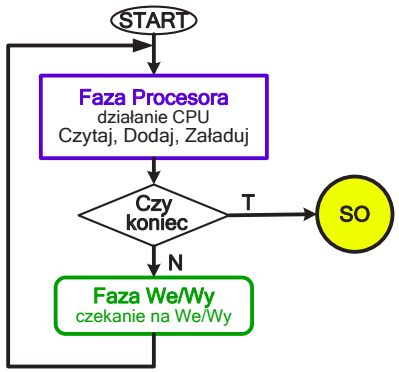
5. PLANOWANIE PRZYDZIAŁU PROCESORA

W PAO znajduje się kilka procesów jednocześnie.

Cel planowania: stale utrzymywać w działaniu określoną liczbę procesów.

Proces wykonywany jest do chwili, gdy musi CZEKAĆ (np. zakończenie operacji We/Wy).

Czekającemu procesowi SO odbiera CPU i oddaje innemu.



Rys. 5.1. Fazy procesora

Wykonanie **procesu** to **cykle** (fazy): -**działanie CPU**,

-**oczekiwanie** na **We/Wy**.

W ostatniej **fazie CPU** proces wysyła do SO żądanie zakończenia swojego działania.

Stwierdzono, że: -istnieje **dużo krótkich** faz CPU,

-mało **długich** faz CPU.

Planista krótkoterminowy wybiera z kolejki procesów Gotowych w PAO, jeden proces i przydziela mu CPU.

Kolejka procesów gotowych może być kolejką FIFO, kolejką priorytetową, drzewem, listą.

Elementami kolejek są Bloki Kontrolne Procesów.

Zmiana przydziału CPU może nastąpić gdy **proces:**

1. Przeszedł od stanu **AKTYWNOŚCI** do stanu **CZEKANIA**
   -zamówienie na We/Wy,
   -rozpoczęcie czekania na zakończenie działania procesu potomnego.
2. Kończy działanie.
3. Przeszedł od stanu **AKTYWNOŚCI** do **GOTOWOŚCI** (np. wskutek wystąpienia przerwania).
4. Przeszedł od stanu **CZEKANIA** do **GOTOWOŚCI** (np. po zakończeniu operacji We/Wy).

Sytuacja **1** i **2** nie daje wyboru.
CPU otrzyma nowy proces z kolejki procesów gotowych.

Planowanie niewywłaszczeniowe (*nonpreemptive*): proces otrzymując CPU, zachowuje go do czasu swojego **ZAKOŃCZENIA** lub przejścia do stanu **CZEKANIA**.

Planowanie wywłaszczeniowego *(preemptive)* odpowiada sytuacjom innym niż 1 i 2.

- **Dwa procesy korzystają ze wspólnych danych.**

  Jeden proces zostaje wywłaszczony w trakcie aktualizowania przez siebie danych, i następuje uaktywnienie drugiego procesu.

  Drugi proces usiłuje czytać dane, których stan jest niespójny.

  \+ Potrzebne są mechanizmy koordynacji dostępu do danych dzielonych.

- **Wykonując funkcję systemową jądro zajęte jest działaniami** na rzecz **procesu.**

  Czynności te mogą dokonywać zmian ważnych danych jądra.

  Co się stanie, jeśli **proces *zostanie wywłaszczony*** w trakcie tych zmian i moduł sterujący urządzenia ma przeczytać lub zmienić tę samą strukturę?

  DOJDZIE DO CHAOSU

  Można czekać z przełączeniem kontekstu do zakończenia wywołania systemowego lub do zablokowania procesu na We/Wy.

  \+ Nie wywłaszcza się procesu w chwilach niespójności struktur danych jądra.

  Taki model jądra nie nadaje się do pracy w czasie rzeczywistym lub wieloprzetwarzania.

- **Przerwania mogą się pojawiać** w **dowolnych chwilach** i **SO musi je przyjmować.**

  Fragmenty **kodu dotyczące przerwań** należy zabezpieczyć przed jednoczesnym użyciem.

  Aby uniemożliwić współbieżny dostęp kilku procesów do **sekcji kodu przerwań**, *wyłącza się **przerwania*** na WEjściu do nich i ponowne włącza przy WYchodzeniu z tych sekcji.

Ekspedytor *(dispatcher)*: odrębny moduł w planowaniu przydziału procesora, który przekazuje CPU do dyspozycji wybranego **procesu** przez planistę krótkoterminowego.

Obowiązki Ekspedytora: -przełączanie kontekstu;
-przełączanie do trybu użytkownika;
-wykonanie skoku do komórki w programie użytkownika.
(wznowienie programu)

Ekspedytor jest wywoływany podczas każdego przełączania procesu.

Opóźnienie ekspedycji *(dispatch latency): c*zas, który ekspedytor zużywa aby wstrzymać jeden proces i uaktywnić inny.

❒ Kryteria planowania

Wykorzystanie CPU w realnym systemie powinno mieścić się w przedziale (40 ÷ 90)%.

Parametry przydatne przy wyborze algorytmów planowania:

- **Przepustowość** *(throughput)*: liczba procesów kończonych pracę w jednostce czasu.
  Dla długich procesów jeden proces na godzinę, dla krótkich 10 procesów na sekundę.

- **Czas cyklu przetwarzania** *(turnaround time)*: czas upływający między chwilą nadejścia procesu do systemu a chwilą zakończenia procesu.
  Suma okresów spędzonych na:
  -czekaniu na wejście do pamięci,
  -czekaniu w kolejce procesów Gotowych,
  -wykonywaniu procesu przez CPU,
  -wykonywaniu operacji We/Wy.

- **Czas oczekiwania:** sumą czasów, spędzonych przez proces w kolejce procesów **Gotowych**.
  Algorytm planowania przydziału procesora nie ma wpływu na czas, w którym proces działa lub wykonuje operacje We/Wy.

- **Czas odpowiedzi** *(response time)*: czas upływający między przedłożeniem zamówienia a pojawieniem się odpowiedzi w systemach **interakcyjnych**.
  Nie obejmuje czasu wyprowadzenia tej odpowiedzi.
  Jest uzależniony od szybkości urządzeń We/Wy.

Wykorzystanie procesora i przepustowość powinny być maksymalne.

Czas cyklu przetwarzania, oczekiwania i odpowiedzi - minimalne.

Zazwyczaj optymalizuje się miarę średnią.

Czasami optymalizacja wartości **minimalnych** lub **maksymalnych** może być bardziej pożądana.
Zapewnienie dobrej obsługi wszystkim użytkownikom sugeruje zmniejszanie maksymalnego **czasu odpowiedzi**.

W systemach interakcyjnych (z podziałem czasu) ważniejsze jest minimalizowanie **wariancji** czasu odpowiedzi aniżeli minimalizowanie średniego czasu odpowiedzi.

System z **określonym** i przewidywalnym czasem odpowiedzi może być bardziej pożądany niż system, który ma przeciętnie szybszy, ale ***zmienny*** czas reakcji.

5.1. Planowanie jednoprocesorowe

Planowanie przydziału CPU to **podejmowanie decyzji, którym procesom z kolejki procesów Gotowych do działania należy przydzielić CPU.**

Uproszczenia do analizy:
-rozważana jest jedna **faza** procesora (ms) przypadającą na każdy proces.
-miarą stosowaną w porównaniach jest **średni czas oczekiwania** na CPU.

5.1.1. Metoda FCFS

„Pierwszy zgłoszony - pierwszy obsłużony" (*First-Come, First-Served* - FCFS)

Proces, który pierwszy zamówi CPU, pierwszy otrzyma go.

Implementacja algorytmu może być zrealizowana za pomocą kolejki FIFO.
Blok kontrolny procesu jest dołączany na **koniec** kolejki.
Proces przydziela się CPU z **czoła** kolejki.

W metodzie FCFS średni czas oczekiwania może być bardzo długi.

● **Rozważmy zbiór trzech procesów nachodzących w chwili t = 0.**

Procesy nadchodzą w kolejności: **P1, P2, P2**.

Proces	Czas trwania fazy [ms]
P1	6
P2	3
P3	2

+ Średni czas oczekiwania: (0 + 6 + 9)/3 = **5 ms**.

Dla **P1** czas oczekiwania **0** ms,
Dla **P2** czas oczekiwania **6** ms,
Dla **P3** czas oczekiwania **9** ms.

Procesy nadchodzą w kolejności: **P3, P2, P1**

Proces	Czas trwania fazy [ms]
P3	2
P2	3
P1	6

+ Średni czas oczekiwania: (0 + 2 + 5)/3 = **2.33** ms.

Średni czas oczekiwania w FCFS zależy od kolejności nadejścia Procesów.

❒ **Efekt konwoju:** procesy czekają na **zwolnienie CPU** przez **jeden WIELKI Proces**

Mamy jeden proces **P_{CPU}** o **długiej** fazie CPU

i wiele **$P_{We/Wy}$** o krótkich fazach CPU.

❶ Proces **P_{CPU}** uzyskał przydział CPU. |_____| **Oczekiwanie We/Wy**

W tym czasie procesy **$P_{We/Wy}$** skończyły swoje operacje We/Wy i przemieściły się do kolejki procesów **Gotowych**. |___| |___| |__|

➔**Urządzenia We/Wy są bezczynne.**

❷ Po pewnym czasie **P_{CPU}** skończy fazę CPU i przechodzi do kolejki **Oczekującej** na We/Wy.

Procesy **$P_{We/Wy}$** działają **szybko** (krótkie fazy CPU) i powrócą do kolejek urządzeń We/Wy.

Od tego momentu CPU jest bezczynny,

Wszystkie procesy czekają na We/Wy

❸ Proces **P_{CPU}** wraca do kolejki procesów **Gotowych** i otrzymuje przydział CPU.

Znowu procesy **$P_{We/Wy}$** po zakończeniu operacji We/Wy przejdą do kolejki procesów **Gotowych** i będą tam oczekiwać, aż **P_{CPU}** wykona swoje obliczenia.

Wystąpił **efekt konwoju:** mniejsze wykorzystanie zasobów, niż gdyby najpierw pozwolono pracować krótszym procesom.

Algorytm FCFS jes t **niewywłaszczający**.

Proces utrzymuje kontrolę nad CPU do czasu zwolnienia go przez siebie (zakończenie działania) lub zamówienia operacji We/Wy.

Algorytm FCFS jest kłopotliwy w systemach z podziałem czasu, w których każdy użytkownik powinien dostawać przydział CPU w regularnych odstępach.

5.1.2. Metoda „najpierw najkrótsze zadanie" (*Shortest-Job-First* - SJF)

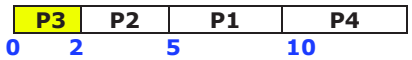
Proces mający najkrótszą następną fazę CPU otrzyma procesor pierwszy.

Jeśli dwa procesy mają następne fazy CPU równej długości, to stosuje się algorytm FCFS.

Właściwsza nazwa algorytmu to: „najkrótsza następna faza procesora", gdyż planowanie polega na sprawdzaniu w procesie długości jego następnej fazy CPU, a nie całej długości procesu.

Rozważmy zbiór czterech ***istniejących*** procesów w chwili t_0.,

Proces	Czas trwania fazy [ms]
P1	5
P2	3
P3	**2**
P4	6



Czas oczekiwania:
dla **P2** wynosi **2** ms,
dla **P4** wynosi **10** ms.

Średni czas oczekiwania: (0 + 2 + 5 + 10)/4 = **4.25** ms.

W FCFS średni czas oczekiwania: (0+5+8+10)/4 = **5.75** ms.

Algorytm SJF może być wywłaszczający lub **niewywłaszczający.**

Konieczność wyboru: w kolejce procesów GOTOWYCH pojawia się nowy proces (faza **12** ms), a poprzedni proces używa jeszcze procesora (przez **20** ms).

Nowy proces może mieć **krótszą następną** fazę procesora niż to, co pozostało do wykonania w procesie **bieżącym**.

☛ Wywłaszczający algorytm SJF usunie w tej sytuacji dotychczasowy proces z CPU.

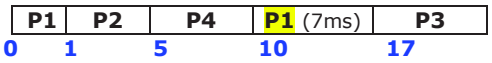
☛ **Niewywłaszczający** algorytm SJF pozwoli bieżącemu procesowi zakończyć fazę CPU.

Wywłaszczający algorytm **SJF** przydziału CPU nazywa się czasami planowaniem metodą **„najpierw najkrótszy pozostały czas"** *(Shortest-Remaining-Time-First).*

Rozważmy cztery procesy, przybywające w **kolejnych chwilach** do kolejki procesów gotowych.

Proces	Czas przybycia	Czas fazy
P1	**0**	8
P2	**1**	4
P3	**2**	9
P4	**3**	5

Wywłaszczające planowanie **SJF**



P1 w chwili **t** = **0** jest jedynym procesem w kolejce.

P2 nadchodzi dopiero w chwili **t** = **1**

Czas pozostały procesowi **P1** (**7 ms**) jest większy od czasu wymaganego przez proces **P2** (4 ms).

Proces **P1** zostaje wywłaszczony, a proces **P2** przejmuje CPU.

Średni czas oczekiwania z wywłaszczeniem:

((**10** - 1) + (1 - **1**) + (17 - **2**) + (5 - **3**))/4 = **6.5** ms.

Średni czas oczekiwania dla **nie**wywłaszczającego: **7.75** ms.

Algorytm SJF jest optymalny, gdyż minimalizuje średni czas oczekiwania dla zbioru procesów.

> Umieszczenie **krótkiego** procesu przed ***długim*** w większym stopniu zmniejsza czas oczekiwania krótkiego procesu, aniżeli wydłuża czas oczekiwania długiego procesu.

☛ Trudnością w **SJF** jest określenie długości następnego zamówienia na przydział CPU.

W długoterminowym planowaniu zadań za ową długość można przyjąć ***limit czasu procesu***, określany przez użytkownika zlecającego zadanie.

Algorytm SJF jest często używany w planowaniu długoterminowym.

Algorytm SJF kłopotliwy dla krótkoterminowego planowania przydziału CPU.

➔ **Nie ma sposobu na poznanie długości następnej fazy CPU.**

☛ Można spróbować **oszacować** długości następnej fazy CPU.

Zakłada się, że długość następnej fazy CPU będzie podobna do długości faz poprzednich.

Następną fazę CPU wyznacza się z pomiarów długości faz poprzednich.

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n \qquad \text{dla } \alpha \in (O < a < 1)$$

gdzie t_n -długość n-tej fazy CPU,

τ_{n+1} -przewidywana długość następnej fazy CPU.

Wartość początkową τ_0 można przyjąć jako stałą lub średnią wziętą z całego systemu.

Parametr α jest wagą między niedawną a wcześniejszą historią.

Zazwyczaj $\alpha = ½$.

Jeśli $\alpha = 0$, to $\tau_{N+1} = \tau_N$: zakłada się, że niedawne warunki były czymś przejściowym.

Jeśli $\alpha = 1$, to $\tau_{N+1} = t_n$: uwzględnia się tylko najnowsze notowanie długości fazy procesora.

5.1.3. Planowanie priorytetowe

CPU przydziela się procesowi o najwyższym priorytecie.

Priorytet procesu **dołączanego** do kolejki procesów gotowych porównywany jest z priorytetem ***bieżąco*** wykonywanego procesu.

☛ Procesom przydziela się priorytet.

Procesy o równych priorytetach planuje się metodą FCFS.

Algorytm SJF (najpierw najkrótsze zadanie) jest szczególnym przypadkiem planowania priorytetowego.

Im większa jest faza CPU, tym niższy może być priorytet.

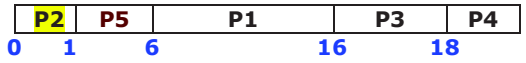
Priorytety należą do pewnego, ustalonego przedziału liczb całkowitych.

Często przyjmuje się, że im **mniejsza liczba** tym **wyższy** priorytet.

Rozważmy zbiór procesów przybyłych w czasie t=0 i w porządku **P1, P2, P3, P4, P5**.

Proces	Priorytet	Czas trwania fazy
P1	3	10
P2	**1**	**1**
P3	*3*	*2*
P4	4	1
P5	**2**	5

Planowanie priorytetowe bez wywłaszczenia



Średni czas oczekiwania: **8.2 ms**.

Priorytety mogą być definiowane **wewnętrznie** lub **zewnętrznie**.

Priorytety wewnętrzne używają mierzalnej właściwości procesu do obliczania priorytet.

Może to być: -limity czasu,
-wielkość obszaru wymaganej pamięci,
-liczba otwartych plików,
-stosunek średniej fazy We/Wy do średniej fazy CPU.

Priorytety zewnętrzne są określane na podstawie kryteriów zewnętrznych wobec SO:

-ważność procesu,
-kwota opłat za użytkowanie komputera,
-instytucja,
-inne czynniki, często o znaczeniu politycznym.

Planowanie priorytetowe może być wywłaszczające lub niewywłaszczające.

Wywłaszczający algorytm priorytetowy odbiera CPU **bieżącemu** procesowi, jeśli jego priorytet jest niższy od priorytetu ***nowo*** przybyłego procesu.

Niewywłaszczający algorytm priorytetowy ustawi **nowy** proces na czele kolejki procesów ***Gotowych*** do wykonania.

❒ **Problemem** w **planowaniu priorytetowym**

N**ieskończone blokowanie** *(indefinite blocking),* **zwane *głodzeniem*** *(starvatiori).*

Proces gotowy do wykonania, lecz pozbawiony CPU, można traktować jako **zablokowany** z powodu oczekiwania na przydział CPU.

Algorytm planowania priorytetowego może pozostawić ***niskopriorytetowe*** Procesy w stanie niekończącego się czekania na CPU.

Napływ procesów o **wyższych** priorytetach może nie dopuścić ***niskopriorytetowe*** procesy do CPU.

Dochodzi wówczas do jednego z dwu zdarzeń:

- oczekujący Proces zostanie w końcu wykonany,
- wystąpi awaria i niedokończone, ***niskopriorytetowe*** Procesy będą utracone.

Postarzanie *(paging)* rozwiązuje problem blokowania Procesów ***niskopriorytetowych***.

☛ Następuję stopniowe **podwyższanie** priorytetów Procesów **długo** oczekujących w systemie.

Można przykładowo podwyższać priorytet procesu o 1 co każde 30 minut.

5.1.4. Planowanie rotacyjne

Planowanie rotacyjne (***R**ound-**R**obin* - RR) -algorytm dla systemów z podziałem czasu.

Ustala się jednostkę czasu, zwaną kwantem czasu (zwykle 10 ÷ 100 ms).

Jest podobny do FCFS, z dodanym wywłaszczanie w celu przełączania procesów.

Kolejka procesów gotowych do wykonania jest traktowana jak **kolejka cykliczna**.

Planista przydziału CPU przegląda kolejkę cykliczną i każdemu procesowi przydziela odcinek czasu o długości **jeden kwant**.

Implementacja **RR** wykorzystuje strukturę kolejki FIFO (pierwszy na We, pierwszy na Wy), do przechowywania kolejki procesów gotowych.

Nowe Procesy dołączane są na **KOŃCU** kolejki.

Planista przydziału CPU bierze pierwszy proces z kolejki, **ustawia czasomierz** na przerwanie po upływie **1-go kwantu** czasu, po czym ekspediuje ten proces do CPU.

Dwie możliwe sytuacje:

1. Proces ma fazę CPU ***krótszą*** niż 1 kwant czasu i z własnej inicjatywy zwolni procesor.
   Planista pobiera następny proces z kolejki procesów gotowych.
2. Proces ma fazę CPU **dłuższą** niż **1** kwant czasu, to wystąpi przerwanie zegarowe.
   Nastąpi przełączenie kontekstu i proces zostanie odłożony na **KONIEC** kolejki procesów GOTOWYCH, planista zaś wybierze następny proces z tej kolejki.

Średni czas oczekiwania w metodzie rotacyjnej może być długi.

Trzy procesy, o zadanych fazach CPU, przybyły w czasie t = 0. **Kwant czasu 4** ms.

Proces	Fazy procesu
P1	**24**
P2	3
P3	*3*

P1	P2	P3	P1	P1	P1	P1	P1
0	**4**	**7**	**10**	**14**	**18**	**22**	**26**

Proces **P1** otrzyma pierwsze **4 ms**; potrzebuje on jeszcze **20** ms, więc zostanie wywłaszczony po upływie pierwszego kwantu.

CPU będzie przydzielony następnemu procesowi w kolejce → **P2**.

Faza procesu **P2** trwa krócej niż **4** ms, cały proces zakończy pracę przed upływem kwantu czasu.

Następny proces **P3** otrzyma przydział CPU i cały zakończy się przed upływem kwantu czasu.

CPU będzie z powrotem przydzielany procesowi **P1**.

Średni czas oczekiwania wynosi: **(0 + 4 +7 + (10 - 4))** = **17/3** = **5.66** ms.

Algorytm planowania rotacyjnego jest wywłaszczający.

Żaden proces nie otrzyma więcej niż **1 kwant** czasu CPU za jednym razem.

Jeżeli podłączenie procesu do CPU **przekracza 1 kwant** czasu, to proces jest wywłaszczany i przenoszony do kolejki procesów ***Gotowych***.

Kolejka procesów Gotowych do wykonania składa się z **n** procesów, zaś kwant czasu wynosi **q.**

Każdy proces dostaje **1/n** czasu CPU porcjami, których wielkość nie przekracza **q** jednostek.

Następny kwant czasu każdy proces otrzyma nie dłużej niż za *(***n - 1)** x **q** jednostek czasu.

Przy **5**-ciu procesach i kwancie **20** ms, każdy proces dostaje po 20 ms co każde **100** ms.

Wydajność algorytmu rotacyjnego zależy od rozmiaru kwantu czasu.

Jeżeli kwant czasu jest ***bardzo długi*** metoda rotacyjna sprowadza się do algorytmu FCFS.

Dzielenie procesora *(processor sharing)* metoda **RR** dla bardzo małego kwantu czasu.

Użytkownik ma wrażenie, że każdy z **n** procesów ma własny CPU działający z **1/n** szybkości rzeczywistego procesora.

➡ **Należy uwzględnić wpływ przełączania kontekstu na zachowanie algorytmu RR.**

Mniejszy kwant czasu zwiększa czas tracony na przełączanie kontekstu.

Mamy **jeden Proces**, którego faza CPU ma długość **10** jednostek czasu.

Kwant czasu **12** j.: proces skończy się w czasie **krótszym** niż 1 kwant.

Kwant czasu **6** j.: proces potrzebuje **dwu** kwantów ⇐ wystąpi przełączanie kontekstu.

Kwant czasu **1** j.: nastąpi **9** przełączeń **kontekstu** ⇐ spowolnienie wykonanie procesu.

Kwant czasu powinien być długi w porównaniu z czasem przełączania kontekstu.

Od rozmiaru kwantu czasu zależy również czas cyklu przetwarzania.

Dla dużego kwantu czasu planowanie rotacyjne sprowadza się do schematu FCFS.

Średni czas przetwarzania zbioru procesów nie musi poprawiać się ze wzrostem kwantu czasu.

Średni czas przetwarzania poprawia się, kiedy większość procesów kończy swoje kolejne fazy CPU w pojedynczych kwantach czasu.

Trzy procesy o długość **10 j.** czasu :

- kwant czasu **1 j**. *średni czas oczekiwania* = 29 j.
- kwancie czasu **10 j**. *średni czas oczekiwania* = 20 j.

5.2. Wielopoziomowe planowanie kolejek

Procesy mogą być: -**pierwszoplanowe** *(foreground)* ,
-**drugoplanowe** *(background).*

Różnią się wymaganiami na czasy odpowiedzi lub metodami planowania.

Algorytm wielopoziomowego planowania kolejek rozdziela kolejkę procesów Gotowych na osobne kolejki, w zależności od cech, jak rozmiar pamięci, priorytet lub typ procesu.

Każda kolejka ma własny algorytm planujący.

Procesy pierwszoplanowe i drugoplanowe mogą być ulokowane w osobnych kolejkach.

Do kolejki procesów pierwszoplanowych można zastosować algorytm planowania rotacyjnego.

Do kolejki procesów drugoplanowych algorytm planowania przydziału procesora metodą FCFS.

Musi istnieć planowanie między **kolejkami,** na ogół realizowane za pomocą **stałopriorytetowego planowania z wywłaszczeniami.**

Kolejka pierwszoplanowa może mieć bezwzględny priorytet nad kolejką drugoplanową.

1. **Procesy systemowe** ← najwyższy priorytet kolejki
2. Procesy interakcyjne
3. Procesy redagowania interakcyjnego
4. Procesy wsadowe
5. Procesy studenckie

Każda kolejka może mieć bezwzględne pierwszeństwo przed kolejkami o **niższych** priorytetach.

Żaden proces z kolejki procesów ***wsadowych*** nie może pracować, dopóki kolejki procesów systemowych, interakcyjnych i interakcyjnego redagowania nie są puste.

Gdyby proces interakcyjny nadszedł do kolejki procesów Gotowych w chwili, w której CPU wykonywałby proces wsadowy, wtedy proces ***wsadowy*** zostałby **wywłaszczony**.

☛ Można operować przedziałami czasu między kolejkami.

Każda kolejka dostaje porcję czasu CPU i rozdziela go między znajdujące się w niej procesy.

Kolejka procesów pierwszoplanowych może otrzymać 80% czasu CPU a kolejka drugoplanowa pozostałe 20% do rozdysponowania między jej procesy.

❒ Planowanie wielopoziomowych kolejek ze sprzężeniem zwrotnym

Istnienie osobnych kolejek dla procesów pierwszo i drugoplanowych wyklucza możliwość przechodzenia procesu z jednej kolejki do drugiej.

Planowanie wielopoziomowych kolejek ze sprzężeniem zwrotnym umożliwia przemieszczanie Procesów między kolejkami.

Ideą jest rozdzieleniu procesów o **różnych długościach faz** CPU.

Proces zużywający **dużo czasu** CPU będzie przeniesiony do kolejki o ***niższym*** priorytecie.

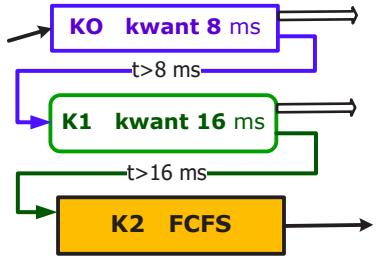
Procesy $\mathbf{P_{We/Wy}}$ i procesy **interakcyjne** można ustawić w kolejkach o wyższych priorytetach.

Proces oczekujący długo w ***niskopriorytetowej*** kolejce może zostać przeniesiony do kolejki o **wyższym** priorytecie.

● Rozważmy trzy kolejki: **K0** (kwant 8 ms), **K1**(kwant 16 ms) , **K2**.

Metodyka tradycyjna:
Planista zleca wykonanie wszystkich procesów z **K0** a następnie wszystkich z **K1**.
Procesy z **K2** nie będą wykonywane dopóki kolejki **K0** i **K1** nie zostaną opróżnione.
+ Proces, który nadejdzie do **K1**, wywłaszczy proces należący do **K2**.
+ Proces, który nadejdzie do **K0** wywłaszczy proces należący do **K1**.

Proces wchodzi do kolejki procesów ***Gotowych***, trafia do **K0** i dostaje **kwant** czasu = **8** ms.
Jeśli nie zostanie w tym czasie zakończony, to zostaje **przeniesiony** na koniec kolejki **K1**.
Gdy **K0** opróżni się, wówczas proces z czoła **K1** dostanie kwant czasu **16** ms.
Jeśli **nie zdąży** ukończyć pracy w tym czasie, to zostanie **wywłaszczony** i trafi do **K2**.



Procesy w **K2** są wykonywane metodą **FCFS** lecz tylko wtedy, gdy kolejki **K0** i **K1** są **puste.**

Taki algorytm planowania daje najwyższy priorytet procesom, których fazy CPU **nie** przekraczają **8** ms (kwant czasu).

Procesy potrzebujące więcej niż **8 ms**, lecz mniej niż **24** ms są obsługiwane z **niższym** priorytetem niż procesy krótsze.

Długie procesy **automatycznie** wpadają do kolejki **K2** i są obsługiwane w porządku FCFS.

5.3. Planowanie wieloprocesorowe

System zawiera **identyczne** procesory *(homogeniczne)* pod względem wykonywanych funkcji.

Do wykonania dowolnego procesu z kolejki można użyć każdego dostępnego CPU.

W systemie urządzenie We/Wy podłączone jest do jednego z CPU za pomocą prywatnej szyny.
Procesy chcące korzystać z tego urządzenia ***muszą trafiać*** do tego CPU.

Dla jednakowych CPU można zastosować dzielenie obciążeń *(load sharing).*

Z każdym CPU można związać **oddzielną kolejkę**

⇒ procesor z pustą kolejką byłby bezczynny, zaś inny byłby przeciążony.

➔ Aby zapobiec tej sytuacji stosuje się **wspólną kolejkę** procesów ***Gotowych*** do działania.

☛ Procesy trafiają do **jednej kolejki** i są przydzielane do dowolnego z dostępnych CPU.

Można zastosować jedną z dwu metod planowania:

❶ Każdy CPU sam planuje swoje działanie.

Każdy CPU przegląda kolejkę procesów Gotowych, z której wybiera **Proces** do wykonania.

Jeśli wiele CPU próbuje korzystać ze wspólnej struktury danych i zmieniać ją, to każdy z nich musi być starannie oprogramowany.

Należy zadbać, aby dwa CPU nie wybrały tego samego **procesu** oraz aby nie ginęły Procesy z kolejki.

❷ Jeden CPU pełnieni funkcję planisty pozostałych CPU - struktura **master-slave**.

Struktura **master-slave** może być rozbudowana.

-Jeden CPU *(serwer główny)* podejmuje wszystkie decyzje planistyczne, wykonuje operacje We/Wy i inne czynności systemowe.

-Pozostałe CPU wykonują tylko kod użytkowy.

Tego typu **asymetryczne wieloprzetwarzanie** *(asymmetric multiprocessing)* jest prostsze od wieloprzetwarzania symetrycznego, ponieważ dostęp do systemowych struktur danych ma tylko jeden CPU → rozwiązany jest problem kontroli dzielenia danych.

5.4. Planowanie w czasie rzeczywistym

❒ **Rygorystyczne systemy czasu rzeczywistego** *(**H**ard **R**eal-**T**ime **S**ystems).*

Proces musi zawierać informację określającą **ilość czasu**, wymaganą do zakończenia.

Na podstawie tych danych planista akceptuje proces, zapewniając jego wykonanie na czas lub odrzuca zlecenie jako niewykonalne - **rezerwacja zasobów** *(resource reservation).*

Planista musi mieć pełne rozeznanie o czasie zużywanym przez wszystkie funkcje SO.

Każdej operacja powinna mieć gwarantowany maksymalny czas wykonania.

Gwarancje nie są możliwe w systemie z pamięcią **zewnętrzna** lub **wirtualną**, gdyż w podsystemach tych występują nieuniknione odchylenia czasu wykonania poszczególnych procesów.

Konieczne jest specjalne oprogramowanie, działające na sprzęcie **przypisanym na stałe** do krytycznych procesów.

❒ **Łagodne systemy czasu rzeczywistego** *(**S**oft **R**eal-**T**ime **S**ystems)* są mniej restrykcyjne.

Wymaga się aby Procesy o decydującym znaczeniu miały **priorytet** nad innymi.

W łagodnych systemach czasu rzeczywistego może dochodzić do **niesprawiedliwego** przydział zasobów i powodować większe opóźnienia lub głodzenie niektórych procesów.

Implementacja SRTS wymaga starannego zaprojektowania planisty.

❶ System musi mieć planowanie priorytetowe.

Procesy działające w czasie rzeczywistym muszą mieć najwyższy priorytet i **nie może on MALEĆ** z upływem czasu w żadnym przypadku.
Można zakazać **postarzania** procesów w obecności procesów czasu rzeczywistego.

❷ Opóźnienie ekspediowania procesów do CPU musi być małe.

Im będzie mniejsze, tym szybciej Proces czasu rzeczywistego będzie mógł rozpocząć działanie, poczynając od chwili, w której jest do niego gotowy.

Trudność spełnienie drugiego wymagania:, wiele SO (także w UNIX), przed przełączeniem kontekstu musi **poczekać** do zakończenia funkcji systemowej lub zablokować Procesu z powodu operacji We/Wy.

Opóźnienie ekspedycji może być znaczne, gdyż niektóre z funkcji systemowych są złożone, a pewne urządzenia zewnętrzne działają powoli.

Niski poziom opóźnienia ekspedycji wymaga wywłaszczania funkcji systemowych.

Można wstawić do **długotrwałych** funkcji systemowych **punkty wywłaszczeń** *(preemption points)*, w których sprawdza się, czy wysokopriorytetowy proces nie wymaga uaktywniania.

Jeśli TAK, to następuje przełączenie kontekstu i działanie przerwanej funkcji systemowej podejmuje się dopiero po wykonaniu procesu o **wysokim** priorytecie.

Punkty wywłaszczeń można umieszczać tylko w „**bezpiecznych**" miejscach jądra, tj. w takich, w których nie są zmieniane struktury danych jądra.

Można spowodować aby całe jądro było wywłaszczalne.

W celu zapewnienia poprawności działań, wszystkie struktury danych jądra muszą być chronione za pomocą ***różnorodnych*** mechanizmów synchronizacji.

Jądro można wywłaszczyć w dowolnej chwili, gdyż wszystkie aktualizowane dane jądra są chronione przed zmianami za pomocą wysokopriorytetowego procesu.

● **Proces** o **wyższym** priorytecie chce zmienić dane **jądra** w chwili, w której korzysta z nich inny proces o ***niższym*** priorytecie !!!!

☛ Wysokopriorytetowy proces musiałby czekać na zakończenie procesu o **niższym** priorytecie.

Sytuacja ta nosi nazwę **odwrócenia priorytetów** *(priority inversiori).*

Może powstać łańcuch procesów korzystających z zasobów potrzebnych procesowi wysokopriorytetowemu.

Problem rozwiązuje **protokół dziedziczenia priorytetów** *(priority-inheritance protocol)*:

-**wszystkie** Procesy (używające zasobów potrzebnych procesowi wysokopriorytetowemu) otrzymują **wysoki** priorytet, ***dopóki nie przestaną*** korzystać ze spornych zasobów.

Po skończeniu działania tych Procesów ich priorytety wracają do swojej pierwotne wartości.

5.5. Kryteria wyboru algorytmu

Jak wybrać algorytm planowania przydziału CPU dla konkretnego systemu?

Należy zdefiniować kryteria stosowane przy wyborze algorytmu (stopień wykorzystania procesora, czas odpowiedzi, przepustowość) oraz określić **względną ważność** tych miar np.:

- maksymalizacja wykorzystania CPU przy założeniu, że maksymalny czas odpowiedzi wyniesie 1 s;
- maksymalizacja przepustowości, aby średni czas cyklu przetwarzania był liniowo proporcjonalny do ogólnego czasu wykonania.

5.5.1. Modelowanie deterministyczne

Modelowanie deterministyczne *(deterministic modeling)* należy do klasy metod analitycznych.

Przyjmuje się z góry określone obciążenie robocze systemu i definiuje zachowanie każdego algorytmu w warunkach tego obciążenia.

Rozważmy pięć procesów, które nadeszły w chwili t = 0.

Proces	Czas trwania fazy
P1	10
P2	29
P3	3
P4	7
P5	12

Porządek wykonania procesów wg. algorytmu **FCFS**

P1	P2	P3	P4	P5
0	10	39	42	49

Dla **P1** czas oczekiwania 0 ms, dla **P2** = 10 ms, dla **P3** = 39 ms, dla **P4** = 42 ms, dla **P5** = 49 ms.

Średni czas oczekiwania = (0 + 10 + 39 + 42 + 49)/5 = **28** ms.

Algorytm planowania **SJF** (*Shortest-Job-First*), niewywłaszczający

Czas oczekiwania:
P1 = 10 ms, P2 = 32 ms,
P3 = 0 ms, P4 = 3 ms.

P3	P4	P1	P5	P2
0	3	10	20	32 1

Średni czas oczekiwania = (10 + 32 + 0 + 3 + 20)/5 = **13** ms.

Algorytm planowania rotacyjny **RR** dla kwantu czasu **10** ms

Czas oczekiwania:
P1 = 0 ms, P2 = 32
P3 = 20 ms, P4 = 23 ms,
P5 = 40 ms.

Proces **P2** po **10** ms zostaje wywłaszczony i wędruje na koniec kolejki.

P1	P2	P3	P4	P5	P2	P5	P2
0	10	20	23	30	40	50	52

Średni czas oczekiwania = (0 + 32 + 20 + 23 + 40)/5 = **23** ms.

Modelowanie deterministyczne pozwala na porównywanie algorytmów.

Wymaga dokładnych wejściowych, zaś odpowiedzi odnoszą się tylko do zadanych przypadków.

Znajduje zastosowanie w opisywaniu algorytmów planowania i dostarczaniu przykładów.

Może służyć do wyboru algorytmu planowania, gdy wykonywanie tych samych programów powtarza się, przy czym istnieje możliwość dokładnego pomiaru wymagań związanych z ich przetwarzaniem.

Dla zbioru przykładów może wskazywać tendencje, które można osobno analizować.

Modelowanie deterministyczne wymaga dokładnej wiedzy lecz nie jest ogólnie użyteczne.

Modele obsługi kolejek

Procesy wykonywane w różnych systemach zmieniają się.

Trudno mówić o **statycznym** zbiorze procesów (pomiarów czasu), który mógłby stanowić podstawę modelowania deterministycznego.

Można wyznaczyć rozkłady statystyczne faz CPU oraz We/Wy i oszacowywać wartości.

➔ Otrzymuje się matematyczny opis prawdopodobieństwa wystąpienia **faz** CPU.

☛ Potrzebny jest rozkład **czasów przybywania** Procesów do systemu.

Na podstawie tych **dwu** rozkładów można obliczyć:
-średnią przepustowość,
-wykorzystanie procesora,
-czas oczekiwania,
itp. dane dla większości algorytmów.

❒ **System komputerowy można opisać jako sieć usługodawców**, czyli serwerów.

Każdy serwer ma kolejkę czekających procesów.

Serwerem jest: -CPU z kolejką procesów gotowych do wykonania,
-system We/Wy z kolejkami do urządzeń.

Znając **tempo** nadchodzenia zamówień i **czasy** wykonywania usług, można obliczyć:
-wykorzystanie CPU,
-średnie długości kolejek,
-średnie czasy oczekiwania itd.

Mówimy o **analizie obsługi kolejek w sieciach** *(queuing-network analysis).*

Oznaczenia: **n** -średnia długość kolejki (nie licząc procesu aktualnie obsługiwanego).
W -średni czas oczekiwania w kolejce,
λ -**tempo** przybywania nowych Procesów do kolejki (np. trzy procesy na sekundę).

Założenie: w czasie **W** (proces czeka), w kolejce pojawi się $\lambda \bullet W$ nowych procesów.

Jeśli system jest ustabilizowany, to **liczba** Procesów opuszczających kolejkę musi się równać liczbie procesów przybywających do niej:

$$n = \lambda \bullet W$$ wzór ***Little'a***

Wzór Little'a obowiązuje dla dowolnego algorytmu planowania i rozkładu przybyć.

Jeżeli w każdej sekundzie przybywa średnio **7** procesów oraz w kolejce znajduje się **14** procesów, to średni czas oczekiwania wyniesie **2 s** na proces.

Rozkłady przybyć i obsługi często definiowane są z pewnymi założeniami, pozwalającymi na wykonanie analiz matematycznych.

Niektóre z przyjętych założeń nie zawsze są realistyczne.

Modele obsługi kolejek są przybliżeniami rzeczywistych systemów.