Systemy czasu rzeczywistego II

Prof. dr hab. inż. Leszek Trybus

Dr inż. Bartosz Trybus

Dr inż. Sławomir Samolej

Literatura

- **Burns A, Wellings A.,**
  Real-Time Systems and Programming Languages, third edition, Pearson Education Limited 2001.
- **Butazzo G. C.**
  Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers 1997.
- **Cottet F., Delacroix L., Kaiser C. Mammeri Z., Scheduling in real-time systems, Wiley 2002.**
- **Szymczyk P.**
  Systemy operacyjne czasu rzeczywistego, Wydawnictwo AGH 2003.
- **WxWorks Documentation**

Spotykane pojęcia

- **Systemy o twardych wymaganiach czasowych (ang. Hard Real-Time Systems)**
  Systemy, gdzie ograniczenia czasowe muszą być zawsze spełnione, np. system kontroli lotu myśliwca.
- **Systemy o miękkich wymaganiach czasowych (ang. Soft Real-Time Systems)**
  Systemy, gdzie ograniczenia czasowe są istotne, ale uznaje się, że działają poprawnie, jeśli od czasu do czasu nastąpi przekroczenie ograniczeń, np. system akwizycji danych.
- **Systemy o solidnych wymaganiach czasowych(ang. Firm Real-Time Systems)**
  Systemy o miękkich wymaganiach czasowych, w których spóźniona odpowiedź jest traktowana jako błędna.

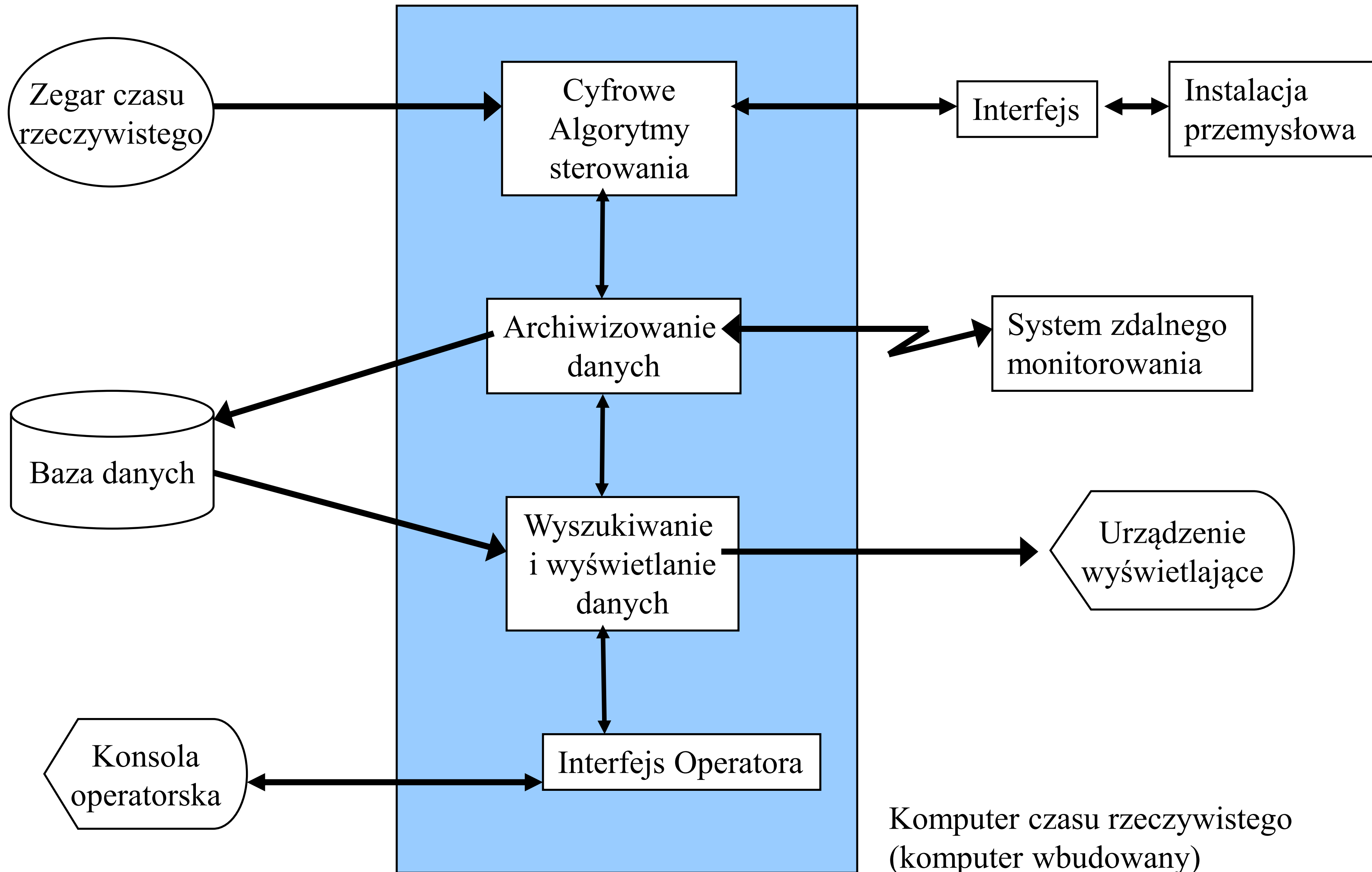
Pojedynczy system czasu rzeczywistego posiada zwykle podsystemy o twardych i miękkich wymaganiach czasowych. Definiowane są również funkcje kosztu przekroczenia ograniczeń czasowych.

Spotykane pojęcia

- **Systemy wysokozintegrowane (ang. High Integity Systems)**
  Systemy, w których oprogramowanie steruje pewnymi procesami mającymi istotny wpływ na ludzi, środowisko, organizacje i społeczeństwo.
  Systemy wysokozintegowane dzieli się na:
  - **Systemy krytyczne ze względu na bezpieczeństwo (ang. Safety Critical Systems), gdzie wynik działania systemu ma bezpośredni wpływ na życie, zdrowie człowieka lub na stan środowiska, np. systemy sterowania samolotu, ruchem pociągów, oprogramowanie samochodów i innych środków transportu, zabezpieczenia systemów energetycznych, urządzenia medyczne.**
  - **Systemy krytyczne ze względu napełniona misję (ang. Mission Critical Systems), gdzie poprawność działania oprogramowania ma poważny wpływ na działanie produkcji, instytucji, organizacji**

Szereg systemów krytycznych ze względu na bezpieczeństwo jest systemami czasu rzeczywistego.

System czasu rzeczywistego – uogólniony schemat



Metody programowania systemów czasu rzeczywistego

1. **Tworzenie aplikacji bezpośrednio zarządzających elementami systemu mikrokomputerowego**
2. **Programowanie sterowników swobodnie programowalnych**
3. **Zastosowanie języków programowania systemów czasu rzeczywistego (Ada2005, Occam, Pearl, Real-Time Java).**
4. **Zastosowanie wbudowanych funkcji systemów operacyjnych czasu rzeczywistego (język C/ C++).**

KLUCZOWE CECHY SYSTEMÓW CZASU RZECZYWISTEGO

- **Z punktu widzenia programisty system czasu rzeczywistego jest złożony ze zbioru współbieżnych zadań wymieniających informacje z otoczeniem i ze sobą.**
- **W związku z tym programowanie SCR obejmuje zagadnienia:**
  - **Programowania wielowątkowego**
  - **Komunikacji i synchronizacji w systemach współbieżnych**
    - **Współdzielenie zmiennych**
    - **Komunikaty**
    - **Współdzielenie zasobów**
  - **Synchronizacji czasowej**
  - **Szeregowania**
  - **Obsługi przerwań (obsługa podgrupy zdarzeń zewnętrznych)**
  - **Niezawodności, odporności na błędy (wyjątki).**

Cechy programowania systemów czasu rzeczywistego

- **Możliwość zdefiniowania spójnych części, realizujących obsługę zdarzeń pojawiających się w otoczeniu – zadań (aktywowanych zdarzeniami lub upływem czasu).**
- **Konieczność istnienia mechanizmów synchronizacji i wymiany informacji pomiędzy zadaniami.**
- **Konieczność określenia dla zadań priorytetu oraz możliwość wywłaszczania zadania.**
- **Możliwość uzależnienia działania systemu od czasu i innych zdarzeń zachodzących w otoczeniu.**
- **Możliwość definiowania procedur obsługi dla nietypowych (np. procesowych) wejść i wyjść.**
- **Istnienie mechanizmów umożliwiających konstruowanie oprogramowania o podwyższonej niezawodności (np. obsługa wyjątków).**

Czas i przedział czasowy

- **Symulacja upływu czasu w komputerze:**
  - **Generator impulsów przesyłanych do licznika**
  - **Przepełnienie licznika powoduje zgłoszenie przerwania obsługiwanego przez system operacyjny**
  - **System w przerwaniu modyfikuje zegar programowy (przesuwa go o 1 tyknięcie)**
  - **Wartość tyknięcia zależy od dokładności symulacji pomiaru czasu i jest kompromisem pomiędzy precyzja a efektywnością.**
- **W klasycznych systemach operacyjnych:**
  - **System operacyjny aktualizuje zegar i kalendarz systemowy**
  - **Obiekty te są dostępne dla programów użytkowych przez wywołanie odpowiednich funkcji**
  - **W typowych aplikacjach dokładność odmierzania czasu nie jest zadaniem krytycznym**

Czas i przedział czasowy

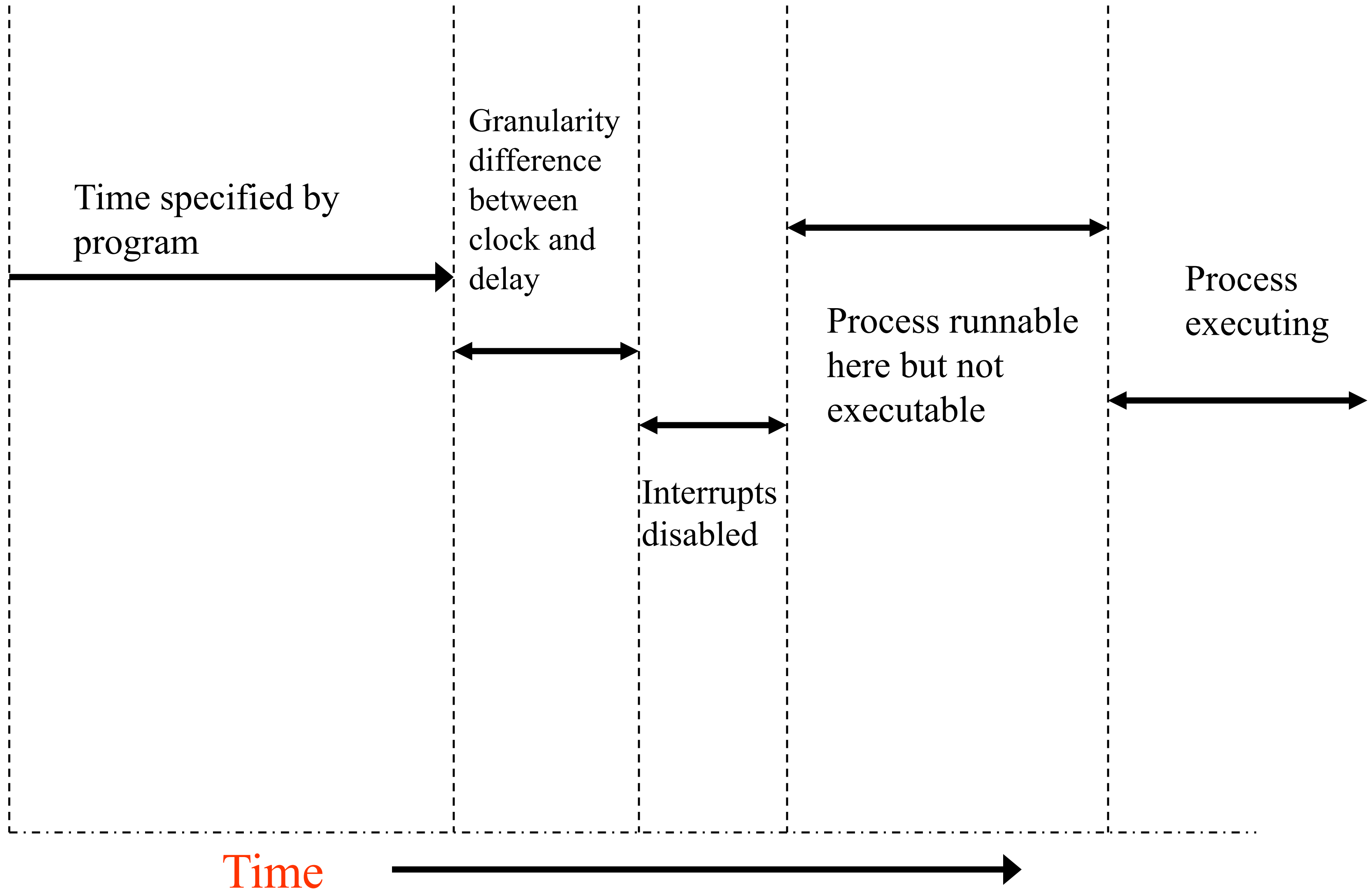
- **W systemach czasu rzeczywistego oczekuje się, że:**
  - **Część zadań realizowanych jest cyklicznie z różnymi okresami aktywacji**
  - **Inne zadania muszą być aktywowane w określonych momentach czasowych**
- **Stąd na potrzeby systemów czasu rzeczywistego definiuje się:**
  - **Budziki (*watch dogs*), które nastawione na**
    - **określony moment czasowy (aktywacja w określonej chwili) lub**
    - **czas trwania (aktywacja po upływie pewnego czasu)**

  decydują o aktywacji przypisanych do nich procesów lub zadań.
- **Częstotliwość aktywacji niektórych zadań może być bardzo duża (systemy monitorowania i sterowania w czasie rzeczywistym procesów szybkozmiennych, np. monitorowanie drgań, sterowanie startem rakiety). W tych wypadkach wymagana jest duża dokładność pomiaru czasu, aby okres aktywacji zadań mógł być mierzony w milisekundach, a niekiedy dokładniej.**

Czas i przedział czasowy

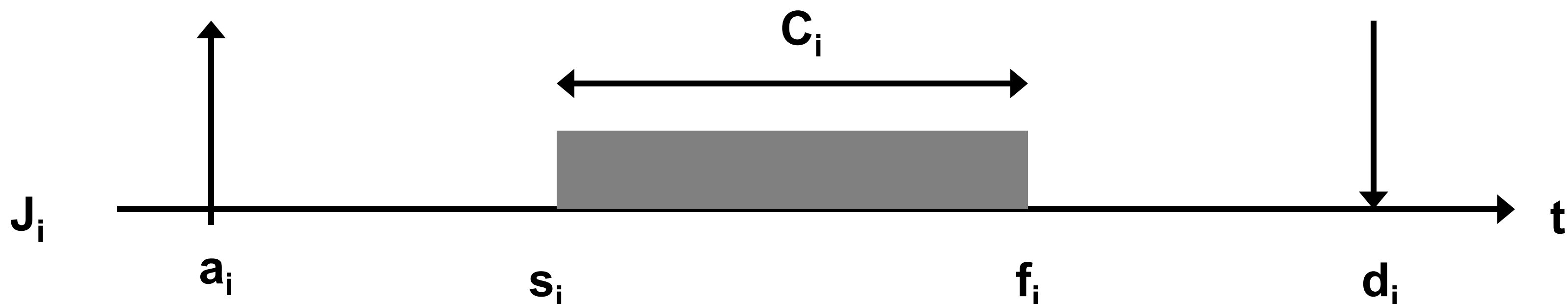
- **Czynniki decydujące o dokładności określenia momentu wykonywania zadania:**
  - **Dokładność aktualizacji zegara**
    (w niektórych systemach można określić ilość impulsów generatora, co którą następuje przepełnienie licznika i zgłoszenie przerwania.)
  - **Faktyczny czas reakcji na przerwanie zegarowe**
    (Budzik przypisany do danego zadania przenosi je w stan gotowości (*ready*) ale nie powoduje jego uruchomienia. Np. jeśli priorytet zadania aktualnie wykonywanego jest wyższy od reaktywowanego zadania, to musi ono czekać na zakończenie obliczeń przez zadanie o wyższym priorytecie.

Jak działania opóźnienie



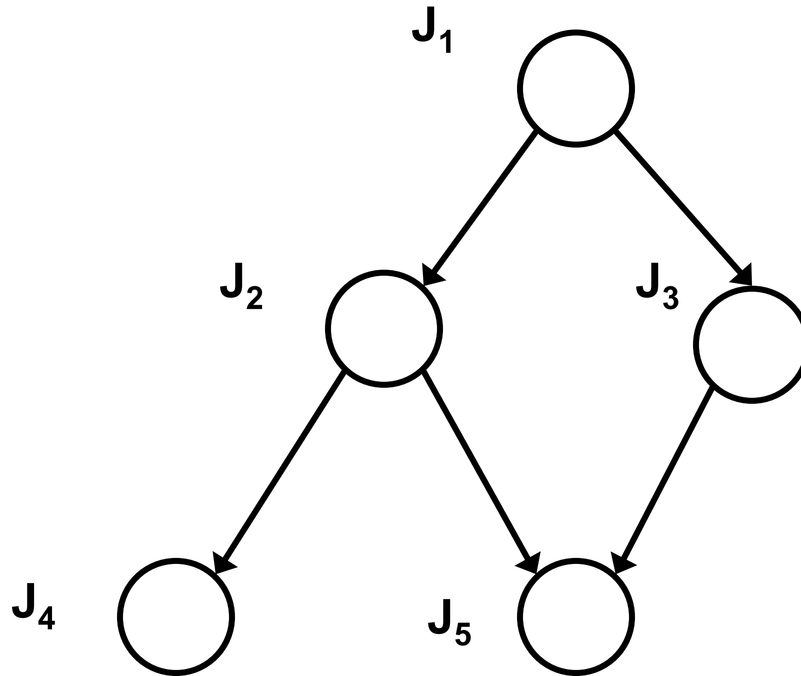
Typy ograniczeń zadań czasu rzeczywistego

- **Ograniczenia czasowe (twarde, miękkie, solidne)**
  - **Parametry opisujące parametry czasowe zadań czasu rzeczywistego:**
    - **Czas napłynięcia zadania (Arrival time) a_i,**
    - **Czas trwania obliczeń (Computation time) C_i,**
    - **Ostateczny termin zakończenia obliczeń (Deadline) d_i,**
    - **Czas rozpoczęcia obliczeń (Start time) s_i,**
    - **Czas zakończenia obliczeń (Finishing time) f_i,**
    - **Krytyczność zadania (twarde, miękkie),**
    - **Wartość zadania v_i,**
    - **Spóźnienie zadania (Lateness) $L_i = f_i – d_i$,**
    - **Luźny czas zadania (Laxity) $X_i = d_i – a_i – C_i$,**



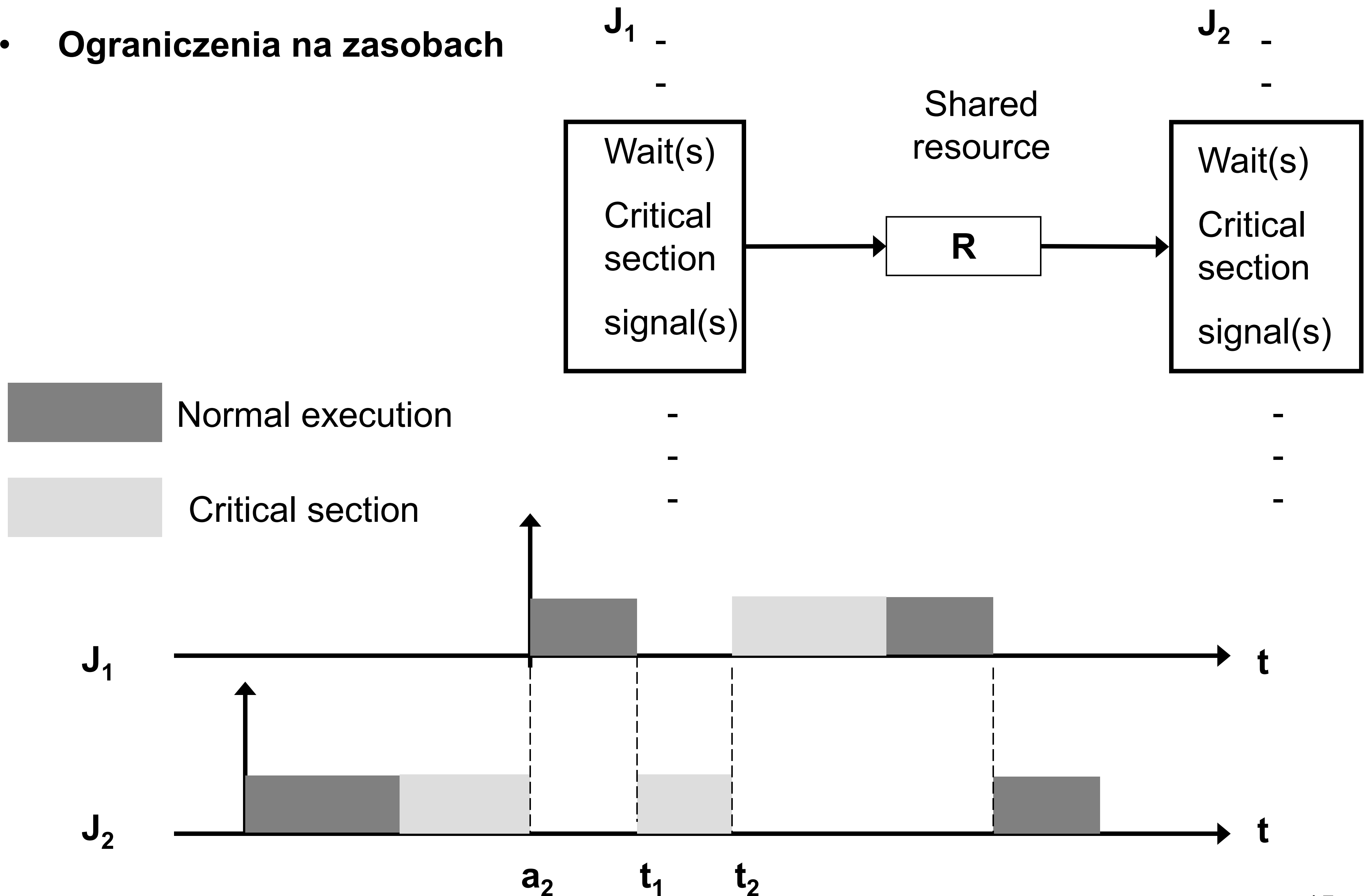
Typy ograniczeń zadań czasu rzeczywistego

- **Ograniczenia pierwszeństwa**



Typy ograniczeń zadań czasu rzeczywistego

- **Ograniczenia na zasobach**



Szeregowanie periodycznych zadań czasu rzeczywistego - wprowadzenie

- **Wiele aplikacji czasu rzeczywistego (np. sterowanie) składa się ze zbioru zadań wykonywanych cyklicznie (odczyt danych sensorycznych, pętle sterowania, monitorowanie).**
- **Oczekuje się, że zadania wykonywać się będą cyklicznie z założoną częstotliwością określoną na podstawie wymagań danej aplikacji (np. wolna, szybka pętla obliczeń)**
- **Kiedy aplikacja ma się składać z wielu współbieżnych zadań z określonymi ograniczeniami czasowymi projektant musi gwarantować, że każda periodyczna instancja jest regularnie aktywowana z prawidłową częstotliwością i kończy swoje obliczenia przed upłynięciem ostatecznego czasu zakończenia obliczeń.**
- **Jednym z pierwszych opracowanych podejść było:**
  - **Wykonywanie cykliczne (ang. Cyclic Executive)**
- **Podstawowe algorytmy szeregowania dla zbioru cyklicznych zadań, to:**
  - **Rate Monotonic (RM)**
  - **Earliest Deadline First (EDF)**
  - **Deadline Monotonic Priority Ordering (DMPO)**

Wykonywanie cykliczne

- Jednym z popularnych podejść do konstruowania systemów czasu rzeczywistego jest zastosowanie wykonywania cyklicznego
- Podejście do projektowania jest współbieżne, ale generowany kod aplikacji jest zestawem procedur wywoływanych zgodnie z tablicą
- Cała tablica obejmuje główny cykl systemu podzielony zwykle na pośrednie cykle o stałej długości
- Pośrednie cykle decydują o minimalnym cyklu w systemie
- Główny cykl określa maksymalny cykl systemu

Główna zaleta:

- Taki system jest w pełni deterministyczny

Wykonywanie cykliczne – przykład

- **Rozważany będzie następujący zestaw procesów:**

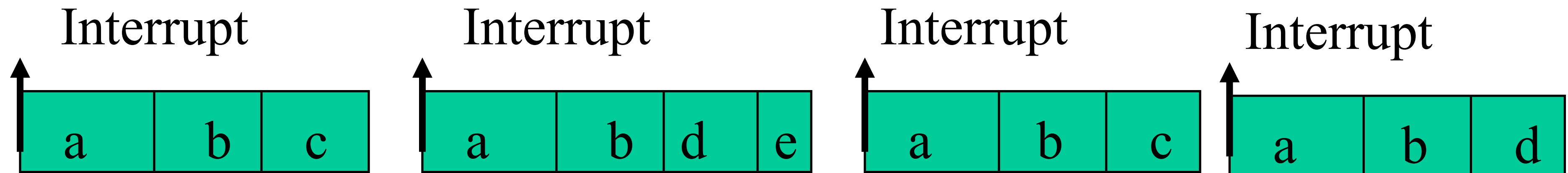
Process	Period, T	Computation Time, C
a	25	10
b	25	8
c	50	5
d	50	4
e	100	2

Wykonywanie cykliczne – przykład

- **Uszeregowanie procesów można zapisać następująco:**

```
loop
  wait_for_interrupt;
  procedure_for_a; procedure_for_b; procedure_for_c;
  wait_for_interrupt;
  procedure_for_a; procedure_for_b; procedure_for_d;
  procedure_for_e;
  wait_for_interrupt;
  procedure_for_a; procedure_for_b; procedure_for_c;
  wait_for_interrupt;
  procedure_for_a; procedure_for_b; procedure_for_d;
end loop;
```

- **Co przekłada się na następujący schemat wykonywania procesów:**



Wykonywanie cykliczne – właściwości

- **Wyeliminowano zestaw procesów (zadań) z systemu. Każdy z pośrednich cyklów jest sekwencją wołań procedur**
- **Procedury współdzielą wspólna przestrzeń adresową stąd mogą wymieniać dane. Dane nie muszą być chronione (np. przez semafory), ponieważ współbieżny dostęp do zasobów jest niemożliwy**
- **Wszystkie okresy „procesów" muszą być wielokrotnością cyklu pośredniego.**

Wykonywanie cykliczne – problemy

- **Mogą pojawić się problemy z włączeniem w system procesów o długim okresie. Główny cykl jest zarazem maksymalnym cyklem w systemie bez konieczności włączania dodatkowych modułów szeregujących**
- **Zadania sporadyczne są trudne (jeśli niemożliwe) do włączenia w system**
- **Stworzenie tabeli wykonywania cyklicznego jest trudne. Trudne jest również dokonywanie uaktualnień. Z punktu widzenia matematycznego otrzymuje się problem NP-trudny.**
- **Może zaistnieć konieczność podzielania jednego „procesu” na sekwencję procedur o jednakowym czasie wykonywania, co może powodować dodatkowe błędy (program traci „eleganckość” z punku widzenia inżynierii oprogramowania)**
- **Podejście praktycznie uniemożliwia zastosowanie innych, bardziej elastycznych metod szeregowania**
- **W praktycznych zastosowaniach można zrezygnować z pełnego determinizmu aplikacji z zachowaniem przewidywalności**

Wniosek:

- **Dla prostych, cyklicznych systemów warto rozważyć wykonywanie cykliczne, dla pozostałych zastosować trzeba inne, zaawansowane rozwiązania.**

Założenia dotyczące zbioru zadań przewidzianych do szeregowania (RM, EDF, DMPO)

1. Instancje (kolejne wystąpienia) zadania periodycznego t_i są regularnie aktywowane ze stałą częstotliwością. Przedział T_i pomiędzy kolejnymi aktywacjami nazywany jest okresem zadania.
2. Wszystkie instancje zadania mają ten sam najdłuższy czas wykonywania (WCET- Worst Case Execution Time) C_i.
3. Wszystkie instancje zadania mają tą samą wartość względnego ostatecznego terminu zakończenia obliczeń (Relative deadline) D_i.
4. Wszystkie szeregowane zadania są niezależne tzn. nie ma pomiędzy nimi relacji poprzedzania ani ograniczeń na zasobach.

Dodatkowo:

5. Zadanie nie może zawiesić samego siebie (np. Dla wykonania operacji wej/wyj).
6. Zadania są wprowadzane do wykonywania z chwilą ich nadejścia do kolejki.
7. W rozważaniach pominięte będą czasy na obsługę przełączania zadań itp.

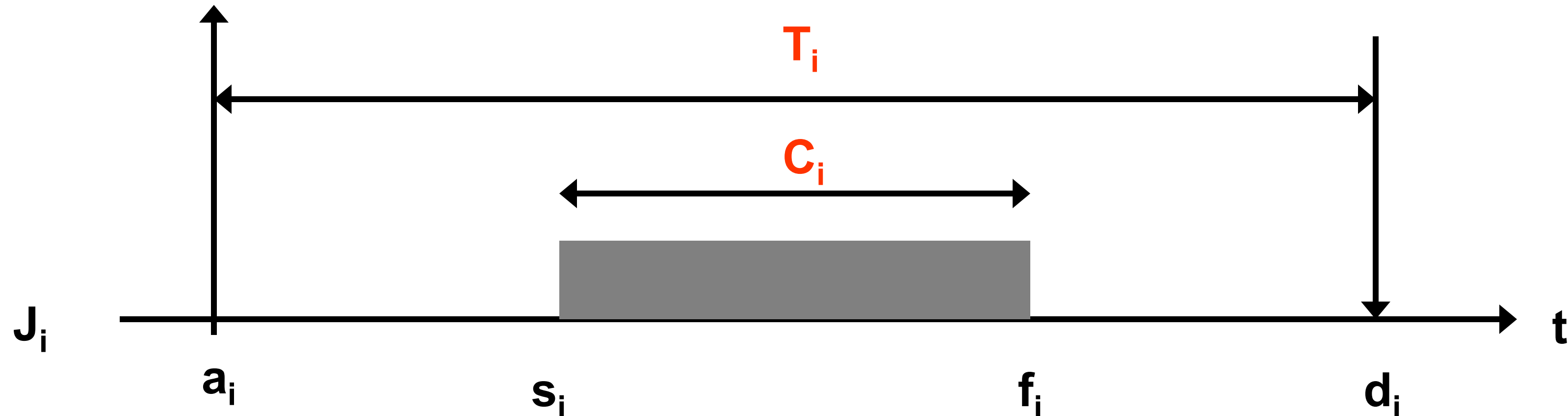
Zadania spełniające założenia 1.-4. Opisać można przez 3 parametry:

- Czas aktywacji pierwszej instancji danego zadania Φ_i,
- Okres danego zadania T_i,
- Najdłuższy czas wykonywania danego zadania C_i.

Współczynnik wykorzystania procesora

- **Wzór:**

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$



- **Współczynnik opisuje obciążenie procesora podczas wykonywania zbioru cyklicznych zadań.**
- **Dla różnych algorytmów szeregowania i różnych zbiorów zadań współczynnik może przyjmować różne graniczne wartości, dla których dany zbiór zadań jest szeregowalny (może wykonać wszystkie swoje obliczenia w zadanych ograniczeniach czasowych).**
- **Jeśli współczynnik przyjmuje wartość większą od 1, dany zbiór zadań nie może być szeregowany przez żaden algorytm.**

Algorytm szeregowania Rate Monotonic (RM) – podstawowe właściwości

- Zasada przydziału priorytetów zadaniom:
  - Priorytet zadaniom przydziela się w zależności od częstotliwości wznawiania obliczeń (okresu).
  - Zadanie najczęściej wznawiane otrzymuje najwyższy priorytet. Kolejnym zadaniom o kolejnych większych okresach przydziela się kolejne niższe priorytety.
  - Priorytety przydziela się na stałe, przed rozpoczęciem wykonywania obliczeń przez zadania.
- Z założenia algorytm RM przyjmuje możliwość przełączenia (wywłaszczenia) bieżącego zadania przez nowo aktywowane zadanie o wyższym priorytecie (mniejszym okresie).
- Wykazano matematycznie, że algorytm RM jest optymalny spośród wszystkich algorytmów szeregowania zadań czasu rzeczywistego ze stałym przydziałem priorytetów.
- Wskazano zasadę obliczanie maksymalnej wartości współczynnika wykorzystania procesora, dla której dany zbiór zadań czasu rzeczywistego jest szeregowalny.

Algorytm szeregowania RM – interpretacja

Process	Period T	Computation Time C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33
				0.82

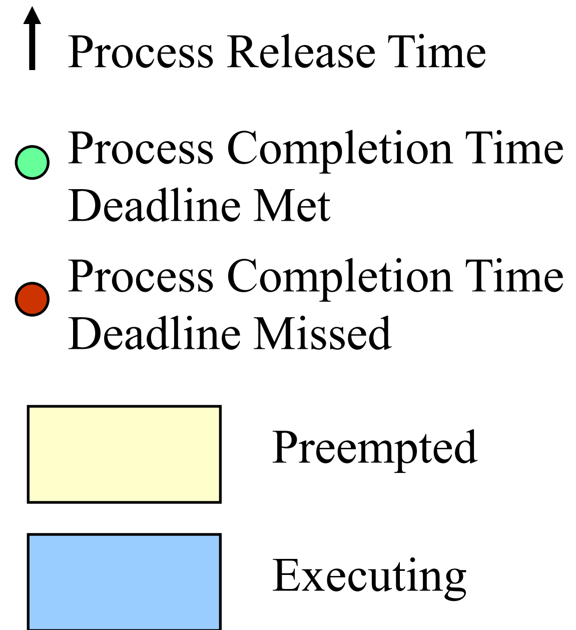
a

b

c

0 10 20 30 40 50 60

Time



Algorytm szeregowania RM – warunek szeregowalności

- **Teoretyczny warunek szeregowlaności (dla małej ilości zadań):**

$$U_{gr} = \sum_{i=1}^{n} \frac{C_i}{T_i} \le n\left(2^{\frac{1}{n}} - 1\right)$$, gdzie n – ilość zadań (1973)

$$U_{gr} = \prod_{i=1}^{n} \left(\frac{C_i}{T_i} + 1\right) \le 2$$, gdzie n – ilość zadań (2003)

- **Dla dużej ilości zadań (1973):**

$$U_{gr} = \ln 2 \approx \mathbf{0.69}$$

- **Dla przypadkowo dobieranych zbiorów zadań wykazano eksperymentalnie graniczną wartość współczynnika wykorzystania procesora, przy którym zbiór zadań jest szeregowalny:**

$$U_{gr} = \mathbf{0.88}$$

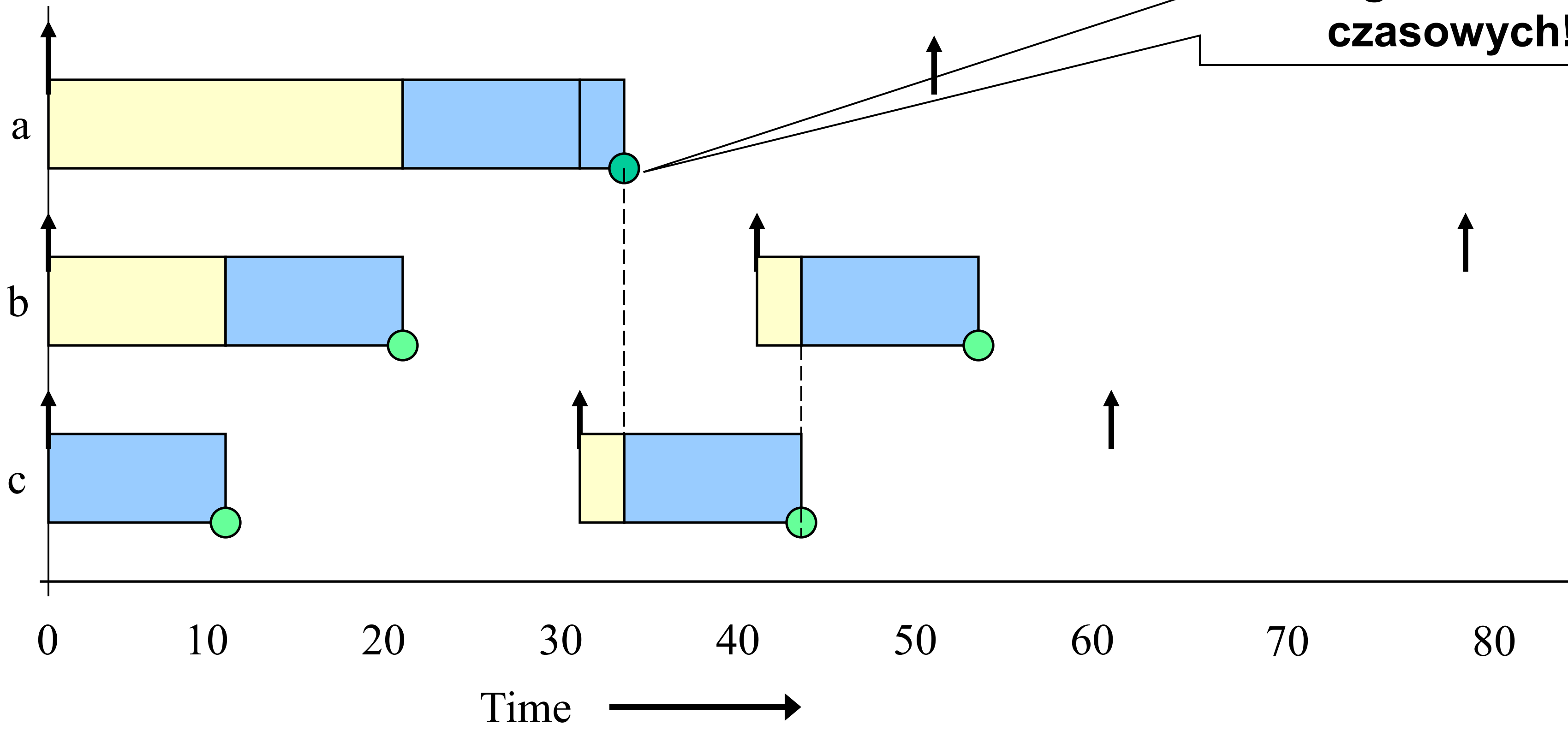
- **Jeśli dla danego zbioru zadań współczynnik wykorzystania procesora mieści się pomiędzy U_{gr} a 1, to nic nie można powiedzieć o wykonalności danego zbioru.**

Algorytm szeregowania Earliest Deadline First (EDF) – podstawowe właściwości

- Zasada przydziału priorytetów zadaniom:
  - Priorytet zadaniom przydziela się dynamicznie w zależności wartości absolutnego ostatecznego terminu zakończenia obliczeń.
  - Zadanie, które musi najszybciej zakończyć obliczenia otrzymuje najwyższy priorytet. Priorytety pozostałych zadań przydziela się według kolejnych zbliżających się dla nich ostatecznych terminów zakończenia obliczeń.
  - W związku z tym, że absolutny ostateczny termin zakończenia obliczeń w czasie wykonywania każdej z instancji zadania może ulec zmianie, zmianie w sposób dynamiczny ulegają również priorytety danych instancji zadań .
- Z założenia algorytm EDF przyjmuje możliwość przełączenia (wywłaszczenia) bieżącego zadania przez nowo aktywowane zadanie o wyższym priorytecie.
- Wykazano matematycznie, że algorytm EDF jest optymalny spośród wszystkich algorytmów szeregowania zadań czasu rzeczywistego z dynamicznym przydziałem priorytetów.
- Wskazano zasadę obliczanie maksymalnej wartości współczynnika wykorzystania procesora, dla której dany zbiór zadań czasu rzeczywistego jest szeregowalny.
- Algorytm jest również optymalny dla zadań nieperiodycznych.

Algorytm szeregowania EDF – interpretacja

Process	Period T	ComputationTime C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33



Algorytm szeregowania EDF – dyskusja

- **Warunek szeregowlaności:** $$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

- **Dlaczego EDF nie jest preferowanym algorytmem szeregowania:**
  - **Algorytmy ze stałymi priorytetami są łatwiejsze w implementacji (mniejszy narzut obliczeniowy dla sytemu operacyjnego).**
  - **W przypadku przepełnienia zachowanie algorytmów ze stałymi priorytetami jest bardziej przewidywalne (procesy o niższym priorytecie nie wykonają się terminowo); Natomiast algorytm EDF może spowodować efekt domino, w którym wiele procesów nie spełni swoich ograniczeń czasowych.**
  - **Zastosowanie współczynnika utylizacji do określenia szeregowalności jest mylące. W przypadku algorytmów ze stałymi priorytetami możliwe jest poprawne działanie systemu pomimo, że przekroczy się teoretycznie wyznaczoną graniczną wartość współczynnika.**

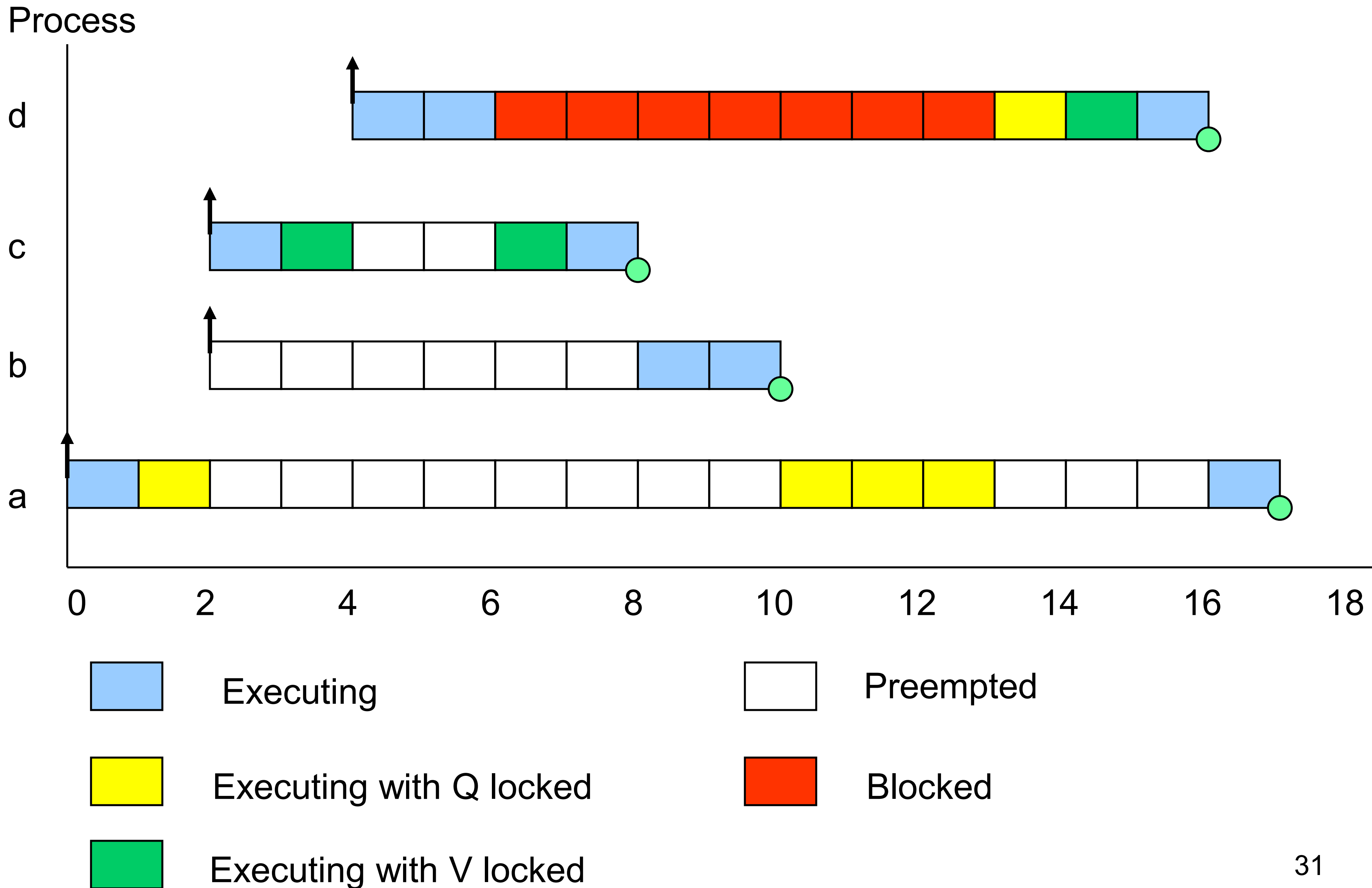
Współdzielenie zasobów - inwersja priorytetów

- **Jeśli dany proces jest zawieszony i oczekuje na zakończenie obliczeń przez proces o niższym priorytecie, wtedy model priorytetów w danym systemie jest w pewnym sensie podważony.**
- **Mówi się wtedy, że w systemie zachodzi inwersja priorytetów.**
- **Proces oczekujący na inny proces o niższym priorytecie nazywa się procesem zablokowanym (blocked).**

- **Dla zilustrowania ekstremalnego przykładu inwersji priorytetów rozważone zostanie wykonywanie 4 periodycznych zadań: a, b, c i d oraz dwu zasobów q i v.**

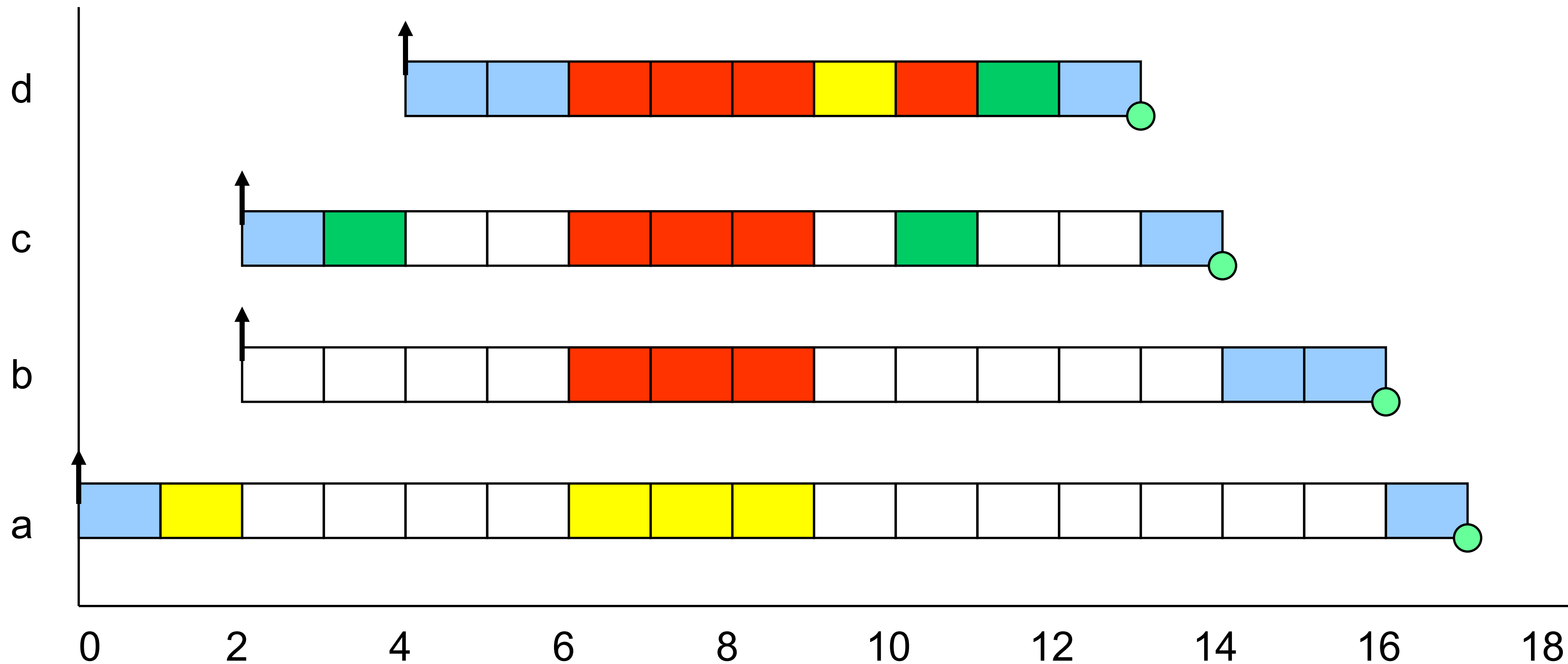
Process	Priority	Execution Sequence	Release Time
a	1	EQQQQE	0
b	2	EE	2
c	3	EVVE	2
d	4	EEQVE	4

Inwersja priorytetów - przykład



Protokół dziedziczenia priorytetów (Priority Inheritance Protocol) - przykład

- **Jeśli proces p jest blokowany przez proces q, wtedy proces q jest wykonywany z priorytetem procesu p.**



Protokół dziedziczenia priorytetów - dyskusja

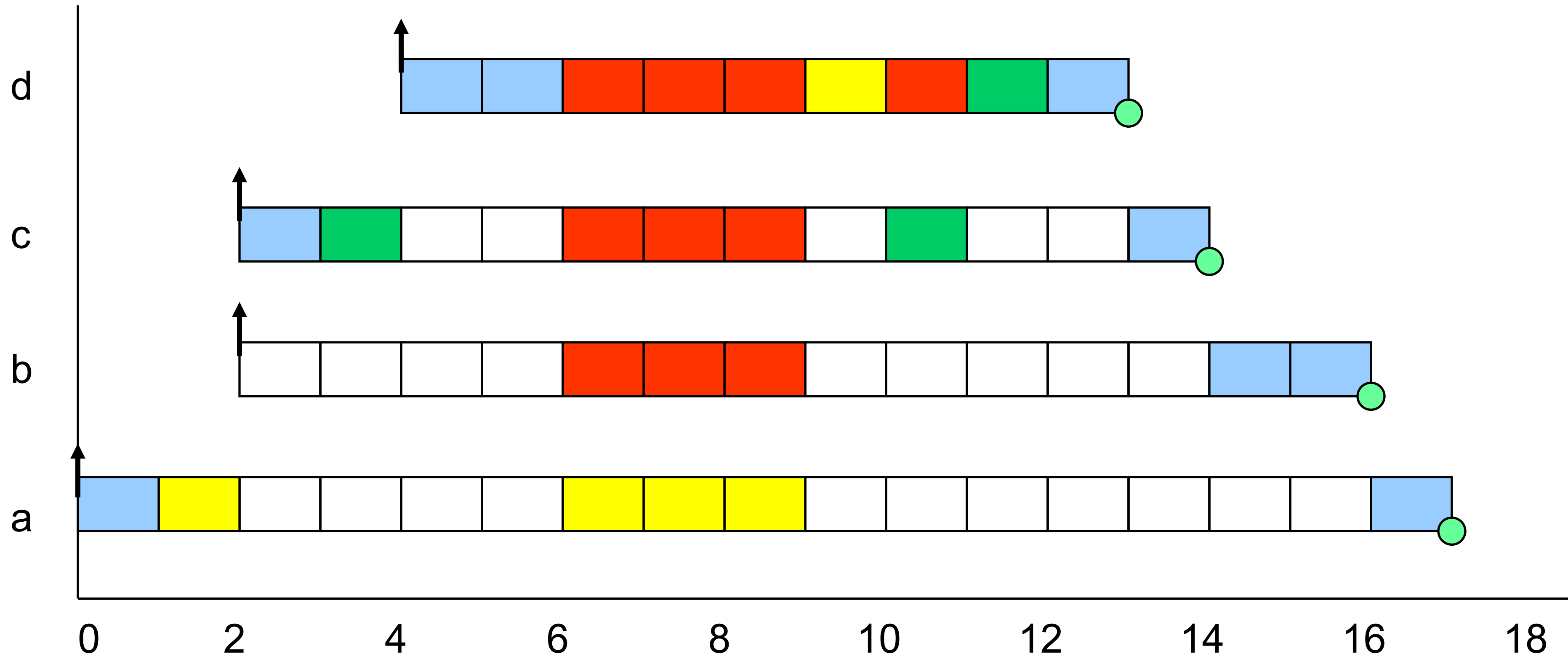
- **Warunek szeregowalności :**

$$\sum_{i=1}^{n}\frac{C_i}{T_i}+\max\left(\frac{B_1}{T_1},\ldots,\frac{B_n}{T_n}\right)\le n(2^{1/n}-1)$$

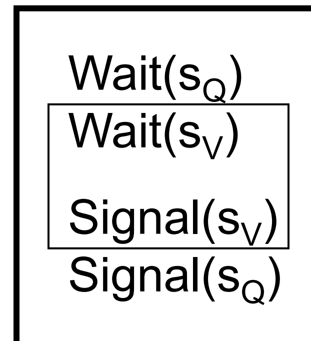
- **Algorytm wyliczania czasów zablokowania podano m. in. w: G. C. Butazzo: Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, Kluver Academic Publishers 1997.**

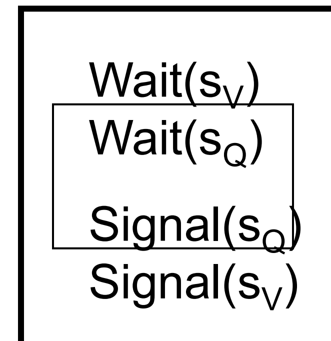
- **Algorytm nie rozwiązuje dwu problemów:**
  - **Łańcucha zablokowań**
  - **Deadlock**

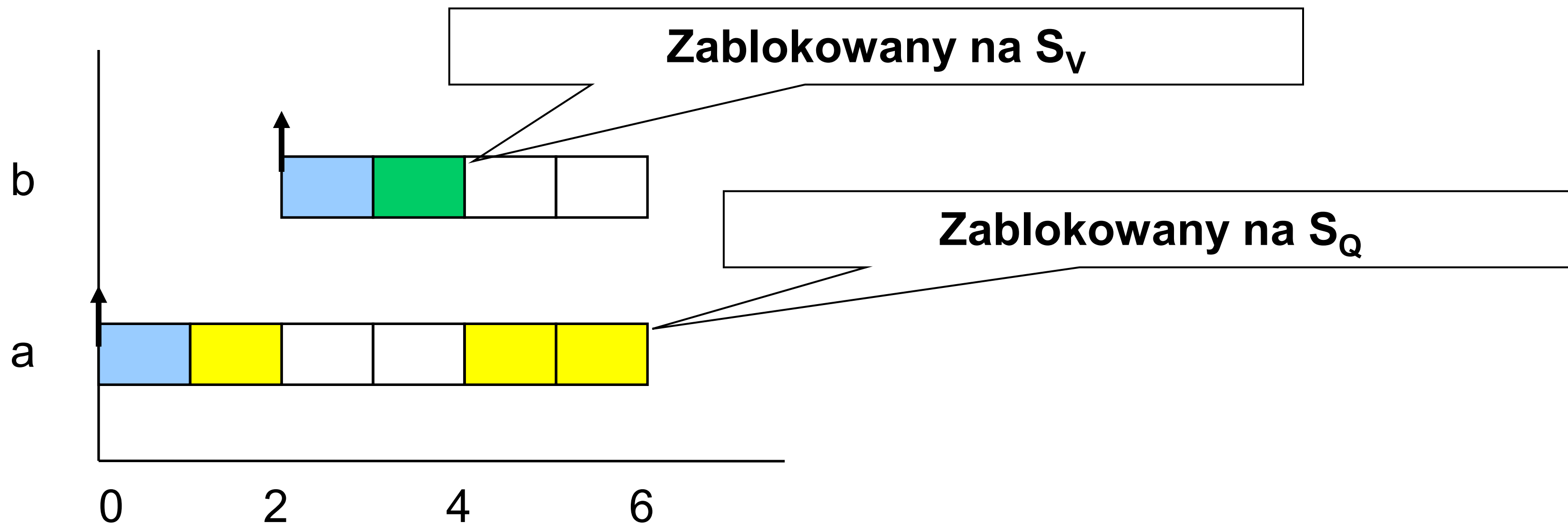
Protokół dziedziczenia priorytetów – łańcuch zablokowań



Protokół dziedziczenia priorytetów – deadlock







Protokół pułapu priorytetów (Priority Ceiling Protocol)

- **Na pojedynczym procesorze:**
  - **Proces o wyższym priorytecie może być zablokowany przez procesy o niższych priorytetach tylko raz podczas swojego wykonywania.**
  - **Nie występuje deadlock**
  - **Nie występuje łańcuch zablokowań**
  - **Wzajemne wykluczanie podczas dostępu do zasobów jest zapewnione**
- **Algorytm:**
  - **Każdy proces posiada przywiązany statycznie określony priorytet**
  - **Każdy zasób posiada określoną statyczną wartość pułapu będącą maksymalnym priorytetem wśród procesów, które go używają**
  - **Dany proces posiada dynamiczny priorytet który określany jest jako maksimum ze statycznie przydzielonego priorytetu i wartości pułapów wszystkich zasobów z których będzie korzystał**
  - **W konsekwencji proces może być zablokowany jedynie na początku swojego wykonywania**
  - **Jeśli proces rozpoczyna swoje wykonywanie, wszystkie zasoby, których potrzebuje muszą być zwolnione. Jeśli nie były, to jakiś inny proces mógł mieć identyczny lub wyższy priorytet i wykonywanie danego procesu było odłożone.**

Protokół pułapu priorytetów - przykład

Process

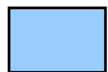
d

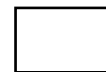
c

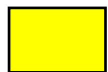
b

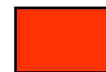
a

0 2 4 6 8 10 12 14 16 18

 Executing

 Preempted

 Executing with Q locked

 Blocked

Executing with V locked

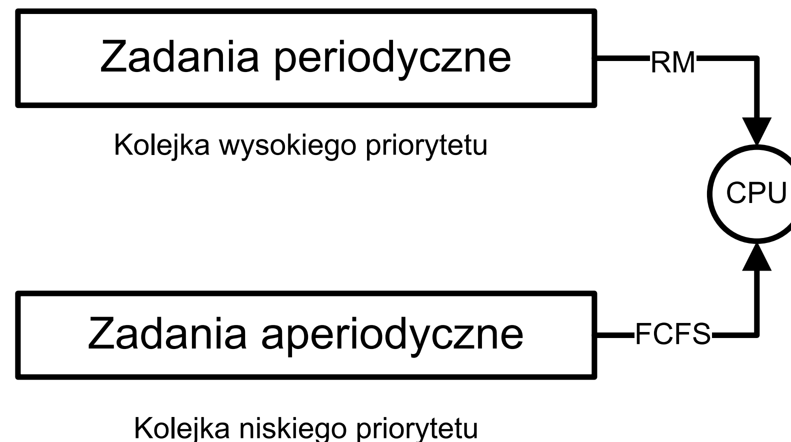
Algorytmy szeregowania zadań periodycznych i aperiodycznych (1)

Cele:

- **Zapewnienie spełnienia ograniczeń czasowych przez zadania periodyczne**
- **Zapewnienie jak najszybszego średniego czasu odpowiedzi dla zadań aperiodycznych o miękkich wymaganiach czasowych**

Najprostsze rozwiązanie:

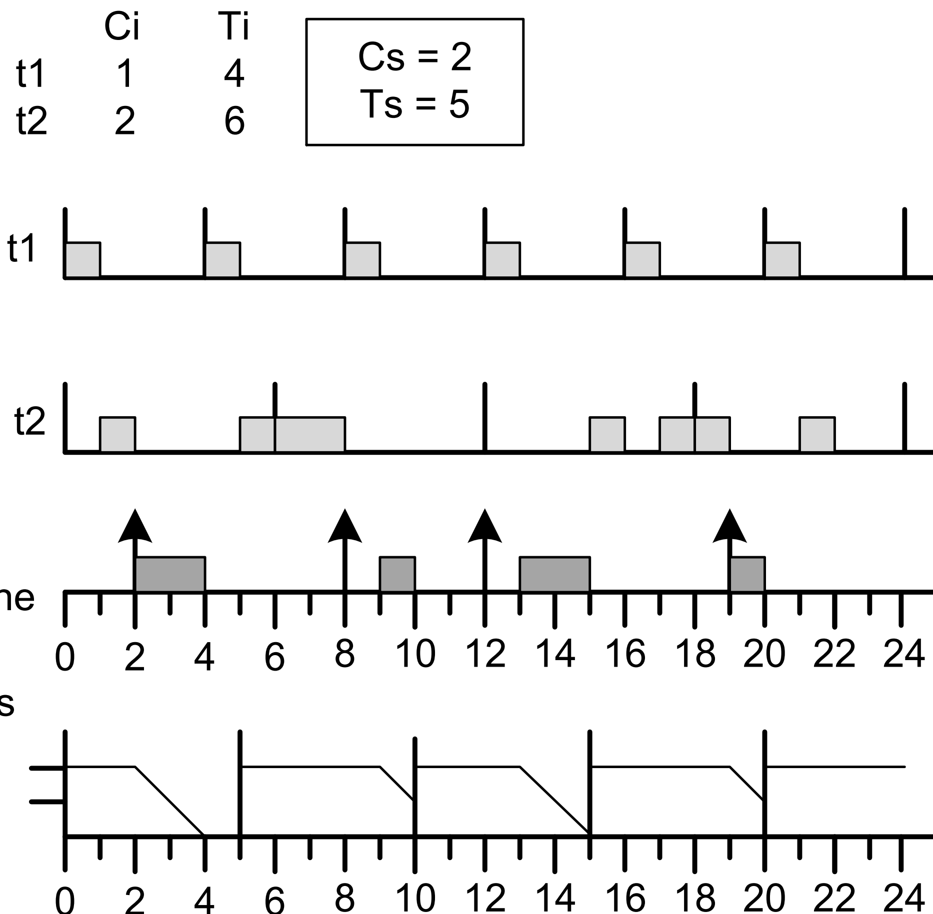
- **Wykonywanie zadań aperiodycznych „niekrytycznych” w tle (tzw. Backgroud Scheduling)**
  - **Kiedy system „wyszereguje” wszystkie zadania krytyczne, to „w wolnym czasie” przydziela czas procesora zadaniom pozostałym**
  - **Zalety: prostota**
  - **Wady: zadania aperiodyczne mogłyby być obsłużone wcześniej**



Algorytmy szeregowania zadań periodycznych i aperiodycznych (2)

Serwer odraczający (ang. Defferable Server) (uproszony serwer sporadyczny – ang. Sporadic Server)

- **Zostaje wyznaczone osobne zadanie periodyczne do obsługi zdarzeń aperiodycznych**
- **Zadanie periodyczne ma przydzielony okres, priorytet oraz „pojemność”**
- **Jeśli w przedziale pomiędzy „aktywacjami” zadania periodycznego pojawia się zadanie aperiodyczne, to zadanie zgodnie ze swoim priorytetem wykonuje całość, lub część obliczeń zadania aperiodycznego**



Algorytmy szeregowania i protokoły przydziału zasobów - uzupełnienie

- Krytyczne zadania aperiodyczne (wywoływane przerwaniami) traktuje się podczas projektowania systemów jako periodyczne o pewnej maksymalnej częstotliwości wznawiania.
- Niekrytyczne zadania aperiodyczne wykonywane są w tle zadań krytycznych, przy czym wprowadza się tzw. serwer nadzorujący wykonywanie tych zadań.
- Przy projektowaniu systemów rozważa się również możliwość przeładowania systemu i konsekwencji takiego stanu.
- Dla systemów wieloprocesorowych istnieją odpowiedniki algorytmów RM i EDF.
- Dla algorytmu EDF nie sformułowano protokołów dziedziczenia i pułapu priorytetów.
- Najprawdopodobniej najlepszym schematem przydziału zasobów jest stosowa strategia zasobów (Stack Resource Policy [Butazzo 1997]).

Rozproszone systemy czasu rzeczywistego

- **Kiedy elementy systemu czasu rzeczywistego są rozproszone**
  - **Zapewnia się spełnienia ograniczeń czasowych na poziomie węzłów.**
  - **Próbuje się stosować mechanizmy komunikacji, które da się oszacować czasowo (np. przemysłowe sieci komputerowe).**
  - **Aplikacje krytyczne czasowo, w których determinizm komunikacji odgrywa kluczową rolę są rzadkością.**

Techniki programowania zadań czasu rzeczywistego – VxWorks (natywny) (1)

- **Priorytety**
  - **Podstawowe szeregowanie – priorytetowe z możliwością wywłaszczania, ew. priorytetowe z round-robin dla zadań o tym samym priorytecie**
  - **Priorytety 100-255 są przeznaczone dla zadań czasu rzeczywistego (100 – najwyższy, 255 – najniższy)**
  - **Priorytety 51-99 są przeznaczone zadań obsługujących sterowniki urządzeń (przerwania)**
  - **Priorytet 50 przeznaczono dla obsługi sieci Ethernet**
  - **Priorytety są nadawane zadaniom w momencie ich uruchamiania**
- **Współdzielenie zasobów:**
  - **Semafory**
    - **Zliczające**
    - **Binarne**
    - **Wzajemnego wykluczania – możliwość uruchomienia protokołu dziedziczenia priorytetów**
    - **Czytelników i pisarzy (polecane do systemów wieloprocesorowych)**
- **Inne mechanizmy komunikacji: kolejki, potoki, zdarzenia, kanały, sygnały.**
- **Liczniki watchdog**

Techniki programowania zadań czasu rzeczywistego – VxWorks (natywny) (2)

- **Tworzenie zadań:**

```
TaskID[T_Test]=
        taskSpawn(proc_table[T_Test].NAME,
                 proc_table[T_Test].BASE_PRIORITY,
                 VX_FP_TASK,
                 proc_table[T_Test].STACK_SIZE,
                 (FUNCPTR)proc_table[T_Test].ENTRY_POINT,
                 0,0,0,0,0,0,0,0,0,0);
```

- **Kod zadania sterowanego czasowo:**

```
void Reg_Od_Po_Prz(void)
{       // What you want to execure once:
        while(1){
                if (wdStart ( WatchdogID[T_REG_Od_Po_Prz],
                             proc_table[T_REG_Od_Po_Prz].PERIOD,
                          (FUNCPTR)Reg_Od_Po_Prz_wd_fun,1) == ERROR){
                logMsg("Watchdog %d Error",T_REG_Od_Po_Prz);
                }
        // CALCULATE ALGORITHM:
        taskSuspend(0);
        }
}
```

Techniki programowania zadań czasu rzeczywistego – VxWorks (natywny) (3)

- **Zapewnienie wzajemnego wykluczania:**
  - Semafory wzajemnego wykluczania (semMCreate( ))
  - Możliwość globalnego zablokowania przerwań (intLock( ) intUnlock( ))
  - Możliwość wyłączenia algorytmu szeregującego (taskLock( ) i taskUnlock( ))

Szeregowanie a POSIX

- POSIX wspiera szeregowanie oparte na priorytetach, posiada mechanizmy wspierające dziedziczenie priorytetów i protokół pułapowy
- Priorytety mogą być ustawiane dynamicznie,
- W obrębie danego priorytetu dostępne są następujące protokoły obsługi zadań:
  - FIFO
  - Round-Robin
  - Sporadic Server
  - OTHER (implementowane przez dany system)
- Dla każdego protokołu minimum 32 poziomy priorytetów muszą być dostępne
- Szeregowanie może być ustalone na poziomie procesu i na poziomie wątku.

Programowanie wielowątkowe POSIX – wstępny przykład (1)

```
#include <pthread.h>

pthread_attr_t attributes;
pthread_t xp, yp, zp;

typedef enum {xplane, yplane, zplane} dimension;

int new_setting(dimension D);
void move_arm(int D, int P);

void controller(dimension *dim)
{
  int position, setting;

  position = 0;
  while (1) {
    setting = new_setting(*dim);
    position = position + setting;
    move_arm(*dim, position);
  };
  /* note, process does not terminate */
}
```

Programowanie wielowątkowe POSIX – wstępny przykład (2)

```
int main() {
  dimension X, Y, Z;
  void *result;

  X = xplane,
  Y = yplane;
  Z = zplane;
  PTHREAD_ATTR_INIT(&attributes);
  /* set default attributes */

  PTHREAD_CREATE(&xp, &attributes, (void *)controller, &X);
  PTHREAD_CREATE(&yp, &attributes, (void *)controller, &Y);
  PTHREAD_CREATE(&zp, &attributes, (void *)controller, &Z);
  PTHREAD_JOIN(xp, &result);
  /* need to block main program */

  exit(-1); /* error exit, the program should not terminate */
}
```

Need JOIN as when a process terminates, all its threads are forced to terminate

SYS_CALL style indicates a call to sys_call with a check for error returns

Funkcje odczytu czasu POSIX - można odczytać dokładnie czas

```
#define CLOCK_REALTIME ...; // clockid_t type

struct timespec {
  time_t tv_sec;   /* number of seconds */
  long   tv_nsec;  /* number of nanoseconds */
};
typedef ... clockid_t;

int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
int clock_getres(clockid_t clock_id, struct timespec *res);

int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
int clock_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
/* nanosleep return -1 if the sleep is interrupted by a */
/* signal. In this case, rmtp has the remaining sleep time */
```

Opóźnienie, zwieszenie procesu POSIX

Funkcje: sleep i nanosleep (można zdefiniować względny albo bezwzględny czas opóźnienia)

Definiowanie cyklicznych zadań POSIX (1)

- **Można tak skonstruować wątek POSIX, aby jego funkcja była wykonywana cyklicznie, co określony przedział czasowy…**

```
#include <signal.h>
#include <time.h>
#include <pthread.h>
void periodic_thread() /* destined to be the thread */
{
  int signum;                     /* signal caught */
  sigset_t set;                   /* signals to be waited for */
  struct sigevent sig;            /* signal information */
  timer_t periodic_timer;         /* timer for a periodic thread */
  struct itimerspec required, old;  /* timer details */
  struct timespec first, period;    /* start and repetition */
  long Thread_Period = ....       /* actual period in nanoseconds */
```

Definiowanie cyklicznych zadań POSIX (2)

```
/* set up signal interface */
sig.sigev_notify = SIGEV_SIGNALS;
sig.sigev_signo = SIGRTMIN; /* for example */

/* allow, e.g., 1 sec from now for system initialisation */
CLOCK_GETTIME(CLOCK_REALTIME, &first);  /* get current time */
first.tv_sec = first.tv_sec + 1;
period.tv_sec = 0;          /* set repetition value to period*/
period.tv_nsec = Thread_Period;
required.it_value = first;  /* initialise timer details */
required.it_interval = period;

TIMER_CREATE(CLOCK_REALTIME, &sig, &periodic_timer);
SIGEMPTYSET(&set);          /* initialise signal set to null */
SIGADDSET(&set, SIGRTMIN);  /* only allow timer interrupts*/
TIMER_SETTIME(periodic_timer, 0, &required, &old);
```

Definiowanie cyklicznych zadań POSIX (3)

```
  /* enter periodic loop */
  while(1) {
    SIGWAIT(&set, &signum);
    /* code to be executed each period here */
  }
}

int init()
{
  pthread_attr_t attributes;      /* thread attributes */
  pthread_t PT;                   /* thread pointer */

  PTHREAD_ATTR_INIT(&attributes); /* default attributes */
  PTHREAD_CREATE(&PT, &attributes,
                 (void *) periodic_thread, (void *)0);
}
```

Zadania w języku Ada – przykład wstępny

```
procedure Example1 is -- zadanie może być definiowane
                      -- w procedurze lub pakiecie lub zadaniu
   task type A_Type;  -- zdefiniowanie typu zadaniowego
   task type B_Type;
   A: A_Type;         -- utworzenie zadań
   B: B_Type;

   task body A is   -- lokalne deklaracje dla zadania A
   begin            -- sekwencja instrukcji dla zadania A
   end A;

   task body B is   -- lokalne deklaracje dla zadania B
   begin            -- sekwencja instrukcji dla zadania B
   end B;
begin  -- Zadania A i B rozpoczną swoje wykonywanie
       -- przed pierwszą instrukcją sekwencji
       -- instrukcji należących do procedury.
       -- System składa się z 3 (!) współbieżnych procesów:
       -- zadań A i B oraz procedury Example1
end Example1;       -- procedura zakończy się dopiero
                    -- gdy zakończą działanie oba zadania
```

Ada – programowanie zadań cyklicznych

```
-- Zadanie cykliczne – eliminacja niekorzystnych składników
-- czasowych
task type Periodic;
   Seconds:   constant Duration := 1.0;
   Period:    constant Duration := 0.5 * Seconds;
   Offset:    constant Duration := 0.2 * Seconds;
   Next_Time : Time := Calendar.Clock + Offset;

task body Periodic is
begin
   loop
      delay until Next_Time;
      -- lub: delay Next_Time – Calendar.Clock;
      Put("Nastepny Tick"); New_Line(2);
      Next_Time := Next_Time + Period;
   end loop;
end Periodic;
```

Zadania w języku Ada – mechanizm synchronizacji

```
-- Instrukcja accept:
with Text_IO; use Text_IO;

procedure spotkanie is
   task type odbiornik is
      entry GET; -- zdefiniowanie wejścia
   end Odbiornik;
   task type Nadajnik;

   task body Odbiornik is
   begin
      loop   -- obsługa wejścia
             accept GET do
                   put("Odebrano komunikat");
                   New_Line;
             end GET;
      end loop;
   end Odbiornik;

   Odb : Odbiornik; -- uruchomienie zadania Odb
```

Zadania w języku Ada – mechanizm synchronizacji

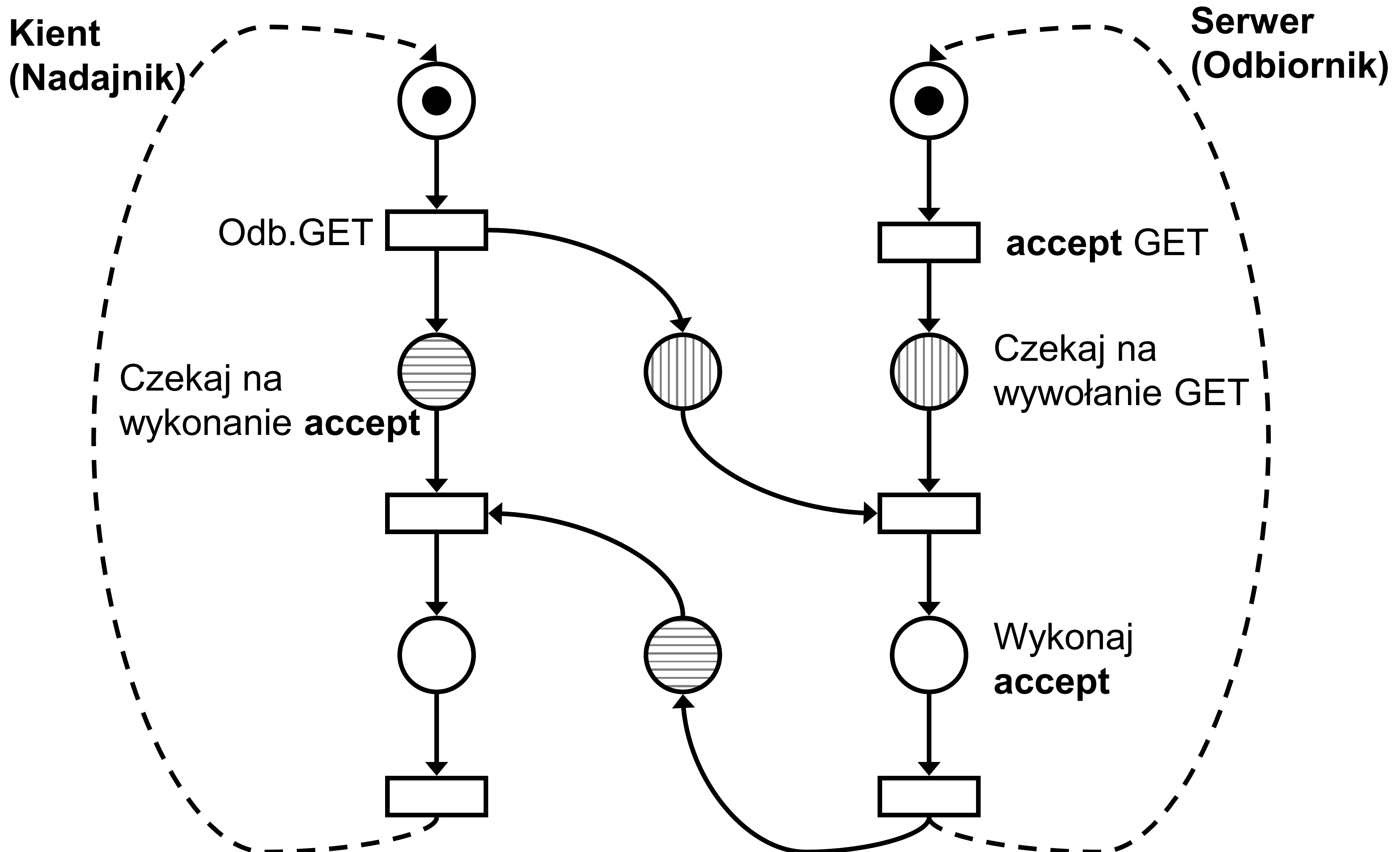
```
-- Instrukcja accept (cd.):

   task body Nadajnik is
   c: Character;
   begin
      loop
            Get(c);
            exit when C='?';
            Odb.GET;      -- Konieczne było wcześniejsze
                          -- utworzenie zadania Odb.
      end loop;
   end Nadajnik;

   Nad : Nadajnik; -- uruchomienie zadania Nad
begin
   Put("Uruchomiono zadania");
   New_Line;
end Spotkanie;
```

Zadania w języku Ada – mechanizm synchronizacji

- Mechanizm spotkania (*randezvous*):



Zadania w języku Ada – przykład – semafor

```
with Text_IO; use Text_IO;
with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io;   use ada.float_text_io;
with Ada.Calendar; use  Ada.Calendar;

procedure main is
-- Specyfikacja semafora
task type SEMAPHORE is
   entry ACCES_TO;
   entry FREE;
end SEMAPHORE;

-- Implmentacja semafora
task body SEMAPHORE is
begin
   loop
      accept ACCES_TO;
      accept FREE;
   end loop;
end SEMAPHORE;

Drukarka : Integer; -- zasób współdzielony
```

Zadania w języku Ada – przykład - semafor

```
Semafor : SEMAPHORE; -- uruchomienie semafora
-- zadanie z parametrami wejściowymi:
task type T(Init: Integer; Del: Integer);

task body T is
    dana : Integer;
begin
    loop
        Dana := Init;
        Semafor.ACCES_TO;    -- początek sekcji krytycznej
                Drukarka:=Dana;
                Put(Drukarka);
                New_Line;
        Semafor.FREE;        -- koniec sekcji krytycznej
        delay Duration(Del);
    end loop;
end T;

Task1: T(2,1);
Task2: T(4,2);

begin  null;end;
```

Zadania w języku Ada – priorytety, szeregowanie, zasoby

- **Odpowiednimi poleceniami języka Ada można ustawić:**
  - **Priorytety zadań < pragma Priority(10); >**
  - **Pułapowy protokół przydzielania zasobów < pragma Locking_Policy(Ceiling_Locking); >**
  - **Algorytmy szeregowania:**
    - **FIFO_Within_Priority --(Ada95)**
    - **Non_Preemptive_FIFO_Within_Priorities --(Ada2005)**
    - **Round_Robin_Within_Priorities --(Ada2005)**
    - **EDF_Across_Priorities --(Ada2005)**
- **Współdzielone zasoby można zorganizować w postaci „obiektów chronionych”**

Obiekty chronione komunikacja przez zmienne dzielone

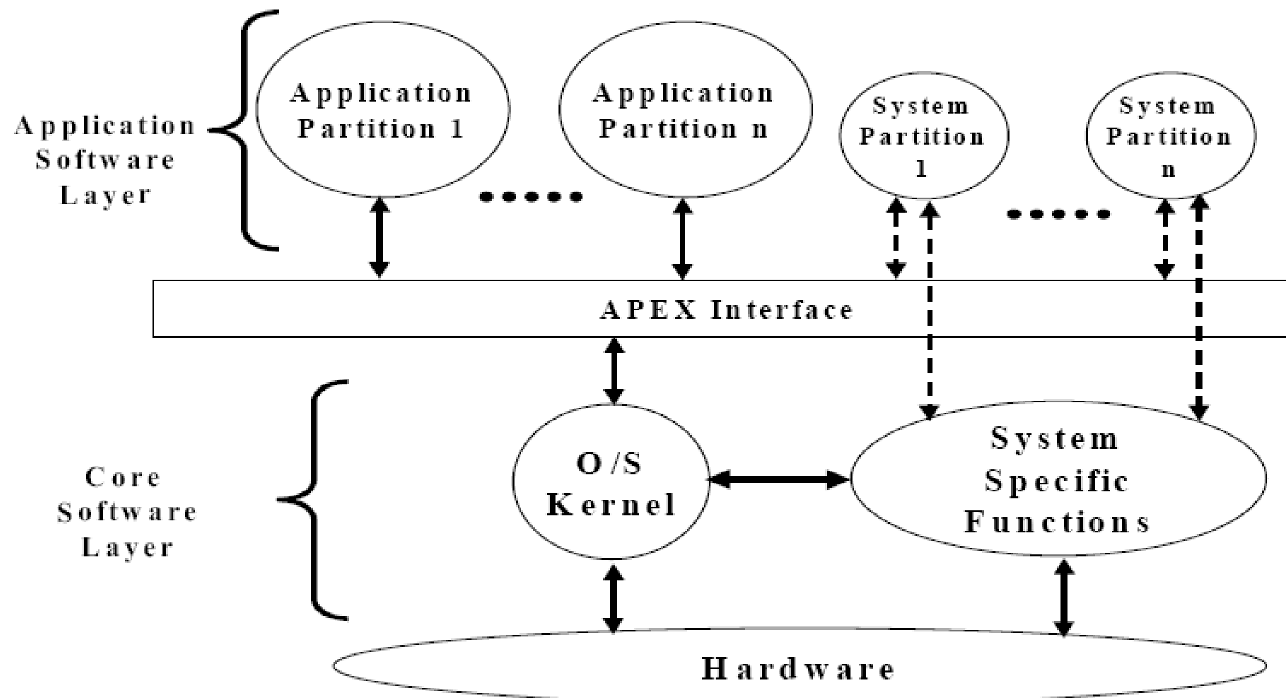
- **Obiekty chronione stanowią konstrukcję zamykająca wewnątrz dane (o dowolnej strukturze), które są dostępne z zewnątrz przez udostępnianie operacji – podprogramów lub wejść.**
- **Wywołanie operacji zewnętrznych obiektu chronionego jest synchronizowane w celu zapewnienia poprawności (wzajemne wykluczanie) współbieżnego przetwarzanie danych wewnątrz obiektu.**
- **Obiekt chroniony odpowiada monitorowi.**
- **Obiekt chroniony w odróżnieniu od zadania jest wykonywany wyłącznie, jeśli wywołana zostanie pewna jego operacja, po zakończeniu której obiekt jest gotowy do realizacji następnej operacji Nie występuje oddzielny własny wątek wykonywania obiektu; synchronizacja dotyczy jedynie zapewnienia wzajemnego wykluczania odpowiednich operacji.**

Semantyka obiektu chronionego

- Typ chroniony zamyka we wnętrzu dane, które są dostępne z zewnątrz przez wywołanie specyfikowanych w interfejsie funkcji, procedur lub wejść. Spełnione są przy tym następujące warunki:
  - Możliwe jest współbieżne wykonywanie wielu wywołanych funkcji obiektu chronionego. Funkcja ma możliwość jedynie czytania obiektu chronionego. Wykonywanie funkcji jest realizowane na zasadzie wyłączności względem procedur i wejść.
  - Procedury obiektu chronionego mogą być wykonywane na zasadzie wyłączności, tzn. jeśli wykonywana jest procedura obiektu chronionego, to jest ona jedyną jednostką wykonywaną w danym momencie.
  - Wykonanie wołanego wejścia możliwe jest na zasadzie wyłączności względem innych elementów (podobnie jak procedury). Możliwość wykonywania wejścia jest dodatkowo ograniczana przez wyrażenie logiczne („Warunek Bariery”). Oznacza to, że oprócz warunków synchronizacji, konieczne jest również spełnienie tego wyrażenia logicznego. Odpowiada to konstrukcji warunkowego obszaru krytycznego.

Specyfikacja ARINC 653P1-2 - partycje

- Nowe podejście do architektury systemu operacyjnego czasu rzeczywistego i zasad tworzenia aplikacji:
  - Partycje - moduły programowe
    - umożliwiające przestrzennie i czasowe wyizolowanie aplikacji
    - komunikujące się z innymi komponentami systemu poprzez ustalony interfejs – APEX (Application/EXecutive)



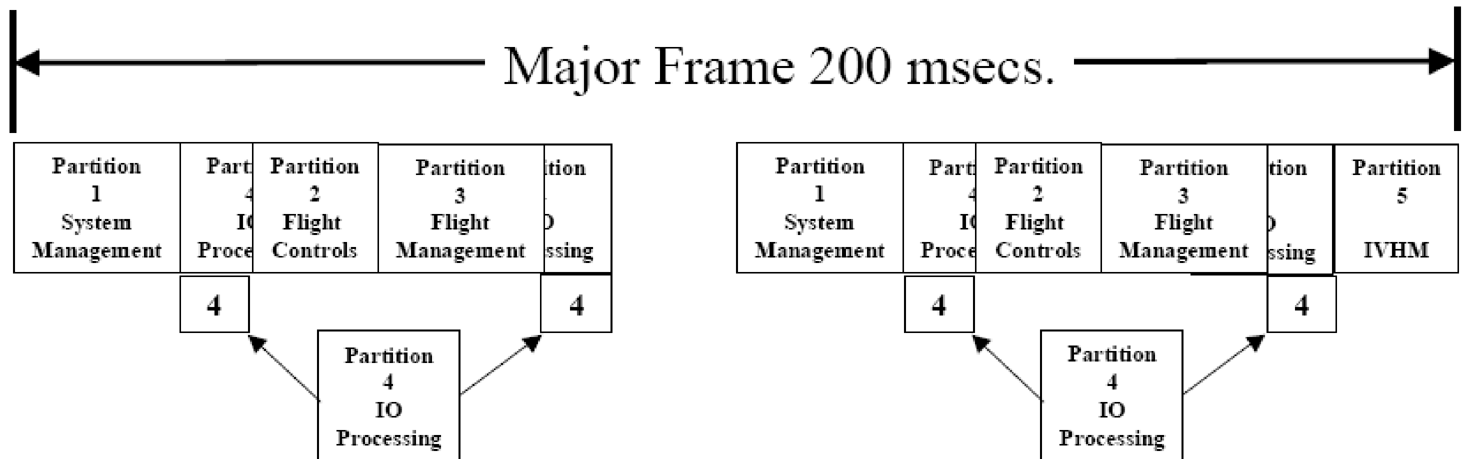
Specyfikacja ARINC 653P1-2 - architektura sprzętowo-programowa (1)

- Każdy z modułów sprzętowych może mieć 1 lub więcej procesorów,
- Struktura sprzętowa może wymagać modyfikacji jądra systemu ale nie interfejsu APEX,
- Procesy uruchamiane na 1 partycji (tworzące aplikację) muszą być uruchamiane na 1 procesorze,
- Oprogramowanie aplikacji powinno być przenośne między procesorami bez modyfikacji interfejsu z systemem operacyjnym,
- Procesy uruchamiane na 1 partycji mogą być wykonywane współbieżnie,
- Komunikacja pomiędzy aplikacjami (partycjami) odbywa się za pomocą PORTÓW.
  - Aplikacja nie dysponuje informacją, gdzie znajduje się adresat informacji,
  - Kanały komunikacyjne pomiędzy portami są definiowane na innym poziomie opisu modułu.

Specyfikacja ARINC 653P1-2 - architektura sprzętowo-programowa (2)

- W module przewidziano osobny składnik – „monitor zdrowia" – odpowiedzialny za monitorowanie i wykrywanie błędów i uszkodzeń na poziomie sprzętu, aplikacji i systemu operacyjnego,
- Czasowa izolacja realizowana jest przez założenie głównej ramy czasowej w której dla poszczególnych partycji są przewidziane tzw. okna czasowe.

Window ID	1.1	4.1	2.1	3.1	4.2	1.2	4.3	2.2	3.2	4.4	5.1
Partition	P1	P4	P2	P3	P4	P1	P4	P2	P3	P4	P5
window offset	0.000	0.020	0.030	0.040	0.070	0.100	0.120	0.130	0.140	0.170	0.180
window duration	0.020	0.010	0.010	0.030	0.010	0.020	0.010	0.010	0.030	0.010	0.020

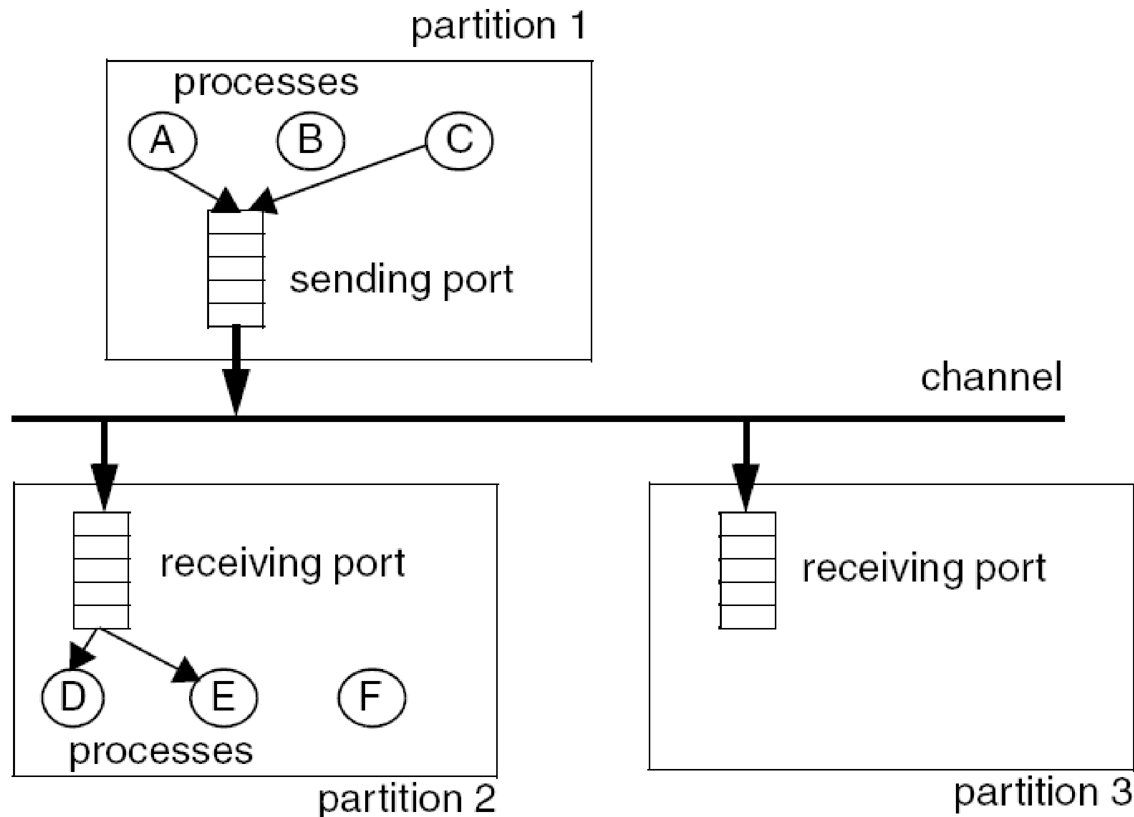


Specyfikacja ARINC 653P1-2 – interfejs APEX (1)

- Zarządzanie partycjami
  GET_PARTITION_STATUS,
  SET_PARTITION_MODE…
- Zarządzanie procesami
  CREATE_PROCESS, START, SET_PRIORITY…
- Zarządzanie czasem
  PERIODIC_WAIT, REPLENISH…
- Brak zarządzania pamięcią – wszystkie obiekty muszą być statyczne i utworzone przed uruchomieniem partycji

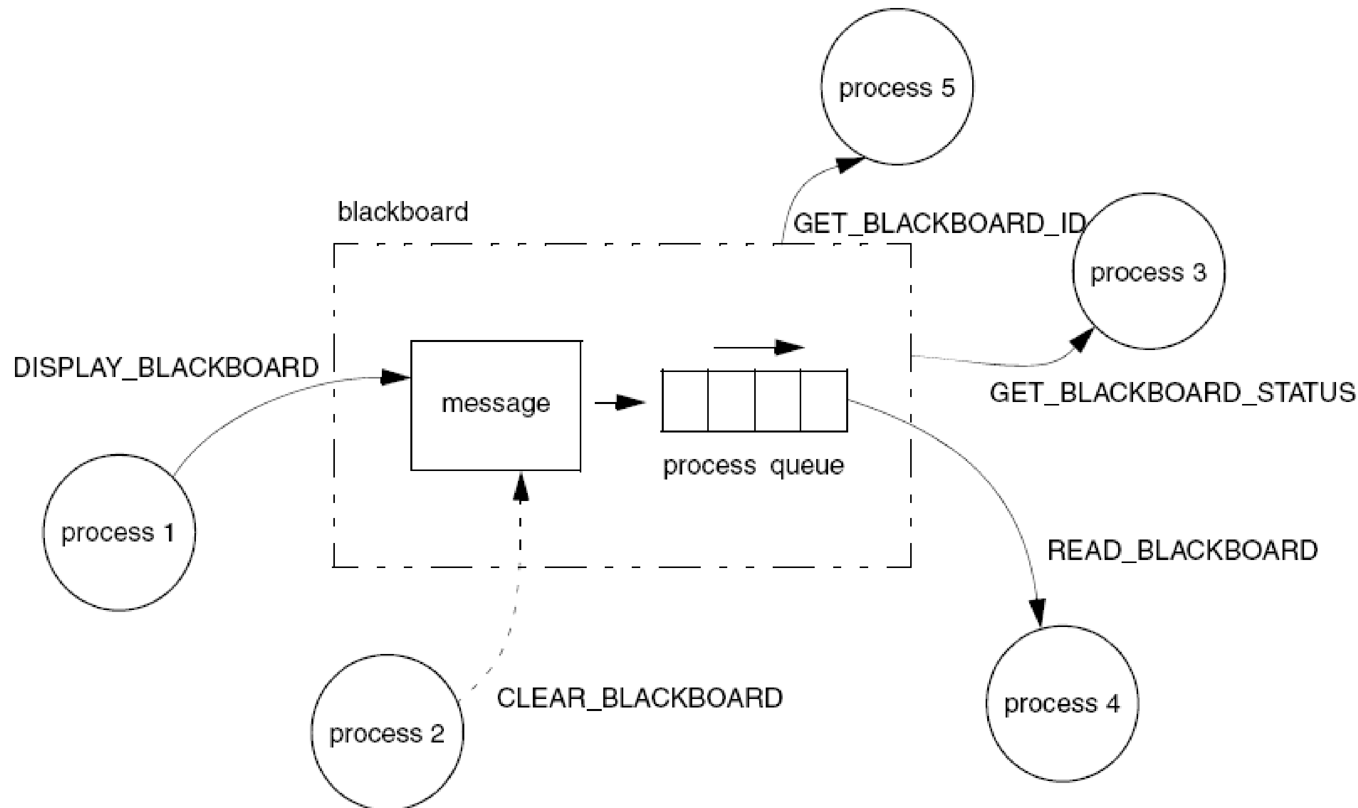
Specyfikacja ARINC 653P1-2 – interfejs APEX (2)

- Komunikacja pomiędzy partycjami – porty i kanały CREATE_SAMPLING_PORT, WRITE_SAMPLING_MESSAGE, READ_SAMPLING_MESSAGE…



Specyfikacja ARINC 653P1-2 – interfejs APEX (3)

- Komunikacja i synchronizacja w obrębie partycji – zdarzenia, semafory, bufory (kolejki), tablice (monitory) CREATE_BLACKBOARD, DISPLAY_BLACKBOARD, READ_BLACKBOARD …



Specyfikacja ARINC 653P1-2 – interfejs APEX (4)

- Procedury zgłaszania błędów (wyjątków) do modułu „monitora zdrowia”
  CREATE_ERROR_HANDLER,
  GET_ERROR_STATUS,
  RAISE_APPLICATION_ERROR …

- Konfiguracja systemu
  - Dostarczyciele aplikacji
  - Integrator aplikacji

System sterowania kątem pochylenia samolotu zgodny z ARINC 653/664