Szeregowanie procesów w Linuxie - trendy rozwojowe

Joanna Świetlicka Anton Liashchou Mikołaj Świątek

11 grudnia 2007

Scheduler - wprowadzenie

Zarządzanie czasem procesora

- Wybór kolejnego procesu do wykonania
- Wybór fizycznego procesora dla procesu
- Przydział odpowiedniego kwantu czasu dla procesu

Scheduler - wprowadzenie

Zarządzanie czasem procesora

- Wybór kolejnego procesu do wykonania
- Wybór fizycznego procesora dla procesu
- Przydział odpowiedniego kwantu czasu dla procesu

Wykorzystywane informacje

- Priorytet procesu
- Czas oczekiwania (uśpienia)
- Czas działania

Historia i zmiany

- Trzy główne wersje
- Oryginalny scheduler wprowadzony w wersji 0.01
  Autor: Linus Torvalds
- Scheduler $O(1)$, wprowadzony w wersji 2.6.8
  Autor: Ingo Molnar
- Completely Fair Scheduler, wprowadzony w wersji 2.6.23
  Autor: Ingo Molnar

Kryteria oceny

Wydajność

- Ile czasu procesora używa sam scheduler?
- Rzadsze przełączanie kontekstu zwiększa wydajność

Kryteria oceny

Wydajność

- Ile czasu procesora używa sam scheduler?
- Rzadsze przełączanie kontekstu zwiększa wydajność

Interaktywność

- Jak szybko system odpowiada na działania użytkownika?
- Częstsze przełączanie kontekstu zwiększa interaktywność

Kryteria oceny

Wydajność

- Ile czasu procesora używa sam scheduler?
- Rzadsze przełączanie kontekstu zwiększa wydajność

Interaktywność

- Jak szybko system odpowiada na działania użytkownika?
- Częstsze przełączanie kontekstu zwiększa interaktywność

Sprawiedliwość

- Wprost z Programowania Współbieżnego - brak zagłodzeń i zakleszczeń

Kryteria oceny

Obsługa technologii sprzętowych

- SMP - Symmetric Multi-Processing
- SMT - Symmetric Multi-Threading
- NUMA - Non-Uniform Memory Access

Kryteria oceny

Obsługa technologii sprzętowych

- SMP - Symmetric Multi-Processing
- SMT - Symmetric Multi-Threading
- NUMA - Non-Uniform Memory Access

- Wsparcie dla procesów czasu rzeczywistego

Wymagania jakościowe dla różnych zastosowań

- Desktop : Interaktywność ponad wydajność

Wymagania jakościowe dla różnych zastosowań

- Desktop : Interaktywność ponad wydajność
- Server : Równowaga między interaktywnością a wydajnością

Wymagania jakościowe dla różnych zastosowań

- Desktop : Interaktywność ponad wydajność
- Server : Równowaga między interaktywnością a wydajnością
- HPC : Wydajność ponad interaktywność.

Wymagania jakościowe dla różnych zastosowań

- Desktop : Interaktywność ponad wydajność
- Server : Równowaga między interaktywnością a wydajnością
- HPC : Wydajność ponad interaktywność.
- Wniosek: Nie istnieje scheduler zachowujący się idealnie dla wszystkich zastosowań

Klasyfikacja funkcjonalna

Typy procesów

- Zależne od czasu procesora - “CPU-bound”
- Zależne od urządzeń wejścia-wyjścia - “I/O-bound”
- Czasu rzeczywistego

- Dodatkowo będziemy szczególnie przejmować się mityczną klasą procesów “interaktywnych”

Pojęcia i atrybuty

- Kwant czasu.
- Priorytet statyczny (*nice value*).
- Priorytet dynamiczny.
- Klasy planowania.
- Kolejka procesów gotowych.

Wprowadzenie i historia zmian

- Wprowadzony w pierwszej wersji jądra (0.01) w 1992 roku
- Autor: Linus Torvalds
- Względnie prosta idea i nieskomplikowana implementacja
- Stosunkowo niewiele zmian:

```
* 1996-12-23 Modified by Dave Grothe to fix bugs in semaphores
* and make semaphores SMP safe
* 1998-11-19 Implemented schedule_timeout and related stuff
* by Andrea Arcangeli
```

Linus Torvalds o oryginalnym schedulerze

- Źródło: linux-activists mailing list, Czerwiec 1992

  (...) the scheduler in linux is pretty simple, but does a reasonably good job at giving good IO response while not being too unfair against cpu-bound processes. (...)

- Źródło: plik *sched.c*, komentarz do funkcji *schedule()*

  (...) 'schedule()' is the scheduler function. This is GOOD CODE! There probably won't be any reason to change this, as it should work well in all circumstances (ie gives IO-bound processes good response etc) (...)

Struktury danych

- Pojedyncza, nieuporządkowana lista zawierająca procesy gotowe do wykonania.
- Każdemu procesowi przyporządkowujemy licznik, który oznacza ilość przydzielonego czasu procesora, jaka mu pozostała.
- Procesy czasu rzeczywistego szeregowane są za pomocą strategii RR lub FIFO.

Algorytm

1. Na początku licznik każdego procesu jest ustawiany proporcjonalnie do jego priorytetu.
2. Proces do wykonania wybierany jest na podstawie sumy priorytetu i licznika.
3. Proces ten wykonuje się do czasu aż jego licznik osiągnie 0.
4. Wybierany jest kolejny proces.
5. W przypadku braku procesów w kolejce, wykonywany jest proces idle.

Algorytm c.d.

Wszystkie procesy w kolejce mają licznik 0

- Liczniki **wszystkich** procesów są aktualizowane:

$$licznik := licznik/2 + NICE\_TO\_TICKS(prio)$$

- Algorytm wykonuje się dalej normalnie (początek nowej epoki).

Algorytm c.d.

Wszystkie procesy w kolejce mają licznik 0

- Liczniki **wszystkich** procesów są aktualizowane:

$$licznik := licznik/2 + NICE\_TO\_TICKS(prio)$$

- Algorytm wykonuje się dalej normalnie (początek nowej epoki).

Cechy algorytmu

- Procesy interaktywne są uprzywilejowane, bo nie wykorzystują całego przydzielonego czasu w epoce.
- Wybieranie kolejnego procesu, jak również aktualizacja liczników, wykonywane są w czasie liniowym.

Zalety

- Poprawność

Zalety

- Poprawność
- Prostota

Zalety

- Poprawność
- Prostota
- Względnie dobra wydajność dla niewielkiej liczby procesów.

Wady

- Słaba wydajność.

Wady

- Słaba wydajność.
- Słaba skalowalność.

Wady

- Słaba wydajność.
- Słaba skalowalność.
- Duże średnie kwanty czasu.

Wady

- Słaba wydajność.
- Słaba skalowalność.
- Duże średnie kwanty czasu.
- Niedoskonała obsługa procesów zależnych od wejścia-wyjścia.

Wprowadzenie i historia zmian

- Wprowadzony w wersji jądra 2.6.8 (rok 2002)
- Autor: Ingo Molnar
- Idea podobna do oryginalnego schedulera.
- O wiele efektywniejsza i bardziej rozbudowana implementacja.
- Wsparcie dla SMP wbudowane w logikę schedulera.

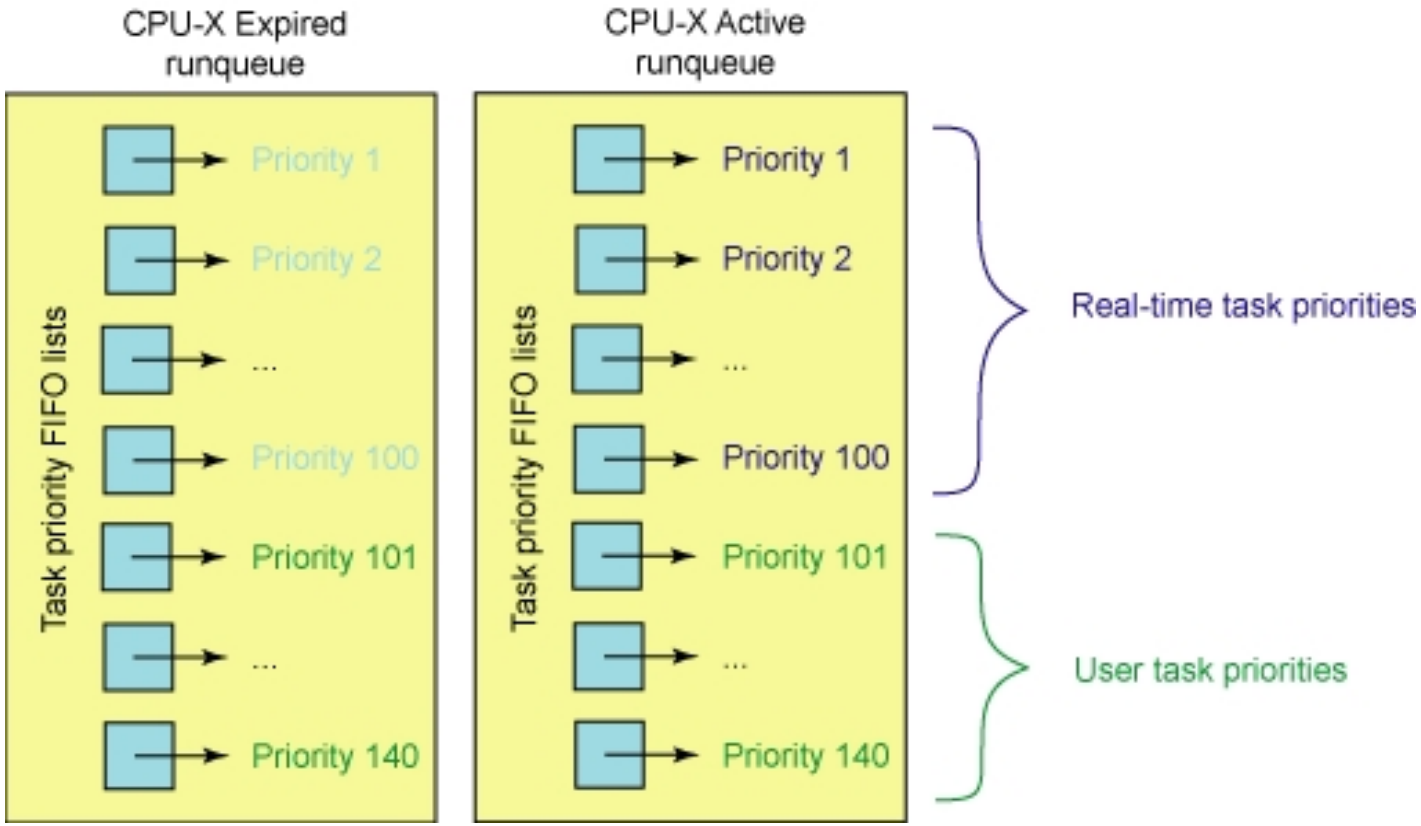
Struktury danych

Kolejka procesów gotowych

- Dwie kolejki dla każdego procesora: Aktywna i Przedawniona (*Active* i *Expired*)
- Składa się tablicy list procesów oraz z bitmapy określającej, czy kolejka dla danego indeksu jest pusta.
- Każda lista w tablicy odpowiada jednemu priorytetowi.
- Listy działają na zasadzie FIFO.
- Dla każdego procesu utrzymujemy aktualną wartość

$$sleep\_avg = (czas\_oczekiwania - czas\_dzialania)$$

Kolejki procesów w Schedulerze O(1)



Algorytm

1. Kwant czasu przydzielony danemu procesowi jest wprost proporcjonalny do jego priorytetu.
   Priorytet 0 tłumaczy się na 100 ms.
2. Wykonywane są procesy z niepustej listy o najwyższym priorytecie.
3. Dla procesów z pojedynczej listy używana jest strategia RR.
4. Proces, który wykorzystał swój kwant czasu, przenoszony jest do odpowiedniej listy w kolejce *Expired*, jednocześnie wyliczany jest nowy kwant czasu dla procesu.
5. W przypadku braku procesów w kolejce *Active*, kolejki zamieniane są miejscami.

Algorytm c.d.

Heurystyka interaktywności

- Scheduler ocenia interaktywność na podstawie wartości *sleep_avg*
- Dla procesów o dostatecznie wysokim *sleep_avg* scheduler zmniejsza priorytet.
- Bonus do priorytetu może wynieść co najwyżej 5.

Algorytm c.d.

Heurystyka interaktywności

- Scheduler ocenia interaktywność na podstawie wartości *sleep_avg*
- Dla procesów o dostatecznie wysokim *sleep_avg* scheduler zmniejsza priorytet.
- Bonus do priorytetu może wynieść co najwyżej 5.

Load balancing

- Każda kolejka zawiera wartość *cpu_load*
- Równoważenie jest wykonywane co 200ms, oraz co 1ms w trakcie wykonywania procesu idle.

Zalety

- Wydajność

Zalety

- Wydajność
- Dobra skalowalność

Zalety

- Wydajność
- Dobra skalowalność
- Obsługa SMP

Zalety

- Wydajność
- Dobra skalowalność
- Obsługa SMP
- Dobra interaktywność w odpowiednich warunkach.

Wady

- Większa złożoność w stosunku do poprzedniego schedulera.

Wady

- Większa złożoność w stosunku do poprzedniego schedulera.
- Niedoskonała heurystyka wykrywania interaktywności.

Problemy z O(1)

Problemy z O(1)

W pewnych warunkach O(1) przyznawał za mało czasu procesora procesom nastawionym na wejście-wyjście

Problemy z O(1)

W pewnych warunkach O(1) przyznawał za mało czasu procesora procesom nastawionym na wejście-wyjście, a co za tym idzie m.in.:

Problemy z O(1)

W pewnych warunkach O(1) przyznawał za mało czasu procesora procesom nastawionym na wejście-wyjście, a co za tym idzie m.in.:

- Czas reakcji na akcje użytkownika był wydłużony

Problemy z O(1)

W pewnych warunkach O(1) przyznawał za mało czasu procesora procesom nastawionym na wejście-wyjście, a co za tym idzie m.in.:

- Czas reakcji na akcje użytkownika był wydłużony
- Odtwarzacze muzyki często ’zacinały się’ i ’przeskakiwały’

Problemy z O(1)

W pewnych warunkach O(1) przyznawał za mało czasu procesora procesom nastawionym na wejście-wyjście, a co za tym idzie m.in.:

- Czas reakcji na akcje użytkownika był wydłużony
- Odtwarzacze muzyki często ’zacinały się’ i ’przeskakiwały’

Sytuacje takie zdarzały się na tyle często, że uznano to za poważny problem.

Próby naprawy O(1)

- Oczywiście podjęto wiele prób naprawienia istniejących błędów

Próby naprawy O(1)

- Oczywiście podjęto wiele prób naprawienia istniejących błędów
- Zmiany ograniczały się głównie do algorytmów szacujących stopień interaktywności procesów

Próby naprawy O(1)

- Oczywiście podjęto wiele prób naprawienia istniejących błędów
- Zmiany ograniczały się głównie do algorytmów szacujących stopień interaktywności procesów
- Wiele problemów faktycznie udało się rozwiązać. Niestety zastosowane rozwiązania generowały nowe błędy

Próby naprawy O(1)

- Oczywiście podjęto wiele prób naprawienia istniejących błędów
- Zmiany ograniczały się głównie do algorytmów szacujących stopień interaktywności procesów
- Wiele problemów faktycznie udało się rozwiązać. Niestety zastosowane rozwiązania generowały nowe błędy
- Okazało się, że konieczna jest całkowita zmiana podejścia

RSDL

RSDL

- Rotating Staircase DeadLine scheduler

RSDL

- Rotating Staircase DeadLine scheduler
- Twórca: Con Kolivas

RSDL

- Rotating Staircase DeadLine scheduler
- Twórca: Con Kolivas
- Główna idea: sprawiedliwość

Opis koncepcji

- Najważniejszym wnioskiem wyciągniętym przez Kolivasa było to, że algorytmy próbujące rozpoznawać procesy interaktywne generują więcej problemów, niż rozwiązują.

Opis koncepcji

- Najważniejszym wnioskiem wyciągniętym przez Kolivasa było to, że algorytmy próbujące rozpoznawać procesy interaktywne generują więcej problemów, niż rozwiązują.
- W związku z tym postanowił w ogóle ich nie stosować.

Opis koncepcji

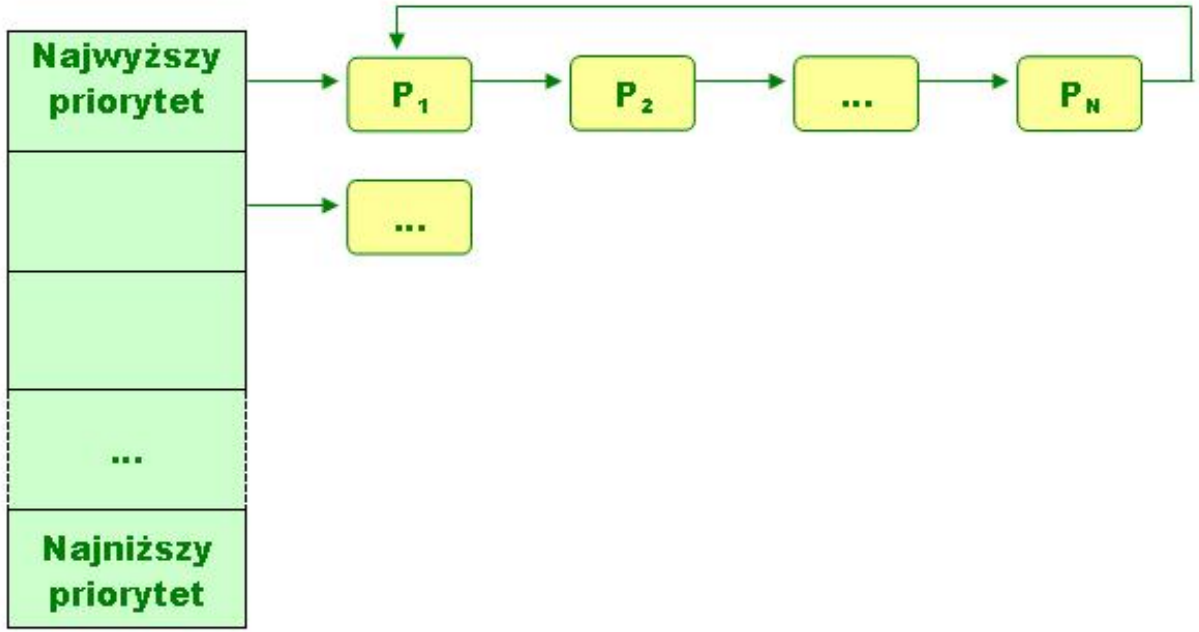
- Najważniejszym wnioskiem wyciągniętym przez Kolivasa było to, że algorytmy próbujące rozpoznawać procesy interaktywne generują więcej problemów, niż rozwiązują.
- W związku z tym postanowił w ogóle ich nie stosować.
- W RSDL wszystkie procesy traktowane są tak samo - dostają tyle samo czasu niezaleznie od tego, czy są nastawione obliczeniowo, czy też na wejście-wyjście.

Opis koncepcji

- Najważniejszym wnioskiem wyciągniętym przez Kolivasa było to, że algorytmy próbujące rozpoznawać procesy interaktywne generują więcej problemów, niż rozwiązują.
- W związku z tym postanowił w ogóle ich nie stosować.
- W RSDL wszystkie procesy traktowane są tak samo - dostają tyle samo czasu niezaleznie od tego, czy są nastawione obliczeniowo, czy też na wejście-wyjście.
- Scheduler ten w ogóle nie wnika w to, z jakiego typu procesem ma do czynienia.

Kolejka procesów

Podobnie jak w O(1) - tablica indeksowana poziomem priorytetu (osobna dla każdego procesora):



Przydział czasu

- Proces początkowo znajduje się na poziomie kolejki odpowiadającym jego priorytetowi statycznemu

Przydział czasu

- Proces początkowo znajduje się na poziomie kolejki odpowiadającym jego priorytetowi statycznemu
- Każdy proces ma limit czasu do wykorzystania na danym poziomie

Przydział czasu

- Proces początkowo znajduje się na poziomie kolejki odpowiadającym jego priorytetowi statycznemu
- Każdy proces ma limit czasu do wykorzystania na danym poziomie
- Kwanty czasu przydzielane są procesom na najwyższym poziomie w kolejce (wg strartegii RR)

Przydział czasu

- Proces początkowo znajduje się na poziomie kolejki odpowiadającym jego priorytetowi statycznemu
- Każdy proces ma limit czasu do wykorzystania na danym poziomie
- Kwanty czasu przydzielane są procesom na najwyższym poziomie w kolejce (wg strartegii RR)
- Po wykorzystaniu limitu na danym poziomie proces przenoszony jest na niższy poziom

Przydział czasu

- Proces początkowo znajduje się na poziomie kolejki odpowiadającym jego priorytetowi statycznemu
- Każdy proces ma limit czasu do wykorzystania na danym poziomie
- Kwanty czasu przydzielane są procesom na najwyższym poziomie w kolejce (wg strartegii RR)
- Po wykorzystaniu limitu na danym poziomie proces przenoszony jest na niższy poziom
- W rezultacie każdy proces wykonuje się na poziomie równym jego priorytetowi statycznemu i niższych

Przydział czasu

- Proces początkowo znajduje się na poziomie kolejki odpowiadającym jego priorytetowi statycznemu
- Każdy proces ma limit czasu do wykorzystania na danym poziomie
- Kwanty czasu przydzielane są procesom na najwyższym poziomie w kolejce (wg strartegii RR)
- Po wykorzystaniu limitu na danym poziomie proces przenoszony jest na niższy poziom
- W rezultacie każdy proces wykonuje się na poziomie równym jego priorytetowi statycznemu i niższych
- Im dłużej proces działa, z tym większą liczbą procesów musi konkurować

Przydział czasu – cd.

- Ponadto każdy poziom kolejki ma swój własny limit

Przydział czasu – cd.

- Ponadto każdy poziom kolejki ma swój własny limit
- Jest to maksymalna ilość czasu, przez jaki łącznie mogą się wykonywać procesy na tym poziomie

Przydział czasu – cd.

- Ponadto każdy poziom kolejki ma swój własny limit
- Jest to maksymalna ilość czasu, przez jaki łącznie mogą się wykonywać procesy na tym poziomie
- Po jego wyczerpaniu nastepuje **mała rotacja**

Mała rotacja

- Polega ona na tym, że wszystkie procesy znajdujące się na najwyższym poziomie kolejki przenoszone są o poziom niżej

Mała rotacja

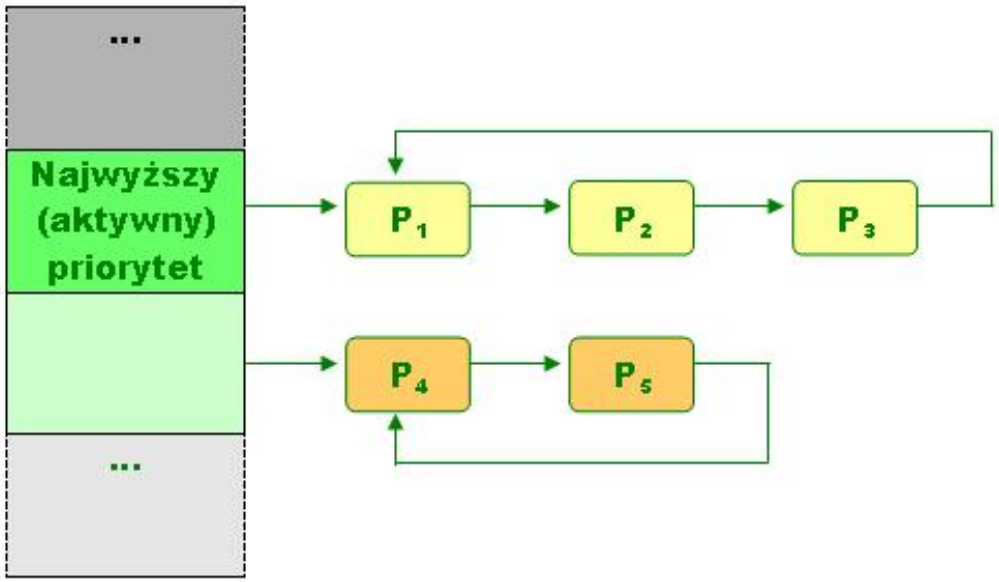
- Polega ona na tym, że wszystkie procesy znajdujące się na najwyższym poziomie kolejki przenoszone są o poziom niżej
- Dzieje się tak niezależnie od tego, czy procesy te wykorzystały swoje indywidualne limity

Mała rotacja

- Polega ona na tym, że wszystkie procesy znajdujące się na najwyższym poziomie kolejki przenoszone są o poziom niżej
- Dzieje się tak niezależnie od tego, czy procesy te wykorzystały swoje indywidualne limity
- Zastosowanie tego mechanizmu eliminiuje możliwość zagłodzenia procesów o niższych priorytetach.

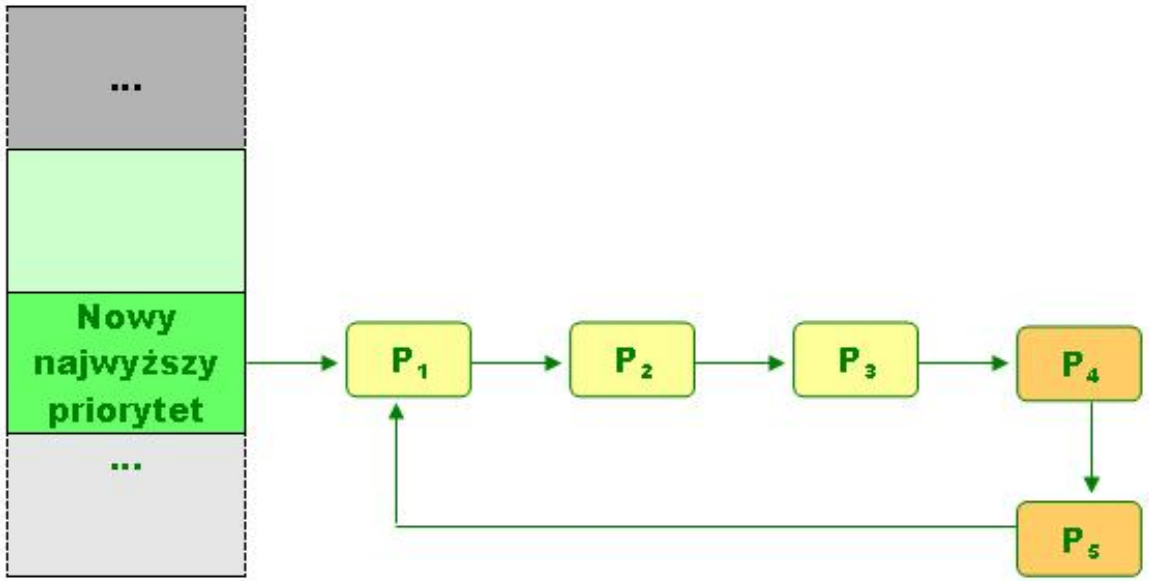
Mała rotacja

Przykładowa sytuacja przed nastąpieniem rotacji:



Mała rotacja – cd.

Skutki rotacji:



Kolejka nieaktywna

- Bliźniacza kolejka dla procesów nieaktywnych

Kolejka nieaktywna

- Bliźniacza kolejka dla procesów nieaktywnych
- Procesy są do niej przenoszone po wykorzystaniu limitu na ostatnim poziomie kolejki aktywnej

Kolejka nieaktywna

- Bliźniacza kolejka dla procesów nieaktywnych
- Procesy są do niej przenoszone po wykorzystaniu limitu na ostatnim poziomie kolejki aktywnej
- Po dezaktywizacji wszystkich procesów następuje zamiana kolejki aktywnej i nieaktywnej, czyli tzw. **duża rotacja**

Duża rotacja

- Czasami działania związane z dużą rotacją nie ograniczają się tylko do zamiany kolejek

Duża rotacja

- Czasami działania związane z dużą rotacją nie ograniczają się tylko do zamiany kolejek
- Dzieje się tak, gdy kolejka aktywna opróżni się w wyniku nastąpienia małej rotacji na jej ostatnim poziomie

Duża rotacja

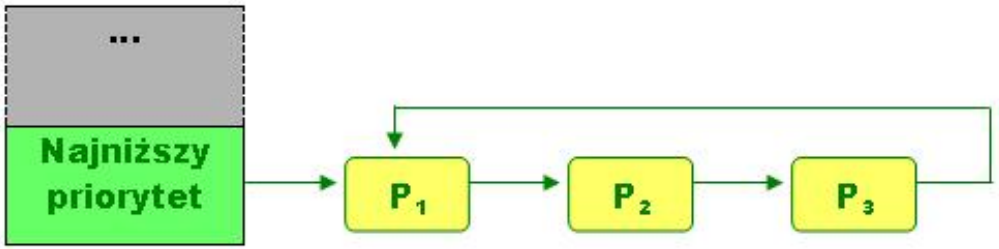
- Czasami działania związane z dużą rotacją nie ograniczają się tylko do zamiany kolejek
- Dzieje się tak, gdy kolejka aktywna opróżni się w wyniku nastąpienia małej rotacji na jej ostatnim poziomie
- Wszystkie procesy, które jeszcze działały, przenoszone są wtedy na najwyższy niepusty poziom kolejki nieaktywnej

Duża rotacja

- Czasami działania związane z dużą rotacją nie ograniczają się tylko do zamiany kolejek
- Dzieje się tak, gdy kolejka aktywna opróżni się w wyniku nastąpienia małej rotacji na jej ostatnim poziomie
- Wszystkie procesy, które jeszcze działały, przenoszone są wtedy na najwyższy niepusty poziom kolejki nieaktywnej
- Może się zatem okazać, że niektóre procesy trafią na wyższy poziom kolejki, niż by to wynikało z ich priorytetu

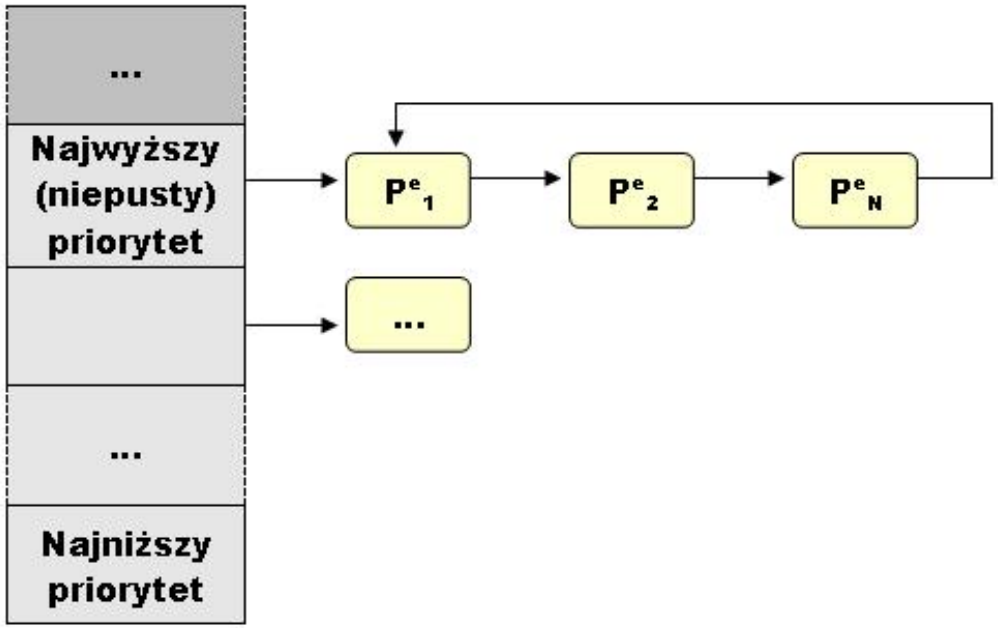
Duża rotacja

Przykładowa kolejka aktywna przed nastapieniem dużej rotacji:



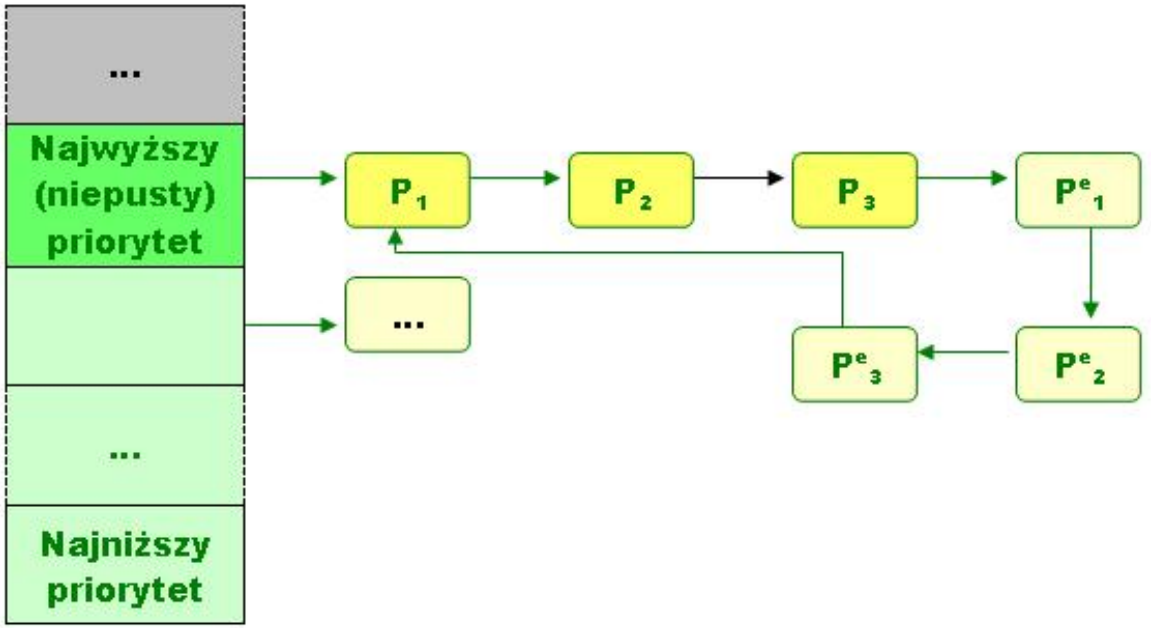
Duża rotacja – kolejka nieaktywna

Przykładowa kolejka nieaktywna przed nastapieniem dużej rotacji:



Duża rotacja – cd.

Nowa kolejka aktywna:



Duża rotacja – cd.

Nowa kolejka nieaktywna:

Informacje przechowywane przez kolejkę

- Numer bieżącej epoki, czyli okresu między kolejnymi dużymi rotacjami (*rq->prio_rotation*)

Informacje przechowywane przez kolejkę

- Numer bieżącej epoki, czyli okresu między kolejnymi dużymi rotacjami (*rq->prio_rotation*)
- Limity dla poszczególnych poziomów (*rq->prio_quota[]*)

Informacje przechowywane przez kolejkę

- Numer bieżącej epoki, czyli okresu między kolejnymi dużymi rotacjami (*rq->prio_rotation*)
- Limity dla poszczególnych poziomów (*rq->prio_quota[]*)
- Informacje o tym, jakie priorytety mają procesy znajdujące się w kolejce (*static_bitmap, dyn_bitmap, prio_queued[]*)

Informacje przechowywane przez proces

- Ilość przysługującego mu czasu na poziomie, na którym aktualnie się znajduje(*p->time_slice*)

Informacje przechowywane przez proces

- Ilość przysługującego mu czasu na poziomie, na którym aktualnie się znajduje(*p->time_slice*)
- Stały limit czasu do wykorzystana na każdym poziomie (*p->quota*)

Informacje przechowywane przez proces

- Ilość przysługującego mu czasu na poziomie, na którym aktualnie się znajduje(*p->time_slice*)
- Stały limit czasu do wykorzystana na każdym poziomie (*p->quota*)
- Nr epoki, w której ostatnio się wykonywał (*p->rotation*)

Informacje przechowywane przez proces

- Ilość przysługującego mu czasu na poziomie, na którym aktualnie się znajduje(*p->time_slice*)
- Stały limit czasu do wykorzystana na każdym poziomie (*p->quota*)
- Nr epoki, w której ostatnio się wykonywał (*p->rotation*)
- Bitmapa poziomów, na których był w danej epoce (*p->bitmap*)

Informacje przechowywane przez proces

- Ilość przysługującego mu czasu na poziomie, na którym aktualnie się znajduje(*p->time_slice*)
- Stały limit czasu do wykorzystana na każdym poziomie (*p->quota*)
- Nr epoki, w której ostatnio się wykonywał (*p->rotation*)
- Bitmapa poziomów, na których był w danej epoce (*p->bitmap*)
- priorytet statyczny i dynamiczny (*p->static_prio, p->prio*)

Wstawianie procesu do kolejki

Jeśli proces nie wykonywał się jeszcze w trakcie aktualnej epoki:

Wstawianie procesu do kolejki

Jeśli proces nie wykonywał się jeszcze w trakcie aktualnej epoki:

- Wartość jego priorytetu tymczasowego staje się równa jego priorytetowi statycznemu

Wstawianie procesu do kolejki

Jeśli proces nie wykonywał się jeszcze w trakcie aktualnej epoki:

- Wartość jego priorytetu tymczasowego staje się równa jego priorytetowi statycznemu
- Pole p->time_slice ustawiane jest na wartość p->quota

Wstawianie procesu do kolejki

Jeśli proces nie wykonywał się jeszcze w trakcie aktualnej epoki:

- Wartość jego priorytetu tymczasowego staje się równa jego priorytetowi statycznemu
- Pole p->time_slice ustawiane jest na wartość p->quota
- Wartość ta dodawana jest również do rq->prio_quota[p->static_prio]

Wstawianie procesu do kolejki

Jeśli proces nie wykonywał się jeszcze w trakcie aktualnej epoki:

- Wartość jego priorytetu tymczasowego staje się równa jego priorytetowi statycznemu
- Pole p->time_slice ustawiane jest na wartość p->quota
- Wartość ta dodawana jest również do rq->prio_quota[p->static_prio]
- W p->bitmap ustawiany jest bit odpowiadający wartości p->static_prio

Wstawianie procesu do kolejki

Jeśli proces nie wykonywał się jeszcze w trakcie aktualnej epoki:

- Wartość jego priorytetu tymczasowego staje się równa jego priorytetowi statycznemu
- Pole p->time_slice ustawiane jest na wartość p->quota
- Wartość ta dodawana jest również do rq->prio_quota[p->static_prio]
- W p->bitmap ustawiany jest bit odpowiadający wartości p->static_prio
- Proces wstawiany jest do kolejki na poziomie p->static_prio

Wstawianie procesu do kolejki

Jeśli proces wykonywał się i nie wykorzystał limitu na danym poziomie (p->time_slice > 0) oraz dany poziom również ma jeszcze limit do wykorzystania (rq->prio_quota[p->prio] > 0):

Wstawianie procesu do kolejki

Jeśli proces wykonywał się i nie wykorzystał limitu na danym poziomie (p->time_slice > 0) oraz dany poziom również ma jeszcze limit do wykorzystania (rq->prio_quota[p->prio] > 0):

- Proces wstawiany jest do kolejki na odpowiednim poziomie

Wstawianie procesu do kolejki

Jeśli proces wykonywał się i nie wykorzystał limitu na danym poziomie (p->time_slice > 0) oraz dany poziom również ma jeszcze limit do wykorzystania (rq->prio_quota[p->prio] > 0):

- Proces wstawiany jest do kolejki na odpowiednim poziomie
- Do limitów procesu i poziomu nie są już dodawane żadne nowe wartości

Wstawianie procesu do kolejki

Jeśli proces wykonywał się i wykorzystał limit na danym poziomie (p->time_slice = 0) lub dany poziom nie ma już limitu do wykorzystania (rq->prio_quota[p->prio] = 0):

Wstawianie procesu do kolejki

Jeśli proces wykonywał się i wykorzystał limit na danym poziomie (p->time_slice = 0) lub dany poziom nie ma już limitu do wykorzystania (rq->prio_quota[p->prio] = 0):

- W p->bitmap wyszukiwany jest następny poziom, na którym proces jeszcze się nie wykonywał. Jego bit zostaje ustawiony na 1

Wstawianie procesu do kolejki

Jeśli proces wykonywał się i wykorzystał limit na danym poziomie (p->time_slice = 0) lub dany poziom nie ma już limitu do wykorzystania (rq->prio_quota[p->prio] = 0):

- W p->bitmap wyszukiwany jest następny poziom, na którym proces jeszcze się nie wykonywał. Jego bit zostaje ustawiony na 1
- Proces dostaje nowy pełny limit p->time_slice, który jest dodawany także do limitu nowego poziomu

Wstawianie procesu do kolejki

Jeśli proces wykonywał się i wykorzystał limit na danym poziomie (p->time_slice = 0) lub dany poziom nie ma już limitu do wykorzystania (rq->prio_quota[p->prio] = 0):

- W p->bitmap wyszukiwany jest następny poziom, na którym proces jeszcze się nie wykonywał. Jego bit zostaje ustawiony na 1
- Proces dostaje nowy pełny limit p->time_slice, który jest dodawany także do limitu nowego poziomu
- Proces wstawiany jest na nowy poziom, a jego priorytet zostaje zmniejszony

Wstawianie procesu do kolejki

Jeśli proces wykorzystał już limit na ostatnim poziomie kolejki:

Wstawianie procesu do kolejki

Jeśli proces wykorzystał już limit na ostatnim poziomie kolejki:

- Jego bitmapa jest zerowana

Wstawianie procesu do kolejki

Jeśli proces wykorzystał już limit na ostatnim poziomie kolejki:

- Jego bitmapa jest zerowana
- Proces wstawiany jest do kolejki nieaktywnej na poziomie odpowiadającym jego static_prio

Wstawianie procesu do kolejki

Jeśli proces wykorzystał już limit na ostatnim poziomie kolejki:

- Jego bitmapa jest zerowana
- Proces wstawiany jest do kolejki nieaktywnej na poziomie odpowiadającym jego static_prio
- Wszelkie limity przypisywane są dopiero w momencie rozpatrywania procesu przez scheduler

Scheduler_tick

- Zmniejszenie wartości *time_slice* aktualnie wykonywanego procesu

Scheduler_tick

- Zmniejszenie wartości *time_slice* aktualnie wykonywanego procesu
- Zmniejszenie limitu poziomu, na którym znajduje się działający proces

RR_INTERVAL

- Główny parametr, od którego zależy to, jak długo wykonują się procesy, to RR_INTERVAL (standardowo – 6 ms).

RR_INTERVAL

- Główny parametr, od którego zależy to, jak długo wykonują się procesy, to RR_INTERVAL (standardowo – 6 ms).
- Wszystkie procesy dostają limity oparte właśnie na tej wartości (są jej równe dla nice od 0 do 19, progresywnie wyższe dla nice od -1 do -20)

Maksymalny czas oczekiwania

- Istnieje możliwość dokładnego obliczenia maksymalnego okresu oczekiwania procesu na czas procesora

Maksymalny czas oczekiwania

- Istnieje możliwość dokładnego obliczenia maksymalnego okresu oczekiwania procesu na czas procesora
- Zależy on tylko od liczby procesów w kolejce i ich priorytetów

Maksymalny czas oczekiwania

- Istnieje możliwość dokładnego obliczenia maksymalnego okresu oczekiwania procesu na czas procesora
- Zależy on tylko od liczby procesów w kolejce i ich priorytetów
- Jest on równy długości okresu od wstawienia rozpatrywanego procesu do kolejki nieaktywnej do zakończenia bieżącej epoki *plus* okres, przez który już w nowej epoce wykonywać się będą tylko procesy o wyższym priorytecie

Maksymalny czas oczekiwania - wzór

Wzór

$$(akt\_poziom + roznica\_nice) \cdot x \cdot RR\_INTERVAL$$

Gdzie:

- *akt_poziom* to poziom, na którym aktualnie znajduje się wskazany proces. Jeśli znajduje się on w kolejce nieaktywnej, akt_poziom = 0

Maksymalny czas oczekiwania - wzór

Wzór

$$(akt\_poziom + roznica\_nice) \cdot x \cdot RR\_INTERVAL$$

Gdzie:

- *akt_poziom* to poziom, na którym aktualnie znajduje się wskazany proces. Jeśli znajduje się on w kolejce nieaktywnej, akt_poziom = 0
- *roznica_nice* to różnica między wartością nice wskazanego procesu oraz procesu, dla którego obliczany jest czas oczekiwania. Jeśli wartość ta jest mniejsza od 0, to pomijamy ten składnik

Maksymalny czas oczekiwania - wzór

Wzór

$$(akt\_poziom + roznica\_nice) \cdot x \cdot RR\_INTERVAL$$

Gdzie:

- *akt_poziom* to poziom, na którym aktualnie znajduje się wskazany proces. Jeśli znajduje się on w kolejce nieaktywnej, akt_poziom = 0
- *roznica_nice* to różnica między wartością nice wskazanego procesu oraz procesu, dla którego obliczany jest czas oczekiwania. Jeśli wartość ta jest mniejsza od 0, to pomijamy ten składnik
- *x* to wartość, przez którą mnożony jest RR_INTERVAL przy obliczaniu limitów dla procesu (dla nice od 0 do 19 x = 1, dla nice < 0 x > 1)

Maksymalny czas oczekiwania - wzór cd.

Jest to oczywiście jedynie pojedynczy składnik całego wzoru na maksymalny okres oczekiwania. Wartość tę należy obliczyć dla każdego procesu znajdującego się w kolejce (aktywnej lub nieaktywnej), po czym – zsumować.

Przykładowe obliczenie

- Maksymalny okres oczekiwania dla procesu o nice = 10, który właśnie trafił do kolejki nieaktywnej, jeśli równolegle wykonuje się proces o nice = 0

Przykładowe obliczenie

- Maksymalny okres oczekiwania dla procesu o nice = 10, który właśnie trafił do kolejki nieaktywnej, jeśli równolegle wykonuje się proces o nice = 0
- $(19 + 9) \cdot 1 \cdot \mathit{RR\_INTERVAL} = 168\mathit{ms}$

Interaktywność

Jak udało się zachować ścisłą sprawiedliwość i utrzymać przy tym wysoką interaktywność?

Interaktywność

Jak udało się zachować ścisłą sprawiedliwość i utrzymać przy tym wysoką interaktywność?

- Procesy interaktywne są naturalnie uprzywilejowane, gdyż często śpią i w związku z tym wolniej wykorzystują przysługujący im czas

Interaktywność

Jak udało się zachować ścisłą sprawiedliwość i utrzymać przy tym wysoką interaktywność?

- Procesy interaktywne są naturalnie uprzywilejowane, gdyż często śpią i w związku z tym wolniej wykorzystują przysługujący im czas
- Mimo zasypiania zachowują swoje limity

Interaktywność

Jak udało się zachować ścisłą sprawiedliwość i utrzymać przy tym wysoką interaktywność?

- Procesy interaktywne są naturalnie uprzywilejowane, gdyż często śpią i w związku z tym wolniej wykorzystują przysługujący im czas
- Mimo zasypiania zachowują swoje limity
- Z reguły budzą się na poziomie równym lub wyższym od tego, na którym aktualnie wykonują się procesy, w związku z czym szybko je wywłaszczają

CFS

- Completely Fair Scheduler

CFS

- Completely Fair Scheduler
- Twórca: Ingo Molnar

CFS

- Completely Fair Scheduler
- Twórca: Ingo Molnar
- Zmieniono ok. 70% kodu

CFS

- Completely Fair Scheduler
- Twórca: Ingo Molnar
- Zmieniono ok. 70% kodu
- Niektóre zmiany są naprawdę przełomowe

CFS

- Completely Fair Scheduler
- Twórca: Ingo Molnar
- Zmieniono ok. 70% kodu
- Niektóre zmiany są naprawdę przełomowe
- Główna idea:

CFS

- Completely Fair Scheduler
- Twórca: Ingo Molnar
- Zmieniono ok. 70% kodu
- Niektóre zmiany są naprawdę przełomowe
- Główna idea: sprawiedliwość (nie jest to oczywiście przypadkowa zbieżność)

CFS

- Completely Fair Scheduler
- Twórca: Ingo Molnar
- Zmieniono ok. 70% kodu
- Niektóre zmiany są naprawdę przełomowe
- Główna idea: sprawiedliwość (nie jest to oczywiście przypadkowa zbieżność)
- Sprawiedliwość sprawiedliwości nierówna – tutaj istotny jest również czas, przez jaki proces śpi

Idealny wielozadaniowy procesor

- Zdolny wykonywać wiele zadań **jednocześnie**

Idealny wielozadaniowy procesor

- Zdolny wykonywać wiele zadań **jednocześnie**
- Każde zadanie jest wykonywane w takim samym tempie...

Idealny wielozadaniowy procesor

- Zdolny wykonywać wiele zadań **jednocześnie**
- Każde zadanie jest wykonywane w takim samym tempie...
- ...równym jego mocy podzielonej przez liczbę działających procesów

Idealny wielozadaniowy procesor

- Zdolny wykonywać wiele zadań **jednocześnie**
- Każde zadanie jest wykonywane w takim samym tempie...
- ...równym jego mocy podzielonej przez liczbę działających procesów
- Oczywiście nie istnieje:-)

Idealny wielozadaniowy procesor

- Zdolny wykonywać wiele zadań **jednocześnie**
- Każde zadanie jest wykonywane w takim samym tempie...
- ...równym jego mocy podzielonej przez liczbę działających procesów
- Oczywiście nie istnieje:-)

Ingo Molnar:

"80% of CFS's design can be summed up in a single sentence: CFS basically models an »ideal, precise multi-tasking CPU« on real hardware."

Wait_runtime

- Wartość przechowywana przez proces

Wait_runtime

- Wartość przechowywana przez proces
- Ilość czasu ’należna’ danemu procesowi

Wait_runtime

- Wartość przechowywana przez proces
- Ilość czasu 'należna' danemu procesowi
- Do wykonania wybierany jest zawsze proces o najwyższym *p->wait_runtime*

Kolejka procesów

- W postaci drzewa czerwono-czarnego

Kolejka procesów

- W postaci drzewa czerwono-czarnego
- Brak kolejki nieaktywnej

Kolejka procesów

- W postaci drzewa czerwono-czarnego
- Brak kolejki nieaktywnej
- Klucz: **rq->fair_clock - p->wait_runtime**

Kolejka procesów

- W postaci drzewa czerwono-czarnego
- Brak kolejki nieaktywnej
- Klucz: **rq->fair_clock - p->wait_runtime**
- *rq->fair_clock* to globalny dla całej kolejki zegar – prezentuje czas, przez który sprawiedliwie powinno się wykonywać aktywne zadanie

Kolejka procesów

- W postaci drzewa czerwono-czarnego
- Brak kolejki nieaktywnej
- Klucz: **rq->fair_clock - p->wait_runtime**
- *rq->fair_clock* to globalny dla całej kolejki zegar – prezentuje czas, przez który sprawiedliwie powinno się wykonywać aktywne zadanie
- Uporządkowana rosnąco

Kolejka procesów

- W postaci drzewa czerwono-czarnego
- Brak kolejki nieaktywnej
- Klucz: **rq->fair_clock - p->wait_runtime**
- *rq->fair_clock* to globalny dla całej kolejki zegar – prezentuje czas, przez który sprawiedliwie powinno się wykonywać aktywne zadanie
- Uporządkowana rosnąco
- Jedna na każdy procesor

Naliczanie nanosekundowe

- Czas działania obliczany z dokładnością do nanosekund (brak bezpośredniego odniesienia do jiffies czy HZ)

Naliczanie nanosekundowe

- Czas działania obliczany z dokładnością do nanosekund (brak bezpośredniego odniesienia do jiffies czy HZ)
- Brak tradycyjnych *time slices* – przydział czasu dokonywany jest dynamicznie, nie statycznie. Nie ma tu stałej długości kwantów, które procesy ’zużywają’

Naliczanie nanosekundowe

- Czas działania obliczany z dokładnością do nanosekund (brak bezpośredniego odniesienia do jiffies czy HZ)
- Brak tradycyjnych *time slices* – przydział czasu dokonywany jest dynamicznie, nie statycznie. Nie ma tu stałej długości kwantów, które procesy ’zużywają’
- Granulacja czasu – warunkuje tempo przełączania procesów

Naliczanie nanosekundowe

- Czas działania obliczany z dokładnością do nanosekund (brak bezpośredniego odniesienia do jiffies czy HZ)
- Brak tradycyjnych *time slices* – przydział czasu dokonywany jest dynamicznie, nie statycznie. Nie ma tu stałej długości kwantów, które procesy 'zużywają'
- Granulacja czasu – warunkuje tempo przełączania procesów
- Virtual slices – zmniejszanie granulacji wraz ze wzrostem obciążenia. Dzięki temu procesy krócej oczekują na procesor

Wstawianie procesu do kolejki

- Zapamiętanie aktualnego czasu

Wstawianie procesu do kolejki

- Zapamiętanie aktualnego czasu
- Utworzenie klucza (= rq->fair_clock)

Wstawianie procesu do kolejki

- Zapamiętanie aktualnego czasu
- Utworzenie klucza (= rq->fair_clock)
- Zwykłe wstawienie węzła do drzewa cz.-cz.

Zmiana priorytetu tymczasowego

- Związana z wartością p->wait_runtime
- Zależna od tego, ile czasu cpu dostaje dany proces

Obsługa procesu

- Wybór procesu do wykonywania: skrajnie lewy w drzewie

Obsługa procesu

- Wybór procesu do wykonywania: skrajnie lewy w drzewie
- Wraz z wykonywaniem procesu – zmniejszanie p->wait_runtime (o czas, przez który proces działał minus czas, który i tak by mu się należał)

Obsługa procesu

- Wybór procesu do wykonywania: skrajnie lewy w drzewie
- Wraz z wykonywaniem procesu – zmniejszanie p->wait_runtime (o czas, przez który proces działał minus czas, który i tak by mu się należał)
- Zmiana procesu do wykonania następuje po odpowiednim obniżeniu wartości p->wait_runtime (gdy działający proces przestanie być pierwszym w drzewie – z dokładnością do granulacji)

Obsługa procesu

- Wybór procesu do wykonywania: skrajnie lewy w drzewie
- Wraz z wykonywaniem procesu – zmniejszanie p->wait_runtime (o czas, przez który proces działał minus czas, który i tak by mu się należał)
- Zmiana procesu do wykonania następuje po odpowiednim obniżeniu wartości p->wait_runtime (gdy działający proces przestanie być pierwszym w drzewie – z dokładnością do granulacji)
- Realizacja: procedury pick_next_task i put_prev_task

Hierarchia modułów

- “Rozszerzalna hierarchia modułów schedulera”

Hierarchia modułów

- “Rozszerzalna hierarchia modułów schedulera”
- Umożliwia implementację obsługi różnego typu procesów w oddzielnych modułach

Hierarchia modułów

- “Rozszerzalna hierarchia modułów schedulera”
- Umożliwia implementację obsługi różnego typu procesów w oddzielnych modułach
- Scheduler składa się z rdzenia i modułów

Hierarchia modułów

- “Rozszerzalna hierarchia modułów schedulera”
- Umożliwia implementację obsługi różnego typu procesów w oddzielnych modułach
- Scheduler składa się z rdzenia i modułów
- Hierarchia przeglądana jest od góry

Hierarchia modułów

- “Rozszerzalna hierarchia modułów schedulera”
- Umożliwia implementację obsługi różnego typu procesów w oddzielnych modułach
- Scheduler składa się z rdzenia i modułów
- Hierarchia przeglądana jest od góry
- sched_rt.c - implementacja obsługi procesów czasu rzeczywistego

Hierarchia modułów

- “Rozszerzalna hierarchia modułów schedulera”
- Umożliwia implementację obsługi różnego typu procesów w oddzielnych modułach
- Scheduler składa się z rdzenia i modułów
- Hierarchia przeglądana jest od góry
- sched_rt.c - implementacja obsługi procesów czasu rzeczywistego
- sched_fair.c - implementacja CFS

Obsługa grup procesów

- Dodatkowe rozszerzenie

Obsługa grup procesów

- Dodatkowe rozszerzenie
- Uogólnienie idei sprawiedliwości na całe grupy procesów

Obsługa grup procesów

- Dodatkowe rozszerzenie
- Uogólnienie idei sprawiedliwości na całe grupy procesów
- Przydatna m.in. do rozdzielania czasu procesora między wielu użytkowników

Osoby mające wpływ na rozwój CFS

Osoby mające wpływ na rozwój CFS

- Con Kolivas - idea fair scheduling

Osoby mające wpływ na rozwój CFS

- Con Kolivas - idea fair scheduling
- Peter Zijlstra - virtual slices, udoskonalenie obliczeń

Osoby mające wpływ na rozwój CFS

- Con Kolivas - idea fair scheduling
- Peter Zijlstra - virtual slices, udoskonalenie obliczeń
- Srivatsa Vaddagiri - obsługa grup procesów

Osoby mające wpływ na rozwój CFS

- Con Kolivas - idea fair scheduling
- Peter Zijlstra - virtual slices, udoskonalenie obliczeń
- Srivatsa Vaddagiri - obsługa grup procesów
- Mike Galbraith - poprawa interaktywności

Osoby mające wpływ na rozwój CFS

- Con Kolivas - idea fair scheduling
- Peter Zijlstra - virtual slices, udoskonalenie obliczeń
- Srivatsa Vaddagiri - obsługa grup procesów
- Mike Galbraith - poprawa interaktywności
- Thomas Gleixner - optymalizacje matematyczne

Osoby mające wpływ na rozwój CFS

- Con Kolivas - idea fair scheduling
- Peter Zijlstra - virtual slices, udoskonalenie obliczeń
- Srivatsa Vaddagiri - obsługa grup procesów
- Mike Galbraith - poprawa interaktywności
- Thomas Gleixner - optymalizacje matematyczne
- Dmitry Adamushko - różne ulepszenia

Dyskusja o CFS

Po opublikowaniu CFS wybuchła gorąca i dość owocna dyskusja na jego temat. W jej trakcie powstało wiele propozycji zmian.
Warto tu wspomnieć o:

Dyskusja o CFS

Po opublikowaniu CFS wybuchła gorąca i dość owocna dyskusja na jego temat. W jej trakcie powstało wiele propozycji zmian.
Warto tu wspomnieć o:

- RFS - Really Fair Scheduler (Roman Zippel)

Dyskusja o CFS

Po opublikowaniu CFS wybuchła gorąca i dość owocna dyskusja na jego temat. W jej trakcie powstało wiele propozycji zmian.
Warto tu wspomnieć o:

- RFS - Really Fair Scheduler (Roman Zippel)
- RSRFS - Really Simple Really Fair Scheduler (Ingo Molnar)

Porównanie RSDL i CFS

Porównanie RSDL i CFS

- Podobna idea

Porównanie RSDL i CFS

- Podobna idea
- Zupełnie różne, w zasadzie nieporównywalne implementacje

Porównanie RSDL i CFS

- Podobna idea
- Zupełnie różne, w zasadzie nieporównywalne implementacje
- Wyniki testów porównawczych - niejednoznaczne

Porównanie RSDL i CFS

- Podobna idea
- Zupełnie różne, w zasadzie nieporównywalne implementacje
- Wyniki testów porównawczych - niejednoznaczne
- Trudno o obiektywna ocenę, który z nich jest lepszy

Publiczna dyskusja

Publiczna dyskusja

- Oczywiście do jądra włączony mógł zostać tylko jeden scheduler

Publiczna dyskusja

- Oczywiście do jądra włączony mógł zostać tylko jeden scheduler
- Długo wydawało się, że będzie to RSDL

Publiczna dyskusja

- Oczywiście do jądra włączony mógł zostać tylko jeden scheduler
- Długo wydawało się, że będzie to RSDL
- W momencie, gdy powstała konkurencja w postaci CFS, pojawiło się wiele kontrowersji w związku z wyborem schedulera

Publiczna dyskusja

- Oczywiście do jądra włączony mógł zostać tylko jeden scheduler
- Długo wydawało się, że będzie to RSDL
- W momencie, gdy powstała konkurencja w postaci CFS, pojawiło się wiele kontrowersji w związku z wyborem schedulera
- Sam Linus Torvalds, czyli osoba, która miała podjąć ostateczną decyzję, ewidentnie skłaniał się ku CFS

Wybór schedulera

- Wybrano CFS

Wybór schedulera

- Wybrano CFS
- Oficjalne włączenie do głównej linii jądra: lipiec 2007 r.

Wybór schedulera

- Wybrano CFS
- Oficjalne włączenie do głównej linii jądra: lipiec 2007 r.
- Pierwsze wydanie zawierające CFS: 2.6.23 (październik 2007 r.)

Podsumowanie trendów

Istotne aspekty nowych schedulerów to m.in.:

Podsumowanie trendów

Istotne aspekty nowych schedulerów to m.in.:

- Przede wszystkim: wspieranie interaktywności

Podsumowanie trendów

Istotne aspekty nowych schedulerów to m.in.:

- Przede wszystkim: wspieranie interaktywności
- Sprawiedliwość

Podsumowanie trendów

Istotne aspekty nowych schedulerów to m.in.:

- Przede wszystkim: wspieranie interaktywności
- Sprawiedliwość
- Względna prostota

Podsumowanie trendów

Istotne aspekty nowych schedulerów to m.in.:

- Przede wszystkim: wspieranie interaktywności
- Sprawiedliwość
- Względna prostota
- Rozszerzalność

Wprowadzenie

- Co można obiektywnie testować?

Wprowadzenie

- Co można obiektywnie testować?
- Wydajność.

Wprowadzenie

- Co można obiektywnie testować?
- Wydajność.
- Interaktywność jest kwestią odczuć, jakkolwiek można próbować tworzyć dla schedulera “trudne scenariusze”.

Wprowadzenie

- Co można obiektywnie testować?
- Wydajność.
- Interaktywność jest kwestią odczuć, jakkolwiek można próbować tworzyć dla schedulera “trudne scenariusze”.
- Sprawiedliwość (lub jej brak) wynika z konstrukcji schedulera.

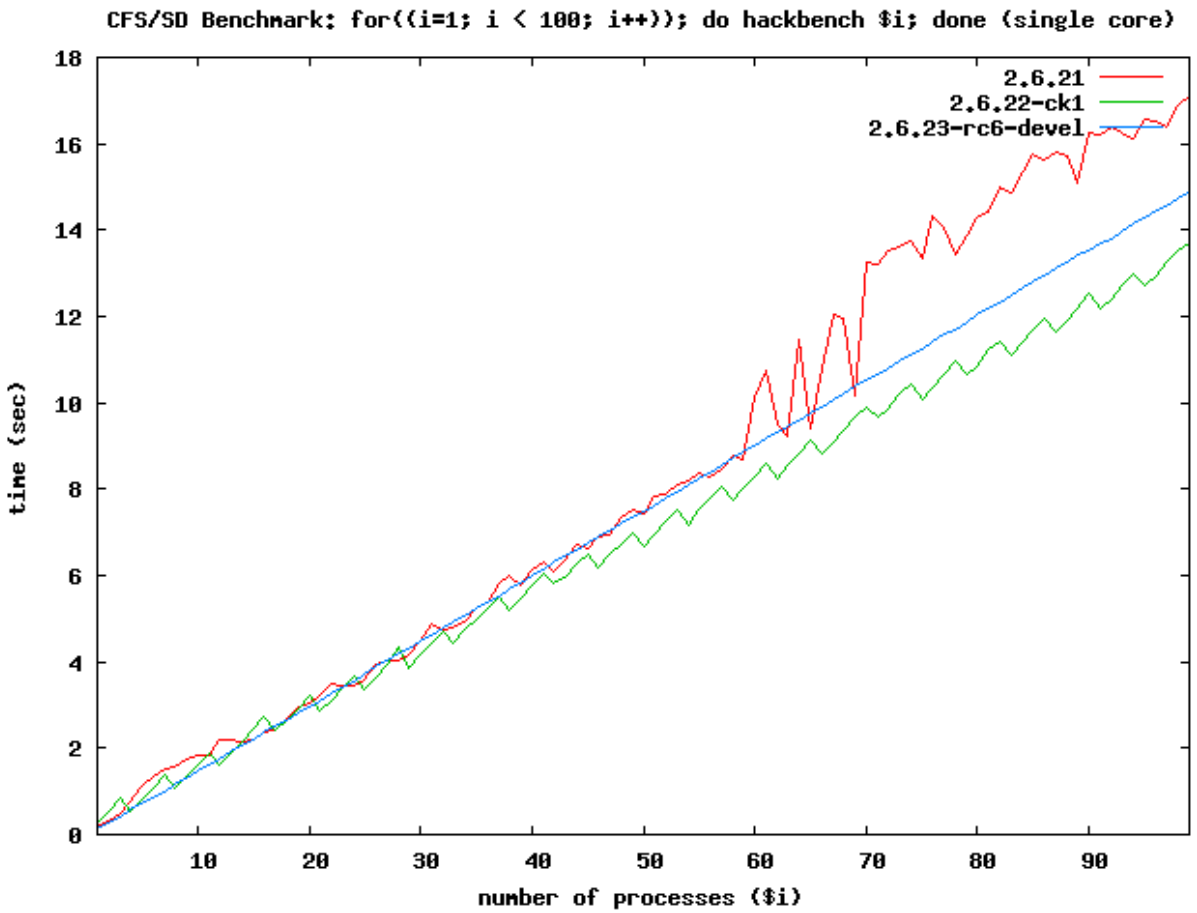
Hackbench - Test wydajności schedulera

Idea

- Tworzymy określoną liczbę grup procesów (liczba ta jest parametrem testu).
- Każda grupa składa się z równej liczby nadawców i odbiorców.
- Każdy nadawca wysyła do każdego odbiorcy komunikat - ciąg bajtów.
- Cały test to program w C, trochę ponad 200 linii kodu.

- Wynikiem testu jest czas jego wykonania w sekundach.

hackbench, O(1) vs CFS vs RSDL



LMBench

- Zestaw mikrotestów służących do testowania szerokości pasma (*bandwidth*) oraz czasu oczekiwania (*latency*) rozmaitych operacji systemowych.

LMBench

- Zestaw mikrotestów służących do testowania szerokości pasma (*bandwidth*) oraz czasu oczekiwania (*latency*) rozmaitych operacji systemowych.
- Działa dla różnych systemów operacyjnych opartych na UNIXie.

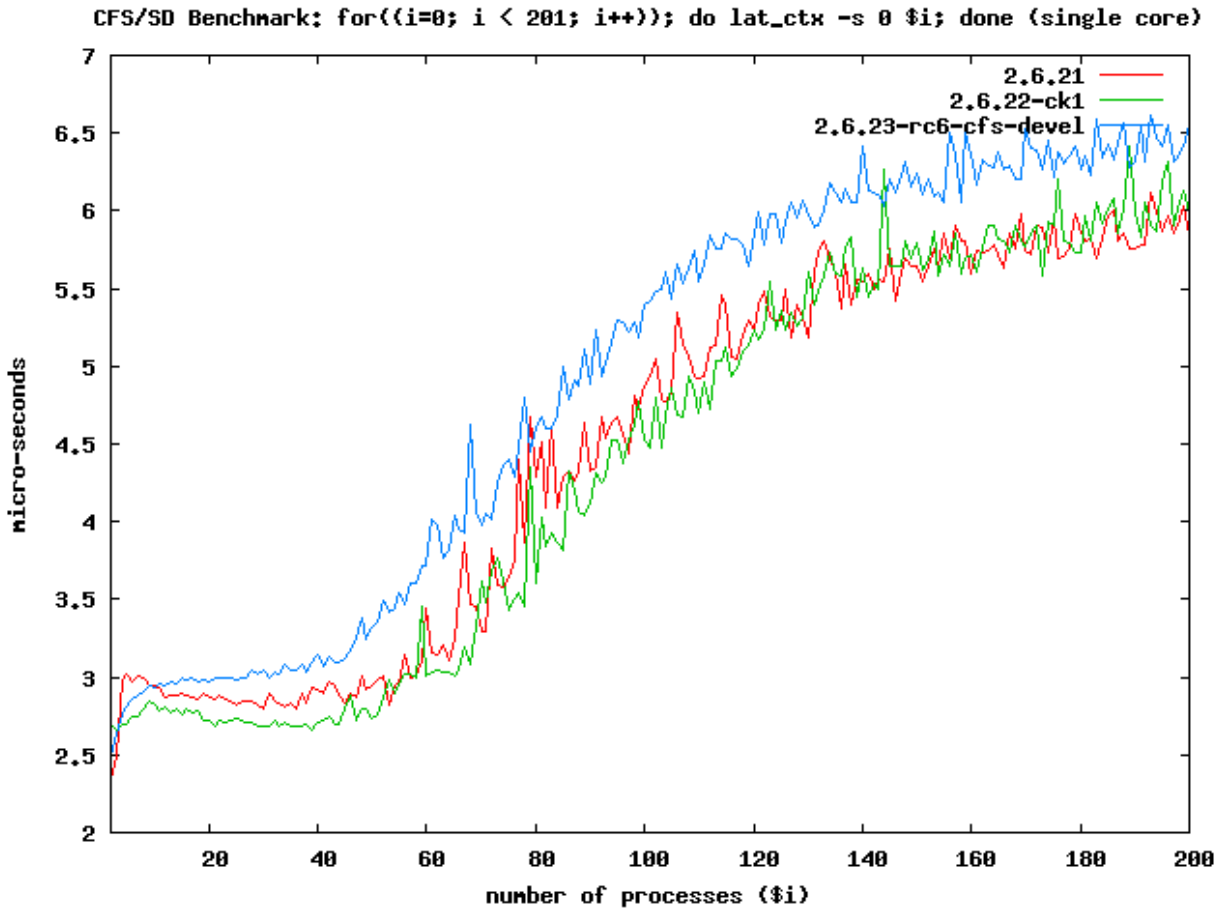
LMBench

- Zestaw mikrotestów służących do testowania szerokości pasma (*bandwidth*) oraz czasu oczekiwania (*latency*) rozmaitych operacji systemowych.
- Działa dla różnych systemów operacyjnych opartych na UNIXie.
- Mierzy m.in.
  - Szybkość dostępu do pamięci.
  - Tworzenie procesów i komunikacja między nimi.
  - Jakość działania sieci.

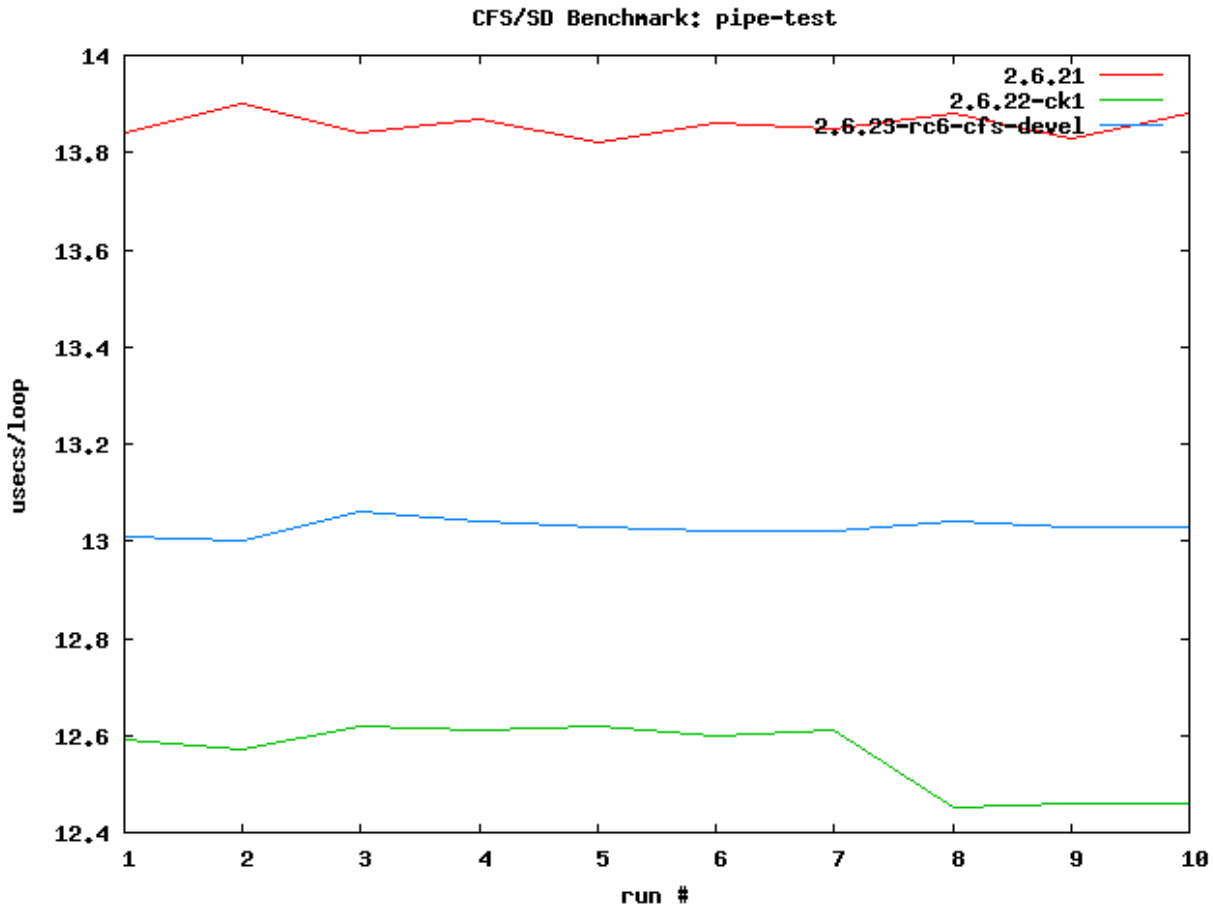
lat_ctx - Test przełączania kontekstu

- Idea: Pierścień procesów połączonych za pomocą łącz nienazwanych (*pipe*), procesy przekazują sobie żeton.
- Kiedy proces otrzymuje żeton, może wykonać pewną pracę przed przekazaniem go dalej.
- Praca polega na wykonywaniu obliczeń na tablicy, i służy psuciu cache'a.
- Zwraca średni czas przełączenia kontekstu
- *pipe-test* - *lat_ctx* dla dwóch procesów, bez żadnej dodatkowej pracy

lat_ctx, O(1) vs RSDL vs CFS



pipe-test, O(1) vs RSDL vs CFS



Trendy rozwojowe - podsumowanie

Wydajność

Główna przyczyna zmiany oryginalnego schedulera na O(1).

Trendy rozwojowe - podsumowanie

Wydajność

Główna przyczyna zmiany oryginalnego schedulera na O(1).

Obsługa nowszych technologii

Brak prawidłowej obsługi SMP był jedną z poważniejszych wad oryginalnego schedulera.

Trendy rozwojowe - podsumowanie

Wydajność

Główna przyczyna zmiany oryginalnego schedulera na O(1).

Obsługa nowszych technologii

Brak prawidłowej obsługi SMP był jedną z poważniejszych wad oryginalnego schedulera.

Interaktywność i Sprawiedliwość

Istotniejsze od wydajności przy wyborze między CFSem i RSDLem.