L13Z2

Linux wyróżnia trzy klasy wątków:

- Wątki czasu rzeczywistego szeregowanie zgodnie z kolejką FIFO
- Wątki czasu rzeczywistego szeregowane cyklicznie
- Podział czasu

Wątki czasu rzeczywistego szeregowane zgodnie z kolejką FIFO mają najwyższy priorytet i ustępują pierwszeństwa tylko nowo dodanym wątkom z tej samej grupy (kolejka priorytetowa???).
Wątki czasu rzeczywistego szeregowane cyklicznie (round robin) różnią się tym, że mają przypisane kwanty czasu i są pozbawione czasu procesora według wskazań zegara. Jeśli wiele wątków z tej grupy jest gotowych do wykonania, każdy z nich otrzmuje kolejno swój kwant czasu, po czym rafia na koniec listy równorzędnych wątków.
Przytoczone algotytmy szeregowania nie gwarantują wykonywania zadań w określonych terminach.
Wymienione klasy mają po prostu wyższy priorytet niż wątki należące do standardowej klasy z podziałem czasu. Wątki czasu rzeczywistego są wewnętrznie reprezentowane z priorytetami od 0 do 99 (0 to najwyższy priorytet). Wątki spoza klasy czasu rzeczywistego mają przypisane priorytety z przedziału 100 do 139.
Im wyższy priorytet (niższa liczba) tym dłuższy kwant czasu otrzymuje proces.

L13Z3

Przeczytaj (Stallings) lub §10.3.4 (Tanenbaum) o algorytmie Completely Fair Scheduler stosowanym w bieżących wersjach jądra Linux. Opowiedz o niedostatkach planisty O(1) i implementacji idei ***sprawiedliwego szeregowania*** *(ang. fair-share scheduling). W jaki sposób algorytm CFS wybiera kolejne zadanie do uruchomienia? W jaki sposób nalicza wirtualny czas wykonania vruntime, aby wymusić sprawiedliwy przydział czasu procesora?*

runqueue – kluczowa struktura danych używana przez program szeregujący do zarządzania wszystkimi zadaniami w systemie możliwymi do uruchomienia i do wybierania tego, które ma być uruchomione jako następne. Struktura ta jest powiązana z każdym procesorem w systemie.

CFS – główną koncepcją tego algorytmu są drzewa czerwono-czarne. Zadania są uporządkowane w drzewie na podstawie czasu, przez jaki realizowane są przez procesor CPU. Ten czas jest określany jako *vruntime*. Czas realizacji zadań jest wyliczany z nanosekundową dokładnością. Każdy wewnętrzny węzeł drzewa odpowiada jednemu zadaniu. Węzły potomne lewej strony odpowiadają zadaniom, które dotychczas wymagały mniej czasu procesora i w związku z tym zostaną zaplanowane wcześniej, natomiast węzły potomne po prawej stronie to te, które do tej pory wymagały więcej czasu procesora. Liście nie odgrywają żadnej roli w szeregowaniu.

Algorytm można streścić następująco: CFS zawsze wyznacza do uruchomienia to zadanie, które miało przydzieloną najmniejszą ilość czasu procesora – zazwyczaj jest to najbardziej skrajny węzeł po lewej stronie drzewa. Okresowo algorytm zwiększa wartość *vruntime* odpowiadającą zadaniu na podstawie czasu, przez jaki już działało, i porównuje tę wartość z bieżącym skrajnym węzłem po lewej stronie drzewa. Jeżeli uruchomione zadanie ma nadaj mniejszą wartość *vruntime,* to będzie ono kontynuowane. W przeciwnym razie zostanie wstawione w odpowiednim miejscu drzewa, a procesor zostanie przydzielony do zadania odpowiadającego nowemu skrajnemu węzłowi po lewej stronie.

Algorytm szeregujący uwzględnia tylko zadania wykonywalne reprezentowane w odpowiedniej strukturze *runqueue*. Zadania, których realizacja jest niemożliwa i które oczekują na rozmaite operacje I/O lub inne zdarzenia jądra, są reprezentowane w innej strukturze: *waitqueue.* Struktura ta jest powiązana z każdym zdarzeniem, na które mogą oczekiwać zadania.

Opowiedz o niedostatkach planisty O(1) i implementacji idei ***sprawiedliwego szeregowania*** *(ang. fair-share scheduling)*

Planista O(1) – popularny program szeregujący we wczesnych wersjach linuksa. Był w stanie wybierać i umieszczać zadania w kolejce *runqueue* w stałym czasie. W programie tym *runqueue* była zorganizowana w postaci dwóch tablic: *active* oraz *expired.*

Algorytm planisty O(1) można opisać następująco: planista wybiera zadanie z tablicy aktywnych zadań o najwyższych priorytetach. Jeżeli przedział czasowy danego zadania wygasł, zadanie jest przenoszone na listę zadań, które wykorzystały już swoje kwanty. Jeżeli przed wygaśnięciem danego kwantu zadanie jest zablokowane (np. w oczekiwaniu na zdarzenie I/O) po wystąpieniu oczekiwanego zdarzenia wykonywanie tego zadania zostaje wznowione, samo zadanie ponownie umieszczone w oryginalnej tablicy aktywnych zadań, a jego kwant czasu pomniejszony o już wykorzystany czas procesora. Kiedy żadna z aktywnych tablic nie zawiera już oczekujących zadań, algorytm szeregujący ogranicza się do takiej zamiany wskaźników, aby tablice zadań, które wcześniej wyczerpały swój czas, stały się aktywne (i odwrotnie). Opisana metoda gwarantuje, że zadania o niskim priorytecie nie będą oczekiwać w nieskończoność.

Niedostatki planisty (*Stallings §10.3*):

- Szedżuler używa jednej struktury *runqueue* dla każdego procesora nawet na systemie wieloprocesorowym (SMP). Efekt – problemy z keszowaniem danych. Przykład: załóżmy, że zadanie jest wykonywane przez CPU1 i jego dane są w keszu tego procesora. Jeżeli zadanie zostanie przekierowane do CPU2 dane te będę musiały zostać unieważnione w CPU1 i przeniesione do CPU2 – niska efektywność.
- Szedżuler używa jednej blokady na *runqueue*. Przez to proces wybierania zadania, które ma się wykonywać blokuje innym procesorom dostęp do struktury i nakazuje czekanie z podejmowaniem decyzji.
- Niemożliwe są wywłaszczenia – prowadzi to do sytuacji, w której zadanie o wyższym priorytecie musi czekać, aż zadanie o niższym priorytecie się zakończy.

Sprawiedliwe szeregowanie (*eng. wiki)*- algorytm dyspozytora, w którym użycie procesora jest rozdzielane pomiędzy użytkowników lub grupy, a nie równo pomiędzy procesy.

Jedną z typowych metod implementacji sprawiedliwego szeregowanie jest rekursywne stosowanie algorytmu karuzelowego (round robin) na każdej warstwie abstrakcji (procesy, użytkownicy, grupy itd.), Kwant czasu wymagany przez robina jest arbitralny, ponieważ każdy równy podział czasu da ten sam rezultat.

W jaki sposób (CFS) nalicza wirtualny czas wykonania vruntime, aby wymusić sprawiedliwy przydział czasu procesora?

Aby uwzględnić różnice w priorytetach zadań CFS modyfikuje tempo, z jakim upływa wirtualny czas dla danego zadania, gdy jest ono uruchomione na procesorze. Dla zadań o niższym priorytecie czas mija szybciej, a wartość powiązanej z nimi zmiennej *vruntime* zwiększa się szybciej i w zależności od innych zadań istniejących w systemie szybciej stracą once czas procesora i zostaną ponownie wstawione do drzewa.