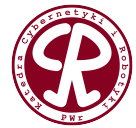
Sterowniki Robotów
Systemy czasu rzeczywistego

Wojciech Domski

Katedra Cybernetyki i Robotyki,
Politechnika Wrocławska

Materiał pomocniczy do notatek z wykładu

Wrocław University
of Science and Technology



Plan prezentacji

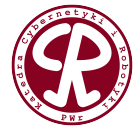
1. System operacyjny
   - Funkcje wspierające implementację OS
   - Hard vs Soft
   - Shadowed Stack Pointer
   - Supervised and unsupervised mode
   - MPU
   - Przełączanie kontekstu
   - Exclusive access
2. FreeRTOS
   - Wprowadzenie
   - Zadania
   - Synchronizacja
   - Kolejki
   - Grupy zdarzeń
   - Liczniki programowe
   - Zarządzanie pamięcią
   - Konfiguracja i monitoring

Wrocław University of Science and Technology

System operacyjny (1/2)

Czym jest system operacyjny?

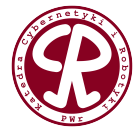
Wrocław University
of Science and Technology



System operacyjny (2/2)

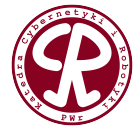
Jest to konkretny rodzaj oprogramowania działający w trybie jądra (tryb uprzywilejowany). Systemy operacyjne zapewniają zestaw interfejsów API do korzystania z zasobów oraz zarządzania tymi zasobami sprzętowymi.

Wrocław University
of Science and Technology



Outline

Wrocław University of Science and Technology

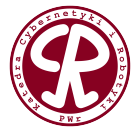


Cortex-M3/M4 core (1/4)

Rdzenie Cortex-M3/M4 wspierają następujące funkcje w celu ułatwienia implementacji systemu wbudowanego.

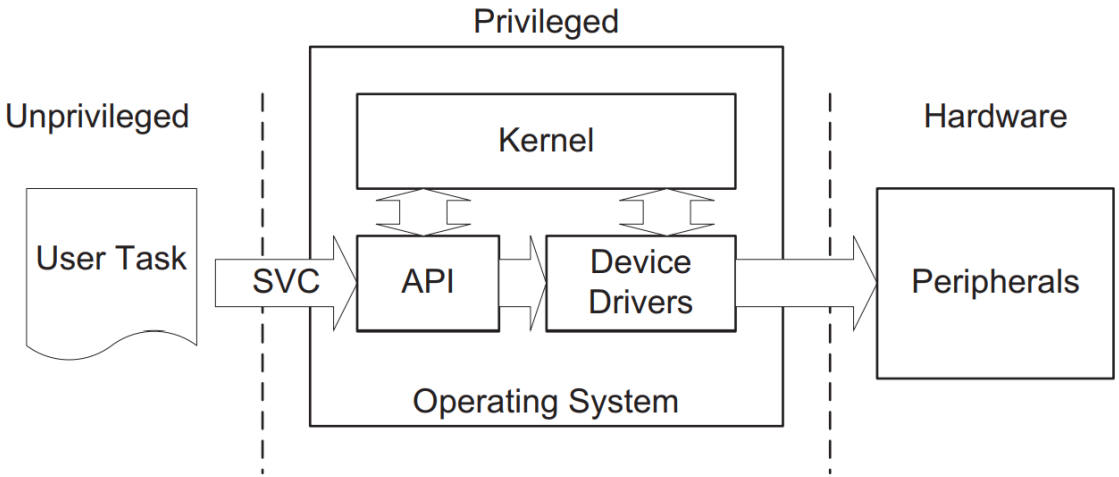
- **Shadowed stack pointer**
  The Main Stack Pointer (MSP) wykorzystywany jest przez system operacyjny oraz podczas obsługi przerwań. Processor Stack Pointer (PSP) wykorzystywany jest przez zadania (z ang. *tasks*).
- **Licznik SysTick**
  Podstawowy licznik występujący w większości mikrokontrolerów. Pozwala on na implementację planisty, który odpowiedzialny jest za przełączanie zadań.

Wrocław University of Science and Technology



Cortex-M3/M4 core (2/4)

- **Supervisor Call (SVC) exception**
  Jest przerwaniem precyzyjnym, wykonywanym zaraz po wykonaniu instrukcji SVC. Przerwanie SVC posiada programowalny poziom przerwania. Pozwala na implementację mechanizmu dostępu do warstwy sprzętowej w systemach o dużej niezawodności.

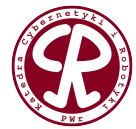


1

Cortex-M3/M4 core (3/4)

- **Pendable Service Call (PendSV) exception**
  Przerwania posiada programowalny poziom wyzwolenia. Wyzwalanie następuje poprzez zapis do Interrupt Control and State Register (ICSR). Przerwanie jest typu nieprecyzyjnego. Wykorzystywane do przełączania zadań, ponieważ pozwala na „dokończenie” wykonywania zadań.
- **Unprivileged execution level**
  Pozwala na implementację podstawowego mechanizmu bezpieczeństwa związanego z ograniczeniem dostępu dla niektórych zadań. Tryb uprzywilejowany i ograniczony mogą być wykorzystywane wraz z układem Memory Protection Unit (MPU).

Wrocław University of Science and Technology



Cortex-M3/M4 core (4/4)

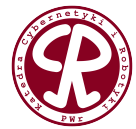
- **Exclusive accesses**
  Pozwala na kontrolę dostępu do zasobów. Mechanizm ten umożliwia implementację takich elementów systemu jak mutexy i semafory.

Wrocław University of Science and Technology

[1] J. Yiu, The definitive quide to ARM Cortex-M3 and Cortex-M4 processors

Outline

Wrocław University of Science and Technology

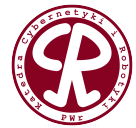


Rodzaje systemu czasu rzeczywistego

Systemy czasu rzeczywistego dzielą się na:

- miękkie, (z ang. *soft real-time operating system*),
- twarde, (z ang. *hard real-time operating system*).

Wrocław University
of Science and Technology



Hard RTOS

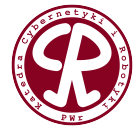
Twardy system czasu rzeczywistego **musi** spełniać surowe ograniczenia czasowe.
W tym celu należy przeprowadzić analizę całego systemu, aby oszacować najdłuższy czas potrzebny na wykonanie zadania.
Tego typu OS znajduje zastosowanie w przemyśle, kontroli procesów, systemach lotniczych, wojskowych i wszędzie tam, gdzie czas gra kluczową rolę.
W tego typu systemach akcja musi być wykonana w ściśle określonych ramach czasowych.
Jeżeli warunek ten zostanie niespełniony może on powodować efekt kaskadowy prowadzący do awarii całego systemu. Z tego względu twardy RTOS powinien posiadać mechanizmy, które pozwalają na spełnienie tych wymagać.

Wrocław University
of Science and Technology

Soft RTOS

W odróżnieniu od twardego systemu czasu rzeczywistego miękki system czasu rzeczywistego dopuszcza okazjonalne przekroczenie reżimu czasu rzeczywistego. Przekroczenie ram czasowych jest wciąż niepożądane, ale nie powoduje trwałego „uszkodzenia” systemu.
Tego typu OS wykorzystywany jest głownie w branży rozrywkowej. Można go spotkać w urządzeniach mobilnych, odtwarzaczach video, czy odtwarzaczach muzycznych.

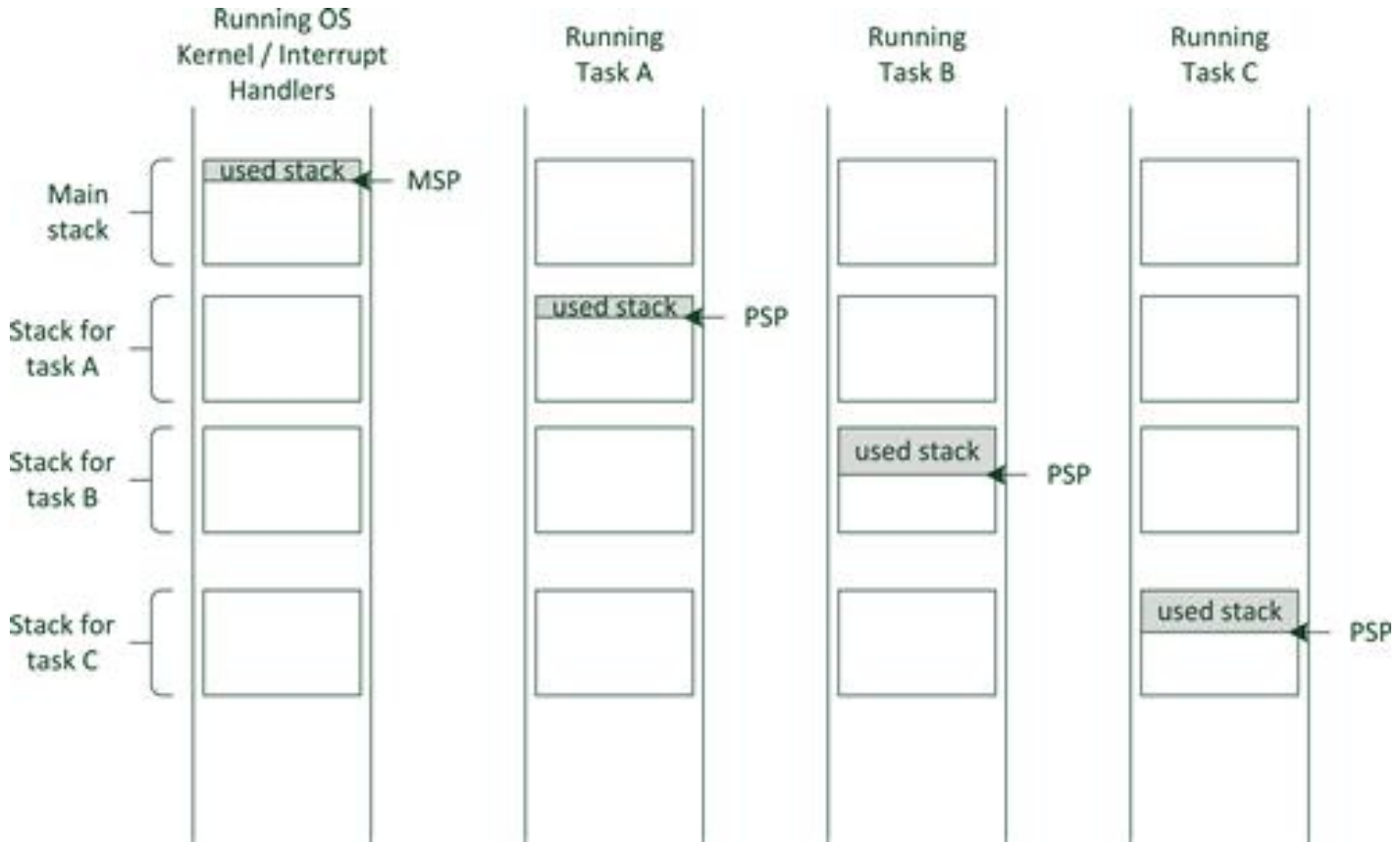
Wrocław University of Science and Technology



Outline

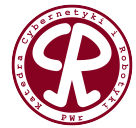
Wrocław University of Science and Technology

Main Stack Pointer (MSP), a Processor Stack Pointer (PSP) (1/4)



1

Wrocław University of Science and Technology

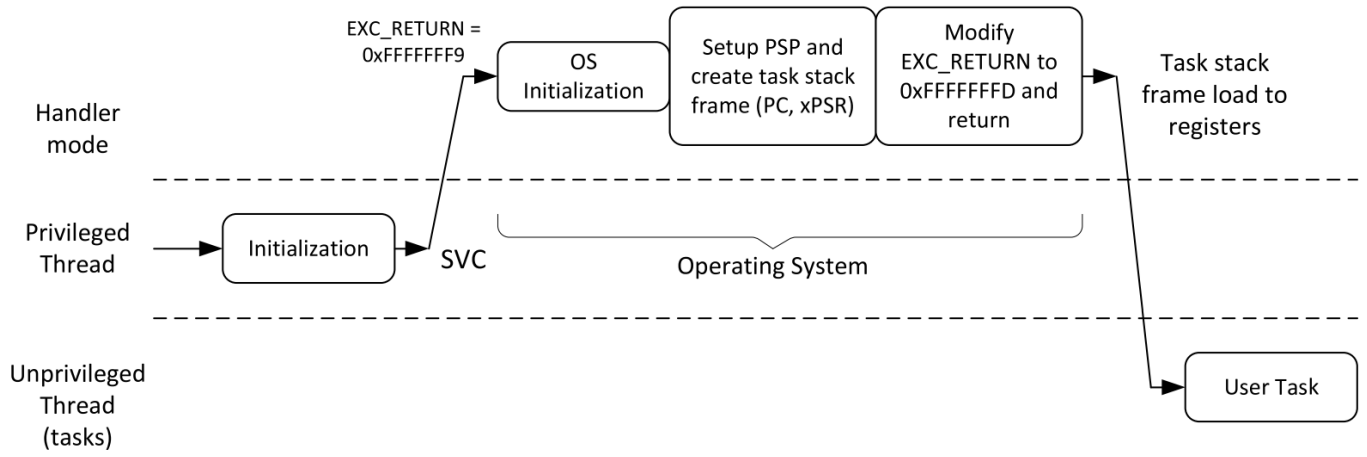


Main Stack Pointer (MSP), a Processor Stack Pointer (PSP) (2/4)

- Ewentualne uszkodzenie stosu w zadaniu nie powinno wpłynąć na działanie OS i innych zadań.
- Pamięć alokowana dla stosu zadania musi jedynie pomieścić elementy odkładane na stos przez zadanie oraz jeden poziom dla odkładanej ramki stosu. Przestrzeń potrzebna na obsługę (zagnieżdżonych) przerwań nie wpływa na stos zadania.
- Ułatwia implementację OS na rdzeniach Cortex-M3/M4.
- Memory Protection Unit (MPU) może zostać wykorzystane do określenia konkretnego regionu na alokację stosu dla zadania.

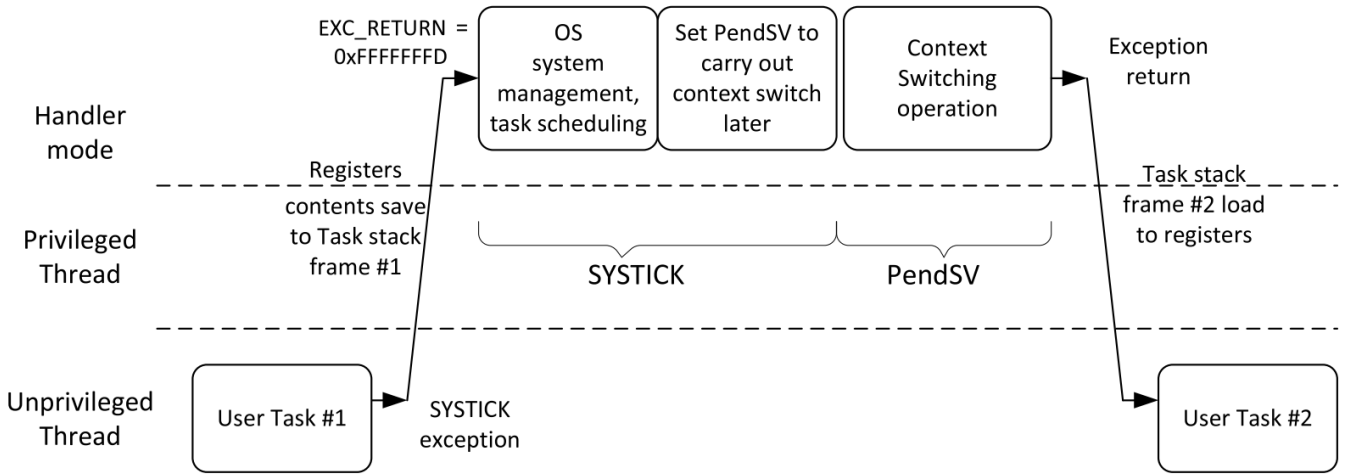
Wrocław University of Science and Technology

Main Stack Pointer (MSP), a Processor Stack Pointer (PSP) (3/4)



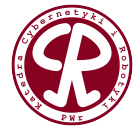
1

Main Stack Pointer (MSP), a Processor Stack Pointer (PSP) (4/4)



[1]

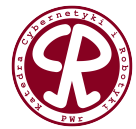
Wrocław University
of Science and Technology



[1] J. Yiu, The definitive guide to ARM Cortex-M3 and Cortex-M4 processors

Outline

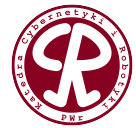
Wrocław University of Science and Technology



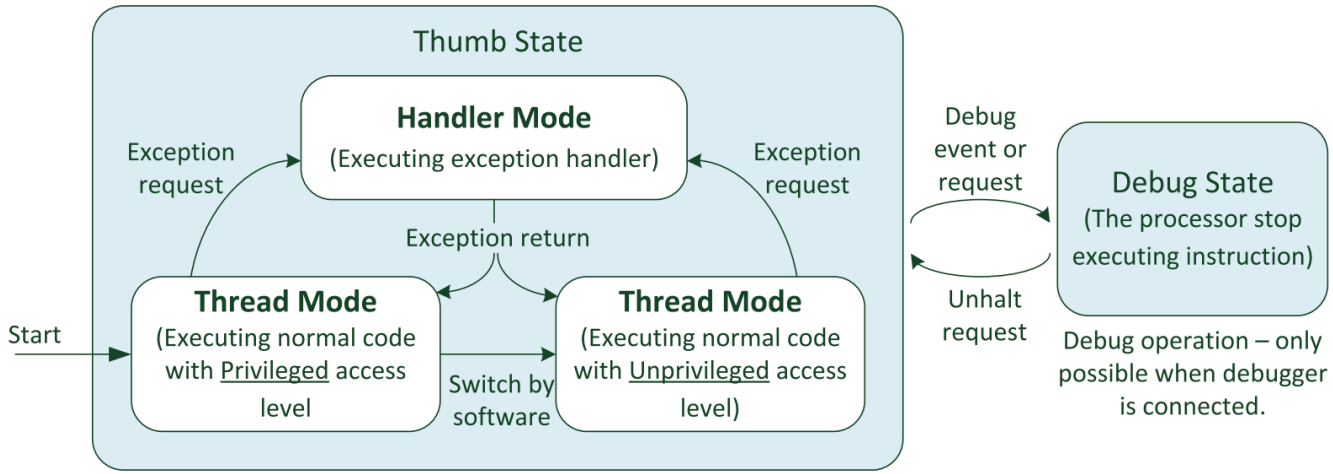
Handler mode and Thread mode (1/4)

Nieuprzywilejowany tryb wykonywania umożliwia implementację podstawowych mechanizmów bezpieczeństwa, które mogą ograniczyć prawa dostępu dla wybranych zadań. Rozdzielenie trybów uprzywilejowanego i ograniczonego w połączeniu z MPU pozwala na zwiększenie odporności systemu wbudowanego.

Wrocław University of Science and Technology



Handler mode and Thread mode (2/4)



1

Handler mode and Thread mode (3/4)

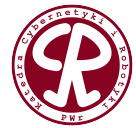
Stany operacyjne:

- Debug state – procesor jest zatrzymany, instrukcje nie są wykonywane,
- Thumb state – procesor wykonuje kod programu.

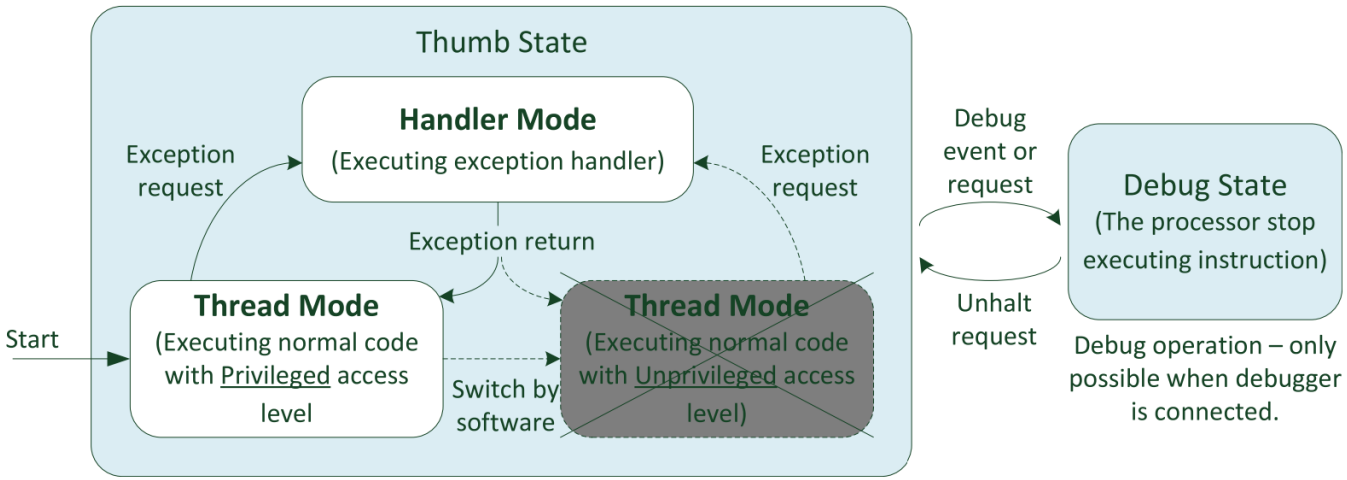
Tryby operacyjne:

- Handler mode – procesor obsługuje przerwanie, w tym stanie procesor jest zawsze w stanie uprzywilejowanym,
- Thread mode – wykonywanie „normalnego” kodu aplikacji. Procesor jest w trybie uprzywilejowanym lub ograniczonym.

Wrocław University of Science and Technology



Handler mode and Thread mode (4/4)



[1]

[1] J. Yiu, The definitive guide to ARM Cortex-M3 and Cortex-M4 processors

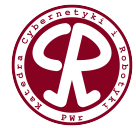
Outline

Wrocław University of Science and Technology

Memory Protection Unit (1/5)

MPU to programowalny układ, który może być użyty do zdefiniowania uprawnień dostępu do pamięci (pełny dostęp lub dostęp w trybie uprzywilejowanym), a także atrybutów pamięci (buforowany, przechowywalny) dla różnych obszarów pamięci.
MPU na rdzeniach Cortex-M3/M4 wspiera do ośmiu programowalnych regionów pamięci. Każdy z regionów określany jest za pomocą adresu początku pamięci, rozmiaru i dodatkowych ustawień.
Dodatkowo możliwe jest ustawienie funkcji tła. Region z tą cechą posiada te same atrybuty co domyślna mapa pamięci, ale jest dostępny jedynie w trybie uprzywilejowanym.

Wrocław University of Science and Technology

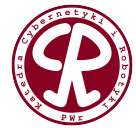


Memory Protection Unit (2/5)

MPU może być wykorzystane w systemach wbudowanych do zapewnienia odporności pamięci. Zwiększenie bezpieczeństwa systemu możliwe jest dzięki:

- Zapobieganiu uszkodzeniu stosu przez współbierzne aplikacje poprzez separację pamięci.
- Zapobieganie użycia wybranych peryferiów, które są krytyczne dla działania systemu operacyjnego, w trybie nieuprzywilejowanym.
- Oznaczenie pamięci RAM jako niewykonywalnej (z ang. *non-executable*) w celu uniemożliwienia wstrzykiwania kodu.

Wrocław University of Science and Technology



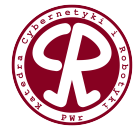
Memory Protection Unit (3/5)

MPU Control Register:

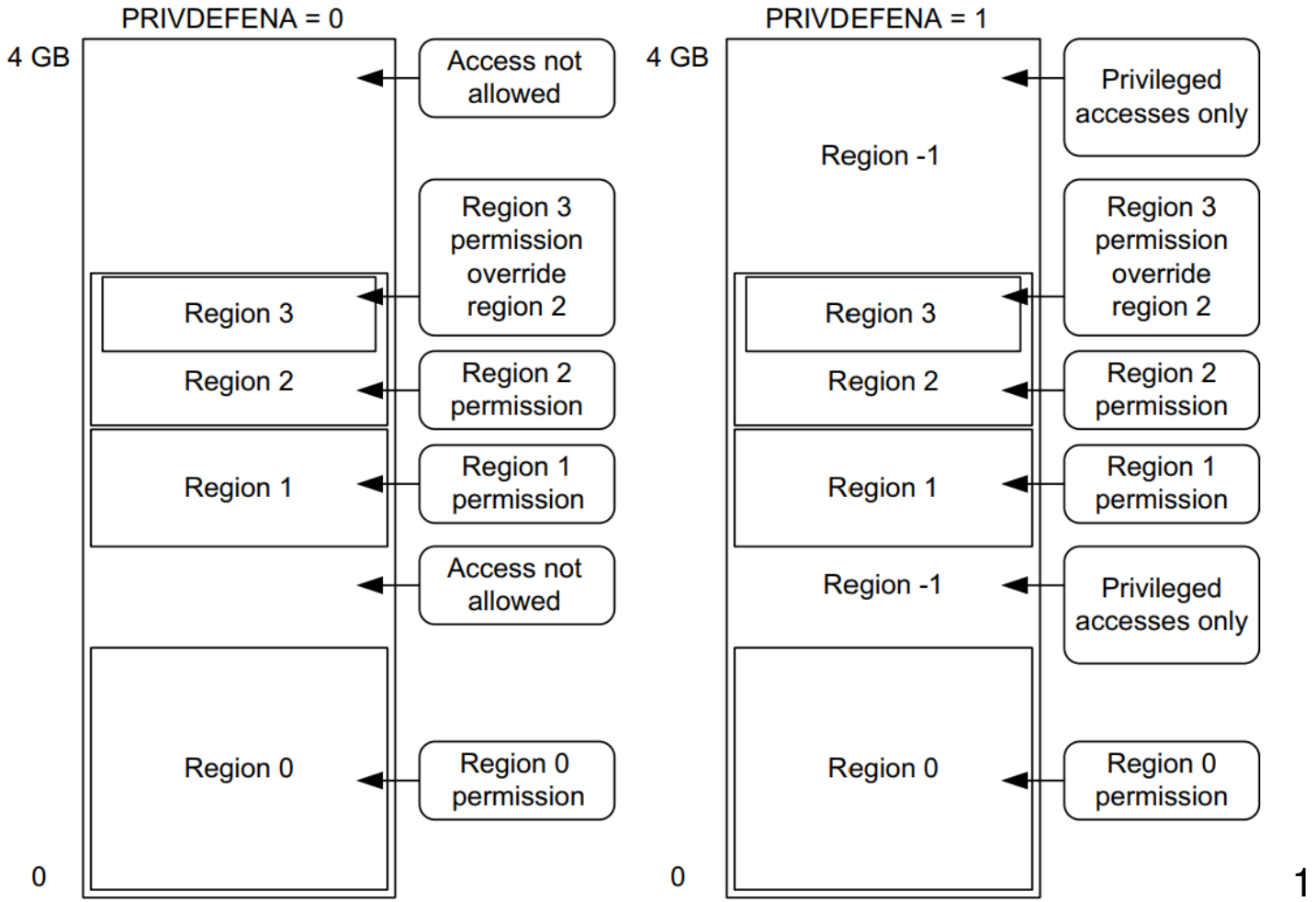
- PRIVDEFENA – odpowiada za domyślną mapę pamięci dla trybu uprzywilejowanego, w przypadku aktywacji domyślna mapa pamięci będzie wykorzystana jako region tła,
- HFNMIENA – MPU aktywne podczas obsługi HardFault,
- ENABLE – aktywuje MPU.

Regiony:
Ustalenie indywidualnych regionów w pamięci poprzez określenie ich adresu, rozmiaru oraz numeru.

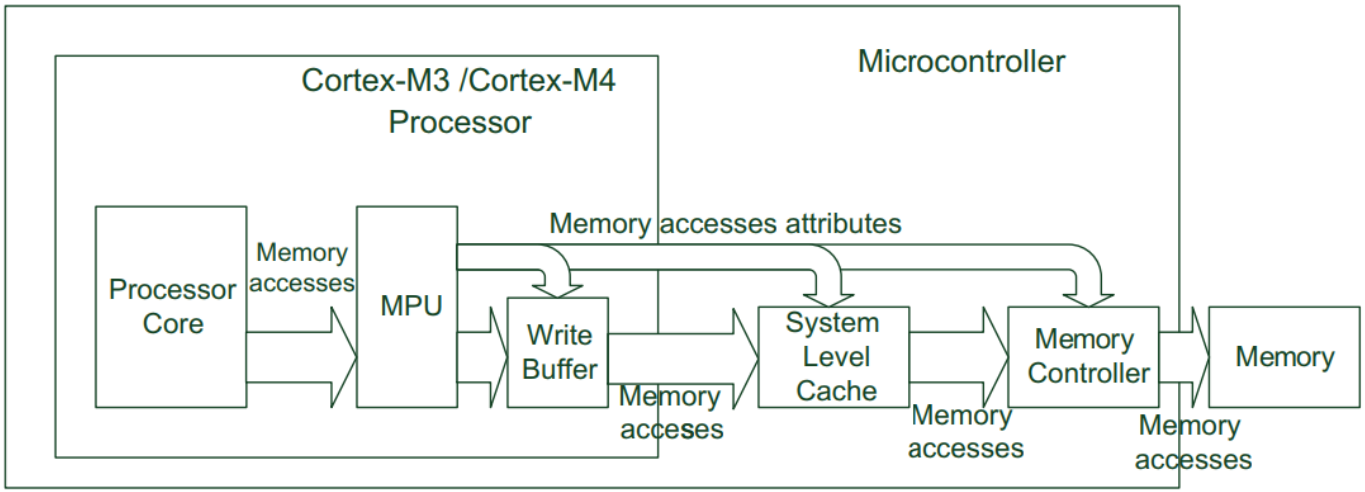
Wrocław University of Science and Technology



Memory Protection Unit (4/5)



Memory Protection Unit (5/5)



[1]

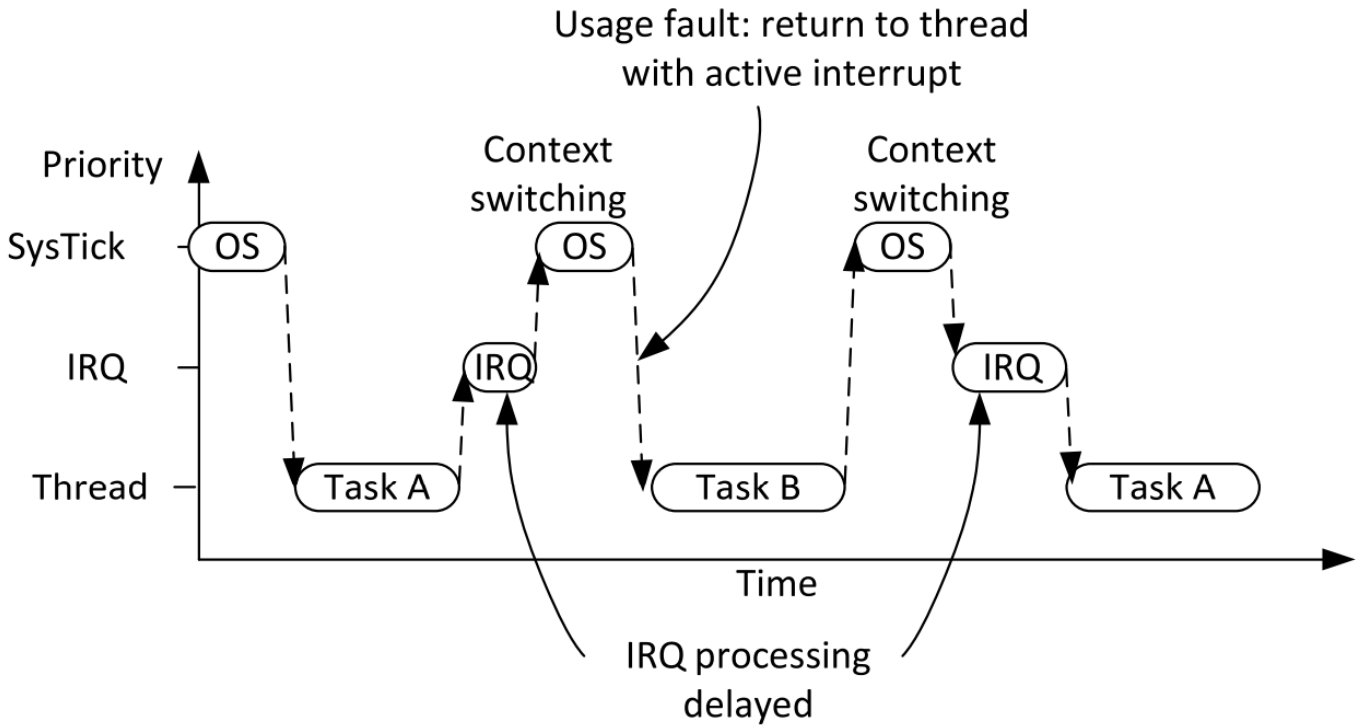
[1] J. Yiu, The definitive guide to ARM Cortex-M3 and Cortex-M4 processors

Outline

Wrocław University of Science and Technology

Przełączanie kontekstu (1/6)

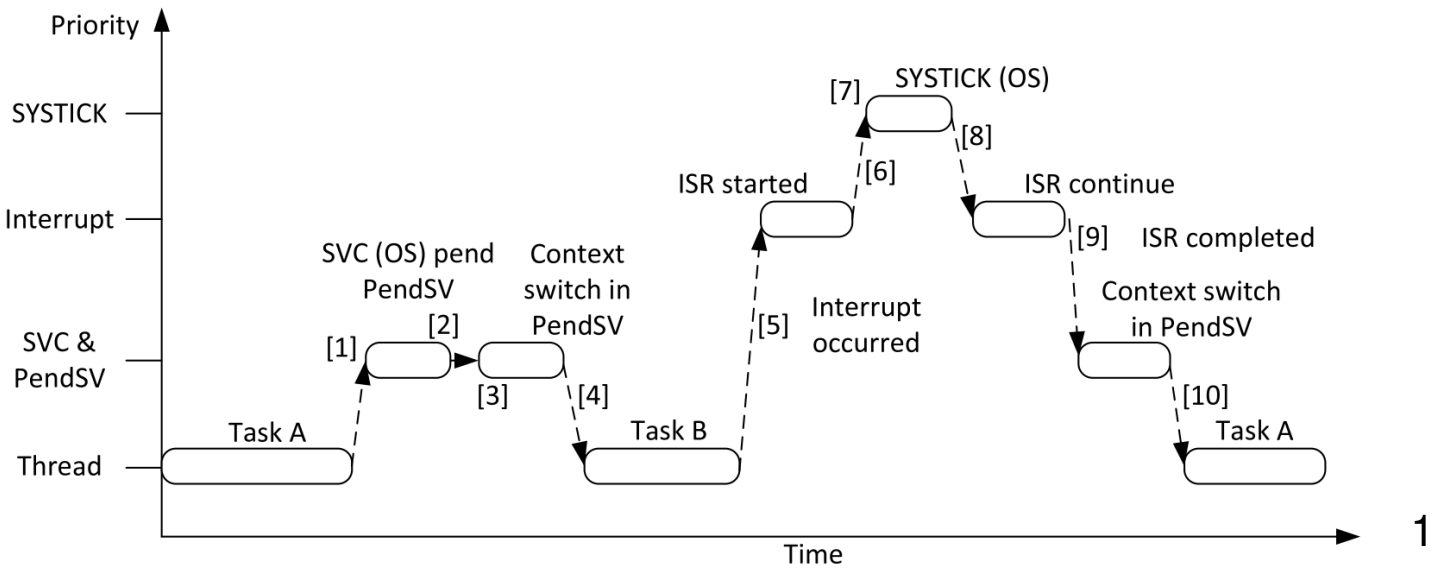
Przełączanie kontekstu w trakcie wykonywania przerwania nie powinno być wykonywane, ponieważ obsługa przerwania może zostać opóźniona.



1

Przełączanie kontekstu (2/6)

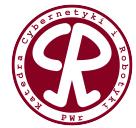
Przełączanie kontekstu z użyciem przerwania PendSV



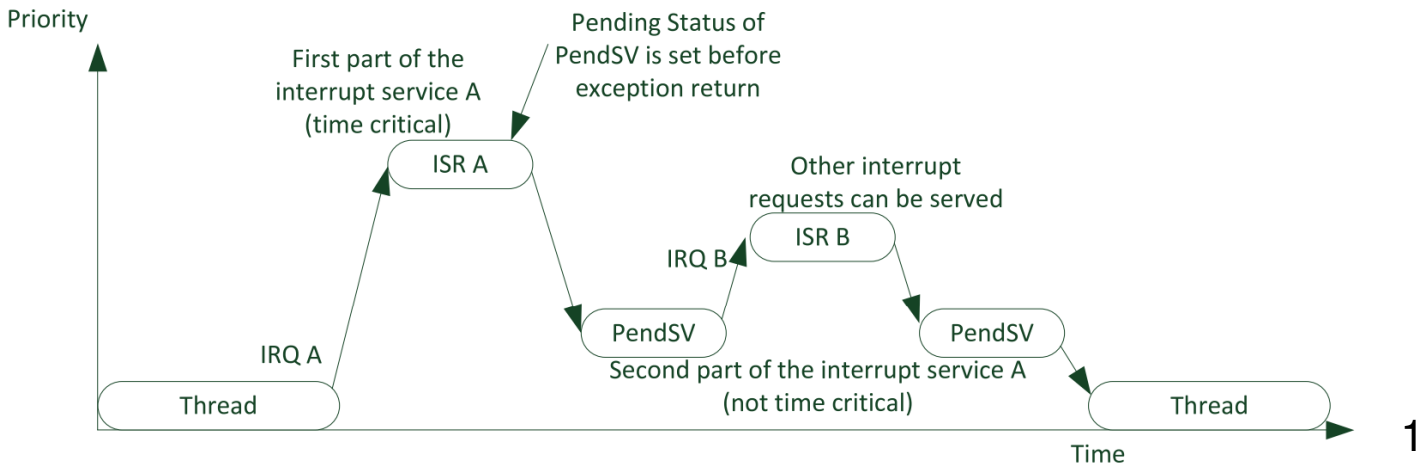
Przełączanie kontekstu (3/6)

- PendSV może zostać wykorzystane do przełączania zadań bez obecności systemu operacyjnego. Jest ono przydane w sytuacji, gdy jest potrzebne priorytetowe wykonanie przerwania.
- Pierwsza część obsługi przerwania jest krytyczna czasowo. Po zakończeniu obsługi przerwania ustawiane jest PendSV, które kontynuuje obliczenia. W przypadku, gdy wystąpi zdarzenie o wyższym priorytecie (od PendSV) obliczenia są przerywane.
- Takie podejście pozawala na oddanie czasu procesora przez podział priorytetów przerwań z wykorzystaniem PendSV.

Wrocław University of Science and Technology

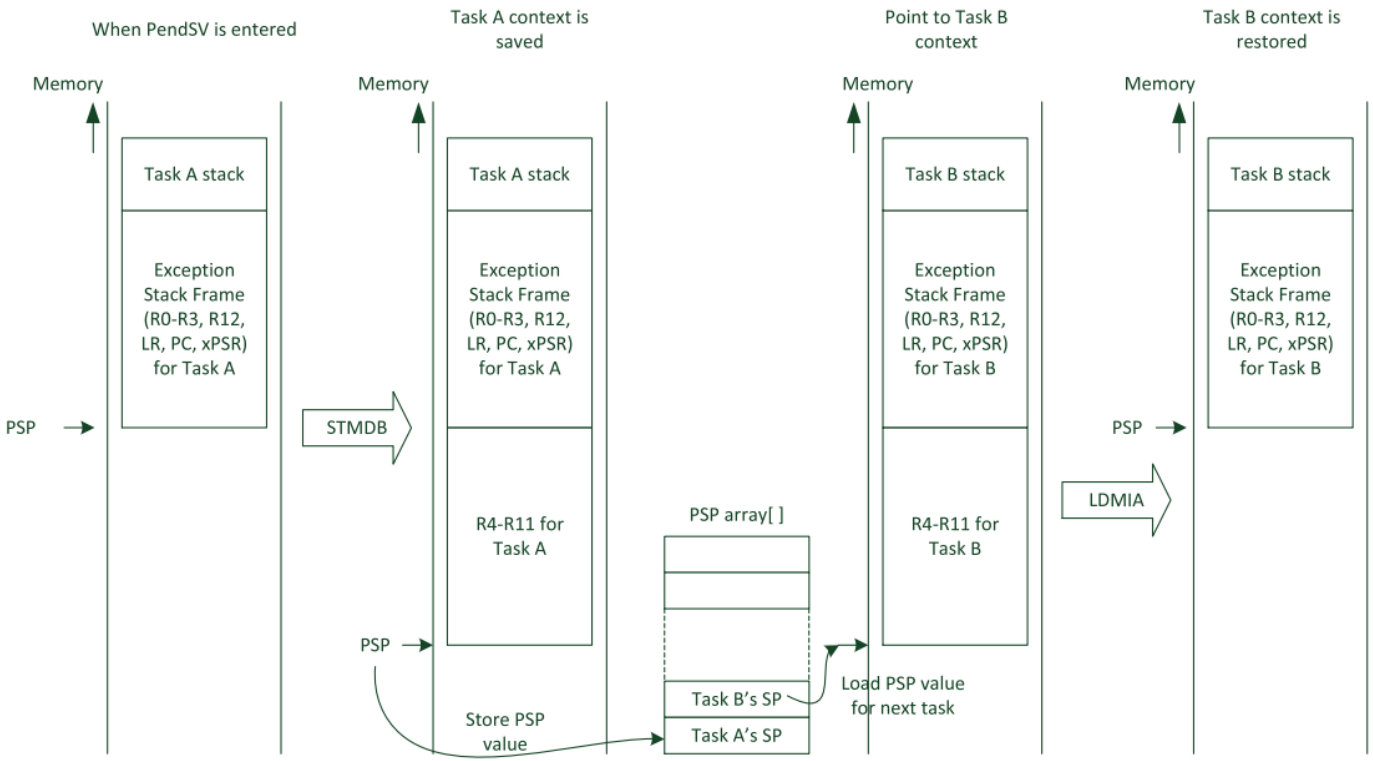


Przełączanie kontekstu (4/6)



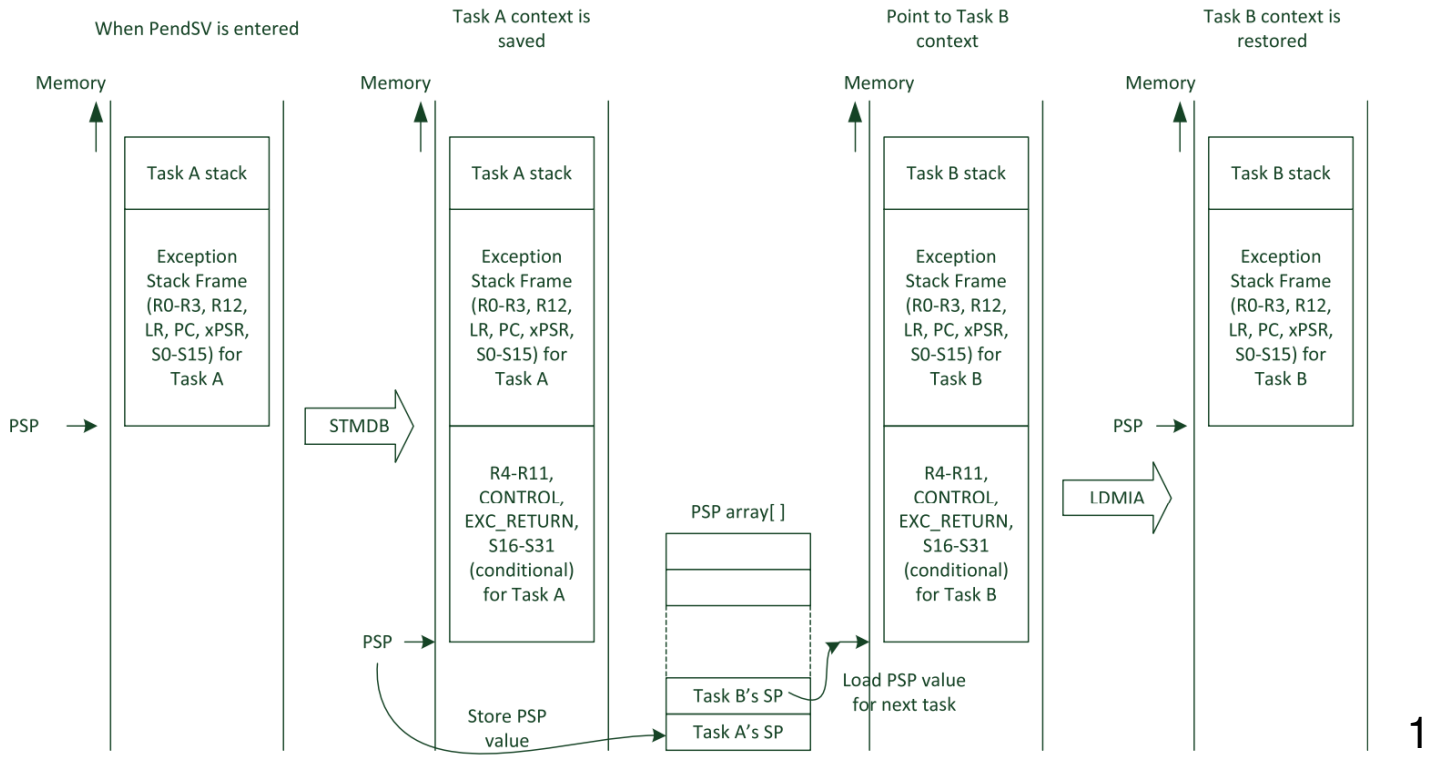
1

Przełączanie kontekstu (5/6)



1

Przełączanie kontekstu (6/6)



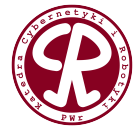
[1]

Wrocław University of Science and Technology

[1] J. Yiu, The definitive guide to ARM Cortex-M3 and Cortex-M4 processors

Outline

Wrocław University of Science and Technology

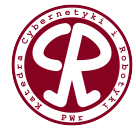


Exclusive accesses (1/5)

Semafory są jednym ze sposobów, aby zarządzać dostępem do zasobów. Semafor może być reprezentowany jako licznik, który przechowuje liczbę dostępnych zasobów.
Aby uzyskać dostęp do zasobu należy:

1. Odczytać obecną liczbę dostępnych zasobów i jeśli wartość ta jest większa niż zero przejdź do następnego punktu.
2. Zmniejsz wartość o jeden.
3. Zapisz pomniejszoną wartość.

Wrocław University of Science and Technology

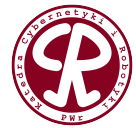


Exclusive accesses (2/5)

Problem powstaje w momencie, gdy mamy do czynienia z procesami współbieżnymi, a w szczególności przełączeniem kontekstu, które może wystąpić w dowolnym punkcie. Zła wartość jest zapisywana do semafora, która odzwierciedla błędny stan systemu.

Specjalnym przypadkiem pojedynczego semafora jest muteks. Bariera typu wzajemne wykluczenie.

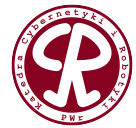
Wrocław University of Science and Technology



Exclusive accesses (3/5)

Rdzenie Cortex-M3/M4 wspiera mechanizm typu dostęp na wyłączność, dzięki czemu implementacja barier jest ułatwiona.
Wartośc semafora jest odczytywana i zapisywana dzięki operacjom chronionego odczytu i zapisu.
Jeśli podczas chronionego zapisu dojdzie do sytuacji, w której dostęp na wyłączność nie może zostać zagwarantowany operacja zapisu nie dojdzie do skutku. Procesor ponowi wówczas sekwencję.

Wrocław University of Science and Technology

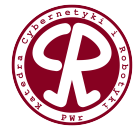


Exclusive accesses (4/5)

W rdzeniu występuje dedykowana jednostka lokalnego monitora (z ang. *local monitor*), która czuwa nad operacjami odczytu/zapisu na wyłączność.
Można wyróżnić trzy instrukcje, które pozwalają na implementację semafora:

- STREX – zapisz dane,
- CLREX – przełącz monitor w stan otwartego dostępu, (z ang. *Open Access*),
- LDREX – przełącz monitor w stan chronionego dostępu, (z ang. *Exclusive Access*).

Wrocław University of Science and Technology

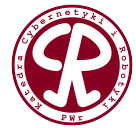


Exclusive accesses (5/5)

Zapis na wyłączność może (instrukcja STREX) może się nie powieść jeśli, któraś z sytuacji miała miejsce:

- CLREX została wykonana przełączając monitor w stan Open Access.
- Nastąpiło przełączenie kontekstu (np. podczas obsługi przerwania),
- LDREX nie zostało wykonane wcześniej,
- Procesor otrzymał status błędu związany z dostępem na wyłączność od zewnętrznego urządzenia za pośrednictwem kanału bocznego na interfejsie magistrali.

Wrocław University of Science and Technology



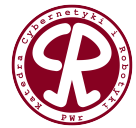
Outline

Wrocław University of Science and Technology

Wprowadzenie

FreeRTOS implementuje jądro czasu rzeczywistego, na którym można zbudować system wbudowany spełniający wymagania **twardego** systemu operacyjnego czasu rzeczywistego. Pozwala na organizację aplikacji (zadań) w zbiór niezależnych wątków. Na procesorze, który posiada tylko jeden rdzeń wyłącznie jeden wątek w danej chwili może być uruchomiony. To jądro decyduje, który wątek powinien być wykonywany w danej chwili. Dodatkowo podczas planowania zadań brany jest pod uwagę priorytet z jakim dane zadanie ma być wykonywane.

Wrocław University of Science and Technology



Właściwości (1/7)

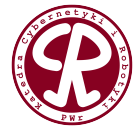
- **Uniezależnienie od czasu**
  To jądro systemu odpowiedzialne jest za czasowe wykonywanie zadań i dostarcza interfejs pozwalający na zarządzanie czasem w ramach zadania, jak np. opóźnienia. Taka dekompozycja pozwala na lepsze zarządzanie aplikacją, przez co staje się ona prostsza, a zarazem pozwala na ograniczenie rozmiaru kodu wynikowego.
- **Utrzymanie i rozszerzanie aplikacji**
  Odseparowanie zależności czasowych skutkuje mniejszą liczbą zależności pomiędzy modułami, a ww rezultacie pozwala to oprogramowaniu na rozwój w kontrolowany i przewidywany sposób. Ponadto, jądro odpowiedzialne jest za dbanie o zależności czasowe, a zatem wydajność zadania jest mniej podatna na zmiany sprzętowe.

Wrocław University of Science and Technology

Właściwości (2/7)

- **Modułowość**
  Zadania realizowane są w niezależnych modułach, a zatem powinny być dobrze zdefiniowane i odpowiadać, za wcześniej zidentyfikowane operacje.
- **Rozwój oprogramowania w zespole**
  Każde z zadań powinno mieć dobrze zdefiniowane interfejsy. Dzięki temu możliwe jest odseparowanie od siebie modułów w taki sposób, aby grupy inżynierów mogły pracować w sposób niezależny.

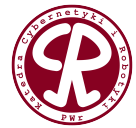
Wrocław University of Science and Technology



Właściwości (3/7)

- **Łatwiejsze testowanie**
  Przez odseparowanie od siebie zadań możliwe jest łatwiejsze testowanie aplikacji.Pisanie testów jednostkowych jest ważnym aspektem inżynierii oprogramowania i pozwala na dostarczenie bardziej niezawodnego oprogramowania.
- **Ponowne wykorzystanie kodu źródłowego**
  Rozwój oprogramowania nastawionego na modułowe programowanie pozwala na znaczne ograniczenie czasu poświęcanego na rozwój aplikacji. Tworzenie niezależnych modułów pozwala na ich łatwiejsze wykorzystanie w nowych projektach.

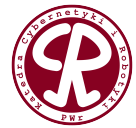
Wrocław University of Science and Technology



Właściwości (4/7)

- **Zwiększona wydajność**
  Wykorzystanie jądra systemu pozwala na tworzenie oprogramowania reagującego na zdarzenia (z ang. *event driven*). Pozwala to na zaoszczędzenie czasu poprzez brak manualnego sprawdzania, czy dane zdarzenie nastąpiło tzw. *pooling*. Kod jest wykonywany jedynie wtedy, gdy zdarzenie wystąpiło.

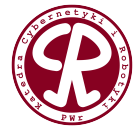
Wrocław University of Science and Technology



Właściwości (5/7)

- **Czas bezczynności**
  Zadanie bezczynności jest tworzone w sposób automatyczny, gdy planista zostaje uruchomiony. Zadanie to jest wykonywane w momencie, gdy inne zwróciły kontrolę do systemu. Co więcej, zadanie bezczynności może zostać wykorzystane w celach debugowania takich jak: pomiar możliwości obliczeniowych systemu, przeprowadzanie diagnostyki w tle, czy przeniesienie procesora w tzw. stan uśpienia.

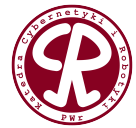
Wrocław University of Science and Technology



Właściwości (6/7)

- **Zarządzanie energią**
  Zysk wydajności osiągnięty przez wykorzystanie RTOS pozwala na to, aby procesor przeszedł w stan uśpienia. Zużycie energii może zostać drastycznie obniżone przez umieszczenie rdzenia i peryferiów w stanie niskiego poboru prądu za każdym razem, gdy proces bezczynności jest aktywny. FreeRTOS dostarcza specjalnego trybu bez taktów. Wykorzystanie tego trybu pozwala na przeniesienie procesora do trybu niskiego zużycia mocy. Dzięki niemu procesor może pozostać dłużej w trybie oszczędzania energii.

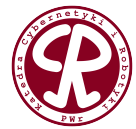
Wrocław University of Science and Technology



Właściwości (7/7)

- **Elastyczne zarządzanie przerwaniami**
  Obsługa przerwań może być skrócona do minimum przez delegowanie obsługi przerwania do zadania lub zadania tworzonego w locie.
  Sposób ten pozwala na kolejkowanie zadań, które mają niższy priorytet i mogą zostać obsłużone z opóźnieniem.
- **Zróżnicowane wymagania dotyczące przetwarzania**
  Dzięki prostocie implementacji zadań i ich obsługi możliwe jest połączenie trzech różnych sposobów przetwarzania: okresowego, ciągłego, a także zdarzeniowego. Dodatkowo, oba twarde oraz miękkie wymagania systemu czasu rzeczywistego mogą być spełnione przez wykorzystanie priorytetów dla zadań oraz przerwań.

Wrocław University
of Science and Technology

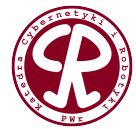


Dedykowane API dla przerwań

Warto podkreślić, że nie każda dostępna funkcja frameworka FreeRTOS może być wykonana wewnątrz obsługi przerwania.

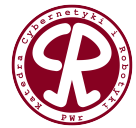
Większość FreeRTOSowego API ma dwie implementacje. Pierwsza grupa jest dedykowana do normalnej obsługi przerwań, tzn. w trybie ograniczonym. Natomiast druga grupa przerwań została zaimplementowana specjalnie z myślą o wywołaniach w przerwaniach (z ang. *Interrupt Service Routine, ISR*).
Dwie implementacje można od siebie odróżnić, ponieważ ta druga ma dodaną frazę `FromISR` w nazwie funkcji.

Wrocław University of Science and Technology

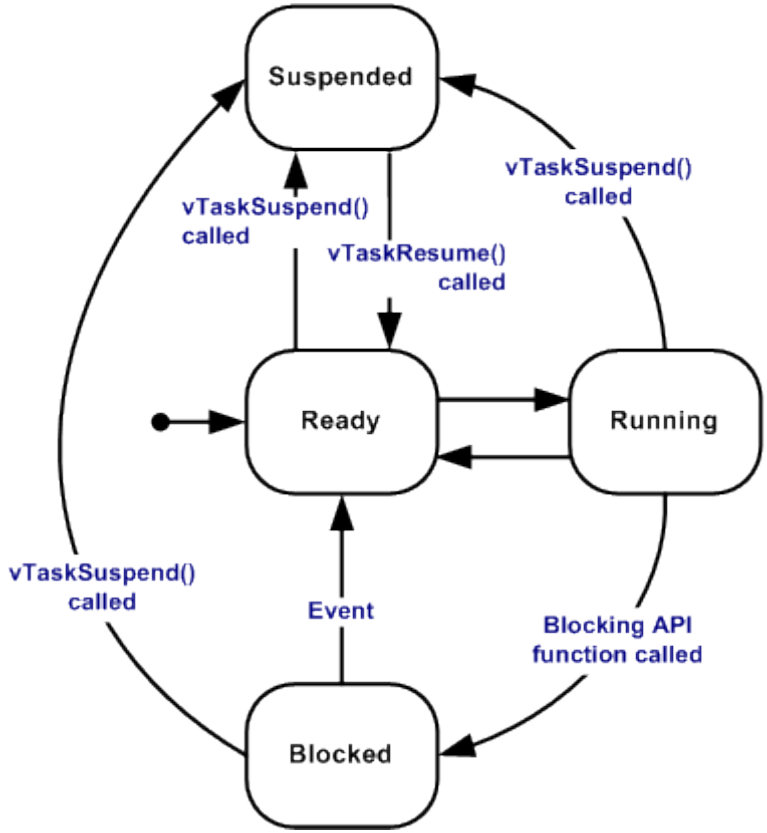


Outline

Wrocław University of Science and Technology



Możliwe stany zadania



[2]

[2] freertos.org

Tworzenie zadań

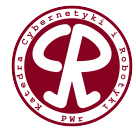
```
1  BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
2  const char * const pcName,
3  uint16_t usStackDepth,
4  void *pvParameters,
5  UBaseType_t uxPriority,
6  TaskHandle_t *pxCreatedTask );
```

```
1  void Task( void *pvParameters ) {
2    for(;;) {
3    }
4  }
```

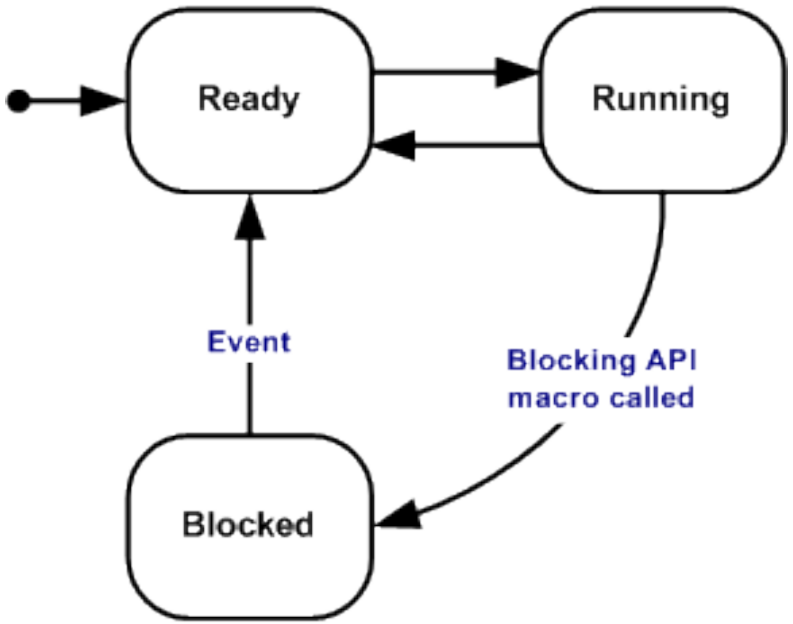
Po utworzenie wszystkich zadań można uruchomić planistę

```
1  vTaskStartScheduler();
```

Wrocław University of Science and Technology



Co-routines (1/5)

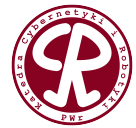


3

Co-routines (2/5)

```
1  void vACoRoutineFunction( CoRoutineHandle_t xHandle, UBaseType_t
       uxIndex )
2  {
3    crSTART( xHandle );
4
5    for( ;; )
6    {
7      -- Co-routine application code here. --
8    }
9
10   crEND();
11 }
```

Wrocław University of Science and Technology

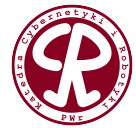


Co-routines (3/5)

Tworzenie funkcji:

```
1 BaseType_t xCoRoutineCreate( crCOROUTINE_CODE pxCoRoutineCode,
2   UBaseType_t uxPriority,
3   UBaseType_t uxIndex);
```

Wrocław University of Science and Technology

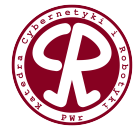


Co-routines (4/5)

Wykonywanie funkcji odbywa się przez powtarzające się wywołania `vCoRoutineSchedule`.

```
1 void vApplicationIdleHook( void )
2 {
3   vCoRoutineSchedule( void );
4 }
```

Wrocław University of Science and Technology



Co-routines (5/5)

Jeżeli zadanie bezczynności nie wykonuje żadnych innych zadań efektywniejszym sposobem jest wykorzystanie go do przełączania funkcji:

```
1 void vApplicationIdleHook( void )
2 {
3   for( ;; )
4   {
5     vCoRoutineSchedule( void );
6   }
7 }
```

Wrocław University of Science and Technology

[3] freertos.org

Opóźnienia (1/3)

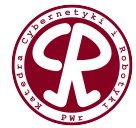
Wykonywane zadanie może zostać opóźnione na dwa sposoby:

- opóźnienie bezwzględne,
- opóźnienie względne.

```
1 void vTaskDelay( TickType_t xTicksToDelay );
```

Opóźnienie bezwzględne jest liczone od momentu jego wywołania. Wartość opóźnienia jest liczona w taktach. Aby przeliczyć wartość opóźnienia wyrażoną w milisekundach należy wykorzystać makro `portTICK_PERIOD_MS`.

Wrocław University of Science and Technology



Opóźnienia (2/3)

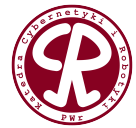
Implementacja zadań okresowych wymaga względnego podejścia do opóźnień.

```
1 void vTaskDelayUntil( TickType_t * pxPreviousWakeTime,
2     TickType_t xTimeIncrement );
```

`vTaskDelayUntil` dokona względnego opóźnienia, tak aby zachować pożądany okres. W odróżnieniu do `vTaskDelay`, `vTaskDelayUntil` przyjmuje dodatkowy argument, który pozwala określić punkt w czasie względem, którego liczone jest opóźnienie.

```
1 TickType_t xLastWakeTime;
2 xLastWakeTime = xTaskGetTickCount();
```

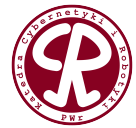
Wrocław University of Science and Technology



Opóźnienia (3/3)

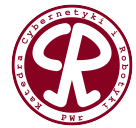
Bezwzględnie należy unikać opóźnień w typu aktywnego opóźnienia (z ang. *busy wait*). Opóźnienia tego typu nie informują planisty o fakcie, że zadanie jest w trybie bezczynności. Może to prowadzić do „głodzenia” zadań.

Wrocław University of Science and Technology



Outline

Wrocław University of Science and Technology



Muteksy (1/4)

Muteks jest szczególnym przypadkiem semafora (semafor jednostkowy). Utworzyć go można za pomocą wywołania:

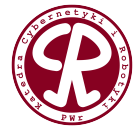
```
1 SemaphoreHandle_t xSemaphoreCreateMutex( void );
```

Zabieranie i zwalnianie zasobu przebiega przy użyciu funkcji:

```
1 BaseType_t xSemaphoreTake( SemaphoreHandle_t xSemaphore,
2     TickType_t xTicksToWait );
```

```
1 BaseType_t xSemaphoreGive( SemaphoreHandle_t xSemaphore );
```

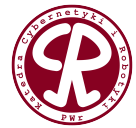
Wrocław University of Science and Technology



Muteksy (2/4)

Obszar kodu źródłowego, który znajduje się pomiędzy zajęciem, a zwolnieniem muteksa nazywa się **sekcją krytyczną** (z ang. *critical section*). Sekcje krytyczne powinny być tak projektowane, aby dwie różne sekcje nie nachodziły na siebie oraz były możliwie krótkie.

Wrocław University of Science and Technology



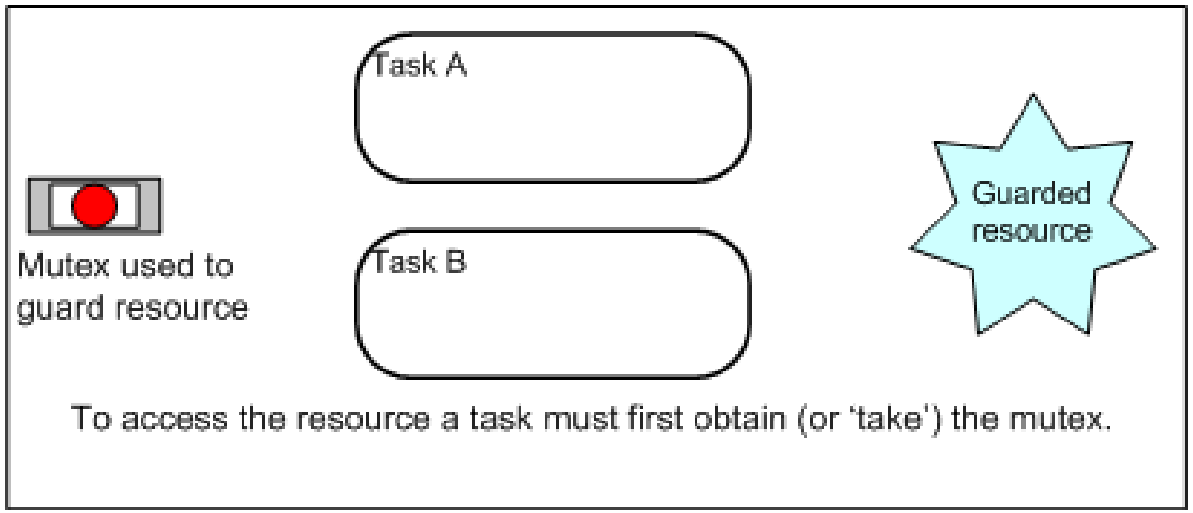
Muteksy (3/4)

Zakleszczenie

Podczas niewłaściwego wykorzystania muteksów może dojść do zjawiska zakleszczenia (z ang. *deadlock*) zasobów. Dzieje się to np., gdy zadanie A czeka na zasoby, które zostały zajęte przez zadanie B, natomiast zadanie B czeka na zasoby, które zostały zajęte przez zadanie A.

Wrocław University of Science and Technology

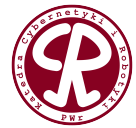
Muteksy (4/4)



[4]

Wrocław University of Science and Technology

[4] freertos.org



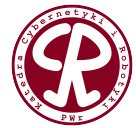
Sekcja krytyczna (1/3)

W FreeRTOSie występuje również inna sekcja określana mianem sekcji krytycznej. Jest to pewien obszar kodu, który otoczony jest przez:

```
1    taskENTER_CRITICAL();
2    // ...
3    taskEXIT_CRITICAL();
```

Pozwala to na tymczasowe wyłączenie przerwań lub do danego priorytetu (wykorzystywany jest mechanizm maskowania BASEPRI). Gdy sekcja krytyczna jest aktywna przełączanie kontekstu jest niemożliwe.

Wrocław University of Science and Technology



Sekcja krytyczna (2/3)

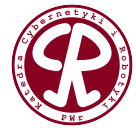
Sekcja konfiguracyjna FreeRTOSa dotycząca mechanizmu sekcji krytycznej odnosi się do definicji:

- configKERNEL_INTERRUPT_PRIORITY
- configMAX_SYSCALL_INTERRUPT_PRIORITY
- configMAX_API_CALL_INTERRUPT_PRIORITY

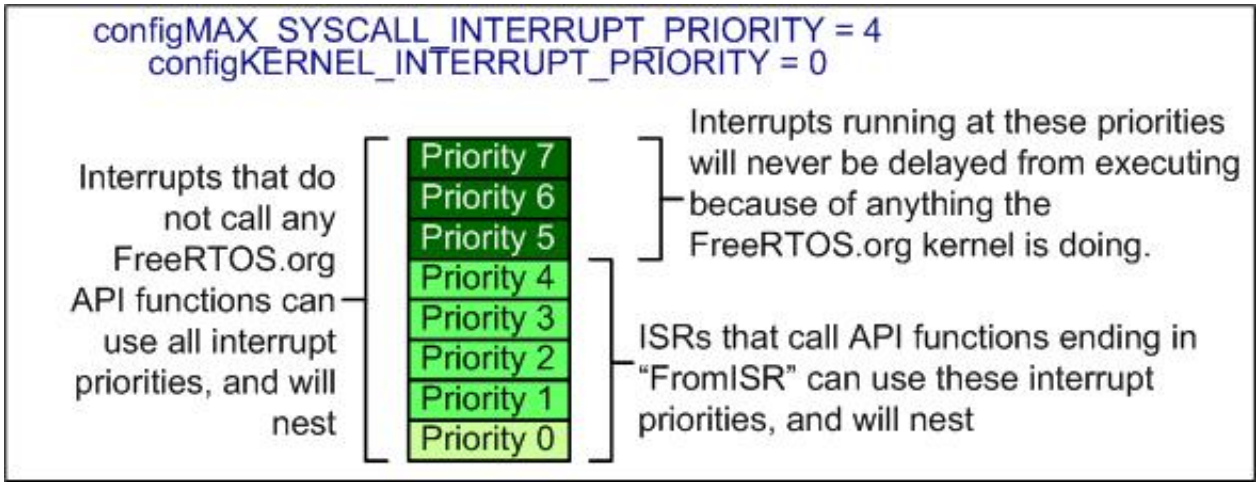
Uwaga!

W przypadku rdzeni Cortex-M3/M4 określenie priorytetu może być mylące. Najwyższy możliwy priorytet ma przypisaną najniższą liczbę 0.

Wrocław University of Science and Technology



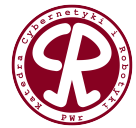
Sekcja krytyczna (3/3)



5

Wrocław University of Science and Technology

[5] freertos.org

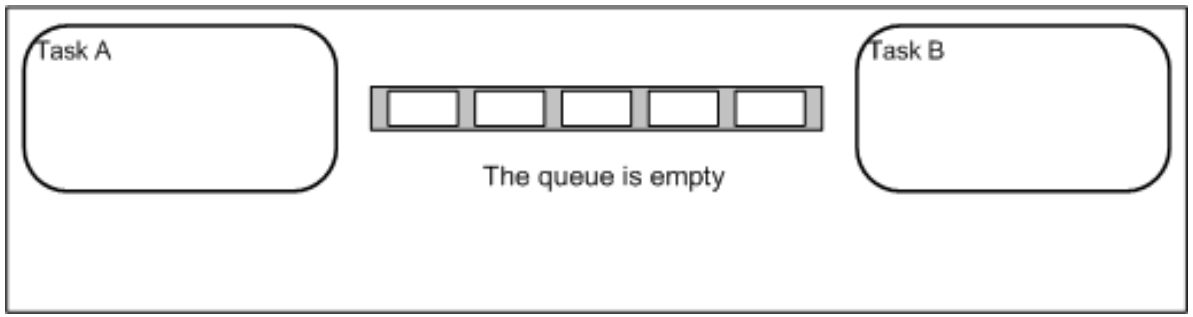


Outline

Wrocław University of Science and Technology

Kolejki (1/5)

Kolejki są zazwyczaj używane jako buforowanie według zasady **First In First Out (FIFO)**, gdzie dane są zapisywane na końcu (ogonie (z ang. *head*)) kolejki i usuwane z przodu (głowy (z ang. *tail*)) kolejki.



6

Kolejki (2/5)

Kolejkę można utworzyć za pomocą:

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,
  UBaseType_t uxItemSize );
```

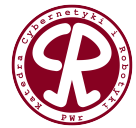
Wiadomość można wysłać do kolejki za pomocą:

```
BaseType_t xQueueSendToFront( QueueHandle_t xQueue,
  const void * pvItemToQueue,
  TickType_t xTicksToWait );
```

lub (zazwyczaj)

```
BaseType_t xQueueSendToBack( QueueHandle_t xQueue,
  const void * pvItemToQueue,
  TickType_t xTicksToWait );
```

Wrocław University of Science and Technology



Kolejki (3/5)

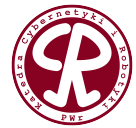
Wiadomość z kolejki można odczytać za pomocą:

```
1 BaseType_t xQueueReceive( QueueHandle_t xQueue,
2   void * const pvBuffer,
3   TickType_t xTicksToWait );
```

Aktualną liczbę elementów w kolejce można sprawdzić za pomocą:

```
1   UBaseType_t uxQueueMessagesWaiting( QueueHandle_t xQueue );
```

Wrocław University of Science and Technology



Kolejki (4/5)

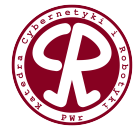
Kolejkę można używać jak **skrzynkę pocztową** (z ang. *mailbox*), gdzie rozmiar kolejki **zawsze** wynosi 1. Aktualną wiadomość można zastąpić za pomocą:

```
1 BaseType_t xQueueOverwrite( QueueHandle_t xQueue,
2   const void * pvItemToQueue );
```

Aby odczytać wiadomość z kolejki, można użyć funkcji `xQueueReceive()` lub jeśli wymagane jest zachowanie wiadomości w kolejce, można użyć poniższej funkcji:

```
1 BaseType_t xQueuePeek( QueueHandle_t xQueue,
2   void * const pvBuffer,
3   TickType_t xTicksToWait );
```

Wrocław University of Science and Technology



Kolejki (5/5)

Wysyłanie dużych fragmentów danych przez kolejkę

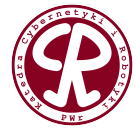
Zaleca się wysyłanie wskaźnika do obszaru zawierającego dane, zamiast wysyłania dużego fragmentu danych przez kolejkę.
W tym celu można użyć pól buforów (z ang. *buffer pools*).

Wrocław University of Science and Technology

[6] freertos.org

Outline

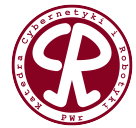
Wrocław University of Science and Technology



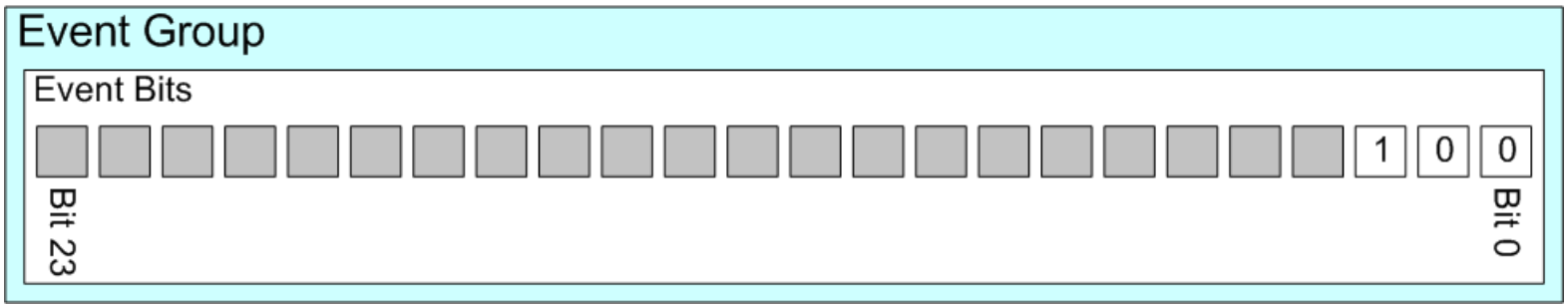
Grupy zdarzeń (1/3)

Grupa zdarzeń (z ang. *Event Group*) to mechanizm dostępny w FreeRTOSie, który pozwala zawiesić zadanie w stanie blokującym, dopóki określony warunek nie zostanie spełniony. Grupę zdarzeń można przedstawić jako zmienną, w której każdy bit nazywany jest flagą. Flaga może być ustawiona na logiczną ’1’. Zadanie może oczekiwać na określoną konfigurację bitów w tej zmiennej lub przynajmniej na jedną z flag.

Wrocław University of Science and Technology



Grupy zdarzeń (2/3)



7

Grupy zdarzeń (3/3)

Tworzenie grupy zdarzeń

```
1 EventGroupHandle_t xEventGroupCreate( void );
```

Bity w grupie zdarzeń można ustawić za pomocą:

```
1 EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup,
2   const EventBits_t uxBitsToSet );
```

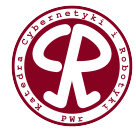
Zadanie jest zawieszane, gdy wywoływana jest funkcja:

```
1 EventBits_t xEventGroupWaitBits( const EventGroupHandle_t
      xEventGroup,
2   const EventBits_t uxBitsToWaitFor,
3   const BaseType_t xClearOnExit,
4   const BaseType_t xWaitForAllBits,
5   TickType_t xTicksToWait );
```

Parametr `uxBitsToWaitFor` jest maską, przez którą zdarzenie może być zidentyfikowane. Funkcja może oczekiwać na ustawienie wszystkich flag (bitów) lub nie (`xWaitForAllBits`).[7]

[7] freertos.org

Wrocław University of Science and Technology



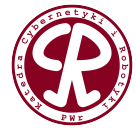
Outline

Wrocław University of Science and Technology

Liczniki programowe (1/4)

Liczniki programowe mogą uruchamiać funkcję zwrotną (z ang. *callback function*) w określonym czasie lub okresowo. Liczniki programowe działają niezależnie od sprzętu i są sterowane przez jądro FreeRTOSa. Rozdzielczość czasowa liczników programowych jest równa ziarnu taktu FreeRTOSa.

Wrocław University of Science and Technology



Liczniki programowe (2/4)

Licznik może zostać utworzony za pomocą:

```
1 TimerHandle_t xTimerCreate( const char * const pcTimerName,
2   TickType_t xTimerPeriodInTicks,
3   UBaseType_t uxAutoReload,
4   void * pvTimerID,
5   TimerCallbackFunction_t pxCallbackFunction );
```

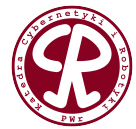
a uruchomiony za pomocą:

```
1 BaseType_t xTimerStart( TimerHandle_t xTimer, TickType_t
      xTicksToWait );
```

oraz zatrzymany za pomocą:

```
1 BaseType_t xTimerStop( TimerHandle_t xTimer, TickType_t
      xTicksToWait );
```

Wrocław University of Science and Technology



Liczniki programowe (3/4)

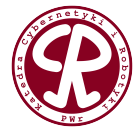
Każdy licznik programowy ma identyfikator (pvTimerID), który może być wykorzystany w dowolnym celu. Jest ustawiany podczas tworzenia licznika i może być zmieniany bezpośrednio za pomocą wywołania funkcji:

```
1 void vTimerSetTimerID( const TimerHandle_t xTimer,
2   void *pvNewID );
```

Identyfikator licznika można uzyskać za pomocą:

```
1 void *pvTimerGetTimerID( TimerHandle_t xTimer );
```

Wrocław University of Science and Technology



Liczniki programowe (4/4)

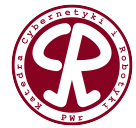
Okres licznika można modyfikować za pomocą:

```
1 BaseType_t xTimerChangePeriod( TimerHandle_t xTimer,
2   TickType_t xNewTimerPeriodInTicks,
3   TickType_t xTicksToWait );
```

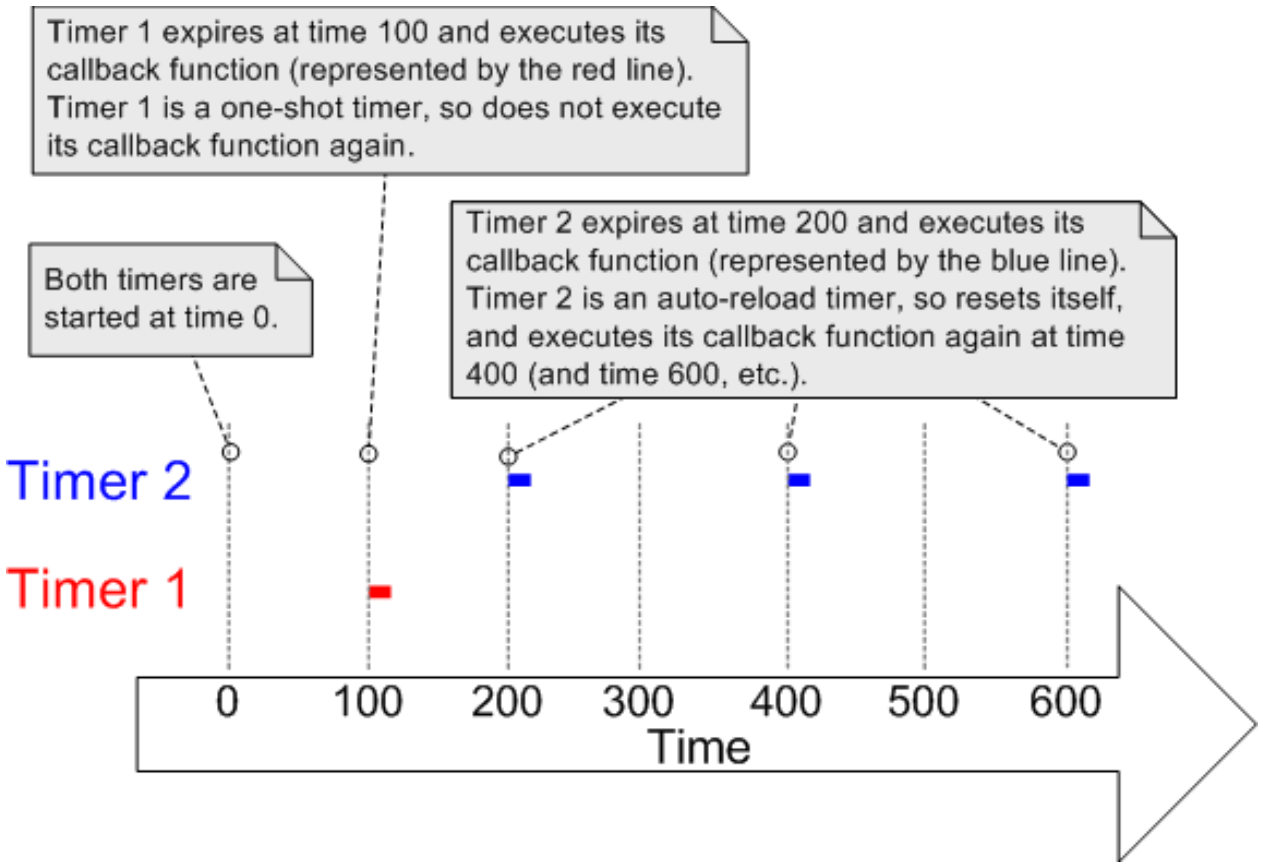
Resetowanie odbywa się za pomocą:

```
1 BaseType_t xTimerReset( TimerHandle_t xTimer,
2   TickType_t xTicksToWait );
```

Wrocław University of Science and Technology

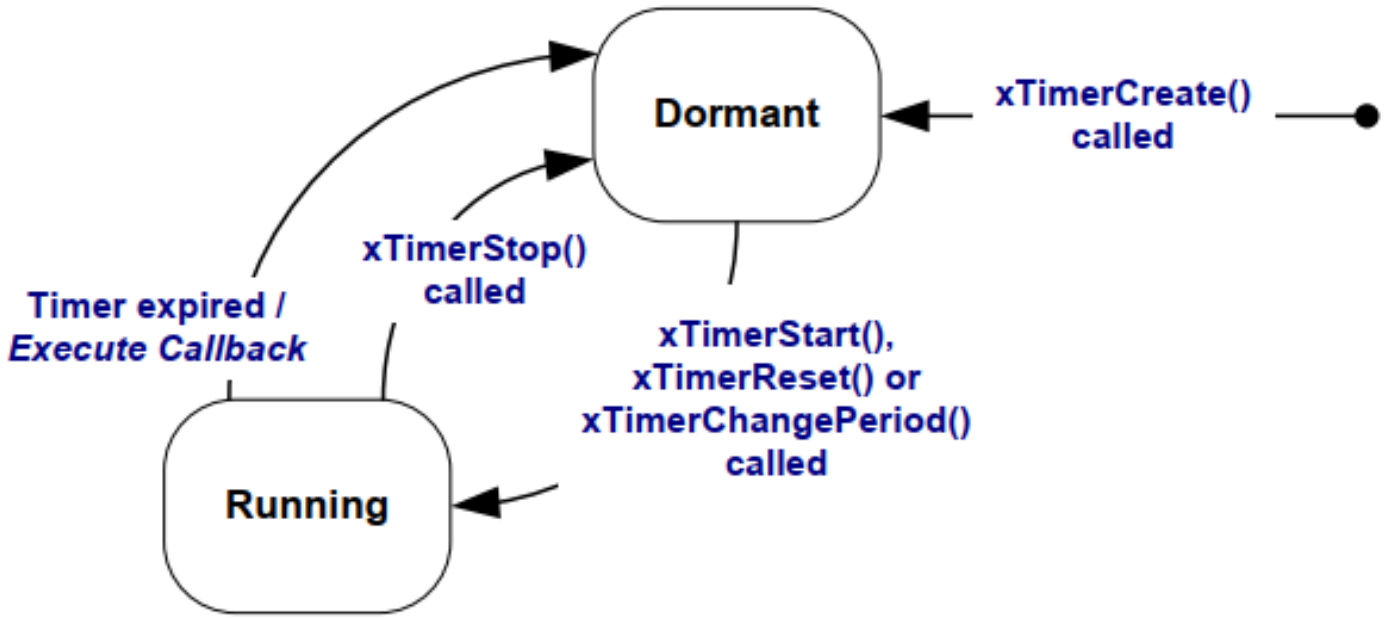


Licznik okresowy, a jednorazowy



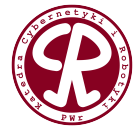
[8]

Maszyna stanu dla licznika jednorazowego (1/1)



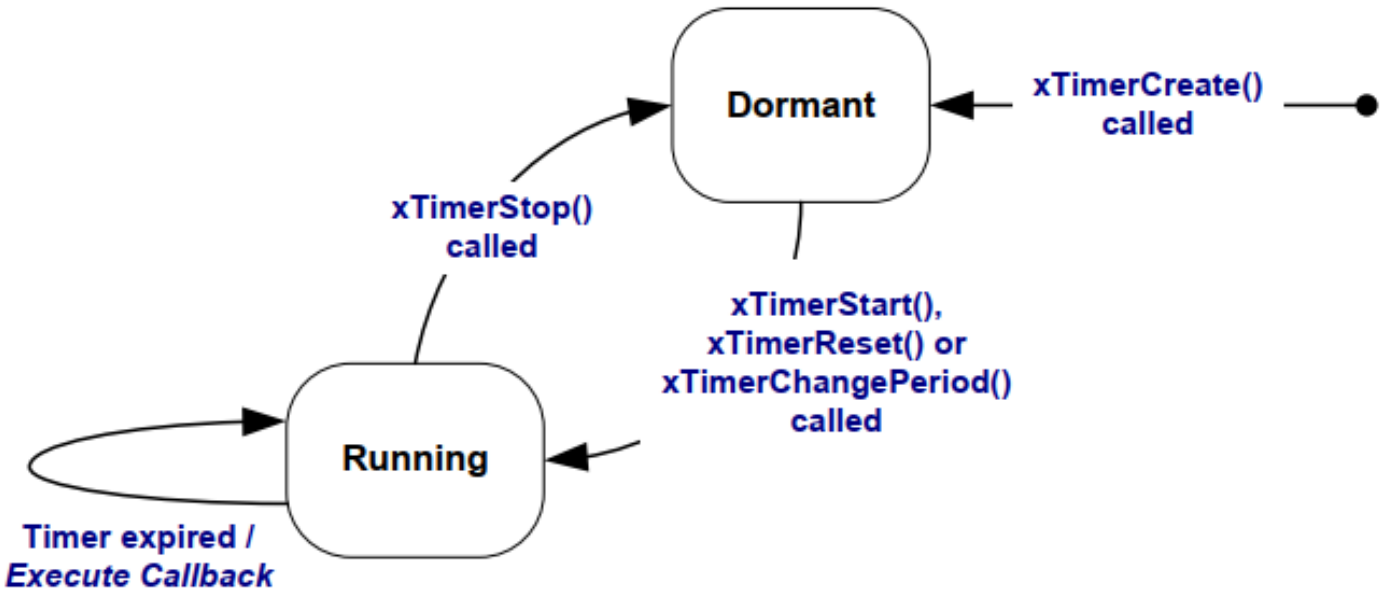
9

Wrocław University of Science and Technology



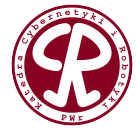
[9] R. Barry, Mastering the FreeRTOS™ Real Time Kernel

Maszyna stanów dla licznika okresowego (1/1)



10

Wrocław University of Science and Technology



[10] R. Barry, Mastering the FreeRTOS™ Real Time Kernel

Outline

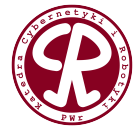
Wrocław University of Science and Technology

Statyczna alokacja pamięci

Tworzenie obiektów RTOS przy użyciu statycznie alokowanej pamięci RAM ma korzyści wynikające z większej kontroli nad aplikacją:

- Obiekty RTOS można umieścić w określonej lokalizacj pamięci.
- Maksymalny rozmiar zużycia pamięci RAM może być określony w czasie konsolidacji (z ang. *linking*), a nie w czasie wykonania (z ang. *run-time*).
- Programista nie musi martwić się o eleganckie obsługiwanie błędów alokacji pamięci.
- Pozwala to na użycie RTOS w aplikacjach, które nie pozwalają na dynamiczną alokację pamięci.

Wrocław University of Science and Technology



Dynamiczna alokacja pamięci (1/2)

Dynamiczne tworzenie obiektów RTOS ma korzyści związane z większą prostotą i potencjalnym zminimalizowaniem maksymalnego zużycia pamięci RAM aplikacji:

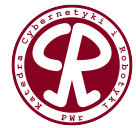
- Wymagane jest mniej parametrów funkcji podczas tworzenia obiektu.
- Alokacja pamięci odbywa się automatycznie wewnątrz funkcji API RTOS.
- Programista nie musi martwić się o alokację pamięci samodzielnie.
- Pamięć używana przez obiekt RTOS może być ponownie wykorzystana po jego usunięciu, co potencjalnie zmniejsza maksymalne zużycie pamięci RAM aplikacji.
- FreeRTOS pozwala na monitorowanie zużytej pamięci sterty (z ang. *heap*), co pozwala zoptymalizować jej rozmiar.

Wrocław University of Science and Technology

Dynamiczna alokacja pamięci (2/2)

- Schemat alokacji pamięci może być wybrany tak, aby najlepiej pasował do aplikacji, czy to **heap_1** dla prostoty i determinizmu często niezbędnego w aplikacjach krytycznych dla bezpieczeństwa, **heap_4** dla ochrony przed fragmentacją, **heap_5** do podziału sterty na wiele obszarów pamięci RAM lub schemat alokacji dostarczony przez samego programistę.

Wrocław University of Science and Technology

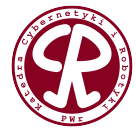


Implementacja

W systemie FreeRTOS istnieje kilka różnych typów stert, w tym:

- **heap_1** - bardzo prosty, nie pozwala na zwalnianie pamięci. Praktycznie nieprzydatny przez wsparcie dla statycznej alokacji pamięci.
- **heap_2** - pozwala na zwalnianie pamięci, ale nie scala sąsiadujących wolnych bloków. Obecnie wyparty przez **heap_4**.
- **heap_3** - po prostu opakowuje standardowe funkcje malloc() i free() dla bezpieczeństwa wątków.
- **heap_4** - scala sąsiadujące wolne bloki, aby uniknąć fragmentacji. Obejmuje opcję absolutnego umieszczenia adresu.
- **heap_5** - jak w przypadku heap_4, z możliwością rozszerzenia sterty na wiele nie sąsiadujących obszarów pamięci.

Wrocław University of Science and Technology



Monitorowanie sterty (1/3)

Uzyskanie informacji na temat zajętości sterty (z ang. *heap*) może być przeprowadzone za pomocą wywołania funkcji:

```
1 void vPortGetHeapStats( HeapStats_t *xHeapStats );
```

Wrocław University of Science and Technology

Monitorowanie sterty (2/3)

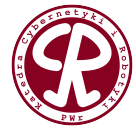
```
1 typedef struct xHeapStats
2 {
3   size_t xAvailableHeapSpaceInBytes;      /* The total heap size
        currently available - this is the sum of all the free
       blocks, not the largest block that can be allocated. */
4   size_t xSizeOfLargestFreeBlockInBytes;  /* The maximum size,
       in bytes, of all the free blocks within the heap at the
       time vPortGetHeapStats() is called. */
5   size_t xSizeOfSmallestFreeBlockInBytes; /* The minimum size,
       in bytes, of all the free blocks within the heap at the
       time vPortGetHeapStats() is called. */
6   size_t xNumberOfFreeBlocks;    /* The number of free memory
       blocks within the heap at the time vPortGetHeapStats() is
        called. */
7   size_t xMinimumEverFreeBytesRemaining;  /* The minimum amount
       of total free memory (sum of all free blocks) there has
       been in the heap since the system booted. */
8   size_t xNumberOfSuccessfulAllocations;  /* The number of calls
        to pvPortMalloc() that have returned a valid memory
       block. */
```

of Science and Technology

Monitorowanie sterty (3/3)

```
9     size_t xNumberOfSuccessfulFrees;   /* The number of calls to
          vPortFree() that has successfully freed a block of memory
          . */
10 } HeapStats_t;
```

Wrocław University
of Science and Technology



Ochrona pamięci stosu (1/3)

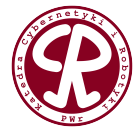
Wykrywanie przepełnienia stosu - Metoda 1

Prawdopodobne jest, że stos osiągnie swoją największą (najgłębszą) wartość po tym, jak jądro RTOS wyłączy zadanie z trybu *Running*, ponieważ wtedy stos będzie zawierał kontekst zadania. W tym momencie jądro RTOS może sprawdzić, czy wskaźnik stosu procesora pozostaje w ramach prawidłowej przestrzeni stosu. Funkcja obsługi przepełnienia stosu jest wywoływana, jeśli wskaźnik stosu zawiera wartość spoza prawidłowego zakresu stosu.

Ta metoda jest szybka, ale nie gwarantuje uchwycenia wszystkich przepełnień stosu.

```
1 #define configCHECK_FOR_STACK_OVERFLOW 1
```

Wrocław University of Science and Technology



Ochrona pamięci stosu (2/3)

Wykrywanie przepełnienia stosu - Metoda 2

Podczas tworzenia zadania jego stos jest wypełniany znaną wartością. Podczas wyłączania zadania z trybu Running, jądro RTOS może sprawdzić ostatnie 16 bajtów w ramach prawidłowego zakresu stosu, aby upewnić się, że te znane wartości nie zostały nadpisane przez zadanie lub działanie przerwań. Funkcja obsługi przepełnienia stosu jest wywoływana, jeśli którykolwiek z tych 16 bajtów jest inny niż wstępnie zainicjalizowany.

Ta metoda jest mniej wydajna niż pierwsza metoda, ale wciąż szybka. Bardzo prawdopodobne jest wykrycie przepełnień stosu, ale nadal nie ma gwarancji, że wszystkie przepełnienia zostaną uchwycone.

```
#define configCHECK_FOR_STACK_OVERFLOW  2
```

Ochrona pamięci stosu (3/3)

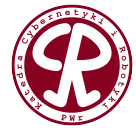
Wykrywanie przepełnienia stosu - Metoda 3

Ta metoda jest dostępna tylko dla wybranych implementacji FreeRTOSa.

Metoda ta umożliwia sprawdzanie stosu przerwań ISR. Po wykryciu przepełnienia stosu przerwania ISR, zostaje wywołana asercja. Funkcja obsługi przepełnienia stosu nie jest wywoływana w tym przypadku, ponieważ dotyczy ona tylko stosu zadania, a nie stosu przerwania ISR.

```
1   #define configCHECK_FOR_STACK_OVERFLOW  3
```

Wrocław University of Science and Technology



Outline

Wrocław University of Science and Technology

Statystyki (1/2)

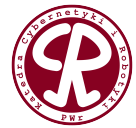
FreeRTOS dostarcza możliwości uzyskania statystyk dotyczących uruchomionych procesów.

```
1 void vTaskGetRunTimeStats( char *pcWriteBuffer );
```

Powyższa funkcja dostarcza informacji o:

- całkowitym czasie wykonywania zadania,
- procentowym udziale całkowitego czasu wykonywania zadania.

Wrocław University of Science and Technology

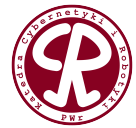


Statystyki (2/2)

Wymagana jest odpowiednia konfiguracja FreeRTOSa oraz implementacja funkcji, które inicjalizują licznik oraz dostarczają jego aktualnej wartości.

```
1 #define configGENERATE_RUN_TIME_STATS    1
2
3 portCONFIGURE_TIMER_FOR_RUN_TIME_STATS();
4
5 portGET_RUN_TIME_COUNTER_VALUE();
```

Wrocław University of Science and Technology



Konfiguracja

Konfiguracja FreeRTOSa może być przeprowadzona przez modyfikację parametrów (definicji) występujących w plik `FreeRTOSConfig.h`. Wybrane parametry:

```
1  #define configUSE_PREEMPTION                    1
2  #define configUSE_PORT_OPTIMISED_TASK_SELECTION 0
3  #define configUSE_TICKLESS_IDLE                 0
4  #define configCPU_CLOCK_HZ                      60000000
5  #define configSYSTICK_CLOCK_HZ                  1000000
6  #define configTICK_RATE_HZ                      250
7  #define configMAX_PRIORITIES                    5
8  #define configMINIMAL_STACK_SIZE                128
9  #define configMAX_TASK_NAME_LEN                 16
10 #define configUSE_16_BIT_TICKS                  0
11 #define configTICK_TYPE_WIDTH_IN_BITS
       TICK_TYPE_WIDTH_16_BITS
12 #define configIDLE_SHOULD_YIELD                 1
13 #define configUSE_TASK_NOTIFICATIONS            1
14 #define configTASK_NOTIFICATION_ARRAY_ENTRIES   3
15 // ...
```

of Science and Technology