Systemy czasu rzeczywistego

Ostatnia modyfikacja: 02.06.2021

Wprowadzenie

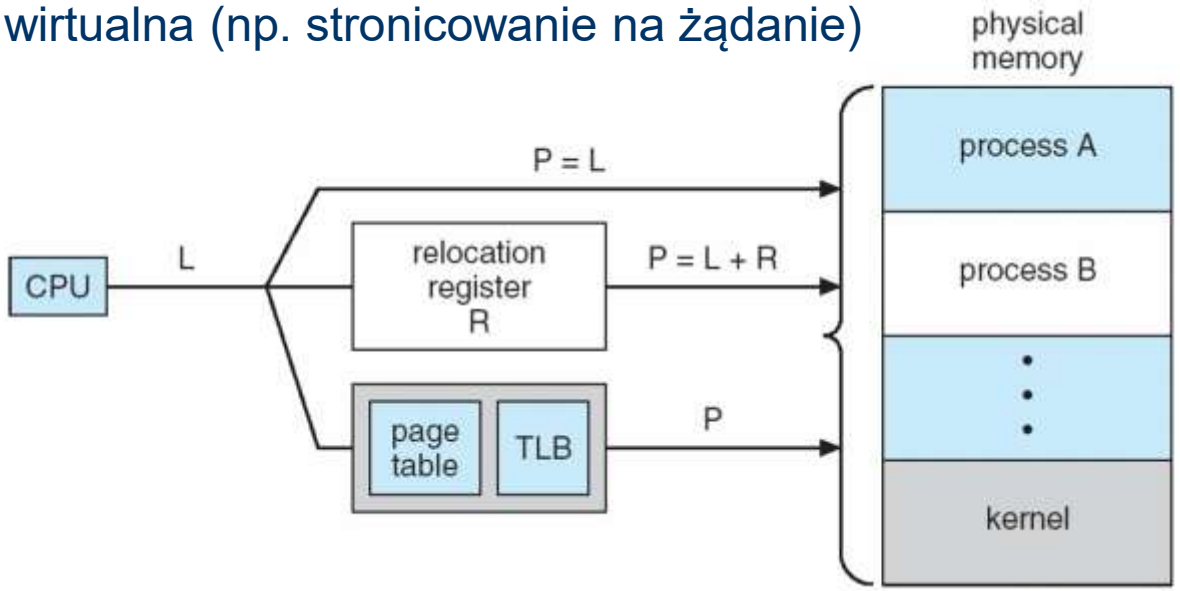
- **System czasu rzeczywistego**, RT (*real-time system*) zapewnia, że wyniki działania procesu są wyznaczone w założonym okresie czasu (*specified deadline period*)
  - **System wbudowany** (*embedded system*) urządzenie obliczeniowe będące częścią większego systemu (np. samochodu, samolotu, pralki)
  - **System krytyczny pod względem bezpieczeństwa** (*safety-critical system*) to system RT, którego awaria może powodować katastrofalne skutki.
- **Rygorystyczny system czasu rzeczywistego** (*hard real-time system*) gwarantuje wykonanie zadania w założonym przedziale czasu.
- **Łagodny system czasu rzeczywistego** (*soft real-time system*) realizuje pierwszeństwo zadań RT względem pozostałych, ale nie daje gwarancji czasowych.

Wymagania dla systemów RT a PC

- Większość systemów RT nie zapewnia funkcjonalności znanej z systemów biurkowych (PC). Niektóre powody:
  - Systemy RT są zwykle dedykowane do zastosowań
  - Systemy RT często nie wymagają interfejsu z użytkownikiem
  - Funkcjonalność systemów biurkowych (PC) wymaga dużo większych zasobów sprzętowych niż te dostępne dla systemów RT.
  - Rozwiązania ekonomiczne zwykle wprowadzają niedeterministyczne opóźnienia (*latency*)
  - Procesy RT muszą mieć kontrolę nad kolejnością wykonywania współpracujących zadań, by spełnić założone wymagania czasowe.

Pamięć wirtualna w systemach RT

- Odwzorowanie adresów pamięci może być realizowane następująco:
  - (1) W trybie adresowania rzeczywistego (real-addressing mode) procesor posługuje się bezpośrednio adresami fizycznymi pamięci
  - (2) W trybie relokacji – wartość rejestru przesunięcia jest dodawana do adresu wytworzonego przez CPU
  - (3) Pełna pamięć wirtualna (np. stronicowanie na żądanie)



- W systemach RT wprowadzono możliwość unieruchamiania bloków pamięci (*memory locking*), aby wyeliminować niedeterminizm realizacji odniesień do pamięci.

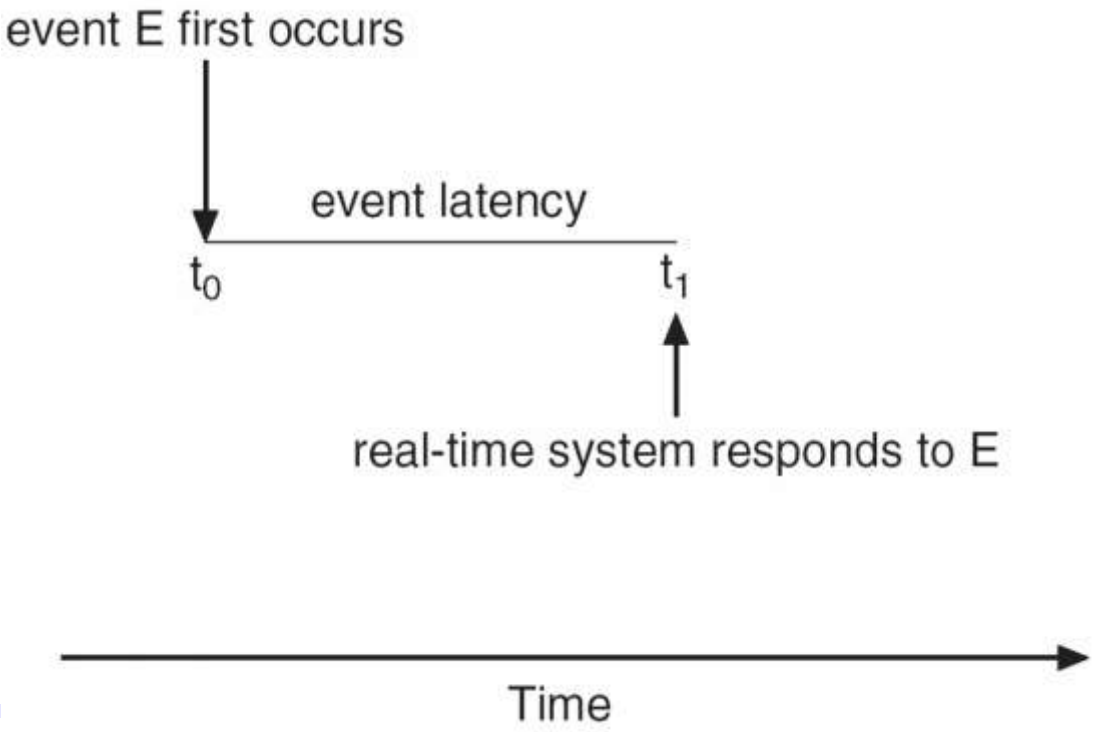
Implementacja systemów operacyjnych RT

- Systemy operacyjne RT **muszą** posiadać:

  (1) **wywłaszczalne planowanie** przydziału procesora, oparte na priorytetach,

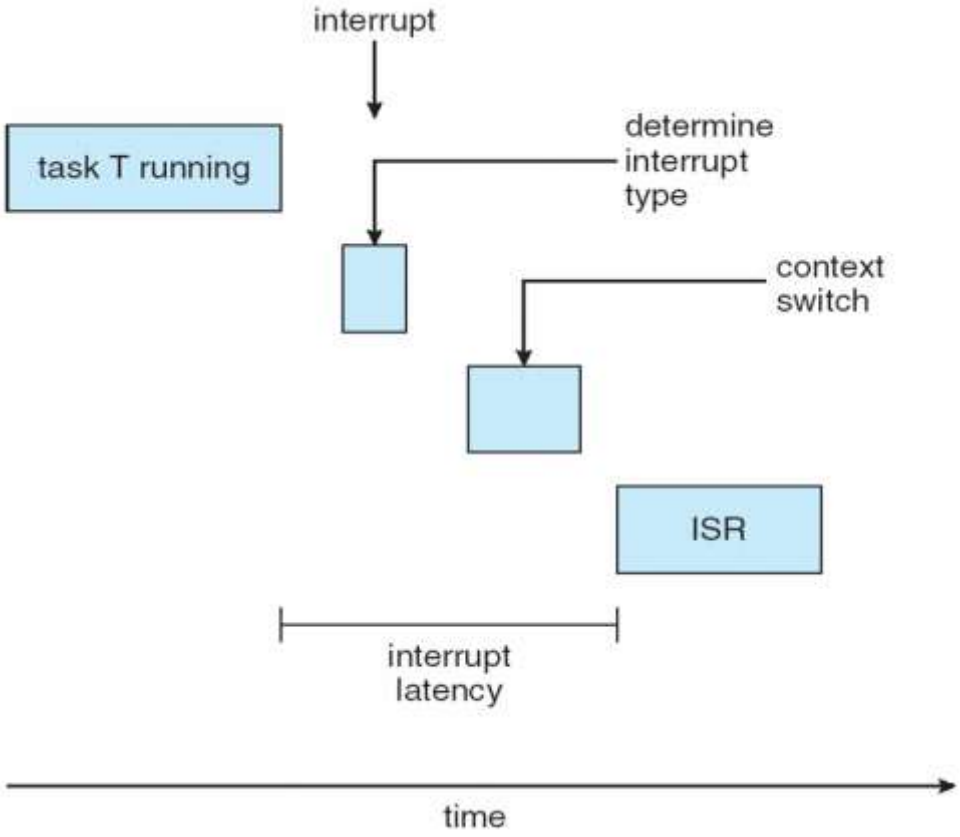
  (2) **wywłaszczalne jądro**

  (3) minimalizowane **opóźnienie reakcji** (*latency*)



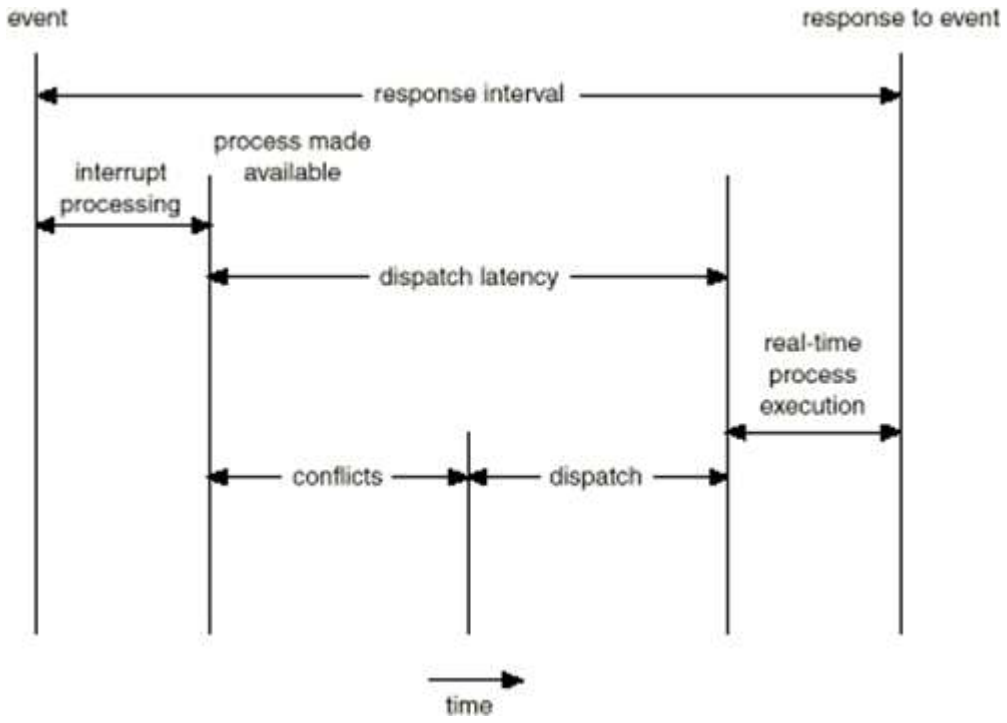
Czas reakcji na przerwanie

- **Czas reakcji na przerwanie** (*interrupt latency*) – czas, który upływa pomiędzy przerwaniem, a reakcją systemu na niego (obsługą przerwania).
- W systemach RT czas reakcji musi być **ograniczony**, aby zachowanie systemu było **deterministyczne**.



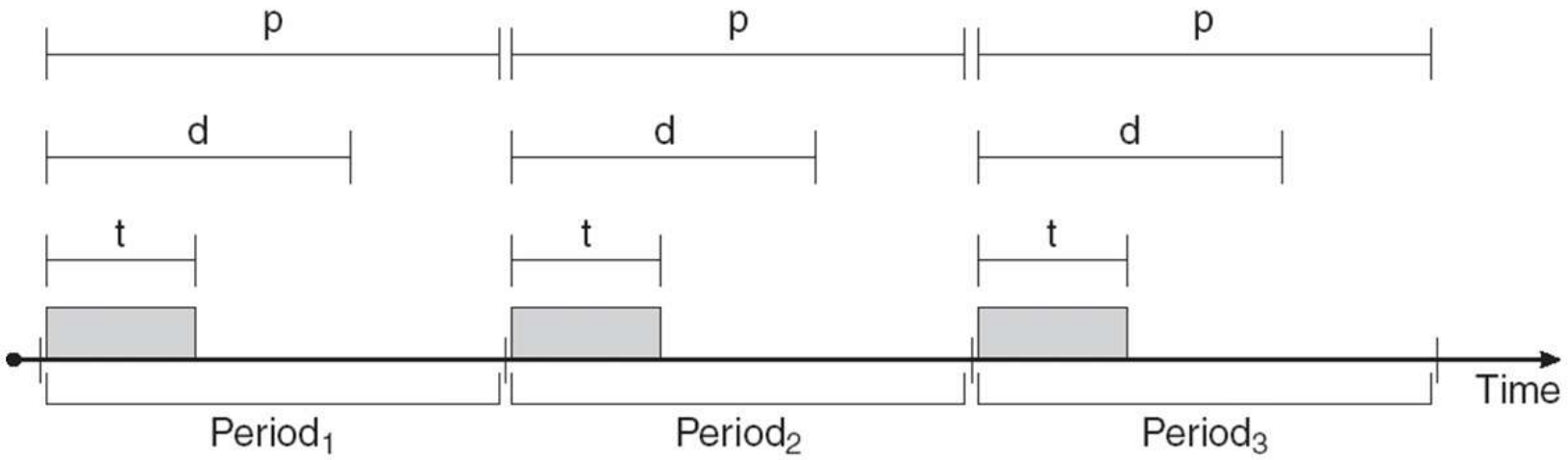
Opóźnienie ekspedycji

- **Opóźnienie ekspedycji** (*dispatch latency*) – czas pomiędzy rozpoczęciem reakcji systemu operacyjnego na przerwanie a rozpoczęciem działania procedury obsługi zdarzenia. Opóźnienie to może być zmniejszone przez:
  - **Punkty wywłaszczania** (*preemption points*) w długich funkcjach systemowych
  - **Wywłaszczalne jądro**



Planowanie RT przydziału procesora

- Zadania aperiodyczne: muszą się rozpocząć lub zakończyć przed terminem
- Zadania okresowe wymagają dostępu do procesora w powtarzających się przedziałach czasu. Cechy:
  - ***p*** – okres powtarzania (*period*)
  - ***d*** - ostateczny termin zakończenia przetwarzania w danym okresie (*deadline*)
  - ***t*** - czas przetwarzania (*processing time*)



- Warunek konieczny wykonalności planu szeregowania: $\sum_{i=1}^{N} \frac{t_i}{p_i} \leq 1$

Planowanie priorytetowe: P_2 ma priorytet wyższy od P_1

p_1=50, p_2=100 – okresy

t_1=20, t_2=35 – czasy przetwarzania

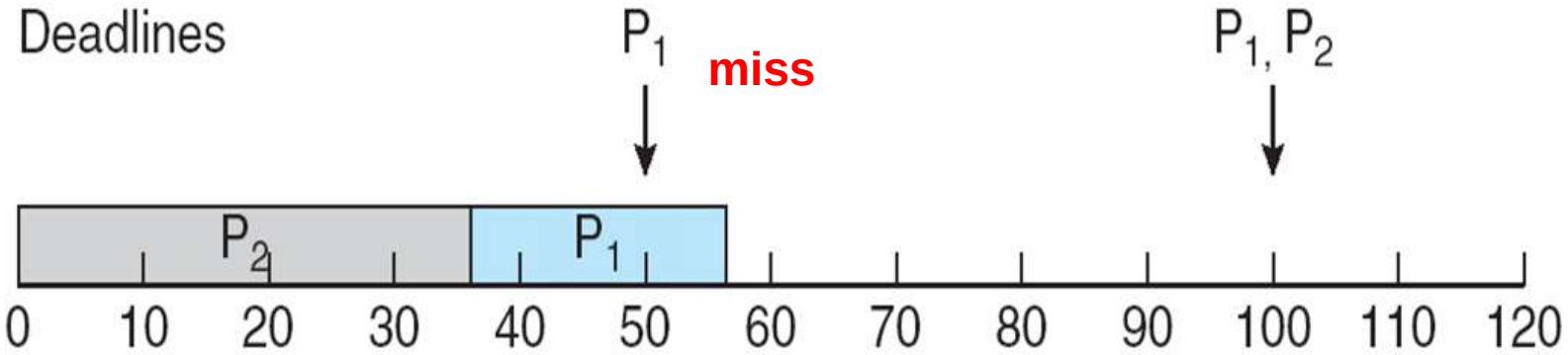
Wykorzystanie procesora:

P_1: t_1/p_1=20/50=0.4

P_2: t_2/p_2=35/100=0.35

Ponieważ: 0.4+0.35=0.75<1 -> mogłoby się wydawać, że uda się spełnić wymagania czasowe (deadlines)

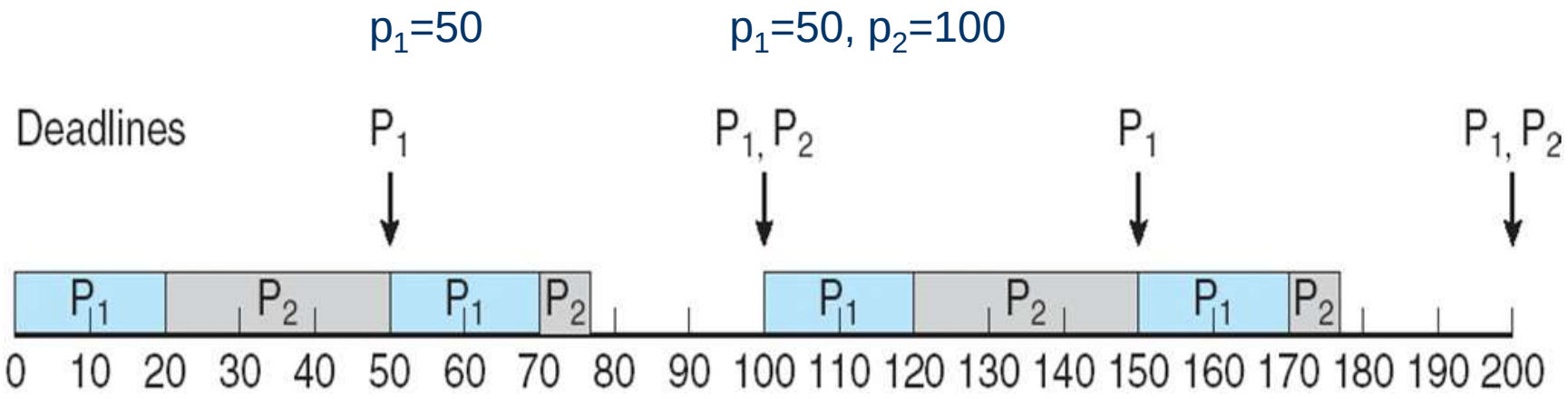
Niespodzianka (?) – **niedotrzymanie terminu** (*miss*) przez szeregowanie priorytetowe z przyjętymi statycznie priorytetami (P_2 ma wyższy priorytet niż P_1)



Szeregowanie częstotliwościowo monotoniczne (Rate Monotonic Scheduling, RMS)

- **Algorytm częstotliwościowo-monotoniczny** (RMS) zakłada, że priorytet jest odwrotnością okresu
  - Szybsze powtórzenia = wyższy priorytet;
  - Dłuższe okresy powtórzeń = niższy priorytet

- Proces P_1 powinien mieć wyższy priorytet niż P_2.(bo t_1=20; t_2=35)



- W tym przypadku algorytm RMS wytworzył dopuszczalny plan przydziału procesora. W jakich sytuacjach tak będzie?

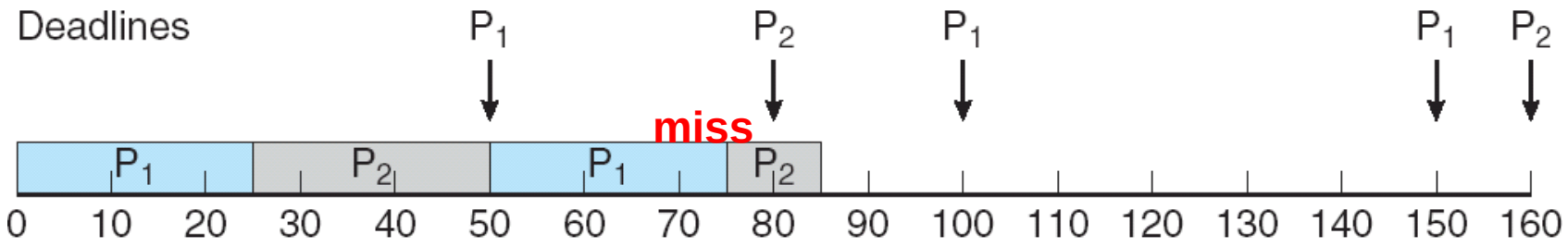
Niepowodzenia szeregowania dla algorytmu RMS

- Algorytm RMS **jest optymalny** w tym sensie, że jeśli zestaw procesów nie może być przez niego uszeregowany, to nie może być uszeregowany przez żaden inny algorytm korzystający ze statycznych priorytetów.
- Przykład.

p_1=50, p_2=80 – okresy; t_1=25, t_2=35 – czasy przetwarzania

Wykorzystanie procesora: P_1: t_1/p_1=25/50=0.5; P_2: t_2/p_2=35/80=0.44

WK istnienia planu dopuszczalnego: 0.5+0.44=0.94<1. Czy RMS uzyska taki plan?

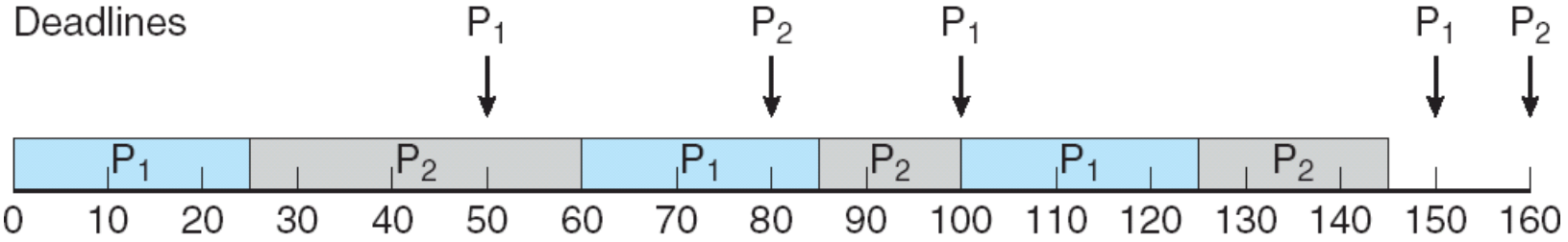


Uwagi:

- W najgorszym przypadku wykorzystanie procesora dla N procesów wynosi tylko $N(2^{1/N}-1)$. Dla N=2 mamy 0.83<0.94 – co tłumaczy niepowodzenie powyżej. $\lim_{N\to\infty} N(2^{1/N}-1) = \ln(2) \approx 0.693$
- Algorytm RMS (tak jak inne priorytetowe) jest podatny na efekt **inwersji priorytetów**.

Algorytm szeregowania wg najwcześniejszego terminu (EDF)

- EDF = **Earliest Deadline First**
- Priorytety są przydzielane wg kolejności najbliższych terminów zakończenia przetwarzania. Im bliższy termin, tym wyższy priorytet procesu



- W przeciwieństwie do algorytmu RMS algorytm EDF nie zakłada, że aktywności procesów są okresowe ani też że w każdym okresie proces wymaga tego samego kwantu czasu procesora.
- EDF wymaga jednak, że algorytm zastanie powiadomiony o terminie zakończenia przetwarzania jak tylko proces stanie się gotowy do wykonania.

POSIX - szeregowanie RT w P-wątkach

Standard POSIX.1b

- Wykonywanie każdego wątku zależy od **polityki szeregowania** (*scheduling policy*) oraz **priorytetu wątku**
- Wątki mogą współzawodniczyć o czas procesora
  - w obrębie procesu (*process contention scope*)
  - globalnie (*system contention scope*)
- Z każdym możliwym priorytetem jest związana **kolejka wątków gotowych do wykonania**.
- **Polityka szeregowania** wyznacza reguły przemieszczania wątku w ramach kolejki lub pomiędzy kolejkami. Polityka planowania może dotyczyć (pod)**zakresu priorytetów**.
- Wątek gotowy, który jest pierwszy w niepustej kolejce wątków o **najwyższym priorytecie**, zostaje wątkiem wykonującym się – niezależnie od tego jakiej polityce szeregowania podlega.

POSIX - szeregowanie RT

POSIX 1003.1 Standard wymaga implementacji (co najmniej) czterech klas (polityk) planowania przydziału procesora (*policies*)

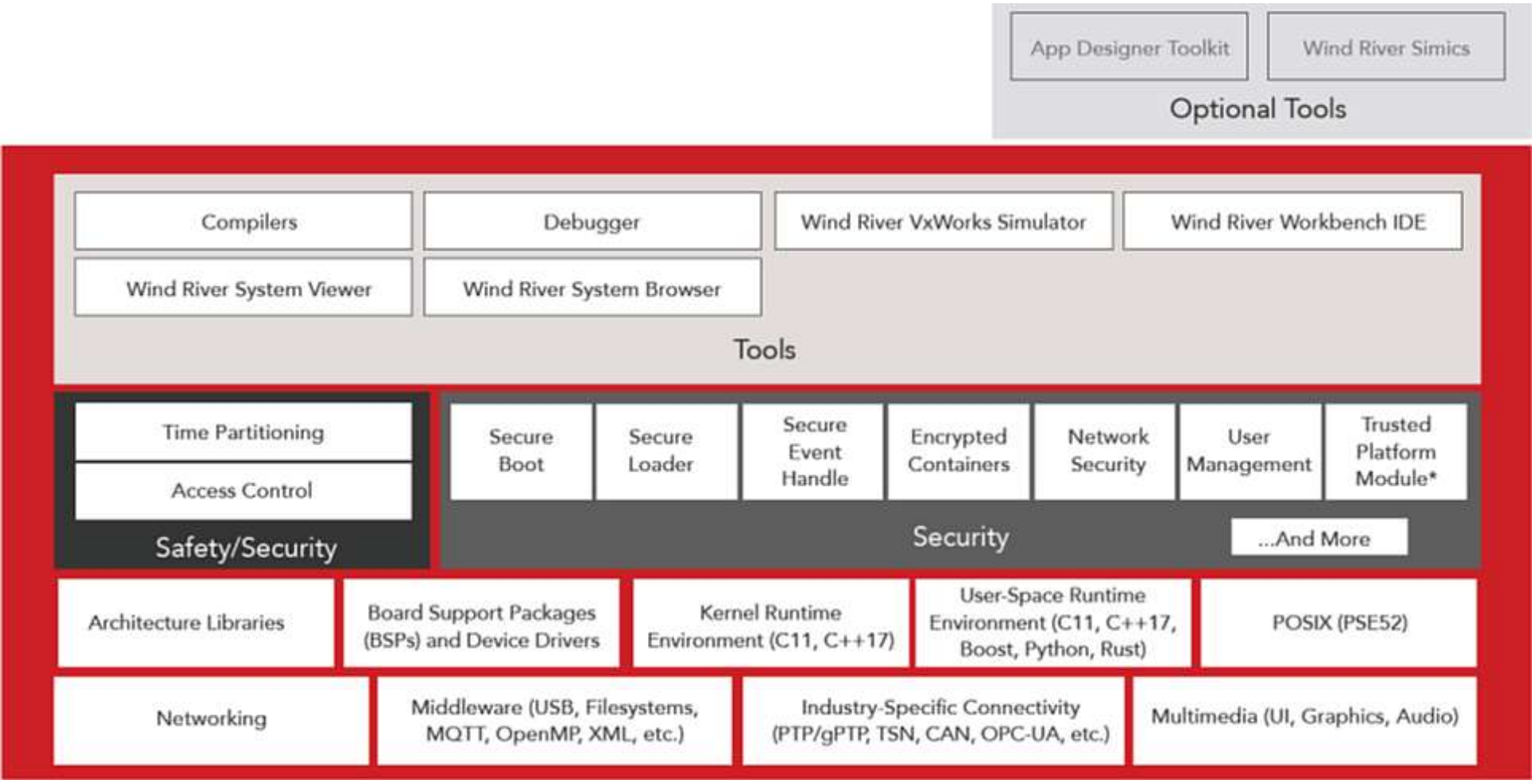
- Polityki czasu rzeczywistego
  1. **SCHED_FIFO** – wątki są szeregowane używając algorytmu FCFS (kolejka FIFO). Nie ma podziału czasu w przypadku wątków o tym samym priorytecie.
  2. **SCHED_RR** – polityka ta różni się tym od SCHED_FIFO , że dla wątków o tym samym priorytecie jest stosowane planowanie karuzelowe (*round-robin*).
  3. **SCHED_SPORADIC** – wątki są wysokopriorytetowe przez ograniczony czas, po przekroczeniu tego czasu – niskopriorytetowe; wysoki priorytet jest okresowo odzyskiwany.
- Polityka planowania nie-RT
  4. **SCHED_OTHER** – zwykle algorytm FIFO czy RR, ale priorytet niższy niż dla wątków RT (stąd wywłaszczanie). W ramach tej klasy szeregowania priorytet może być dynamiczny (np. dla realizacji postarzania).

Linux: dodatkowe klasy procesów zwykłych: SCHED_BATCH, SCHED_IDLE oraz procesów RT: **SCHED_DEADLINE** (szeregowanie Global Earliest Deadline First), zamiast SCHED_SPORADIC. Patrz man sched(7)

Profile POSIX RT

- POSIX 1003.1 Standard
  - Umożliwia pisanie przenośnych aplikacji czasu rzeczywistego (RT)
  - Implementacje są zbyt duże dla systemów wbudowanych
- POSIX 1003.13 – definiuje 4 podzbiory (profile)
  - *Minimal* (PSE51) – dla małych systemów wbudowanych; system komputerowy bez MMU, dysku terminala. Przykład użycia: sterownik tostera.
  - *Controller* (PSE52) – dla kontrolerów przemysłowych; mikrokontrolerowy system bez MMU, ale z dyskiem zawierającym prosty system plików. Przykład użycia : sterownik robota przemysłowego. PSE51,52 zakładają, że system obsługuje jeden proces, ew. z P-wątkami
  - *Dedicated* (PSE53) – dla dużych systemów wbudowanych; system komputerowy obsługujący procesy, z MMU, dyskowym systemem plików i obsługą komunikacji sieciowej. Przykłady użycia: układ sterowania awioniki, telefon komórkowy.
  - *Multi-purpose* (PSE54) – dla dużych systemów ogólnego przeznaczenia, z wymaganiami RT i obsługą użytkowników/grup; dyskowy system plików, komunikacja sieciowa, GUI, itd. Przykłady użycia: stacja robocza dla systemu kontroli ruchu powietrznego, czy telemetrii samochodów Formuły 1.

Przykładowy system RT: VxWorks



https://www.windriver.com/products/vxworks

Architektury: ARM (64b, v7), Intel (IA), Power PC, RISC-V,….
Wieloprocesorowść AMP i SMP; multi-OS (Type 1 hypervisor)
Open source Raspberry Pi Board Support Package

Wind Microkernel

- Własności mikrojądra Wind :
  - mały rozmiar (20kB)
  - wielozadaniowość: procesy jak i wątki
  - planowanie karuzelowe (*round-robin*) z wywłaszczaniem bądź bez
  - obsługa przerwań z ograniczonym czasem obsługi przerwania i czasu ekspedycj (interrupt and dispatch latency times)
  - dostępne są środki komunikacji międzyprocesowej: pamięć wspólna oraz przekazywanie komunikatów
- Wsparcie pamięci wirtualnej:
  - można zabronić umieszczania wybranych stron w pamięci podręcznej (cache)
  - dodatkowy komponent softwarowy VxVMI oraz sprzęt (MMU) mogą ograniczać dostęp do wybranych obszarów pamięci (tylko dla wybranych zadań)
- Certyfikacja zgodności z POSIX PSE52. Wiele „branżowych” certyfikacji =>

RTLinux

- Rygorystyczny system czasu rzeczywistego
- W systemie RTLinux standardowe jądro systemu Linux jest wykonywane jako zadanie.
- Jądro RTLinux obsługuje wszystkie przerwania – ew. przekierowując część obsługi do jądra systemu Linux.
- RTLinux nie pozwala standardowemu jądru systemu Linux na wyłączenie obsługi przerwań; pozwala to wyeliminować dodatkowe (niekontrolowane przez RTLinux) opóźnienia obsługi przerwań.
- RTLinux implementuje dodatkowe algorytmy szeregowania zadań, w tym RMS i EDF.