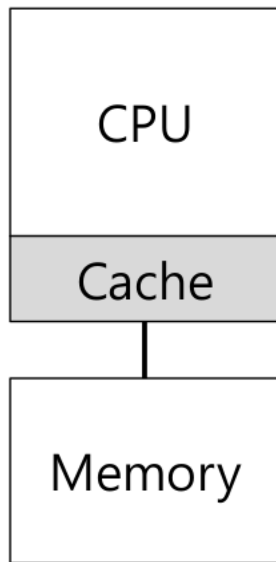
Szeregowanie wieloprocesorowe — wstęp

Na pozór można zrealizować szeregowanie w systemach wieloprocesorowych tymi samymi metodami co w systemach jednoprocesorowych. To znaczy, w przypadku szeregowania zdarzeniowego można zdarzenia wyzwalające pracę planisty i decyzję o rozpoczęciu wykonywania kolejno wybranego procesu odnieść do wszystkich procesorów. Za każdym razem kiedy zwolni się jeden z procesorów, planista wyznaczałby kolejne zadanie do uruchomienia na tym procesorze, a w przypadku planowania z wywłaszczaniem, za każdym razem gdyby pojawiło się nowe zadanie, planista sprawdzałby czy któregoś z zadań obliczanych na wszystkich procesorach nie należałoby wywłaszczyć.

Podobnie w przypadku planowania okresowego, jak Round Robin, można tę metodę rozszerzyć na przypadek wielu procesorów i — po upłynięciu kolejnego kwantu czasu dla każdego z procesorów — uruchamiać na nim kolejne zadania z kolejki gotowych.

Jednak to podejście jest z pewnych względów zwykle nieoptymalne. Gdyby to samo zadanie miało być ponownie wykonywane w kolejnym kwancie czasu, to bardzo korzystna byłaby kontynuacja jego wykonywania na tym samym procesorze, po pierwsze ze względu na brak konieczności przełączenia kontekstu, czyli zachowanie zawartości rejestrów procesora. Po drugie, i co znacznie ważniejsze, korzystne byłoby zachowanie zawartości pamięci buforowej *cache* procesora, zawierającej zapewne wszystkie dane, które proces w tym okresie potrzebuje do kontynuacji obliczeń.

Wykorzystanie pamięci buforowej *cache* procesora

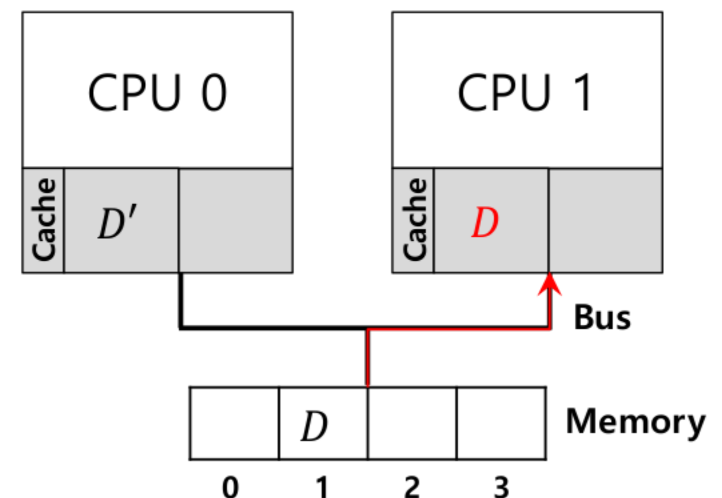


Pamięć buforowa *cache* przechowuje kopie danych pobranych z pamięci RAM, i być może zmodyfikowanych tutaj, lub wysłanych przez procesor do RAM.

Cache ma ograniczoną wielkość, ale jej skuteczność wynika z zasady lokalności i zwykle zawiera wszystkie dane, z których korzysta procesor, zapewniając super szybkie wykonywanie instrukcji, z minimalnymi odwołaniami do pamięci RAM.

Pamięć RAM jest dużo wolniejsza, ale odwołania do niej są rzadkie. Należy jednak pamiętać, że komórki pamięci RAM, załadowane do *cache* mogą mieć w RAM nieaktualną zawartość.

W przypadku systemu wieloprocesorowego dane pamiętane w *cache* jednego procesora nie mogą być normalnie pobrane z RAM i wykorzystane przez inny procesor, bo mogą być nieaktualne. Procesor z pamięcią *cache* obserwuje magistralę pamięci, i w przypadku zapisu do komórki, którą posiada w swojej *cache* albo aktualizuje jej zawartość, albo oznacza ją jako nieaktualną.



Powinowactwo pamięci *cache*

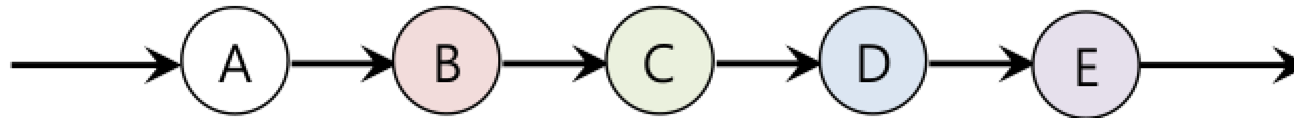
Sytuacja zapełnienia pamięci *cache* procesora danymi procesu na niej obliczanemu nazywana jest **powinowactwem (*affinity*) procesu z procesorem**, albo alternatywnie **powinowactwem pamięci *cache***. Powoduje ono, że w szeregowaniu wieloprocesorowym warto uwzględniać nie tylko kwestię który proces powinien być w danym momencie uruchomiony, ale również na którym procesorze powinien być uruchomiony. Dokładniej, jeśli wykonywanie jakiegoś procesu ma być w kolejnym kwancie czasu kontynuowane, to powinno być kontynuowane na tym samym procesorze.

W praktycznej realizacji strategii planowania opartej na tej zasadzie pojawia się jeszcze jedna ważna kwestia. W systemie może istnieć jedna globalna pula (kolejka) procesów gotowych, bądź mogą być one podzielone pomiędzy procesory i każdy procesor może przetwarzać tylko swoją własną kolejkę gotowych.

Poniżej rozważymy oba podejścia. W tym drugim przypadku oprócz zagadnienia początkowego przydziału procesu do kolejki jednego z procesorów, może jeszcze pojawiać się zjawisko nierównego obciążenia procesorów.

Szeregowanie wieloprocesorowe z pojedynczą kolejką (SQMS)

Rozważmy przykład szeregowania wieloprocesorowego z pojedynczą kolejką zadań:



CPU0	A	E	D	C	B
CPU1	B	A	E	D	C
CPU2	C	B	A	E	D
CPU3	D	C	B	A	E

Kolejne zadania są obsadzane na kolejne wolne procesory, podobnie jak w jednoprocesorowej metodzie RR (*Round Robin*), ale tu metoda działa dokładnie wbrew powinowactwu.

Efekt: metoda kompletnie nieefektywna.

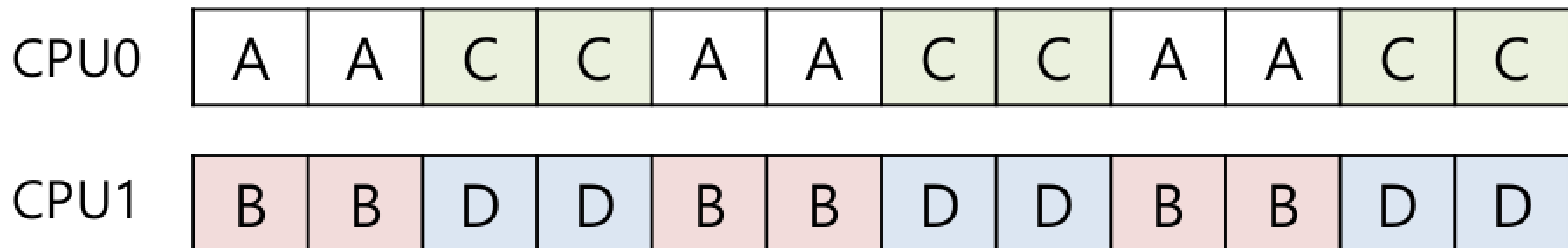
CPU0	A	E	A	A	A
CPU1	B	B	E	B	B
CPU2	C	C	C	E	C
CPU3	D	D	D	D	E

Można próbować realizować schemat zbliżony do RR, ale usiłujący przydzielać procesy na procesory w miarę możliwości wykorzystując powinowactwo.
Jednak taki schemat jest trudny do uogólnienia i do realizacji — praktycznie nie realizuje on zasady RR, ani nie przestrzega powinowactwa.

Szeregowanie wieloprocesorowe z wieloma kolejkami (MQMS)

Okazuje się, że rozdzielenie zbioru wszystkich zadań na kolejki przyporządkowane procesorom pozwala rozwiązać wiele problemów. Rozważmy prosty przykład czterech zadań podzielonych na kolejki dla dwóch procesorów:

i następujący schemat szeregowania każdego procesora metodą RR:

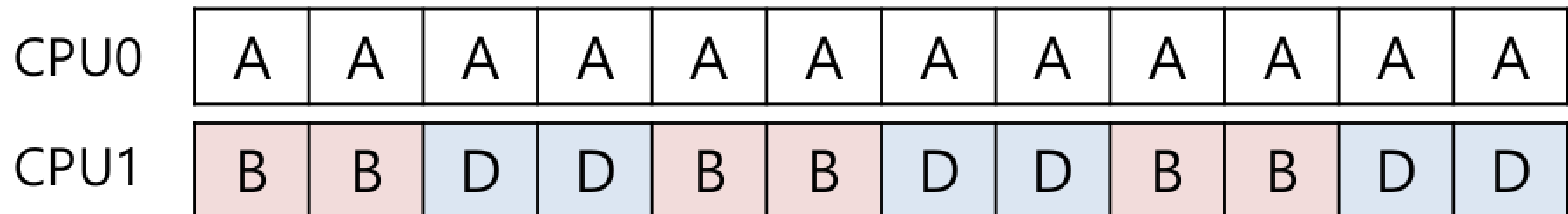


Jak widać, przy niezbyt dużej liczbie procesów metoda może działać bardzo dobrze. Jest czasami możliwe powinowactwo kilku procesów z jednym procesorem naraz, jeśli ich zbiory robocze stron pamięci zmieszczą się łącznie w pamięci *cache* procesora.

Nawet jeśli tak nie jest, bo procesów w lokalnej kolejce jest zbyt dużo, jest nadal możliwe wykorzystania powinowactwa przez takie wydłużenie kwantu czasu obliczeń pojedynczego procesu, jak jest to możliwe z punktu widzenia realizowanej strategii.

Sytuacja komplikuje się nieco, gdy wskutek kontynuacji obliczeń kolejki przypisane różnym procesorom stają się niezrównoważone. Rozważmy sytuację po zakończeniu procesu C:

Metoda RR nadal działa dobrze lokalnie, wykorzystując powinowactwo:

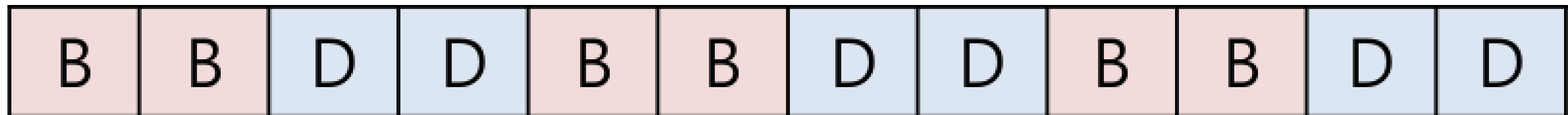


Jednak pojawił się problem nierównomierności obciążenia procesorów, i nierównego dostępu do CPU dla procesów w różnych kolejkach.

Równoważenie obciążeń w metodzie z wieloma kolejkami (MQMS) — migracja procesów

Sytuacja może jeszcze bardziej zdegenerować się po zakończeniu procesu A:

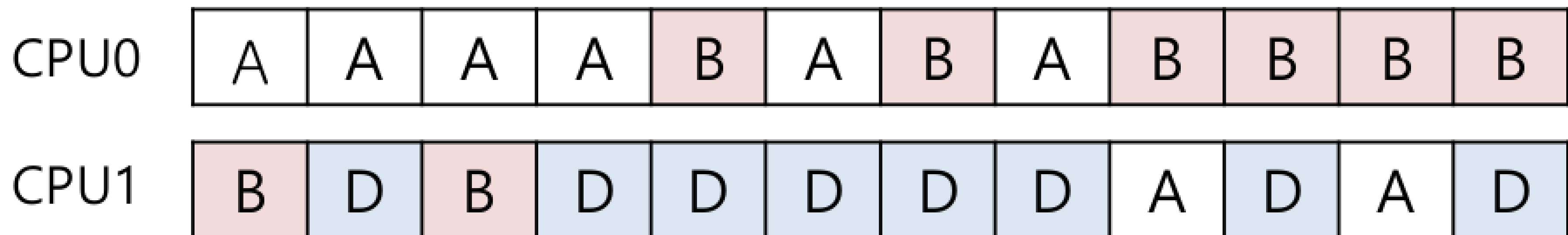
Na jednym procesorze nadal działa poprawnie RR, ale drugi jest kompletnie niewykorzystany:



Rozwiązaniem problemu może być **migracja procesów**. Jeżeli system operacyjny będzie nadzorował stan wszystkich kolejek, to może przenieść jeden z procesów B albo D do kolejki CPU0, i przywrócić poprawne działanie całego schematu MQMS.

Oczywiście, sytuacja nie zawsze jest tak klarowna jak w poprzednim przypadku. Możemy również rozważyć sytuację nierównego obciążenia jak na przykład:

W tym wypadku interwencja systemu i migracja procesu/ów nie przywróci równowagi, tylko ją zmieni na korzyść innego procesora i innej grupy procesów:



Jednak zwykle należy kontynuować taką migrację procesów, dla sukcesywnego równoważenia obciążeń (i sprawiedliwości w przydziale CPU procesom w różnych kolejkach), nawet kosztem okresowego naruszania powinowactwa. Zauważmy, że w dłuższym okresie naruszenie powinowactwa nie ma większego znaczenia, ponieważ aktualność danych pamiętanych w *cache* jest ogólnie krótkotrwała.

Prosty algorytm migracji procesów może polegać na **podkradaniu zadań** przez mało obciążony procesor, z kolejki bardziej obciążonego procesora.

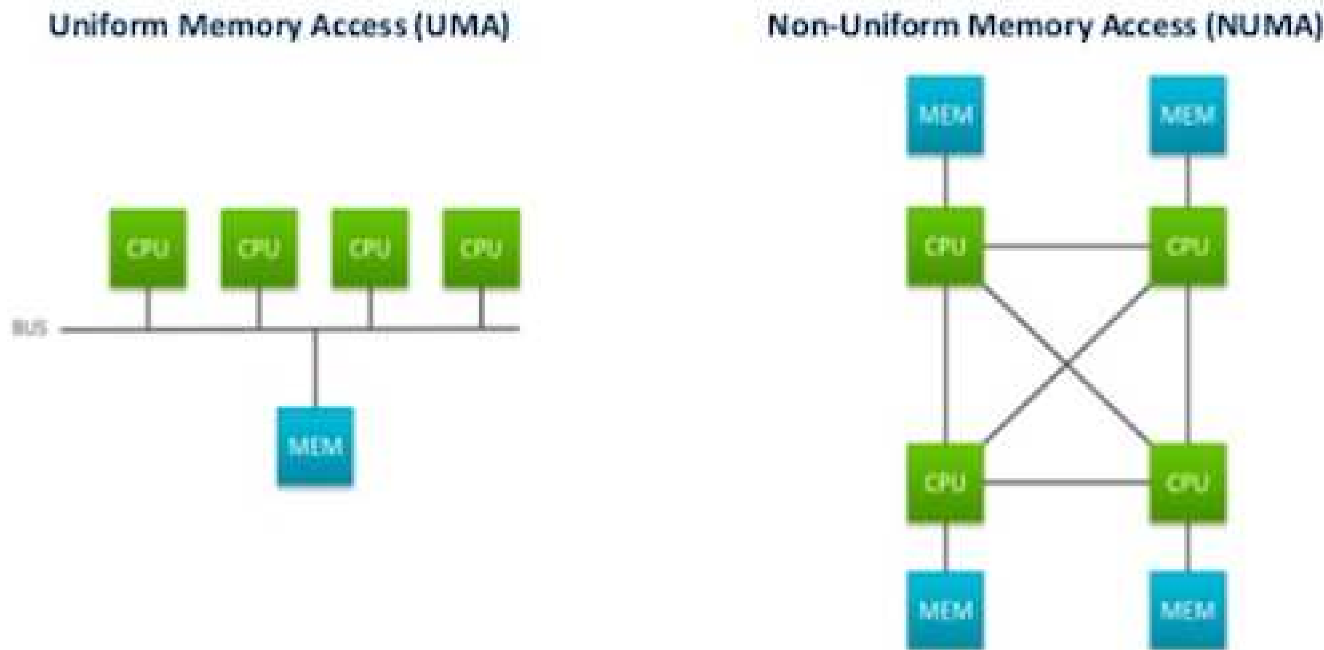
Szeregowanie asymetryczne

Dotychczas omawiane przykłady ilustrowały tak zwane **szeregowanie symetryczne** (*Symmetric Multiprocessing*, SMP). Jego zasadą jest, że wszystkie procesory zajmują się zarówno szeregowaniem jak i wykonywaniem programów.

Istnieje jednak również inne podejście do szeregowania wieloprocesorowego, zwane **szeregowaniem asymetrycznym** (*Asymmetric Multiprocessing*). W tej metodzie jeden z procesorów — desygnowany jako ***Master*** — jest dedykowany do planowania wszystkich zadań, jak również innych zadań globalnych systemu operacyjnego, jak obsługi operacji I/O, itp. Pozostałe procesory wykonują tylko programy użytkowe.

Szeregowanie w architekturze NUMA

Dodatkową okolicznością w szeregowaniu wieloprocesorowym są pewne specjalne architektury sprzętowe. W szczególności architektura wieloprocesorowa NUMA (*Non-Uniform Memory Access*) stwarza takie specjalne wymagania.



W przypadku architektury NUMA ma sens takie szeregowanie procesów, aby były one wykonywane na procesorze fizycznie bliskim blokowi pamięci, w którym przechowywane są dane procesu. W tym wypadku bliskość modułu pamięci, i koszt transferów z i do RAM, ma znaczenie zasadnicze. Powinowactwo pamięci *cache* jest nadal brane pod uwagę, gdy już zostanie zapewnione, że wcześniejsze wykonanie danego procesu odbyło się na „właściwym" z punktu widzenia architektury NUMA procesorze.

Krótkie podsumowanie — pytania sprawdzające

Odpowiedz na poniższe pytania:

1. Jak działa pamięć buforowa *cache* procesora? W jaki sposób wpływa ona na przyspieszenie obliczeń?
2. W jaki sposób może dojść do niespójności zawartości *cache* i jak można rozwiązać ten problem?
3. Co nazywamy powinowactwem procesora, i jaki ma ona związek z szeregowaniem procesów?
4. Na czym polega szeregowanie symetryczne z jedną kolejką zadań SQMS, i dlaczego ta metoda nie zawsze się sprawdza?
5. W jakich sytuacjach źle działa szeregowanie symetryczne z wieloma kolejkami zadań MQMS, i w jaki sposób można przywrócić poprawne działanie tej metody?
6. Na czym polega asymetryczne szeregowanie wieloprocesorowe?
7. Jak można uwzględnić specyfikę architektury NUMA w szeregowaniu procesów?

Planowanie statyczne i dynamiczne

Algorytmy planowania mają na celu zapewnienie spełnienia wymagań czasowych całego systemu. Muszą podejmować decyzje o przydzielaniu zasobów systemu biorąc pod uwagę najgorszy możliwy przypadek, lub czas odpowiedzi. Można podzielić strategie planowania na: planowanie przed wykonaniem, i planowanie w czasie wykonywania.

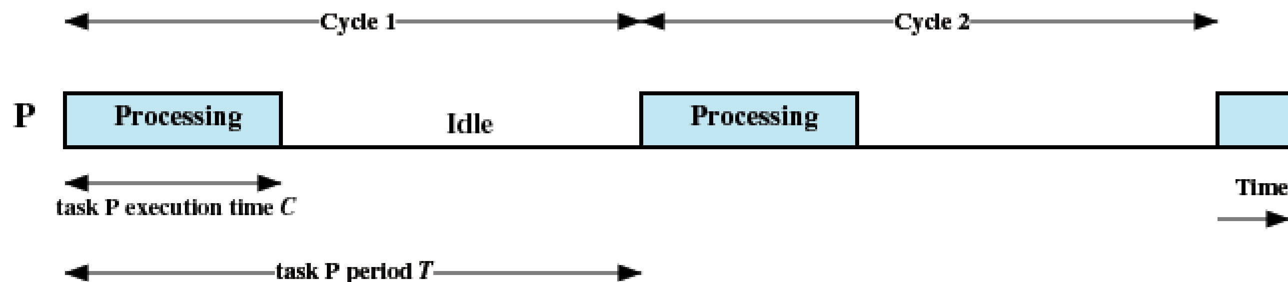
Celem **planowania przed wykonaniem**, albo inaczej: **planowania statycznego**, jest wyznaczenie odpowiedniej kolejności wykonywania zapewniającej spełnienie ograniczeń i bezkolizyjny dostęp do zasobów systemu. Planowanie przed wykonaniem może również minimalizować pewne narzuty systemowe, takie jak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

Planowanie w czasie wykonywania, inaczej: **planowanie dynamiczne**, polega na przyznawaniu zadaniom priorytetów, a następnie przydzielanie zasobów zadaniom 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.

Na przykład, poznana wcześniej metoda planowania priorytetowego jest metodą planowania statycznego, a z wykorzystaniem dynamicznej modyfikacji priorytetów jest również metodą planowania dynamicznego.

Zadania okresowe

W systemach czasu rzeczywistego często mamy do czynienia z zadaniami okresowymi. Zadania takie muszą być wykonywane cyklicznie w nieskończonej pętli powtórzeń. Ponieważ każdorazowe tworzenie, a następnie kasowanie zadania byłoby nieefektywne, zatem system operacyjny, który wspiera zadania okresowe, po zakończeniu zadania reinicjalizuje je, oblicza ich czas kolejnego uruchomienia, i umieszcza w kolejce zadań oczekujących, a na początku kolejnego okresu **wyzwala je** (*release*), tzn. umieszcza w kolejce zadań gotowych.

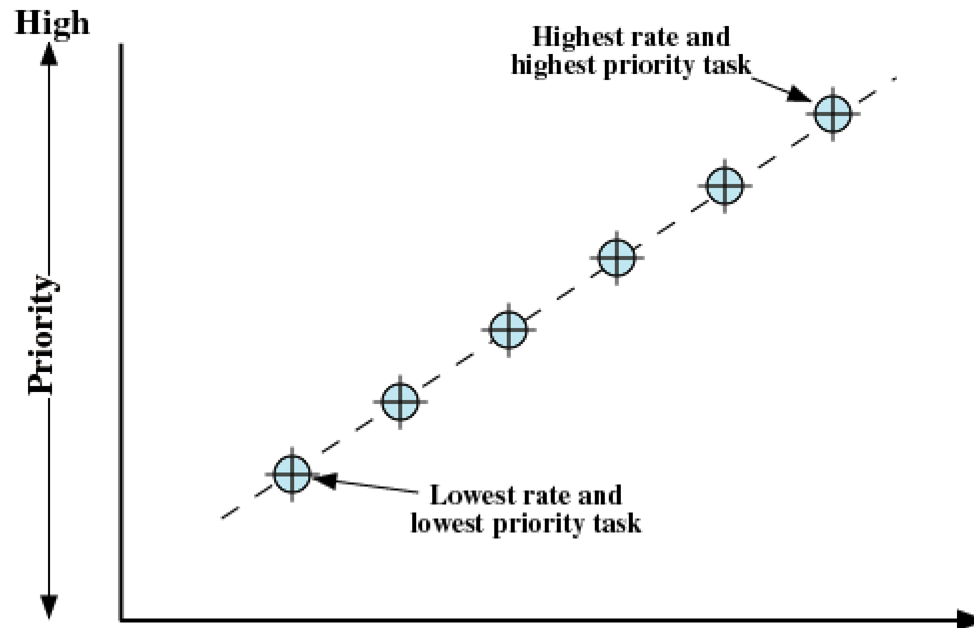


Możemy traktować instancje danego zadania jako oddzielne zadania, które podlegają szeregowaniu przez system.

Tylko niektóre systemy operacyjne wspierają zadania okresowe. 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.

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 jest metodą szeregowania z wywłaszczaniem.

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 zadań okresowych 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 wydajność 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 zapewnić teoretyczną gwarancję, że system będzie w stanie wykonywać wszystkie zadania zgodnie z ich okresami, a gdyby nie było to pewne dla danego zestawu zadań, to jaka dodatkowa moc procesora jest potrzebna aby tę gwarancję uzyskać.

Warunek szeregowalności podany w twierdzeniu nie jest jedynym znanym warunkiem wystarczającym szeregowalności algorytmem RMS. Istnieją również inne warunki wystarczające, gwarantujące szeregowalność RMS dla zestawów zadań spełniających dodatkowe założenia, przy wyższym limicie wykorzystania procesora. Na przykład, gdy okresy każdej pary zbioru zadań mają relację harmoniczną (jeden jest wielokrotnością drugiego), to taki zestaw zadań jest szeregowalny algorytmem RMS do $U = 100\%$.

Szeregowanie częstotliwościowe (RMS) — własności (cd.)

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

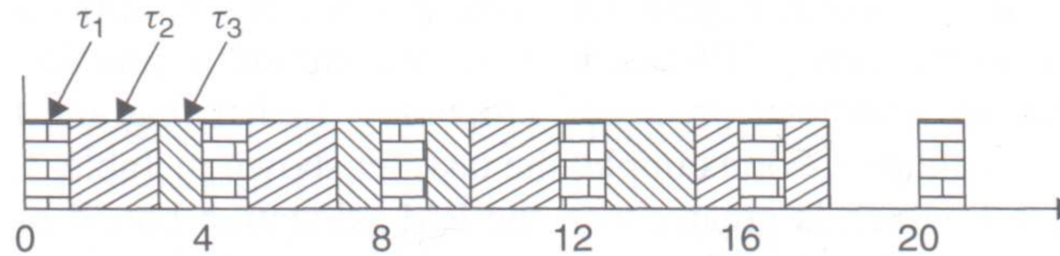
Pomimo podanego teoretycznego limitu szeregowalności algorytm RMS jest prawdopodobnie najpopularniejszym algorytmem szeregowania stosowanym w systemach czasu rzeczywistego. Składa się na to bardzo prosta implementacja, oraz brak dodatkowych założeń dla zestawu zadań (takich jak w algorytmie EDF), których nie wspiera wiele systemów operacyjnych czasu rzeczywistego.

W wielu przypadkach jest również możliwe planowanie algorytmem RMS zestawu zadań przekraczających limit teoretyczny wykorzystania procesora. Złożone systemy czasu rzeczywistego osiągają często wykorzystanie procesora rzędu 80% bez większych problemów. Dla losowo wygenerowanego zestawu zadań okresowych, poprawne szeregowanie jest możliwe do wartości około 85%, ale bez żadnych gwarancji (niektórzy autorzy podają wartość 88%).

Zauważmy ponadto, że jeśli uruchomione zadania okresowe wykorzystują mniej niż $U = 100\%$ procesora, to można w takim systemie uruchomić dodatkowe zadania, niedziałające w czasie rzeczywistym, które mogą wykorzystać pozostałą moc procesora.

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 EDF

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

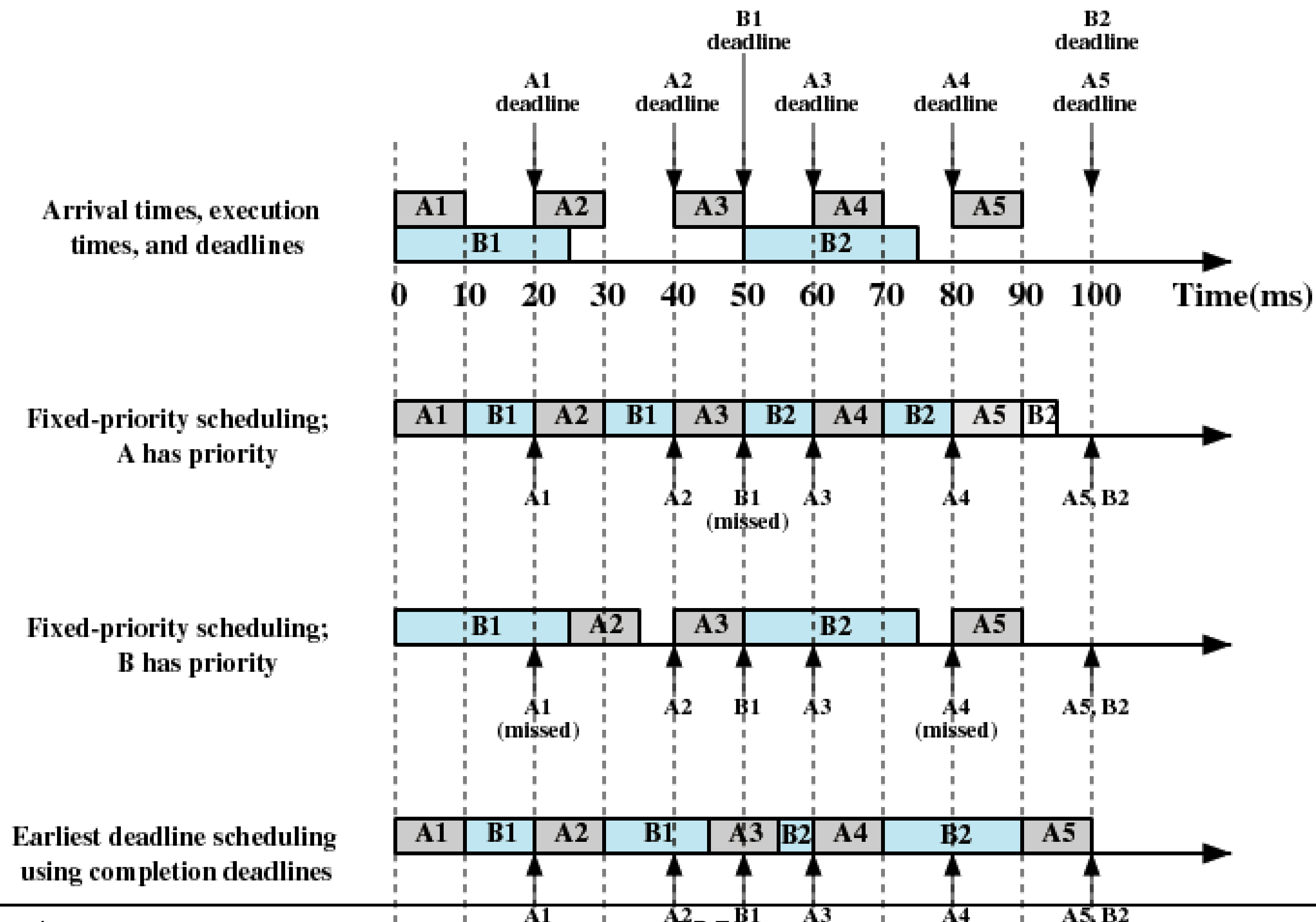
	czas uwolnienia	czas wykonania	termin zakończenia
τ_1	0	8	20
τ_2	3	2	5
τ_3	5	3	10

Szeregowanie terminowe bierze pod uwagę termin zakończenia wszystkich zadań określony względem czasu ich rozpoczęcia. Dla zadań okresowych typowym terminem 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 wymagany termin wykonania jest najwcześniejszy.

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 okresowych — własności

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

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

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

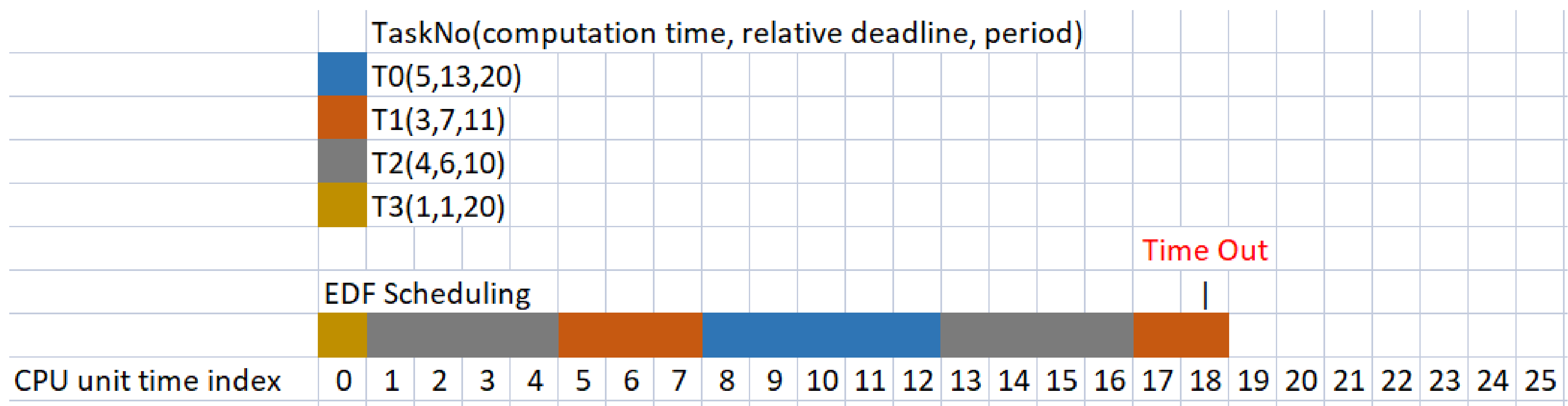
Algorytm szeregowania terminowego EDF dla pojedynczego procesora z wywłaszczaniem jest optymalny w takim sensie, że jeśli dany zbiór zadań — każde z określonym okresem, czasem uwolnienia, czasem obliczeń, i terminem zakończenia równym końcowi jego okresu — łącznie nie przekraczają współczynnika wykorzystania procesora $U = 100\%$, to ten zbiór zadań jest szeregowalny algorytmem EDF.

Warunek szeregowalności zestawu zadań algorytmem EDF do 100% wykorzystania procesora stanowi przewagę tego algorytmu nad RMS. Jednak ta strategia szeregowania nie jest dostępna we wszystkich systemach operacyjnych czasu rzeczywistego.

Szeregowanie terminowe — przypadki szczególne

Typowym przypadkiem szeregowania terminowego dla zadań okresowych jest gdy terminy zadań przypadają na koniec okresu. Nie zawsze musi to być właściwe. Dany system może wymagać by pewne zadania były ukończone w określonym momencie, przed końcem ich okresu. Jednak wtedy nie obowiązuje gwarancja szeregowalności do limitu wykorzystania procesora $U = 100\%$.

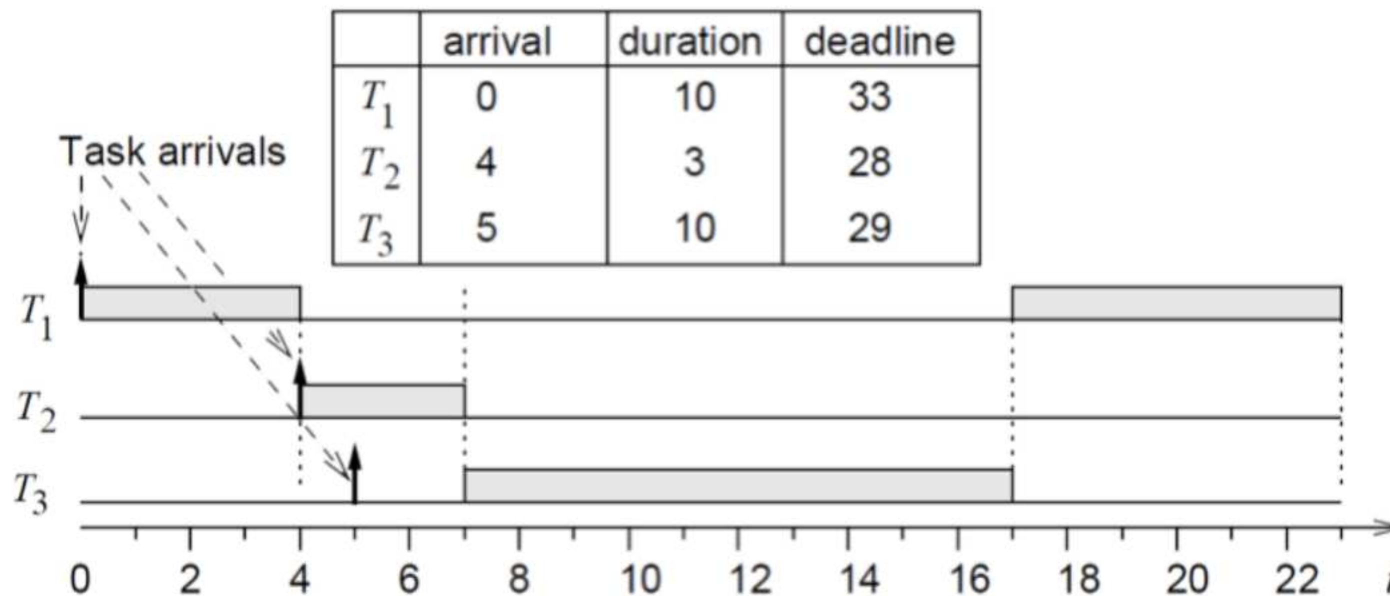
Rozważmy następujący przykład szeregowania czterech zadań okresowych (źródło: Wikipedia). Zadania są określone przez trzy parametry: czas obliczeń jednej instancji, względny termin wykonania w ramach okresu, i okres zadania. W tym przykładzie łączne wykorzystanie procesora $U = 5/20 + 3/11 + 4/10 + 1/20 \approx 0.97\%$, jednak ten zestaw zadań nie jest szeregowalny algorytmem EDF.



Szeregowanie terminowe — 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 również zastosować algorytm EDF.

Rozważmy przykład:



	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29

Zadanie T_1 jest jedyne w chwili 0, więc jest natychmiast uruchamiane. W chwili 4 pojawia się zadanie T_2 z wcześniejszym terminem, więc T_1 zostaje wywłaszczone. W chwili 5 pojawia się zadanie T_3 z późniejszym terminem, więc wywłaszczenia nie ma. Zadanie T_2 wykonuje się do końca, potem T_3 i ostatnie wznawiane jest T_1.

Algorytm EDF dla zadań nieokresowych — własności

Algorytm EDF jest optymalny dla pojedynczego procesora z wywłaszczaniem. Inaczej można powiedzieć, że jeśli istnieje poprawny harmonogram wykonania zestawu zadań zgodnego z ich terminami, to EDF będzie działał poprawnie.

Jednak w przypadku szeregowania bez wywłaszczania EDF nie jest optymalny. Dlatego traktujemy EDF jako algorytm szeregowania z wywłaszczaniem.

EDF również nie jest optymalny dla szeregowania z więcej niż jednym procesorem!

Przykład: przedstawiony po prawej zestaw zadań nieokresowych jest szeregowalny dla dwóch procesorów, lecz EDF doprowadzi T_3 do przekroczenia terminu.

	wyzw.	czas	termin
T_1	0	1	1
T_2	0	1	2
T_3	0	5	5

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: $E_1 = 25$, $P_1 = 50$, $E_2 = 30$, $P_2 = 75$. Przedstaw wynik na diagramie czasowym. 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. Wynik przedstaw na diagramie czasowym.