1. Problemy z planistą O(1)

Planista O(1) jest efektywną realizacją schematu szeregowania z wykorzystaniem wielopoziomowych kolejek ze sprzężeniem zwrotnym. Mimo, że jest to najczęściej implementowane rozwiązanie w systemach operacyjnych, to posiada ono kilka wad, które są nierozerwalnie z nim związane:

1. kwanty czasu związane z określonym poziomem uprzejmości są niezmienne, co oznacza, że jeśli w systemie występują tylko dwa procesy niskopriorytetowe, to mogą korzystać z procesora przez bardzo krótki czas i podlegają częstemu wywłaszczaniu,
2. granulacja kwantów czasu może być zbyt mała, kwanty czasów przydzielane dwóm procesom o wysokich, ale różnych priorytetach mają zbliżoną długość, podczas gdy kwanty czasu przydzielane dwóm procesom o różnych, ale niskich priorytetach różnią się znacząco długością,
3. pomiar upływu kwantu czasu nie jest precyzyjny,
4. heurystyki używane do określania stopnia interaktywności procesu nie są „nieomylne" i mogą przyznawać zbyt duże udziały czasu procesora procesom, które w rzeczywistości tego w ogóle nie wymagają.

Część z tych problemów została rozwiązana w implementacji planisty O(1), ale wyeliminowanie wszystkich problemów okazało się niemożliwe. Wady szeregowania opartego na wielopoziomowych kolejkach ze sprzężeniem zwrotnym ujawniały się szczególnie często w komputerach typu desktop. Dlatego od wersji 2.6.23 jądra zastąpiono planistę O(1) planistą CFS (ang. Completely Fair Scheduler).

2. Klasy szeregowania

Przebudowa mechanizmu szeregowania w jądrze jest dosyć obszerna. Wprowadzono nowe struktury typu `struct sched_class`, które opisują politykę szeregowania poszczególnych grup procesów. Każda z tych struktur zawiera wskaźniki na funkcje realizujące określone czynności w ramach danej polityki:

`enqueue_task()` umieszcza proces w kolejce zadań aktywnych,

`dequeue_task()` usuwa proces z kolejki zadań aktywnych,

`yield_task()` implementuje dobrowolne zrzeczenie się procesora przez proces,

`check_preempt_curr()` sprawdza, czy bieżące zadanie musi być wywłaszczone przez proces, który został obudzony (wyszedł ze stanu oczekiwania),

`pick_next_task()` wybiera kolejne zadanie do wykonania,

`put_prev_task()` związana jest ze zmianą kontekstu procesora i zachowaniem stanu poprzedniego procesu,

`set_curr_task()` wywoływana jest jeśli następuje zmiana polityki szeregowania bieżącego zadania,

`new_task()` odpowiedzialna za przydział procesora nowo powstałym zadaniom.

Mechanizm szeregujący Linuksa udostępnia sześć klas szeregowania:

- SCHED_FIFO - procesy czasu rzeczywistego szeregowane za pomocą algorytmu FCFS,
- SCHED_RR - procesy czasu rzeczywistego szeregowane za pomocą algorytmu rotacyjnego,
- SCHED_DEADLINE - procesy czasu rzeczywistego szeregowane według algorytmu EDF (Earliest Deadline First); ta klasa dostępna jest od wersji 3.14 jądra,
- SCHED_NORMAL - zwykłe procesy szeregowane przez planistę CFS, ta klasa jest odpowiednikiem klasy SCHED_OTHER w standardzie POSIX,

- SCHED_BATCH - jest to klasa procesów niskopriorytetowych ograniczonych przez procesor, które obsługiwane są przez planistę CFS,
- SCHED_IDLE - klasa procesów o jeszcze niższym priorytecie niż SCHED_BATCH.

Warto odnotować, że nowy mechanizm szeregowania może szeregować nie tylko pojedyncze procesy, ale i grupy procesów, zatem to co podlega szeregowaniu jest określane jako jednostka szeregowania i jest opisane strukturą typu `struct sched_entity`. Pole tego typu jest umieszczone w deskryptorze procesu. Przykładem takiej grupy jest grupa procesów czasu rzeczywistego, która od wersji jądra 2.6.25 podlega szeregowaniu `rt_bandwidth`. To szeregowanie zakłada, że 95% jednej sekundy pracy procesora ma być poświęcone na realizację procesów czasu rzeczywistego, a 5% na realizację „zwykłych" procesów. Te proporcje są konfigurowalne. Podstawowym celem wprowadzenia tej metody szeregowania jest zapobieganie monopolizacji procesora przez procesy z klasy szeregowania SCHED_FIFO.

3. Priorytety

Priorytety procesów od czasu wprowadzenia mechanizmu CFS stały się prawie niezmienne. Służą one do określenia tak zwanej wagi procesu. Istnieje jednak sytuacja wyjątkowa, która pozwala zmienić tymczasowo priorytet procesu normalnego na priorytet czasu rzeczywistego. Tą sytuacją jest korzystanie przez proces z wywołania systemowego, które używa muteksu czasu rzeczywistego (ang. RT-Mutex). Ma to na celu zapobieżenie zjawisku *inwersji priorytetów*.

4. Sprawiedliwe szeregowanie - koncepcja

W szeregowaniu sprawiedliwym nie występuje pojęcie kwantów czasu, a przynajmniej nie jest ono obecne w takiej postaci jak w szeregowaniu rotacyjnym. Głównym kryterium jest tu moc obliczeniowa procesora. Planista dąży do tego, aby każdy z procesów otrzymał należną mu porcję tej mocy. Gdyby istniał procesor idealny, to każdemu z n identycznych zadań gotowych do wykonania przydzielałby $\frac{1}{n}$ mocy tego procesora. Każde z tych zadań wykonywałoby się n razy wolniej, niż gdyby korzystało samodzielnie z procesora, ale za to wszystkie zdania kończyłyby się w tym samym czasie od przydzielenia procesora i wykonywałyby się bez zbędnych przerw. W rzeczywistych komputerach taka koncepcja nie jest możliwa do realizacji. Można jednak przydzielać procesom procesor na podstawie czasu, przez który one z niego *nie* korzystały. Przykładowo, jeśli w systemie jest tylko jeden proces gotowy do uruchomienia to może on korzystać dowolnie długo z procesora, czyli do czasu aż skorzysta z wywołania systemowego, które wprowadzi go w stan oczekiwania. Jeśli jednak pojawi się nowy proces gotów do wykonania, to wywłaszcza on bieżące zadanie, bo krócej korzystał z procesora. Planista CFS przydziela więc procesor temu procesowi, który najkrócej z niego korzystał. Zamiast jednak liczyć czas oczekiwania na przydział procesora dla każdego z procesów, prościej jest policzyć właśnie czas korzystania z procesora przez bieżący proces. Ten czas liczony jest na podstawie wskazań zegara wirtualnego, który odmierza upływ czasu proporcjonalnie do liczby procesów, które znajdują się w kolejce procesów gotowych do wykonania[1]. Jeśli proces otrzymuje procesor, to jego wirtualny czas wykonania rośnie, podczas gdy dla pozostałych procesów pozostaje on bez zmian. Ten prosty schemat nie może jednak być całkowicie zrealizowany w praktyce, bo należy uwzględnić czas przełączania kontekstu i priorytety poszczególnych procesów.

5. Szeregowanie sprawiedliwe - realizacja

Priorytety procesów przeliczane są przez planistę CFS na tak zwane wagi. Służy do tego 40-elementowa tablica o nazwie `prio_to_weight`. Zadania o domyślnym poziomie uprzejmości mają wagę równą 1024. Wagi związane z niższymi priorytetami wyliczane są poprzez sukcesywne dzielenie przez 1,25, a wagi dla priorytetów wyższych, przez sukcesywne mnożenie przez 1,25. Tworzą one zatem ciąg geometryczny. Jądro utrzymuje również tablicę odwrotności tych wag, która nazywa się `prio_to_wmul`. Jest to konieczne, gdyż wartości wag są używane w wyrażeniach, które wymagają przeprowadzania operacji dzielenia i które związane są z wyliczeniem czasu oczekiwania procesu.

[1] Dla n procesów czas wirtualny płynie n-krotnie wolniej.

Kolejka zadań dla planisty CFS zorganizowana jest w postaci drzewa czerwono-czarnego. Takie drzewa są odmianą drzew BST, w których z każdym węzłem związana jest cecha nazywana kolorem. Kolory podlegają następującym regułom:

1. Korzeń ma kolor czarny.
2. Każdy węzeł ma kolor czarny lub czerwony.
3. Potomkowie węzła czerwonego muszą być zawsze czarni.
4. Liście drzewa są czarne.
5. Wszystkie proste ścieżki od wybranego węzła do dowolnego podlegającego mu liścia zawierają tyle samo węzłów czarnych.

Reguła pierwsza jest stosowana czasem zamiennie z czwartą. Liśćmi drzewa czerwono-czarnego są wskaźniki puste. Jeśli wskutek wstawiania lub usuwania węzłów z drzewa któraś z powyższych reguł zostanie złamana, to taka sytuacja jest naprawiana poprzez operacje zmiany kolorów węzłów i/lub wykonywanie rotacji prawo i lewostronnych. Utrzymywanie prawdziwości tych reguł ma na celu zapewnienie, że drzewo będzie przynajmniej częściowo zrównoważone i że czas realizacji operacji z nim związanych będzie proporcjonalny do jego wysokości ($O(log_2(n))$, gdzie n jest liczbą elementów drzewa). Linux dostarcza gotowej implementacji drzew czerwono-czarnych, podobnie jak ma to miejsce w przypadku list. Planista CFS wykorzystuje te drzewa do utrzymywania w postaci posortowanej zbioru procesów aktywnych. Kryterium sortowania jest czas korzystania z procesora. Skrajnie lewy element drzewa czerwono-czarnego jest związany z procesem, który najmniej korzystał z tego procesora.

Zegar wirtualny, który jest jednym z elementów wyznaczających czas, przez który proces bieżący korzystał z procesora nie został wprost zaimplementowany w jądrze Linuksa. Czas jaki wskazywałby ten zegar jest wyliczany na postawie czasu fizycznego mierzonego z rozdzielczością nanosekundową, liczby procesów w kolejce zadań gotowych do wykonania oraz wagi procesu bieżącego. Wirtualny czas wykonania dla tego procesu jest aktualizowany przez okresowo uruchamianą funkcję `update_curr()` i zapisywany w odpowiednim polu deskryptora[2]. Dla procesów z priorytetem 0 (domyślnym) upływ czasu wirtualnego, bez uwzględnienia liczby procesów aktywnych, jest taki sam jak upływ czasu rzeczywistego. W przypadku procesów o innych priorytetach jest on wyliczany zgodnie ze wzorem:

$$\text{okres\_korzystania\_z\_procesora} \times \frac{\text{waga\_dla\_priorytetu\_0}}{\text{waga\_dla\_priorytetu\_procesu}} \tag{1}$$

Jeśli wartość wirtualnego czasu korzystania z procesora dla bieżącego procesu staje się większa od wartości czasu, przez który korzystał z procesora proces reprezentowany w drzewie czerwono-czarnym przez skrajnie lewy element, to następuje wywłaszczenie bieżącego procesu na rzecz tego zdania. Funkcja wstawiająca do drzewa proces, który uzyskał stan gotowości do uruchomienia, przegląda w pętli drzewo czerwono-czarne i w każdej iteracji sprawdza, czy nastąpiło przejście w prawą stronę. Jeśli nie wystąpiło ono w ogóle, to wstawiony element jest skrajnie lewym elementem drzewa i funkcja ustawia osobny wskaźnik, aby na niego wskazywał. Dzięki temu wskaźnikowi wybór kolejnego zdania, które powinno otrzymać procesor jest stosunkowo szybki - jest ono wskazywane przez ten właśnie wskaźnik. Jeśli jego wartość wynosi NULL, to znaczy, że żaden z procesów z klasy SCHED_NORMAL nie jest gotów do wykonania lub, że nie ma w ogóle takich zadań w systemie i należy przydzielić procesor zadaniu z klasy SCHED_BATCH lub SCHED_IDLE.

Jądro ma jeszcze jeden mechanizm, który zapewnia, że szeregowanie procesów powinno być sprawiedliwe. Ten mechanizm dba, aby wszystkie procesy znajdujące się w kolejce zadań otrzymały procesor w okresie czasu określonym parametrem o nazwie `sched_latency_ns`. Domyślna wartość tego parametru wynosi 20ms, ale może być regulowana z nanosekundową rozdzielczością. Inny parametr związany z tym mechanizmem nazwany jest `sched_nr_latency` i określa maksymalną liczbę procesów, które w tym okresie muszą być obsłużone. Jego wartość jest zmieniana przez jądro dynamicznie, w zależności od liczby procesów aktywnych. Warto zauważyć, że przy dużej liczbie procesów gotowych do wykonania, każdy z nich dostałby bardzo mały udział czasu procesora, mniejszy nawet niż czas przełączania kontekstu. Takie rozwiązanie jest bardzo niepożądane, zatem mechanizm szeregujący wprowadza dolne ograniczenie

[2] Jest to jedno z pól struktury typu `struct sched_entity` wbudowanej w deskryptor.

czasu korzystania przez pojedyncze zadanie z procesora, które wynosi 1ms. Zatem dla dużej liczby procesów aktywnych CFS nie jest całkowicie sprawiedliwy, ale dzięki temu nie dochodzi do sytuacji, w której procesy nie mogą efektywnie korzystać z procesora.

Planista CFS dba także o to, aby nowo powstały proces był uruchomiony przed swoim rodzicem. Aby to osiągnąć, jeśli zachodzi taka konieczność zamienia wirtualne czasy wykonania obu procesów.

Sposób szeregowania procesów czasu rzeczywistego nie uległ zmianie, z wyjątkiem dodania wcześniej opisanych usprawnień. Funkcja `schedule()` została przebudowana w ten sposób, że przegląda wszystkie klasy szeregowania w kolejności określonej ich priorytetami (od zadań czasu rzeczywistego do procesów bezczynności) i przydziela procesor zadaniu o najwyższym priorytecie z pierwszej niepustej klasy jaką napotka.