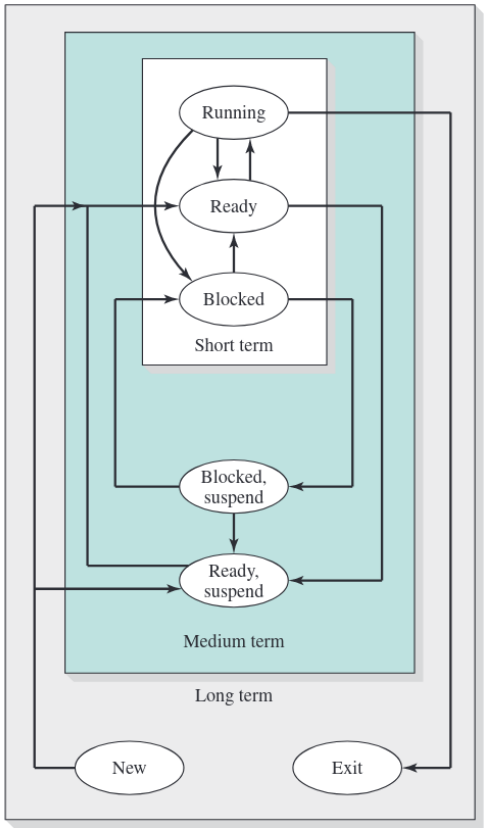
Planowanie długoterminowe, krótkoterminowe i średnioterminowe



Ostatnia aktualizacja: 2 grudnia 2025

- **planowanie długoterminowe** (*long-term scheduling*) – decyduje o tym, które nowe procesy zostaną załadowane do pamięci głównej i będą mogły konkurować o zasoby systemowe (CPU, pamięć, urządzenia wejścia-wyjścia)
- **planowanie krótkoterminowe** (*short-term scheduling*) – decyduje o tym, który z gotowych do wykonania procesów otrzyma czas procesora (CPU) na wykonanie swoich instrukcji
- **planowanie średnioterminowe** (*medium-term scheduling*) – zarządza wymianą procesów między pamięcią główną a pamięcią wtórną (swapowaniem), co pozwala na lepsze wykorzystanie zasobów systemowych i kontrolę nad liczbą aktywnych procesów w pamięci

Algorytmy przydziału procesora

Planista krótkoterminowy (ang. scheduler) lub **dyspozytor** – część systemu operacyjnego odpowiedzialna za przydzielanie czasu procesora w ramach przełączania zadań. Jeśli dwa lub więcej procesów znajduje się w stanie *gotowy do wykonania*, to o kolejności przydziału CPU decyduje planista wykonując **algorytm szeregowania** realizujący politykę szeregowania.

Polityka szeregowania określa ogólne zasady i cele, jakimi kieruje się system operacyjny przy przydzielaniu czasu procesora procesom.

Planista powinien gwarantować:

- sprawiedliwość przydziału CPU
- dobrą wydajność CPU
- mały czas odpowiedzi
- mały czas przetwarzania (*turnaround time*)
- dużą przepustowość (*throughput*)

Procesy ograniczone obliczeniowo i operacjami wej-wyj

Klasyfikacja procesów ze względu na charakterystykę wykorzystania CPU:

- zorientowane na wejście-wyjście (*I/O-bound*)
  Wymagają szybkiego przydziału czasu procesora i częstszych przełączeń kontekstu, aby efektywnie obsługiwać operacje wejścia-wyjścia.
- zorientowane na obliczenia (*CPU-bound*)
  Wymagają dłuższego czasu procesora i rzadszych przełączeń kontekstu, aby zmaksymalizować wydajność obliczeniową

Wraz ze wzrostem szybkości CPU procesy mają tendencję stawać się bardziej ograniczone przez operacje wej-wyj

Szeregowanie bez wywłaszczenia

Algorytm szeregowania **bez wywłaszczania** (*non-preemptive*) wybiera proces do uruchomienia a następnie pozwala mu działać do czasu zablokowania (np. operacja we-wyj) lub do momentu, gdy proces dobrowolnie zwolni CPU

- prosty w implementacji, małe narzuty związane z przełączaniem kontekstu
- przerwanie zegarowe nie powoduje podjęcia decyzji o szeregowaniu
- wykorzystywany w systemach wsadowych, nie wymagających interakcji z użytkownikiem
- może prowadzić do mniejszej responsywności systemu dla procesów interaktywnych
- ryzyko głodzenia procesów o niższym priorytecie

Szeregowanie z wywłaszczeniem

Algorytm szeregowania **z wywłaszczaniem** (*preemptive*) wybiera proces i pozwala mu działać maksymalnie przez ustalony czas (kwant czasu)

- procesy mogą być przerwane w dowolnym momencie, zazwyczaj przerwaniami zegara, aby umożliwić planiście podjęcie decyzji o szeregowaniu
- umożliwiają sprawiedliwsze przydzielanie czasu procesora w systemach wielozadaniowych
- krótkie kwanty czasu faworyzują procesy interaktywne, poprawiając responsywność systemu
- większe narzuty związane z częstymi przełączeniami kontekstu, zbyt krótkie kwanty mogą prowadzić do spadku wydajności systemu
- dłuższe kwanty czasu zmniejszają narzuty związane z przełączaniem kontekstu, co jest korzystne dla procesów obliczeniowych, kosztem responsywności systemu
- trudniejsze w implementacji

Cele algorytmów szeregowania w różnych środowiskach

Wszystkie systemy

- **sprawiedliwość** - przydział procesom uczciwego dostępu do CPU
- **równowaga** - wszystkie części systemu powinny być zajęte

Systemy interaktywne

- **czasu odpowiedzi** (*response time*) - maksymalizacja responsywności procesów interaktywnych
- **proporcjonalność** - spełnienie oczekiwań użytkowników dotyczących czasu wykonania

Systemy wsadowe

- **przepustowość** - maksymalizacja liczby wykonanych zadań na godzinę
- **czas cyklu przetwarzania** - minimalizacja czasu pomiędzy rozpoczęciem i zakończeniem procesu
- **wykorzystanie CPU** - maksymalizacja zajętości

Systemy czasu rzeczywistego

- **dotrzymywanie terminów** - unikanie utraty danych
- **przewidywalność** - unikanie degradacji jakości w systemach multimedialnych

Planowanie przydziału procesora - mechanizmy

- pierwszy nadszedł, pierwszy obsłużony (FCFS, *First-Come, First-Served*)
- najkrótsze zadanie najpierw (SJF, *shortest job first*)
- rotacyjne, cykliczne (*Round-Robin*)
- priorytety: statyczne i dynamiczne
- klasy szeregowania: dwupoziomowe, wielopoziomowe

Pierwszy nadszedł, pierwszy obsłużony (FCFS)

First-Come, First-Served

- pojedyncza kolejka gotowych procesów
- procesy są wykonywane w kolejności, w jakiej zgłaszają się do systemu, bez względu na ich czas trwania czy priorytet
- proces jest wywłaszczany w momencie zablokowania, kolejny proces zostaje wybrany do wykonania
- odblokowany proces trafia na koniec kolejki gotowych

FCFS: wady i zalety

Zalety:

- prosty w implementacji
- sprawiedliwy, każdy proces jest obsługiwany w kolejności zgłoszeń
- minimalne narzuty związane z przełączaniem kontekstu
- przydatny w systemach wsadowych, gdzie zadania są długotrwałe i nie wymagają interakcji

Wady:

- długie procesy mogą nadmiernie blokować CPU, na niekorzyść krótkich zadań, tzw. zjawisko **konwoju** (*convoy effect*)
- nie jest optymalny dla systemów interaktywnych, gdzie responsywność jest kluczowa

Najkrótsze Zadanie Najpierw (SJF, *Shortest Job First*)

- procesy gotowe są sortowane i wykonywane w kolejności ich przewidywanego czasu wykonania, od najkrótszego do najdłuższego
- proces o najkrótszym przewidywanym czasie wykonania jest wybierany jako pierwszy do uruchomienia
- SJF minimalizuje średni czas oczekiwania procesu, ponieważ krótsze zadania są obsługiwane szybciej, co zmniejsza opóźnienia w kolejce
- w praktyce trudno jest dokładnie oszacować czas wykonania każdego zadania
- dłuższe procesy mogą być stale opóźniane (głodzenie), jeśli nowe, krótsze zadania ciągle się pojawiają w kolejce
- działa efektywne w systemach, gdzie czas wykonania procesów jest znany lub może być oszacowany z dużą dokładnością (systemy wsadowe)

Najkrótsze Zadanie Najpierw (SJF, *Shortest Job First*)



Figure 2-41. An example of shortest-job-first scheduling. (a) Running four jobs in the original order. (b) Running them in shortest job first order.

Czas oczekiwania w oryginalnym porządku:
8, 12, 16, 20, średnio 14

Czas oczekiwania w kolejności od najkrótszego zadania:
4, 8, 12, 20, średnio 11

Algorytm **najkrótszego pozostałego czasu** (SRT, *Shortest Remaining Time*)

- wariant SJF z wywłaszczeniem
- proces o najkrótszym pozostałym czasie wykonania jest wybierany do uruchomienia
- SRT stale monitoruje pozostały czas wykonania procesów w kolejce gotowych
- gdy nowy proces o krótszym czasie wykonania pojawia się w kolejce gotowych, bieżący proces jest wywłaszczany i nowy proces jest uruchamiany
- minimalizuje średni czas oczekiwania procesu

Algorytm **najkrótszego następnego procesu** (SPN, *Shortest Process Next*)

- odmiana SJF, która używa historii uruchomień i zachowania procesów do estymacji czasu wykonywania
- Przykładowa estymacja czasu wykonywania procesu (*aging*) oparta na średniej wykładniczej:

$$T_{n+1} = aT_n + (1 - a)T_{n-1}$$

gdzie $0 \leqslant a \leqslant 1$.

Algorytm rotacyjny (*Round-Robin*)

- procesy gotowe do wykonania są zorganizowane w postaci kolejki, każdy proces otrzymuje kwant czasu
- działa analogicznie do FCFS ale z wywłaszczeniem, wybierany jest zawsze pierwszy proces z kolejki
- gdy kwant czasu dobiegnie końca proces jest wywłaszczany i umieszczany na końcu kolejki
- jeśli proces zakończy się przed upływem kwantu czasu lub zostanie zablokowany to CPU jest przydzielane kolejnemu procesowi z kolejki
- kwant czasu typowo ok. 20-50 msec, wartość optymalna zależy od charakterystyki obciążenia systemu

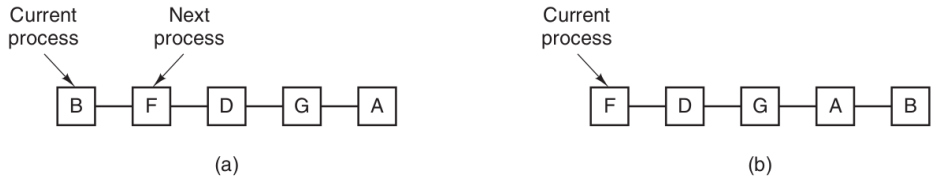


Figure 2-42. Round-robin scheduling. (a) The list of runnable processes. (b) The list of runnable processes after *B* uses up its quantum.

Planowanie priorytetowe

- decyzja planisty zależna od **priorytetów** przypisanych do zadań, ważniejsze zadania otrzymują wyższe priorytety i mają pierwszeństwo
- priorytety mogą być przydzielane **statycznie** lub **dynamicznie**
- statyczne priorytety są przypisywane na podstawie właściwości procesu i nie zmieniają się w trakcie jego życia
- dynamiczne priorytety mogą zmieniać się w zależności od zachowania procesu, np. procesy interaktywne (które często czekają na dane od użytkownika) mogą być promowane przez podwyższenie priorytetu aby zwiększyć responsywność
- procesy mogą zostać pogrupowane względem priorytetów tworząc **klasy priorytetów**. Planowanie między klasami odbywa się zgodnie z priorytetami, zaś w ramach konkretnej klasy wg. wybranego algorytmy szeregowania (np. cyklicznego)

Klasy priorytetów

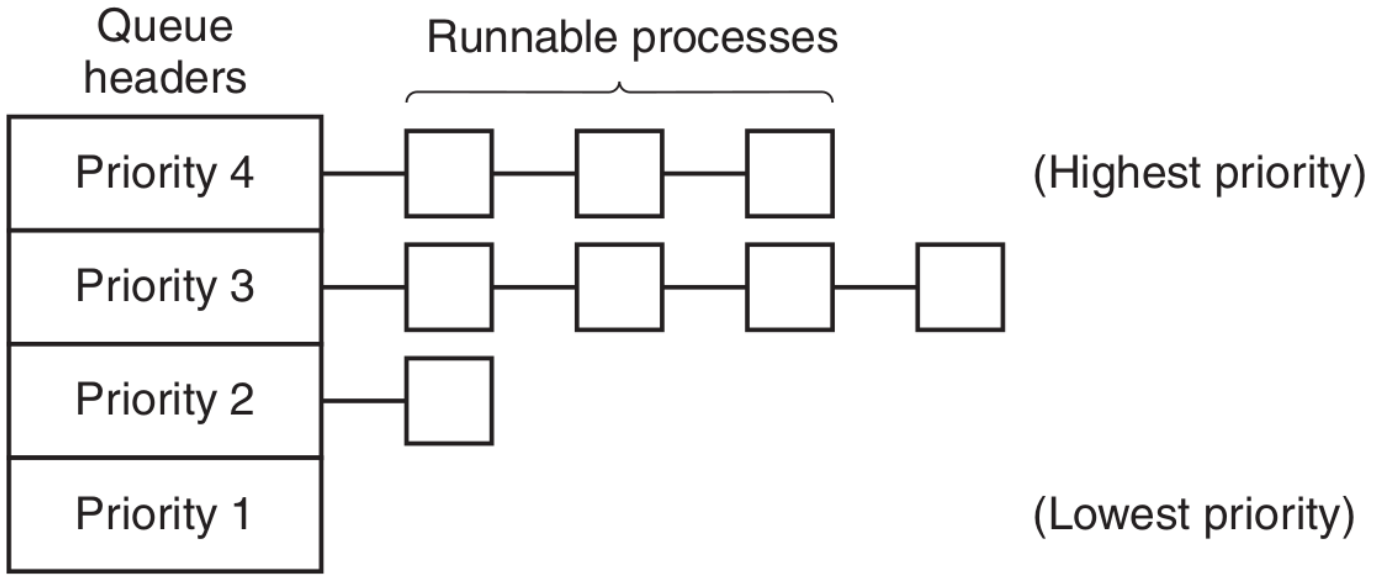


Figure 2-43. A scheduling algorithm with four priority classes.

Algorytm szeregowania w Linuksie

- szeregowaniu podlegają wątki jądra
- 140 priorytetów: [0,139], gdzie niższa wartość oznacza wyższy priorytet

Dwie główne klasy zadań:

- zadania czasu rzeczywistego: priorytety statyczne [0,99]
  polityki planowania: `SCHED_FIFO`, `SCHED_RR`
- zadania normalne: priorytety dynamiczne [100,139]
  polityki planowania: `SCHED_OTHER`

Algorytmy planowania:

- **planista O(1)** używany od wersji jądra 2.5 (11-2001)
- **planista całkowicie sprawiedliwy** (CFS, *Completely Fair Scheduler*) o złożoności O(log N) pojawił się w wersji 2.6.23 (10-2007)

,

Nikita Ishkov, *A complete guide to Linux*

nazwa nie odnosi się do systemów czasu rzeczywistego, nie oznacza spełniania rygorystycznych ograniczeń czasowych

Priorytety czasu rzeczywistego

- priorytety zadań czasu rzeczywistego są z zakresu 0-99, priorytety nie ulegają zmianie w czasie wykonywania
- zadania wykonywane są ściśle w kolejności malejących priorytetów, najpierw zadania czasu rzeczywistego, potem pozostałe zadania

Polityki szeregowania czasu rzeczywistego:

- `SCHED_FIFO` (*First-In, First-Out*) – szeregowanie bez wywłaszczenia analogicznie do FCFS. Do zadań, które muszą być wykonane jak najszybciej i nie mogą być przerwane przez inne procesy o tym samym priorytecie.
- `SCHED_RR` (*Round-Robin*) - szeregowanie rotacyjne (z wywłaszczeniem)

Procesy `SCHED_FIFO` mają zawsze pierwszeństwo przed procesami `SCHED_RR` o tym samym priorytecie.

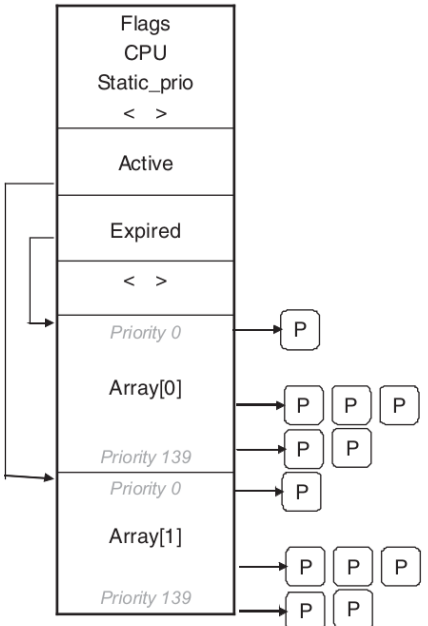
RHEL/CentOS/Fedora ograniczają maksymalne zużycie CPU dla wątków czasu rzeczywistego do 95%; zob. Real Time Scheduler Throttling

Priorytety dynamiczne w planiście O(1)

- klasa `SCHED_OTHER` - standardowe procesy użytkownika o priorytetach dynamicznych z zakresu 100-139
- początkowa wartość priorytetu dynamicznego jest określana przez wartość *poziomu uprzejmości*
- poziomy uprzejmości (*nice*) od -20 do +19 są mapowane na wartości od 100 do 139, domyślna wartość uprzejmości 0 to priorytet 120
- wartość priorytetu procesu jest zmieniana w granicach ± 5 w zależności od interaktywności zadania (stosunek czasu jaki zadanie spędza w zawieszeniu do czasu aktywności jest miarą uzależnienia zadania od operacji wejścia-wyjścia)
- wielkość kwantu czasu procesu zależy od priorytetu dynamicznego, procesy o wyższym priorytecie otrzymują dłuższy kwant czasu
- domyślnie proces otrzymuje kwant 100 ms. Ten kwant może być zwiększony do 800 ms (100) lub zmniejszony do 5 ms (139).
  Nowy proces dziedziczy kwant po swoim rodzicu.

Planista O(1)

- struktura zarządzania aktywnymi zadaniami (*runqueue*) zawiera dwie tablice o rozmiarze 140: zadania aktywne (*active*) i wygasłe (*expired*)
- każdy element tych tablic wskazuje na początek kolejki zadań o wspólnym priorytecie



(a) Per CPU runqueue in the Linux O(1) scheduler

- planista wybiera pierwsze zadanie z listy aktywnych o najwyższym priorytecie
- po wyczerpaniu kwantu czasu zadanie jest umieszczane na liście wygasłych (możliwa jest zmiana priorytetu)
- jeśli zadanie zostanie zablokowane przed upływem kwantu czasu to powraca do listy aktywnych z odpowiednio skróconym kwantem
- gdy wszystkie listy aktywnych zadań się wyczerpią, następuje zamiana tablic, zadania wygasłe stają się znów aktywne
- wybór zadania odbywa się w stałym czasie $O(1)$, niezależnie od liczby zadań
- algorytm gwarantuje, że wszystkie zadania, nawet o niskim priorytecie, mają szansę wykonania
- główną wadą algorytmu jest skomplikowana heurystyka ustalania priorytetów dynamicznych, promująca zadania interaktywne, kosztem zadań obliczeniowych

Co oznacza PRI?

proces/wątek	jądro	"top/htop"	"ps -eo pri"	"ps -eo rtprio"	"ps -el"	"nice"
migration	99	RT	139	99	-40	-
kworker	100	0	39	-	60	-20
bash	120	20	19	-	80	0
khugepaged	139	39	0	-	99	19

	priorytet wg jądra $\geqslant 100$	priorytet wg jądra < 100
top/htop	PRI='priorytet wg jądra' - 100	PRI=RT
ps -eo pri	PRI=139 - 'priorytet wg jądra'	PRI='priorytet wg jądra' + 40
ps -el	PRI='priorytet wg jądra' - 40	PRI='priorytet wg jądra' - 139

Obecne klasy szeregowania w jądrze Linux

Klasy zadań objętych wspólną polityką przydziału CPU:

- czasu rzeczywistego (*real time*): `SCHED_RR`, `SCHED_FIFO`
- sprawiedliwa (*fair*): `SCHED_NORMAL` (dawniej `SCHED_OTHER`) większość procesów użytkownika, zarządzana przez planistę CFS
- `SCHED_BATCH` dla zadań obciążających CPU nieinteraktywnych o niskim priorytecie
- bezczynna: `SCHED_IDLE` dla procesów o bardzo niskim priorytecie (niżej niż +19), które mogą być wykonywane tylko wtedy, gdy system jest bezczynny
- terminowa (*deadline*): `SCHED_DEADLINE` dla zadań o ścisłych wymaganiach czasowych (systemy czasu rzeczywistego)

Przykład: polityka szeregowania procesów w Linuxie

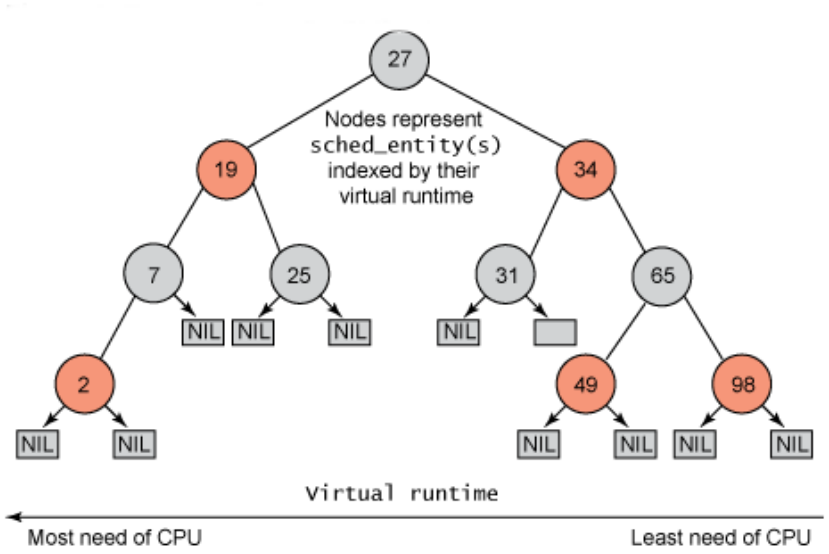
```
$ ps -e -o s,pid,policy,pri,ni,rtprio,command
S     PID POL PRI  NI RTPRIO COMMAND
S       1 TS   19   0      - /sbin/init splash
S       2 TS   19   0      - [kthreadd]
I       4 TS   39 -20      - [kworker/R-rcu_g]
...
S      19 FF  139   -     99 [migration/0]
S    4565 B     4  15      0 /usr/libexec/packagekitd
S    5670 TS    7  12      - /usr/bin/python3 /usr/lib/pop-transition/service.py
R   13056 IDL   0   -      0 /usr/libexec/tracker-miner-fs-3
S   30080 TS   13   6      - /usr/bin/python3 /usr/bin/gnome-terminal --wait
R  464202 TS   19   0      - ps -e -o s,pid,policy,pri,ni,rtprio,command
```

Polecenie chrt ustawia lub odczytuje atrybuty szeregowania

```
$ chrt -p 19
pid 19's current scheduling policy: SCHED_FIFO
pid 19's current scheduling priority: 99
```

Planista kompletnie sprawiedliwy CFS

- struktura zarządzania zadaniami (`runqueue`) jest zorganizowana w postaci drzewa czerwono-czarnego
- zadania w drzewie są uporządkowane w kolejności względem ilości czasu wirtualnego przez jaki zajmowały CPU (wartość `vruntime`)



http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/

Planista CFS

- planista zawsze wybiera do wykonania zadanie o najmniejszym czasie wirtualnym (skrajny węzeł z lewej)
- okresowo planista aktualizuje wirtualny czas działającego zadania porównując z wartością czasu kolejnego zadania znajdującego się w drzewie
- jeżeli w drzewie znajduje się węzeł o nizszym czasie to nastepuje wywłaszczenie i wykonywane zadanie zostaje umieszczone ponownie w drzewie we właściwym miejscu
- różnice w priorytetach zadań oraz uprzejmości (`nice`) są odzwierciedlone w postaci szybkości upływu wirtualnego czasu
- dla zadań o niskim priorytecie czas płynie szybciej, więc szybciej ulegają wywłaszczeniu, na korzyść zadań o wyższym priorytecie