
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 oprogramowania na zdarzenia

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 równolegle działających pętli zdarzeń
  - Łatwy koncepcyjnie projekt złożonego oprogramowania z wieloma pętlami zdarzeń obsługującymi poszczególne zdarzenia, 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
  - Zadanie ma własną, prywatną pamięć, zawierającą jego kod i dane
  - W systemie może działać wiele zadań (w tym również wiele instancji tego samego programu), z których każde ma oddzielną pamięć
  - W systemach wielozadaniowych jednowątkowych zadanie/proces=wątek

Program, zadanie, wątek (2)

- Wątek (thread) – instancja wykonania w obrębie 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 (opcja)
  - W prostszych komputerach bez sprzętowego zarządzania pamięcią nie ma zadań (rozdzielności pamięci), ale mogą istnieć 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 – mogą one zostać dodane jako składniki opcjonalne

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” ;) – *Tilen Majerle*
- 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

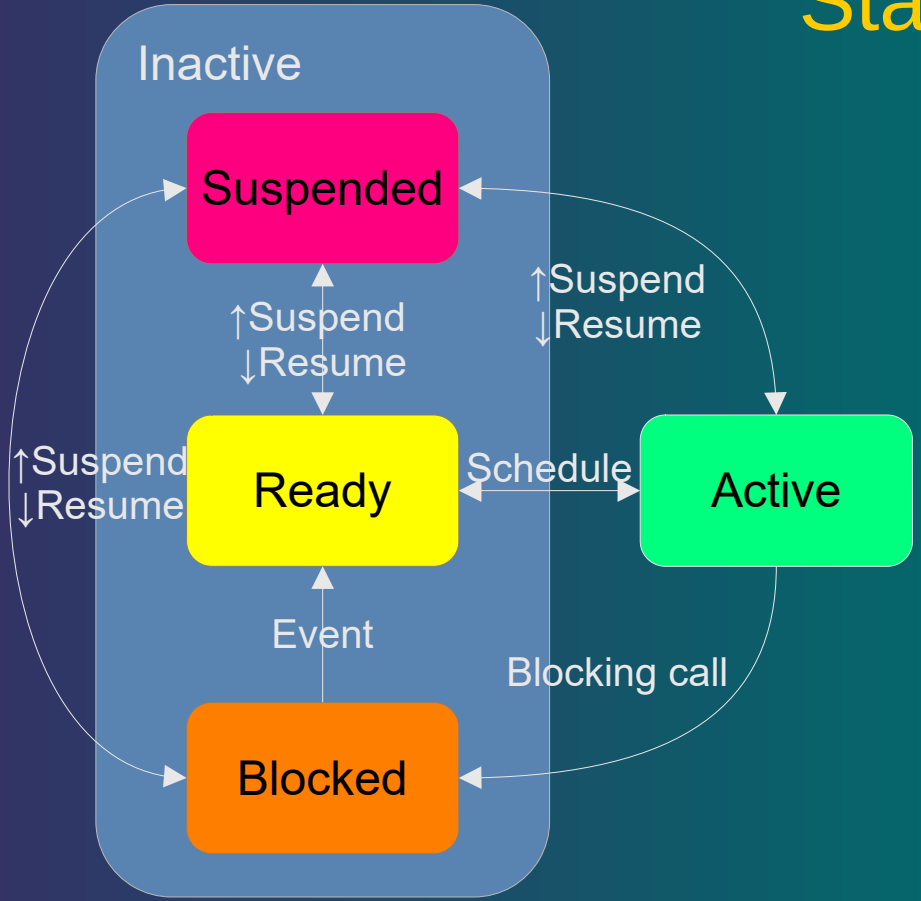
Wątek w systemie FreeRTOS

- Fragment kodu, zwykle w postaci procedury z pętlą nieskończoną, który może być wykonywany quasirównolegle (naprzemiennie) 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, pełniącej rolę danych statycznych wątku
  - 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
  - W architekturze ARM jest do tego celu używany timer rdzenia – SysTick

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 napisanych w języku C, odpowiedzianych za różne aspekty funkcjonalności systemu
- Niezbędny moduł zarządzania wątkami
- (prawie) niezbędny 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 nazwy i 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)
  - Dokumentacja: https://arm-software.github.io/CMSIS_5/RTOS2/html/index.html

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
  - Uwaga na jednostkę zmiennopozycyjną – 32 rejestry
- Narzędzia dostępne w IDE umożliwiają określenie zapotrzebowania na przestrzeń stosu

Procedura wątku

- Procedura wątku ma opcjonalny parametr
  - Typ dowolny, rzutowalny na void*
  - W praktyce może to być dana skalarna lub wskaźnik na strukturę
    - Struktura zawiera dane statyczne wątku
  - W ten sposób można użyć jednej procedury dla wielu podobnych wątków, np. obsługujących różne instancje tego samego modułu peryferyjnego
- Powrót z procedury zadania jest niedozwolony
  - Typowo procedura kończy się pętlą nieskończoną for (;;)
  - ew. może wywoływać funkcję usunięcia wątku

Współprocedury (coroutines)

- Dawny model współbieżności bazujący na jawnym i dobrowolnym przekazywaniu sterowania pomiędzy wykonywanymi quasirównolegle fragmentami kodu.
  - Używany dawniej w komputerach o bardzo ograniczonych zasobach pamięci (np. < 1 KiB)
- Współprocedury korzystają naprzemiennie z jednego stosu
  - Nie mogą mieć zmiennych lokalnych
- Mechanizm dostępny we FreeRTOS z powodów historycznych
  - Użycie niezalecane

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
    - Poprawne użycie osDelayUntil() umożliwia uniknięcie kumulacji opóźnień
  - Programowe timery systemowe – osTimerNew(), osTimerStart()
    - Jednokrotny – wywołuje procedurę po określonym czasie
    - Periodyczny – wywołuje procedurę co określony okres – bez kumulacji opóźnień
- Wątek lub procedura timera mogą być aktywowane później, niż zostało to zaplanowane przez programistę

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
- Kolejki są globalne dla systemu – każdy wątek może używać każdej kolejki
- podstawowe wywołania: osMessageQueueGet(), osMessageQueuePut()
- Przy wywołaniu osMessageQueuePut() producent czeka (zostaje zablokowany), gdy kolejka jest pełna
- Przy wywołaniu osMessageQueueGet() konsument czeka (zostaje zablokowany), 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 - zbiory semaforów binarnych
  - 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() - blokuje wątek gdy semafor ma wartość 0
- Zwolnienie zasobu: osSemaphoreRelease() - zawsze możliwe

Semafor binarny

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

Mutex

- Semafor binarny służący wyłącznie do alokacji pojedynczego zasobu, z którego może korzystać kilka wątków na zasadzie wyłączności
  - zasób jest zajmowany, a następnie zwalniany przez ten sam wątek
- Musi zostać zwolniony przez wątek, który go zajął
- Priorytet wątku, który zajął mutex, jest czasowo podwyższany przy próbie zajęcia mutexa przez wątek o wyższym priorytecie
- Alokacja zasobu: osMutexAcquire()
- Zwolnienie zasobu: osMutexRelease()
- Używany np. w celu uzyskania wyłącznego dostępu do interfejsu

Znaczniki wątku

- Związany z wątkiem zbiór zmiennych logicznych, sterujących działaniem danego wątku
- Funkcjonalność podobna do semaforów binarnych
  - Operacje na znacznikach są szybsze od operacji na semaforach
- Wątek może czekać na dowolny podzbiór znaczników:
  - Pojedynczy znacznik
  - Jeden znacznik z określonego zbioru
  - Wszystkie znaczniki podzbioru
- Znaczniki, na które wątek czekał, są zerowane z chwilą aktywacji wątku
  - Opcja, nieobowiązkowa – wątek może jawnie zerować wybrane znaczniki

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 określać 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 wątków o każdym priorytecie, stosów wątków, kolejek, zbiorów kolejek, listy timerów programowych itp.)
- Zarządzanie stertą – moduł heap_x.c
  - Kilka do wyboru; różne algorytmy alokacji
- 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 – wątki, kolejki, timery
- FreeRTOS jest uruchamiany przez wywołanie procedury osKernelStart() [FreeRTOS: vTaskStartScheduler()]

Czas na praktykę...

- Narzędzia do ćwiczeń
- FreeRTOS w STM32CubeIDE

Środowisko do ćwiczeń

- Płytka uruchomieniowa serii Nucleo-64+nakładka lub 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:
  - Wątków
  - Kolejek i semaforów
  - Timerów
- Domyślnie pamięć wszystkich obiektów jest alokowana dynamicznie
  - Warto zmienić alokację na statyczną

Tworzenie wątków

- Wątki są na ogół tworzone przed uruchomieniem systemu – funkcja osThreadCreate() [FreeRTOS: xTaskCreate()]

Odmierzanie czasu – oczekiwanie wątków

- FreeRTOS zlicza okresy timera systemowego i używa programowego licznika czasu do odmierzania odcinków czasu dla wątków
- Wywołanie osDelay() blokuje wątek na określony czas (w ms)
- osDelayUntil() blokuje wątek do nadejścia zadanego czasu
  - osKernelGetTickCount() pobiera czas systemowy, który może służyć do obliczenia argumentu dla osDelayUntil()
  - W celu uzyskania stałego okresu aktywacji wątku
    - Pobieramy wartość czasu z osKernelGetTickCount() jeden raz, przy starcie wątku
    - wyznaczamy argument dla osDelayUntil() zwiększając poprzednią wartość czasu o okres (a nie przez dodanie okresu do osKernelGetTickCount() !)

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 timera 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

Parametryzacja wątków i timerów

- Procedury wątków i timerów mają argument typu void *, który może być rzutowany na dowolny typ wskaźnikowy lub numeryczny
- Tworząc wątek lub timer można określić wartość jego argumentu
- Zazwyczaj jest to wskaźnik na strukturę zawierającą dane specyficzne dla instancji wątku lub timera
  - W naszym przykładzie struktura zawiera pozycję cyfry, okres zmian i bieżącą wartość

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
  - Stała osWaitForever → bez limitu czasu

Semafory

- Semafory są kolejkami pustych elementów
- Operacje semaforowe zaimplementowane we FreeRTOS jako makra rozwijane w operacje na kolejkach
- xSemaphoreGive()/osSemaphoreRelease() - „podniesienie” semafora
  - Zawsze możliwe, nieblokujące, przy maksymalnej wartości nie zmienia stanu semafora
- xSemaphoreTake()/osSemaphoreAcquire() – wejście pod semafor („opuszczenie”)

Zbiory kolejek

- Zbiór kolejek umożliwia oczekiwanie wątku na jedno z kilku możliwych zdarzeń
- Semafory też są kolejkami, więc mogą być elementami zbiorów
- Wątek zostaje odblokowany przy wystąpieniu dowolnego ze zdarzeń ze zdefiniowanego zbioru

FreeRTOS i przerwania

- FreeRTOS „nie wie” o przerwaniach obsługiwanych przez procesor
- Procesor ARM pracuje w dwóch trybach – wątku i obsługi wyjątku
- Zadania we FreeRTOS działają w w trybie wątku
- W trybie wyjątku pracuje moduł szeregujący
- Na Cortex-M FreeRTOS korzysta z trzech wyjątków rdzenia:
  - Przerwania timera systemowego SysTick
  - Przerwania programowego PendSV – do wywołania procedury szeregującej
    - Priorytet PendSV jest ustawiany na najniższy możliwy poziom
  - Pułapki SVC – do uruchomienia pierwszego zadania (*)

Sprzętowy priorytet procesora w ARM

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

Wywłaszczanie – RTOS i ARM

- Oba mechanizmy wywłaszczania (sprzętowe procesora i programowe RTOS) są niezależne
- RTOS nie widzi wywłaszczania sprzętowego
  - Przerwanie zużywa kwant czasowy wątku
- Oprogramowanie (RTOS) może w razie potrzeby blokować przerwania, uniemożliwiając wywłaszczanie sprzętowe
  - Zablokowanie przerwań przez zadanie skutkuje niemożnością wywłaszczenia przez RTOS (bo zostaje zablokowane przerwanie timera systemowego) 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 podwyższonym priorytecie procesora
  - Wartość priorytetu usług jest 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 procedur obsługi 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 gotowe zadanie
  - osThreadYield()
- przerwanie jest obsługiwane w kontekście wątku, w czasie wykonania którego wystąpiło
- Obsługa przerwania może korzystać z mechanizmów synchronizacji RTOS, co może skutkować przejściem do stanu gotowości wątku o priorytecie wyższym od tego, podczas wykonywania którego wystąpiło przerwanie

Wymuszenie wywołania modułu szeregującego

- Odblokowanie wątku o priorytecie wyższym od bieżącego powinno skutkować jego aktywacją
  - Często przyjmuje się również, że odblokowanie wątku oczekującego o priorytecie równym bieżącemu powinno powodować jego aktywację
    - Tzw. I/O priority boost – chwilowe podbicie priorytetu wątku dotychczas oczekującego
- Można to uzyskać poprzez wywołanie Yield z obsługi przerwania
- CMSIS-RTOS automatycznie wywołuje Yield przy każdym użyciu mechanizmu komunikacji wywołanym z przerwania

Kolejki, semafor, przerwania – demo

- Wątek przetwarzania:
  - pobiera dane z kolejki odbiorczej, przetwarza, zapisuje do kolejki nadawczej
- Wątek 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 powinno być wyłączane PRZED wysłaniem znaku, gdy nie ma następnego znaku do wysłania

Blokowanie przełączania wątków

- Sekcje krytyczne
  - Wejście: taskENTER_CRITICAL(), wyjście: taskEXIT_CRITICAL()
  - Blokują przerwania – ograniczone zastosowanie
- Zatrzymanie szeregowania (przełączania wątków) – bieżący wątek pozostaje aktywny, przerwania są obsługiwane
  - osKernelLock(), osKernelUnlock()

Technika programowania w RTOS

- Wątek powinien oczekiwać wyłącznie na zdarzenia sygnalizowane przez RTOS przy użyciu mechanizmów komunikacji i zarządzania aktywnością
- Wątek nie powinien zawierać pętli aktywnego oczekiwania na zdarzenie nie sygnalizowane za pośrednictwem RTOS, nie zawierającej jakiegokolwiek wywołania synchronizującego (np. osDelay())
  - Dotyczy to również wywoływanych przez wątek synchronicznych funkcji bibliotecznych korzystających z aktywnego oczekiwania
    - W STM32 HAL wszystkie funkcje obsługi modułów peryferyjnych nie zawierające na końcu nazwy _IT lub _DMA są funkcjami synchronicznymi

Interakcja RTOS z asynchronicznymi funkcjami HAL

- Asynchroniczne wywołania HAL wracają po zainicjowaniu operacji, a nie po jej zakończeniu
- Kolejne wywołanie funkcji jest możliwe po zakończeniu operacji
- W celu wykrycia zakończenia operacji można:
  - Podnieść semafor w wywołaniu zwrotnym HAL przy zakończeniu operacji
  - Testować periodycznie stan zakończenia operacji w strukturach danych HAL
- Można również zapewnić odpowiedni odstęp czasowy operacji bez konieczności sprawdzania stanu

Aktywne oczekiwanie w RTOS

- Zmiany stanu i sygnalizacja gotowości powinny korzystać z przerwań i mechanizmów komunikacji RTOS
- Jeśli nie jest to możliwe, programowy test gotowości w RTOS można zrealizować:
  - W procedurze timera wywoływanej periodycznie
    - Pozytywny wynik testu wyzwala mechanizm komunikacji RTOS – semafor, znacznik, kolejkę
  - W pętli wątku zawierającej
    - funkcję oczekiwania osDelay(), osDelayUntil()
    - wywołanie osThreadYield() przy negatywnym wyniku testu

RTOS i STM32 HAL

- Procedury HAL nie są wielobieżne
  - Istniejące w HAL zabezpieczenia wielobieżności są zawodne
- W celu uniknięcia aktywnego oczekiwania w wątkach, do obsługi modułów peryferyjnych należy używać wyłącznie asynchronicznych wersji procedur HAL korzystających z przerwań lub DMA
  - Przed zainicjowaniem operacji wątek czeka z użyciem mechanizmu synchronizacji – zwykle semafora związanego z danym modułem peryferyjnym
  - Należy zdefiniować procedury wywołań zwrotnych HAL zwalniające semafor przy zakończeniu operacji (uwaga na priorytety przerwań!)

Ankieta

https://sube.nobleprog.com//open-tef/36655