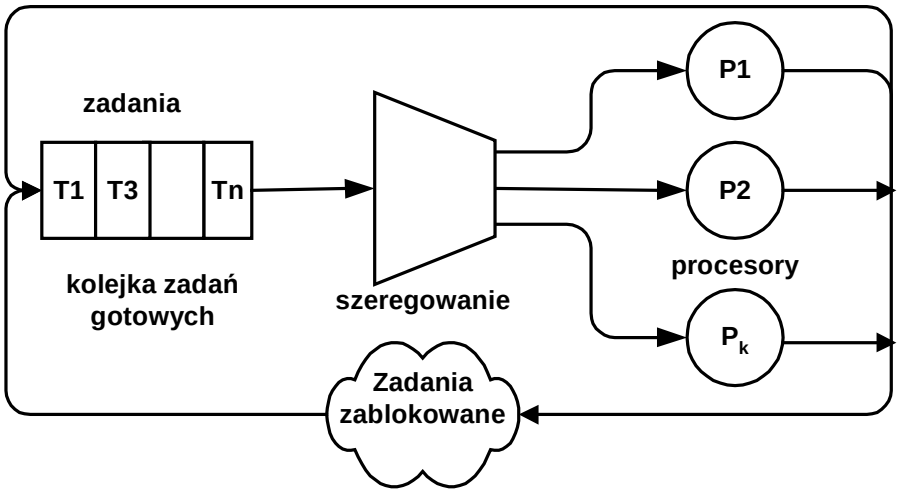
7. Szeregowanie procesów

W środowisku systemu pracuje zwykle więcej procesów gotowych do wykonania niż dostępnych jest procesorów. Stąd istnieje potrzeba decydowania który z procesów ma być wykonywany na którym z procesorów.



Rys. 7-1 Obieg zadań w systemie operacyjnym

Decyduje o tym procedura szeregująca (ang. *scheduler*). Jest to szereg funkcji zaimplementowanych w jądrze systemu które decydują które procesy ze zbioru procesów gotowych ma być wykonywany i na którym procesorze.

Funkcja procedury szeregującej – wybranie ze zbioru procesów gotowych procesu który ma być teraz wykonywany i na którym procesorze.

Procedura szeregująca jest aktywowana gdy:
1. Wystąpiło przerwanie zegarowe – proces bieżący wykorzystał przydzielony mu kwant czasu.
2. Wystąpiło przerwanie od urządzenia zewnętrznego – proces zablokowany na operacji wejścia / wyjścia stał się gotowy.
3. Proces bieżący wykonał wywołanie systemowe na skutek którego inny proces stał się gotowy.
4. Proces bieżący dobrowolnie oddał procesor lub zakończył się
5. Proces bieżący naruszył mechanizm ochrony procesora co spowodowało przerwanie wewnętrzne procesora.

☞ **Priorytet procesu** – liczbowa miara ważności procesu względem innego procesu.

Ze względu na wymogi szeregowania wyróżnia się trzy klasy procesów:
- Procesy interaktywne (ang. *Interactive*)
- Procesy wsadowe lub tła (ang. *batch, background*)
- Procesy czasu rzeczywistego (ang. *Real Time*)

Procesy interaktywne – oddziałują interaktywnie z użytkownikiem. Stąd oczekują zablokowane na nacisnięcie klawisza lub kliknięcie myszki. Gdy to nastąpi reakcja programu powinna być szybka (50-150 mS) gdyż inaczej system uznany zostanie jako wolny. Typowe aplikacje: edytory, aplikacje graficzne, interpretery poleceń.

Procesy wsadowe lub tła
Procesy które nie muszą bezpośrednio oddziaływać z użytkownikiem. Mogą to być procesy systemowe np. serwer WWW serwer bazy danych. Ich pilność zależy od funkcji którą wykonują.

Procesy czasu rzeczywistego
Czas reakcji tych procesów powinien spełniać rygorystyczne wymagania czasowe i powinien być deterministyczny. Proces taki nie może być blokowany przez proces o niższym priorytecie. Przykład: proces sterujący robotem, aplikacja wideo.

Ze względu na wewnętrzną specyfikę zapotrzebowania na zasoby wyróżnia się dwie klasy procesów:
- Ograniczone przez czas procesora (ang. *CPU bounded*)
- Ograniczone przez system wejścia/wyjścia (ang. IO *bounded*)

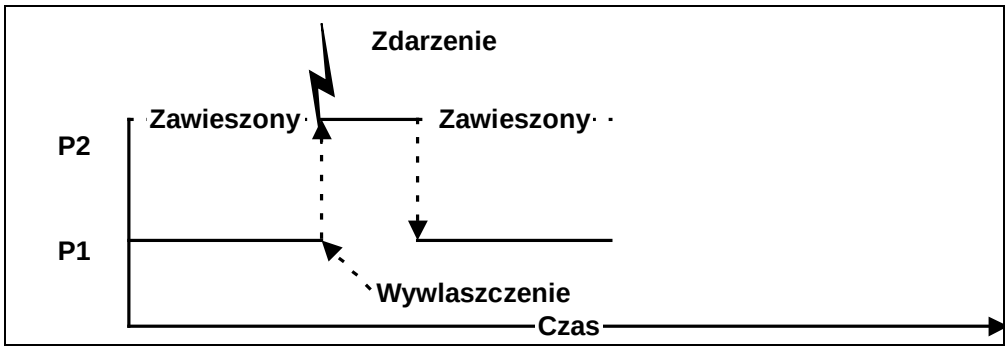
Ograniczone przez czas procesora – większość czasu zajmują funkcje wykonywane przez procesor. Przykładem mogą być programy optymalizacyjne.

Ograniczone przez system wejścia/wyjścia – większość czasu zajmują funkcje wejścia / wyjścia. Przykładem mogą być bazy danych, aplkacje interaktywne.

Decyzja szeregująca pogodzić musi wiele przeciwstawnych celów. Zaliczyć do nich można:
- Krótki czasu odpowiedzi dla procesów interaktywnych.
- Duża przepustowość dla procesów wsadowych
- Uniknięcie zagłodzenia procesów
- Znalezienie kompromisu między procesami o wysokich i niskich priorytetach

☞ **Podstawowa zasada szeregowania w systemach czasu rzeczywistego** - do wykonania wybierany jest zawsze proces gotowy o najwyższym priorytecie.

Szeregowanie wywłaszczające (*ang. preemptive scheduling*)
Gdy proces P2 o wyższym priorytecie niż aktualnie wykonywany proces P1 stanie się gotowy, proces P2 bezzwłocznie otrzymuje procesor a proces P1 jest wywłaszczany (*ang. preeempted*) i zawieszany.



Rys. 7-2 Wywłaszczająca strategia szeregowania

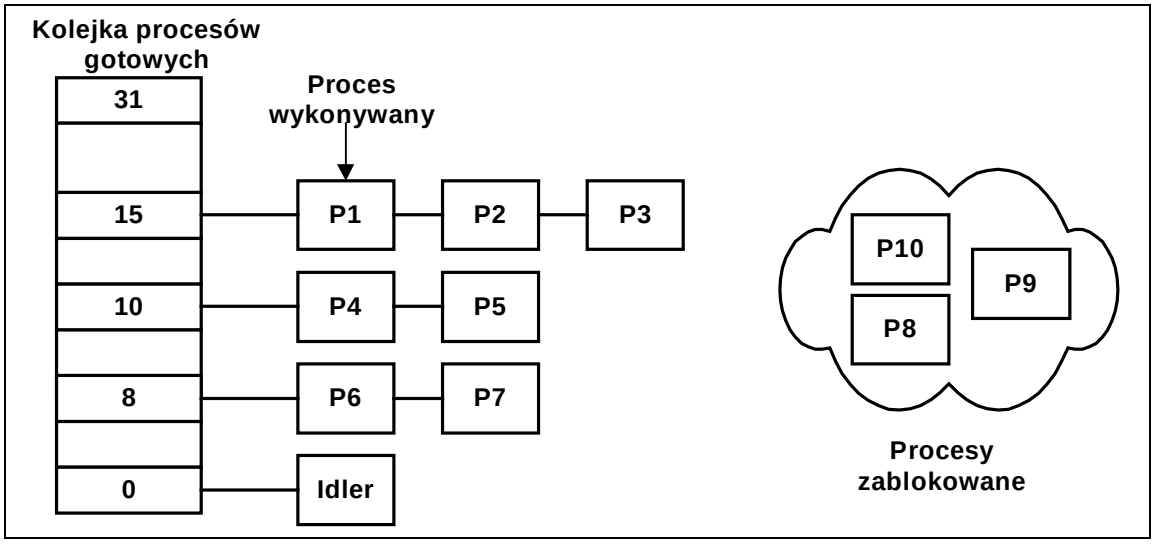
Określanie kwantu czasu przez który będzie się wykonywał proces:
- Gdy za krótki – większość czasu procesora zostanie zużyta na przełączanie kontekstu
- Gdy za długi – czas odpowiedzi systemu będzie długi.

7.1 Metody szeregowania - ogólne

W systemach operacyjnych popularne są następujące strategie szeregowania wykorzystujące priorytety.

1. Szeregowanie karuzelowe (*ang. Round Robin scheduling*).
2. Szeregowanie FIFO (*ang. FIFO scheduling*).
3. Szeregowanie adaptacyjne (*ang. Adaptive scheduling*).

Szeregowanie karuzelowe
Proces wykonywany jest aż do czasu gdy:
1. Samoistnie zwolni procesor.
2. Zostanie wywłaszczony przez proces o wyższym priorytecie.
3. Wyczerpie swój kwant czasu (*ang. timeslice*).



Rys. 7-3 Kolejka procesów gotowych i procesy zablokowane

Szeregowanie FIFO

Proces wykonywany jest aż do czasu gdy:

1. Samoistnie zwolni procesor.
2. Zostanie wywłaszczony przez proces o wyższym priorytecie.

Gdy procesy wykonywane z tym samym priorytetem stosują algorytm FIFO i operują na pewnym niepodzielnym zasobie, wzajemne wykluczanie zapewnione jest automatycznie.

Szeregowanie adaptacyjne

Szeregowanie adaptacyjne przebiega według następujących zasad:

1. Gdy proces wyczerpie swój kwant czasu i nie zablokuje się sam, jego priorytet obniżany jest o 1. Zjawisko to nazywane jest redukcją priorytetu (*ang. priority decay*).
2. Gdy proces sam się zablokuje, przywracany jest mu początkowy priorytet.
3. Gdy proces nie zostanie wybrany do wykonania po 1 sekundzie priorytet jego zwiększany jest o 1.

Szeregowanie adaptacyjne jest domyślną strategią szeregowania dla procesów uruchamianych z interpretera poleceń (*ang. shell*).

7.2 Metody szeregowania w systemie LINUX

7.2.1 Informacje wstępne

Wersja jądra 2.6 umożliwia pracę w systemie wieloprocesorowym. Stąd procedura szeregująca musi to uwzględniać i została zmodyfikowana w porównaniu z poprzednimi wersjami.

Wymagania na procedurę szeregowania

- Szeregowanie zadań w czasie stałym - wszystkie algorytmy powinny działać w czasie niezależnym od liczby szeregowanych procesów.
- Osobne kolejki zadań dla każdego procesora.
- Sprawiedliwość w przydziale czasu procesora.
- Zabezpieczenie przed zagłodzeniem i blokowaniem.
- Uwzględnienie wielu procesorów.
- Zapewnienie wysokiego stopnia interaktywności – nawet obciążony system powinien szybko reagować na polecenia użytkownika.

Ogólna zasada szeregowania w systemie Linux

- Szeregowanie w systemie jest wywłaszczające i oparte o dynamiczne priorytety i kredyty (czasy wykorzystania procesora).
- Gdy gotowe zadanie nie otrzymuje procesora jego priorytet stopniowo wzrasta.
- Gdy przekroczy priorytet bieżącego procesu jest on wywłaszczany.

W systemie Linux rozróżnia się następujące klasy szeregowania

• procesy czasu rzeczywistego szeregowane według algorytmu FIFO - SCHED_FIFO

• procesy czasu rzeczywistego szeregowane według algorytmu karuzelowego - SCHED_RR

• procesy zwykłe szeregowane według algorytmu SCHED_OTHER

Priorytety

- Do wyznaczania zadań do wykonania wykorzystuje się priorytety. Mają one zakres od 0 (najwyższy priorytet) do 139 (najniższy priorytet).
- Zadania czasu rzeczywistego SCHED_FIFO i SCHED_RR mają priorytety w zakresie 0 – 99 które są im przydzielane przez użytkownika.
- Zadania zwykłe mają priorytety w zakresie 100 – 139. Wartość ta zależy od parametru **`niceval`** (zakres od –20 do 19) nadawanego w poleceniu i funkcji **`nice niceval`**. Standardowo **`niceval`** = 0 co odpowiada priorytetowi 120.

7.2.2 Szeregowanie zadań czasu rzeczywistego

Zasady szeregowania zadań czasu rzeczywistego

- Jeżeli w systemie są zadania czasu rzeczywistego SCHED_FIFO lub SCHED_RR to będą one wykonywane w kolejności priorytetów i zgodnie z algorytmami FIFO lub karuzelowym.
- Zadania zwykłe SCHED_OTHER są wykonywane tylko wtedy gdy nie ma gotowych zadań czasu rzeczywistego

Szeregowanie karuzelowe dla procesów czasu rzeczywistego - SCHED_RR

Zasady:

1. Wykonywany jest proces gotowy o najwyższym priorytecie.
2. Proces wykonywany jest aż do czasu gdy:
   - Samoistnie zwolni procesor - zażąda niedostępnego zasobu lub wykona funkcję **`sched_yield()`**.
   - Zostanie wywłaszczony przez proces o wyższym priorytecie.
   - Wyczerpie swój kwant czasu (*ang. timeslice*).

Szeregowanie FIFO - SCHED_FIFO

Zasady:

1. Wykonywany jest proces gotowy o najwyższym priorytecie.
2. Proces wykonywany jest aż do czasu gdy:
   - Samoistnie zwolni procesor – zażąda niedostępnego zasobu lub wykona funkcję **`sched_yield()`**.
   - Zostanie wywłaszczony przez proces o wyższym priorytecie.

7.2.3 Szeregowanie zadań zwykłych

Kredyty

Procesom przydzielane są kredyty związane z ich priorytetem. Gdy zadanie jest wykonywane kredyt zmniejsza się z każdym przerwaniem zegarowym. Proces bieżący będzie wywłaszczony wtedy gdy wyczerpie on swój kredyt (ang. *time slice*). Po pewnym czasie kredyt będzie odnowiony.

Kredyt bazowy
Kredyt domyślny definiuje się jak poniżej:

```
#define DEF_PRIORITY (20*HZ/100)
```

Gdzie Hz jest częstotliwością przerwań zegarowych komputera. W PC wynosi ona 100 przerwań na sekundę , stąd **DEF_PRIORITY** wynosi 20 (jest to ok. 200 mS).

Każdy z procesów ma kredyt bazowy (ang. *base time quantum*) przydzielany mu gdy zaczyna się epoka. Wartość tego kredytu zależy od parametru **niceva**l ustawianego w funkcji **nice(...)** i **setpriority(...).**

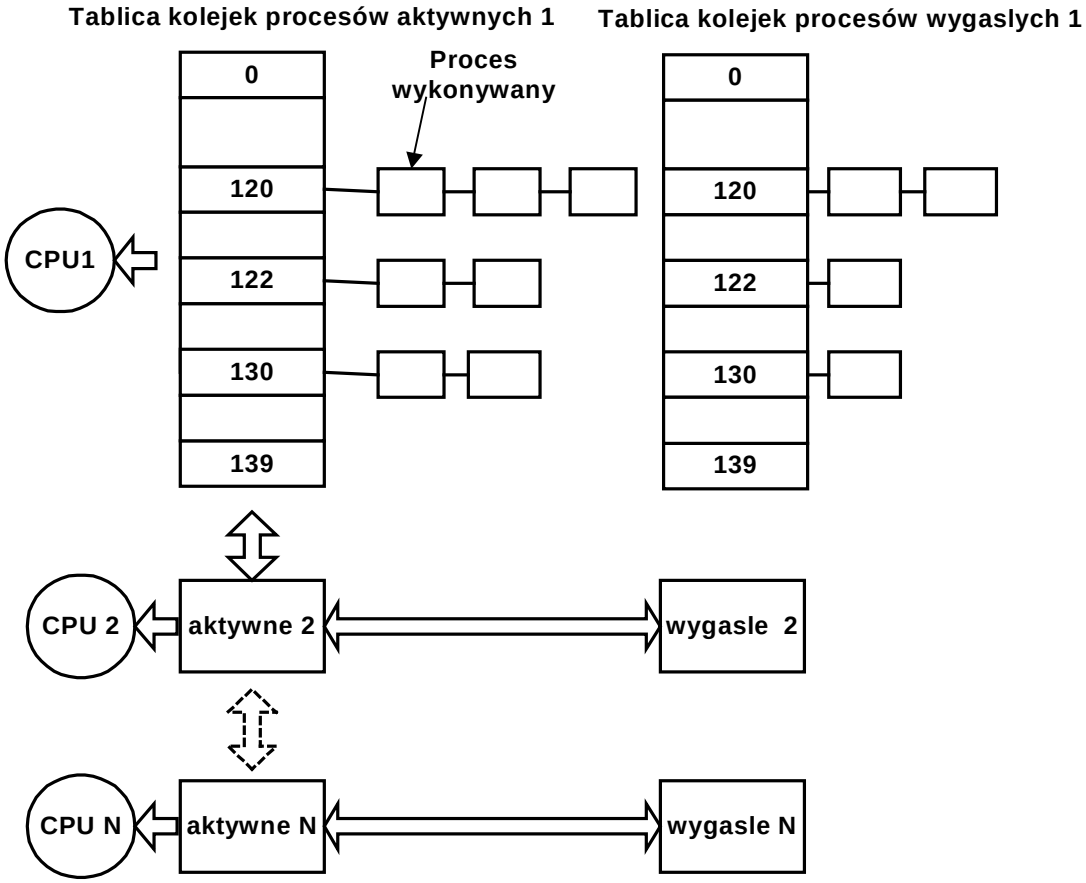
Epoki
Szeregowanie oparte jest o pojęcie epoki (ang. *epoch*). Gdy zaczyna się epoka każdemu procesowi przydziela się kredyt bazowy czasu procesora (ang. *time slice*) który ma do wykorzystania w tej epoce. Gdy kredyt procesu się skończy jest on wywłaszczany i zastępowany przez inny proces. W czasie epoki proces może być wielokrotnie wywłaszczany i szeregowany np. wtedy gdy sam zwalnia procesor. Epoka kończy się gdy wszystkie procesy gotowe wyczerpią swe kredyty. Wtedy zaczyna się nowa epoka i kredyty przydzielane są procesom na nowo.

Priorytet statyczny
Priorytet statyczny nie jest zmieniany przez procedurę szeregującą.
Zdania czasu rzeczywistego: 1 – 99.
Zadania zwykłe: 120 - niceval

Kolejki

Dla każdego procesora utrzymywane są dwie tablice kolejek.

- Tablica kolejek procesów aktywnych
- Tablica kolejek procesów wygasłych



Rys. 7-4 Kolejki procesów aktywnych w systemie wieloprocesorowym

Tablice zawierają kolejki dla każdego z poziomów priorytetów.

Gdy zadanie staje się gotowe otrzymuje priorytet i kredyt bazowy i umieszczane jest w kolejce procesów aktywnych. Z każdym przerwaniem zegarowym kredyt zmniejszany jest o jednostkę i wyczerpuje się.

W kolejce procesów aktywnych przebywają procesy które nie wykorzystały swego kredytu. Gdy wykorzystają przenoszone zostają do kolejki procesów wygasłych.
Gdy w kolejce zadań aktywnych nie ma już procesów wtedy:

- Następuje przełączenie kolejek.
- Przydziela się procesom nowe kredyty.

Szeregowanie

Funkcja **`schedule()`** implementuje scheduler. Jej zadaniem jest wyznaczenie procesu który ma być wykonywany jako następny.

Funkcja jest wywoływana gdy:

- W wywołaniach systemowych zmieniających status gotowości procesów
- Przy powrocie z przerwania i po każdym wywołaniu systemowym gdy ustawiona jest to jest potrzebne.

W szczególności funkcja wykonywana jest gdy:

- Bieżący proces nie może być kontynuowany z powodu braku zasobów.
- Gdy proces bieżący wyczerpał swój kredyt.
- Gdy jakiś proces zyskał status gotowości i jego priorytet jest większy niż priorytet procesu bieżącego.
- Kiedy wywoływane są funkcje **`sched_setscheduler()`** **`lub`** **`sched_ yield()`**

Priorytet dynamiczny

Wybór procesu do wykonania następuje na podstawie priorytetu dynamicznego. Stosuje się do zwykłych procesów.

Zmiany priorytetu dynamicznego:

- Priorytet dynamiczny zmniejsza się w miarę jak proces wyczerpuje swój kredyt czasu wykonania.
- Dla procesów które pozostają zablokowane na operacjach we/wy priorytet dynamiczny jest zwiększany.

 Podstawowa zasada szeregowanie w systemie Linux

Do wykonania zostaje wybrany gotowy proces czasu rzeczywistego który ma najwyższy priorytet statyczny. Gdy nie ma takich procesów wybrany zostaje proces zwykły który nie wyczerpał kredytu i ma najwyższy priorytet dynamiczny.

7.2.4 Funkcje modyfikujące szeregowanie i priorytet.

System dostarcza niezbędnych funkcji do zmiany i testowania priorytetu procesów i strategii szeregowania.

Wywołanie systemowe	Opis
nice()	Zmiana priorytetu
getpriority()	Testowanie maksymalnego priorytetu procesu lub grupy procesów
setpriority()	Ustawienie priorytetu procesu lub grupy procesów
sched_getscheduler()	Testowanie strategii szeregowania
sched_setscheduler()	Ustawienie priorytetu i strategii szeregowania
sched_getparam()	Testowanie priorytetu
sched_setparam()	Ustawianie priorytetu
sched_yield()	Zwolnienie procesora
sched_get_ priority_min()	Testowanie minimalnego priorytetu dla klasy szeregowania zadań
sched_get_ priority_max()	Testowanie maksymalnego priorytetu dla klasy szeregowania zadań
sched_rr_get_interval()	Testowanie kwantu czasu dla strategii SCHED_RR

Ustawienie priorytetu funkcja nice:

int nice(int *inc*)

inc Modyfikacja parametru **niceval** z zakresu od -20 do 20. (-20 najwyższy, 20 najniższy)

Funkcja zwraca:
gdy 0 – sukces
gdy -1 – błąd

Funkcja zmienia priorytet procesu. Dodatnia wartość parametru oznacza zmniejszenie priorytetu procesu, natomiast ujemna – zwiększenie. Tylko użytkownik **root** może ustawiać wartości ujemne.

Istnieje także polecenie systemowe **nice**

Ustawienie priorytetu polecenie nice:

```
nice [-n niceval] [-niceval]
     [polecenie [argumenty]]
```

niceval Modyfikacja priorytetu z zakresu od -20 do 20. (-20 najwyższy, 20 najniższy). Tylko użytkownik **root** może ustawiać wartości ujemne.

polecenie Polecenie które ma być wykonane na tym priorytecie

Przykład:

```
nice –n 10 tar –cvf kopia.tar /home
```

Ustawienie priorytetu polecenie renice:

```
renice niceval [[-p] pid ...] [[-g] pgrp ...] [[-
u] user ...]
```

Polecenie to zmienia priorytet aktualnie wykonywanego procesu.

Ustawienie priorytetu:

```
int setpriority(int which, int who, int niceval)
```

which **PRIO_PROCESS -** pojedynczy proces o identyfikatorze **who**,
PRIO_PGRP - grupa procesów o identyfikatorze **who**
PRIO_USER - wszystkie procesy użytkownika o identyfikatorze **who**

who **pid** procesu lub identyfikator grupy procesów lub identyfikator użytkownika

niceval Parametr nice

Funkcja zwraca:
gdy 0 – sukces
gdy -1 – błąd

Funkcja ta ustawia parametr nice i przez to zmienia priorytet wszystkich procesów w zadanym zbiorze lub też pojedynczego procesu.

Testowanie parametru nice:

`int getpriority(int which, int who)`

gdzie:

`which` **`PRIO_PROCESS -`** pojedynczy proces o identyfikatorze **`who`**,
`PRIO_PGRP -` grupa procesów o identyfikatorze **`who`**
`PRIO_USER -` wszystkie procesy użytkownika o identyfikatorze **`who`**

`who` **`pid`** procesu lub identyfikator grupy procesów lub identyfikator użytkownika

Funkcja zwraca:
gdy > 0 – wartość
gdy -1 – błąd

Funkcja podaje wartość parametru *nice* zadanego zbioru procesów , czyli najwyższy priorytet spośród parametrów nice wszystkich procesów należących do tego zbioru. Zbiór zadaje się poprzez odpowiednie wartości parametrów **`which`** i **`who`**.

Ustawienie metody szeregowania:

```
int sched_setscheduler(int pid,int alg, struct
sched_param *param)

struct sched_param {
            ...
            int sched_priority;
            ...
        }
```

pid PID procesu któremu zmieniamy strategię szeregowania priorytet (0 – gdy proces bieżący)

alg Nowy algorytm szeregowania: S**CHED_FIFO, SCHED_RR, SCHED_OTHER, SCHED_BATCH, SCHED_IDLE**

param W polu **param.sched_priority** tej struktury ustawiamy nowy priorytet

SCHED_FIFO	Szeregowanie FIFO czasu rzeczywistego
SCHED_RR	Szeregowanie karuzelowe czasu rzeczywistego
SCHED_OTHER	Szeregowanie standardowe
SCHED_BATCH	Szeregowanie wsadowe
SCHED_IDLE	Szeregowanie wsadowe o bardzo niskim priorytecie

Tab. 7-1Oznaczenia symboliczne strategii szeregowania

Funkcja zwraca:
gdy - 0 – sukces
gdy -1 – błąd

Funkcja ta ustawia tryb i parametry szeregowania procesu identyfikowanego przez **pid** lub procesu bieżącego, gdy **pid**=0. Parametry szeregowania zawarte są w strukturze typu **sched_param**, ktora w obecnej wersji zawiera tylko jedno pole - statyczny priorytet dla procesów czasu rzeczywistego.

Do zmiany parametrów szeregowania procesu uprawniony jest jego właściciel oraz administrator **root**. Do ustawienia trybów czasu rzeczywistego i zmiany ich parametrów uprawniony jest jedynie administrator systemu.

Testowanie metody szeregowania:

`int sched_getscheduler(int pid)`

`pid` PID procesu dla którego testujemy strategię szeregowania (0 – proces bieżący)

Funkcja zwraca:
gdy > 0 – aktualną strategię szeregowania procesu (patrz powyższa tabela)
gdy -1 – błąd

Funkcja zwraca tryb szeregowania procesu określonego przez pid lub procesu bieżącego gdy **`pid`**=0.

Ustawienie parametrów szeregowania:

`int sched_setparam(pid_t pid, struct sched_param *param)`

`pid` pid testowanego procesu lub 0 (dla procesu biezącego)
`param` Struktura z parametrami szeregowania

Funkcja zapisuje do struktury **`*param`** nowe parametry szeregowania procesu określonego przez **`pid`**, lub procesu bieżącego gdy **`pid`**=0. Wywołanie funkcji ma sens jedynie dla procesów czasu rzeczywistego i musi być wykonane przez administratora systemu.

Funkcja zwraca:
gdy - 0 – sukces
gdy -1 – błąd

Testowanie parametrów szeregowania:

`int sched_getparam(pid_t pid, struct sched_param *param)`

`pid` pid testowanego procesu lub 0 (dla procesu biezącego)
`param` Struktura z parametrami szeregowania

Funkcja zapisuje do struktury **`*param`** aktualne parametry szeregowania procesu określonego przez **`pid`**, lub procesu bieżącego gdy **`pid`**=0.

Funkcja zwraca:
gdy - 0 – sukces
gdy -1 – błąd

Testowanie priorytetu minimalnego i maksymalnego

```
sched_get_priority_min(int policy)
sched_get_priority_max(int policy)
```

Gdzie:
policy Algorytm szeregowania: S**CHED_FIFO, SCHED_RR, SCHED_OTHER, SCHED_BATCH, SCHED_IDLE**

Funkcja zwraca:
gdy > 0 funkcja zwraca maksymalny i minimalny priorytet dla danej metody szeregowania
gdy -1 – błąd

Wywołanie szeregowania

```
int sched_yield(void)
```

Funkcja zwraca 0.

Wywołanie funkcji spowoduje wywołanie procedury szeregującej i przesunięcie procesu bieżącego na koniec kolejki. Jest to oddanie procesora na własne życzenie

```
#include <sys.sched.h>
void main(void){
  struct sched_param param;
  int prio, alg;
  // Ustawienie strategii szereg. i priorytetu
  param.sched_priority  = 11;
  sched_setscheduler(0, SCHED_RR,&param);
  // Testowanie strategii szereg. i priorytetu

  alg = sched_getscheduler(0);
  printf(„Startegia szeregowania: %d\n”,alg);
}
```

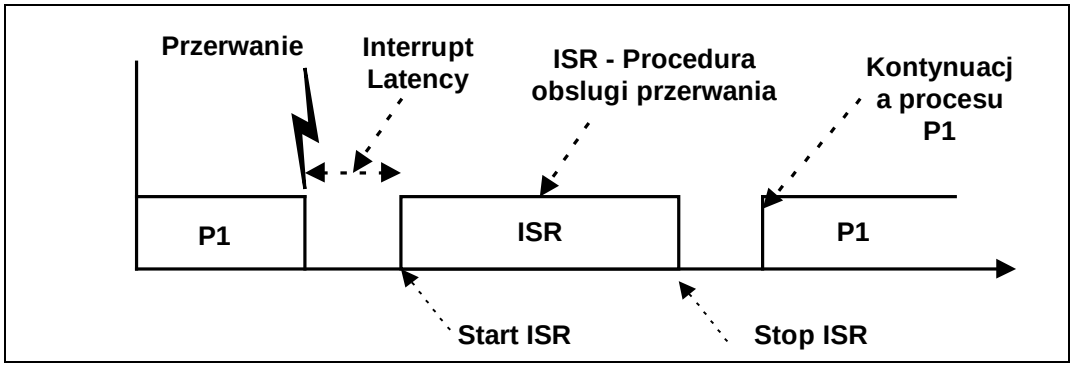
Przykład ustawiania i testowania parametrów szeregowania

7.3 Miary efektywności systemów czasu rzeczywistego

W systemach czasu rzeczywistego stosuje się następujące ilościowe kryteria ich oceny:

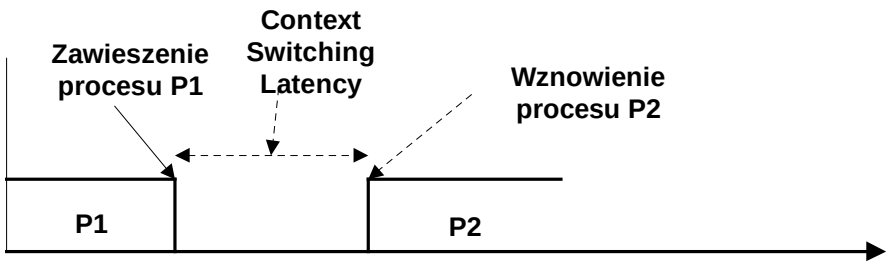
1. Zwłoka obsługi przerwania.
2. Czas przełączania kontekstu.
3. Czas szeregowania.

Zwłoka obsługi przerwania (*ang. Interrupt Latency*)
Jest to maksymalny czas pomiędzy wystąpieniem przerwania a wykonaniem pierwszej instrukcji procedury obsługi tego przerwania (*ang. Interrupt Service Routine – ISR)*.



Rys. 7-5 **Ilustracja zwłoki obsługi przerwania**

Czas przełączania kontekstu (*ang. Context Switching Latency*)
Czas przełączania kontekstu jest to czas potrzebny na zachowanie kontekstu procesu bieżącego P1 i odtworzenie kontekstu procesu P2 podlegającego zaszeregowaniu.

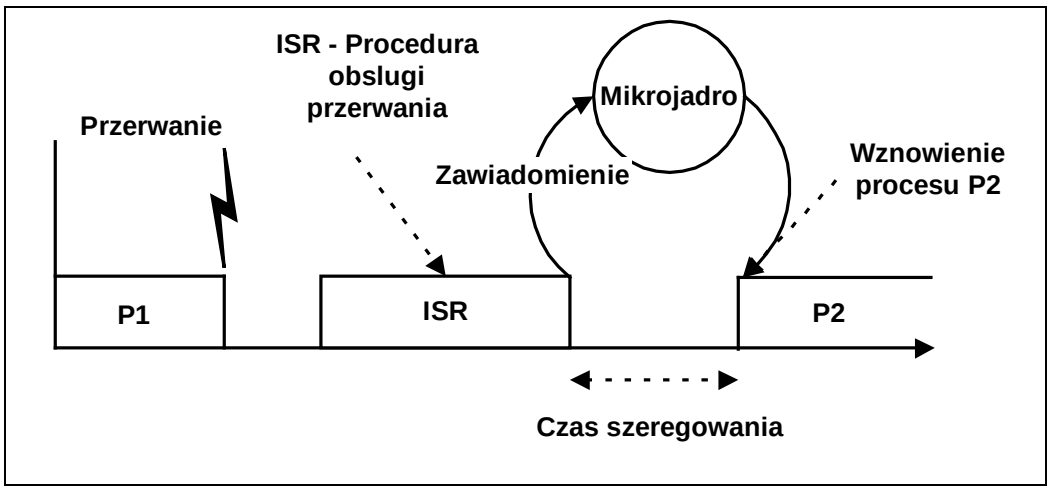


Rys. 7-6 Ilustracja czasu przełączenia kontekstu

Czas szeregowania (*ang. Scheduling Latency*)
Czas pomiędzy zakończeniem procedury obsługi przerwania a wznowieniem procesu obsługującego przerwanie nazywamy czasem szeregowania.
W tym czasie system wykonuje następujące czynności:

Szeregowanie

1. Szeregowanie procesów.
2. Zachowanie kontekstu procesu bieżącego P1.
3. Wznowienie procesu obsługującego zdarzenie P2.



Rys. 7-7 Ilustracja czasu szeregowania