Szeregowanie w systemach czasu rzeczywistego

Witold Paluszyński
Katedra Cybernetyki i Robotyki
Wydział Elektroniki, Politechnika Wrocławska
http://www.kcir.pwr.edu.pl/~witold/

Szeregowanie procesów i wątków

Proces jest dynamicznym obiektem wykonującym w systemie operacyjnym określony program. Istotną cechą procesu jest izolacja jego środowiska i zasobów od innych procesów, np. proces posiada własny przydział pamięci. W systemach z pamięcią wirtualną przestrzeń adresowa procesu jest adresowana liniowo od zera.

Większość nowoczesnych systemów wspiera **wątki**, nazywane czasem **procesami lekkimi** (*lightweight processes*). Wątki funkcjonują w ramach procesu, wykonują kod procesu, i współdzielą między sobą jego dane globalne i globalnie alokowane zasoby. Każdy wątek posiada własną sekwencję wykonywanych instrukcji programu, zwaną **śladem wykonania** (*execution trace*).

W kontekście systemów czasu rzeczywistego najczęściej używa się pojęcia **zadania** (*task*). W wielu przypadkach zadanie jest identyczne z procesem, aczkolwiek w niektórych systemach jest utożsamiane z wątkiem. System operacyjny czasu rzeczywistego może wspierać tylko procesy, procesy z wątkami, albo tylko wątki.

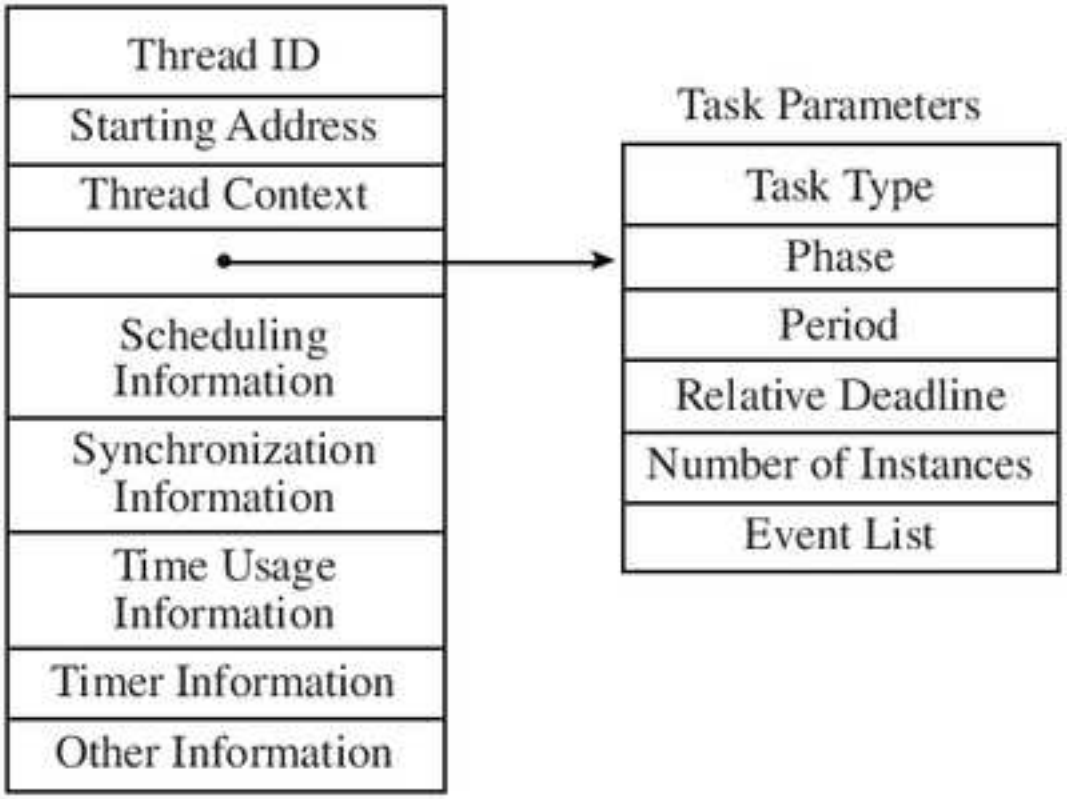
Podstawową rolą systemu w odniesieniu do zadań jest ich **planowanie** (*scheduling*), zwane również szeregowaniem, umożliwiające ich równoległe lub **quasi-równoległe** wykonywanie. Planowanie oznacza wyznaczenie zadania, które powinno następnie rozpocząć wykonywanie na procesorze.

Zadania

Utworzenie zadania polega na przydzieleniu mu pamięci i załadowaniu doń kodu programu, oraz zainicjalizowaniu struktury zwanej TCB (*Task Control Block*).

Informacje typowo zawarte w TCB:

- ID zadania
- adres startowy
- kontekst: zawartość rejestrów, PC, rejestru stanu, itp.
- informacje do planowania: typ zadania, faza, okres, termin, liczba instancji, itp.
- ...



Umieszczenie przez planistę zadania w kolejce oznacza umieszczenie TCB tego zadania na liście innych TCB zadań w tej kolejce.

Usunięcie zadania z systemu polega na wykasowaniu jego TCB i dealokacji zajmowanej pamięci.

Zadania okresowe

Zadania okresowe wykonują się w sposób powtarzalny. Ponieważ każdorazowe tworzenie, a następnie kasowanie zadania byłoby nieefektywne, zatem system operacyjny, który wspiera zadania okresowe, po zakończeniu zadania reinicjalizuje je i umieszcza w kolejce zadań oczekujących, a na początku następnego okresu wyzwala je (*release*), tzn. umieszcza w kolejce zadań gotowych.

Do parametrów zadań okresowych, dostarczanych w chwili tworzenia zadania, i przechowywanych w TCB zadania należą: faza, okres, względny termin, i liczba instancji. Jeśli zadanie okresowe ma skończoną liczbę instancji, to system może je wykasować po uruchomieniu zadaną liczbę razy.

Wiele systemów operacyjnych nie wspiera zadań okresowych. Mogą one być zaimplementowane w postaci programu, który naprzemiennie wykonuje jakiś fragment swojego kodu, i zasypia do początku następnego okresu. Przykładowe systemy wspierające zadania okresowe: Real-Time Mach, EPIQ.

Zadania aperiodyczne i sporadyczne

Innym specyficznym rodzajem zadań są zadania **aperiodyczne**. Są one wyzwalane w odpowiedzi na wystąpienie określonych zdarzeń. Zdarzenia te występują sporadycznie i mogą być wyzwolone przez zewnętrzne przerwanie. Po zakończeniu, zadanie aperiodyczne jest również reinicjalizowane i zawieszane.

Zadania aperiodyczne mogą być różnego rodzaju, mogą pojawiać się w dowolnym momencie, i mogą mieć wymagania czasowe zarówno twarde jak i miękkie. Wśród nich jako specjalny rodzaj wyróżnia się zadania **sporadyczne**. Zadania sporadyczne mają typowo twarde wymagania czasu rzeczywistego, ale jednocześnie mają określony maksymalną częstotliwość pojawiania się. Gdyby nie ta gwarancja, to nie dałoby się obliczyć bilansu czasowego całego systemu, i zapewnić spełnienie przezeń ograniczeń czasu rzeczywistego.

Planowanie zadań

Algorytmy planowania mają na celu zapewnienie spełnienia wymagań czasowych całego systemu. Muszą podejmować decyzję o przydzielaniu zasobów systemu biorąc pod uwagę najgorszy możliwy przypadek, lub czas odpowiedzi.

Główne grupy strategii planowania to: planowanie przed wykonaniem, i planowanie w czasie wykonywania.

Celem planowania przed wykonaniem, albo inaczej: planowania statycznego, jest wyznaczenie odpowiedniej kolejności wykonywania zapewniającej spełnienie ograniczeń i bezkolizyjny dostęp do zasobów systemu. Planowanie przed wykonaniem może również minimalizować pewne narzuty systemowe, takie jak przełączanie kontekstu, co zwiększa szanse na wyznaczenie poprawnego porządku wykonania.

Planowanie w czasie wykonywania, inaczej: planowanie dynamiczne, zadaniom przyznawane są priorytety, i zasoby są następnie przydzielane według tych priorytetów. W tym podejściu zadania mogą generować przerwania i żądać zasobów w dowolny sposób. Jednak aby potwierdzić poprawność pracy systemu, konieczne są testy i symulacje, w tym stochastyczne.

Algorytmy planowania ogólnego przeznaczenia

W systemach operacyjnych ogólnego przeznaczenia (GPOS - *General Purpose Operating System*) rozważa się szereg modeli planowania, takich jak:

- algorytm FCFS — *First-Come-First-Served*
- algorytm SJTF — *Shortest-Job-Time-First*
- algorytm SRTF — *Shortest-Remaining-Time-First*

Algorytmy te są zorientowane na maksymalizację wykorzystania procesora oraz skrócenie czasu oczekiwania dla zadań typowo obliczeniowych. W systemach czasu rzeczywistego nie mają one zastosowania.

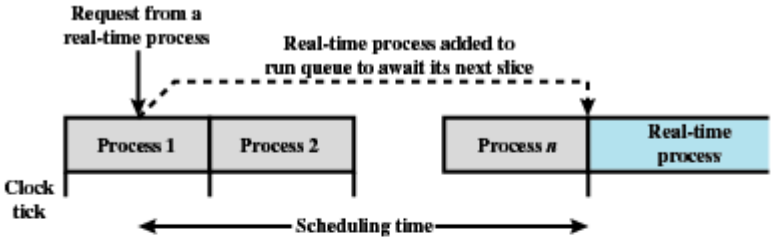
Do pewnego stopnia stosowane są natomiast algorytmy:

- algorytm *Round-Robin*, inaczej planowania rotacyjnego
- algorytmy planowania oparte na priorytetach

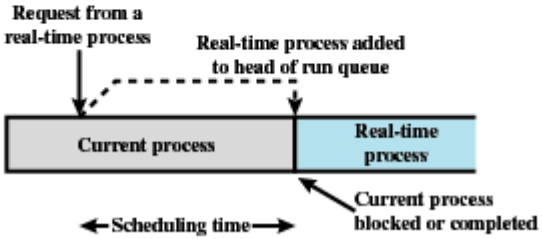
W przypadku wszystkich algorytmów planowania może być stosowane wywłaszczanie.

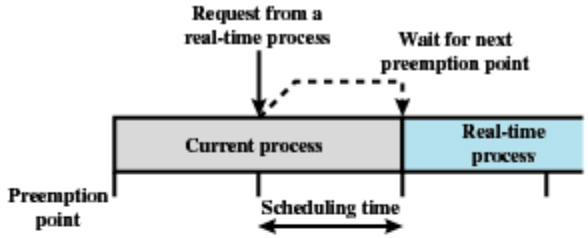
Priorytety i wywłaszczanie

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

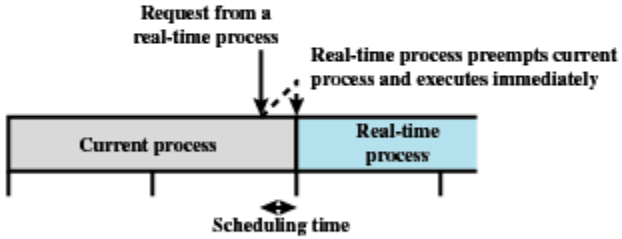


(a) Round-robin Preemptive Scheduler





(c) Priority-Driven Preemptive Scheduler on Preemption Points



(d) Immediate Preemptive Scheduler

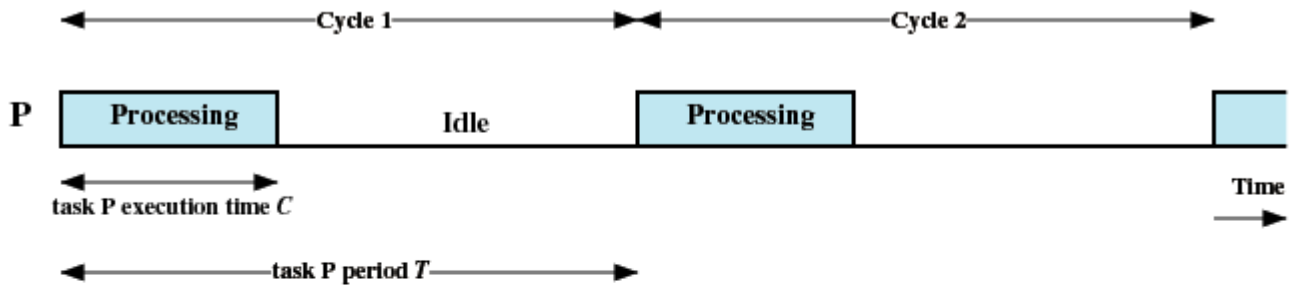
Problemy z priorytetami

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.

Zadania okresowe

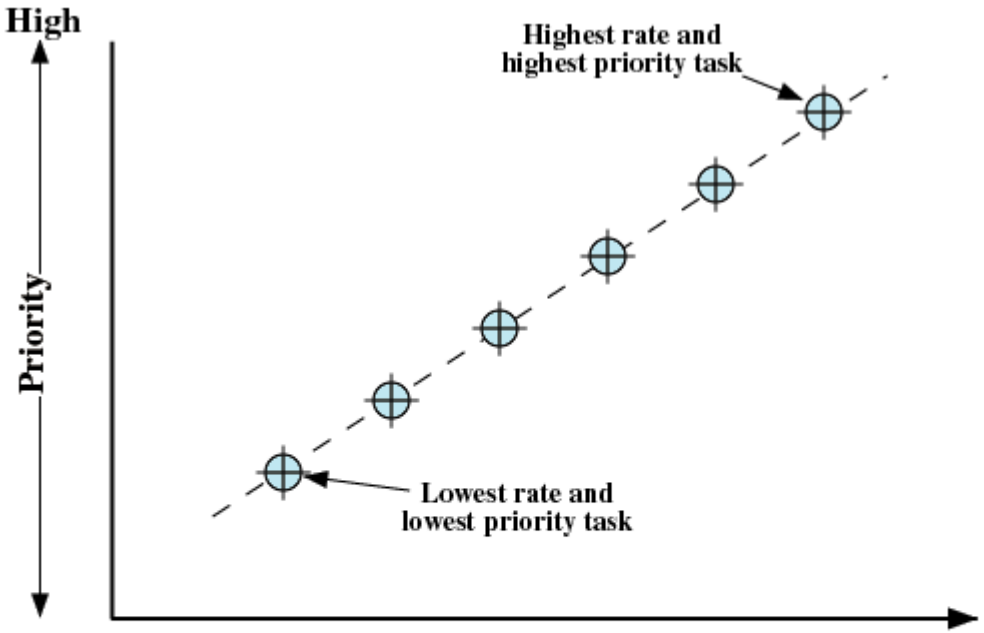
W systemach czasu rzeczywistego często mamy do czynienia z zadaniami okresowymi. Zadania takie muszą być wykonywane cyklicznie w nieskończonej pętli powtórzeń. Zadaniem systemu jest albo uruchamianie takich zadań z określonym okresem, albo — bardziej typowo — okresowe wznawianie takich zadań, które nigdy 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.

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.



Szeregowanie częstotliwościowe (RMS) — własności

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.

Twierdzenie (Liu): dla zestawu zadań okresowych, i planowania priorytetowego z wywłaszczaniem, przydział priorytetów nadający wyższe priorytety zadaniom z krótszym okresem wykonywania, daje optymalny algorytm planowania.

Szeregowanie częstotliwościowe (RMS) — własności (cd.)

Będziemy się posługiwać wartością maksymalnego wykorzystania procesora U:

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

gdzie dla n zadań, e_i jest czasem wykonania a p_i okresem i-tego zadania.

Twierdzenie (o kresie dla algorytmu RMS): dla dowolnego zestawu n istnieje poprawny harmonogram planowania jeśli:

$$U \leq n(2^{1/n} - 1)$$

Szeregowanie częstotliwościowe (RMS) — własności (cd.)

Przedstawione twierdzenie określa, że im więcej zadań, tym trudniej będzie znaleźć harmonogram planowania wykorzystujący pełną wydajności procesora. Można obliczyć limit teoretyczny wykorzystania procesora dla nieskończonego zestawu zadań. Wynosi on $\ln 2 \approx 0.69$. Co więcej, już dla kilku zadań zbliża się on do 70%. Dokładniej:

n zadań	1	2	3	4	5	6	...	∞
kres RMS	1.0	0.83	0.78	0.76	0.74	0.73	...	0.69

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.

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.

Szeregowanie częstotliwościowe (RMS) — własności (cd.)

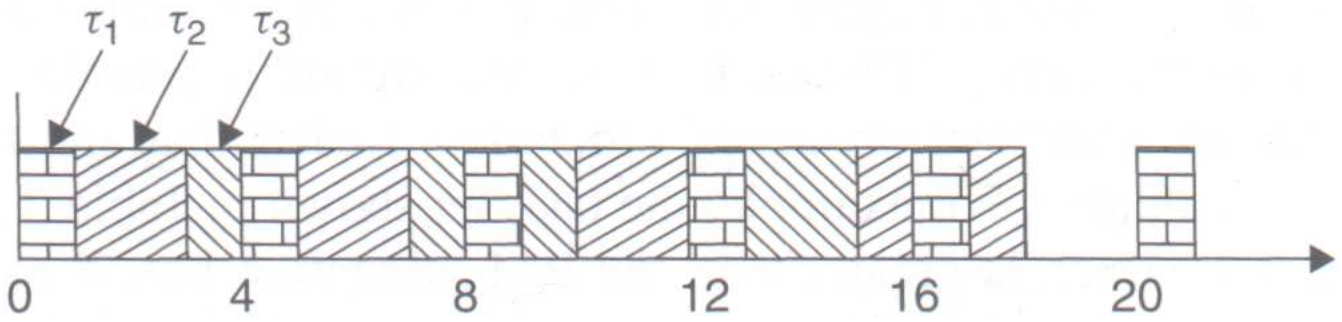
RMS nie jest strategią globalnie optymalną. Określony w twierdzeniu warunek jest tylko warunkiem wystarczającym, ale nie jest koniecznym dla realizowalności danego zestawu zadań.

W określonych przypadkach jest możliwe planowanie zestawu zadań, których wykorzystanie procesora przekracza podany limit teoretyczny. Złożone systemy czasu rzeczywistego osiągają często wykorzystanie procesora rzędu 80% bez większych problemów.

Dla losowo wygenerowanego zestawu zadań okresowych szeregowanie jest możliwe do wartości około 0.85 (ale bez gwarancji).

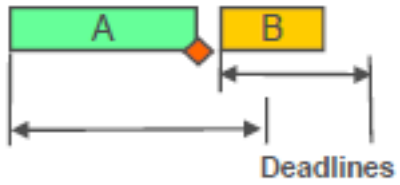
Planowanie RMS — przykład

τ_i	e_i	p_i	$u_i = e_i / p_i$
τ_1	1	4	0.25
τ_2	2	5	0.4
τ_3	5	20	0.25



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*).

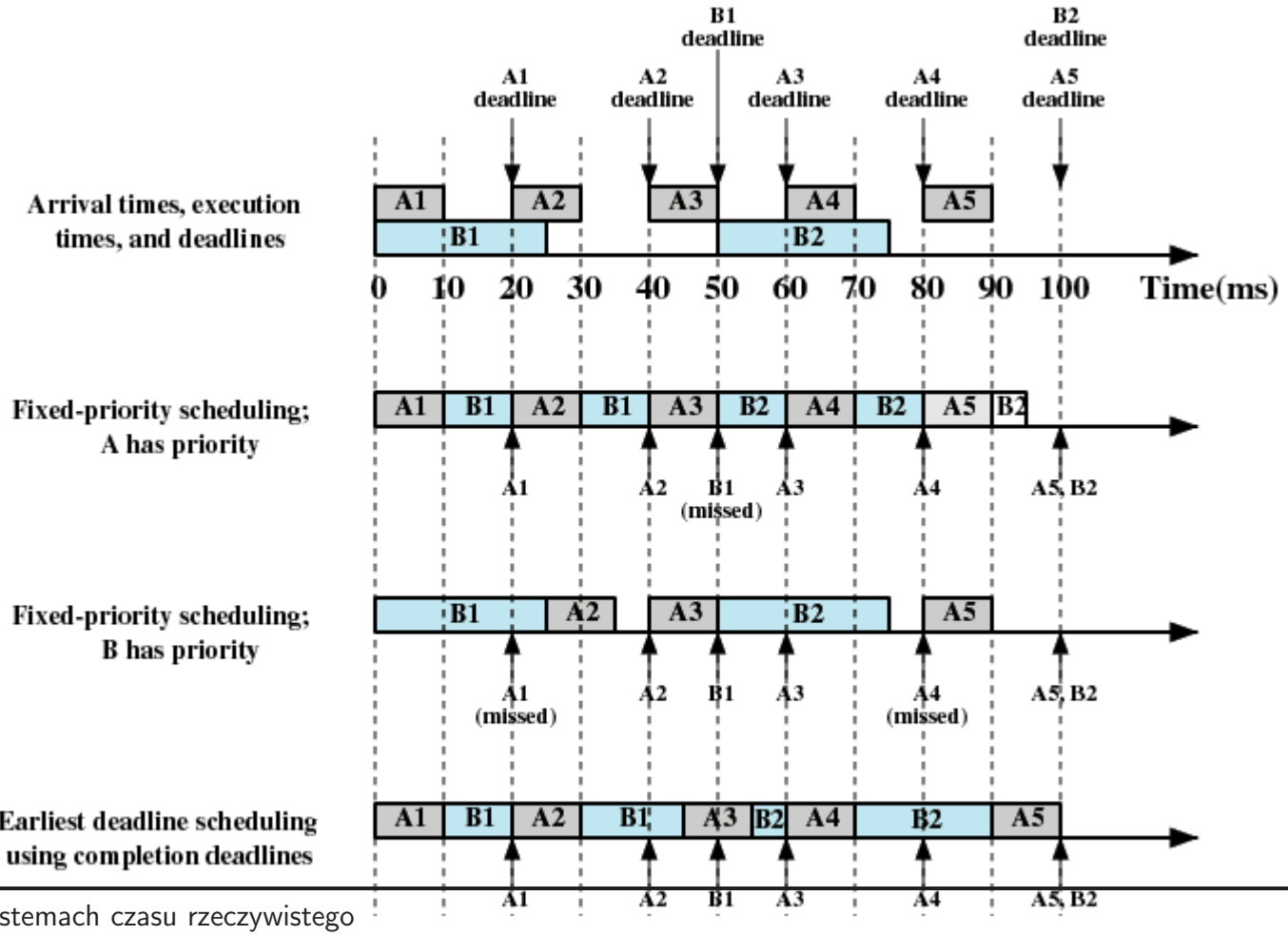


Szeregowanie terminowe może uwzględniać terminy rozpoczęcia lub zakończenia zadań. Typowym, często spotykanym terminem wykonania (zakończenia) instancji zadania jest moment czasowy uwolnienia następnej jego instancji.

Algorytm jest bardzo intuicyjny, ponieważ ludzie często podejmują decyzje w podobny sposób. Człowiek, który jest obciążony dużą liczbą zadań, do tego stopnia, że nie wie za co się zabrać najpierw, często zabiera się za zadanie, którego nieuchronny termin wykonania nadchodzi najpierw.

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 — 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.

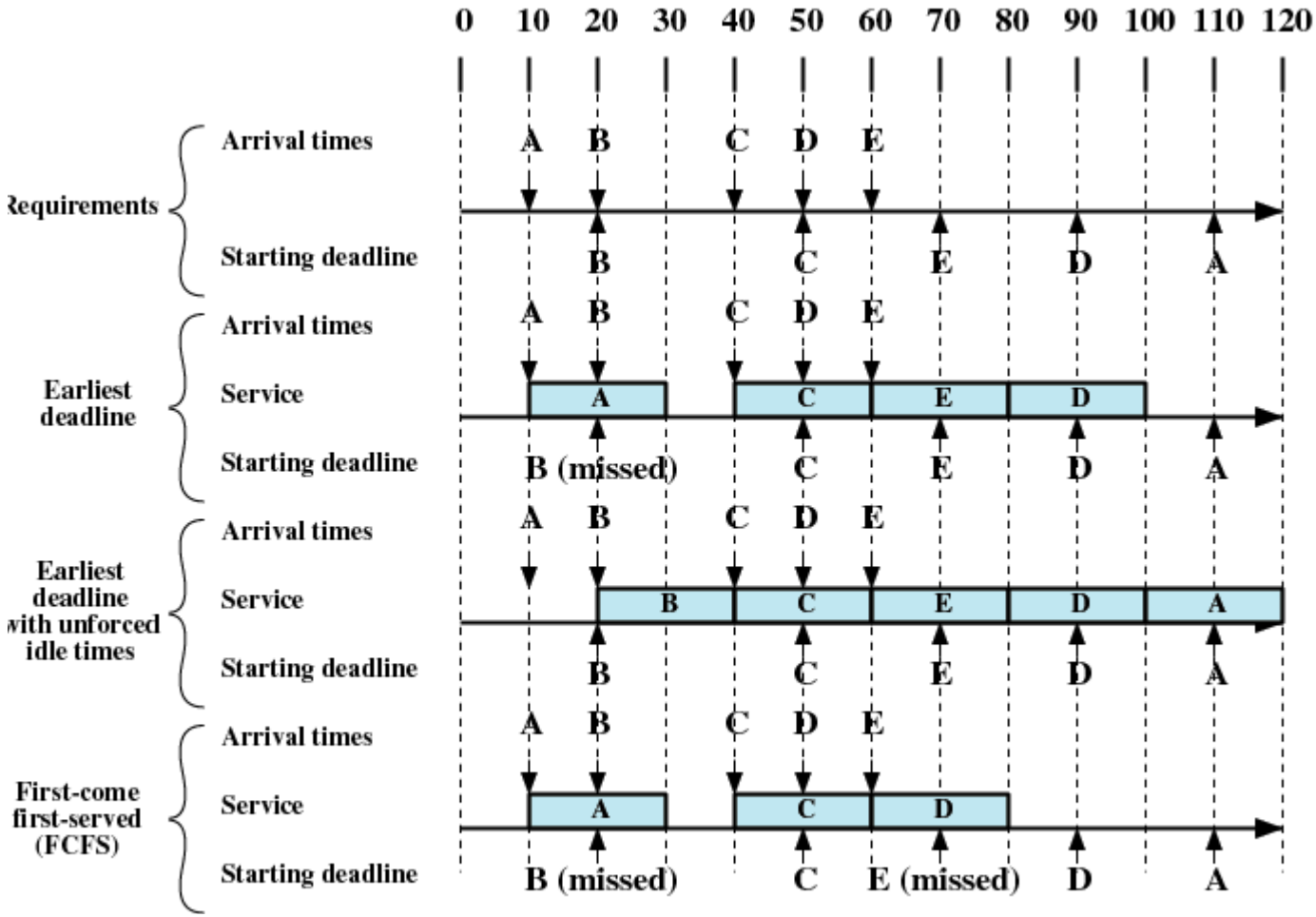
Zwróćmy uwagę, że szeregowanie terminowe (okresowe lub nie) ma sens (również) bez stosowania wywłaszczania w przypadku uwzględniania nieprzekraczalnych terminów rozpoczęcia, natomiast w przypadku nieprzekraczalnych terminów zakończenia należy stosować wywłaszczanie.

Planowanie dynamiczne — procesy nieokresowe

W przypadku gdy zestaw zadań nie ma stałego, okresowego charakteru, metody planowania przed wykonaniem nie mają zastosowania. Planowanie w czasie wykonania, inaczej planowanie dynamiczne, musi brać pod uwagę wszystkie ograniczenia aktualnie istniejących zadań. Możemy wtedy zastosować algorytm EDF.

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

Zwróćmy uwagę, że szeregowanie terminowe (okresowe lub nie) ma sens bez stosowania wywłaszczania w przypadku uwzględniania nieprzekraczalnych terminów rozpoczęcia, natomiast w przypadku nieprzekraczalnych terminów zakończenia zwykle właściwa jest strategia szeregowania z wywłaszczaniem.



Algorytm EDF — własności

Można sformułować następujące twierdzenie dla algorytmu EDF zastosowanego do zestawu zadań okresowych:

Twierdzenie (o kresie dla algorytmu EDF): zestaw n zadań okresowych, których termin wykonania jest równy ich okresowi, może być poprawnie planowany algorytmem EDF jeśli:

$$\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$$

Algorytm EDF jest optymalny dla pojedynczego procesora z wywłaszczaniem. Inaczej, jeśli istnieje poprawny harmonogram, to EDF będzie działał poprawnie.

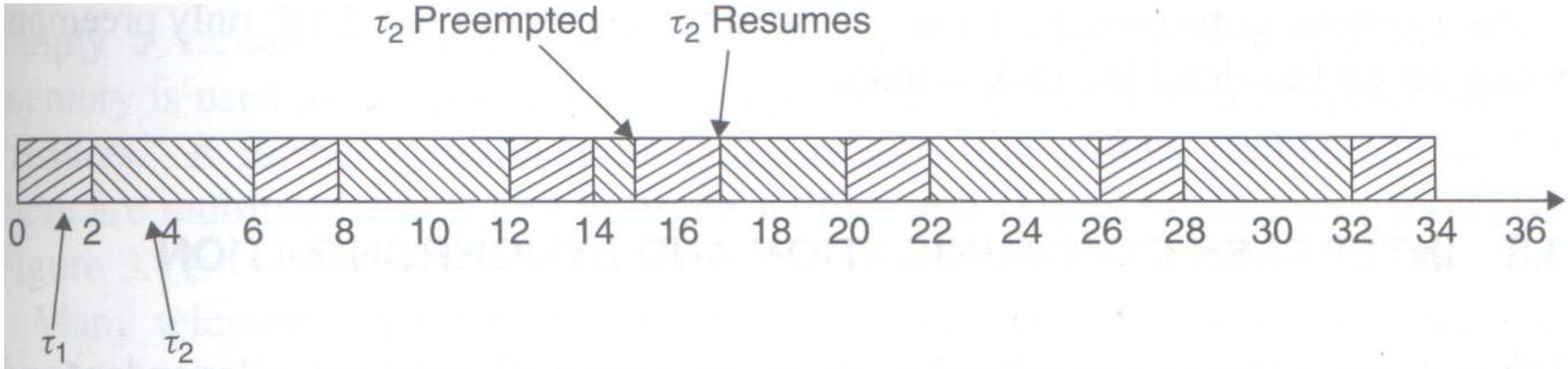


Figure 3.10 EDF task schedule for task set in Table 3.4.

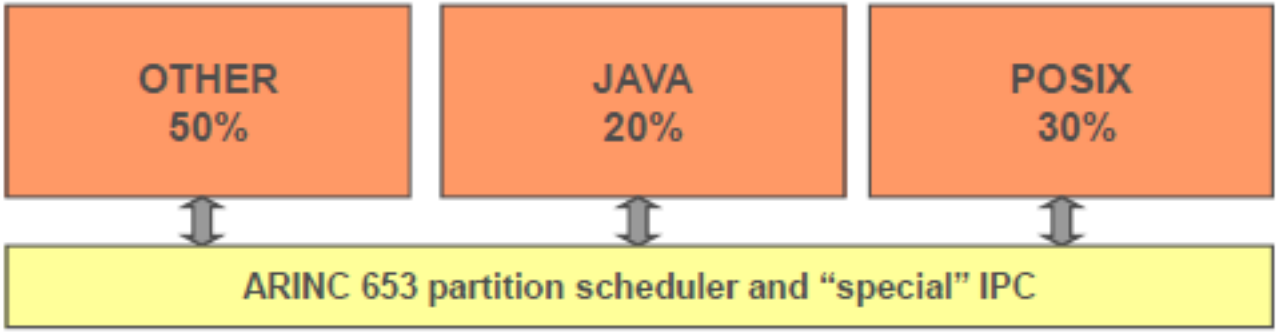
Table 3.4 Task set for example of EDF scheduling

τ_i	e_i	p_i
τ_1	2	5
τ_2	4	7

Zadania są uwalniane równocześnie, lecz τ_1 jest uruchamiane jako pierwsze ze względu na bliższy termin.

Szeregowanie przez partycjonowanie

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



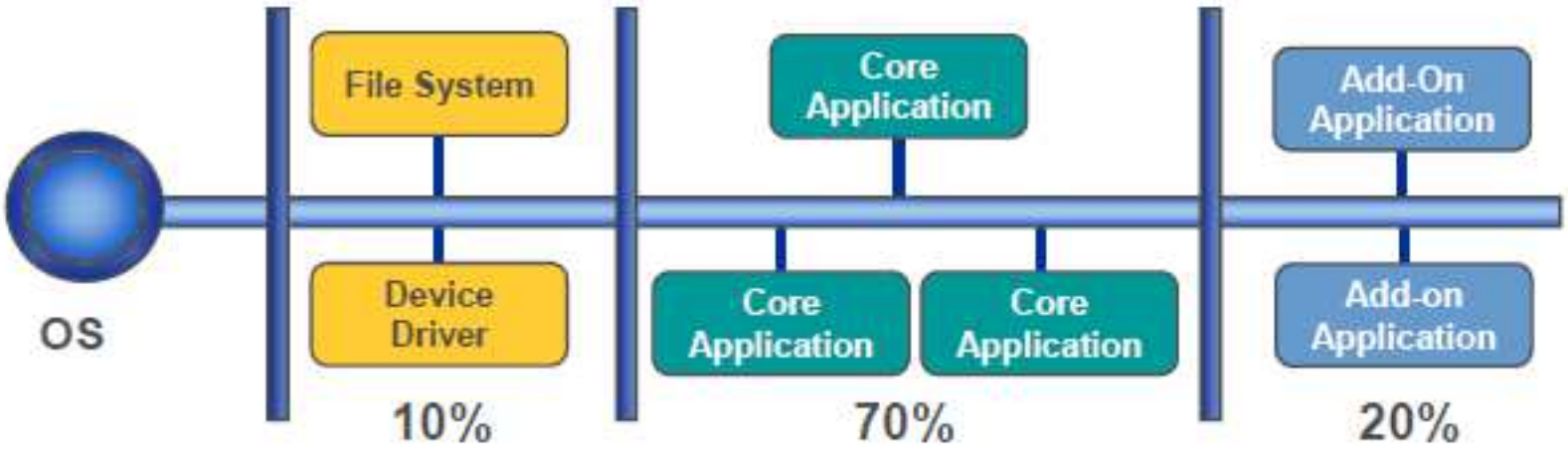
W tej architekturze system zapewnia przydział procesora, ochronę pamięci, i komunikację międzyprocesową pomiędzy partycjami, natomiast w ramach danej partycji funkcjonuje szeregowanie priorytetowe.

Szeregowanie przez 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.

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.