Algorytmy przydziału procesora

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

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 I:

- 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 wej-wyj

W systemach Unix/Linux planista zwykle jawnie faworyzuje procesy ograniczone przez operacje wejścia-wyjścia.

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

- procesy mogą być przerwane (wywłaszczone) w dowolnym momencie, zazwyczaj przez przerwania zegara (*timer interrupts*), aby dać miejsce procesom o wyższym priorytecie
- lepsza responsywność dla procesów interaktywnych
- sprawiedliwsze przydzielanie czasu procesora w systemach wielozadaniowych
- trudniejsze w implementacji
- możliwe większe narzuty związane z częstymi przełączeniami kontekstu

Cele algorytmów szeregowania w różnych środowiskach

Wszystkie systemy

- sprawiedliwość - przydział procesom uczciwego dostępu do CPU
- równowaga - wszystkie części systemu powinny być zajęte
- wymuszenie polityki szeregowania

Systemy interaktywne

- czas odpowiedzi - responsywność
- 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

- pierwszy nadszedł, pierwszy obsłużony (FCFS, *First-Come, First-Served*)
- najkrótsze zadanie najpierw (SJF, *shortest job first*)
- rotacyjne, cykliczne (*Round-Robin*)
- priorytetowe (polityka planowania i mechanizm planowania)
- 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
- przy wznowieniu pracy zablokowanego procesu trafia na koniec kolejki
- prosty w implementacji i sprawiedliwy, bo każdy proces jest obsługiwany w kolejności zgłoszeń
- długie procesy moga nadmiernie blokować CPU, na niekorzyść krótkich zadań
- przydatny w systemach wsadowych, nie jest optymalny dla systemów interaktywnych, gdzie responsywność jest kluczowa

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

- procesy 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
- minimalizuje średni czas oczekiwania, 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
- 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*) - algorytm SJF z wywłaszczeniem, które następuje w momencie pojawienia się nowego zadania o krótszym spodziewanym czasie niz pozostały czas bieżącego zadania

Algorytm **najkrótszego następnego procesu** (SPN, *Shortest Process Next*) - używa historii uruchomień i zachowania procesów do estymacji czasu wykonywania. Stosowany w systemach interaktywnych

Np. średnia ważona aktualnego czasu i poprzedniego (*aging*)

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

Algorytm rotacyjny (*Round-Robin*)

- procesy gotowe do wykonania są zorganizowane w postaci listy, każdy proces otrzymuje kwant czasu
- gdy kwant dobiegnie końca proces jest wywłaszczany i umieszczany na końcu listy
- proces jest również wywłaszczany w momencie zablokowania
- kwant czasu ok. 20-50 msec, gdy za któtrki może powodować spadek wydajności z powodu zbyt częstych przełaczen kontektu, zbyt długi kwant powoduje słabą responsywnośc krótkich interaktywnych zadań

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

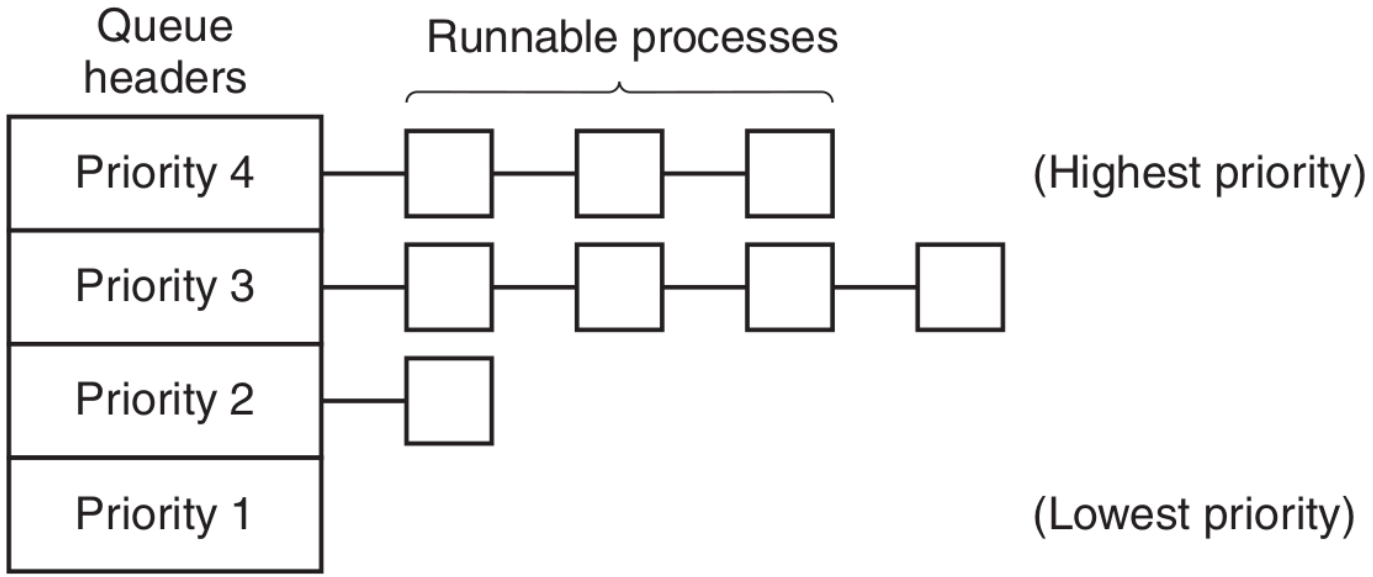


Figure 2-43. A scheduling algorithm with four priority classes.

Algorytm szeregowania w Linuksie

- szeregowaniu podlegają wątki jądra
- 140 priorytetów: [0,139], gdzie niższa wartość oznacza wyższy priorytet
- klasa zadań czasu rzeczywistego: priorytety statyczne [0,99]
  polityki planowania: `SCHED_FIFO`, `SCHED_RR`
- zadania normalne: priorytety dynamiczne [100,139]
  polityki planowania: `SCHED_OTHER`
- **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

Klasy priorytetów czasu rzeczywistego:

- `SCHED_FIFO` (*First-In, First-Out*) – szeregowanie bez wywłaszczenia, zadania są wykonywane w kolejności ich przybycia, aż do zakończenia lub do momentu pojawienia się gotowego zadania o wyższym priorytecie
- `SCHED_RR` (*Round-Robin*) - szeregowanie rotacyjne, każdy proces dostaje równy kwant czasu do wykorzystania, po wyczerpaniu kwantu czasu proces jest przesuwany na koniec kolejki i ponownie czeka na swoją kolej

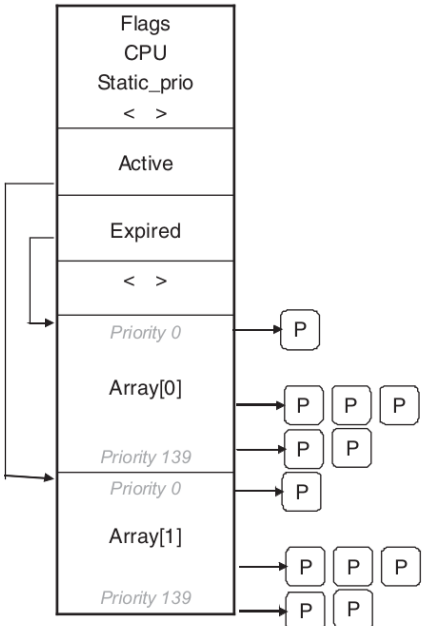
RHEL/CentOS/Fedora ograniczają maksymalne zużycie CPU dla wątków czasu rzeczywistego do 95%; zob. Real Time Scheduler Throttling

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 tablicy 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, niezależnie od liczby zadań
- algorytm gwarantuje, że wszystkie zadania, nawet o niskim priorytecie, mają szansę wykonania
- głowna 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

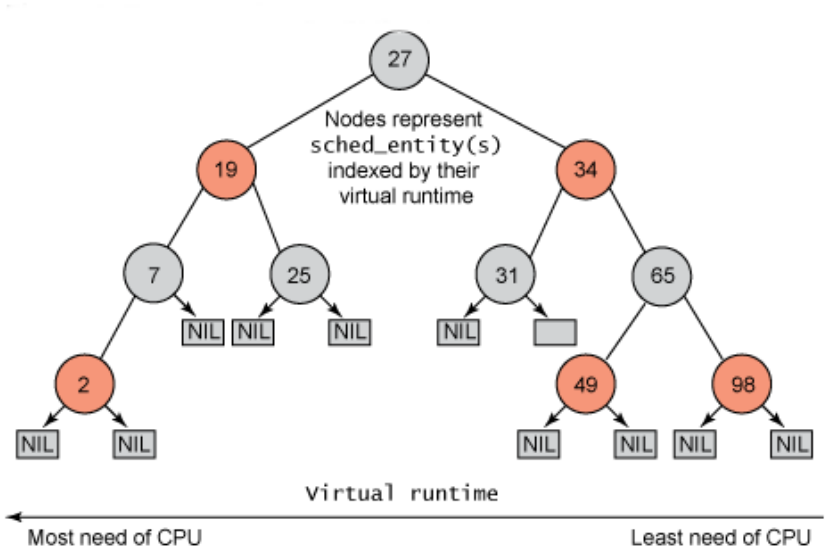
```
$ ps -e -o s,pid,policy,pri,ni,rtprio,command
S     PID POL PRI  NI RTPRIO COMMAND
S       1 TS   19   0      - /sbin/init splash
S       2 TS   19   0      - [kthreadd]
I       4 TS   39 -20      - [kworker/R-rcu_g]
...
S      19 FF  139   -     99 [migration/0]
S    4565 B     4  15      0 /usr/libexec/packagekitd
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 w stosunku do zadań o wyższym priorytecie