Planowanie procesów

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 nie występują procesy, natomiast mogą być tworzone wątki 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 zauważalny na poziomie systemu.

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.

Blok kontrolny procesu PCB

Blok kontrolny procesu (*Process Control Block PCB*) zawiera główne informacje o stanie procesu. Stan procesu jest zapisywany przy przełączaniu kontekstu i odtwarzany na podstawie zawartości PCB przy wznawianiu obliczeń na procesorze. PCB zawiera również informacje o procesie wykorzystywane w algorytmach planowania. W szczególności:

- identyfikator procesu,
- adres startowy,
- kontekst: stan procesu, licznik instrukcji (PC), zawartość rejestrów,
- informacje związane z planowaniem,
- informacje związane z synchronizacją,
- informacje związane z zarządzaniem pamięcią,
- informacje dotyczące zużycia czasu,
- informacje o stanie wejścia-wyjścia,
- ...

Umieszczenie przez planistę procesu w kolejce procesów wykonywanych oznacza umieszczenie PCB tego procesu na liście innych PCB w tej kolejce. Usunięcie procesu z systemu polega na wykasowaniu jego PCB i dealokacji zajmowanej pamięci.

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 nanojądrem) i wykonywany jako oddzielny proces.

W literaturze systemów operacyjnych wyróżnia się zadania planisty krótkoterminowego, średnioterminowego, i długoterminowego. Właściwym planistą, o którym tu będzie mowa jest planista krótkoterminowy. Planiści średnioterminowy i długoterminowy biorą udział w utworzeniu zadania w kolejce zadań gotowych i podjęciu decyzji o przydziale mu pamięci. Te zagadnienia nie będą tu omawiane.

Okres planowania

Łączny czas pracy planisty i dyspozytora jest typowym narzutem administracyjnym, i ich działania muszą być podejmowane na tyle rzadko, by nie wpływały negatywnie 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.

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.

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.

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.

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. Na czym polega sprawiedliwość w planowaniu procesów?

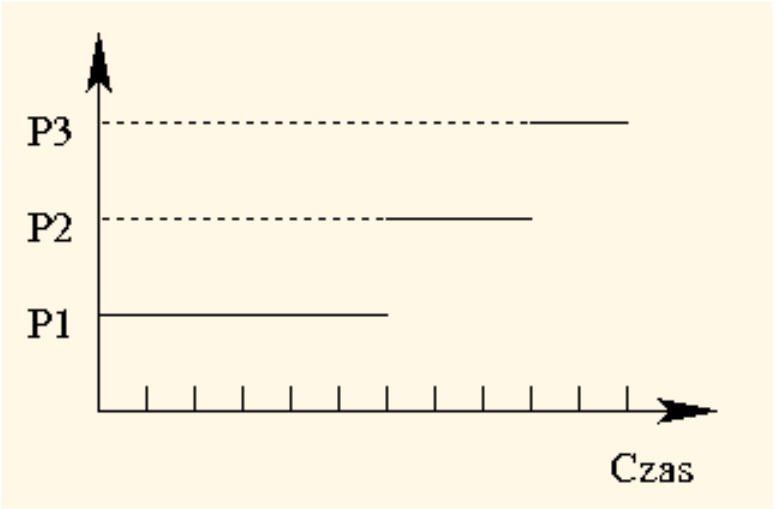
5. Jakie kryteria może optymalizować algorytm planowania?

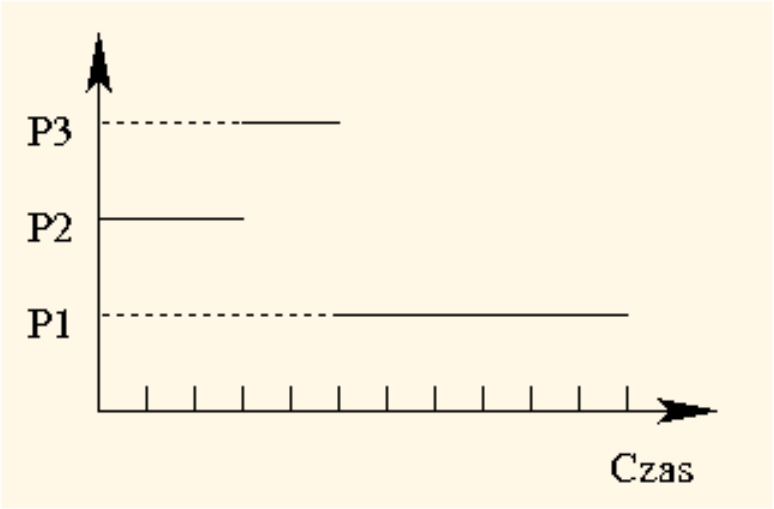
Strategia FCFS

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.

Dygresja — operacje I/O

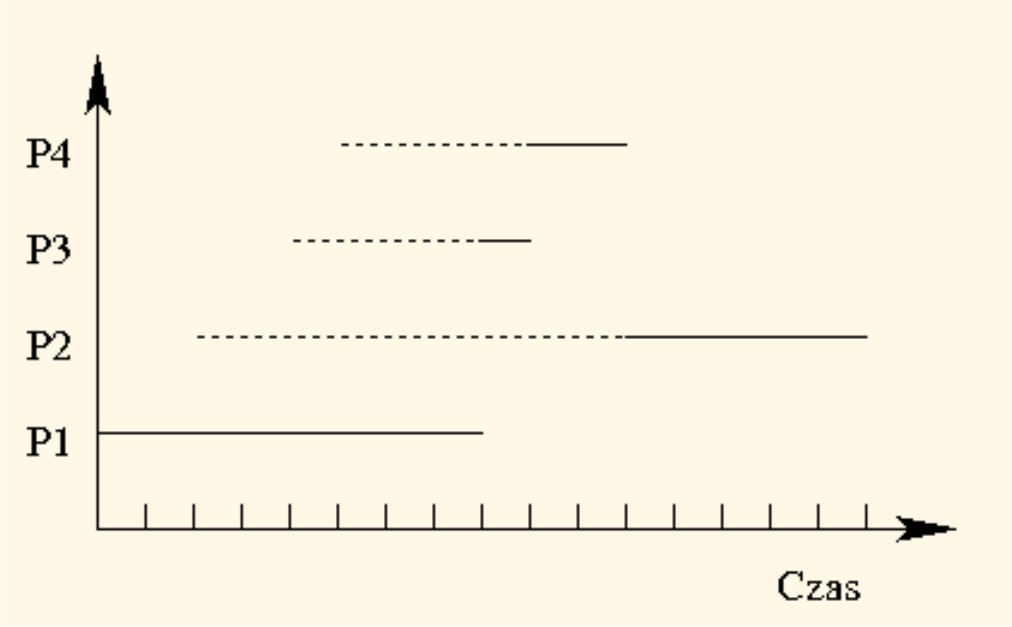
Algorytm FCFS w oryginalnej wersji z lat 70-tych XX wieku był realizowany ściśle według zasady: zadanie załadowane na procesor liczy aż do ostatecznego zakończenia programu. Jednak typowo zadania mają fazy obliczeniowe, kiedy wykorzystują procesor w blisko 100%, i fazy wejścia/wyjścia, kiedy wykonują szereg operacji I/O na różnych urządzeniach, takich jak dyski. Ponieważ operacje I/O są znacznie wolniejsze niż obliczenia w procesorze, powoduje to marnowanie mocy procesora, który w fazie I/O w większości czeka bezczynnie na ukończenie transferu.

Można wyobrazić sobie zmodyfikowaną wersję FCFS, która uruchamia zadanie na procesorze tylko do momentu, kiedy wskutek jakiegoś żądania musi ono czekać. Traktujemy więc czas życia procesu jako składający się z wielu kolejnych faz obliczeniowych, i w planowaniu bierzemy pod uwagę nie cały czas życia procesu, tylko jego poszczególne fazy. Przerwane zadanie będzie wznowione na procesorze gdy tylko zostanie on zwolniony, ale w międzyczasie inne zadanie/zadania mogą zostać na jakiś czas uruchomione.

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.

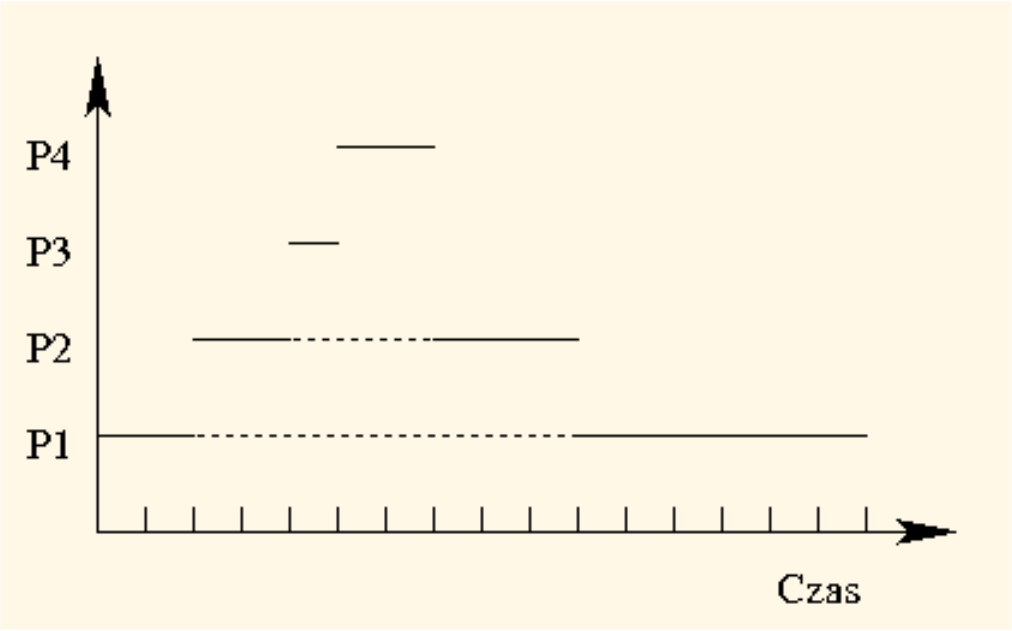
Proces	Czas uwoln.	Czas wykon.
P4	5	2
P3	4	1
P2	2	5
P1	0	8



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.

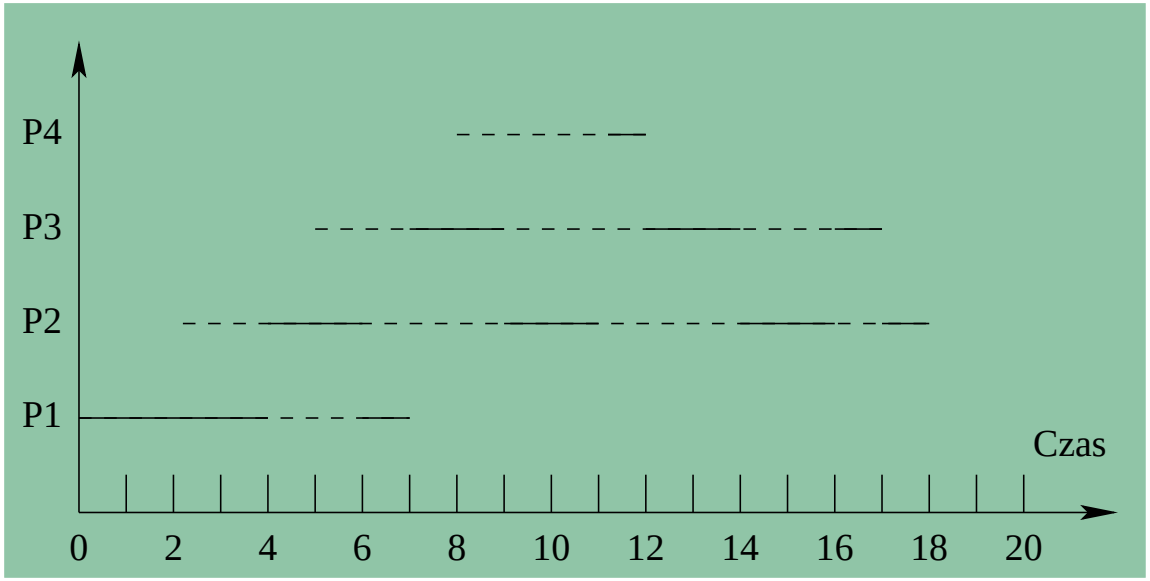
Zadania interakcyjne mogą być związane z pracą człowieka na komputerze typu PC, ale mogą to być również serwery odpowiadające na żądania napływające z sieci, albo przetwarzające dane z pomiarów, itp. Odmienny profil tych zadań powoduje, że przy ich szeregowaniu znaczenie mają różne kryteria, i lepiej 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, jak również wymusić, by dany proces był wykonywany przed innymi. 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 spełnienia określonych wymagań jest proste tylko gdy system jest mały, tzn. wykonuje kilka lub kilkanaście 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 tła mogą być obsługiwane strategią SJF, ponieważ w przypadku pracy wsadowej ważna jest minimalizacja średniego czasu oczekiwania. Natomiast procesy czoła mogą być obsługiwane strategią RR, ponieważ w przypadku pracy interakcyjnej istotna jest minimalizacji czasu reakcji.

Rozróżnienie rodzajów procesów może być przez deklarację użytkownika, albo przez rozpoznanie charakterystyki procesów przez system, jak w przypadku dynamicznych priorytetów.

Pozostaje jeszcze rozstrzygnąć wzajemny stosunek obu tych grup procesów. Oczywiście procesy czoła powinny być uprzywilejowane, ale w jakim stopniu? Można przyjąć, że zawsze wykonujemy procesy czoła przed wsadowymi w oczekiwaniu, że zawsze pojawiają się momenty bezczynności użytkowników. Jeśli jednak chcemy zapobiec możliwemu zagłodzeniu procesów tła, można przyjąć stały podział czasu między kolejki procesów tła i czoła, np. 80% dla czoła i 20% dla tła.

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?

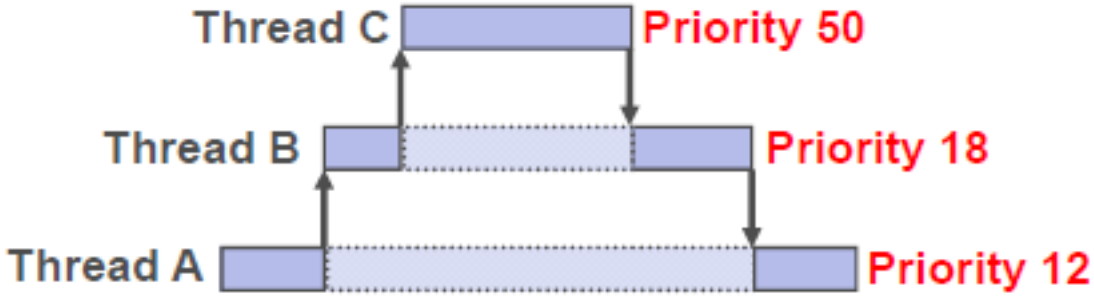
3. Jakie kryteria optymalizuje algorytm planowania rotacyjnego (RR) i kosztem jakich parametrów się to odbywa?

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

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

Planowanie w systemach czasu rzeczywistego

Opisane powyżej planowanie priorytetowe można stosować w systemach czasu rzeczywistego, ponieważ daje ono niezbędną w tych systemach przewidywalność. Przypisanie priorytetów zadaniom gwarantuje, że zadania o wyższych priorytetach będą wykonywane przed tymi o priorytetach niższych.



W prostych przypadkach można dla wszystkich zadań obliczyć czas ich wykonywania uwzględniając możliwe interakcje z zadaniami o wyższych priorytetach.

Jednak w sytuacji gdy zadania charakteryzują się zróżnicowanymi czasami i częstotliwością wykonania, a rywalizują w dostępie do zasobów systemu, wyznaczenie priorytetów zapewniające poprawne wykonanie wszystkich zadań często nie jest łatwe.

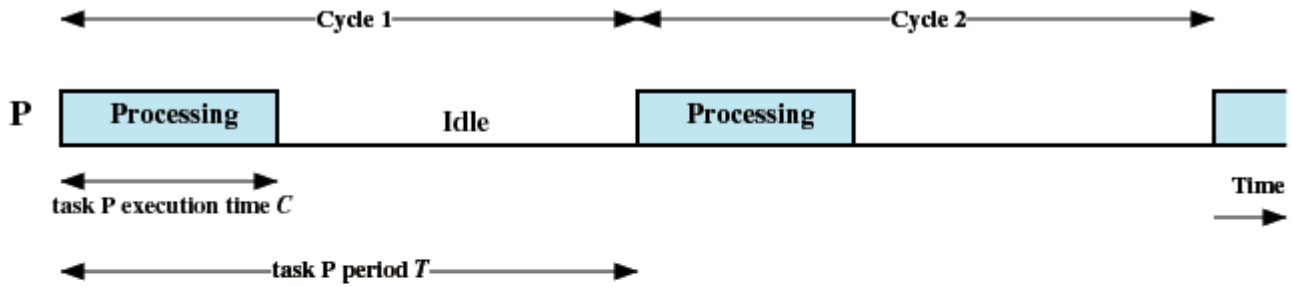
Strategie szeregowania RTS – problemy z priorytetami

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.

Strategie szeregowania RTS – zadania okresowe

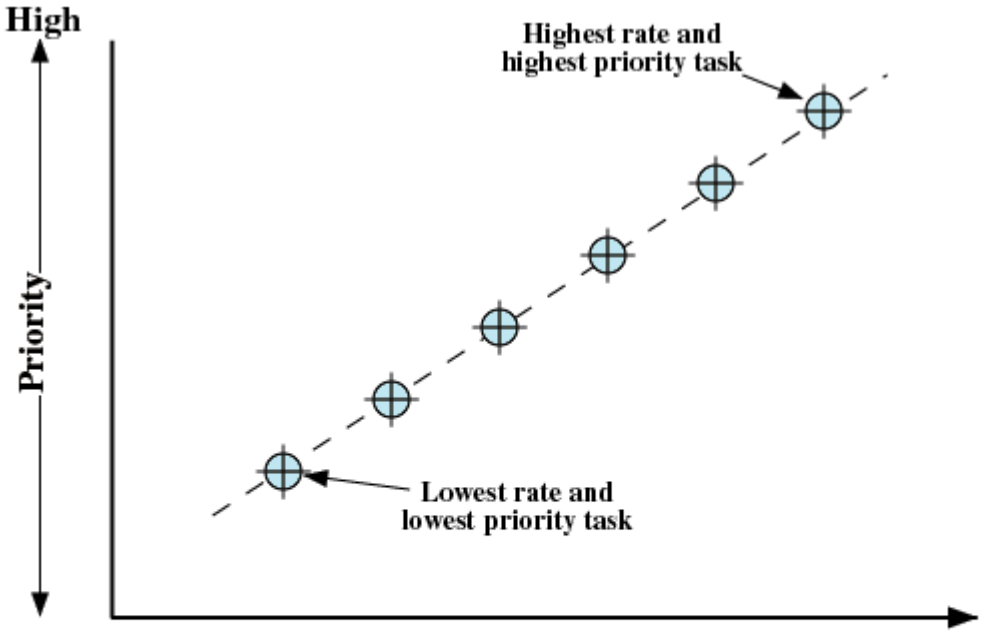
W systemach czasu rzeczywistego często mamy do czynienia z zadaniami okresowymi. Zadania takie muszą być wykonywane cyklicznie w nieskończonej pętli powtórzeń. Zadaniem systemu jest albo uruchamianie takich zadań z określonym okresem, albo — bardziej typowo — okresowe wznawianie takich zadań, które nigdy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

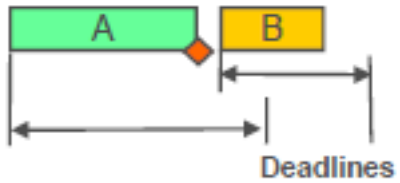
$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

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

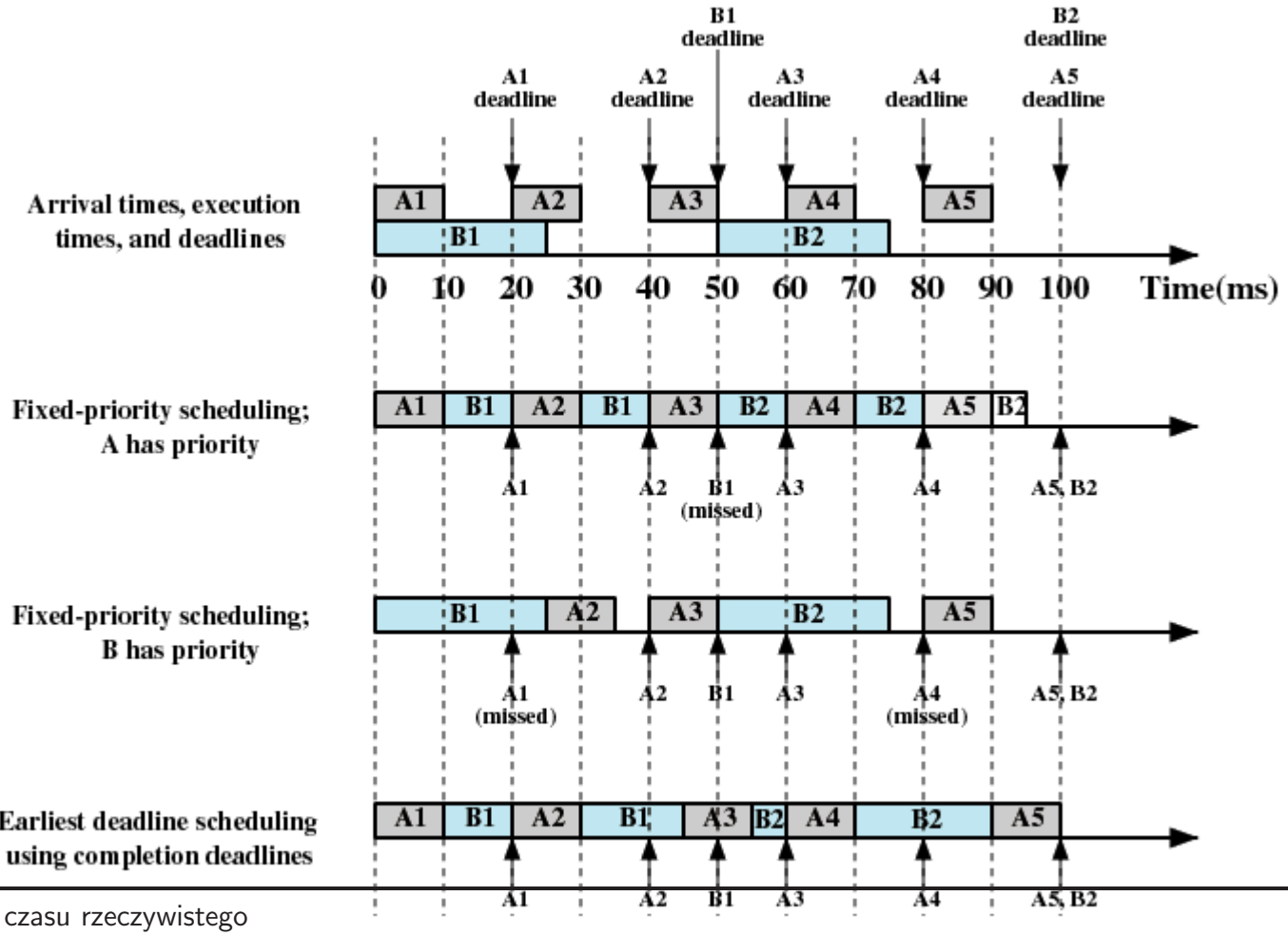
Istotą przetwarzania w RTOS jest aby określone zdarzenia występowały w określonym czasie. Jeśli zdefiniujemy pożądany czas rozpoczęcia i/lub zakończenia wszystkich zadań, to system może obliczyć właściwe szeregowanie zapewniające spełnienie wszystkich ograniczeń. Taka metoda jest nazywana **szeregowaniem 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.

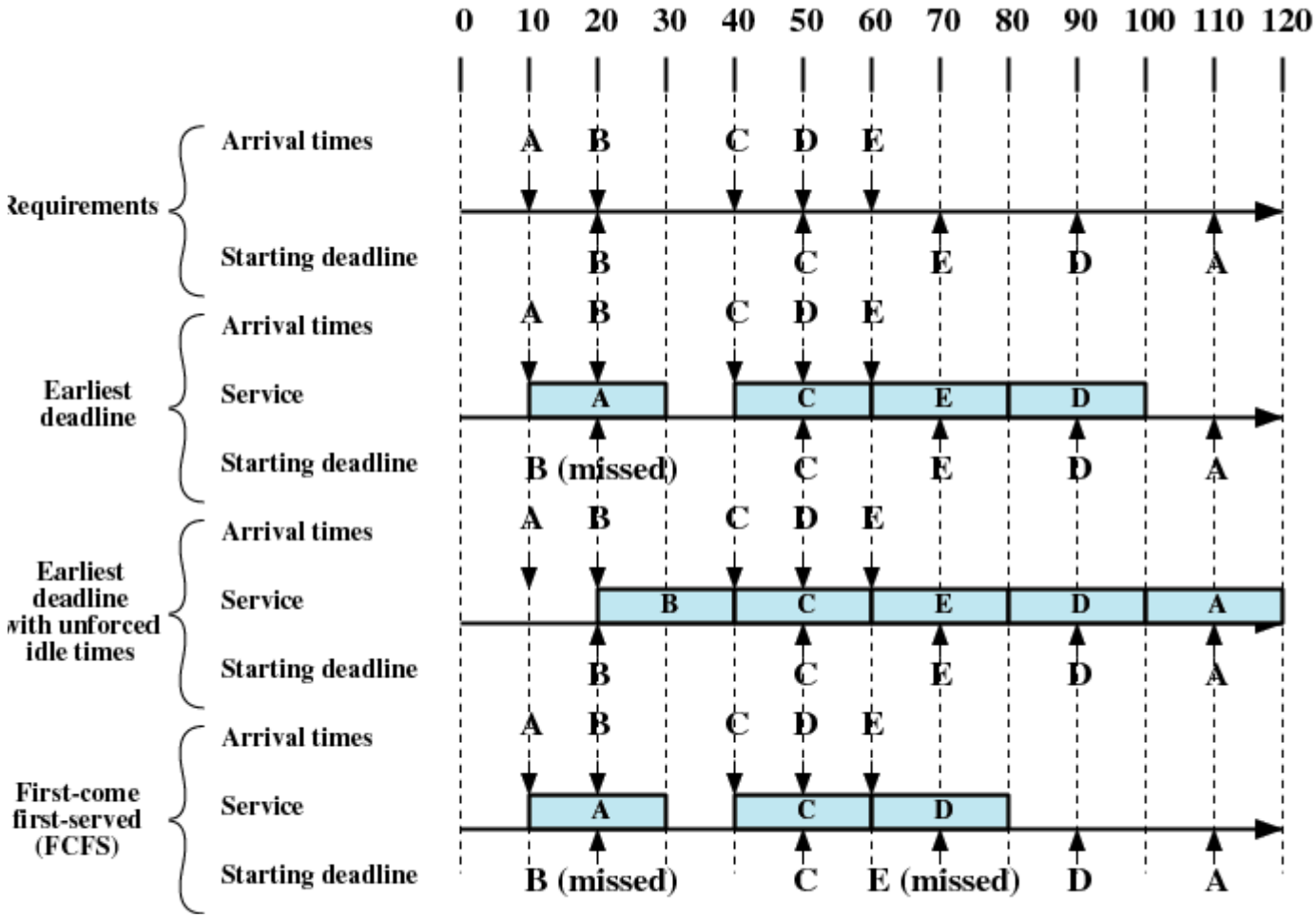
Szeregowanie terminowe dla zadań okresowych

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

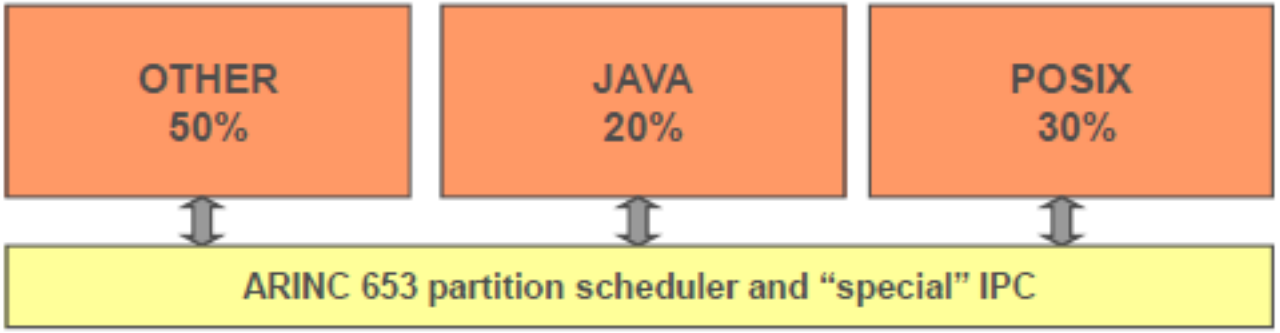
Algorytm szeregowania terminowego EDF dla pojedynczego procesora z wywłaszczaniem jest optymalny w takim sensie, że jeśli dany zbiór zadań, każde z określonym okresem, czasem uwolnienia, czasem obliczeń, i terminem zakończenia, jest szeregowalny jakimkolwiek algorytmem zapewniającym dotrzymanie czasów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



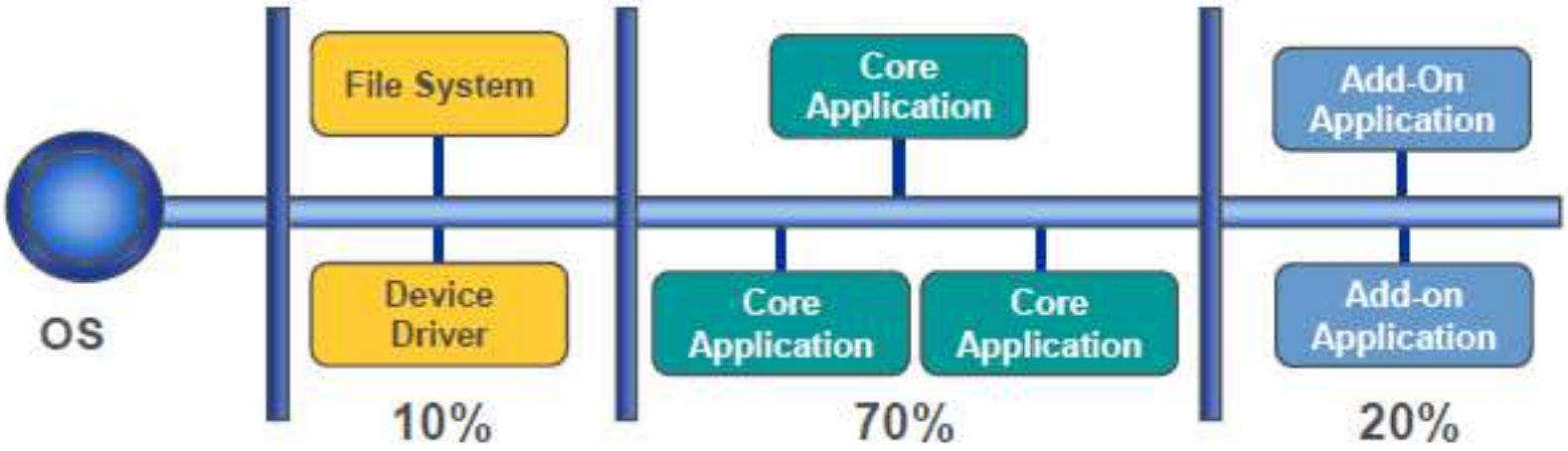
W tej architekturze system zapewnia przydział procesora, ochronę pamięci, i komunikację międzyprocesową pomiędzy partycjami, natomiast w ramach danej partycji funkcjonuje szeregowanie priorytetowe.

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.

Inne przesłanki w zagadnieniach planowania

Warto zwrócić uwagę na fakt, że mogą wystąpić okoliczności, które nie są znane planiście, ale mogą zasadniczo wpłynąć na efektywność pracy systemu.

Jeśli istnieje grupa procesów zorientowanych na obliczenia (*compute-bound*), i grupa procesów zorientowanych na operacje wejścia-wyjścia (*I/O-bound*), i planista zaplanuje najpierw wykonywanie tych pierwszych, to będą one rywalizowały o dostęp do procesora, a w tym czasu dysk, i inne urządzenia I/O, będą bezczynne. Gdy następnie uruchomiona zostanie druga grupa procesów, to będą one stale blokowały się na operacjach I/O i zwalniały procesor, który będzie mało obciążony, podczas gdy podsystem dyskowy będzie pracował na 100%.

Przewidując to, planista (długoterminowy) mógłby uruchomić najpierw tylko część procesów pierwszej i drugiej grupy pozwalając aby procesy obliczeniowe wykonywały się w chwilach zwolnienia procesora przez procesy I/O.

Jednak rzadko kiedy planista ma dostęp do takich informacji. Natomiast w systemach, gdzie wykonują się dobrze zaprojektowane aplikacje, programista i/lub administrator mogą mieć możliwość ingerencji w algorytm planowania, w celu poprawy wydajności pracy systemu (patrz mechanizm i funkcja systemowa *nice*).

Optymalizacja w planowaniu

Algorytmy planowania optymalizują sposób wykonywania procesów przez system:

- z punktu widzenia systemu, co oznacza minimalizację strat, czyli przełączeń kontekstu; skrajnym przypadkiem są strategie FCFS i SJF, które nie wykonują żadnych przełączeń kontekstu niepotrzebnych z punktu widzenia systemu; wszystkie strategie z wywłaszczaniem idą na kompromis, akceptując pewną liczbę „niepotrzebnych" przełączeń kontekstu,

- z punktu widzenia użytkownika, co oznacza wybór właściwych procesów do wykonywania, gdzie przez „właściwe" rozumie się:
  - procesy, które wiadomo, że powinny być wykonywane pierwsze, zgodnie z ideą danego algorytmu, np. procesy o najwyższym priorytecie w planowaniu priorytetowym, albo procesy interakcyjne w planowaniu z wielopoziomowymi kolejkami,
  - równomierne wykonywanie wszystkich „pozostałych" procesów, tzn. takich, co do których system nie ma żadnych preferencji wykonywania.

„Sprawiedliwość" w planowaniu

Rozważmy jak zasadę „sprawiedliwości", czyli egalitaryzmu, stosują poznane algorytmy planowania.

- Algorytm FCFS realizuje pewien specyficzny rodzaj sprawiedliwości, mianowicie prawo pierwszeństwa. Nie jest ono jednak zgodne z zasadą egalitaryzmu i nie uznajemy go za sprawiedliwość.

- Algorytmy SJF i SRTF faworyzują zadania krótkie, mogą doprowadzić do zagłodzenia zadań długich, i nie są sprawiedliwe.

- Algorytm RR jest czystą realizacją zasady sprawiedliwości, kosztem maksymalnych strat na przełączanie kontekstu.

- Strategię opartą na priorytetach możnaby uznać za częściowo sprawiedliwą, jeśli procesy o równych priorytetach byłyby szeregowane algorytmem sprawiedliwym, takim jak RR. Byłaby to jednak sprawiedliwość „kastowa" dzieląca obywateli na „równych" i „równiejszych."
  Szeregowanie priorytetowe uznajemy więc za niesprawiedliwe.

Podsumowanie i inne rodzaje planowania

Istnieje grupa strategii planowania szczególnie nadających się do (optymalizujących) wykonywania zadań obliczeniowych: FCFS, SJF, SRTF.

Inna grupa strategii jest bardziej odpowiednia do zadań interakcyjnych: RR i strategie mieszane (np. w połączeniu z priorytetami, albo wielopoziomowe).

Omawiane tu strategie powstały w odniesieniu do systemów jednoprocesorowych (na co zresztą wskazuje polskie określenie: szeregowanie). Mogą być one również stosowane do systemów wieloprocesorowych i wielordzeniowych, jakkolwiek istnieją nieomówione tu wyspecjalizowane wersje algorytmów planowania dla takich systemów, biorące pod uwagę ich specyfikę.

Odmienne wymagania prezentują również systemy czasu rzeczywistego. Specjalizowane systemy operacyjne czasu rzeczywistego (*RTOS — Real Time Operating System*) mają ogólnie swoją specyfikę, odróżniającą je od systemów GPOS. Na przykład, obce są im obowiązujące w GPOS zasady sprawiedliwości. W systemie RTOS dopuszczalne jest, by jeden proces wysokiego priorytetu zdominował cały system, włącznie z wywłaszczeniem jądra systemu.

Krótkie podsumowanie — pytania sprawdzające

Odpowiedz na poniższe pytania:

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

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

Ciekawe materiały

Wykład wideo z UC Berkeley o planowaniu procesów – prawie 1.5 godziny
`http://www.youtube.com/watch?v=aODdUp_eHQs`