

7. Szeregowanie procesów w systemie QNX6 Neutrino

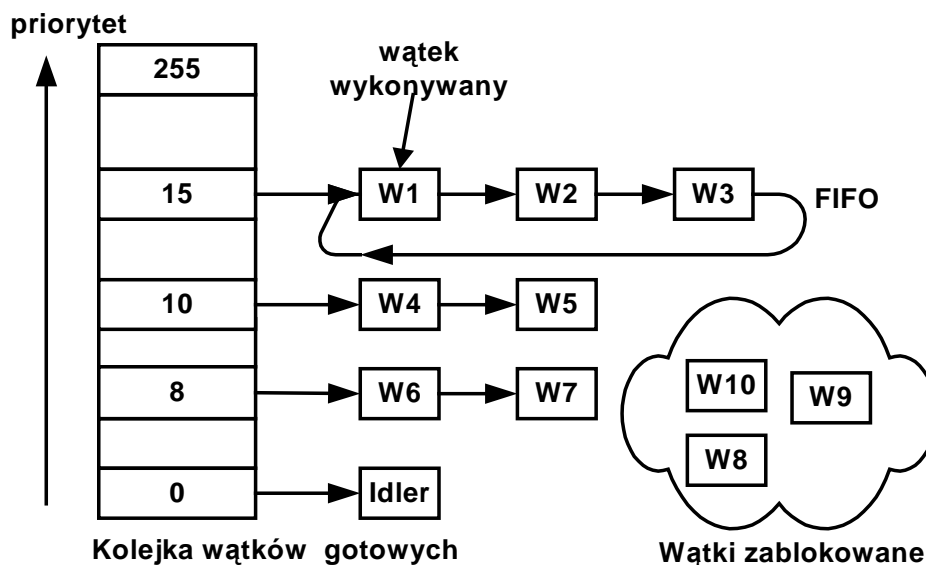
7.1 PRIORYTETY

Każdy z procesów wykonywanych w systemie RTS ma przyporządkowany priorytet. W systemie QNX6 Neutrino priorytet jest liczbą z zakresu od 0 do 255


Użytkownik	Priorytet
Proces jałowy	0
Wątki zwykłego użytkownika	1-63
Wątki użytkownika <code>root</code>	1-255

Tabela 7-1 Priorytety w systemie QNX6 Neutrino

Kolejka wątków gotowych składa się z 256 kolejek pomocniczych, z których każda odpowiada jednemu z priorytetów. W systemie zawsze wykonywany jest proces gotowy o najwyższym priorytecie. Takich procesów może być jednak więcej niż jeden.



Rys. 7-1 Struktura kolejki wątków gotowych

 Podstawowa zasada szeregowania - do wykonania wybierany jest zawsze wątek gotowy o najwyższym priorytecie.

Podana zasada szeregowania może być nie wystarczająca gdyż w systemie może być więcej wątków gotowych o najwyższym priorytecie. Stąd należy określić dodatkowe reguły dotyczące wyboru wątków do wykonania.

W systemie QNX6 Neutrino dostępne są trzy strategie szeregowania:

1. Szeregowanie karuzelowe (*ang. Round Robin scheduling*).
2. Szeregowanie FIFO (*ang. FIFO scheduling*).
3. Szeregowanie sporadyczne (*ang. sporadic scheduling*).

Procedura szeregująca jest aktywowana gdy:

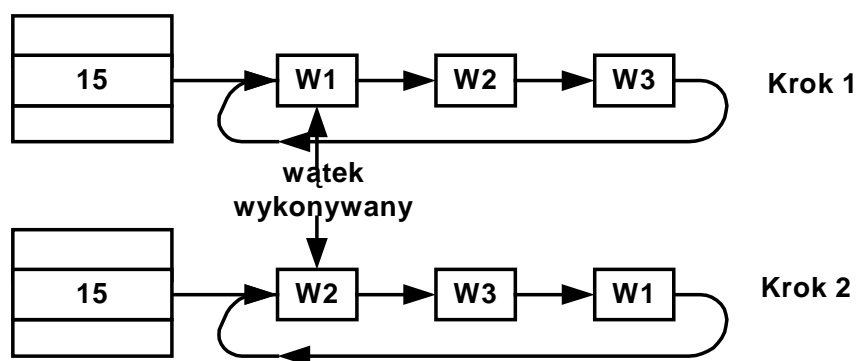
1. Wystąpiło przerwanie zegarowe – proces bieżący wykorzystał przydzielony mu kwant czasu.
2. Wystąpiło przerwanie od urządzenia zewnętrznego – proces zablokowany na operacji wejścia / wyjścia stał się gotowy.
3. Proces bieżący wykonał wywołanie systemowe na skutek którego inny proces stał się gotowy.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

7.2 METODY SZEREGOWANIA WĄTKÓW

7.2.1 Szeregowanie karuzelowe

Proces wykonywany jest aż do czasu gdy:

1. Samoistnie zwolni procesor.
2. Zostanie wyłączone przez proces o wyższym priorytecie.
3. Wyczerpie swój kwant czasu (*ang. timeslice*).



Rys. 7-2 Pojedynczy krok w szeregowaniu karuzelowym

7.2.2 Szeregowanie FIFO

Proces wykonywany jest aż do czasu gdy:

1. Samoistnie zwolni procesor.
2. Zostanie wyłączone 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.

7.2.3 Szeregowanie adaptacyjne (QNX4)

Szeregowanie adaptacyjne przebiega według następujących zasad:

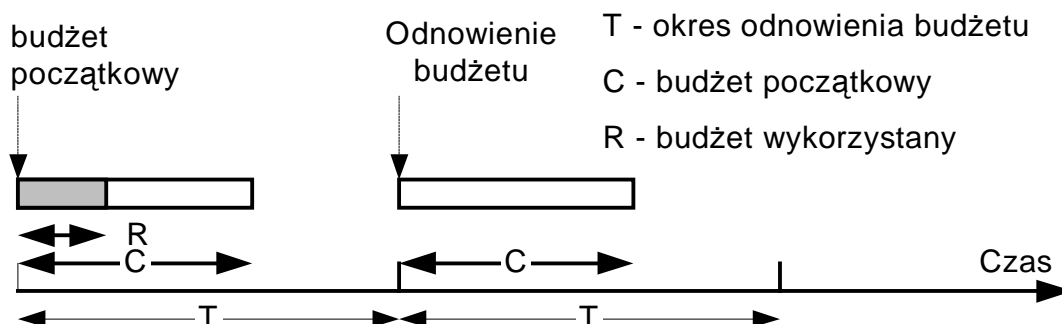
1. Gdy proces wyczerpie swój kwant czasu i nie zablokuje się sam, jego priorytet obniżany jest o 1. Zjawisko to nazywane jest redukcją priorytetu (*ang. priority decay*).
2. Gdy proces sam się zablokuje, przywracany jest mu początkowy priorytet.
3. Gdy proces nie zostanie wybrany do wykonania po 1 sekundzie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

7.2.4 Szeregowanie sporadyczne

Szeregowanie sporadyczne przeznaczone jest do wykonywania zadań cyklicznych typowych dla systemów czasu rzeczywistego. Jego celem jest zapewnienie że cykliczne procesy będą się wykonywały, nie stwarzając jednak ryzyka monopolizacji procesora.

W tym typie szeregowania aktualny priorytet wątku oscyluje pomiędzy normalnym N (początkowym) priorytetem a obniżonym priorytetem L . Szeregowanie sporadyczne definiowane jest w oparciu o następujące parametry:

1. Budżet początkowy (*ang. initial budget*) C – ilość czasu przez którą wątek może wykonywać się na normalnym priorytecie N zanim priorytet jego obniży się do poziomu L .
2. Priorytet obniżony (*ang. low priority*) L – priorytet niższy od priorytetu początkowego N , wątek będzie się wykonywał na tym priorytecie gdy jego budżet zostanie wyczerpany.
3. Okres odnowienia (*ang. replenishment period*) T – okres po którym budżet wątku zostanie „odnowiony” do wielkości budżetu początkowego.
4. Maksymalna liczba odnowień (*ang. max number of pending replenishments*) M – specyfikacja ile zdarzeń odnawiających może być przez system pamiętanych. Zbyt duża takich zdarzeń obciąża system i stąd jest ograniczona do M .



Rys. 7-3 Parametry szeregowania sporadycznego

Zasada szeregowania sporadycznego:

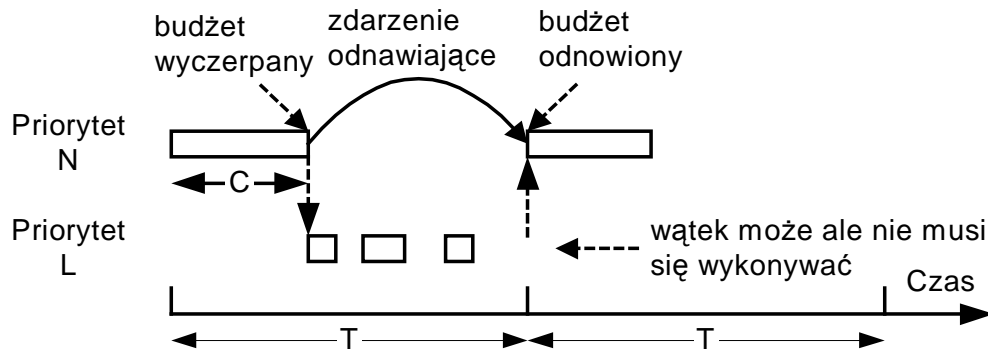
1. Wątek wykonywany jest gdy nie ma innych wątków gotowych o wyższym priorytecie.
2. Przy starcie wątek ma pewien priorytet początkowy N z którym zaczyna być wykonywany oraz budżet początkowy C .
3. Gdy wątek wyczerpie swój budżet procedura szeregująca obniża priorytet procesu do poziomu L .
4. Po obniżeniu priorytetu, wątek może być wykonywany lub też nie, co zależy od obciążenia procesora.
5. Wyczerpany budżet jest odnawiany przez zdarzenia odnawiające (*ang. Replenishment events*). Zdarzenia odnawiające generowane są w momencie gdy wątek traci sterowanie czy to w wyniku dobrowolnej blokady czy też wyczerpania budżetu.

Zdarzenie odnawiające charakteryzuje się dwoma parametrami:

1. Budżetem odnowienia – jest on równy wielkości czasu procesora zużytego przez wątek od ostatniego wznowienia do ostatniego zablokowania.
2. Czasem odnowienia – wątkowi przyznawany jest dodatkowy budżet (parametr pierwszy) w tym właśnie czasie. Czas ten równy jest czasowi ostatniego wznowienia wątku do którego dodany jest okres odnawiania T .

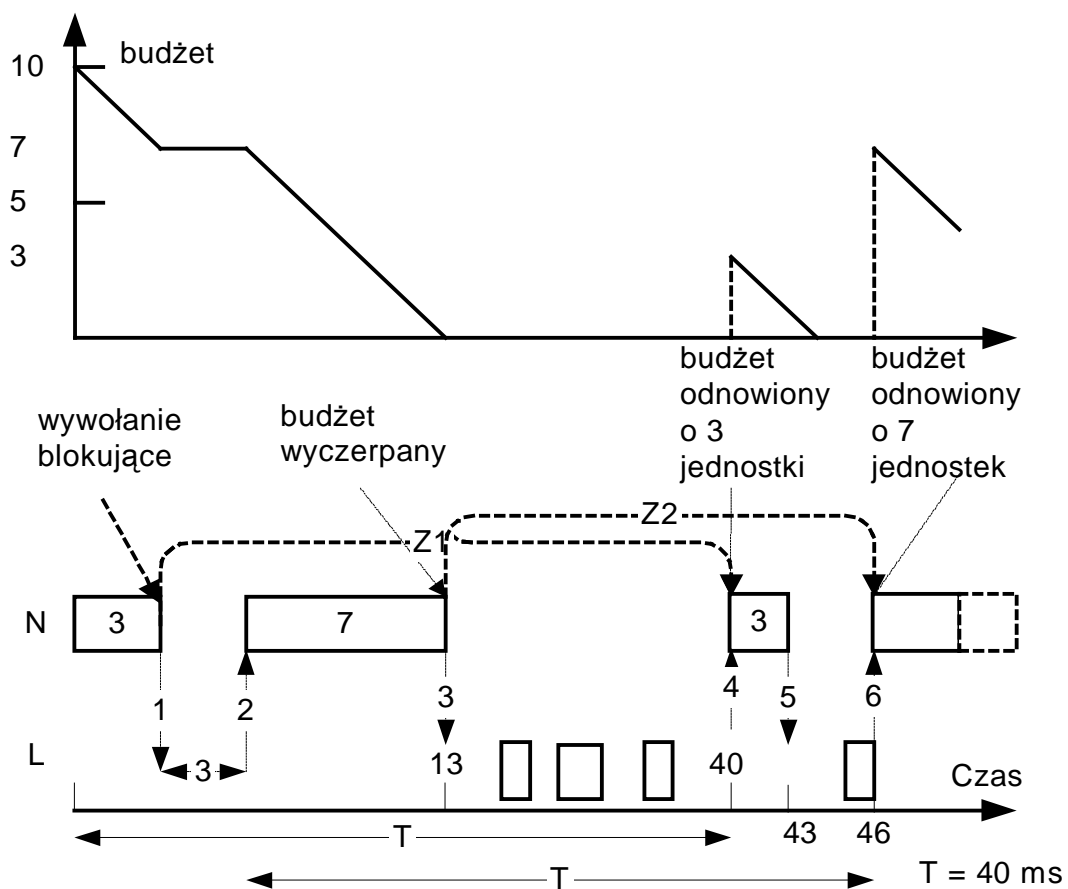
Maksymalna liczba zdarzeń odnawiających (długość kolejki) jest ograniczona do M .

Gdyby przedstawiony na Rys. 7-3 wątek zablokował się w momencie R to powstało by nowe zdarzenie odnawiające. Jego budżet wyniósłby R a czas odnowienia T . Blokada taka ma miejsce gdy wykonane będzie wywołanie systemowe żądające zablokowania wątku na określony czas lub w oczekiwaniu na pewne zdarzenie. Odnawianie budżetu na skutek wywołań blokujących jest nagrodą za „przyzwoite” zachowanie się wątku który sam zwalnia procesor.



Rys. 7-4 Oscylacja priorytetu wątku pomiędzy normalnym N a obniżonym L

Na Rys. 7-4 przedstawiono sytuację gdy po upływie czasu C wątek zużył cały swój budżet i nie wykonał żadnego wywołania blokującego. Tak więc w momencie C wątek traci sterowanie i jednocześnie generowane jest nowe zdarzenie odnawiające które w czasie T odnowi budżet o C.



Rys. 7-5 Szeregowanie wątku dla przypadku jego zablokowania

Na początku wątek otrzymuje budżet 10 ms a okres odnowienia wynosi 40 ms.

1. Po upływie 3 ms wątek samoistnie się blokuje co powoduje generację zdarzenia odnawiającego Z1 o liczbie jednostek 3 ms i czasie 40 ms.
2. Po kolejnych 3 ms wątek jest wznawiany i ma jeszcze 7 ms do wykorzystania.
3. Wątek wykorzystuje cały budżet 7 ms i system obniża mu priorytet do L. W tym czasie generowane jest zdarzenie odnawiające Z2 na 7 jednostek i o czasie odnowienia 46. Dalej wątek może być wykonywany lub nie co zależy od obciążenia systemu.
4. Zdziałało zdarzenie odnawiające Z1 i wątek zyskał 3 jednostki do budżetu po czym został mu przywrócony priorytet N i zyskał sterowanie.
5. Po upływie 3 ms wątek utracił sterowanie gdyż skonsumował swój budżet.
6. Zdziałało zdarzenie odnawiające Z2 i budżet wątku został powiększony o 7 jednostek. Wątek powrócił do priorytetu N.

Cel sporadycznej strategii szeregowania:

Zapewnienie aby w każdym okresie T wątek miał przyznane C jednostek czasu procesora a więc mógł skonsumować C/T procent mocy procesora.

5.1 STANY PROCESÓW I WĄTKÓW W SYSTEMIE QNX6 NEUTRINO

Informacje o stanie procesów można uzyskać za pomocą systemowego inspektora procesów lub też polecenia `pidin`.

```

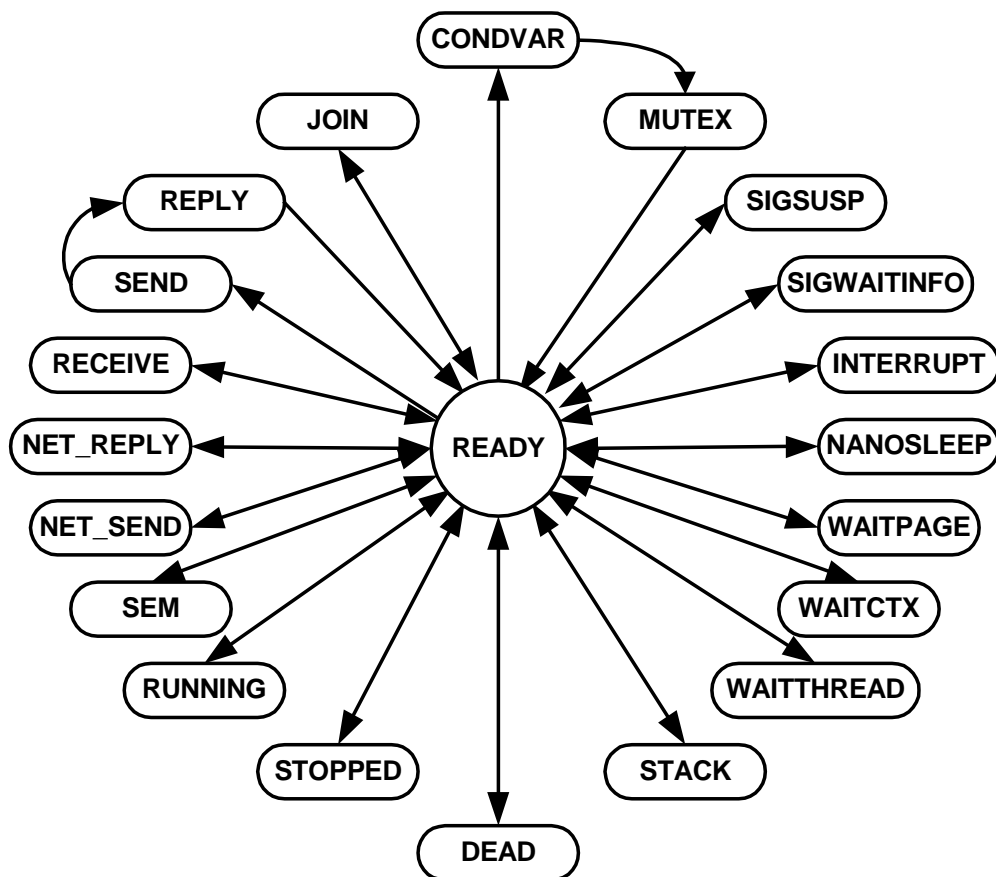
$pidin
pid tid name prio STATE Blocked
  1  1 /sys/procnto-instr 0f READY
  1  2 /sys/procnto-instr 11r RECEIVE 1
  1  3 /sys/procnto-instr 10r RECEIVE 1
  . . .
  1  8 /sys/procnto-instr 10r RUNNING
  1  9 /sys/procnto-instr 10r RECEIVE 1
  1 10 /sys/procnto-instr 10r RECEIVE 1
  2  1 sbin/tinit 10o REPLY 1
  3  1 proc/boot/slogger 10o RECEIVE 1
  5  1 proc/boot/pci-bios 12o RECEIVE 1
    
```

Przykład 7-1 Uzyskiwanie listy procesów i wątków za pomocą polecenia `pidin`

READY	Wątek posiada wszystkie potrzebne zasoby oprócz procesora.	Wątek gotowy
RUNNING	Wątek jest w trakcie wykonywania	Wątek wykonywany
CONDVAR	Oczekiwanie sygnalizacji na zmiennej warunkowej	Wątek zablokowany
DEAD	Stan zwany też „zombie”. Proces się zakończył ale jego proces macierzysty nie wykonał funkcji <code>wait()</code> .	
INTR	Oczekiwanie na przerwanie.	
JOIN	Oczekiwanie na zakończenie innego wątku np. po wykonaniu funkcji <code>pthread_join()</code> .	
MUTEX	Oczekiwanie na obiekt typu <code>mutex</code> np. po wykonaniu funkcji <code>pthread_mutex_lock()</code> .	
NANOSLEEP	Proces zablokowany jawnie na pewien czas np. po wykonaniu funkcji <code>nanosleep()</code> .	
NET_REPLY	Oczekiwanie na potwierdzenie komunikatu z sieci.	
NET_SEND	Oczekiwanie aż sygnał lub impuls będzie dostarczony przez sieć do adresata np. po wykonaniu funkcji <code>MsgSendPulse()</code> , <code>MsgDeliverEvent()</code> , <code>SignalKill()</code> .	

RECEIVE	Oczekiwanie na komunikat np. po wykonaniu funkcji <code>MsgReceive()</code> .
REPLY	Wątek jest zablokowany w oczekiwaniu na potwierdzenie komunikatu wysłanego funkcją <code>MsgSend()</code> w przypadku gdy serwer odebrał już komunikat.
SEM	Oczekiwanie na semaforze np. po wykonaniu funkcji <code>sem_wait()</code> .
SEND	Wątek jest zablokowany (np. po wykonaniu funkcji <code>MsgSend()</code>) w oczekiwaniu aż serwer odbierze komunikat.
SIGSUSPEND	Wątek jest zablokowany w oczekiwaniu na sygnał, np. wykonał funkcję <code>sigsuspend()</code> .
SIGWAITINFO	Wątek jest zablokowany w oczekiwaniu na sygnał, np. wykonał funkcję <code>sigwaitinfo()</code> .
STACK	Wątek jest zablokowany w oczekiwaniu na rozszerzenie stosu.
STOPPED	Proces zatrzymany po odebraniu sygnału <code>SIGSTOP</code> i oczekuje na sygnał <code>SIGCONT</code>
WAITCTX	Oczekiwanie na kontekst rejestrów (tylko w systemach wieloprocesorowych)
WAITPAGE	Oczekiwanie na alokację pamięci fizycznej dla adresu wirtualnego
WAITTHREAD	Oczekiwanie na utworzenie wątku potomnego przez wątek macierzysty, np. po wykonaniu funkcji <code>ThreadCreate()</code> .

Tabela 7-2 Stany procesu w systemie QNX6 Neutrino



Rys. 7-6 Przejścia pomiędzy stanami procesu w systemie QNX6 Neutrino

7.3 FUNKCJE MODYFIKUJĄCE PRIORYTET I STRATEGIĘ SZEREGOWANIE PROCESÓW.

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

Atrybut procesu	Testowanie	Ustawianie
Priorytet procesu	<code>getprio()</code> , <code>sched_getparam()</code>	<code>setprio()</code> , <code>sched_setparam()</code>
Strategia szeregowania	<code>sched_getparam()</code>	<code>sched_setparam()</code>

Tabela 7-3 Atrybuty procesów w systemie QNX6 Neutrino

Testowanie priorytetu:

```
int getprio(int pid)
```

pid PID procesu którego priorytet testujemy

Funkcja zwraca:

gdy > 0 – priorytet procesu

gdy -1 – błąd

Ustawienie priorytetu:

```
int setprio(int pid, int prio)
```

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

prio Nowy priorytet

Funkcja zwraca:

gdy > 0 – poprzedni priorytet procesu

gdy -1 – błąd

Przy tworzeniu procesu potomnego priorytet i metoda szeregowania jest dziedziczona z procesu macierzystego.

Testowanie parametrów szeregowania

```
int sched_getparam(pid_t pid, struct sched_param *par)
pid  PID testowanego procesu lub 0 dla procesu bieżącego
par  par struktura której elementami są pola określające
      priorytet.
```

Funkcja zwraca: 0 gdy sukces, -1 gdy błąd.

Struktura `sched_param` zawiera pola `sched_priority` i `sched_curpriority`.
`sched_priority` - priorytet jaki został ustawiony przy starcie procesu.
`sched_curpriority` - bieżący priorytet procesu nadany mu tymczasowo przez procedurę szeregującą.

Ustawienie priorytetu i parametrów szeregowania

Priorytet procesu ustawia się funkcją `sched_setparam()`.

Należy ustawić (na nowy priorytet) pole `sched_priority` zmiennej typu `sched_param` będącej argumentem tej funkcji.

Np: `par.sched_priority = 11`

```
int sched_setparam(pid_t _pid, struct sched_param *par)
pid  PID testowanego procesu lub 0 dla procesu bieżącego.
par  Struktura której elementami są pola określające priorytet.
```

Funkcja zwraca 0 gdy sukces, -1 gdy błąd.

Testowanie strategii szeregowania

Szeregowanie procesów odbywa się nie tylko na podstawie ich priorytetu ale także na podstawie strategii szeregowania która jest także atrybutem procesu.

Strategię szeregowania uzyskuje się przy pomocy funkcji `sched_getscheduler()`.

<pre>int sched_getscheduler(pid_t pid)</pre>
<pre>pid PID testowanego procesu lub 0 gdy procesu bieżącego</pre>

Funkcja zwraca numer strategii szeregowania.

Numer	Symbol	Opis
1	SCHED_FIFO	Szeregowanie FIFO
2	SCHED_RR	Szeregowanie karuzelowe
4	SCHED_SPORADIC	Szeregowanie sporadyczne

Tabela 7-4 Strategie szeregowania systemie QNX6 Neutrino

Ustawienie strategii szeregowania

Strategię szeregowania ustawia się funkcją `sched_setscheduler()`.

<code>int sched_setscheduler(pid_t pid, int typ, struct sched_param *par)</code>	
<code>pid</code>	PID procesu lub 0 gdy proces bieżący.
<code>typ</code>	Nowa strategia szeregowania.
<code>par</code>	Struktura określająca priorytet.

Funkcja zwraca 0 gdy sukces, -1 gdy błąd.

```
#include <sched.h>
#include <stdio.h>
int main (void) {
    struct sched_param par;
    int str;
    sched_getparam(0, &par);
    printf("Przydzielony priorytet: %d\n",
        par.sched_priority);
    str = sched_getscheduler(0);
    printf("Strategia szeregowania: %d\n", str);
    par.sched_priority = 9;
    sched_setparam(0, &par);
    sched_setscheduler(0, SCHED_FIFO, &par);
    sleep(20);
    return(0);
}
```

Przykład 7-2 Program testujący i ustawiający priorytet i strategię szeregowania dla procesu bieżącego

7.4 FUNKCJE MODYFIKUJĄCE PRIORYTET I STRATEGIĘ SZEREGOWANIE WĄTKÓW

Gdy proces składa się tylko z wątku głównego funkcje ustawiania priorytetu i strategii szeregowania są wystarczające. Jednak gdy w ramach procesu tworzone są inne wątki należy zastosować odmienny mechanizm.

Atrybut	Funkcja testowania	Funkcja ustawiania
Dziedziczenie	<code>pthread_attr_getinheritsched()</code>	<code>pthread_attr_setinheritsched()</code>
Parametry szeregowania	<code>pthread_attr_getschedparam()</code>	<code>pthread_attr_setschedparam()</code>
Strategia szeregowania	<code>pthread_attr_getschedpolicy</code>	<code>pthread_attr_setschedpolicy</code>

Tabela 7-5 Ważniejsze funkcje do ustalania priorytetów wątku

W normalnej sytuacji priorytet wątku i strategia szeregowania dziedziczone są z wątku macierzystego.

Aby jednak priorytet i strategia szeregowania była pobrana z jego atrybutów a nie z wątku macierzystego należy ustawić flagę `PTHREAD_EXPLICIT_SCHED`. Flagę tę ustawia się za pomocą funkcji `pthread_attr_setinheritsched()`.

Ustawianie dziedziczenia parametrów szeregowania

Aby ustawić nowy priorytet wątku i jego strategię szeregowania należy ustawić flagę `PTHREAD_EXPLICIT_SCHED` w atrybutach wątku i atrybuty wątku odpowiedzialne za jego priorytet i strategię szeregowania.

```
int pthread_attr_setinheritsched(pthread_attr_t *attr,
int inheritsched)
```

`attr` Wskaźnik na strukturę definiującą atrybuty wątku.
`inheritsched` Nowa wartość atrybutu określającego sposób uzyskiwania parametrów szeregowania.

Aby utworzyć wątek z zadaniem priorytetem należy:

1. Zadeklarować strukturę `par` typu `sched_param` i strukturę `attr` typu `pthread_attr_t`.
2. Zainicjować zmienną `attr` za pomocą funkcji `pthread_attr_init(&attr)`.
3. Ustawić sposób uzyskiwania parametrów szeregowania przez wykonanie funkcji `pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICITLY_SCHED)`.
4. Ustawić element `sched_priority` struktury `par` na zadaną wartość nowego priorytetu .
5. Za pomocą funkcji `pthread_attr_setschedparam(&attr, &par)`; dokonać modyfikacji struktury `attr` na wartość podaną w strukturze `par`.
6. Przy użyciu funkcji `pthread_create()` utworzyć nowy wątek z atrybutami `attr`.

Pobranie aktualnych parametrów szeregowania

<code>int pthread_attr_getschedparam(pthread_attr_t *attr, struct sched_param *par)</code>	
<code>attr</code>	Wskaźnik na strukturę definiującą atrybuty wątku
<code>par</code>	Wskaźnik na strukturę typu <code>sched_param</code> której pole <code>sched_priority</code> określa priorytet procesu.

Ustawianie priorytetu wątku

Do modyfikowania priorytetu wątku służy funkcja `pthread_attr_setschedparam()`.

<code>int pthread_attr_setschedparam(pthread_attr_t *attr, struct sched_param *par)</code>	
<code>attr</code>	Wskaźnik na strukturę definiującą atrybuty wątku
<code>par</code>	Wskaźnik na strukturę typu <code>sched_param</code> której pole <code>sched_priority</code> określa priorytetwątku

```

#include <pthread.h>
#include <stdio.h>
#include <math.h>
#define KROKI 10000000 // Dobrac do szybkości procesora
#define WMAX 16
int t1, tid[WMAX];

void * kod(void *arg) {
    float y;
    int ss, t2, i = 0;
    struct sched_param spar;
    int numer = (int) arg;
    pthread_getschedparam(tid[numer], &ss, &spar);
    printf("Start watku: %d Strategia: %d priorytet:
           %d\n", numer, ss, spar.sched_priority);
    while(i < KROKI) {
        i++; y = i ; y = y/7;
    }
    t2 = time(NULL);
    printf("Watek: %d zak., czas: %d\n", numer, t2-t1);
    return (NULL);
}

int main(int argc, char *argv[]) {
    int res, i, num = 0;
    pthread_attr_t attr[WMAX];
    struct sched_param p, p2;
    num = argc - 1; // liczba watkow
    for(i=0; i<num; i++) {
        pthread_attr_init(&attr[i]);
        pthread_attr_setinheritsched(&attr[i], PTHREAD_EXPLICIT
                                     _SCHED);
        p.sched_priority = atoi(argv[i+1]);
        pthread_attr_setschedparam(&attr[i], &p);
        pthread_attr_getschedparam(&attr[i], &p2);
        printf("Prior. watku: %d wynosi: %d\n", i, p2.
        }
    t1 = time(NULL);
    for(i=0; i<num; i++) {
        res = pthread_create(&tid[i], &attr[i], kod, (void *)i);
    }
    for(i=0; i<num; i++) {
        pthread_join(tid[i], NULL);
    }
    return 0;
}

```

Przykład 7-3 Ustawianie i testowanie priorytetów wątków

Ustawienie strategii szeregowania

Strategia szeregowania jest atrybutem wątku. Do ustawienia strategii szeregowania służy funkcja:

<code>int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy)</code>	
<code>attr</code>	Wskaźnik na strukturę definiującą atrybuty wątku
<code>par</code>	Strategia szeregowania

<code>SCHED_RR</code>	Karuzelowa strategia szeregowania
<code>SCHED_FIFO</code>	Szeregowanie FIFO
<code>SCHED_OTHER</code>	Obecnie szeregowanie karuzelowe
<code>SCHED_SPORADIC</code>	Szeregowanie sporadyczne

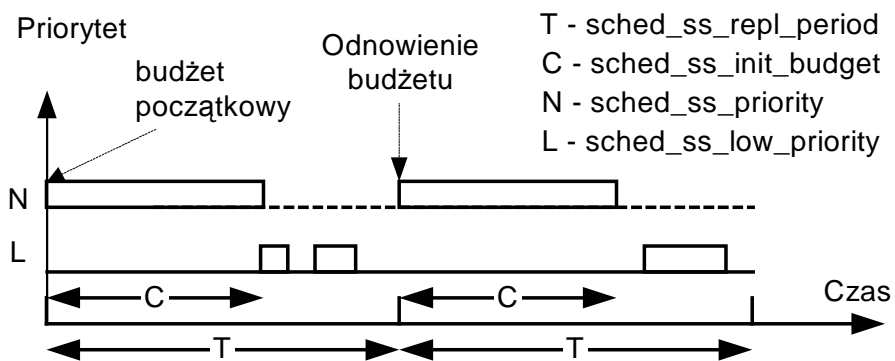
Tabela 7-6 Strategie szeregowania w systemie QNX6 Neutrino

- W przypadku domyślnego szeregowania (szeregowanie karuzelowe) jedynym parametrem jest priorytet procesu.
- W przypadku szeregowania sporadycznego liczba parametrów które należałoby ustalić jest większa.

Są one elementami struktury `sched_param` która jest parametrem funkcji `pthread_attr_setschedparam(...)`.

<code>sched_priority</code>	Priorytet normalny N na którym ma być wykonywany dany wątek.
<code>sched_ss_low_priority</code>	Priorytet obniżony L. Będzie on ustawiony przez system gdy wątek wyczerpie swój aktualny budżet C.
<code>sched_ss_max_repl</code>	Maksymalna liczba zdarzeń odnawiających.
<code>sched_ss_repl_period</code>	Okres T po którym budżet wątku zostanie odnowiony do wielkości budżetu początkowego C.
<code>sched_ss_init_budget</code>	Ilość czasu przez którą wątek może wykonywać się na normalnym priorytecie zanim priorytet jego obniży się do L.

Tabela 7-7 Parametry szeregowania sporadycznego – elementy struktury `sched_param`



Rys. 7-7 Parametry szeregowania sporadycznego

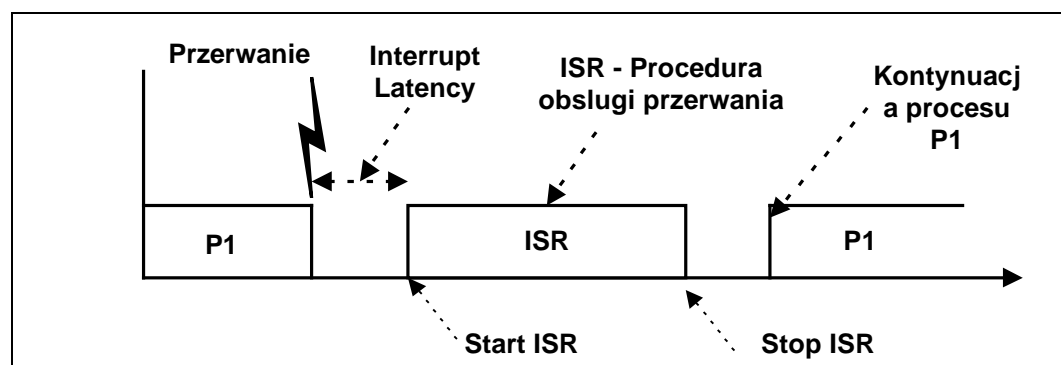
7.5 MIARY EFEKTYWNOŚCI SYSTEMÓW CZASU RZECZYWISTEGO

Stosuje się następujące miary efektywności systemów RTS:

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

Zwłoka obsługi przerwania (*ang. Interrupt Latency*)

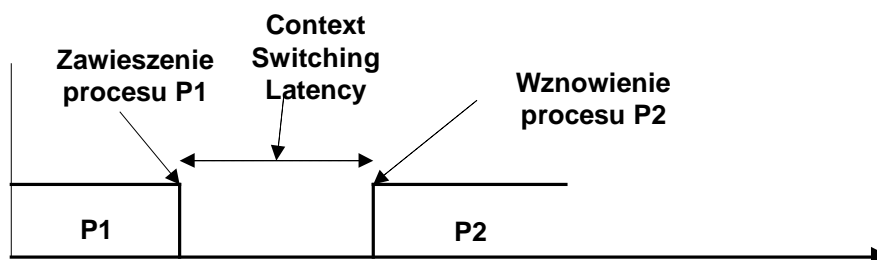
Jest to maksymalny czas pomiędzy wystąpieniem przerwania a wykonaniem pierwszej instrukcji procedury obsługi tego przerwania (*ang. Interrupt Service Routine – ISR*).



Rys. 7-8 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.



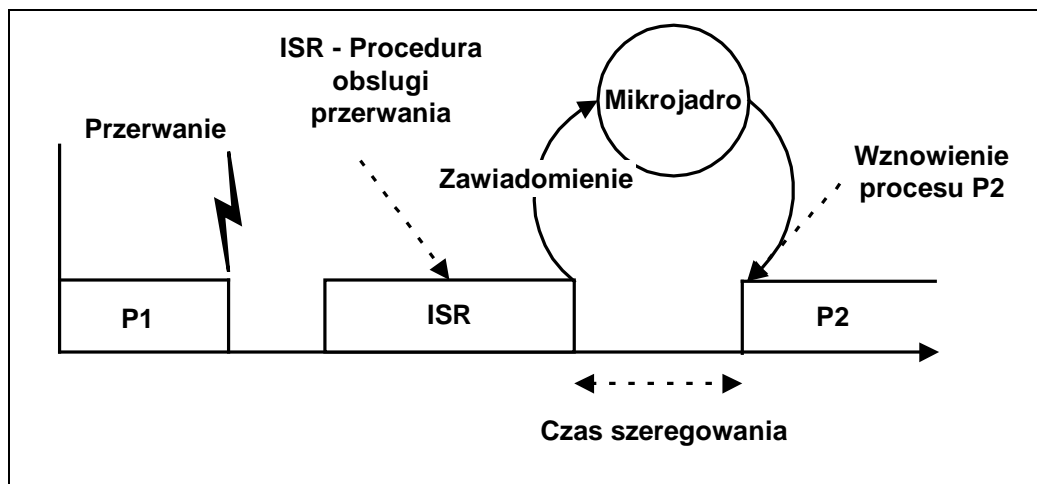
Rys. 7-9 Ilustracja czasu przełączenia kontekstu

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:

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



Rys. 7-10 Ilustracja czasu szeregowania