Planowanie zadań

Proces jest dynamicznym obiektem wykonującym w systemie operacyjnym określony program. **Planowaniem** (*scheduling*), lub alternatywnie **szeregowaniem**, nazywamy czynność rozdzielania czasu procesora na poszczególne procesy, zapewniającą ich *quasi*-równoległe wykonywanie.

Proces jest podstawową jednostką roboczą planowaną przez system. W niektórych systemach operacyjnych występują wątki — w dodatku lub zamiast procesów — i wtedy one podlegają planowaniu w podobny sposób. W kontekście systemów czasu rzeczywistego często mówi się o planowaniu **zadań** nie rozróżniając, czy są one procesami (z oddzielnym przydziałem pamięci), czy wątkami (współdzielącymi przydział pamięci programu).

Przełączanie kontekstu

Zastępowanie procesu dotychczas wykonywanego na procesorze innym procesem, wynikające z planowania, wymaga zapisania stanu obliczeń w taki sposób, aby było możliwe ich wznowienie w sposób niezauważony przez proces. Następnie, załadowanie i uruchomienie innego procesu wymaga odtworzenia stanu wszystkich rejestrów procesora, wcześniej zapamiętanych.

Ta operacja składowania i odtwarzania stanu procesora ze wszystkimi rejestrami nazywana jest **przełączeniem kontekstu**. Stanowi ona istotny narzut na wykonywanie procesu, albo dodatkowe obciążenie systemu, ponieważ czas poświęcony na przełączanie kontekstu jest tracony, procesor nie wykonuje w tym czasie zadanych obliczeń. Przełączanie kontekstu nie może zatem być wykonywane zbyt często, ponieważ w takim przypadku czas na nie tracony byłby znaczący.

Szybkość przełączania kontekstu zależy też od wsparcia sprzętowego. W niektórych architekturach występuje kilka zbiorów rejestrów. Wówczas przełączenie kontekstu może polegać na przełączeniu wskaźnika bieżącego zestawu rejestrów, i nie trzeba ich zapamiętywać i odtwarzać.

Planista i dyspozytor

W realizacji planowania biorą udział następujące moduły systemu operacyjnego:

- **planista** (*scheduler*) — podejmuje decyzje, który z procesów gotowych powinien być następnie wykonywany na procesorze,
- **dyspozytor** (*dispatcher*), zwany czasami alternatywnie **egzekutorem** albo **ekspedytorem** — wykonuje wszystkie czynności administracyjne związane z przełączaniem kontekstu przy przenoszeniu jednych procesów do kolejki gotowych, a innych na procesor.

Moduły planisty i dyspozytora normalnie zlokalizowane są w **jądrze** (*kernel*) systemu operacyjnego. W niektórych systemach, moduł planowania może być wydzielony z jądra (zwanego wtedy mikrojądrem) i wykonywany jako oddzielny proces.

Łączny czas pracy planisty i dyspozytora jest dodatkowym narzutem administracyjnym. Zatem ich działania muszą być podejmowane na tyle rzadko, by nie wpływały nadmiernie na efektywność działania systemu.

Z drugiej strony, z punktu widzenia *quasi*-równoległego wykonywania procesów wymiana procesu obliczanego na procesorze powinna następować często.

Jest to podstawowy kompromis, który muszą rozwiązywać strategie planowania.

Okres planowania

Wykonywanie algorytmu planisty, który wybiera następny proces do wykonywania na procesorze, następuje okresowo, w momencie **przerwania zegarowego**, kiedy system aktualizuje swoje informacje o czasie i timerach, i oblicza swój budżet czasowy. Typowym okresem przerwania zegarowego jest 10 milisekund (starsze systemy, Linux 2.4), albo 1 milisekunda (Linux 2.6 i nowsze systemy).

Planista jest wywoływany również po wystąpieniu zdarzeń, które mogą mieć wpływ na decyzje planistyczne, takich jak:

- utworzenie nowego procesu, który jest gotowy i powinien być planowany,
- zakończenie procesu i zwolnienie procesora,
- dobrowolne zwolnienie procesora,
- żądanie zasobu, które nie może być natychmiast spełnione (np. semafora),
- przerwanie od urządzenia wejścia/wyjścia; jeśli jakieś urządzenie zakończyło transfer, to może trzeba uruchomić proces, który oczekiwał na te dane.

Zatem system operacyjny wykonuje jednocześnie **planowanie zegarowe** i **zdarzeniowe**.

Wywłaszczanie

Można wyróżnić dwa rodzaje algorytmów planowania:

niewywłaszczeniowe

Proces jest usuwany z procesora wyłącznie w momencie jego zakończenia, albo gdy sam dobrowolnie poprosi o przeniesienie do kolejki gotowych (patrz np. funkcja systemowa `sched_wait`).

Takie planowanie ogranicza do minimum aktywność planisty i dyspozytora i maksymalizuje użyteczny czas procesora.

wywłaszczeniowe

Proces może być usunięty z procesora w wyniku podjęcia takiej decyzji przez planistę w którymkolwiek z momentów wymienionych wcześniej.

Przy stosowaniu planowania niewywłaszczeniowego jest możliwe, że proces będzie wykonywał się na procesorze bardzo długo (np. wiele godzin lub dni) nie dopuszczając innych procesów do wykonania.

Zagłodzenie i sprawiedliwość

Niektóre strategie planowania mogą doprowadzić do sytuacji, kiedy pewien proces, lub procesy, będą wielokrotnie pomijane i nieuruchamiane przez planistę, w zgodzie z przyjętym algorytmem planowania. Jest to możliwe jeśli stale będą się pojawiały nowe procesy, które powinny być uruchomione wcześniej.

To zjawisko zwane jest **zagłodzeniem** procesu, i jest niepożądanym efektem ubocznym pewnych algorytmów planowania. W związku z tym w tych algorytmach stosuje się dodatkowe zabezpieczenia, ingerujące w algorytm planowania i nie dopuszczające do zagłodzenia.

Unikanie zagłodzenia procesów jest wyrazem ogólnej zasady „sprawiedliwości" (ang. *fairness*), albo **egalitaryzmu**, obowiązującą w systemach operacyjnych ogólnego przeznaczenia (*GPOS — General Purpose Operating System*). Zasada ta mówi, że równoprawne z punktu widzenia systemu procesy powinny otrzymać równy dostęp do zasobów systemu. Obowiązuje ona w przydziale: procesora, pamięci, i innych zasobów.

Zauważmy, że zasadę tę można odnosić do wszystkich procesów równo, albo też do wszystkich użytkowników, to znaczy do grup procesów poszczególnych użytkowników.

Cele planowania

Można brać pod uwagę różne kryteria jakości planowania procesów, i dostosować do nich strategię planowania. Te kryteria są jednak do pewnego stopnia, lub całkowicie, ze sobą sprzeczne. Planowanie może zatem optymalizować:

wykorzystanie procesora

średni czas efektywnych obliczeń, odwrotność czasu traconego na puste cykle, oraz czasu zużytego na przełączenia kontekstu,

przepustowość

średnią liczbę procesów wykonanych na jednostkę czasu,

czas przetwarzania

średni całkowity czas przetwarzania zadania, od pojawienia się do zakończenia,

czas oczekiwania

średni sumaryczny czas spędzony przez proces w kolejce procesów gotowych.

W dodatku do powyższych kryteriów obowiązują ograniczenia: równe traktowanie procesów i niedopuszczenie do zagłodzenia.

Krótkie podsumowanie — pytania sprawdzające

Odpowiedz na poniższe pytania:

1. Co to jest przełączanie kontekstu?

2. Jaki jest związek pomiędzy zadaniami planisty i dyspozytora?

3. Jaką rolę pełni wywłaszczanie procesów?

4. Dlaczego może dojść do zagłodzenia procesu w systemie operacyjnym?

5. Na czym polega sprawiedliwość w planowaniu procesów?

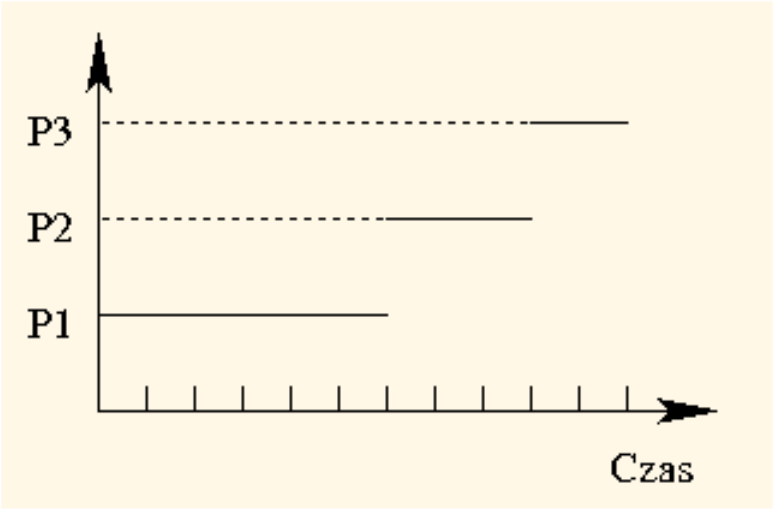
6. Jakie kryteria może optymalizować algorytm planowania?

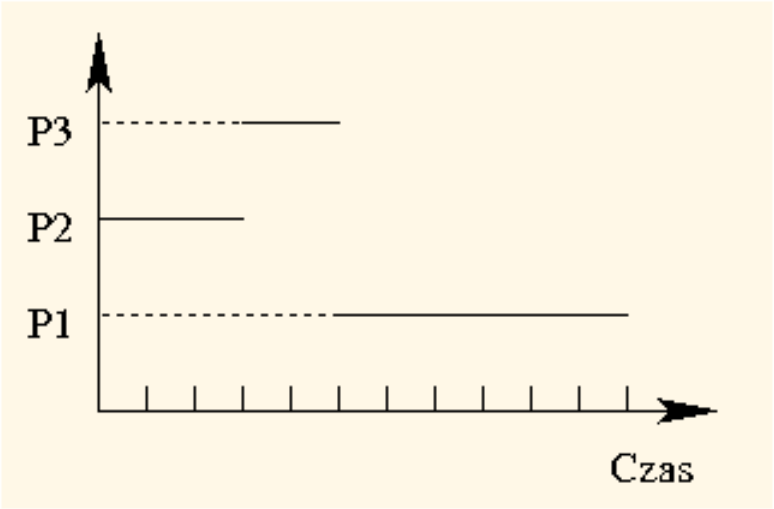
Strategia FCFS

Najprostszym algorytmem planowania jest FCFS (*First-Come, First-Served*), czyli „pierwszy przyszedł pierwszy obsłużony.” FCFS jest strategią bez wywłaszczania. Procesy są wykonywane od początku do końca w takiej kolejności, w jakiej się pojawiły.

Proces	P1	P2	P3
Czas wykon.	6	3	2

Proces	P2	P3	P1
Czas wykon.	3	2	6



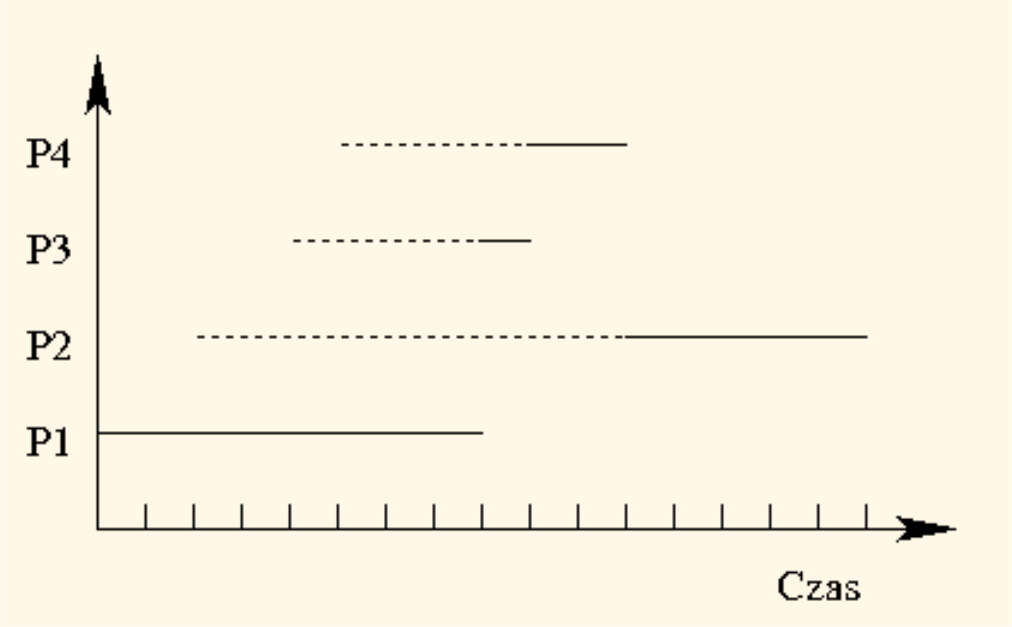


Warto zwrócić uwagę na to, że zadanie planisty przy tej strategii jest zredukowane do minimum (utrzymywanie kolejki). Zatem algorytm minimalizuje narzuty planowania. Planista nie musi też znać wymagań czasowych zadania.

Strategia SJF

Inną strategią planowania bez wywłaszczania jest SJF (*Shortest-Jobtime-First*), czyli „najpierw najkrótsze zadanie." Spośród wszystkich gotowych uruchomiony zostaje proces, który ma najkrótszy całkowity czas wykonywania. Jego celem jest minimalizacja średniego czasu przetwarzania zadania. W tym celu jednak ten czas dla wszystkich procesów musi być znany planiście.

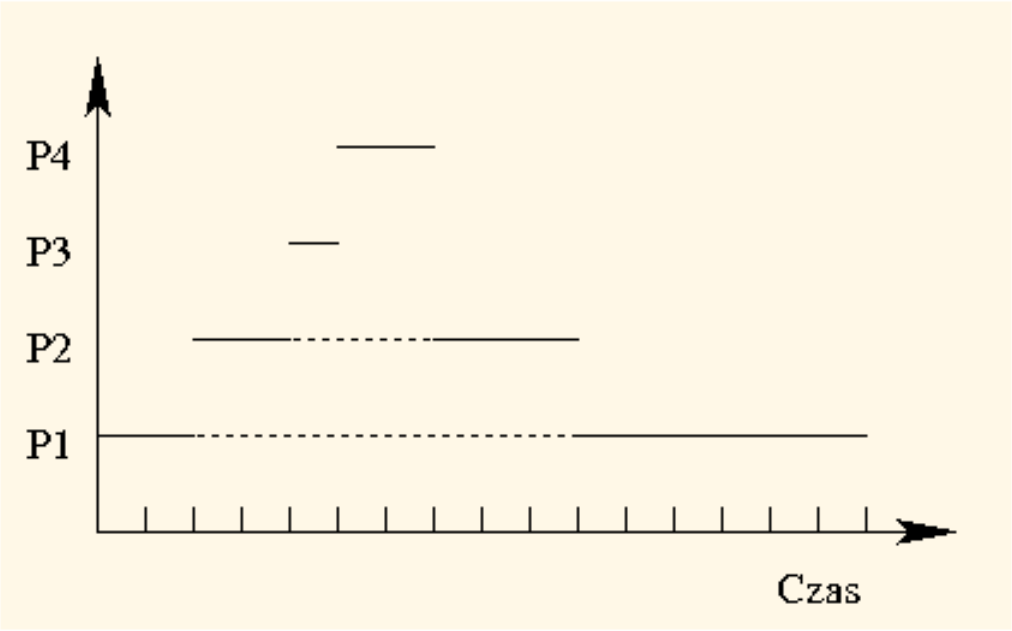
Proces	Czas uwoln.	Czas wykon.
P4	5	2
P3	4	1
P2	2	5
P1	0	8



Strategia SRTF

Odmianą algorytmu SJF z dodanym wywłaszczaniem jest SRTF (*Shortest-Remaining-Time-First*), czyli „najpierw proces o najkrótszym pozostałym czasie wykonania." Zauważmy, że tu planista musi być wywołany, poza momentem zakończenia każdego procesu jak zwykle, jedynie w chwili pojawienia się nowego procesu, wybierając ten z najkrótszym pozostałym przewidywanym czasem obliczeń.

Proces	Czas uwoln.	Czas wykon.
P4	5	2
P3	4	1
P2	2	5
P1	0	8



Proces	P1	P2	P3	P4	P2	P1
Czas rozpoczęcia	0	2	4	5	7	10
Czas wykonany	2	2	1	2	3	6
Czas pozostały	6	3	0	0	0	0

Szacowanie fazy procesora

Zastanówmy się, jak oszacować czas następnej fazy procesora, potrzebny do realizacji strategii SJF i SRTF?

Planista z reguły nie wie ile będzie trwała kolejna faza procesora. Jednak gdy traktujemy czas życia procesu jako składający się z szeregu faz obliczeniowych, przedzielonych fazami I/O, to system zna długości jego poprzednich faz obliczeniowych. Zwykle fazy kolejne mają podobną charakterystykę jak poprzednie.

Jedną ze stosowanych metod szacowania następnej fazy procesora na podstawie długości poprzednich jest uśrednianie wykładnicze. Długość kolejnej fazy procesora szacujemy jako średnią ważoną długości poprzednich faz, przy czym wagi tworzą rosnący ciąg geometryczny (największą wagę ma ostatnia faza).

Zadania wsadowe i interakcyjne

W systemach operacyjnych mogą pojawiać się zadania o różnej charakterystyce. Dwie ważne grupy zadań stanowią: (i) **zadania obliczeniowe** (zwane również wsadowymi), które typowo wykonują obliczenia na procesorze i odwołują się głównie do pamięci RAM, i (ii) **zadania interakcyjne**, które typowo oczekują na dane z interfejsów I/O i wykonują krótkie obliczenia, lub kolejne transfery I/O, w odpowiedzi na otrzymane dane.

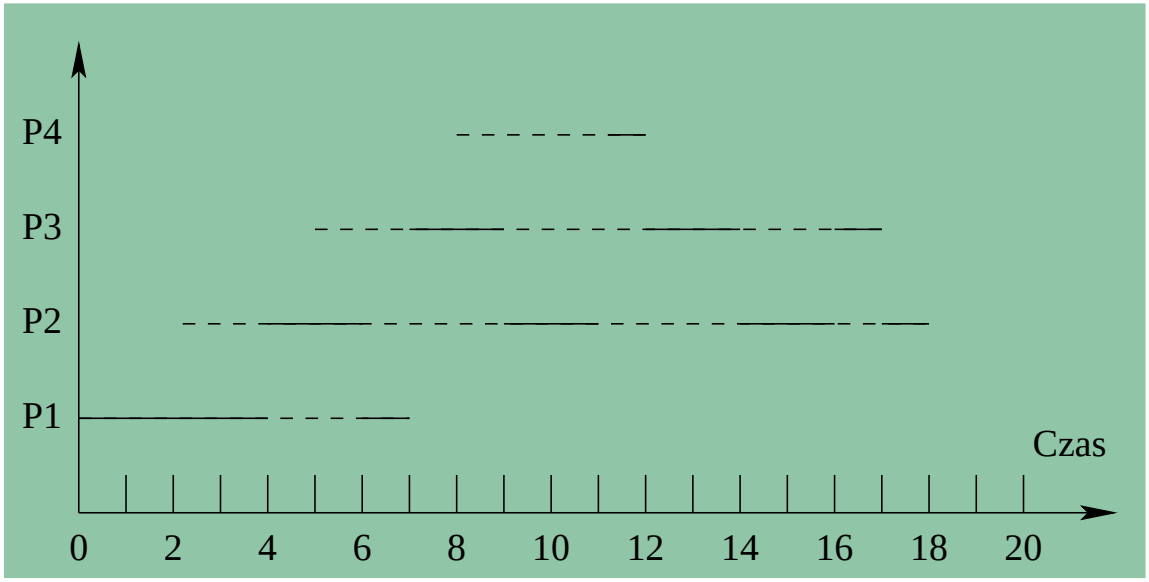
Zadania interakcyjne mogą być związane z pracą człowieka na komputerze typu PC, ale mogą to być również serwery odpowiadające na żądania napływające z sieci, albo przetwarzające dane z pomiarów, itp. Odmienny profil tych zadań powoduje, że przy ich szeregowaniu znaczenie mają różne kryteria, i lepiej nadają się do nich różne algorytmy szeregowania.

Omówione dotychczas algorytmy szeregowania FCFS, SJF, i SRTF optymalizują głównie wykorzystanie procesora i czas przetwarzania. Powoduje to, że dobrze nadają się do planowania zadań obliczeniowych, natomiast nie bardzo do zadań interakcyjnych, gdzie podstawowym kryterium powinny być: czas oczekiwania i przepustowość.

Planowanie rotacyjne

RR (*Round-Robin*), czyli planowanie rotacyjne, to strategia, w której każdy proces po kolei otrzymuje kwant czasu procesora do wykorzystania, po czym jest wywłaszczany. Zwykle jest to około 10-100 milisekund. Żaden proces nie czeka więc dłużej na procesor niż długość kwantu razy liczba procesów.

Proces	Czas uwoln.	Czas wykon.
P4	8	1
P3	5	5
P2	2	7
P1	0	5



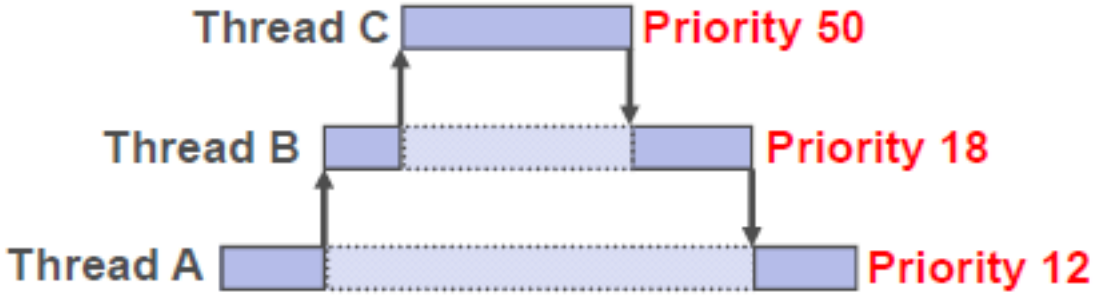
Proces	P1	P1	P2	P1	P3	P2	P4	P3	P2	P3	P2
Czas rozpoczęcia	0	2	4	6	7	9	11	12	14	16	17
Czas wykonany	2	2	2	1	2	2	1	2	2	1	1
Czas pozostały	3	1	5	0	3	3	0	1	1	0	0

Planowanie rotacyjne jest zupełnie inną strategią planowania od przedstawionych wcześniej. Nadaje się ono głównie do planowania zadań interakcyjnych, gdzie najważniejszym kryterium jest krótki czas reakcji.

Planowanie rotacyjne powoduje znacznie więcej przełączeń kontekstu niż poprzednie algorytmy planowania, jest więc dość kosztowną strategią. Kwant czasu przydzielany procesom musi być co najmniej o rząd wielkości większy niż czas przełączenia kontekstu. W przeciwnym wypadku narzuty generowane przez planowanie będą znaczące, a nawet mogą przewyższyć rzeczywistą pracę wykonywaną przez system na obliczenia procesów.

Planowanie priorytetowe

Jeżeli dla każdego procesu określimy liczbę reprezentującą jego **priorytet**, to możliwe jest **planowanie priorytetowe**. Procesor jest przydzielany procesowi, który ma największy priorytet. Możemy rozróżnić wersję wywłaszczeniową, gdy proces jest usuwany z procesora natychmiast po pojawieniu się procesu o wyższym priorytecie, oraz wersję niewywłaszczeniową, gdy procesy okresowo dobrowolnie zwalniają procesor, i żaden nie jest wywłaszczany.



Planowanie priorytetowe jest atrakcyjne ponieważ pozwala łatwo określić, który proces będzie wykonywany przed którym. Dlatego bywa chętnie stosowane w systemach gdzie konieczne jest zagwarantowanie określonej kolejności wykonania procesów, jak systemy czasu rzeczywistego, systemy wbudowane, systemy wojskowe, itp.

Jednak prawidłowe zdefiniowanie priorytetów dla określonych wymagań jest proste tylko gdy system jest mały, typu kilku do kilkunastu procesów. Gdy liczba procesów jest większa, określenie właściwych wartości priorytetów zaczyna być problemem.

Priorytety statyczne i dynamiczne

Priorytety mogą być przydzielone zadaniom sztywno (statycznie), według schematu określonego przez programistę lub administratora systemu. Jednak to da się zrobić jedynie w odniesieniu do systemów zamkniętych, w których istnieje określona pula zadań o dobrze znanych własnościach.[1]

W systemach otwartych priorytety muszą być przydzielane pojawiającym się nieznanym zadaniom. Inteligentny planista może przydzielać zadaniom wstępne priorytety według jakiegoś przyjętego schematu, a następnie dynamicznie aktualizować je na podstawie obserwacji ich pracy.

Sensowną strategią byłoby rozpoznawanie zadania jako obliczeniowego, i obniżanie jego priorytetu, lub jako interakcyjnego, i podwyższanie priorytetu. Wtedy maksimum zasobów obliczeniowych będzie przydzielane procesom interakcyjnym, natomiast procesy obliczeniowe będą wykonywane tylko w przerwach, gdy wszystkie procesy interakcyjne dobrowolnie zwolnią procesor. W ten sposób, planowanie priorytetowe nadaje się zarówno do planowania procesów obliczeniowych, interakcyjnych, jak i mieszanych.

[1] A nawet w takich systemach statyczny przydział priorytetów jest nietrywialnym zadaniem, gdy pojawia się duża liczba procesów o złożonych zadaniach.

Strategie złożone

W systemie planującym wykonanie zadań algorytmem priorytetowym może pojawić się wiele zadań o tym samym priorytecie. Wcale nie musi to być przypadek ani sytuacja wyjątkowa. Zadania zwykle mają priorytety przydzielane według pewnego schematu. Jeśli więc zostanie uruchomiona pewna liczba podobnych zadań to będą one mieć równe priorytety.

A wtedy, spośród grupy zadań, które wszystkie mają najwyższy priorytet, które powinno zostać uruchomione pierwsze? Przypadkowy ich wybór zwykle nie jest dobrym rozwiązaniem, bo jego wynikiem będzie nierównomierny czas oczekiwania na wykonanie zadań, które teoretycznie powinny być traktowane jednakowo.

Dobrym rozwiązaniem jest zastosowanie drugorzędnego algorytmu planowania, który będzie wybierał zadania spośród tych o najwyższym priorytecie. Ponieważ powinny one być równoważne, i równo traktowane, dlatego w roli drugorzędnego algorytmu planowania często stosowany jest RR.

Wielopoziomowe kolejki

Przedstawiliśmy strategie dobre do szeregowania procesów obliczeniowych (SJF i SRTF) oraz procesów interakcyjnych (RR). Jeśli w systemie będą obecne procesy obu rodzajów, to system stosujący wyłącznie jedną wybraną strategię będzie przetwarzał część procesów nieoptymalnie.

Zamiast tego, można użyć dla każdej z tych grup procesów innej, odpowiedniej dla danej grupy, strategii planowania. Procesy obliczeniowe mogą być obsługiwane strategią SJF, ponieważ w przypadku pracy wsadowej ważna jest minimalizacja średniego czasu oczekiwania. Natomiast procesy interakcyjne mogą być obsługiwane strategią RR, ponieważ w przypadku pracy interakcyjnej istotna jest minimalizacji czasu reakcji.

Typ procesu może być określony przez deklarację użytkownika, albo przez rozpoznanie jego charakterystyki przez system, jak w przypadku dynamicznych priorytetów.

Należy jeszcze określić wzajemny stosunek obu tych grup procesów. Oczywiście procesy interakcyjne powinny być uprzywilejowane, stąd często nazywa się je procesami **pierwszoplanowymi**, albo procesami **czoła**, a procesy obliczeniowe procesami **tła**. W najprostszym przypadku można przyjąć, że procesy czoła będą wykonywane zawsze gdy są gotowe, w oczekiwaniu, że pozostawią one część cykli obliczeniowych niewykorzystanych, umożliwiając wykonywanie zadań tła.

Partycjonowanie

Opisane wielopoziomowe kolejki z uprzywilejowanymi procesami czoła mogą w pewnych przypadkach doprowadzić do zagłodzenia procesów tła. Takie ryzyko może być akceptowalne w pewnych przypadkach, lecz ogólnie zjawisko jest niepożądane w systemach operacyjnych.

Można temu zapobiec, przyjmując stały podział czasu między kolejki procesów tła i czoła, np. 90% dla czoła i 10% dla tła. Jest to strategia zwana **partycjonowaniem**.

Partycjonowanie może samo w sobie być uważane za algorytm planowania. Jednak podobnie jak planowanie priorytetowe, może ono wydzielić grupę zadań do wykonania bez możliwości wyboru zadania w ramach grupy. Jest to zatem algorytm wysokiego poziomu, który należy uzupełnić o dodatkową strategię, pozwalającą szeregować zadania w ramach poszczególnych partycji.

Wadą planowania przez partycjonowanie jest potencjał do marnowania wolnej mocy procesora. Jeśli zadania z jednej partycji nie wykorzystują wszystkich cykli procesora dostępnych w tej partycji, to zadania z innej partycji nie mogą ich wykorzystać. Jest możliwe **partycjonowanie adaptacyjne** udostępniające wolne cykle procesora z jednej partycji zadaniom z innej partycji.

Krótkie podsumowanie — pytania sprawdzające

Odpowiedz na poniższe pytania:

1. Porównaj własności strategii planowania FCFS i SJF (przewagi jednej i drugiej).

2. Czym różni się zachowanie zadań wsadowych od interakcyjnych?

3. Jakie kryteria optymalizuje algorytm planowania rotacyjnego (RR) i kosztem jakich parametrów się to odbywa?

4. Dlaczego planowanie priorytetowe typowo łączy się z inną strategią planowania?

5. W jakim celu stosuje są planowanie procesów z wielopoziomowymi kolejkami?

6. Które strategie planowania nadają się do planowania procesów obliczeniowych, które do interakcyjnych, a które można stosować dla mieszanej grupy procesów?

7. Które strategie planowania mają charakter niewywłaszczeniowy, które są typowo wywłaszczeniowe, a które można stosować zarówno z wywłaszczaniem jak i bez niego?

Planowanie statyczne i dynamiczne

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.

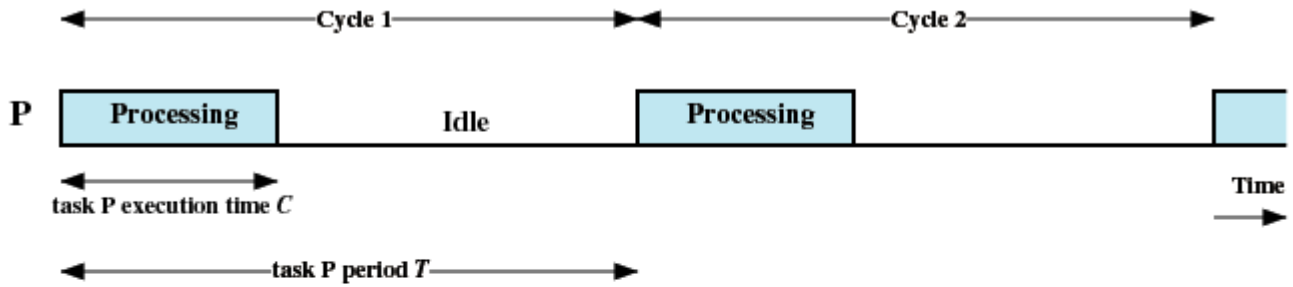
Celem **planowania przed wykonaniem**, albo inaczej: **planowania statycznego**, jest wyznaczenie odpowiedniej kolejności wykonywania zapewniającej spełnienie ograniczeń i bezkolizyjny dostęp do zasobów systemu. Planowanie przed wykonaniem może również minimalizować pewne narzuty systemowe, takie jak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

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

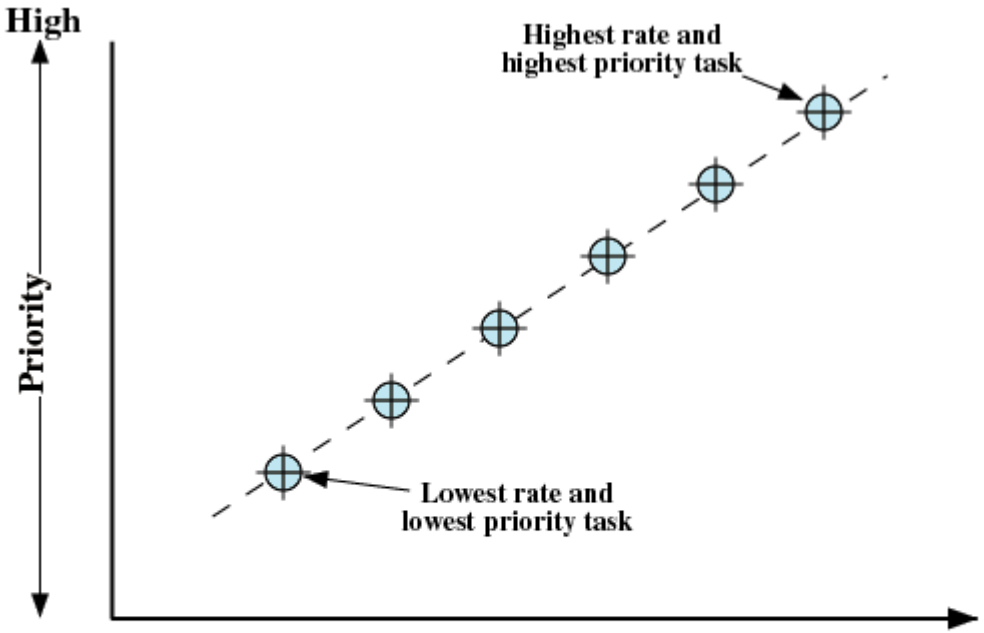


Możemy traktować instancje danego zadania jako oddzielne zadania, które podlegają szeregowaniu przez system.

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 RMS

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 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.

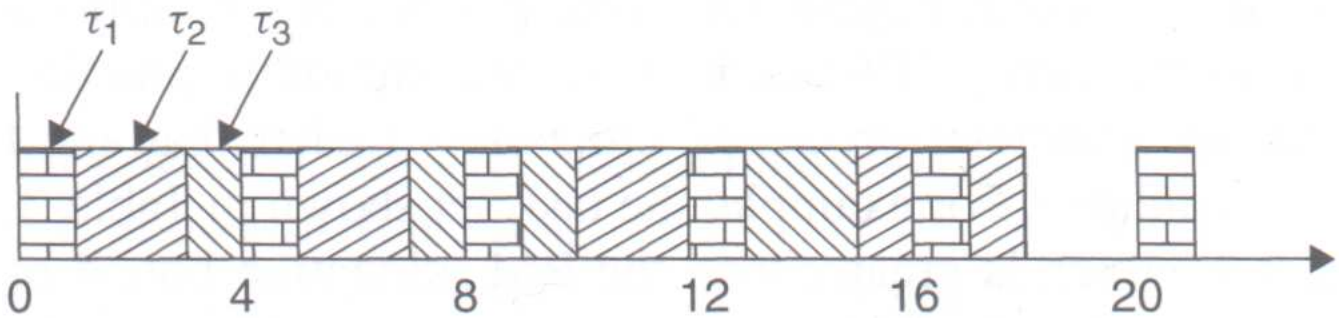
Przyczyną nieoptymalności strategii RMS jest sztywny przydział priorytetów. Można uzyskać lepsze wykorzystanie procesora przez bardziej elastyczne metody planowania.

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 EDF

Istotą przetwarzania w RTOS jest aby określone zadania wykonały się 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*).

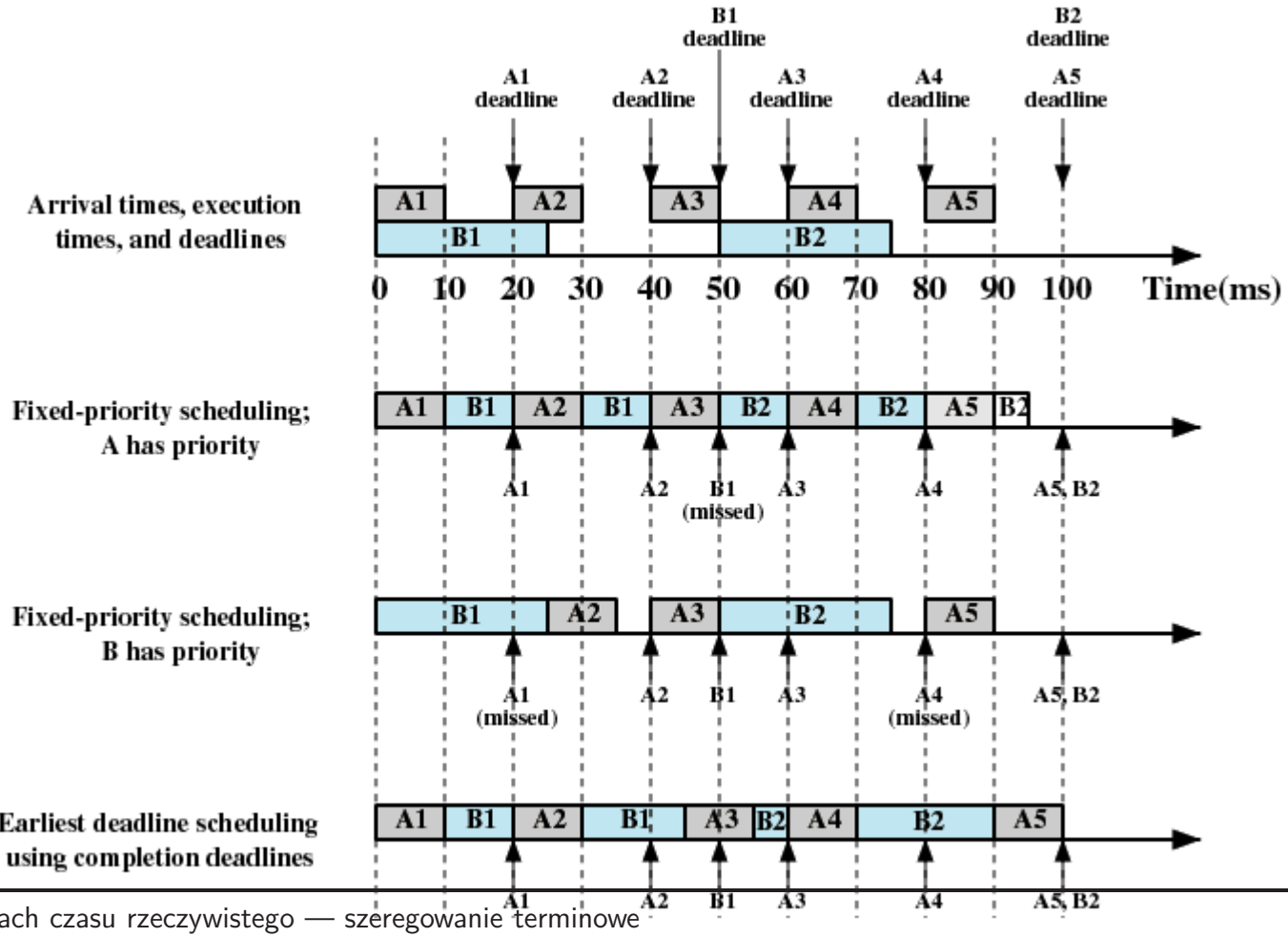
	czas uwolnienia	czas wykonania	termin zakończenia
τ_1	0	8	20
τ_2	3	2	5
τ_3	5	3	10

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 — własności

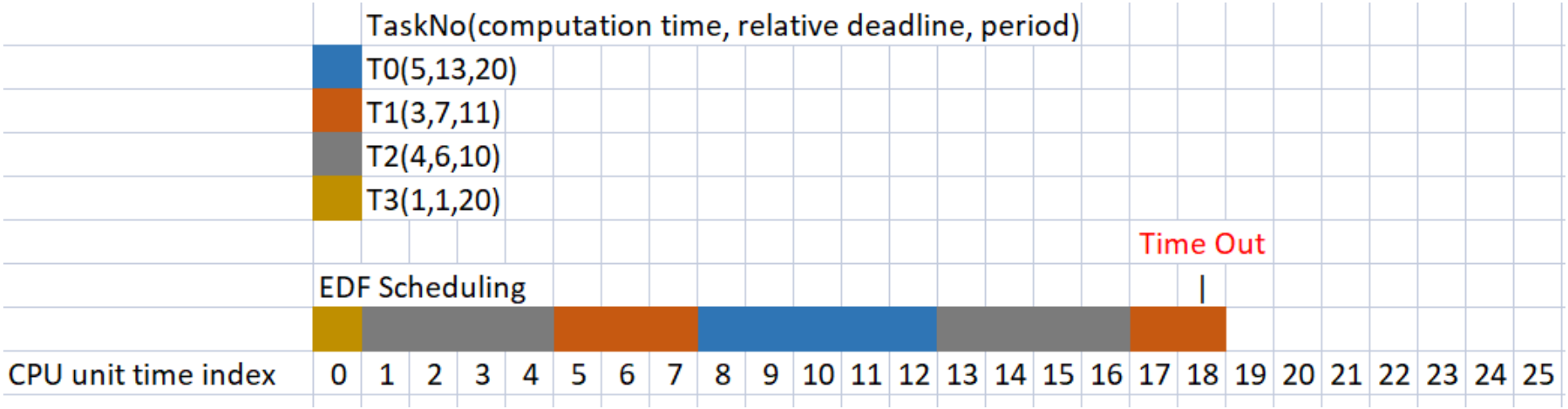
Algorytm szeregowania terminowego EDF dla pojedynczego procesora z wywłaszczaniem jest optymalny w takim sensie, że jeśli dany zbiór zadań, każde z określonym okresem, czasem uwolnienia, czasem obliczeń, i terminem zakończenia, jest szeregowalny jakimkolwiek algorytmem zapewniającym dotrzymanie czasów 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 konkretnego rozkładu terminów zadań, oraz momentu, w którym wystąpi przeciążenie). Stanowi to istotną wadę tego algorytmu, w odróżnieniu od algorytmu RMS.

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.

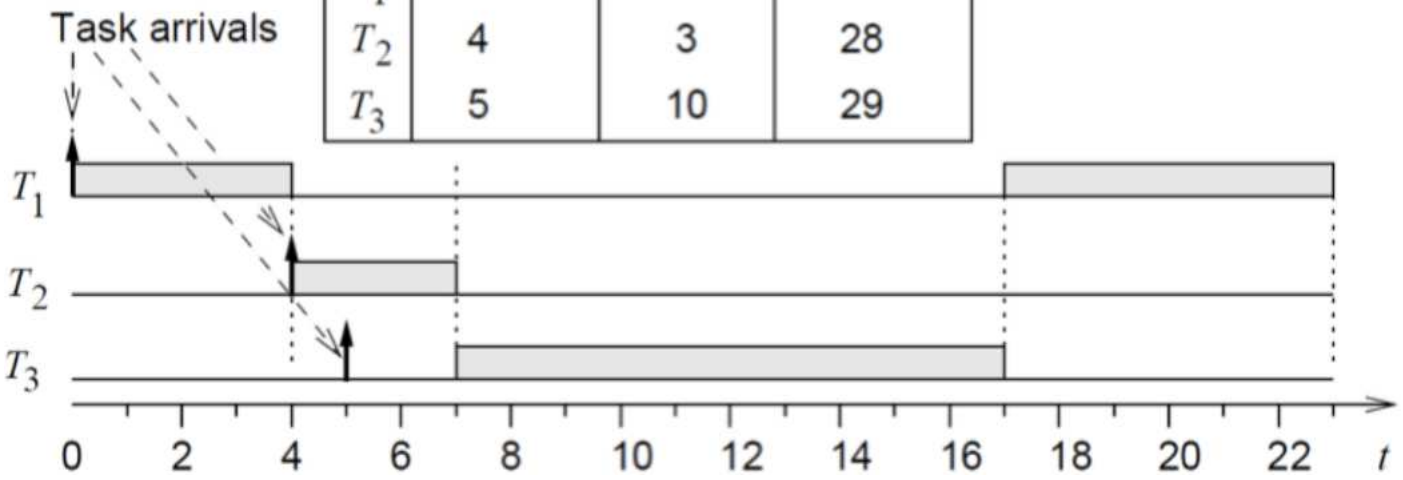


Szeregowanie terminowe — procesy nieokresowe

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 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

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 można powiedzieć, że jeśli istnieje poprawny harmonogram, to EDF będzie działał poprawnie. W przypadku szeregowania bez wywłaszczania EDF nie jest optymalny.

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: $C_1 = 25$, $T_1 = 50$, $C_2 = 30$, $T_2 = 75$. 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. Wynik przedstaw na diagramie czasowym.