Systemy Operacyjne 2

Szeregowanie Procesów, część 1

Arkadiusz Chrobot

Katedra Systemów Informatycznych

14 marca 2024

Plan

1. Szeregowanie Procesów
2. Oryginalny planista uniksowy
3. Planista O(1)
4. Api planisty dla przestrzeni użytkownika

Szeregowanie Procesów

Szeregowanie, w kontekście systemów operacyjnych, to podejmowanie decyzji dotyczących przydzielania zasobów procesom. Zazwyczaj, ale nie zawsze, pojęcie to oznacza *szeregowanie procesów*, czyli decydowanie o przydziale CPU określonemu procesowi. Podsystem jądra odpowiedzialny za decyzję, który z procesów jako następny powinien używać procesora jest nazywany *planistą procesów* lub po prostu *planistą*.

Większość współczesnych systemów operacyjnych jest *wielozadaniowa*, co oznacza, że przeplatają one wykonanie procesów. Występują dwa typy takich systemów: *z kooperacją* i *z wywłaszczaniem*. W przypadku pierwszych proces dobrowolnie zrzeka się przydzielonego wcześniej procesora. Główną wadą takiego rozwiązania jest to, że źle działający proces użytkownika może zablokować cały system. Systemy z wywłaszczaniem nie mają tej wady, bo mogą w każdej chwili odebrać CPU bieżąco wykonywanemu procesowi. Linux, podobnie jak inne popularne systemy operacyjne, jest systemem z wywłaszczaniem.

Planista Linuksa

Głównym tematem tego wykładu jest *planista O(1)*, który był używany w jądrze Linuksa od wersji 2.6.0 do 2.6.22. Rozwiązanie, które go zastąpiło będzie omówione na następnym wykładzie. Pomimo, że planista O(1) nie jest już w użyciu, to posiadał on wiele interesujących elementów, które godne są uwagi. Bazował on na oryginalnym planiście systemu Unix, dlatego część kolejnych slajdów jest poświęcona temu mechanizmowi.

Oryginalny planista uniksowy

Oryginalny planista systemu Unix używał szeregowania opartego na schemacie wielopoziomowych kolejek ze sprzężeniem zwrotnym. Procesy w jego przypadku są podzielone na dwie grupy: zwykłe, podlegające polityce SCHED_OTHER oraz czasu rzeczywistego z dwiema politykami szeregowania SCHED_RR i SCHED_FIFO. Ta druga grupa może być uruchamiana tylko przez uprzywilejowanych użytkowników. Procesy z polityką SCHED_FIFO nie podlegają wywłaszczaniu. W przypadku polityki SCHED_RR są one szeregowane z użyciem algorytmu rotacyjnego z długimi kwantami czasu (ang. *time slice*). Priorytety procesów czasu rzeczywistego są liczbami naturalnymi z zakresu od 99 (najwyższy priorytet) do 1 (najniższy). Dodatkowo ten priorytet jest stały (niezmienny w czasie). To oznacza, że jeśli proces typu SCHED_RR zużyje swój kwant czasu lub proces typu SCHED_FIFO zrzeknie się CPU, to w kolejnej rundzie szeregowania otrzymają ten sam priorytet.

Oryginalny planista uniksowy

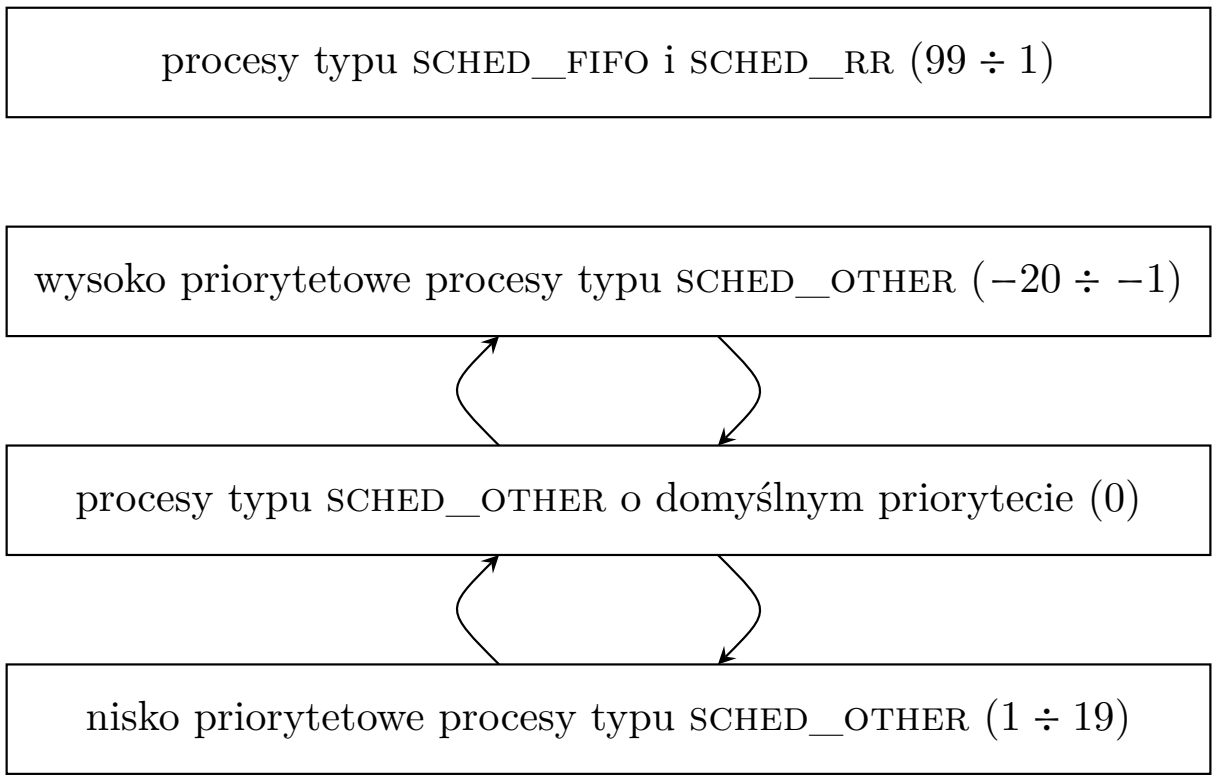
Procesy typu SCHED_OTHER mogą być uruchomione przez dowolnego użytkownika. Otrzymują one zawsze niższy priorytet niż procesy czasu rzeczywistego i podlegają szeregowaniu z użyciem algorytmu rotacyjnego. Ich priorytet jest dwuelementowy. Pierwszym elementem jest priorytet statyczny, który jest nadawany procesowi przez użytkownika i nazwany *poziomem uprzejmości* (ang. *nice level*). Jest to liczba całkowita z zakresu od -20 (najwyższy priorytet) do 19 (najniższy). Drugi element to priorytet dynamiczny, który jest dodawany do statycznego (nazywanego również *priorytetem bazowym*), po każdej rundzie szeregowania. Ten komponent może zwiększyć lub zmniejszyć całkowity priorytet procesu. Oznacza to, że po każdej rundzie szeregowania proces może zmienić kolejkę procesów gotowych (Rys. 1).

Oryginalny planista uniksowy

Domyślną wartością poziomu uprzejmości jest 0. Zgodnie ze standardem POSIX z tym poziomem musi być związany kwant o długości równej lub większej niż $20ms$. Unix faworyzuje *procesy zorientowane na wejście-wyjście* (ang. I/O–*bound*), ponieważ z reguły są one *procesami interaktywnymi*. Z drugiej strony stara się on również być sprawiedliwy dla *procesów zorientowany na przetwarzanie* (ang. CPU–*bound*).

Oryginalny planista uniksowy

Schemat wielokolejkowy ze sprzężeniami zwrotnymi



Rysunek 1: Wielokolejkowe szeregowanie procesów ze sprzężeniami zwrotnymi (ilustracja poglądowa)

Planista O(1)

Planista O(1) jest modyfikacją oryginalnego planisty systemu Unix autorstwa węgierskiego programisty Ingo Molnára. Posiada on następujące cechy:

1. implementuje szeregowanie o złożoności O(1),
2. implementuje (prawie) idealne skalowanie dla systemów SMP,
3. implementuje powiązanie wątku z procesorem w trybie SMP,
4. faworyzuje procesy interaktywne,
5. jest zoptymalizowany pod względem często występującego przypadku, kiedy więcej niż jeden proces jest gotów do działania.

Planista O(1)

Planista O(1) dla każdego CPU w systemie tworzy kolejkę procesów czekających na przydział procesora. Jest ona strukturą danych typu `struct runqueue`, który jest zdefiniowany w pliku `kernel/sched.c`, razem z kilkoma użytecznymi makrami. Dostęp do każdej takiej kolejki jest chroniony przy pomocy rygla pętlowego. Aby uniknąć zakleszczeń te blokady są zajmowane zawsze w tej samej kolejności. Pojedyncza struktura `runqueue` zawiera dwa wskaźniki do *tablicy priorytetów aktywnych* i *tablicy priorytetów przeterminowanych*.

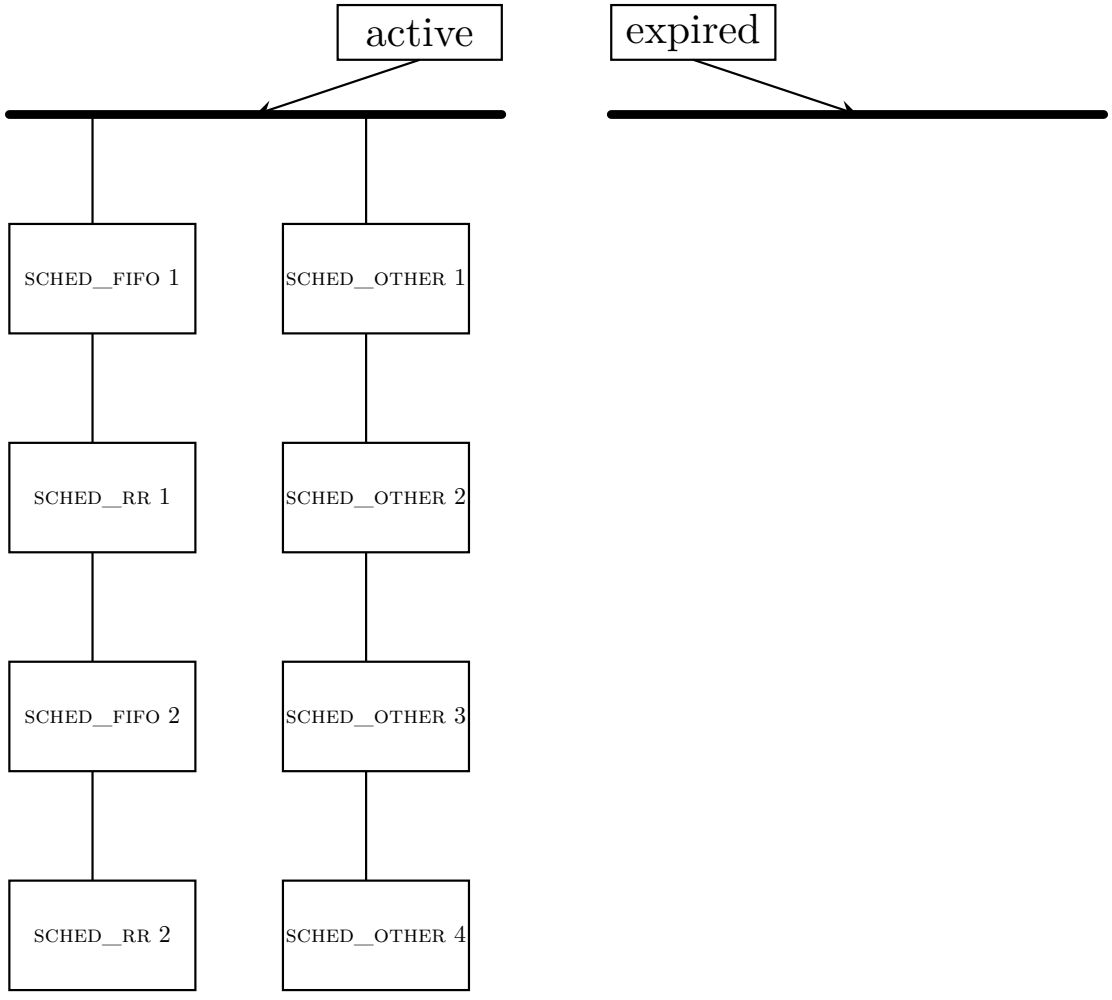
Planista O(1)

Tablica priorytetów jest strukturą typu `struct prio_array`. Posiada ona dwa pola. Pierwszym jest tablica o 140 elementach będących wskaźnikami na cykliczne, dwukierunkowe listy procesów, a właściwie, to elementów, które je reprezentują na potrzeby planisty. Priorytety tych procesów są przeliczane wewnątrz jądra w taki sposób, że procesy czasu rzeczywistego uzyskują wartości od 0 (najwyższy) do 98 (najniższy; wartość 99 nie jest używana), a zwykłe procesy otrzymują priorytety z zakresu od 100 (najwyższy) do 139 (najniższy). Dzięki temu te priorytety mogą być użyte bezpośrednio jako indeksy w tablicy.
Drugim polem struktury typu `struct prio_array` jest bitmapa (Rys. 3), która pozwala szybko znaleźć w tablicy pierwszą, niepustą listę procesów o najwyższym priorytecie. Bit związany z tą listą będzie miał wartość 1, a numer jego pozycji w bitmapie jest indeksem w tablicy.

Planista O(1)

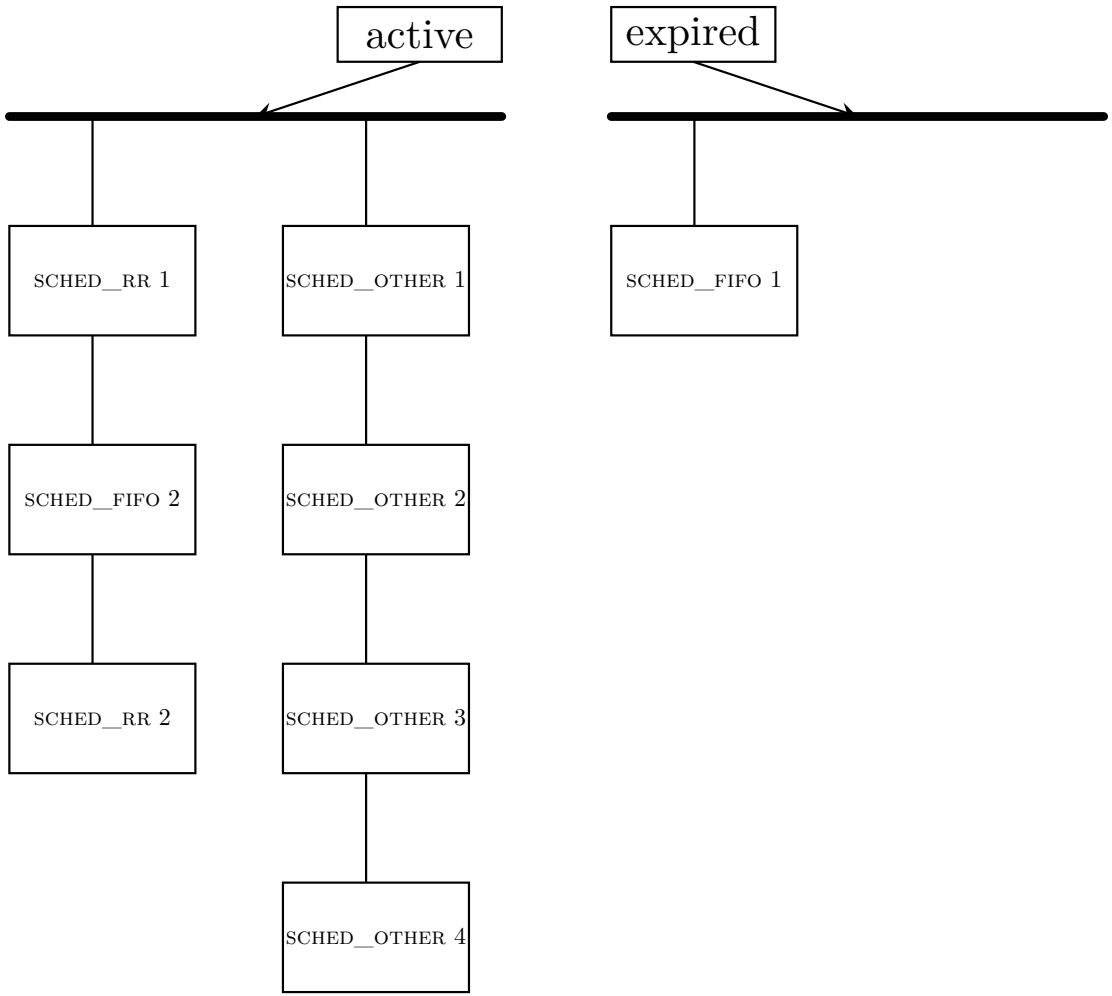
Zgodnie z tym co zostało wcześniej napisane, kolejka `runqueue` ma dwa wskaźniki do tablic priorytetów. Pierwszy wskazuje na tablicę *aktywnych* (ang. *active*) priorytetów. Procesy typu SCHED_RR i SCHED_OTHER, które nie wykorzystały jeszcze swojego kwantu czasu lub procesy typu SCHED_FIFO, które oczekują na przydzielenie CPU znajdują się w tej tablicy. Drugi wskaźnik wskazuje na tablicę *przeterminowanych* (ang. *expired*) priorytetów. Proces, który zrzeka się CPU lub całkowicie zużywa swój kwant czasu staje się przeterminowany i trafia do tej tablicy, a następnie oczekuje na kolejną rundę szeregowania. Linux wylicza nowy priorytet dla procesu typu SCHED_OTHER, zaraz po tym, jak staje się on przeterminowany. W Uniksie priorytety dla procesów były wyliczane na nowo dopiero po tym, kiedy wszystkie uległy przeterminowaniu. Po pewnym czasie tablica priorytetów aktywnych staje się pusta, a wszystkie procesy znajdują się w tablicy priorytetów przeterminowanych. Jądro dokonuje zamiany ich ról w efektywny sposób, zamieniając miejscami adresy we wspomnianych wskaźnikach (Rys. 2).

Planista O(1)



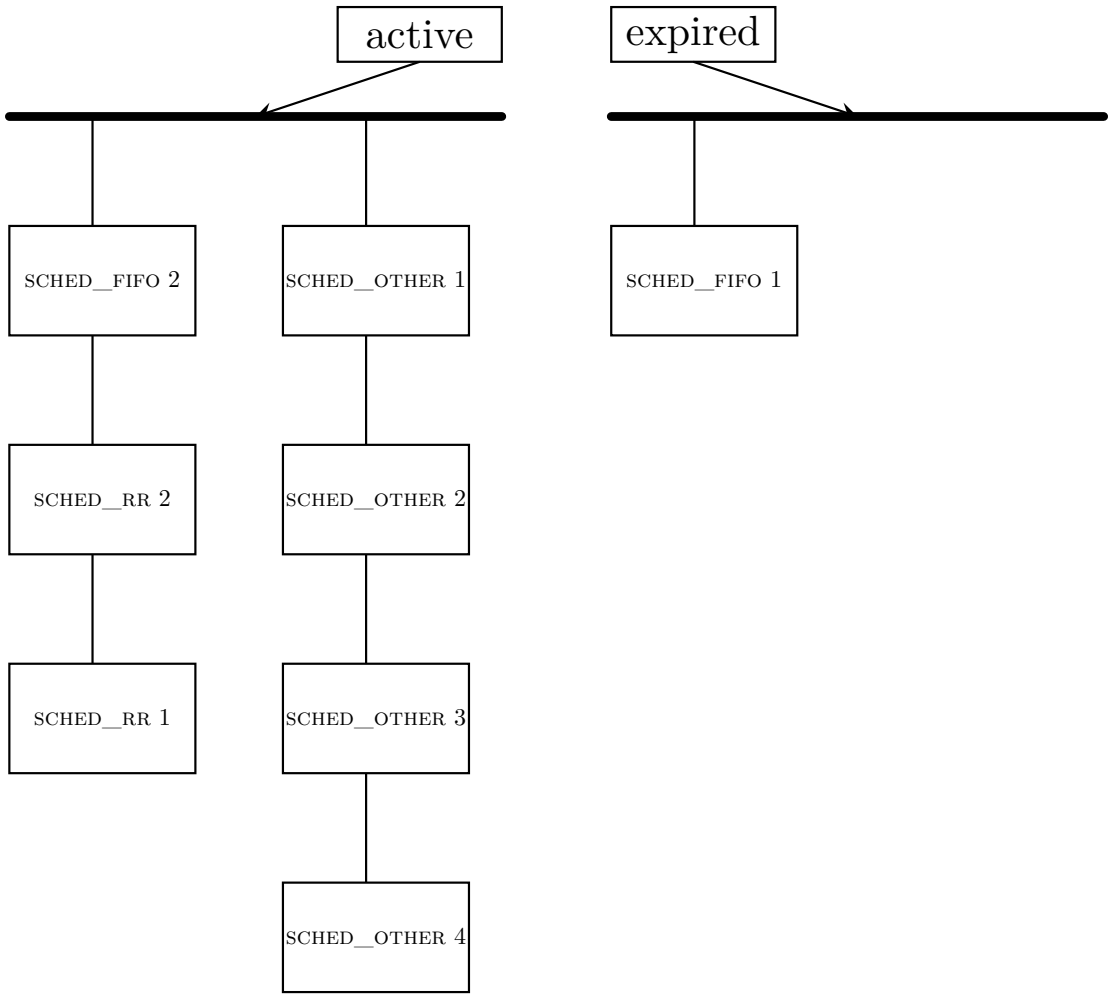
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



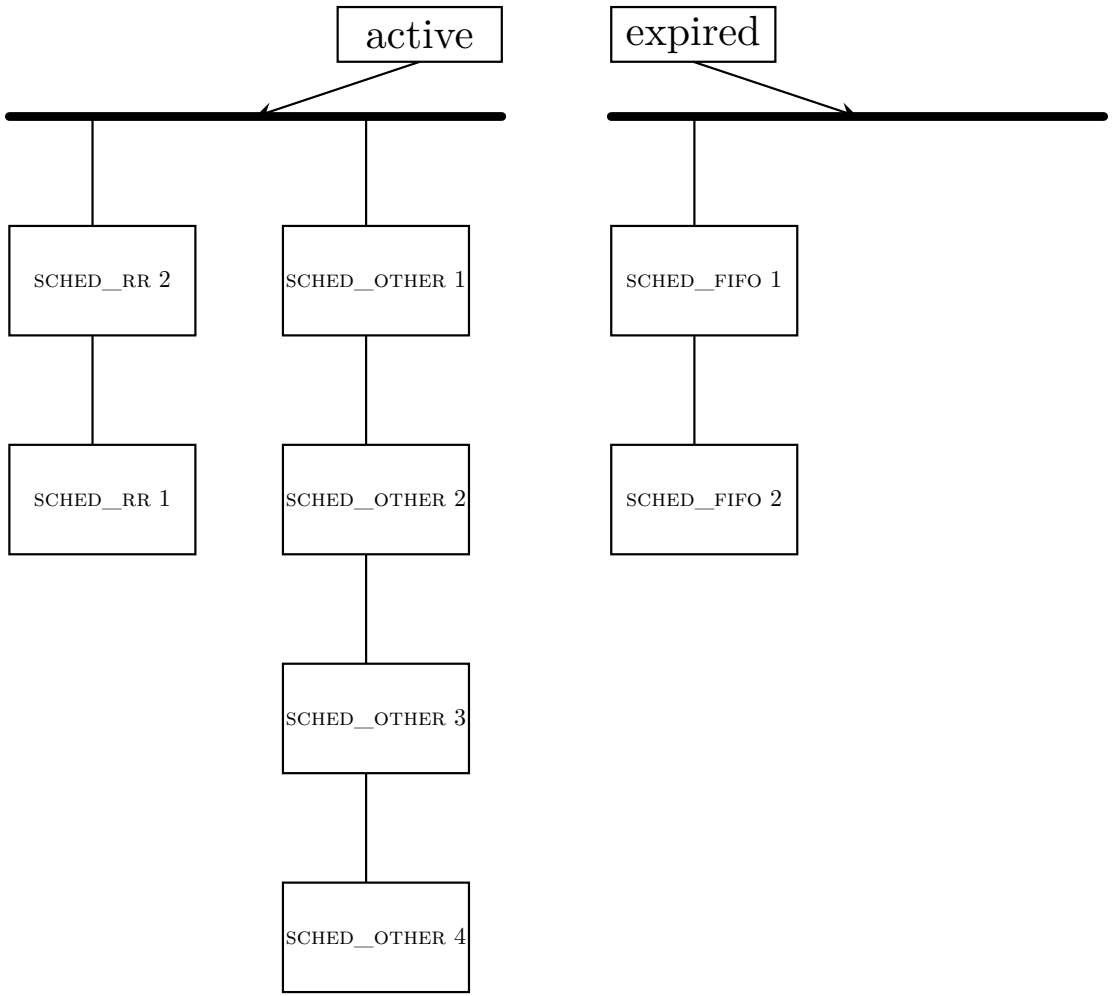
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



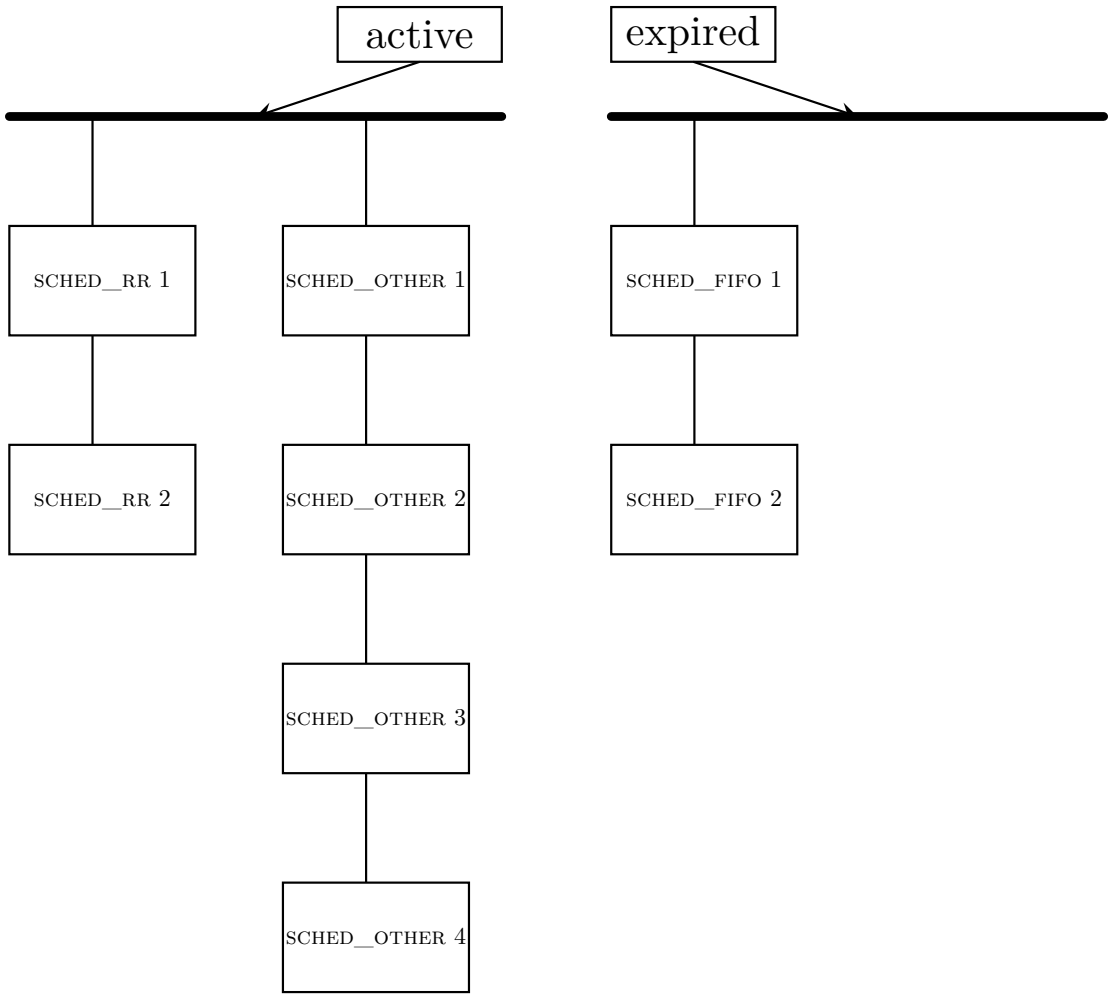
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



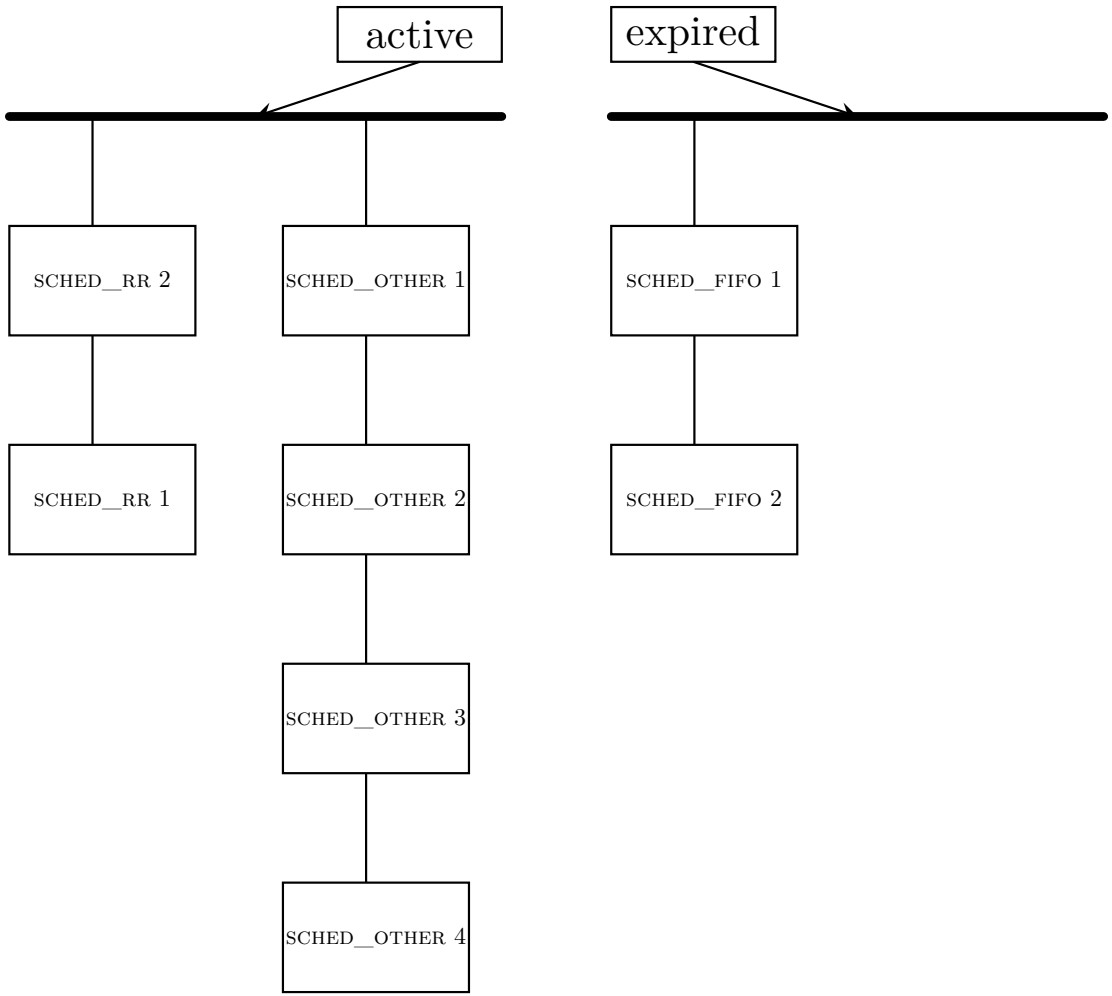
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



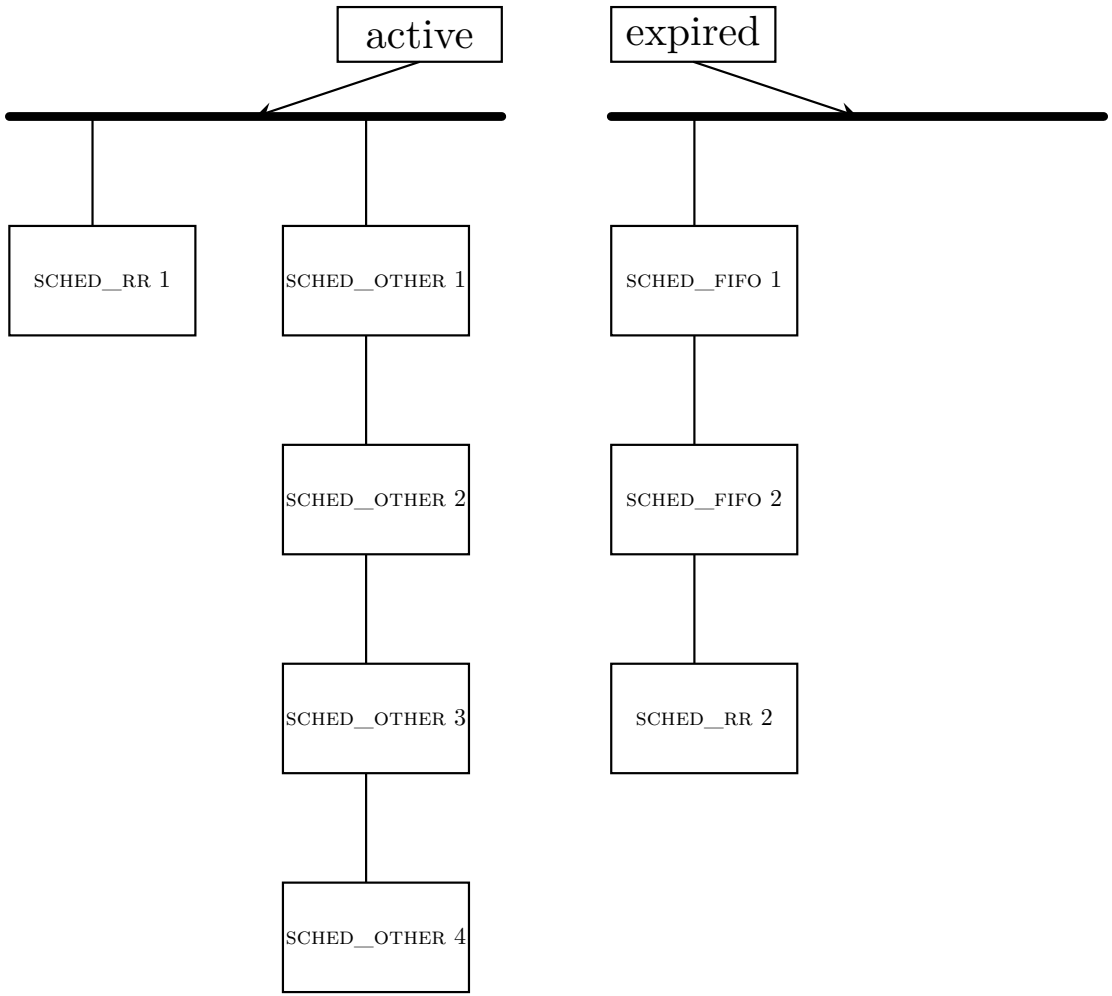
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



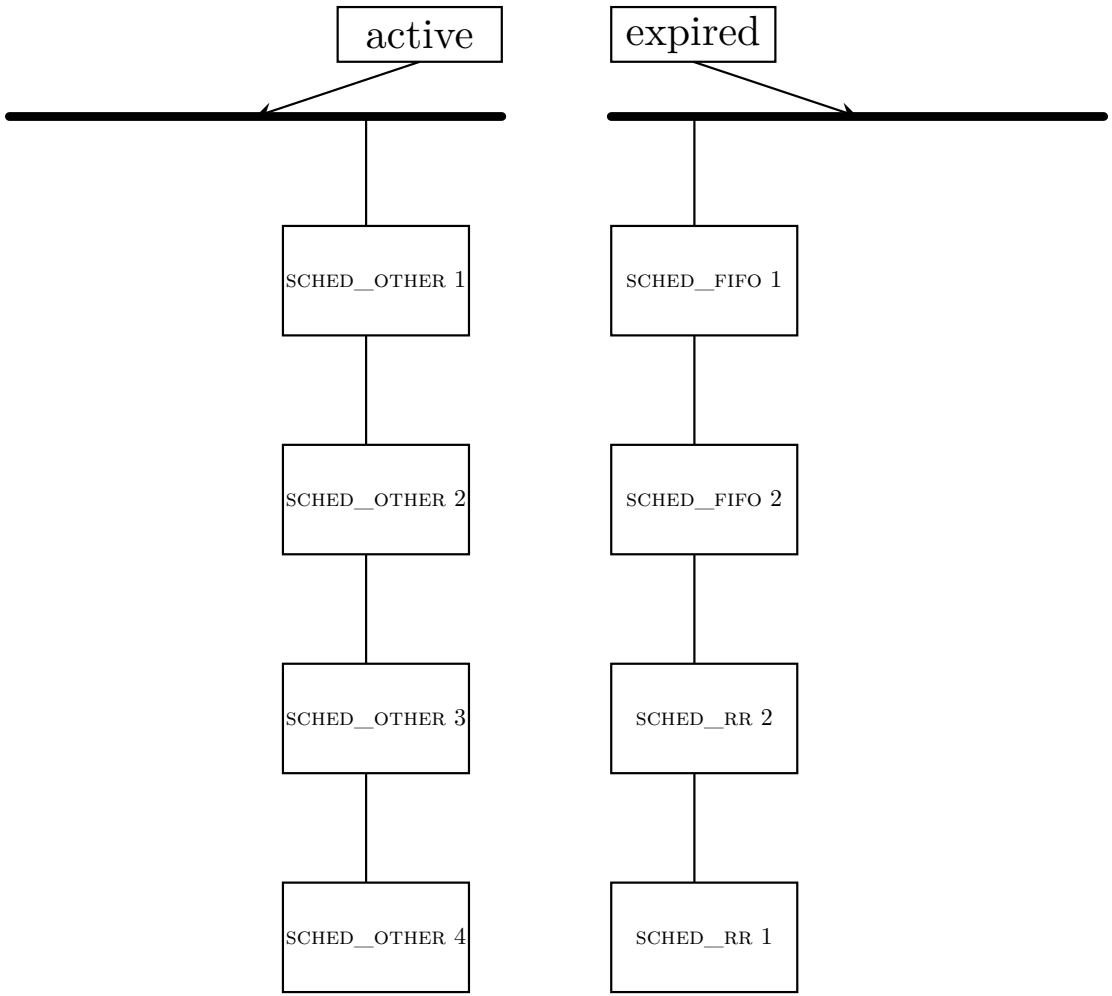
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



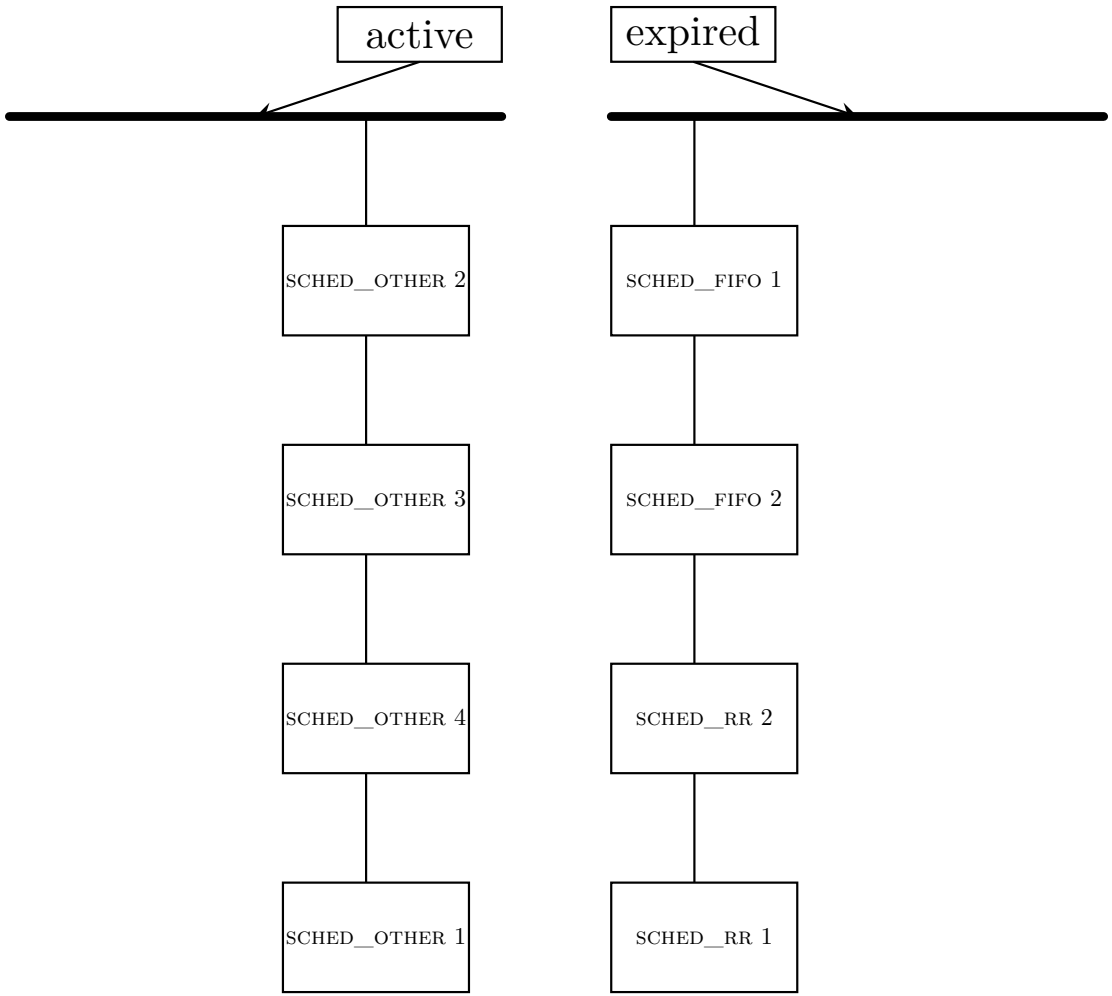
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



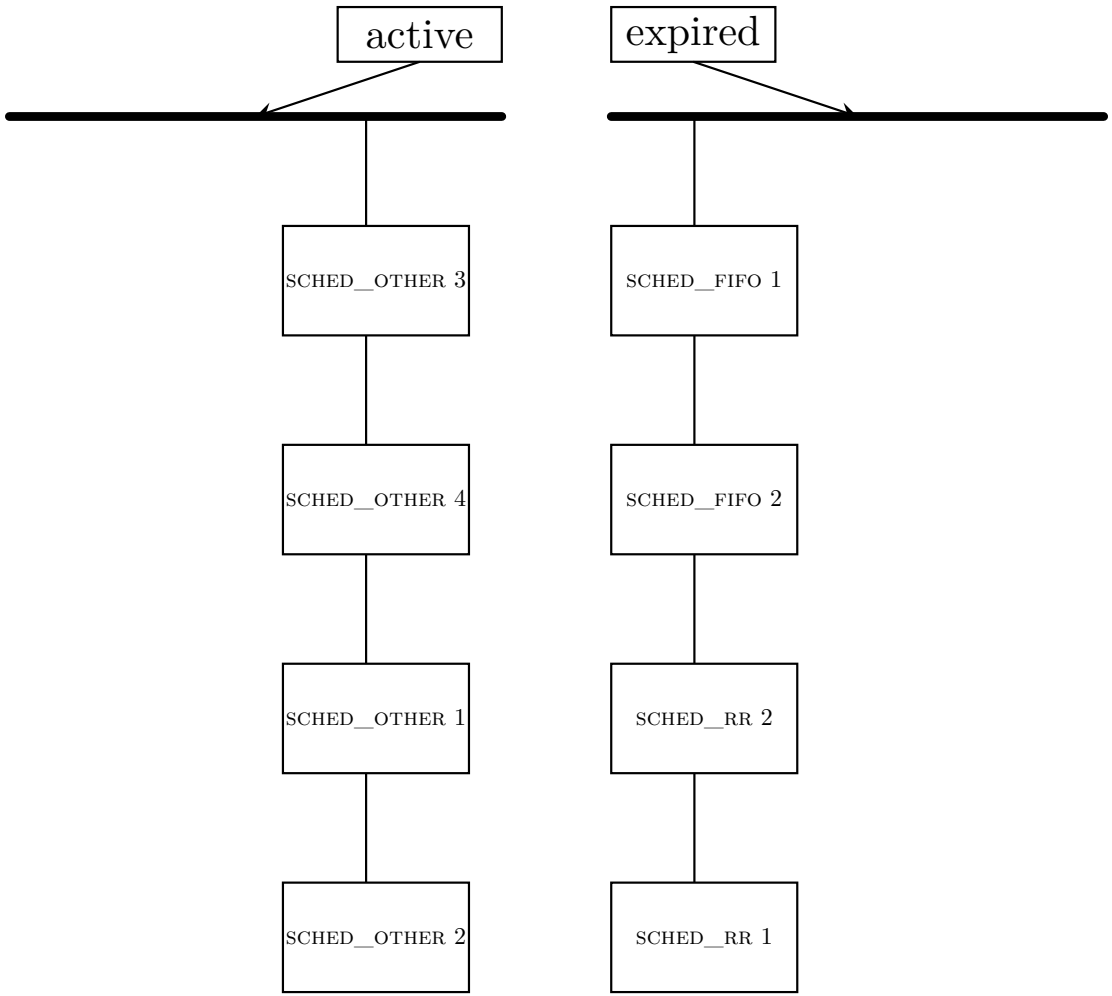
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



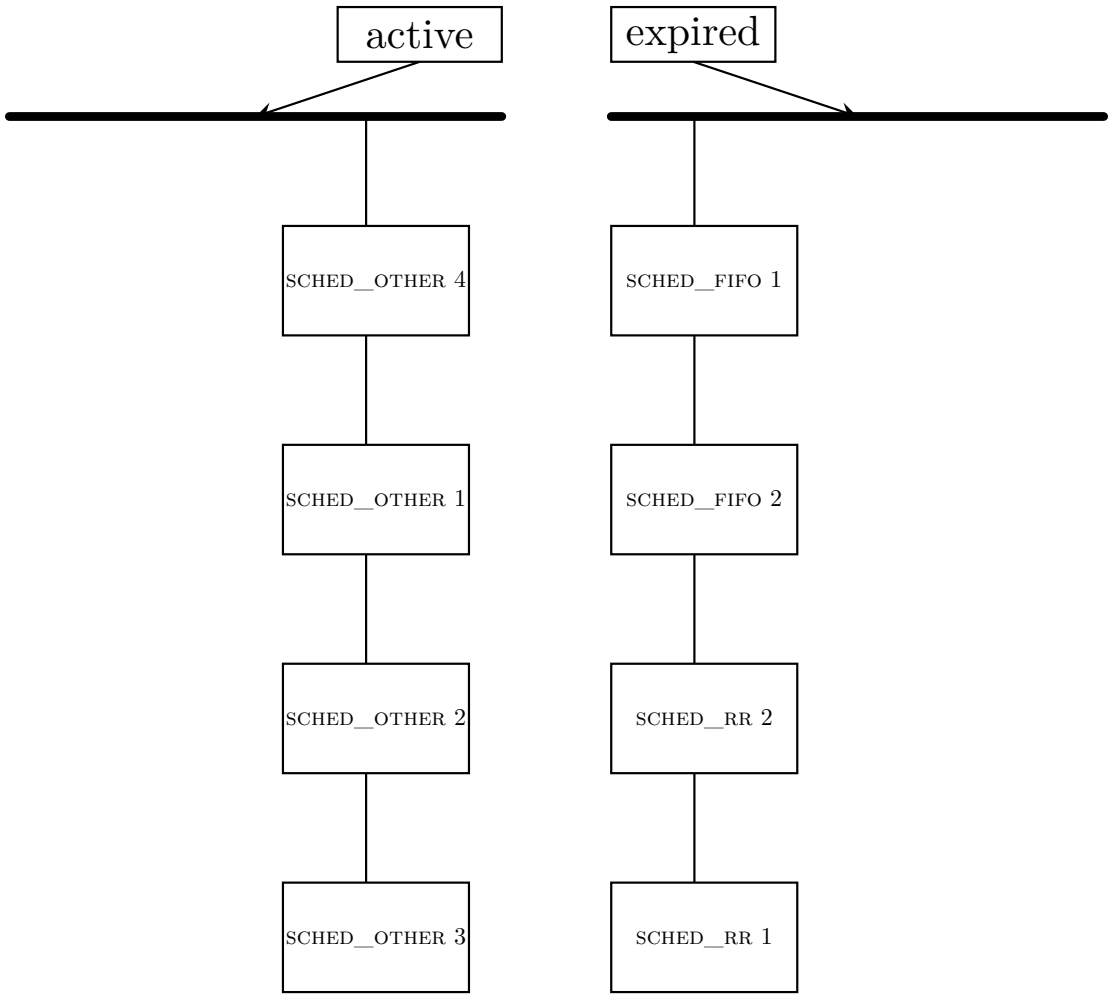
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



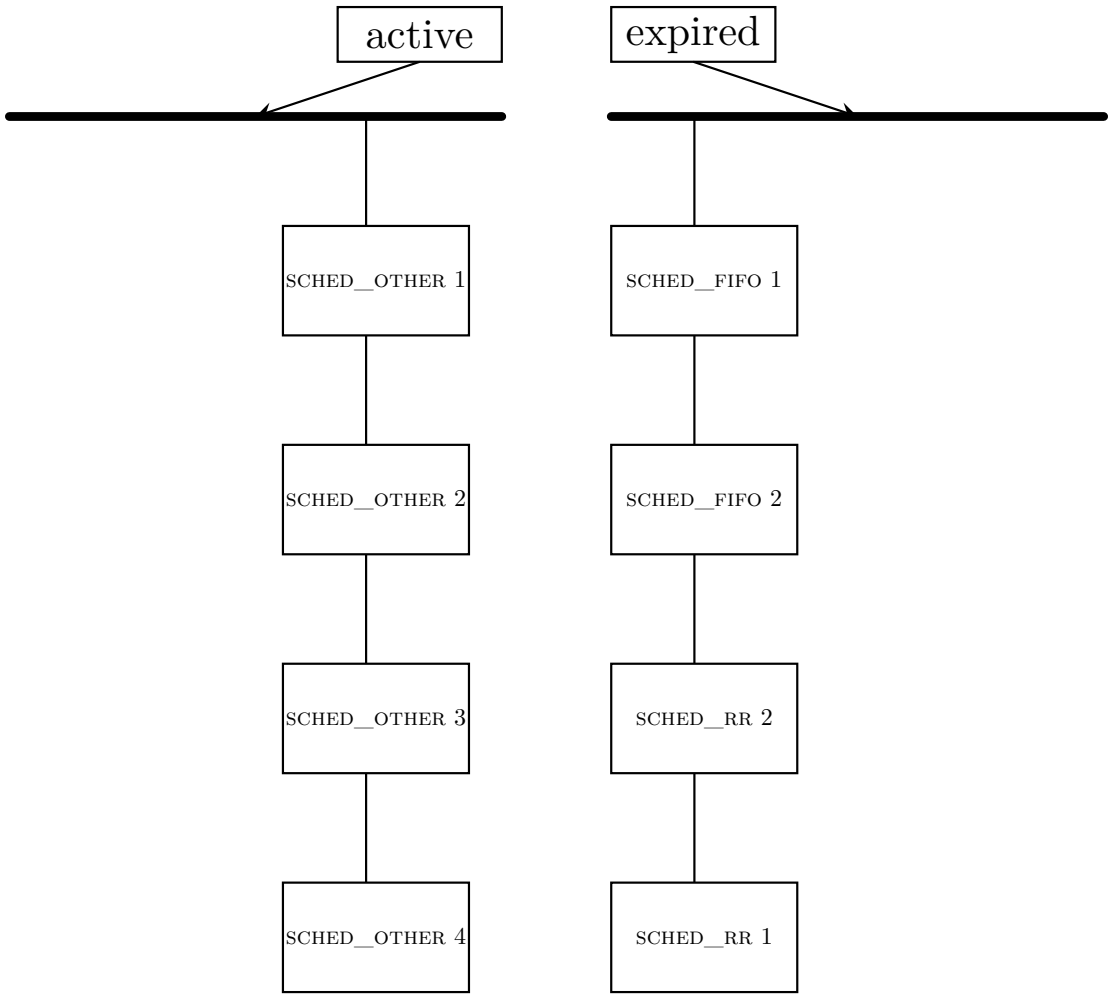
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



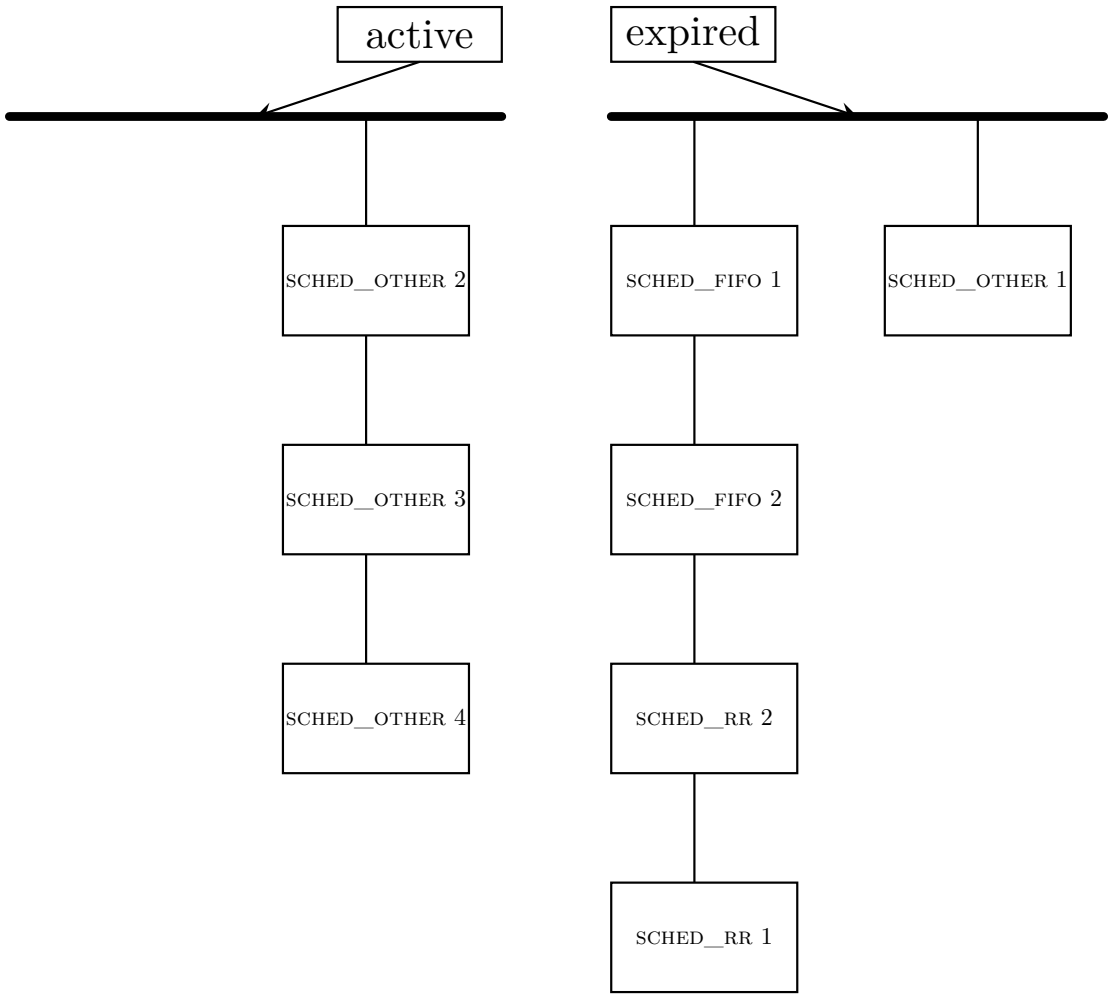
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



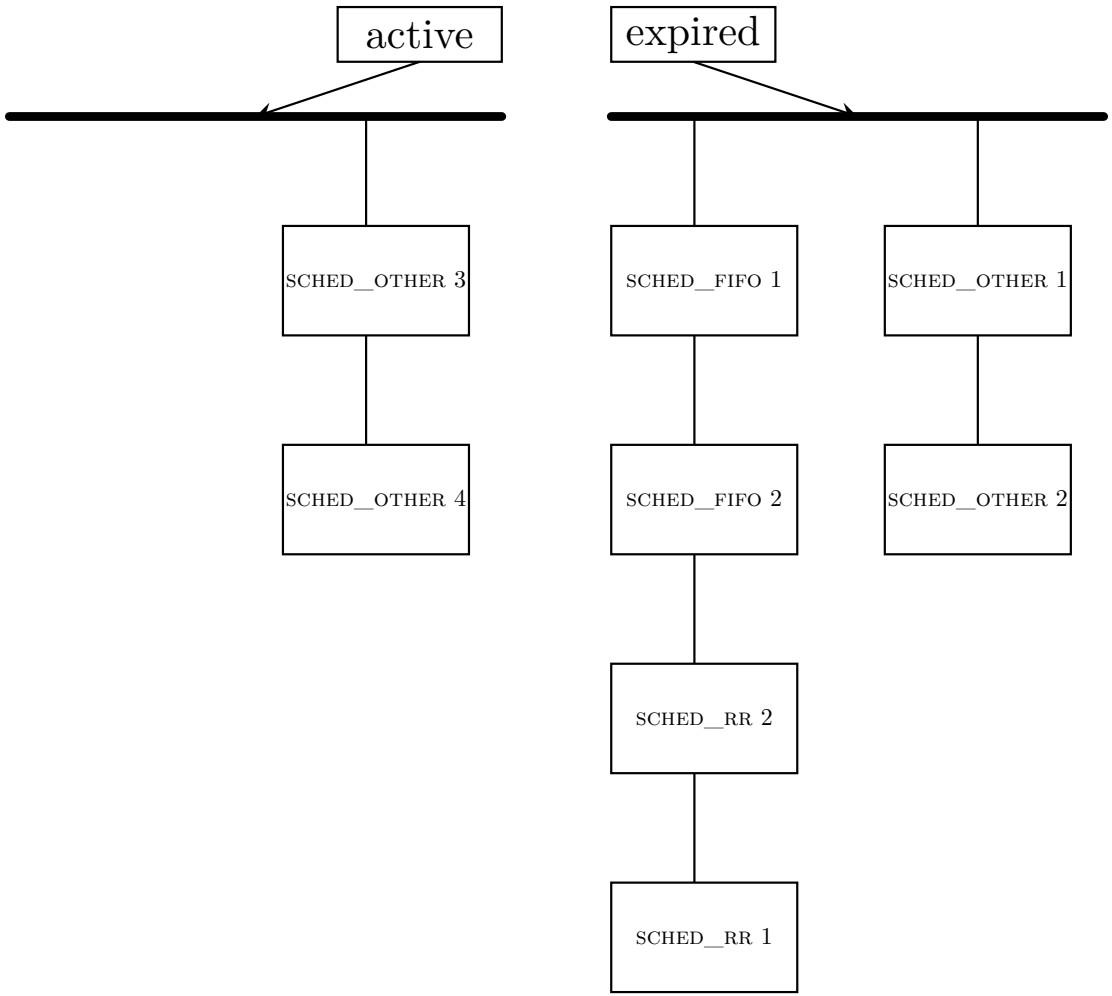
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



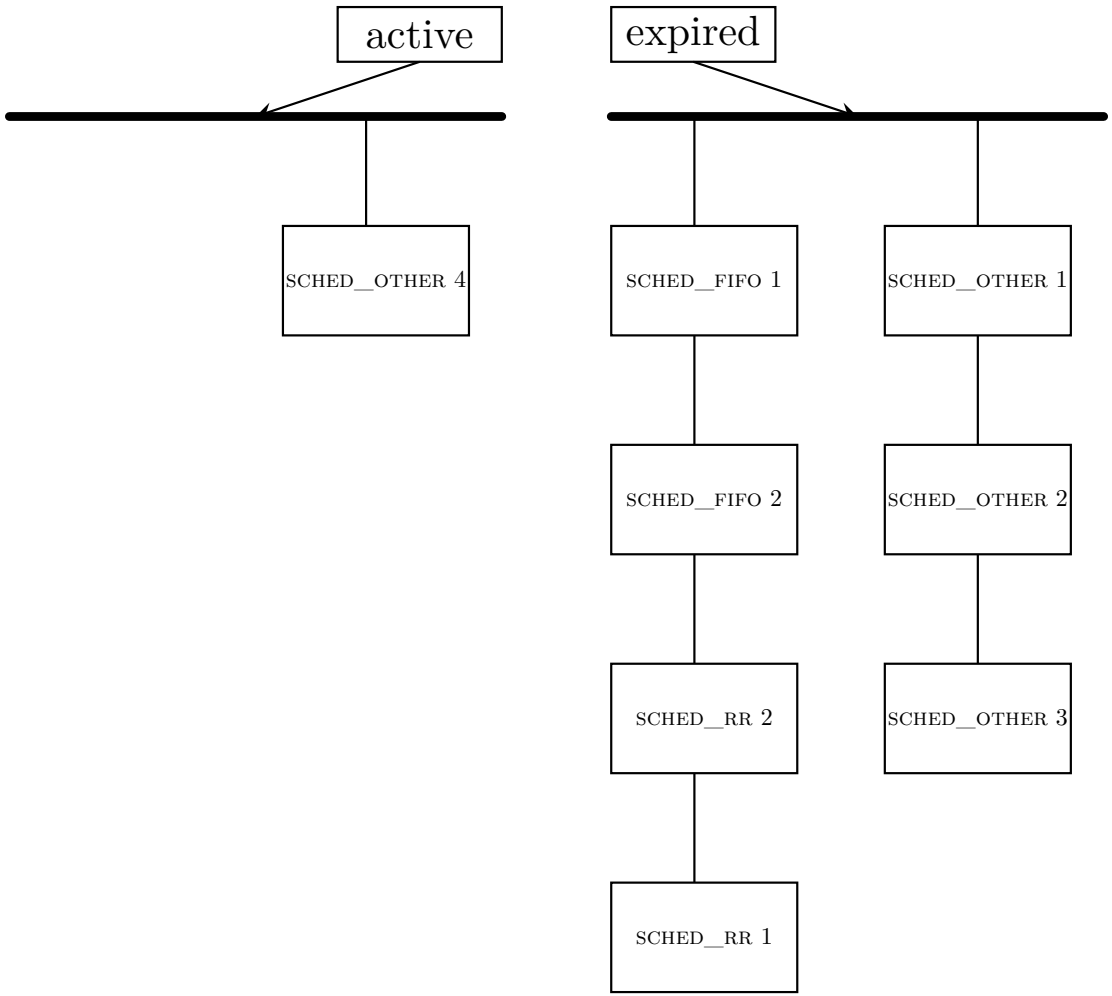
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



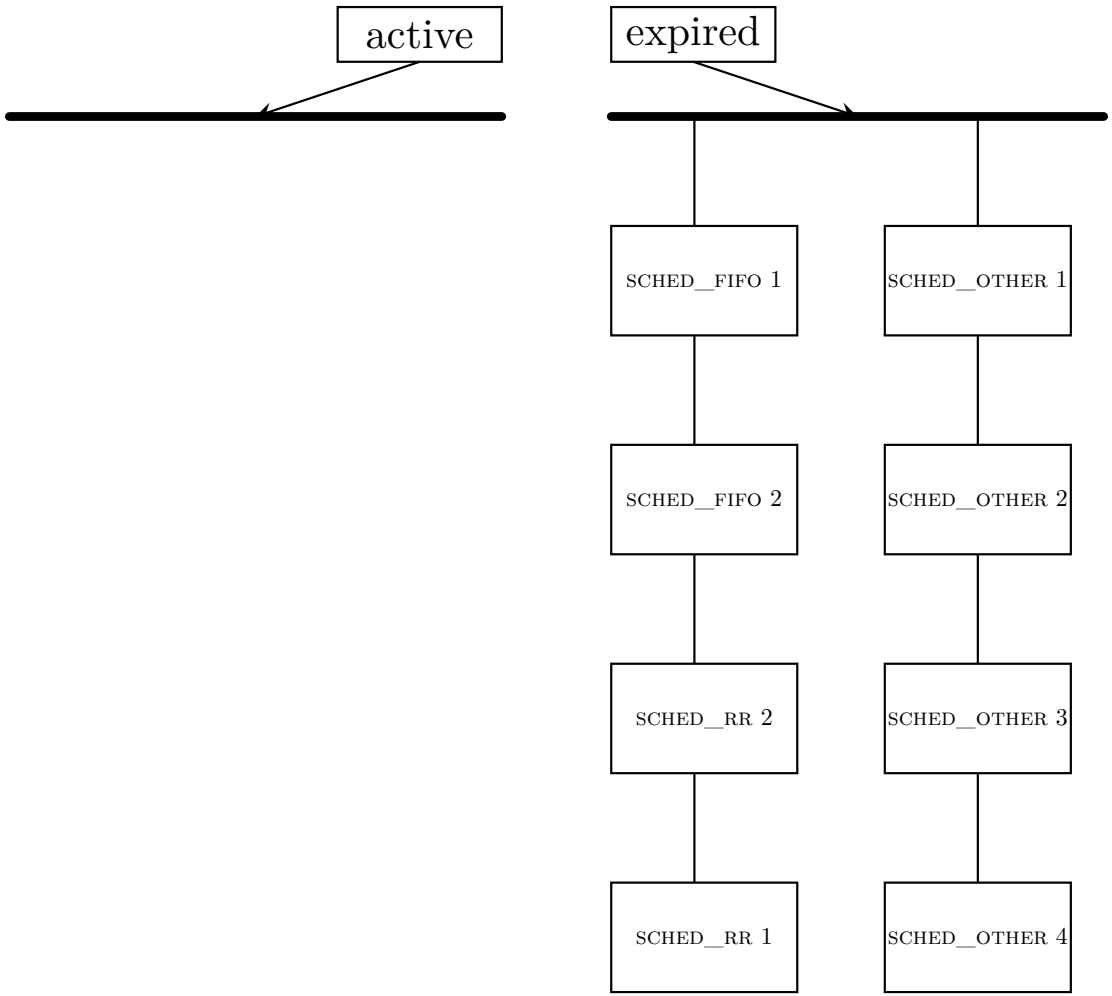
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



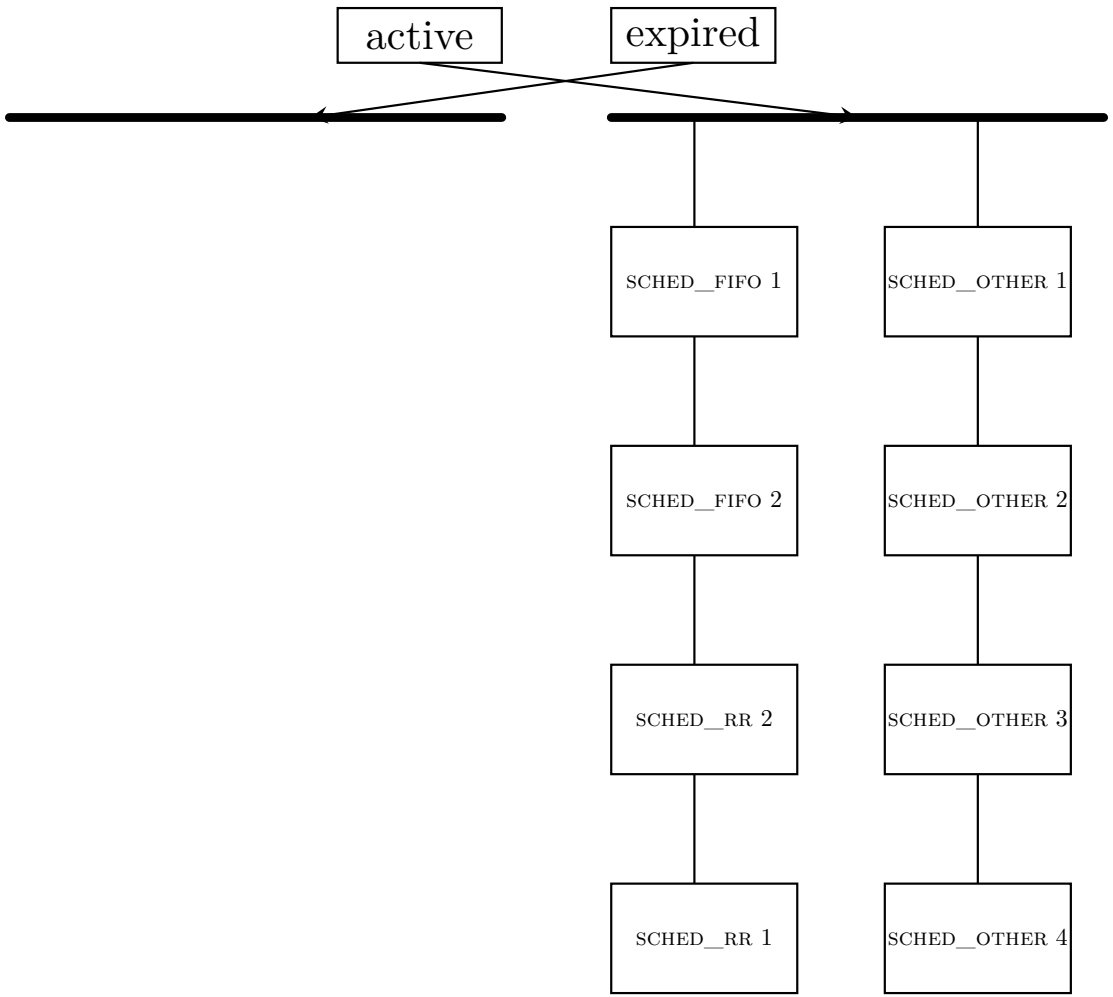
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



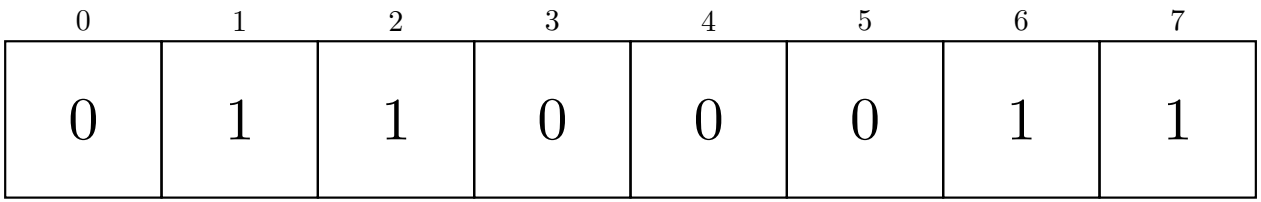
Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



Rysunek 2: Tablice priorytetów planisty O(1) (ilustracja poglądowa)

Planista O(1)



Rysunek 3: Przykładowa bitmapa (ilustracja poglądowa)

Planista O(1)

Przełączanie kontekstu

Algorytm szeregowania O(1) jest zrealizowany w funkcji `schedule()`. Jej głównym zadaniem jest wybranie kolejnego procesu do wykonania, spośród tych, które czekają na przydział CPU, ale oprócz tego wywołuje ona funkcję `context_switch()`, która dokonuje przełączenia kontekstu. Kod funkcji `schedule()` jest stosunkowo prosty, całkowicie niezależny od architektury CPU i efektywny. Czas wyboru kolejnego procesu do wykonania nie zależy od całkowitej liczby procesów gotowych do działania.
Z kolei kod funkcji `context_switch()` i jego efektywność są zależne od architektury procesora. Zazwyczaj przełączanie kontekstu zawiera więcej czasu w przypadku procesorów RISC niż CISC. Związane jest to z liczbą rejestrów, których stan należy zapamiętać lub przywrócić w trakcie tej operacji.

Planista O(1)

Szeregowanie procesów typu SCHED_OTHER

Jądro może zmienić priorytet zwykłego procesu, w zależności od poziomu jego interaktywności w poprzedniej rundzie szeregowania. Priorytety mało interaktywnych procesów są zmniejszane o +5, a priorytety bardziej interaktywnych podnoszone o −5. Linux stosuje heurystyki, aby wyznaczyć ten poziom dla każdego z procesów. Wyliczenia te bazują na stosunku całkowitego czasu kiedy proces faktycznie korzystał z CPU do całkowitego czasu spędzonego przez ten proces w stanie oczekiwania. Im więcej proces oczekuje, tym bardziej jest interaktywny. Procesy interaktywne otrzymują dłuższe kwanty czasu. Domyślny kwant, dla poziomu uprzejmości 0, wynosi $100ms$, najdłuższy dostępny dla zwykłego procesu (poziom uprzejmości −20) to $200ms$, a najkrótszy to $10ms$ (poziom uprzejmości 19).

Planista O(1)

Szeregowanie procesów typu SCHED_OTHER

Jeśli proces ma wysoki poziom interaktywności, to po wykorzystaniu kwantu czasu nie staje się automatycznie przeterminowany, ale otrzymuje ten sam kwant czasu i trafia z powrotem do tablicy procesów aktywnych. Jednakże to rozwiązanie może powodować głodzenie procesów, które już zostały przeterminowane. Aby temu zapobiec, jądro okresowo uruchamia makro `EXPIRED_STARVING()`, które sprawdza, czy te procesy nie są głodzone.

Planista O(1)

Nowy proces

Potomek otrzymuje połowę kwantu czasu, który pozostał rodzicowi. Taka sama ilość czasu jest także przyznawana jego rodzicowi. Po tym jak oba procesy ulegną przeterminowaniu jest wyznaczany dla nich nowy priorytet i tym samym przydzielany nowy kwant czasu.

Planista O(1)

Procesy oczekujące

Proces nie musi od razu wykorzystać całego kwantu czasu, który został mu przydzielony. Może na przykład aktywować wywołanie systemowe i oczekiwać na jego wynik. W tym przypadku nie może znajdować się w stanie TASK_RUNNING oraz w kolejce `runqueue`. W związku z tym, musi zostać przeniesiony do odpowiedniej kolejki procesów oczekujących, typu `wait_queue_head_t` (szczegółowy opis w piątej instrukcji do laboratoriów). Wywoływana jest wtedy funkcja `schedule()`, która wybiera inny proces do wykonania. Uśpiony proces może być wybudzony, np. z użyciem funkcji `wake_up()`, kiedy nastąpi zdarzenie na które on oczekuje. W takim przypadku jest przenoszony do kolejki `runqueue`, gdzie przysługuje mu część kwantu czasu, której jeszcze nie wykorzystał. Jeśli wybudzony proces ma wyższy priorytet od bieżącego, to ustawiana jest flaga `need_resched`.

Planista O(1)

Wywłaszczanie procesów

Proces jest wywłaszczany, gdy ustawiona jest flaga `need_resched`. Jest to jeden z bitów w polu `flags` struktury `thread_info`. W wyniku jej ustawienia wywoływana jest funkcja `schedule()`, która wybiera następny proces do wykonania i wywołuje `context_switch()` przełączającą bieżący i następny proces poprzez zamianę odwzorowania ich pamięci wirtualnej (w tym celu wywołuje `switch_mm()`) i kontekstu CPU. To ostatnie zadanie jest wykonywane przez funkcję `switch_to()` (wywoływaną z poziomu `context_switch()`), która również zachowuje stos jądra i zwartość rejestrów dla bieżącego procesu.

Planista O(1)

Wywłaszczanie procesów

Proces może być wywłaszczony kiedy sterowanie wraca do przestrzeni użytkownika, po wykonaniu wywołania systemowego lub obsługi przerwania. Wątek jądra może być wywłaszczony kiedy sterowanie wraca z obsługi przerwania, kiedy jego licznik `preempt_count` (jedno z pól struktury `task_info`) jest ustawiony na 0, kiedy bezpośrednio wywoła funkcję `schedule()` lub kiedy rozpoczyna oczekiwanie na dowolne zdarzenie.

Planista O(1)

Szeregowanie w systemach wieloprocesorowych

Planista O(1) obsługuje również szeregowanie w systemach wieloprocesorowych. W takich komputerach niektóre procesy mogą być powiązane z jednym i tylko jednym procesorem, ale zazwyczaj większość z nich może wykonywać się na dowolnym z dostępnych procesorów. W tym przypadku niekiedy konieczne jest zrównoważenie obciążenie tych CPU. To oznacza, że niektóre z procesów mogą być przeniesione z kolejki `runqueue` jednego z procesorów do kolejki innego. Tym zajmuje się funkcja `load_balance()` wywoływana przez jądro, gdy jedna z kolejek `runqueue` staje się pusta. Jest ona również okresowo uruchamiana z poziomu przerwania zegarowego. W tym przypadku dokonuje przeniesienie procesów, jeśli jedna z kolejek `runqueue` jest o 25% dłuższa niż pozostałe.

API planisty dla przestrzeni użytkownika

W przestrzeni użytkownika dostępne są funkcje pozwalające procesom zmienić parametry szeregowania (niektóre zmiany wymagają dodatkowych uprawnień):

- `nice()` — ustawia poziom uprzejmości,
- `sched_setscheduler()` — ustawia politykę szeregowania,
- `sched_setparam()` — ustawia parametry polityki szeregowania,
- `sched_get_priority_max()` — zwraca maksymalny priorytet dostępny w danej polityce szeregowania,
- `sched_get_priority_min()` — zwraca minimalny priorytet dostępny w danej polityce szeregowania,
- `sched_rr_get_interval()` — zwraca długość kwantu czasu dla procesu typu SCHED_RR,
- `sched_setaffinity()` — wykonuje powiązanie procesu lub wątku z określonymi procesorami,

API planisty dla przestrzeni użytkownika

- `sched_getaffinity()` — zwraca bitmapę, która określa na których procesorach proces lub wątek może być wykonywany,
- `sched_yield()` — powoduje, że proces zrzeka się CPU.

API planisty dla przestrzeni użytkownika

W przypadku planisty O(1) wywołania funkcji `sched_yield()` może być kosztowne dla procesu, ponieważ musi on czekać aż do następnej rundy szeregowania, aby móc zostać wykonanym (otrzymać CPU). Dzieje się tak, ponieważ w wyniku działania tej funkcji proces ulega przeterminowaniu. Nie jest ona dostępna dla wątków jądra, ale zamiast niej mogą one użyć funkcji jądra o nazwie `yield()`, która działa podobnie.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!