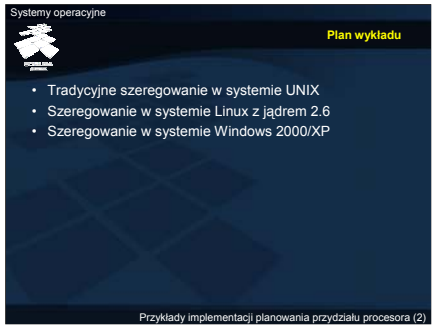
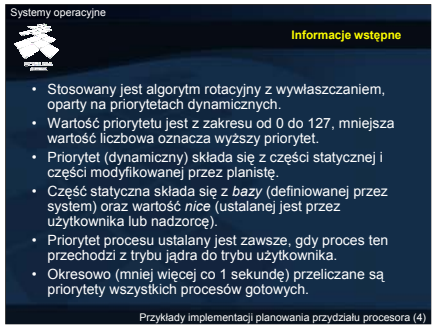
Celem wykładu jest przedstawienie podejść do planowania przydziału procesora w najbardziej popularnych systemach operacyjnych: tradycyjnym systemie UNIX, systemie Linux i systemie Windows 2000/XP. W ramach prezentowanych podejść omawiane są struktury danych oraz koncepcje realizacji algorytmów przeliczania priorytetów, wyboru procesów lub wątków do wykonania oraz przeciwdziałania głodzeniu.



Wszystkie prezentowane podejścia stosują algorytm rotacyjny z wywłaszczaniem opartym na priorytetach dynamicznych, chociaż w systemach Linux i Windows wydzielone jest pasmo priorytetów czasu rzeczywistego, gdzie stosowane są priorytety statyczne. W wszystkich tych systemach dąży się do maksymalnego wykorzystania zasobów, w związku z czym jest preferencja dla zadań ograniczonych wejściem-wyjściem. Sposób faworyzowania zadań ograniczonych wejściem-wyjściem w każdym systemie jest jednak zupełnie inny.

Cechą wspólną implementacji prezentowanych rozwiązań jest stosowanie kolejki priorytetowej, czyli tablicy kolejek, których nagłówki lokalizowane są w tablicy na pozycjach odpowiadających poziomom priorytetów. Specyfiką rozwiązania w systemie Linux jest operowanie dwoma takimi kolejkami.

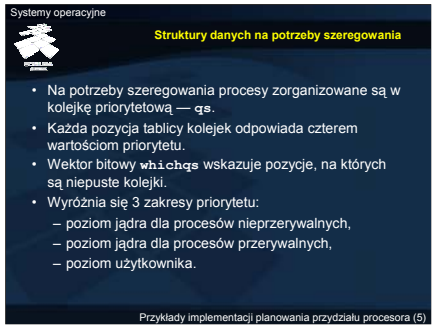

Systemy operacyjne
Szeregowanie w systemie UNIX
Przykłady implementacji planowania przydziału procesora (3)



Algorytm rotacyjny z wywłaszczaniem polega na tym, że po upływie kwantu czasu (około 1 sekundy) następuje przełączenie kontekstu na inny proces o takim samym priorytecie. Jeśli nie ma procesu o takim samym priorytecie, proces kontynuuje działanie przez kolejny kwant czasu. Jeśli jednak przed upływem kwantu czasu pojawi się proces gotowy o wyższym priorytecie nastąpi wywłaszczenie, tym samym również przełączenie kontekstu. Wyjątkiem od tej reguły jest przypadek wykonywania programu jądra. Jądro w **tradycyjnym** systemie UNIX jest niewywłaszczalne, co oznacza, że nie można przerwać procesu, który działa w trybie jądra, nawet jeśli pojawi się proces o wyższym priorytecie. Proces może oczywiście wejść w stan oczekiwania, wówczas jądro stanie się dostępne dla innych procesów. Przełączenie kontekstu następuje zatem w następujących przypadkach:

- bieżący proces wchodzi w stan oczekiwania w wyniku zażądania dostępu do zasobu (operacji wejścia-wyjścia, synchronizacji itp.) lub kończy swoje działanie,
- w wyniku przeliczenia priorytetów jakiś proces gotowy uzyskuje wyższy priorytet niż proces bieżący,
- następuje wejście w stan gotowości (obudzenie) procesu o wyższym priorytecie.

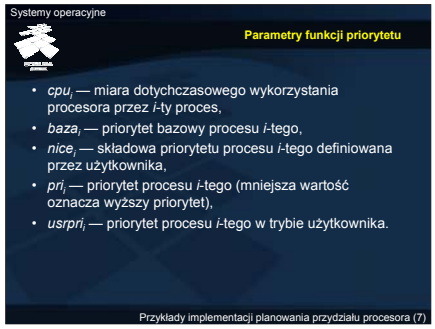
Zakres priorytetu od 0 do 49 zarezerwowany jest dla procesów w trybie jądra, a pozostałe wartości są priorytetami procesów w trybie użytkownika.



W systemie dąży się do tego, aby procesu jak najszybciej opuszczały tryb jądra, gdyż jest to zasób krytyczny dla funkcjonowania systemu i niewywłaszczalny. Dlatego większość procesów, znajdujących się w trybie jądra jest uśpiona w oczekiwaniu na jakieś zdarzenia (np. zakończenie operacji wejścia-wyjścia). Jeśli jakiś proces jest budzony w wyniku zajścia oczekiwanego zdarzenia, otrzymuje on tzw. priorytet uśpienia tego zdarzenia, który jest oczywiście priorytetem poziomu jądra. Na przykład: priorytet uśpienia dla wyjścia z terminala wynosi 28, a priorytet uśpienia dla operacji dyskowej wynosi 20. Obudzony w ten sposób proces ma wyższy priorytet niż wykonywany proces poziomu użytkownika, więc następuje wywłaszczenie. Gdyby jednak wykonywany był kod jądra przez jakiś proces, wznowienie procesu oczekującego nastąpi dopiero po zwolnieniu jądra (opuszczeniu bądź wejściu w oczekiwanie) przez proces wykonywany.

Struktura `qs` jest tablicą kolejek, czyli odpowiednich wskaźników. Sama kolejka jest zbudowana na liście dwukierunkowej i łączy deskryptory procesów, dokładnie struktry `proc`, stanowiące część pełnego deskryptora procesu.

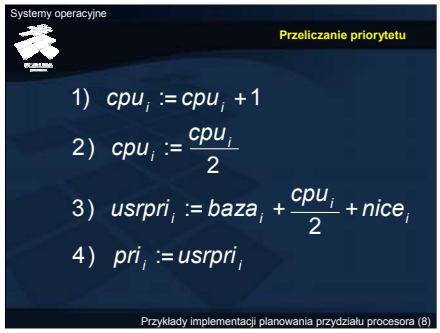
W systemie jest 1 proces o priorytecie z przedziału 0 – 3, 2 procesy o priorytecie z przedziału 4 – 7, 3 procesy o priorytecie z przedziału 8 – 11, nie ma procesu o priorytecie z przedziału 12 – 15 itd. Wektor `whichqs` wskazuje pozycje niepuste i umożliwia szybką lokalizację kolejki o najwyższym priorytecie, w które jest jakiś proces.



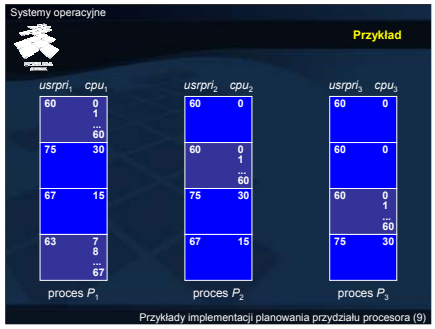
Parametr *cpu* jest wartością liczbową, która jako miara wykorzystania procesora jest zwiększana, gdy proces jest wykonywany oraz stopniowo zmniejszana, gdy proces nie wykorzystuje procesora.

Wartość *nice* może być zwiększana przez użytkownika zwykłego, co skutkuje zmniejszeniem priorytetu, natomiast zmniejszana (w celu zwiększenia priorytetu) może być przez nadzorcę. Domyślnie wartość *nice* ustawiana jest na 20.

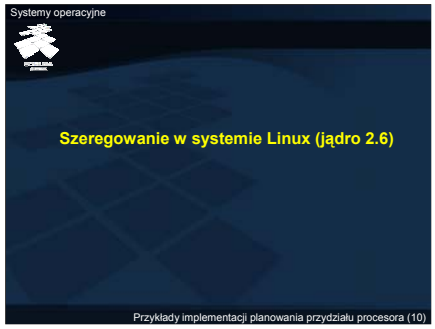
Wyodrębnienie parametrów *pri* i *usrpri* wynika z różnicy wartości priorytetu w trybie użytkownika i w trybie jądra. W trybie jądra proces otrzymuje priorytet wyższy, wynikający np. z rodzaju operacji wejścia-wyjścia, jakiej zażądał. Przed powrotem do trybu użytkownika priorytet musi jednak wrócić do poprzedniej wartości. W trybie użytkownika zatem wartości parametrów *pri* i *usrpri* są takie same, a w trybie jądra wartość parametru *pri* jest mniejsza (wyższy priorytet), niż *usrpri*.

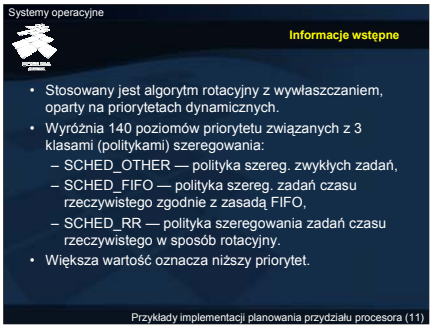


1) Każdy takt zegara (lub co któryś, np. co 4, zależnie od implementacji) zwiększa wartość *cpu* bieżącego (wykonywanego) procesu.
2) Przy każdym przeliczaniu priorytetów procesów gotowych (co około 1 sekundę) wartość ta, niezależnie od tego, czy była zwiększana, czy nie, dzielona jest przez 2 (współczynnik zaniku).
3) Przy każdym przeliczaniu priorytetów, po zmniejszeniu wartości parametru *cpu* następuje przeliczeniem priorytetu użytkownika, zgodnie z podanym wzorem. Na priorytet ten składa się więc:
   - *baza* — umożliwiająca podział na pasma priorytetów,
   - *cpu* — miara wykorzystania procesora,
   - *nice* — wartość określana przez użytkownika.
4) Jeśli proces jest w trybie użytkownika (jego priorytet *pri* jest na poziomie użytkownika) następuje zmiana parametru *pri*. Konsekwencją tej zmiany może być jednak konieczność umieszczenia procesu w innej kolejce. Operacja przebiega zatem tak, że najpierw proces jest usuwany z kolejki, następuje podstawienie nowej wartości parametru *pri*, a następnie ponowne powiązanie deskryptora w kolejce na właściwej pozycji. Taki sposób przeliczania ogranicza skalowalność, gdyż czas potrzebny na tę operację jest proporcjonalnym do liczby procesów. Zwiększająca się liczba procesów powoduje więc wydłużenie tego czasu.

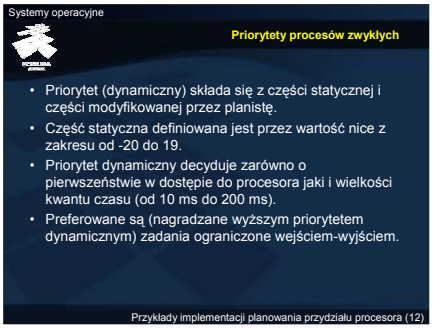


Przykład obrazuje zmiany priorytetów trzech procesów w systemie. Początkowy priorytet (załóżmy, że jest to *baza* + *nice*) każdego z procesów wynosi 60, a miara wykorzystania procesora wynosi 0. 60 razy na sekundę następuje tak zegara, który zmienia parametr *cpu*. Proces P_1 wykorzystuje jako pierwszy procesor (oś czasu skierowana jest w dół,). Po 60 taktach następuje przeliczenie priorytetów. Parametr cpu_1 osiąga wówczas wartość 60 i jest dzielony przez 2. Podobnie jest to robione dla pozostałych procesów, ale dla nich wartość parametru *cpu* jest cały czas 0, gdyż nie wykorzystywały one procesora. Połowa przeliczonej wartości parametru *cpu* dodawana jest do statycznej części priorytetu procesu P_1, co daje wynik 75. Wyższy priorytet mają zatem procesy P_2 i P_3, czyli następuje przełączenie kontekstu na proces P_2. Po kolejnych 60 taktach następuje przeliczenie priorytetów. W międzyczasie wartość cpu_2 zwiększyła się do 60 i nie zmieniła się w przypadku pozostałych procesów. Po przemnożeniu przez współczynnik zaniku cpu_1 wynosi 15, cpu_2 — 30, a cpu_3 — cały czas 0. Po dodaniu połowy wartości parametru *cpu* do statycznej części otrzymujemy $usrpri_1$ o wartości 67, $usrpri_2$ o wartości 75 i $usrpri_3$ o wartości 60. Najwyższy priorytet ma więc proces P_3, a po kolejnych 60 taktach priorytet procesu P_1 ponownie wzrośnie do największego w systemie (wartość 63).

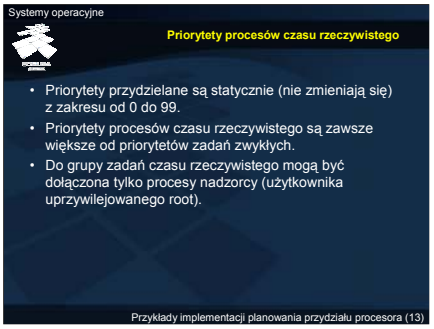

Systemy operacyjne
Szeregowanie w systemie Linux (jądro 2.6)
Przykłady implementacji planowania przydziału procesora (10)



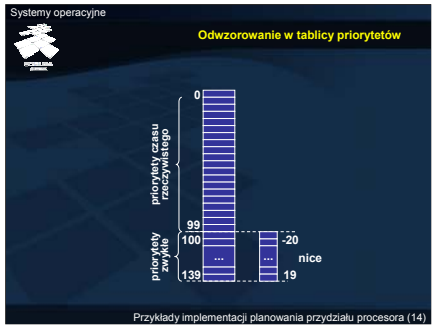
W systemie Linux, podobnie jak w innych współczesnych systemach operacyjnych (Windows 2000, współczesne implementacje systemów uniksopodobnych) uwzględniono szeregowanie procesów czasu rzeczywistego. Są to rozwiązania spełniające wymagania tzw. łagodnego (tolerancyjnego) czasu rzeczywistego, co znaczy, że system próbuje wykonywać je szybciej niż inne zadania, ale nie gwarantuje ich zakończenia w określonym czasie (przed linią krytyczną). Priorytety dla zadań czasu rzeczywistego (czyli klas SCHED_FIFO i SCHED_RR) są zawsze wyższe — mają mniejsze wartości — niż priorytety zadań zwykłych.



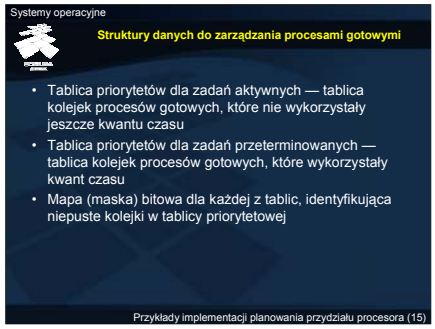
Wartość *nice* ma podobny charakter jak w systemie UNIX. Określa się jako składową statyczną priorytetu. W rzeczywistości nie jest ona liczbową wartością priorytetu, bo jest odwzorowywana odpowiednie na wartość z zakresu 100 – 139. W przeciwieństwie do tradycyjnego rozwiązania w systemie UNIX Linux oferuje zmiennej długości kwant czasu.



W przypadku procesów zwykłych priorytet może się zmieniać, podobnie jak to było w systemie UNIX, ale zasady zmiany są inne. Dlatego ogólnie określa się go jako dynamiczny (chociaż ma również składnik statyczny). Priorytety zadań czasu rzeczywistego nie zmieniają się. Biorąc pod uwagę fakt, że priorytety te są wyższe, niż priorytety procesów zwykłych, istnieje ryzyko, że jeśli proces czasu rzeczywistego otrzyma procesor, zagłodzi inne zadania. Dlatego możliwość zmiany priorytetu (i tym samym strategii szeregowania) w zakresie czasu rzeczywistego zastrzeżona jest dla nadzorcy.



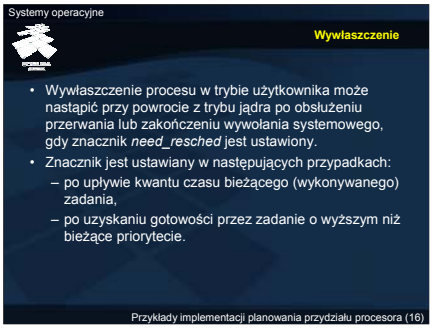
Tablica priorytetów obejmuje 140 pozycji o wartościach od 0 (najwyższy priorytet) do 139 (najniższy priorytet). Wartości z zakresu 1 – 99 odpowiadają procesom czasu rzeczywistego, a większe wartości odpowiadają procesom zwykłym i odwzorowane są na odpowiednie wartości *nice*.



Mechanizm szeregowania w jądrze 2.6 (2.5) został dość istotnie przebudowany w stosunku do rozwiązań wcześniejszych i również rozwiązań stosowanych w innych systemach operacyjnych. Ma to swoje skutki między innymi w strukturach danych. Podobnie, jak w przypadku tradycyjnego systemu UNIX, a także Windows 2000/XP podstawową strukturą jest kolejka priorytetowa, zwana *tablicą priorytetów*. W przeciwieństwie jednak do systemu UNIX nie ma tu przeliczania priorytetu każdego procesu co określony czas, priorytet dynamiczny wyznaczany jest na bieżąco, np. po zakończeniu kwantu czasu. Takie podejście mogłoby jednak oznaczać głodzenie, bo nowo wyliczony priorytet mógłby być większy niż priorytet innych procesów gotowych i oczekujących już przez dłuższy czas na procesor. W tradycyjnym systemie UNIX tego problemu unikało się w ten sposób, że co sekundę były przeliczane priorytety wszystkich procesów, a w wyniku tego przeliczenia karane były procesy, które dużo korzystały z procesora i nagradzana takie, które dłużej czekały.

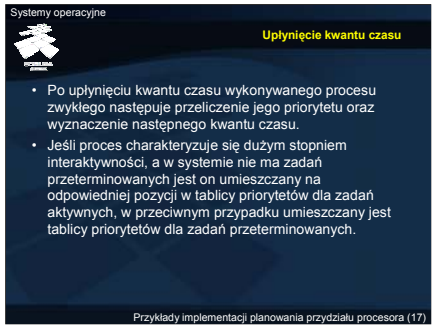
W celu niedopuszczenia do głodzenia w systemie Linux stosowane są dwie kolejki priorytetowe. Jedna kolejka jest dla zadań, które mają prawo korzystać z procesora, zwanych zadaniami aktywnymi, a druga jest dla tzw. zadań przeterminowanych, które chwilowo wykorzystały już przydzielone im limity.

Podobnie, jak w systemie UNIX, w celu szybkiej identyfikacji pozycji niepustych w kolejkach priorytetowych, wykorzystywany jest wektor bitowy. W przeciwieństwie jednak do UNIXa, każdy bit odpowiada dokładnie jednej pozycji, czyli jednemu priorytetowi.



Jeśli przetwarzanie odbywa się w trybie jądra, to wywłaszczenie (przełączenie kontekstu) nie może nastąpić w dowolnym momencie, choć ogólnie jądro systemu Linux jest w pewnych warunkach wywłaszczalne. Wykrycie potrzeby przeszeregowania w trakcie wykonywania kodu jądra (np. w trakcie obsługi przerwania zegarowego), powoduje ustawienie znacznika *need_resched* i kontynuację przetwarzania w trybie jądra. Dopiero przy powrocie do trybu użytkownika następuje sprawdzenie znacznika podjęcie decyzji o przełączeniu kontekstu.

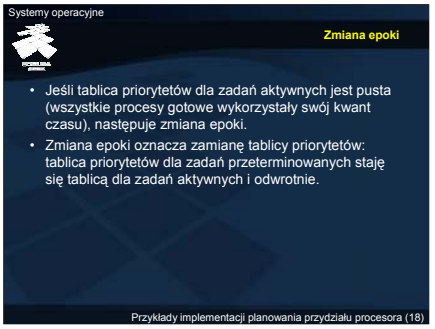
Wywłaszczenie nastąpi, gdy będzie gotowe zadania o nie mniejszym priorytecie.



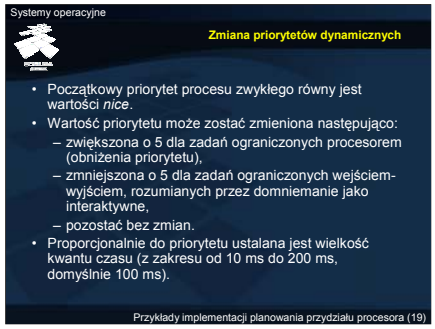
Przeliczenie priorytetu i związanego z nim kwantu czasu procesu następuje zaraz po upłynięciu kwantu czasu, a nie w wyniku specjalnej, okresowo uruchamianej procedury. Bardzo często jest tak, że nie wymaga się żadnych specjalnych przeliczeń. Priorytet i kwant czasu procesu pozostają bez zmian. Należy jednak zrobić coś z procesem, który wykorzystał swój kwant, a po przeliczeniu ma dość wyskoki priorytet, żeby dać szansę przetwarzania innym procesom o niższych priorytetach. W większości przypadków proces który wykorzystał już swój kwant czasu identyfikowany jest jako przeterminowany i trafia do odpowiedniej tablicy priorytetów. Procesy w tablicy priorytetów dla zadań przeterminowanych nie są uwzględniane przez planistę, jako kandydaci do przydziału procesora, do końca tzw. *epoki*.

W szczególnych przypadkach proces, który wykorzystał już swój kwant czasu, może trafić ponownie bezpośrednio do tablicy priorytetów dla zadań aktywnych. Zależy to od stopnia interaktywności zadania oraz stanu tablicy priorytetów dla zadań przeterminowanych. Jeśli w nie ma zbyt długo oczekujących zadań przeterminowanych a stopień interaktywności wywłaszczanego procesu jest dostatecznie duży, po przeliczeniu priorytetu może on trafić na koniec kolejki właściwej dla swojego priorytetu w tablicy priorytetów dla zadań aktywnych.

Warto podkreślić, że proces, który zażąda operacji wejścia-wyjścia lub z innych powodów wejdzie w stan oczekiwania, nie traci swojego kwantu. Po uzyskaniu gotowości, ma szansę wykorzystać resztę przysługującego mu w danej epoce czasu procesora. Natomiast część kwantu czasu, pozostała do wykorzystania przez proces, jest dzielony na pół w momencie, gdy proces ten tworzy nowy proces (potomny). Połowa kwantu pozostaje więc do dyspozycji procesu macierzystego, a druga połowa jest początkowym kwantem czasu potomka.

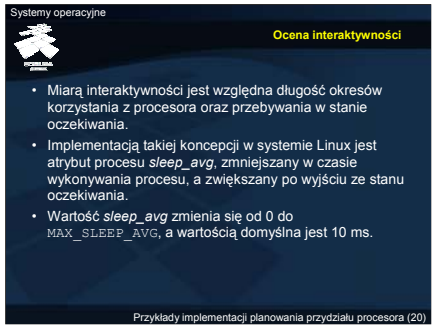


Zamiana tablic jest operacją bardzo szybką, gdyż polega na zamianie wartości dwóch wskaźników, wskaźnika na tablicę priorytetów dla zadań aktywnych i tablicę priorytetów dla zadań przeterminowanych. Wszystkie przeterminowane zadania stają się ponownie aktywne.



Priorytet dynamiczny dotyczy procesów zwykłych. Dla procesów czasu rzeczywistego jest ustalony i może być zmieniony w wyniku działań nadzorcy, a nie planisty.

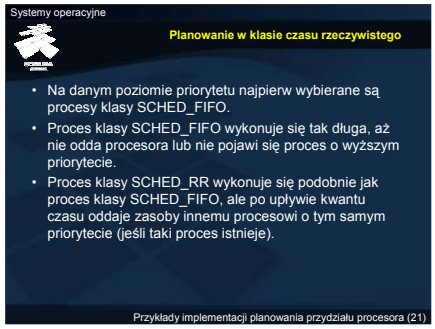
W zależności od oceny interaktywności procesu, jego priorytet dynamiczny jest odpowiednio modyfikowany. Wyznaczenie kwantu czasu sprowadza się do przeskalowania wartości priorytetu dynamicznego z przedziału –20 – 19 (albo 100 do 139) na wartość kwantu czasu z przedziału 10 ms do 200 ms z uwzględnieniem proporcjonalności odwrotnej, gdyż mniejsza wartość priorytetu to wyższy poziom preferencji w dostępie do procesora.



W czasie wykonywania procesu (gdy proces korzysta z procesora), każdy takt zegara powoduje zmniejszenie wartości zmiennej *sleep_avg* o 1. Po obudzeniu procesu w stanie oczekiwania wartość ta jest z kolei zwiększana o czas oczekiwania.

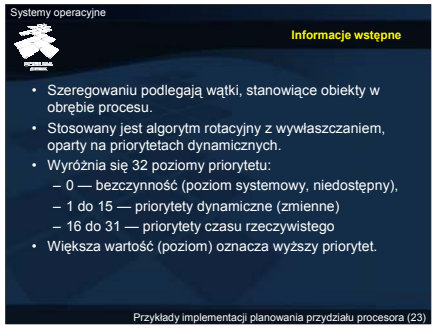
Określanie charaktery aplikacji na podstawie porównywania czasu wykorzystania procesora oraz oczekiwania na zakończenie operacji wejścia-wyjścia może niekiedy prowadzić błędów:

- Przykład 1: aplikacja multimedialna. Zastosowanie multimedialne wymagają na ogół sporej ilości czasu procesora, a mogą być wykorzystywane w bezpośredniej interakcji przez użytkownika.
- Przykład 2: proces zarządzania bazą danych. Serwer bazy danych może spędzać dużo czasu w oczekiwaniu na wykonanie operacji dyskowych, ale nie wchodzi w bezpośrednią interakcję z użytkownikiem.



Planista nie modyfikuje priorytetu procesów czasu rzeczywistego. Jeśli proces taki nie zwolni procesora (i jądra) w wyniku wejścia w stan oczekiwania, wywłaszczenie może nastąpić tylko przez proces o wyższym priorytecie (czyli inny proces czasu rzeczywistego). W przypadku klasy SCHED_RR wywłaszczenie może też nastąpić po upływie kwantu czasu, gdy jest gotowy inny proces o tym samym priorytecie. Jeśli nie ma takiego procesu, proces dotychczas wykonywany otrzymuje kolejny kwant czasu i kontynuuje działanie. Nieodpowiednie użycie klasy priorytetów czasu rzeczywistego może więc skutkować zawłaszczeniem procesora.


Systemy operacyjne
Szeregowanie w systemie Windows 2000/XP
Przykłady implementacji planowania przydziału procesora (22)

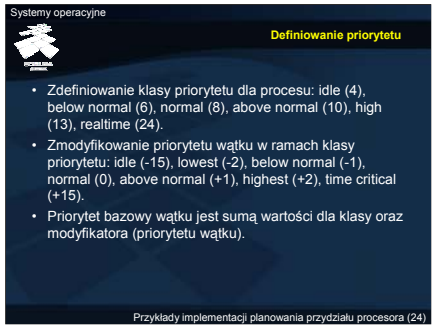


Ogólne podejście do szeregowania w Windows jest podobne do wcześniej przedstawionych. Wyróżniono dwa zakresu (pasma) priorytetów, z których wynika taż sposób zarządzania priorytetami wątków.

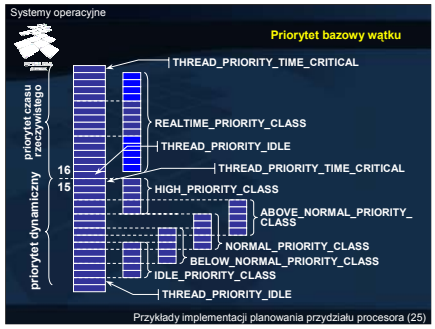
W paśmie zmiennych priorytetów stosowany jest algorytm rotacyjny oparty jest na priorytetach dynamicznych. Priorytet wątku w tym paśmie może ulec podwyższeniu po zakończeniu oczekiwania, a następnie jest stopniowo obniżany.

W paśmie priorytetów czasu rzeczywistego stosowany jest również algorytm rotacyjny, ale priorytety nie ulegają zmianie. Tak rozumiany czas rzeczywisty oznacza tylko bardziej przewidywalne zachowanie, nie gwarantuje natomiast utrzymania jakiegokolwiek reżimu czasowego.

W sumie wyróżniono 32 poziomy priorytetu, ale najniższy poziom zastrzeżony jest dla wątku bezczynności.



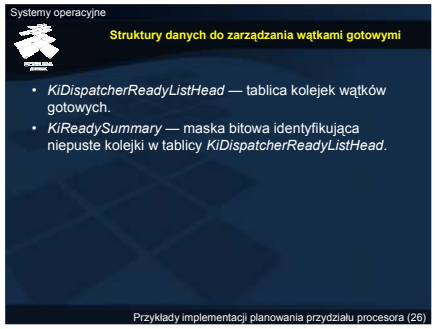
Wyróżniono kilka klas priorytetu, z których większość jest w paśmie priorytetów dynamicznych. Każda klasa reprezentowana jest przez pewną wartość priorytetu, która ustalana jest dla procesu. Wartość ta dziedziczona jest przez wątek procesu, ale jego *priorytet bazowy* może zostać w stosunku do klasy procesu skorygowany poprzez zastosowanie odpowiedniego modyfikatora. Oprócz podanych modyfikatorów w paśmie wątków czasu rzeczywistego możliwe są też wartości –7, -6, ..., -3, 3, 4, 5, 6. W wyniku zsumowania klasy priorytetu oraz modyfikatora nie można wyjść poza pasmo, czyli suma ta jest odpowiednio ograniczana.



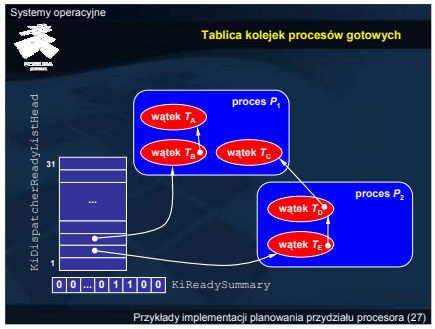
Dwa zasadnicze pasma priorytetu, decydujące o sposobie przeliczania wartości priorytetów wątków, to:

- pasmo priorytetów dynamicznych z zakresu od 1 do 15,
- pasmo priorytetów czasu rzeczywistego z zakresu od 16 do 31.

Warto zwrócić uwagę, że modyfikatory *THREAD_PRIORITY_TIME_CRITICAL* oraz *THREAD_PRIORITY_IDLE* reprezentują odpowiednio najwyższy i najniższy dostępny priorytet **w danym paśmie**. Modyfikatory takie oznaczają odpowiednio 31 i 16 w paśmie czasu rzeczywistego oraz wartości 15 i 1 w paśmie priorytetów dynamicznych, niezależnie od klasy priorytetu.

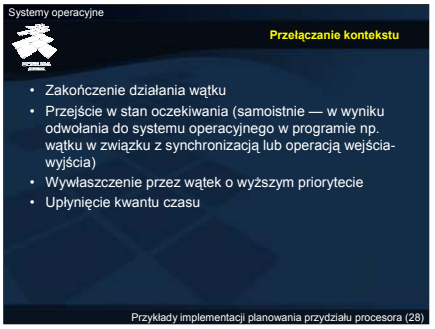


Podobnie jak w systemach UNIX i Linux podstawową strukturą danych, ułatwiającą pracę planisty — zwanego w polskich tłumaczeniach *dyspozytorem* (ang. dispatcher) — jest kolejka priorytetowa. W celu przyspieszenia wyszukiwania kolejki o najwyższym priorytecie stosowany jest wektor bitowy, wskazujący kolejki niepuste.



Przykład pokazuje stan kolejki priorytetowej, za pomocą której identyfikowanych jest pięć wątków gotowych, należących do dwóch procesów.

Dwa wątki procesu P_1 (T_A i T_B) oraz jeden wątek procesu P_2 (T_C) mają priorytet na poziomie 3, a pozostałe 2 wątki procesu P_2 (T_C i T_E) mają priorytet na poziomie 2.



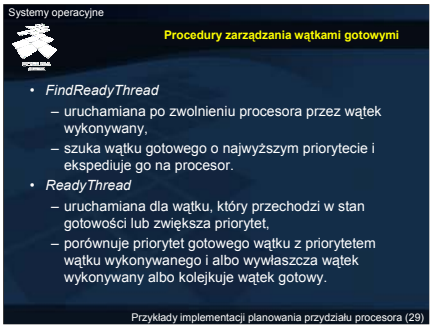
Przypadki przełączania kontekstu są typowe dla planowania rotacyjnego z wywłaszczaniem na podstawie priorytetu.

Wywłaszczenie przez wątek o wyższym priorytecie może być konsekwencją:

- wejścia wątku o wyższym priorytecie w stan gotowości,
- podwyższenia priorytetu wątku gotowego.

Wywłaszczony wątek umieszczany jest **na początku** kolejki procesów gotowych.

Upłynięcie kwantu czasu nie musi oznaczać przełączenia kontekstu, gdyż może się okazać, że nie ma w systemie wątku gotowego o równym lub wyższym priorytecie.

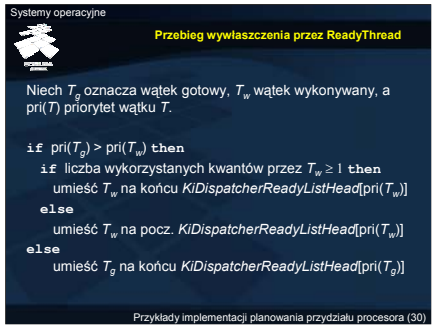


Realizacja szeregowania sprowadza się do wywołania jednej z dwóch procedur *FindReadyThread* lub *ReadyThread*.

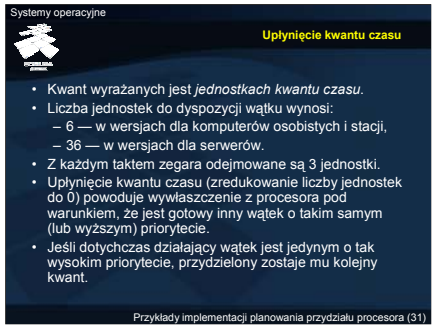
Wywołanie *FindReadyThread* oznacza, że wątek wykonywany dobrowolnie zrzekł się procesora lub upłynął przydzielony mu kwant czasu. W związku z tym konieczne jest wybranie nowego wątku do wykonania.

Procedura *ReadyThread* uruchamiana jest dla konkretnego wątku, który wchodzi w stan gotowości lub któremu podwyższono priorytet, a jej celem jest zdecydować o dalszym losie tego wątku. Zależnie od wartości priorytetów, skutkiem wykonania procedury jest:

- wywłaszczenie dotychczas wykonywanego wątku i wyekspediowanie na procesor wątku, dla którego uruchomiona została procedura, albo
- umieszczenie wątku, dla którego została uruchomiona procedura, we właściwej dla jego priorytetu kolejce.



Algorytm pokazuje zasady wywłaszczania. Jeśli gotowy wątek T_g, dla którego uruchomiona została procedura ma wyższy priorytet niż wątek wykonywany T_w, to następuje wywłaszczenie. W przeciwnym przypadku wątek T_g trafia na koniec właściwej dla jego priorytetu kolejki procesów gotowych. W przypadku wywłaszczenia wątek wywłaszczony trafia najczęściej na koniec kolejki wątków gotowych, właściwej dla swojego priorytetu. Wyjątkiem jest przypadek, gdy wątek wywłaszczony nie wykorzystał jeszcze pierwszego kwantu czasu, wówczas trafia na początek właściwej dla jego priorytetu kolejki.



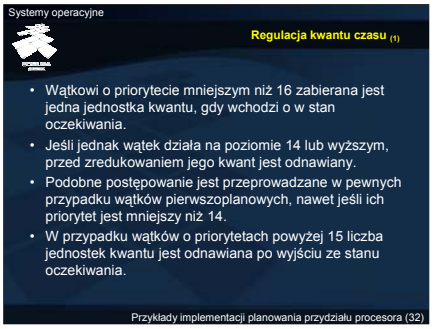
Kwant czasu nie jest wyrażany w jednostka bezpośrednio związanych z czasem. Jednostka ma związek z taktem zegara, ale również nie jest to liczba taktów, gdyż każde przerwanie zegarowe zmniejsza liczbę jednostek o 3. Liczbę jednostek można zmieniać modyfikując w rejestrze Windows wartość *HKLM\SYSTEM\CurrentControlSet*

Control\PriorityControl\Win32PrioritySeparation.

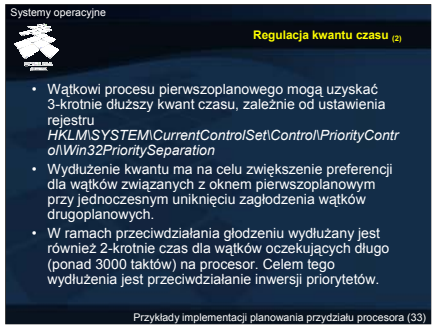
Wykorzystanie kwantu czasu skutkuje podobnymi działaniami, jak w innych podejściach z wywłaszczaniem, opartych na priorytecie, czyli rozpoczyna się poszukiwanie innego wątku do wykonania. Ponieważ jednak stosowane jest podejście priorytetowe, poszukiwany wątek nie może mieć priorytetu niższego niż wątek, który stracił kwant czasu. Jeśli takiego wątku nie ma, wątek dotychczas wykonywany otrzymuje kolejny kwant czasu.

Jeśli następuje przełączenie kontekstu, ta prawdopodobnie znaleziony został wątek o takim samym priorytecie, jak dotychczas wykonywany. Gdyby w stanie gotowości był wątek o priorytecie wyższym, wcześniej nastąpiłoby wywłaszczenie i wątek wykonywany (o niższym priorytecie) nie doczekałby końca kwantu czasu.

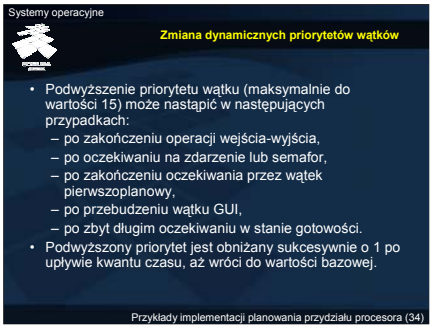
W pewnych warunkach priorytet wątku, który wykorzystał swój kwant czasu może zostać obniżony (patrz następne slajdy). W wyniku tego obniżenia może zaistnieć stan, w którym innych wątek gotowy ma priorytet wyższy, niż wątek dla którego kwant czasu właśnie upłynął.



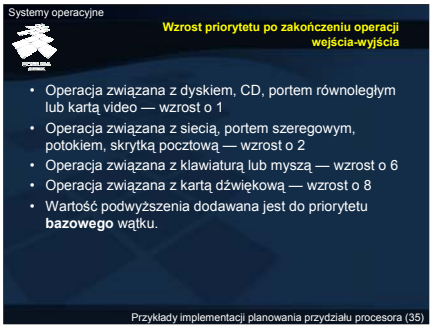
Wyrażenie kwantu w jednostkach nie związanych bezpośrednio z długością interwału zegara umożliwia zrealizowanie częściowego zanikania kwantu. Przykładem jest wejście w stan oczekiwania przed upłynięciem kwantu. Brak zanikania oznaczałby, że wątek nigdy nie utraci swojego kwantu, jeśli tylko będzie zgłaszał jakieś żądanie tuż przed jego upływem. Taki przywilej mają natomiast wątki w paśmie czasu rzeczywistego.



Wydłużenie kwantu zamiast zwiększenie priorytetu nie powoduje głodzenie przetwarzania w tle (zakładając ten sam priorytet). Ogranicza jedynie czas procesora dla takiego przetwarzania. Wyższy priorytet wątków związanych z obsługą okna mógłby oznaczać, że wątki w tle nie otrzymywałyby w ogóle procesora.

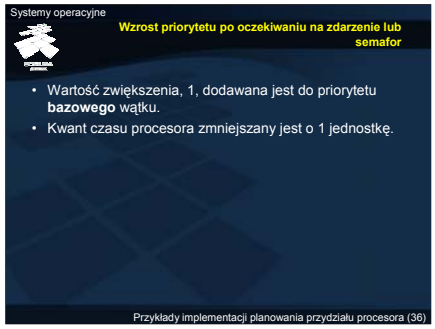


W zależności od zdarzenia, które spowodowało przejście wątku w stan oczekiwania, priorytet procesu, wchodzącego ponownie w stan gotowości, jest odpowiednio zwiększany. Zwiększenie priorytetu ma jednak charakter tymczasowy. Po upływie każdego kolejnego kwantu czasu jest on zmniejszany o 1, aż osiągnie poziom bazowy dla danego wątku.

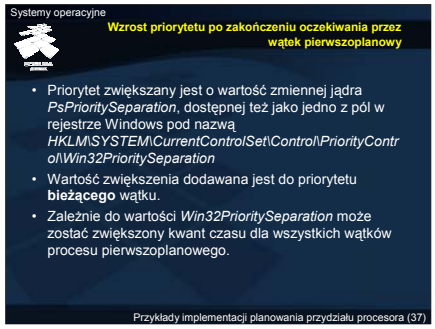


System Windows faworyzuje procesy interaktywne. Stąd wysoki wzrost priorytetu w przypadku zakończenie oczekiwania na dane z klawiatury lub reakcję myszy. Są to typowe urządzenia, którymi posługuje się użytkownik interaktywny. Również inne wątki, które zakończyły oczekiwanie na urządzenie zewnętrzne, otrzymują wyższy priorytet, gdyż mogą być ograniczone wejściem-wyjściem. Po uzyskaniu procesora nawet na krótki czas mogą ponownie zażądać operacji wejścia-wyjścia. W ten sposób równoważone jest obciążenie systemu.

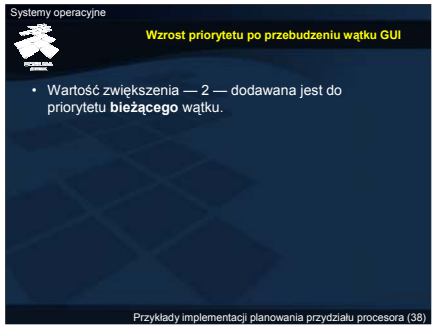
Wielkość podwyższenia jest definiowana przez moduł sterujący urządzenia wejścia-wyjścia.



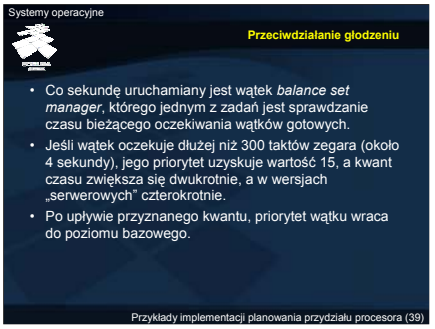
Zdarzenia i semafory związane są z synchronizacją między procesami. Zwiększony priorytet będzie obowiązywał tylko przez 1 kwant czasu, gdyż zaraz po nim zostanie obniżony o 1, wracając tym samym do poziomu bazowego.



Zwiększenie kwantu czasu zostało już omówione. Warto zwrócić uwagę, że zwiększenie priorytetu jest tymczasowe, podczas gdy zwiększenie kwantu będzie obowiązywać tak długo, jak długo okno będzie na pierwszym planie.



Wątki, które obsługują okno (nie koniecznie pierwszoplanowe) mają zwiększany priorytet o 2, gdy pojawi się jakieś zdarzenie (jakiś komunikat) związane z tym oknem.



Skutkiem zbyt niskiego priorytetu wątku może być jego głodzenie w dostępie do procesora, jeśli zawsze będzie jakiś wątek gotowy o wyższym priorytecie. Zwiększenie jego priorytetu do największej możliwej wartości w paśmie zmiennych priorytetów daje mu możliwość „złapania" procesora chociaż na krótką chwilę. Jeśli upłynie przyznany czas, priorytet wraca do wartości bazowej, co może wymagać kolejnych kilku sekund oczekiwania, zanim wątek dostanie kolejną szansę na dostęp do procesora.

Działanie takie ma między innymi na celu przeciwdziałanie inwersji priorytetów poprzez uruchamianie wątków, które być może przetrzymują jakieś zasoby potrzebne innym wątkom o wyższych priorytetach.