Systemy czasu rzeczywistego

Co to jest system czasu rzeczywistego RTS (*Real-Time System*)?

Często proponowana definicja RTS mówi, że są to systemy, w których można określić maksymalny czas wykonania poszczególnych operacji.

Szersza definicja: w systemach RTS poprawność procesu obliczeniowego zależy nie tylko od samego wyniku, ale również od czasu, w którym został on osiągnięty.

Ograniczenia czasowe RTS

Jakie konkretnie są te ograniczenia czasowe, których dotrzymanie powinien zagwarantować RTS?

- 1 sekunda?
- 1 milisekunda?
- 1 mikrosekunda?

Odpowiedź: każde z powyższych może być wymaganym czasem reakcji RTS.

Czy RTS po prostu musi być bardzo szybki?

Czy w dzisiejszych czasach wielordzeniowych procesorów z 4-gigahercowym zegarem, zwykły system operacyjny nie mógłby pełnić roli RTOS?
Zwłaszcza gdyby nie był nadmiernie obciążony?
Czy nie będzie dostatecznie szybki?

Odpowiedź: być może, ale niekoniecznie. Jeśli jest choć niewielkie prawdopodobieństwo, że jakieś operacje systemowe nie zmieszczą się w określonych rygorach czasowych, to w warunkach rzeczywistej pracy, prędzej czy później taka sytuacja wystąpi, i taki system po prostu nie spełni wymagań czasowych aplikacji, aczkolwiek tylko z rzadka.

Kiedy RTS jest naprawdę potrzebny?

Rozważmy dalej możliwość stosowania 4-GHz wielordzeniowych procesorów z zapasem mocy, do prostych zadań, w miejsce RTS. Czy sporadyczne niedotrzymanie rygorów czasowych operacji ma wielkie znaczenie praktyczne?

Odpowiedź: to zależy od aplikacji i od wymagań użytkownika. Łatwo wyobrazić sobie, że system sterowania silnikiem odrzutowym musi zareagować w odpowiednim czasie na sytuację wymagającą natychmiastowego odcięcia dopływu paliwa.

Dla odmiany: system sterowania reklamą świetlną może być akceptowalny nawet wtedy, gdy będzie okresowo na chwilę zatrzymywał przesuwanie napisu na wyświetlaczu. Albo gdy system telefonii VOIP w pewnych sytuacjach opóźni zakodowanie i wysłanie co któregoś pakietu dźwiękowego.

Ale uwaga: aplikacja dekodująca obraz w odtwarzaczu multimedialnym może być równie bezużyteczna jak zły system sterowania silnikiem, jeśli będzie systematycznie spóźniała się ze zdekodowaniem na czas co którejś klatki obrazu wideo.

Kiedy RTS jest naprawdę potrzebny (cd.)?

Odpowiedź 2: Jednak czasami można osiągnąć zaplanowany cel w systemie, który gwarantuje dotrzymanie danych reżimów czasowych na ogół, ale nie zawsze.

Silny system czasu rzeczywistego (*hard RTS*) musi bezwzględnie spełniać wszystkie wymagania RTS, i nadaje się do zastosowań, w których ich niedotrzymanie może powodować niebezpieczeństwo, katastrofę, itp.

Przykład: sterowanie lotem

Słaby system czasu rzeczywistego (*soft RTS*) dopuszcza pewne odchyłki od wyznaczonych reżimów czasowych, aczkolwiek ogólnie musi również działać szybko i niezawodnie. Systemy takie nadają się do zastosowań, w których nieznacznie spóźniona reakcja nadal może być przydatna.

Przykład: rozrusznik serca

Kiedy RTS jest naprawdę potrzebny (cd.)?

Odpowiedź 3: Wydajne procesory o dużej mocy obliczeniowej są kosztowne i mają duże zapotrzebowanie na moc zasilania. Jeśli zadanie nie wymaga dużej mocy obliczeniowej, to można je czasem zrealizować za pomocą systemu wbudowanego z 200-MHz mikrokontrolerem o minimalnych wymaganiach.

Wymaga to jednak zastosowania w RTS innego podejścia — pewnego minimalizmu, prostoty konstrukcji, i konsekwencji w projekcie całej architektury systemu. Ułatwia to zaprojektowanie systemu wykonującego dokładnie zaplanowane zadanie i nic więcej, a potem dokonanie jego analizy, w celu obliczenia najdłuższych ścieżek obliczeń, maksymalnych opóźnień, zlokalizowania wąskich gardeł, itp.

Przykłady aplikacji wymagających działania w czasie rzeczywistym

Oczywiste:

- sterowanie produkcją
- aparatura medyczna
- sterowanie elektroniką samochodową, np. ABS
- lotnictwo i aeronautyka

Mniej oczywiste:

- systemy nawigacji
- systemy rozpoznawania głosu
- odtwarzacze multimedialne, np. MP3
- interfejsy użytkownika
- urządzenia konsumenckie

Wymagania systemów czasu rzeczywistego

Dla spełnienia wymagań czasu rzeczywistego jego projektant musi określić:

- kiedy pewne operacje muszą być wykonane
- kiedy pewne operacje muszą być zakończone
- co zrobić w sytuacji, gdy wszystkie wymagania czasowe nie mogą być jednocześnie spełnione

Systemy czasu rzeczywistego nie są podobne do „zwykłych" systemów operacyjnych. Są inaczej projektowane i inaczej wykorzystywane. Zawierają inne podsystemy. Dostarczają programiście innych narzędzi. Przerzucają na programistę zadanie globalnego sterowania przydziałem zadań i priorytetów.

Systemy operacyjne przeznaczonych do budowy systemów RTS, zwane systemami operacyjnymi czasu rzeczywistego (RTOS = *Real Time Operating System*) posiadają specjalne mechanizmy niezbędne przy uruchamianiu i eksploatacji RTS. Różnią się one zasadniczo od zwykłych systemów operacyjnych (GPOS = *General Purpose Operating System*)?

Systemy operacyjne czasu rzeczywistego

Na przykład, w systemach RT nie ma równości ani sprawiedliwości, którymi kierują się algorytmy szeregowania GPOS. Wątek o wysokim priorytecie może wykonywać się tak długo jak zechce, uniemożliwiając wykonanie wątków o niższym priorytecie (oczywiście o ile sam nie zostanie wywłaszczony przez wątek o jeszcze wyższym priorytecie). Takie szeregowanie priorytetowe z wywłaszczaniem pozwala wątkom o wysokich priorytetach w RTOS zmieścić się w wyznaczonym reżimie czasowym.

System RTOS musi nie tylko dostarczać mechanizmów i usług dla wykonywania, planowania, i zarządzania zasobami aplikacji, ale powinien również sam sobą zarządzać w sposób oszczędny, przewidywalny, i niezawodny.

System operacyjny czasu rzeczywistego powinien być modularny i rozszerzalny.
W przypadku systemów wbudowanych, jądro systemu musi być małe, ze względu na lokalizację w ROM, i często ograniczoną wielkość pamięci RAM.

Podstawowymi zaletami jest prostota i oszczędność. Dlatego RTOS może mieć mikrojądro dostarczające tylko podstawowych usług planowania, synchronizacji, i obsługi przerwań. Gdy niektóre aplikacje wymagają większego spektrum usług systemowych (systemu plików, systemu wejścia/wyjścia, dostępu do sieci, itp.), RTOS zwykle dostarcza tych usług w postaci modułów dołączonych do jądra.

Czy/kiedy potrzebny jest nam system operacyjny?

Dotychczas mówiliśmy ogólnie o „systemach czasu rzeczywistego".

W projektowaniu systemów czasu rzeczywistego i systemów wbudowanych rozważa się dwa możliwe podejścia:

- z systemem operacyjnym - aplikacja tworzy wątki, które działają pod kontrolą systemu operacyjnego, który zarządza pracą całego systemu i dostarcza usług wątkom, podobnie jak w zwykłych systemach komputerowych

- bez systemu operacyjnego - aplikacja jest jednym programem, który musi sam gospodarować zasobami komputera.

Rozwiązanie drugie stosuje się w przypadku mniejszych aplikacji, w których programista jest w stanie w miarę łatwo zaimplementować wszystkie niezbędne mechanizmy obsługi wątków, alokacji pamięci i zasobów, itp.

Jeśli aplikacja przekracza pewien stopień złożoności, to opłaca się ją zbudować w oparciu o system operacyjny, godząc się na pewne nieuchronne narzuty pobierane przez ten system, w zamian za oferowane przezeń usługi.

Krótkie podsumowanie — pytania sprawdzające

1. Co to jest system czasu rzeczywistego?

2. Czym różni się silny system czasu rzeczywistego od słabego?

3. Jakie są podstawowe cechy systemów operacyjnych czasu rzeczywistego?

4. Kiedy system czasu rzeczywistego powinien być budowany na bazie systemu operacyjnego czasu rzeczywistego?

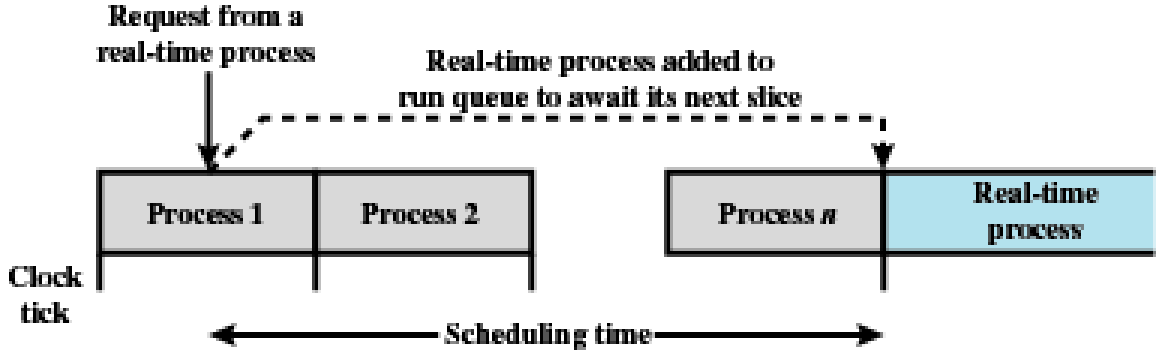
Mechanizmy systemów czasu rzeczywistego

Mechanizmy systemu operacyjnego, które umożliwiają pracę aplikacji w czasie rzeczywistym to:

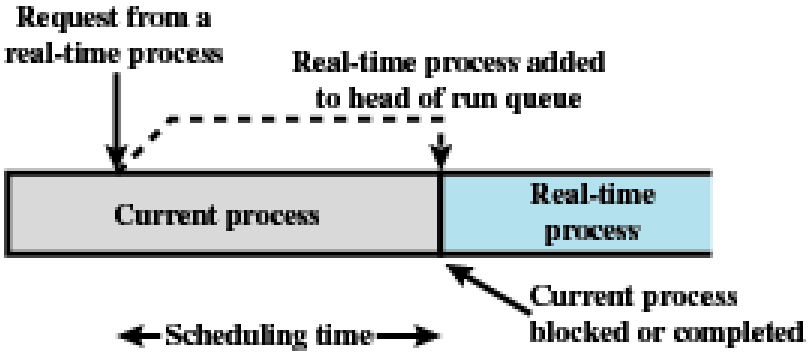
- przewidywalne szeregowanie wykonywania procesów
- mechanizmy zapobiegania inwersji priorytetów
- zarządzanie pamięcią
- wywłaszczalne jądro systemu
- ustalony maksymalny czas obsługi przerwań

Strategie szeregowania RTOS – priorytety i wywłaszczanie

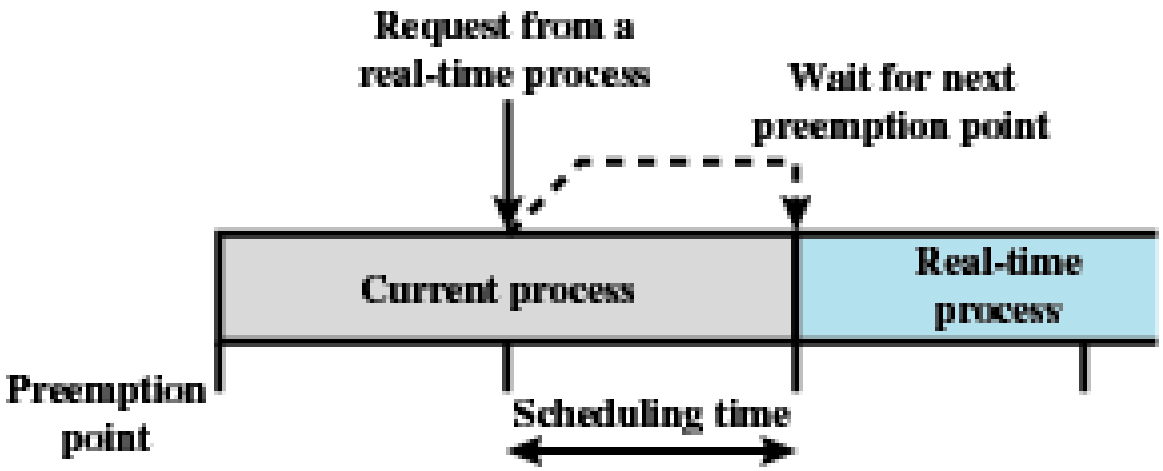
W najprostszym podejściu można osiągnąć wymagane szeregowanie procesów stosując priorytety połączone z wywłaszczaniem:



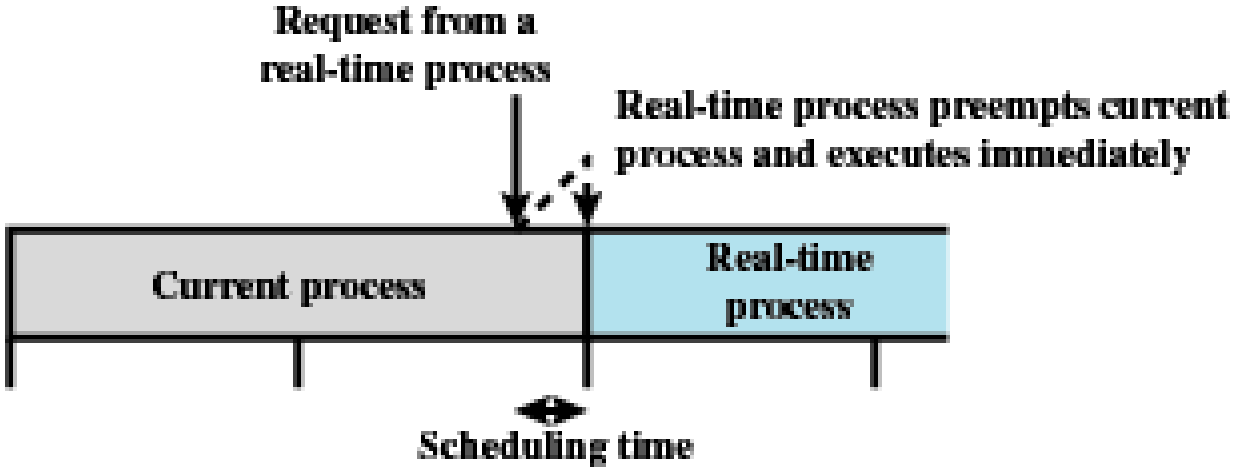
(a) Round-robin Preemptive Scheduler



(b) Priority-Driven Nonpreemptive Scheduler



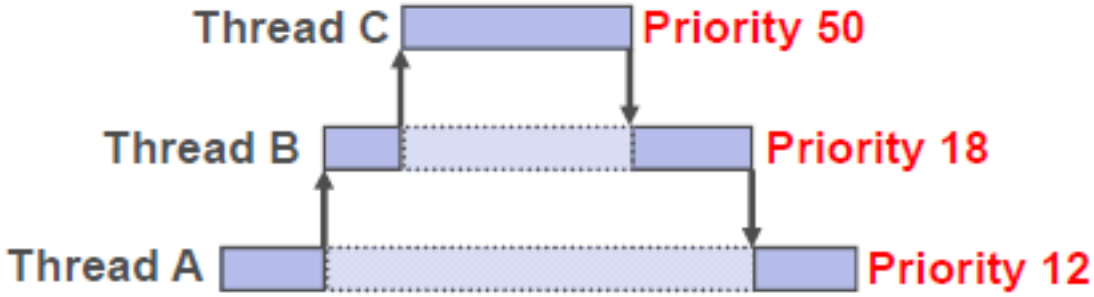
(c) Priority-Driven Preemptive Scheduler on Preemption Points



(d) Immediate Preemptive Scheduler

Strategie szeregowania RTOS – problemy z priorytetami

Przypisanie priorytetów procesom gwarantuje, że procesy o wyższych priorytetach będą wykonywane przed procesami o niższych priorytetach — i niestety tylko tyle.



W oczywisty sposób może to doprowadzić do sytuacji, gdy proces o niższym priorytecie nie zmieści się w swoich wymaganiach czasowych, które też mogą być krytyczne dla poprawności działania całego systemu.

Jednak w sytuacji gdy procesy charakteryzują się zróżnicowanymi czasami i częstotliwością wykonania, a rywalizują w dostępie do zasobów systemu, poprawne wyznaczenie priorytetów zwykle nie jest łatwe.

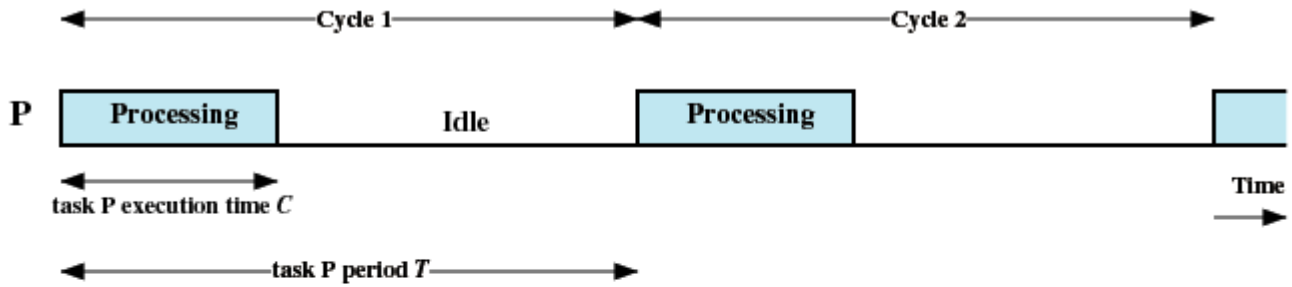
Strategie szeregowania RTOS – problemy z priorytetami (cd.)

Przykładem zadania, któremu trudno przypisać właściwy priorytet w systemie z wywłaszczaniem sterowanym priorytetami jest zadanie *watchdoga*. Zadanie takie wykonuje się okresowo i bardzo długo. Jego szeregowanie (cykliczne) można opóźniać w czasie, ale tylko do pewnego momentu. Nadanie mu zbyt niskiego priorytetu w oczywisty sposób może doprowadzić do jego zagłodzenia przez inne zadania. Jednak nadanie mu wysokiego priorytetu może spowodować, że watchdog wywłaszczy jakiś istotnie krytyczny wątek.

Przykładem innych zadań trudnych do szeregowania metodą priorytetów są zadania typowo obliczeniowe. Zadania te zajmują dużo czasu procesora, i jakkolwiek mogą być okresowo wywłaszczane przez zadania o krytycznych wymaganiach czasowych, to trudno zapewnić, by wiele takich zadań poprawnie się wykonywało. Właściwym sposobem szeregowania dla nich byłby algorytm typu RR (*Round-Robin* — rotacyjny) ignorujący priorytety. Innym sposobem, zgodnym z szeregowaniem priorytetowym, byłoby okresowe dobrowolne zwalnianie procesora przez takie zadania, jednak taką metodę trudno jest poprawnie zaimplementować.

Strategie szeregowania RTS – zadania okresowe

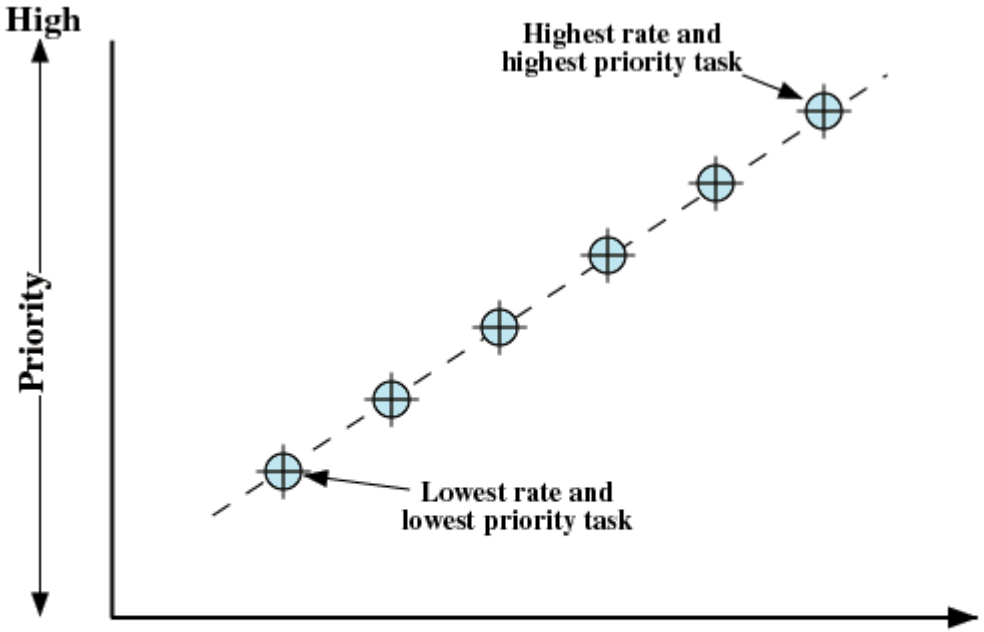
W systemach czasu rzeczywistego często mamy do czynienia z zadaniami okresowymi. Zadania takie muszą być wykonywane cyklicznie w nieskończonej pętli powtórzeń. Zadaniem systemu jest albo uruchamianie takich zadań z określonym okresem, albo — bardziej typowo — okresowe wznawianie takich zadań, które nigdy się nie kończą, tylko po wznowieniu wykonują swoje obliczenia i same zawieszają się zwalniając procesor.



Możemy traktować instancje danego zadania jako oddzielne zadania, które podlegają szeregowaniu przez system.

Strategie szeregowania RTOS – szeregowanie częstotliwościowe

Często stosowaną strategią dla zadań okresowych w RTS jest szeregowanie **częstotliwościowe monotoniczne** RMS (*Rate Monotonic Scheduling*). Metoda polega na przypisaniu zadaniom statycznych priorytetów proporcjonalnych do ich częstotliwości wykonywania. Zadanie o wyższej częstotliwości ma zawsze priorytet nad zadaniem o częstotliwości niższej. Wykonujące się zadanie jest wywłaszczane gdy uwolniona została kolejna instancja zadania o wyższej częstotliwości.



RMS jest algorytmem optymalnym pośród algorytmów z priorytetami statycznymi. Jeśli zbiór zadań da się szeregować algorytmem z priorytetami statycznymi, to RMS również będzie je poprawnie szeregować.

Szeregowanie częstotliwościowe (cd.)

Strategia RMS pozwala z góry obliczyć czy system będzie w stanie wykonywać wszystkie zadania zgodnie z ich wymaganiami, a także, jeśli byłoby to niemożliwe, to można obliczyć które zadania nie zmieszczą się w swoich reżimach czasowych.

RMS nie jest strategią globalnie optymalną. Istnieją zbiory zadań szeregowalne, ale nieszeregowalne algorytmem RMS. Jednak można udowodnić, że będzie poprawnie szeregować zbiór n zadań jeśli łączne obciążenie procesora spełni warunek:

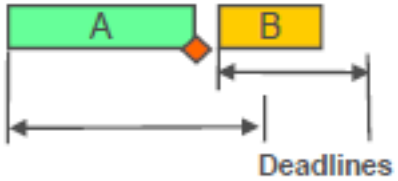
$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

gdzie C_i jest czasem obliczeń pojedynczej instancji zadania i, a T_i jego okresem.

Powyższe wyrażenie dla dwóch zadań ma wartość około 0.8284 a dla nieskończonej liczby zadań dąży do wartości $\ln 2 \approx 0.693147...$.
Podany warunek jest warunkiem wystarczającym, ale nie koniecznym. Dla wielu zbiorów zadań poprawne szeregowanie jest możliwe nawet do obciążenia zbliżającego się do wartości 1.0. Dla losowo wygenerowanego zestawu zadań okresowych szeregowanie jest możliwe do wartości około 0.85 (ale bez gwarancji). Zauważmy, że jeśli uruchomione w systemie zadania okresowe obciążają mniej niż 0.69 procesora, to można uruchomić dodatkowe zadania, niedziałające w czasie rzeczywistym, które mogą wykorzystać pozostałe wolne 30% mocy procesora.

Strategie szeregowania RTOS – szeregowanie terminowe

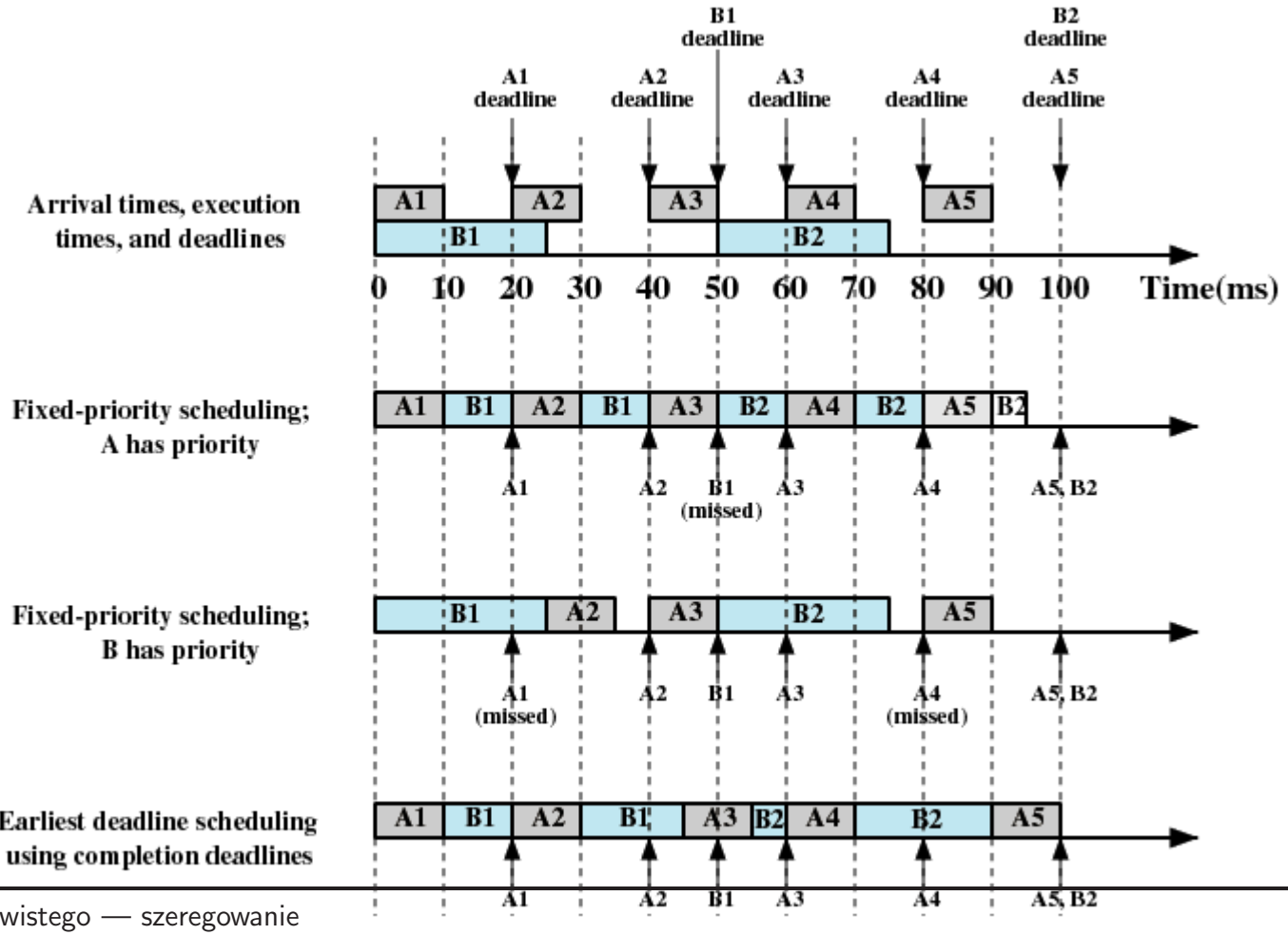
Istotą przetwarzania w RTOS jest aby określone zdarzenia występowały w określonym czasie. Jeśli zdefiniujemy pożądany czas rozpoczęcia i/lub zakończenia wszystkich zadań, to system może obliczyć właściwe szeregowanie zapewniające spełnienie wszystkich ograniczeń. Taka metoda jest nazywana **szeregowaniem terminowym** (*deadline scheduling*). Stosowana jest skrótowa nazwa tego algorytmu EDF (*earliest deadline first*).



Szeregowanie terminowe może uwzględniać terminy rozpoczęcia lub zakończenia zadań. Typowym, często spotykanym terminem wykonania (zakończenia) instancji zadania jest moment czasowy uwolnienia następnej jego instancji.

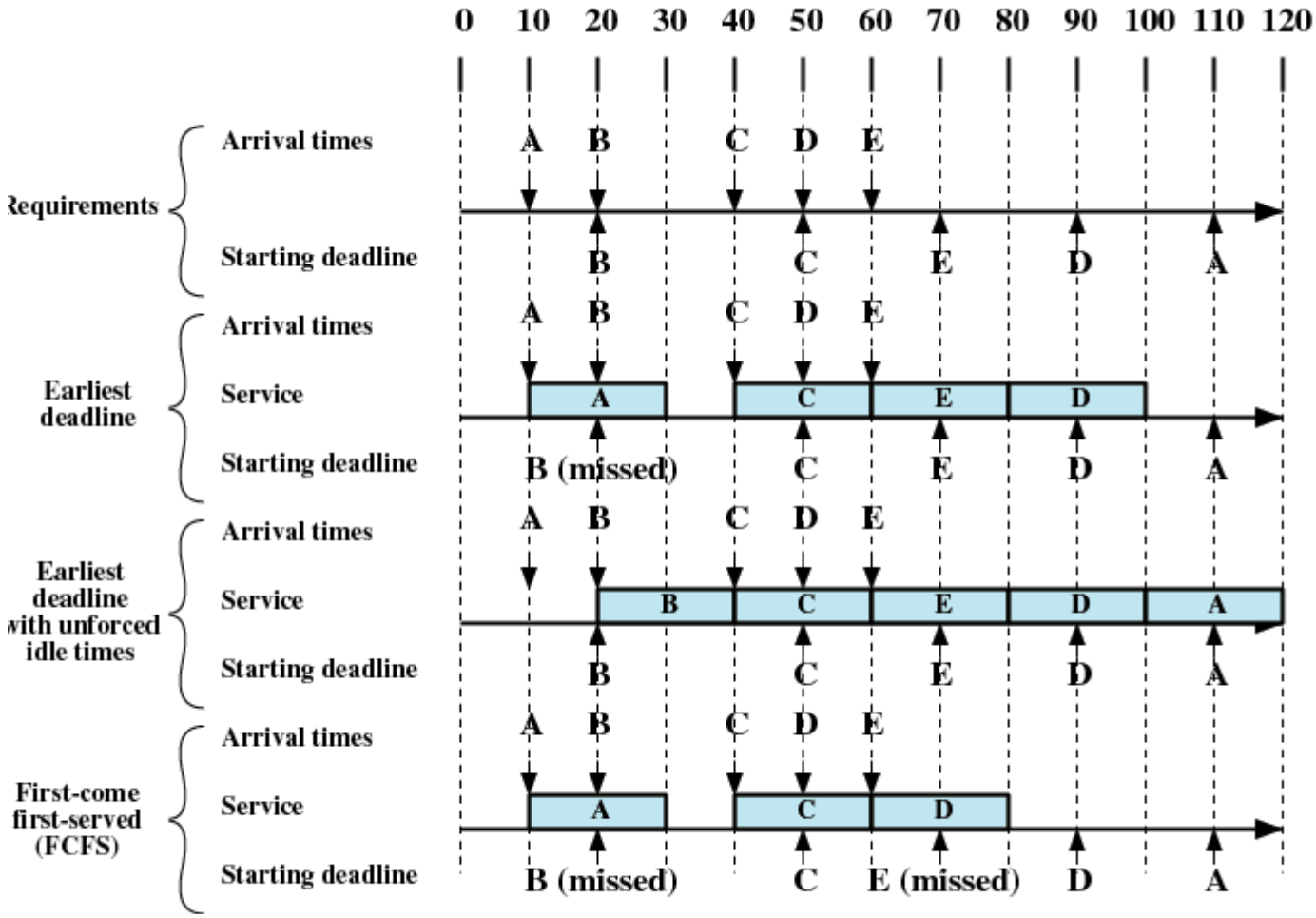
Szeregowanie terminowe dla zadań okresowych

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
•	•	•	•
•	•	•	•
•	•	•	•
B(1)	0	25	50
B(2)	50	25	100
•	•	•	•



Szeregowanie terminowe dla zadań nieokresowych

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70



Szeregowanie terminowe — własności

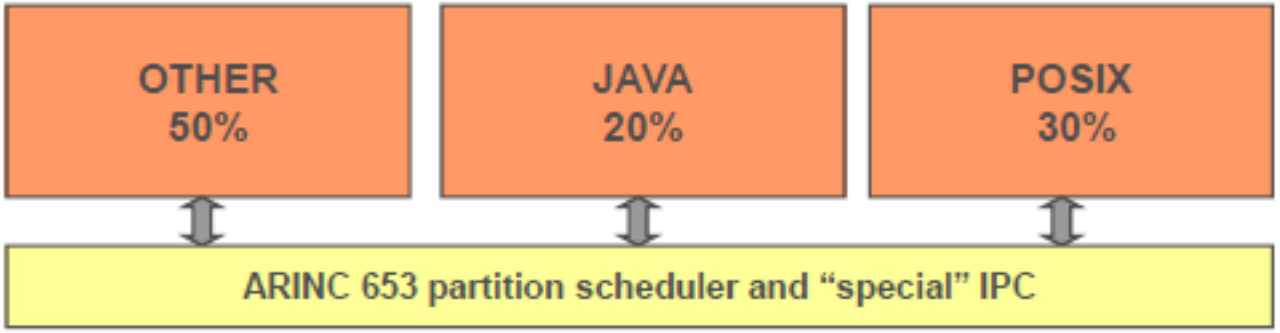
Algorytm szeregowania terminowego EDF dla pojedynczego procesora z wywłaszczaniem jest optymalny w takim sensie, że jeśli dany zbiór zadań, każde z określonym okresem, czasem uwolnienia, czasem obliczeń, i terminem zakończenia, jest szeregowalny jakimkolwiek algorytmem zapewniającym dotrzymanie czasów ukończenia, to EDF również będzie poprawnie szeregować ten zbiór zadań.

Jeśli terminy zakończenia zadań są równe końcowi ich okresów, to EDF będzie je poprawnie szeregował włacznie do współczynnika wykorzystania procesora $U = 100\%$. Zatem warunkiem szeregowalności zestawu zadań algorytmem EDF jest nieprzekroczenie 100% wykorzystania procesora, co stanowi przewagę tego algorytmu nad RMS. Jednak gdy system zaczyna być przeciążony to nie jest możliwe wyznaczenie zadania, które przekroczy swój termin (bo zależy to od konkretnych zadań terminów, oraz momentu, w którym wystąpi przeciążenie). Stanowi to istotną wadę tego algorytmu, w odróżnieniu od algorytmu RMS.

Zwróćmy uwagę, że szeregowanie terminowe (okresowe lub nie) ma sens (również) bez stosowania wywłaszczania w przypadku uwzględniania nieprzekraczalnych terminów rozpoczęcia, natomiast w przypadku nieprzekraczalnych terminów zakończenia należy stosować wywłaszczanie.

Strategie szeregowania RTOS – partycjonowanie

Innym podejściem do algorytmów szeregowania RTOS jest partycjonowanie, czyli podział na podsystemy, które otrzymują z góry określoną, gwarantowaną część czasu procesora.



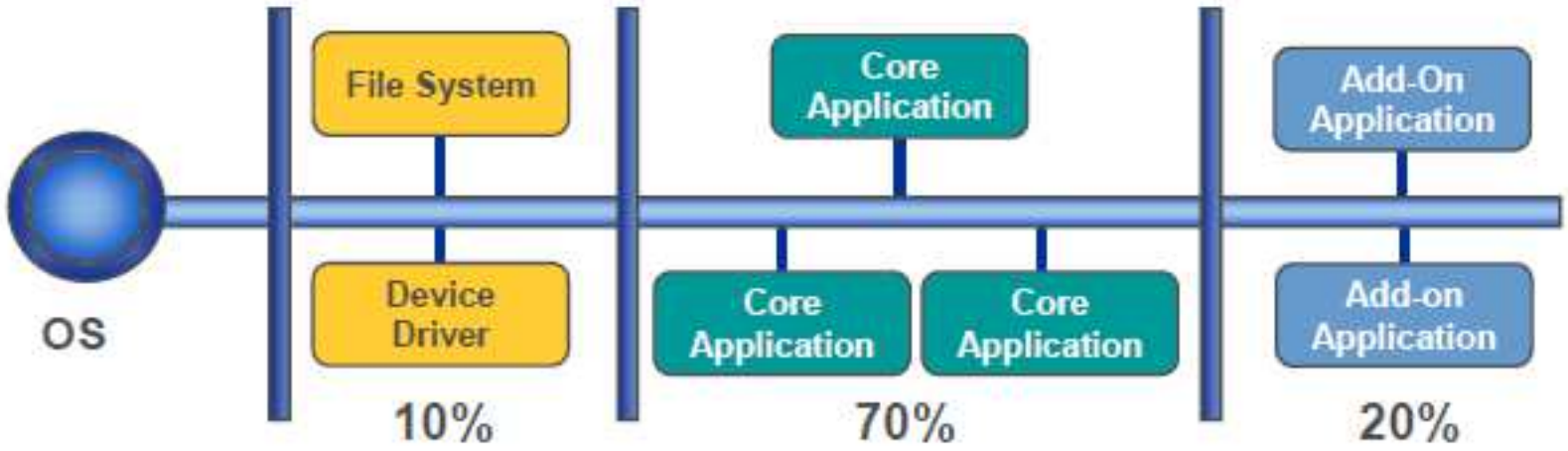
W tej architekturze system zapewnia przydział procesora, ochronę pamięci, i komunikację międzyprocesową pomiędzy partycjami, natomiast w ramach danej partycji funkcjonuje szeregowanie priorytetowe.

Strategie szeregowania RTOS – partycjonowanie (cd.)

Metoda partycjonowania działa poprawnie i bardzo niezawodnie, i jest szeroko stosowana w silnych systemach RT, szczególnie wojskowych, naprowadzania rakiet, awioniki, itp. Jej typowe zastosowania to są systemy statyczne, gdzie wykonano bardzo dokładne analizy (np. analizy RMS) weryfikujące poprawność działania, i nie wprowadza się już żadnych modyfikacji do kodu. W przypadku częstych zmian metoda staje się kosztowna.

Innymi oczywistymi wadami tej metody są: nieefektywne wykorzystanie procesora, oraz niemożność obsługi przerwań z opóźnieniem mniejszym niż okres przydziału procesora dla danej partycji.

Strategie szeregowania RTOS – partycjonowanie adaptacyjne



W partycjonowaniu adaptacyjnym system operacyjny zapewnia w każdej partycji przydział CPU zgodny ze specyfikacją, jednak w przypadku zwalniania procesora w jednej partycji system stopniowo przydziela ten czas innym partycjom zgłaszającym zapotrzebowanie na CPU.

Krótkie podsumowanie — pytania sprawdzające

1. Jakie strategie szeregowania stosowane są w systemach czasu rzeczywistego? Wymień te właściwe dla zadań okresowych i nieokresowych?

2. Które strategie szeregowania czasu rzeczywistego wymagają wywłaszczania?

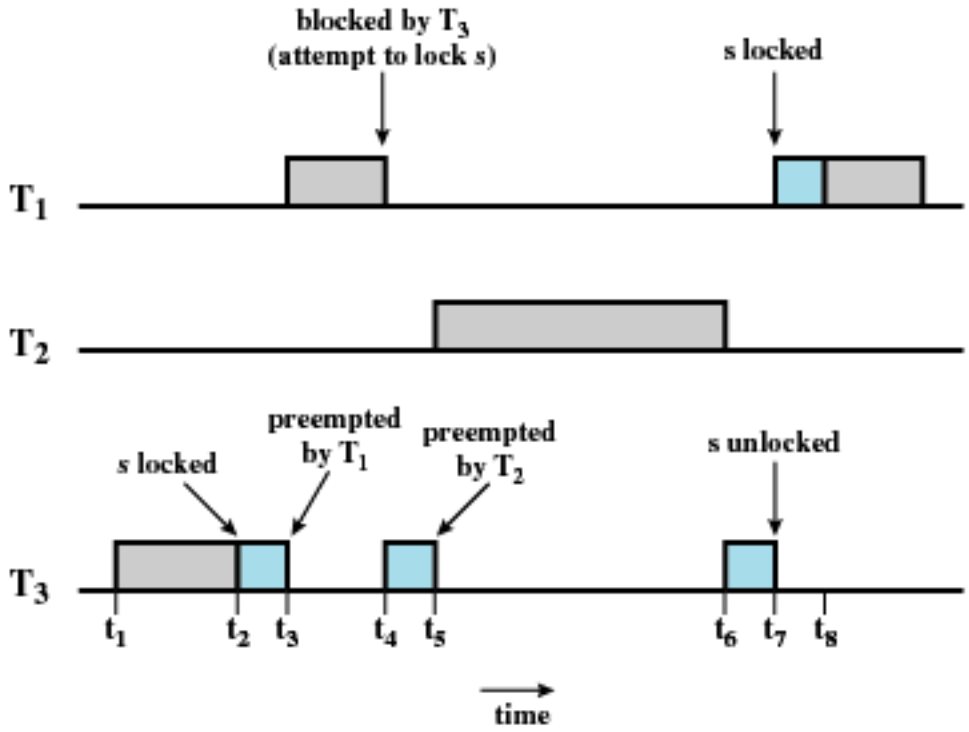
3. Kiedy strategia szeregowania może być stosowana bez wywłaszczania?

4. Przeanalizuj pracę algorytmu RMS dla dwóch zadań z parametrami: $C_1 = 25$, $T_1 = 50$, $C_2 = 30$, $T_2 = 75$. Jak ma się uzyskany wynik do podanego wyżej warunku teoretycznego szeregowalności zbioru zadań?

5. Zastosuj do przykładowych zadań z poprzedniego pytania algorytm EDF z czasami uwolnienia zadań równymi początkom ich okresów, i terminami zakończenia równymi końcom okresów.

Odwrócenie priorytetów

Jak widać na rysunku, wątek o niskim priorytecie T_3 poprawnie uzyskuje dostęp do jakiegoś zasobu s (semafora), po czym zostaje wywłaszczony przez wątek T_1 o wysokim priorytecie. Jednak gdy T_1 zgłasza żądanie dostępu do s, wykonywanie wraca do T_3. Co gorsza, wątek T_1 może być zablokowany przez czas dowolnie długi, gdyż T_3 może być wywłaszczany przez inne wątki, nawet o priorytetach niższych niż T_1.



(a) Unbounded priority inversion

Odwrócenie priorytetów — dziedziczenie priorytetów

Metoda **dziedziczenia priorytetów** polega na tym, że w momencie gdy wątek o wyższym priorytecie T_1 zostaje zablokowany w oczekiwaniu na zasób s zajęty przez wątek T_3 o niższym priorytecie, T_3 chwilowo dziedziczy wysoki priorytet T_1 i nie podlega wywłaszczeniu przez wątki typu T_2.

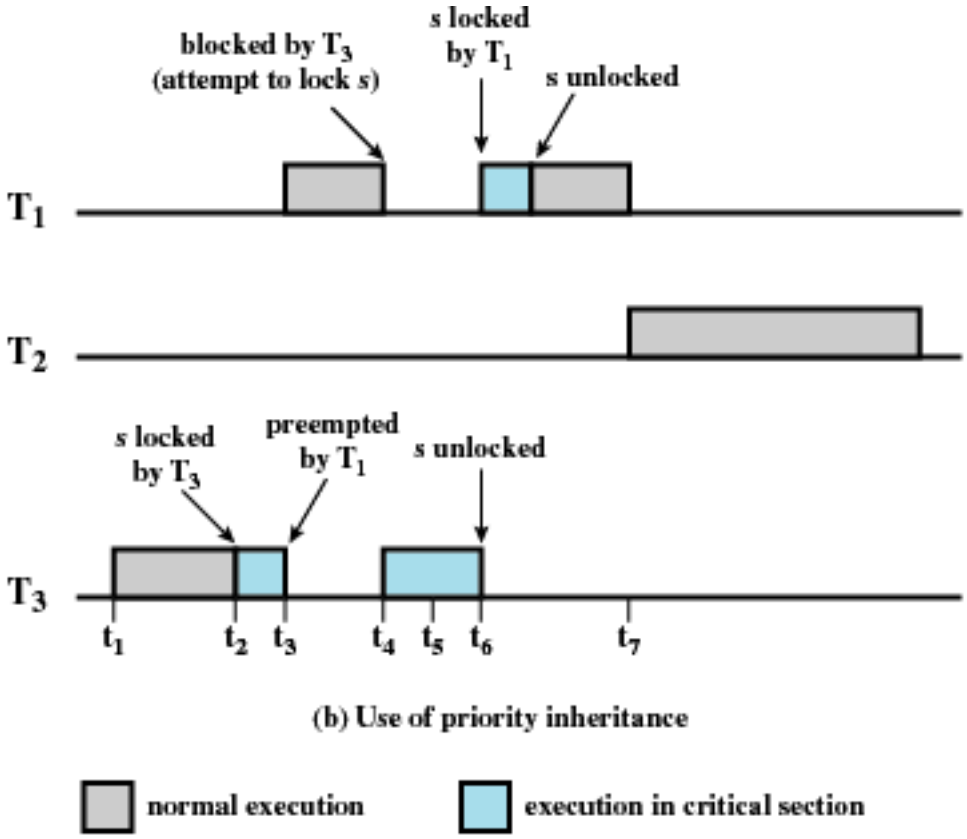


Figure 10.9 Priority Inversion

Odwrócenie priorytetów — pułap priorytetów

Inna metoda rozwiązania problemu inwersji priorytetów, zwana **pułapem priorytetów** (*priority ceiling*) polega na przypisaniu priorytetów zasobom, przy czym najniższy priorytet zasobu jest wyższy od najwyższego priorytetu wszystkich wątków. W chwili zajęcia zasobu wątek uzyskuje chwilowo priorytet równy temu zasobowi, i może wykonywać się do chwili zwolnienia zasobu, kiedy jego priorytet jest obniżany do pierwotnego poziomu.

Krótkie podsumowanie — pytania sprawdzające

1. Na czym polega zjawisko odwrócenia priorytetów?

2. Jakie są podstawowe mechanizmy rozwiązywania tego problemu?

Zarządzanie pamięcią

Typowe mechanizmy zarządzania pamięcią w systemach operacyjnych nie sprawdzają się w systemach RT:

- Pamięć wirtualna oparta na stronicowaniu i wymiataniu może spowodować, że gdy wątek o krytycznych wymaganiach czasowych zostanie uruchomiony, jakaś strona jego pamięci nie znajdzie się w RAM.

  Z tego względu w systemach RTOS albo w ogóle nie używa się pamięci wirtualnej, albo blokuje się stronicowanie dla wątków RT.

- Zwykłe mechanizmy dynamicznego przydziału pamięci (malloc/free) są typowo dość wolne, w porównaniu np. z samodzielnym zarządzaniem pamięcią przez wątek, który wstępnie przydzielił sobie pewną pulę pamięci.

  Zadania RT powinny zatem stosować metodą wstępnego przydziału RAM, i nie korzystać z systemowych mechanizmów pamięci dynamicznej.

Wywłaszczalne jądro w systemach RT

Typowe jądro systemu GPOS nie podlega wywłaszczaniu, tzn. funkcje jądra — zarówno operacje systemowe, np. obsługa przerwań, jak i zwykłe funkcje wywołane przez programy użytkowników — wykonywane są bez ograniczeń. W systemach RTOS może to powodować sytuacje, kiedy wykonywanie funkcji dla wątku o niskim priorytecie mogłoby trwać przez czas nieokreślony, lub ogólnie długi. W oczywisty sposób stoi to w sprzeczności z wymaganiami RTOS. Dlatego systemy RTOS mogą mieć jakąś formę wywłaszczania jądra.

Jednak procedury jądra każdego systemu mają okna czasowe, kiedy nie może dojść do jego wywłaszczenia. W tych okienkach nie działają mechanizmy czasu rzeczywistego, i muszą one być minimalizowane.

Niektóre systemy zapewniają wewnątrz procedur jądra **punkty wywłaszczania** (*checkpoints*), w których procedura może być przerwana i jądro wywłaszczone. Jednak analizując taki system z punktu widzenia wymagań RT należy pamiętać o uwzględnieniu wszystkich elementów jądra, w tym kodu sterowników.

Krótkie podsumowanie — pytania sprawdzające

1. Jakie zasady zarządzania pamięcią różnią systemy czasu rzeczywistego, od systemów operacyjnych ogólnego przeznaczenia?

2. Dlaczego w systemach czasu rzeczywistego przydatny jest mechanizm wywłaszczania jądra systemu, i jakie konsekwencje to powoduje?

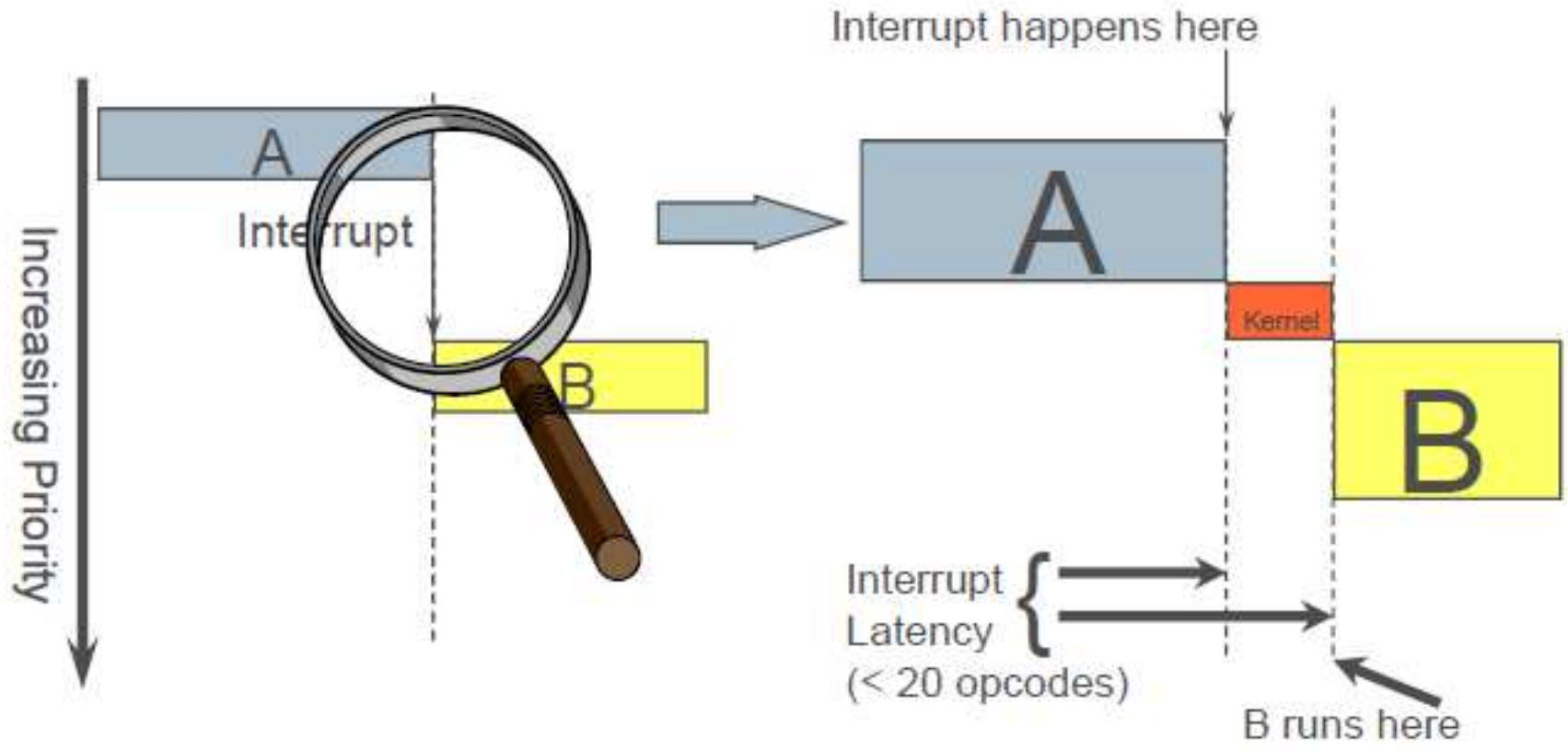
Obsługa przerwań

Elementy poprawnej obsługi przerwań:

- Efektywność
  - opóźnienie obsługi przerwania (*interrupt latency*)
  - opóźnienie szeregowania (*scheduling latency*)
- Wykonywanie łańcucha procedur obsługi przerwania
- Obsługa przerwań zagnieżdżonych

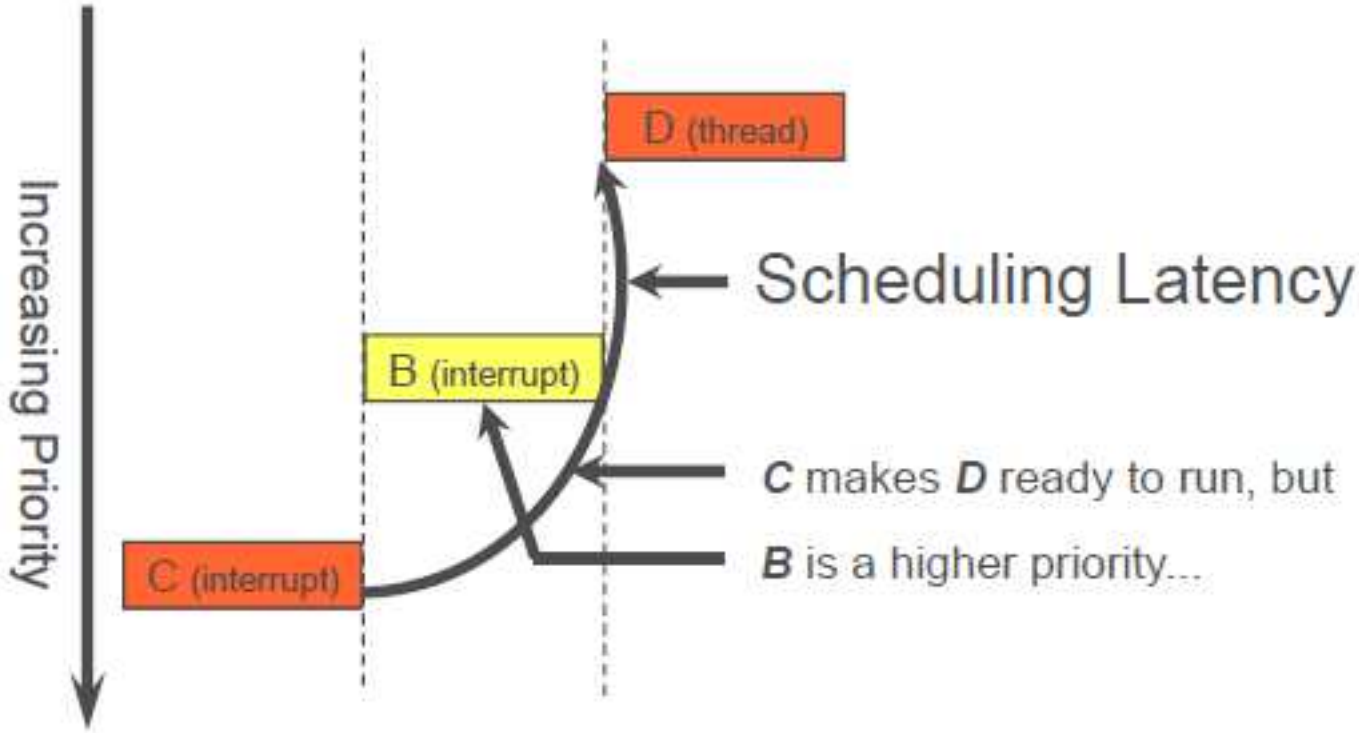
Obsługa przerwań – opóźnienie obsługi

Opóźnienie obsługi przerwania (*interrupt latency*) to jest czas od wystąpienia przerwania do wykonania pierwszej instrukcji procedury handlera:

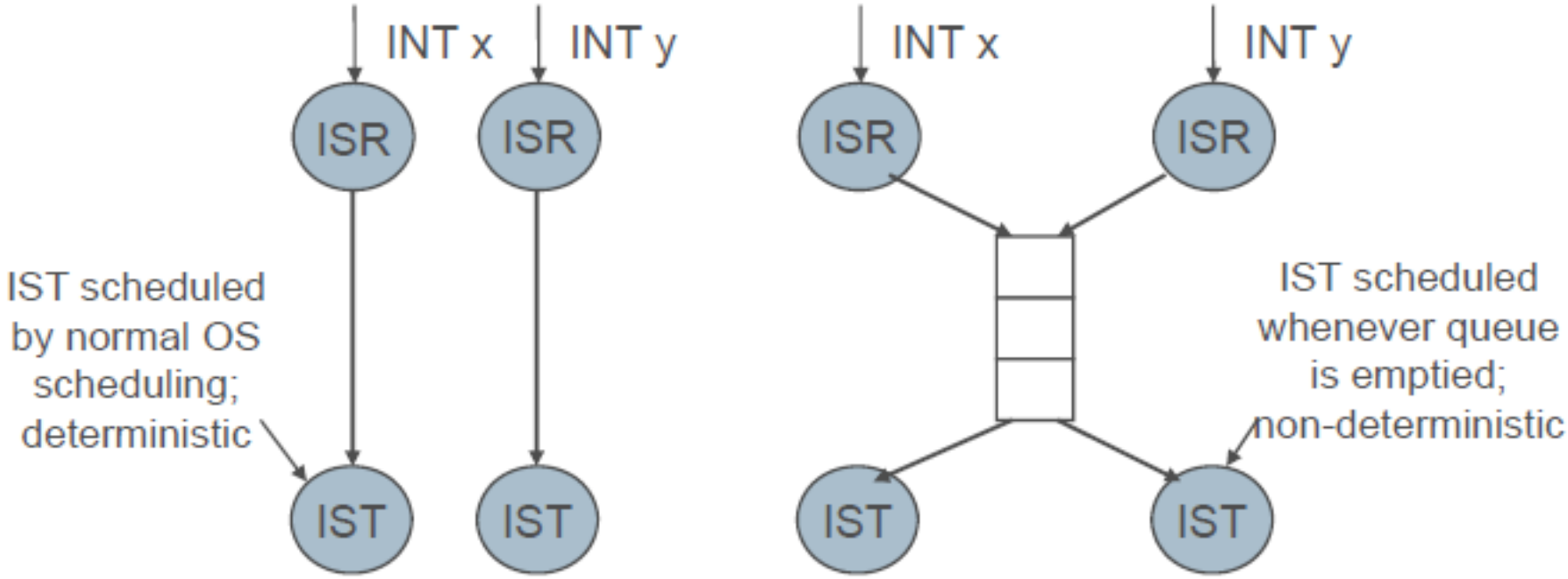


Obsługa przerwań – opóźnienie szeregowania

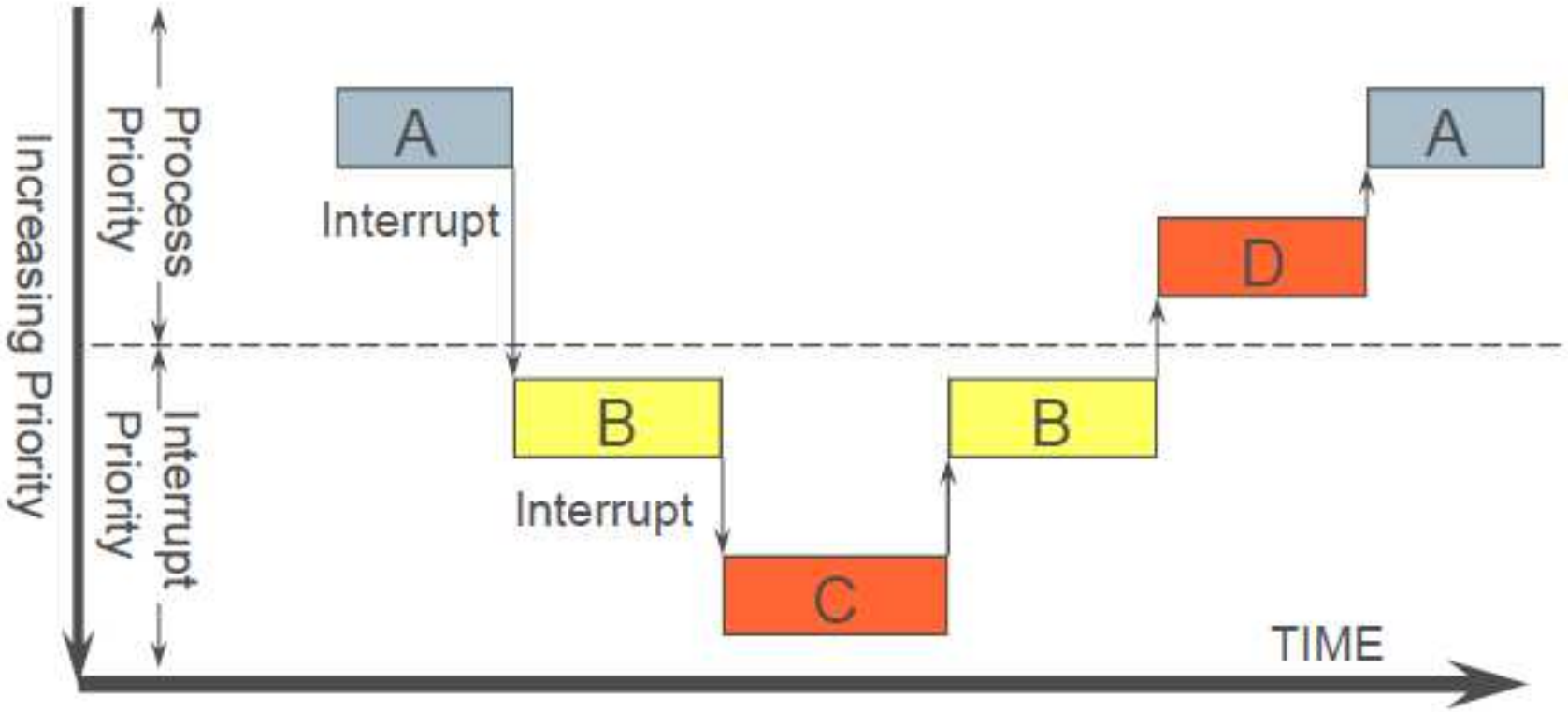
Opóźnienie szeregowania przerwania (*scheduling latency*) to jest czas od zakończenia ostatniej instrukcji procedury handlera do uruchomienia pierwszej instrukcji wznowionego wątku, który wywołał przerwanie:



Obsługa przerwań – wykonywanie łańcucha procedur obsługi



Obsługa przerwań – przerwania zagnieżdżone



Systemy czasu rzeczywistego — niezawodność

Systemy czasu rzeczywistego są często systemami zagnieżdżonymi, pracującymi w warunkach, w których nie można nacisnąć przycisku Reset w przypadku zawieszenia aplikacji lub całego systemu.

Ponadto systemy te pełnią często funkcje krytyczne, gdzie awaria może prowadzić do katastrofy.

Systemy czasu rzeczywistego — niezawodność (cd.)

- Niezawodne, wypróbowane metody projektowania i testowania.
- Minimalna wielkość kodu.
- Watchdog sprzętowy często niezbędny dla zapewnienia stałej pracy systemu. Wywołuje restart sprzętowy, po którym przez jakiś czas system wykonuje dobrze przetestowane ścieżki obliczeń. Jednak w oczywisty sposób, w czasie wykonywania restartu poprawność działania w czasie rzeczywistym załamuje się. Watchdog nie jest też panaceum na niedopracowany kod aplikacji. Jeśli system doszedł do sytuacji, dla której kod nie działa poprawnie, to po restarcie problem wystąpi ponownie.
- Monitory programowe do nadzorowania wykonania poszczególnych procesów; typowo niewielkim nakładem obliczeń, bez absorbowania mechanizmów systemowych

Systemy czasu rzeczywistego — architektura systemu