniu dowolnego ze zdarzeń ze zbioru

FreeRTOS i przerwania

- FreeRTOS „nie wie” o przerwaniach
- Procesor ARM pracuje w dwóch trybach – wątku i obsługi wyjątku
- Zadania we FreeRTOS działają w w trybie wątku
- W trybie wyjątku pracuje moduł szeregujący

Priorytet procesora w ARM

- W trybie wątku procesor ma zawsze ten sam, najniższy priorytet
- Przerwania mogą mieć różne priorytety
  - Większa liczba poziomów – większe kłopoty dla programisty
- Priorytet przerwania jest zawsze wyższy od priorytetu wątku
  - Przyjęcie przerwania skutkuje wywłaszczeniem (sprzętowym) wątku – zadania lub jądra systemu

Wywłaszczanie – RTOS i ARM

- Oba mechanizmy wywłaszczania są niezależne
- RTOS nie widzi wywłaszczania sprzętowego
- Oprogramowanie (RTOS) może w razie potrzeby blokować przerwania
- Zablokowanie przerwań przez zadanie skutkuje niemożnością wywłaszczenia przez RTOS i sprzęt

Blokowanie przerwań

- Krytyczne operacje jądra systemu muszą być wykonywane przy podwyższonym priorytecie procesora (zablokowanych niektórych lub wszystkich przerwaniach)

Przerwania i usługi FreeRTOS

- Procedury obsługi przerwań mogą wykonywać operacje synchronizacyjne podobne do dostępnych dla zadań
  - Semafory, kolejki
- W natywnym FreeRTOS operacje te korzystają z alternatywnych wywołań, o nazwach kończących się frazą FromISR
  - Nieistotne jeśli używamy interfejsu CMSIS-RTOS

Obsługa przerwań

- Przerwanie jest obsługiwane w kwancie czasowym aktywnego zadania
  - Niejako „na jego koszt”
- We FreeRTOS dla ARM obsługa wyjątków nie korzysta ze stosów zadań
  - Określając rozmiar stosu zadania nie trzeba brać pod uwagę wyjątków

Wołanie usług systemowych z przerwań

- Usługi systemowe są wykonywane przy podniesionym priorytecie procesora
  - Wartość priorytetu konfigurowana przez programistę w ustawieniach FreeRTOS
- Usługa systemu może być wołana z przerwania tylko wtedy, gdy priorytet przerwania jest niższy lub równy priorytetowi usług!
  - należy zadbać o odpowiednią konfigurację priorytetów przerwań i priorytetu wywołań jądra systemu
- Przy wołaniu usług RTOS z przerwań parametr czasu oczekiwania musi mieć wartość 0

Wymuszenie wywołania modułu szeregującego

- Jeśli zadanie nie ma co robić i na nic nie czeka – może ono oddać swój czas systemowi, który natychmiast aktywuje kolejne zadanie
  - osThreadYield()
- Operacja ta jest również często wywoływana przez procedury obsługi przerwań, w celu szybkiego aktywowania zadania czekającego na zdarzenie wynikające z przerwania
  - przerwanie jest obsługiwane w kontekście wątku, w czasie wykonania którego wystąpiło

Kolejki, semafor, przerwania – demo

- Zadanie przetwarzania:
  - pobiera dane z kolejki odbiorczej, przetwarza, zapisuje do kolejki nadawczej
- Zadanie nadawania:
  - Pobiera dane z kolejki nadawczej
  - Włącza przerwanie nadawania UART
  - Czeka na semafor zezwolenia na nadawanie
  - Wysyła dane przez UART

Kolejki, semafor, przerwania – demo

- Przerwanie UART:
  - Odbiór: zapisuje dane do kolejki odbiorczej
  - Nadawanie: wyłącza przerwanie nadawania i podnosi semafor zezwalający na nadawanie
- Jedynym sposobem wycofania zgłoszenia przerwania nadawania UART jest zapis danej do wysłania, dlatego przerwanie to musi być wyłączane, gdy nie ma następnego znaku do wysłania

Blokowanie przełączania zadań

- Sekcje krytyczne
  - taskENTER_CRITICAL() taskEXIT_CRITICAL()
  - Blokują przerwania – ograniczone zastosowanie
- Zatrzymanie szeregowania – bieżące zadanie pozostaje aktywne, przerwania są obsługiwane
  - osKernelLock(), osKernelUnlock()

Ankieta

https://plbe.nobleprog.com/tf/126943