

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej

Skrypt do przedmiotu SCZR (Systemy Czasu Rzeczywistego)

*Praca napisana na podstawie wykładów w semestrze 04L
wykładowca: prof. nzw. dr hab. Krzysztof Sacha*

Piotr Andrzej Topór
nr albumu 180142

Warszawa, 16.06.2004

Spis treści

Spis treści	2
1 Wykład 1 [27.02.2004]	5
1.1 Definicja Systemu Czasu Rzeczywistego	5
1.2 Charakterystyka System Czasu Rzeczywistego	5
1.3 Klasyfikacja Systemów Czasu Rzeczywistego	5
1.3.1 Hard Real-Time System	5
1.3.2 Soft Real-Time System	6
1.4 Przykłady Systemów Czasu Rzeczywistego	6
1.4.1 Sterowanie przejazdem kolejowym	6
1.4.2 Regulacji temperatury	7
1.4.3 Transmisja pakietów	8
1.5 Model działania Systemu Czasu Rzeczywistego	8
1.6 Sterowanie zdarzeniami	8
1.7 Wymagania na system	8
1.8 Weryfikacja i ocena Systemu Czasu Rzeczywistego	9
2 Wykład 2 [05.03.2004]	10
2.1 Zapewnianie terminowości wykonywania zadań	10
2.1.1 Procedura postępowania	10
2.1.2 Twierdzenie Liu&Layland'a - 1973	11
2.1.3 Przykład szeregowania zadań	11
2.1.4 Inne twierdzenia	12
2.1.5 Symulacje	13
2.1.6 Podejście realistyczne	13
2.2 Organizacja oprogramowania Systemu Czasu Rzeczywistego	13
2.2.1 Cykliczny program sekwencyjny	13
3 Wykład 3 [12.03.2004]	15
3.1 Organizacja oprogramowania Systemu Czasu Rzeczywistego – ciąg dalszy	15
3.1.1 Cykliczny program sekwencyjny - ciąg dalszy	15
3.1.2 System dwuplanowy	17
3.1.3 Systemy wielozadaniowe (multitasking)	18
3.2 Podsumowanie wstępu do Systemów Czasu Rzeczywistego	19
3.2.1 Ostre ograniczenia czasowe	19
3.2.2 Łagodne ograniczenia czasowe	19
3.2.3 Cechy Systemu Czasu Rzeczywistego	20
4 Wykład 4 [19.03.2004]	21
4.1 Model budowy Systemu Operacyjnego	21
4.1.1 Standard POSIX	21
4.1.2 Definicja programu i procesu	21
4.1.3 Stany procesu	21
4.1.4 Zdarzenia	22
4.1.5 Tryby pracy systemu operacyjnego	23
4.1.6 Model budowy systemu operacyjnego opartego na mikrojądrze	23
4.1.7 Model budowy systemu operacyjnego opartego jądrze monolitycznym	24

5	Wykład 5 [26.03.2004]	26
5.1	Tablice systemowe	26
5.2	Wątek a proces	26
5.3	Model działania egzekutora	27
6	Wykład 6 [02.04.2004]	30
6.1	Tworzenie procesów	30
6.1.1	Funkcja FORK	30
6.1.2	Funkcje EXEC...	31
6.2	Tworzenie wątków	31
6.3	Algorytmy szeregowania procesów	32
6.4	Mechanizm sygnałów	33
6.5	Przegląd sygnałów	34
7	Wykład 7 [16.04.2004]	36
7.1	Synchronizacja i komunikacja zadań	36
7.1.1	Semaforey	37
7.1.2	Anomalia synchronizacji	40
8	Wykład 8 [23.04.2004]	41
8.1	Synchronizacja i komunikacja zadań - cd	41
8.1.1	Anomalia synchronizacji	41
8.1.2	Kolejki wiadomości	42
8.1.3	Spotkanie (rendez-vous)	45
9	Wykład 9 [07.05.2004]	46
9.1	Synchronizacja i komunikacja zadań - cd	46
9.1.1	Komunikacja przez obszary pamięci wspólnej (shared memory object)	46
9.2	Rozszerzenia czasu rzeczywistego	48
9.2.1	Problemy	48
9.2.2	Programowe liczniki czasu	48
9.3	System QNX	49
10	Wykład 10 [21.05.2004]	51
10.1	System QNX - cd	51
10.1.1	Komunikacja i synchronizacja w systemie QNX	51
10.1.2	Komunikacja ze sprzętem	53
10.1.3	Przerwania	53
10.2	Przemysłowe sieci miejscowe	53
10.2.1	Wprowadzenie	53
11	Wykład 11 [28.05.2004]	56
11.1	Przemysłowe sieci miejscowe - cd	56
11.1.1	Warstwa linowa (łącza danych)	56
11.2	Sieć PROFIBUS	60
11.2.1	Warstwa fizyczna: sprzęg RS-485	60
11.2.2	Warstwa liniowa (łącza danych)	61

12 Wykład 12 [04.06.2004]	63
12.1 Sieć PROFIBUS - cd	63
12.1.1 Warstwa liniowa (łącza danych) - cd	63
12.1.2 Warstwa aplikacyjna	66
13 Wykład 13 [15.06.2004]	71
13.1 Sieć PROFIBUS - cd	71
13.1.1 Warstwa aplikacyjna - cd	71
14 Wykład 14 [18.06.2004]	79
14.1 Rozproszony system sterujący w praktyce	79
Index	84
Rysunki	84
Literatura	85

1 Wykład 1 [27.02.2004]

1.1 Definicja Systemu Czasu Rzeczywistego

System Czasu Rzeczywistego (*Real Time System*) – jest to system komputerowy, w którym obliczenia prowadzone równolegle z przebiegiem *zewnętrznego procesu* mają na celu nadzorowanie, sterowanie lub *terminowe* reagowanie na zachodzącego w tym procesie zdarzenia.

(Definicja oparta o Standard Computer Dictionary IEEE Std 610, 1990)

1.2 Charakterystyka Systemu Czasu Rzeczywistego

Cechy charakteryzujące System Czasu Rzeczywistego:

- Połączenie z zewnętrznym procesem (sterowanie instalacją)
- Ograniczenia czasowe, np.:
 - zadany termin zakończenia akcji
 - zadany okres powtarzania czynności
 - zadane opóźnienie
- Groźne konsekwencje przekroczenia ograniczeń:
 - utrata zdrowia lub życia
 - straty finansowe

1.3 Klasyfikacja Systemów Czasu Rzeczywistego

1.3.1 Hard Real-Time System

Hard Real-Time System – system o ostrych (twardych) ograniczeniach czasowych

- Przykłady:
 - systemy sterowania instalacjami przemysłowymi, transportowymi, wojskowymi czy medycznymi
 - systemy dowodzenia, np. wspomaganie kontrolerów ruchu lotniczego
- Opis strat:
 - grozi utratą życia lub zdrowia
 - straty rosną skokowo po przekroczeniu ograniczeń
- Kryteria oceny:
 - ocena w najmniej korzystnym (najgorszym) przypadku
 - ocena przy maksymalnym obciążeniu

1.3.2 Soft Real-Time System

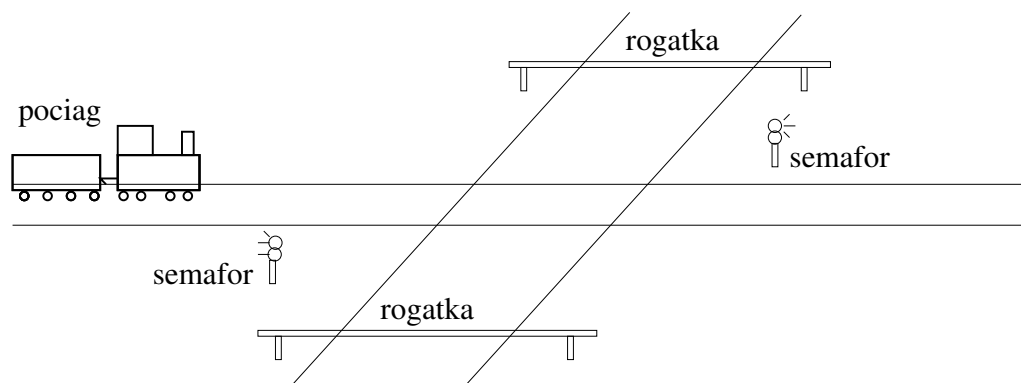
Soft Real-Time System – system o łagodnych (miękkich) ograniczeniach czasowych

- Przykłady:
 - interakcyjne systemy komercyjne, np. banki, rezerwacja miejsc lotniczych
 - systemy telekomunikacyjne
 - elektronika użytkowa, np. odtwarzacz CD
- Opis strat:
 - straty finansowe
 - straty rosną proporcjonalnie do opóźnienia
- Kryteria oceny:
 - ocena dotyczy parametrów przeciętnych
 - ocena przy przeciętnym obciążeniu

1.4 Przykłady Systemów Czasu Rzeczywistego

1.4.1 Sterowanie przejazdem kolejowym

Przykład opisuje system odpowiedzialny za sterowanie rogatek i semaforami na przejeździe kolejowym. Schemat jak na rysunku nr 1.



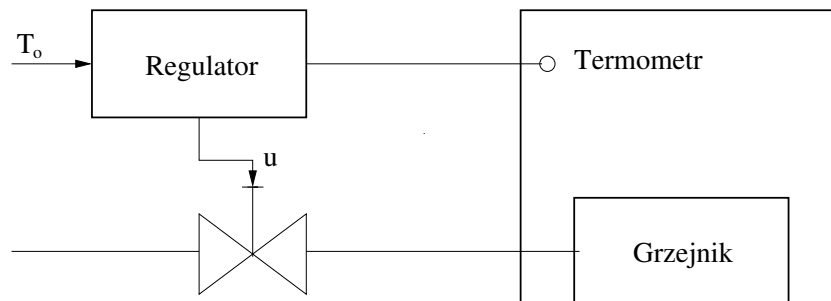
Rysunek 1: Przejazd kolejowy

- Ograniczenia czasowe:
 - zamknięcie rogatki musi nastąpić w czasie $< \Delta t_1$,
w przeciwnym wypadku nastąpi katastrofa
 - wyświetlenie zielonego światła w czasie $< \Delta t_2$,
w przeciwnym wypadku nastąpi konieczność awaryjnego hamowania
- Uwagi:
 - system nie musi być szybki
 - system nie może przekroczyć ograniczeń

Zdarzenie	Reakcja
pociąg zbliża się	zamknąć rogatekę wyświetlić zielone światło
pociąg odjeżdża	wyświetlić czerwone światło otworzyć rogatekę

1.4.2 Regulacji temperatury

Przykład opisuje system odpowiedzialny za regulację temeperatury w pomieszczeniach (np. w inteligentnych budynkach). Schemat jak na rysunku nr 2.



Rysunek 2: Regulacja Temperatury

- Wymagania:
 - stabilizacja temperatury z dokładnością $\pm 1^\circ C$

- Regulator:
 - PID (z reguły PI, PID w serwomechanizmach):

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right)$$

$$u_n = K_p \left(e_n + \frac{1}{T_i} \sum e_k \Delta t + T_d \frac{\Delta e}{\Delta t} \right)$$

- implementacja PID:

repeat every Δt

...

suma = *suma* + e_n

$u_n = K_p \cdot e_n + K_i \cdot \text{suma} + K_d \cdot (e_n - e_{old})$

$e_{old} = e_n$

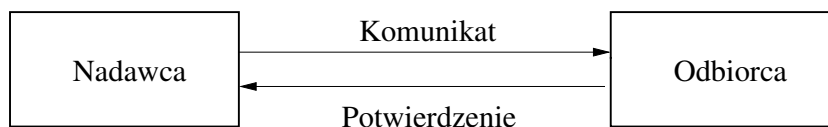
...

end repeat

- Uwagi:
 - wymagane jest utrzymanie stałego rytmu Δt , zazwyczaj równe 100 ms

1.4.3 Transmisja pakietów

Przykład opisuje system dyskretny odpowiedzialny za transmisję pakietów w sieci. Schemat jak na rysunku nr 3.



Rysunek 3: Transmisja pakietów

Zdarzenie	Reakcja
O – odbiór komunikatu	wysłanie potwierdzenia
N – odbiór komunikatu	wysłanie następnego komunikatu
N – brak potwierdzenia	retransmisja

- Ograniczenia czasowe:
 - wysłanie potwierdzenia w czasie $< \Delta t_1$
 - wysłanie retransmisji w czasie $> \Delta t_2$ od nadania (czas oczekiwania na potwierdzenia)
 - $\Delta t_1 > \text{czas transmisji komunikatu}$
 - $\Delta t_2 > \Delta t_1 + \text{czas transmisji komunikatu}$

1.5 Model działania Systemu Czasu Rzeczywistego

- działanie taktowane czasem (time-triggered system)
- działanie taktowane zdarzeniami (event-triggered),
tzw. systemy reaktywne (reactive systems)

1.6 Sterowanie zdarzeniami

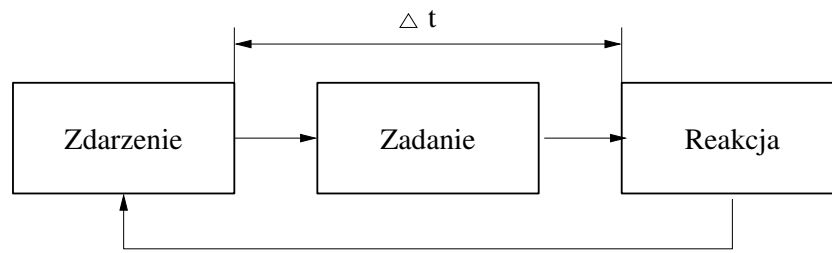
Zdarzenie – sytuacja w procesie zewnętrznym wymagająca interakcji systemu

Zadanie – algorytm, moduł programu wykonywany przez system równoległe z innymi zadaniami

Reakcja – zmiana stanu, sygnał sterujący lub komunikat do obsługi

1.7 Wymagania na system

- określają ścieżki pochodzenia zdarzenie - reakcja (stimulus response path)
- narzucają ograniczenia czasowe



Rysunek 4: Sterowanie zdarzeniami

1.8 Weryfikacja i ocena Systemu Czasu Rzeczywistego

- poprawność funkcjonalna
- terminowość wykonania

2 Wykład 2 [05.03.2004]

2.1 Zapewnianie terminowości wykonywania zadań

- Rodzaje zadań:
 - **zadania cykliczne**, powtarzane ze stałym, zadany okres (cycle) c_i (np. sterowania temperaturą [rodział 1.4.2])
 - **zadania sporadyczne (acykliczne)**, wykonywane w odpowiedzi na zdarzenie, z narzuconym terminem zakończenia (deadline) d_i (np. sterowania rogatkami i semaforami [rodział 1.4.1])

Zadania sporadyczne sprowadza się do cyklicznych poprzez określenie minimalnego odstępu między zadaniami i przyjmuje się go jako okres powtarzania c_i . Ważne jest, że każde zadanie po okresie c_i musi się zakończyć w ramach tego okresu ($d_i = c_i$).

2.1.1 Procedura postępowania

1. Określenie zadań $\{z_i\}$ i zdefiniowanie ograniczeń:
 - okres cyklu – c_i
 - termin zakończenia – d_i
2. Oszacowanie czasu wykonywania zadań $\{t_i\}$
3. Zaplanowanie kolejności wykonywania zadań:
 - ręczne rozplanowywanie kolejności (*pre-run scheduling*)
 - automatyczne planowanie kolejności przez system operacyjny (*run-time scheduling*):
 - szeregowanie priorytetowe (najpopularniejsze w systemach czasu rzeczywistego):
 - * według ważności
 - * według pilności wykonania (RMS – Rate Monotonic Scheduling), wysoki priorytet mają zadania o najkrótszym czasie wykonywania
 - szeregowanie według terminów wykonywania (EDF – Earliest Deadline First), algorytm ten jest trudniejszy do realizacji, gdyż ciężko jest określić terminy zakończenia wszystkich zadań.
4. Analiza projektu:
 - dany jest zbiór zadań $\{z_i\} = z_1, z_2, z_3, \dots, z_n$
 - dla każdego rodzaju zadania mamy określone:
 - czas cyklu – c_i
 - czas wykonania – t_i
 - problem polega na znalezieniu takiego uszeregowania zadań, które gwarantuje dotrzymanie terminów zakończenia wszystkich zadań (dotrzymanie okresów c_i).

2.1.2 Twierdzenie Liu&Layland'a - 1973

Jeżeli zachodzi nierówność:

$$\sum_{i=1}^n \frac{t_i}{c_i} \leq n(2^{\frac{1}{n}} - 1)$$

oraz szeregowanie zadań jest zgodne z pilnością (**algorytm RMS** - priorytet tym wyższy im krótszy czas c_i) to wszystkie zadania zostaną wykonane w terminie.

Uwagi:

- $\frac{t_i}{c_i}$ – obciążenie wnoszone przez jedno zadanie
- warunek, że łączne obciążenie $\leq 70\%$ jest **warunkiem dostatecznym** szeregowania zadań w algorytmie RMS, nie jest zaś warunkiem koniecznym (wynika to ze skierowania implikacji)
- system powinien mieć 30% rezerwy mocy
- szeregowalność w zależności od obciążenia można przedstawić w sposób następujący:

algorytm	RMS	?	zaden
obciążenie	...	70%	...
		...	100%
			...

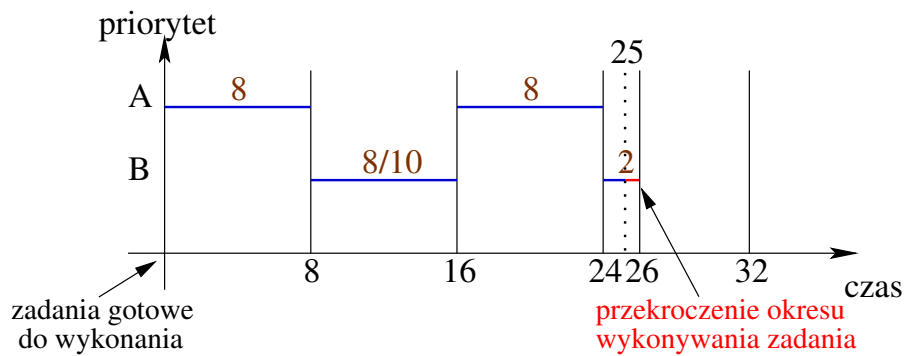
2.1.3 Przykład szeregowania zadań

zadanie	t_i	c_i	$\frac{t_i}{c_i}$	obciążenie
A	8	16	$\frac{1}{2}$	50%
B	10	25	$\frac{2}{5}$	40%
			łącznie	90%

- RMS - Rate Monotonic Scheduling (rysunek nr 5)
 - RMS jest algorytmem o stałych priorytetach.
 - Rozkład priorytetów:
 - * **A** - wysoki priorytet (bo częściej, krótszy czas wykonywania)
 - * **B** - niski priorytet
 - RMS jest algorytmem bardziej popularnym i łatwiejszym w realizacji.

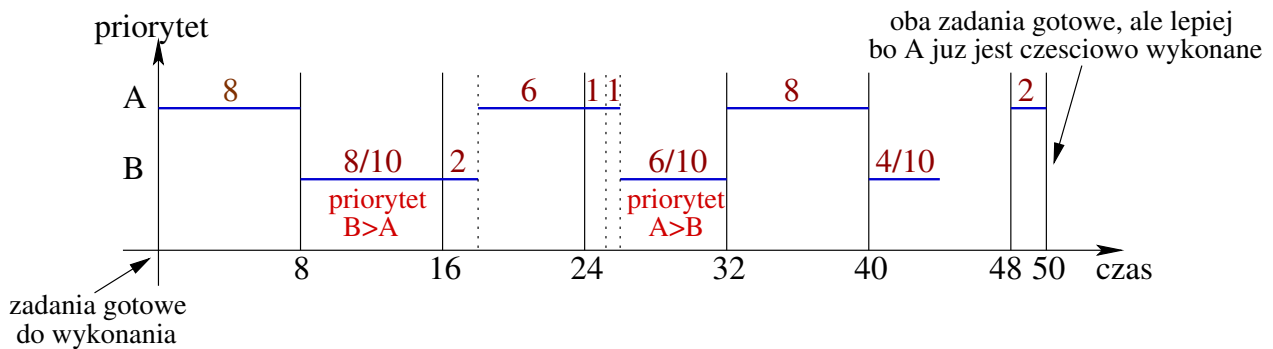
Wniosek:

Obciążenie systemu wynosiło 90%, co okazało się jest wartością za dużą i algorytm RMS się wyłożył.



Rysunek 5: Szeregowania algorytmem RMS

- EDF - Earliest Deadline First (rysunek nr 6)
 - EDF jest algorytmem o zmiennych priorytach.
 - EDF jest algorytmem trudniejszym do zrealizowania.



Rysunek 6: Szeregowania algorytmem EDF

Wniosek:

Pomimo dużego obciążenia systemu, algorytmem EDF udało się uszeregować zadania. Jest to dobry dowód na to, że twierdzenie Liu&Layland'a mówi o warunku dostatecznym, a nie koniecznym szeregowalności.

2.1.4 Inne twierdzenia

Udowodniono twierdzenia:

- algorytm RMS jest optymalny wśród algorytmów o stałych priorytetach (twierdzenie Liu&Layland'a - 1973)
- algorytm EDF jest optymalny wśród algorytmów o zmiennych priorytetach (twierdzenie Spuri, Buttazzo, Sensini - 1993)
- jeśli $\sum_{i=1}^n \frac{t_i}{c_i} \leq 1$ to algorytm EDF uszereguje zadania

2.1.5 Symulacje

Przeprowadzane symulacje dla losowo generowanych zbiorów zadań wykazują, że algorytm RMS uszereguje zadania jeśli:

$$\sum_{i=1}^n \frac{t_i}{c_i} \leq 0.88$$

jednak nie jest to w żaden sposób udowodnione. Zatem w systemach, gdzie dotrzymanie terminowości zadań nie jest konieczne (np. odtwarzacz MP3) zakłada się, że algorytmem RMS można szeregować zadania systemu o obciążeniu około 88%.

2.1.6 Podejście realistyczne

- dany jest zbiór zadań $\{z_i\}$
- dla każdego zadania określamy:
 - okres cyklu – $c_i = d_i$
 - średni czas wykonywania – t_{ia}
 - maksymalny czas wykonywania – t_{im}

Zalecenia:

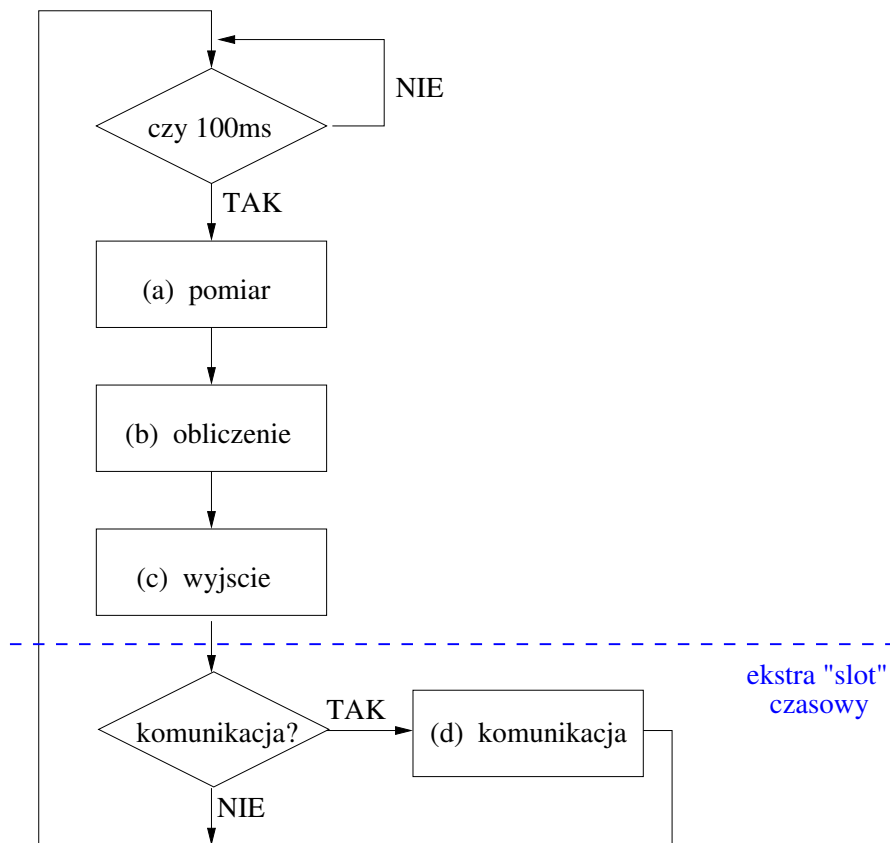
1. Wyodrębnić grupę zadań krytycznych (ostre ograniczenia czasowe) i pozostałych (łagodne ograniczenia czasowe).
2. Nadać zadaniom krytycznym wyższe priorytety i uszeregować je zgodnie z algorytmem RMS, a następnie wykazać szeregowalność dla maksymalnych czasów wykonywania.
3. Uszeregować pozostałe zadania zgodnie z RMS i wykazać szeregowalność dla średnich czasów wykonywania.

2.2 Organizacja oprogramowania Systemu Czasu Rzeczywistego

2.2.1 Cykliczny program sekwencyjny

Cykliczny program sekwencyjny (rysunek nr 7 jest bardzo popularny w urządzeniach powszechnego użytku, np. tak zrealizowane mają oprogramowanie sterowniki PLC.

zadanie	czas wykonywania	czas powtarzania
(a) pomiar	20 ms	≤ 100 ms
(b) obliczenia	30 ms	≤ 100 ms
(c) wyjście	10 ms	≤ 100 ms
suma czasów	60 ms	



Rysunek 7: Cykliczny program sekwencyjny

Zalety:

- Pełna gwarancja terminowości.
- Brak jakichkolwiek współbieżności, a co za tym idzie brak jakiejkolwiek synchronizacji zadań.
- Możliwość dodawania ekstra „slotów” czasowych na czynności związane ze zdarzeniami.

3 Wykład 3 [12.03.2004]

3.1 Organizacja oprogramowania Systemu Czasu Rzeczywistego – ciąg dalszy

3.1.1 Cykliczny program sekwencyjny - ciąg dalszy

zadanie	czas wykonywania	czas powtarzania	obciążenie
(a) pomiar	20 <i>ms</i>	$\leq 100 \text{ ms}$	20%
(b) obliczenia	30 <i>ms</i>	$\leq 200 \text{ ms}$	15%
(c) wyjście	10 <i>ms</i>	$\leq 50 \text{ ms}$	20%
(d) komunikacja	40 <i>ms</i>	$\leq 200 \text{ ms}$	20%
suma czasów	100 <i>ms</i>		75%

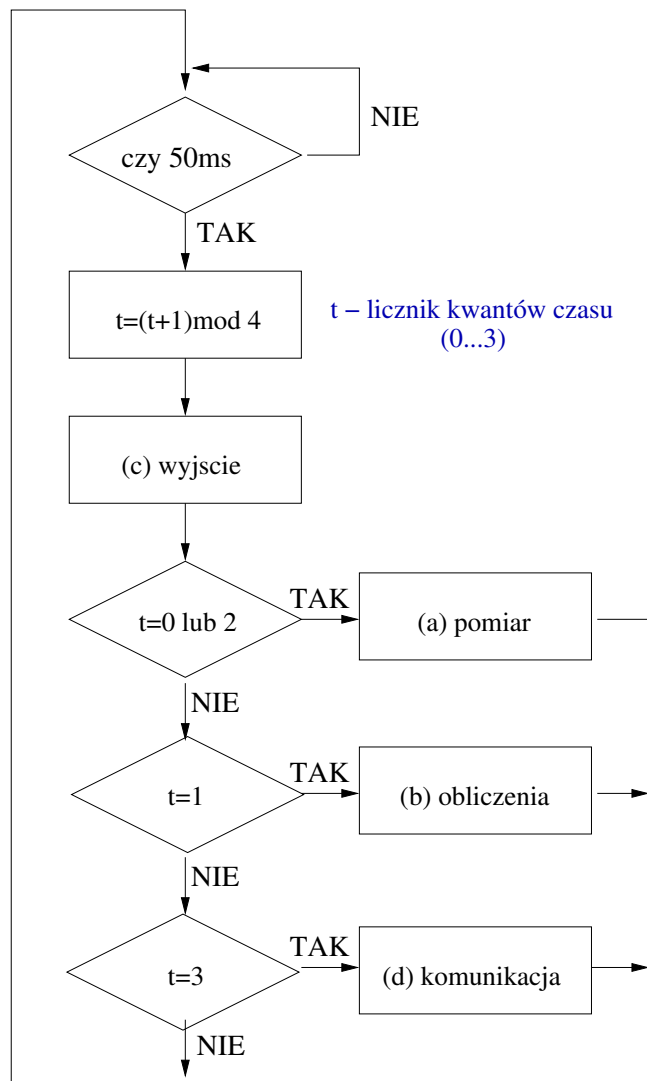
Dla takiego wariantu przykładowy *pre-run scheduling* może wyglądać następująco:

	0	1	2	3
(a)	X		X	
(b)		X		
(c)	X	X	X	X
(d)				X
suma czasu	30 <i>ms</i>	40 <i>ms</i>	30 <i>ms</i>	50 <i>ms</i>

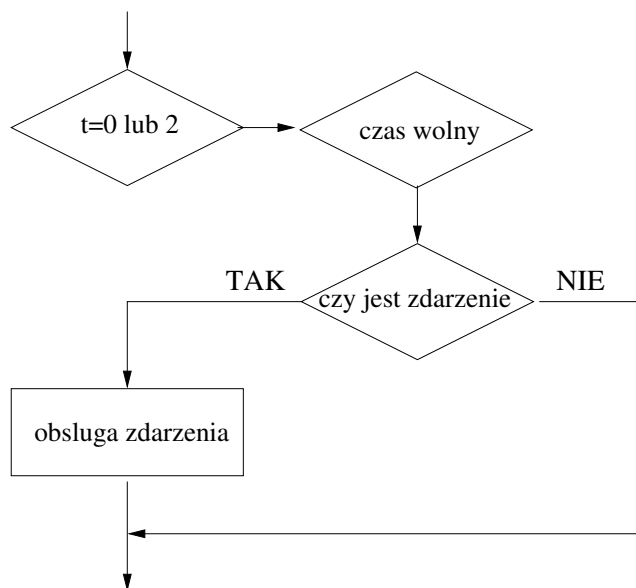
Schemat drugiej wersji cyklicznego programu sekwencyjnego na rysunku nr 8

Uwagi:

- Gwarancja dotrzymania ograniczeń czasowych
- Brak problemów synchronizacyjnych
- Możliwość uwzględniania zdarzeń (rysunek nr 9)
- Nie ma potrzeby wprowadzania przerw



Rysunek 8: Cykliczny program sekwencyjny (2 wersja)



Rysunek 9: Obsługa zdarzenia

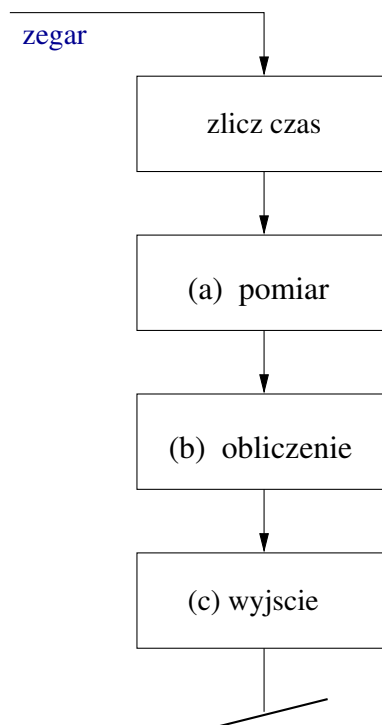
3.1.2 System dwuplanowy

System dwuplanowy (Foreground / Background System)

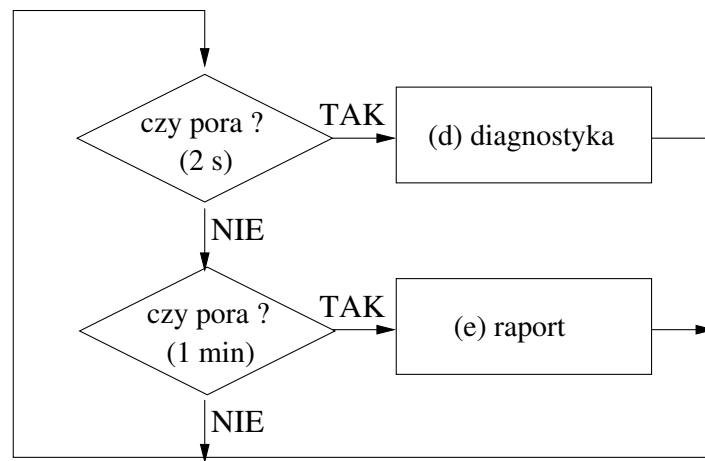
Podział zadań na dwie grupy:

- Plan I – czynności taktowane czasem, np. wykonywane po przerwaniu zegarowym, odpowiedzialne za bezpieczeństwo (ostre ograniczenia czasowe)
- Plan II (tło) – czynności wykonywane w czasie wolnym (łagodne ograniczenia czasowe)

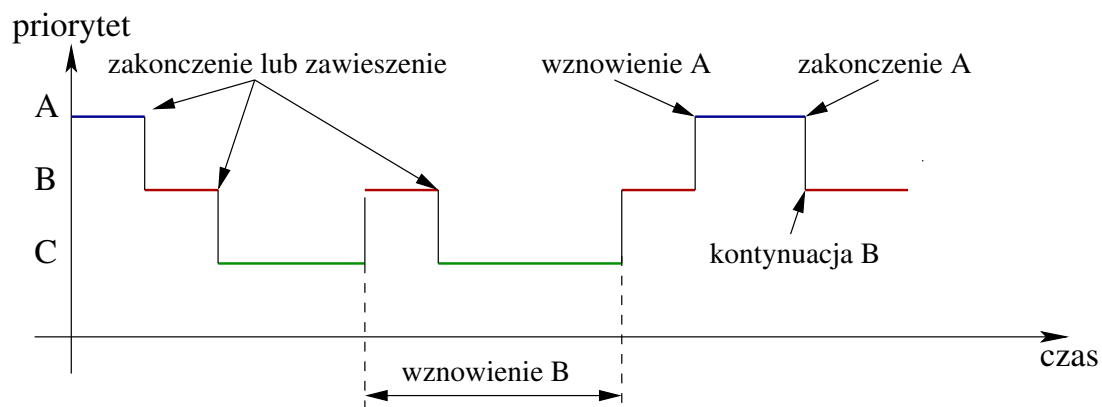
Zadanie	czas wykonania	cykl	obciążenie	
(a) pomiar	20 <i>ms</i>	$\leq 200 \text{ ms}$	10%	PLAN I
(b) obliczenie	30 <i>ms</i>	$\leq 200 \text{ ms}$	15%	
(c) wyjście	10 <i>ms</i>	$\leq 200 \text{ ms}$	5%	
(d) diagnostyka	200 <i>ms</i>	$\leq 2 \text{ s}$	10%	PLAN II
(e) raport	1 <i>s</i>	$\leq 1 \text{ min}$	1,5%	
Łączne obciążenie			41,5%	



Rysunek 10: System Dwuplanowy - Foreground



Rysunek 11: System Dwuplanowy - Background



Rysunek 12: Współrzędne wykonywanie zadań przez podział czasu procesora

Uwagi:

- gwarancja terminowości dla zadań planu pierwszego oraz możliwość wystąpienia nieprzewidywanych przerw w tle – możliwa dokładna analiza czasowa
- Potrzebny system operacyjny:
 - pomiar czasu
 - obsługa przerw
 - komunikacja (synchronizacja) zadań

3.1.3 Systemy wielozadaniowe (multitasking)

Wiele zadań:

- sterowanie
- archiwizowanie i dokumentowanie
- GUI
- komunikacja w sieci

Zadania nie są już tylko programami, które wykonują się od początku do końca bez przeszkód, lecz mają okresy zawieszania i oczekiwania na warunki zewnętrzne.

Nie można rozplanować kolejności wykonywania zadań przed uruchomieniem programu. Kolejność wykonywania zadań ustala System Operacyjny, biorąc pod uwagę priorytety przypisane zadaniom przez programistę.

Uwagi:

- Trudna (niemożliwa) analiza kolejności wykonania zadań – brak gwarancji dotrzymania ograniczeń czasowych.
- Trudne oszacowanie czasu wykonywania zadań
- Brak gwarancji terminowości (lub gwarancja dla wąskiego zbioru zadań)

3.2 Podsumowanie wstępu do Systemów Czasu Rzeczywistego

3.2.1 Ostre ograniczenia czasowe

Ostre ograniczenia czasowe – system gwarantuje dotrzymanie ograniczeń czasowych.

- zastosowanie - systemy mające wpływ na bezpieczeństwo:
 - systemy sterowania instalacjami przemysłowymi, transportowymi, wojskowymi czy medycznymi
 - systemy dowodzenia, np. wspomaganie kontrolerów ruchu lotniczego
- implementacja:
 - program sekwencyjny
 - systemy dwuplanowe
- przykłady urządzeń sterujących:
 - sterowniki PLC
 - sterowniki dedykowane

3.2.2 Łagodne ograniczenia czasowe

Łagodne ograniczenia czasowe – system stara się dotrzymywać ograniczeń czasowych, zadania pilne wykonywane są w pierwszej kolejności.

- zastosowanie - systemy mające wpływ na bezpieczeństwo:
 - interakcyjne systemy komercyjne, np. banki, rezerwacja miejsc lotniczych
 - systemy telekomunikacyjne
 - elektronika użytkowa, np. odtwarzacz CD
- implementacja:
 - systemy wielozadaniowe

3.2.3 Cechy Systemu Czasu Rzeczywistego

- deterministyczny czas operacji systemowej (kontrprzykład – pamięć wirtualna)
- kontrola czasu (kontrola operacji potencjalnie nieskończonych)
- ścisła kontrola programisty nad mechanizmem szeregowania zadań
- wspomaganie współpracy zadań (komunikacja i synchronizacja) i ograniczanie wynikającego stąd niedeterminizmu (które zadanie wznowić po podniesieniu semafora)
- dostęp do przerw i urządzeń we/wy
- konfiguralność systemu i zdolność do pracy w różnych konfiguracjach (host-target)

4 Wykład 4 [19.03.2004]

4.1 Model budowy Systemu Operacyjnego

4.1.1 Standard POSIX

Standard POSIX (IEEE/ANSI 1003.1) – opisuje system operacyjny od strony użytkownika, czyli API (Application Programming Interface) systemu operacyjnego.

1. Podstawowe rekomendacje przejęte z UNIX-a:

- *wielodostęp*: sesja, kontrola dostępu, zmienne środowiskowe
- *wielozadaniowość*: tworzenie i usuwanie procesów, zawieszanie i opóźnianie, sygnały, wątki, drzewo procesów
- *system plików*: przestrzeń nazw ścieżkowych, kontrola dostępu, podstawowe operacje, potoki
- *terminal*: relacje terminal-sesja-proces, operacje we/wy

2. Zalecenia do pracy w Systemie Czasu Rzeczywistego

- algorytmy szeregowania priorytetowego
- synchronizacja i komunikacja zadań (semafony i kolejki wiadomości)
- statyczny przydział pamięci (bez pamięci wirtualnej)
- dostęp do obszarów pamięci wspólnej
- programowe liczniki czasu (realizacja cykliczności)

4.1.2 Definicja programu i procesu

System wielozadaniowy – wykonuje współbieżnie wiele zadań.

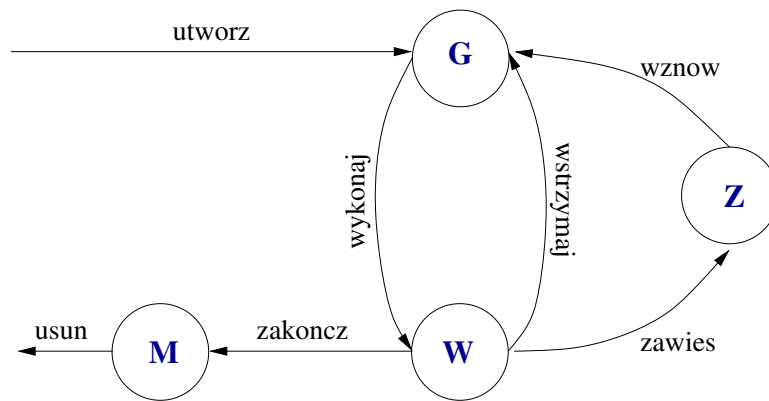
- *program* – statyczny zapis algorytmu w języku programowania
- *proces* – dynamiczne wykonywanie programu \approx *zadanie*

Proces:

- program
- obszar danych
- tablica stanu procesu

4.1.3 Stany procesu

- *gotowy* – gotowy do wykonania, tzn. wszystkie warunki do wykonania są spełnione, zasoby przydzielone, proces czeka tylko na procesor
- *wykonywany* – procesor wykonuje instrukcje zadania
- *zawieszony* – jakiś warunek wykonywania zadania nie jest spełniony, np. brak zasobu na który proces czeka
- *martwy* – proces zakończył wykonywanie, oczekuje na usunięcie z systemu, tzn. że pozostały jeszcze po nim jakieś ślady



Rysunek 13: Graf stanów zadania

4.1.4 Zdarzenia

Akcja systemu operacyjnego polega na rozpoznaniu zdarzeń i reagowaniu na nie, natomiast reakcja na zmianie stanu jakiegoś zadania.

Rodzaje zdarzeń

1. Zdarzenia zewnętrzne:

- odpowiadają wystąpieniom sytuacji wyjątkowych w otoczeniu komputera
- przykład: zakończenie operacji we/wy
- sygnalizowane są z reguły przerwaniami zewnętrznymi, przez przerwania zgłaszane przez urządzenia współpracujące, np. wyłączniki krańcowe

2. Zdarzenia czasowe:

- sygnalizują upływ określonych odcinków czasu
- przykład: upływ czasu Δt podanego w instrukcji *Delay* (Δt) wykonywanej przez zadanie
- sygnalizowane są z reguły przez moduł obsługi zegara

3. Zdarzenia wewnętrzne:

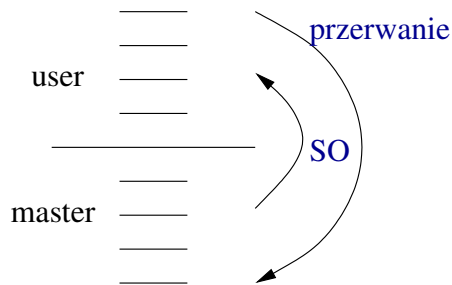
- odpowiadają błędom (wyjątkom) powstającym podczas wykonywania programów
- przykład: dzielenie przez zero
- sygnalizowane przez przerwania wewnętrzne, czyli przerwania generowane przez odpowiednie układy komputera, tj. koprocesor arytmetyczny lub zgłaszane przez podprogramy, np. podprogram dzielenia w podwójnej precyzji

4. Zdarzenia programowe:

- odpowiadają zdarzeniom zachodzącym podczas wykonywania zadań
- przykład: wysłanie wiadomości do innego węzła sieci
- generowane przez jawne odwołania zadań do systemu operacyjnego, przerwania programowe (trap)

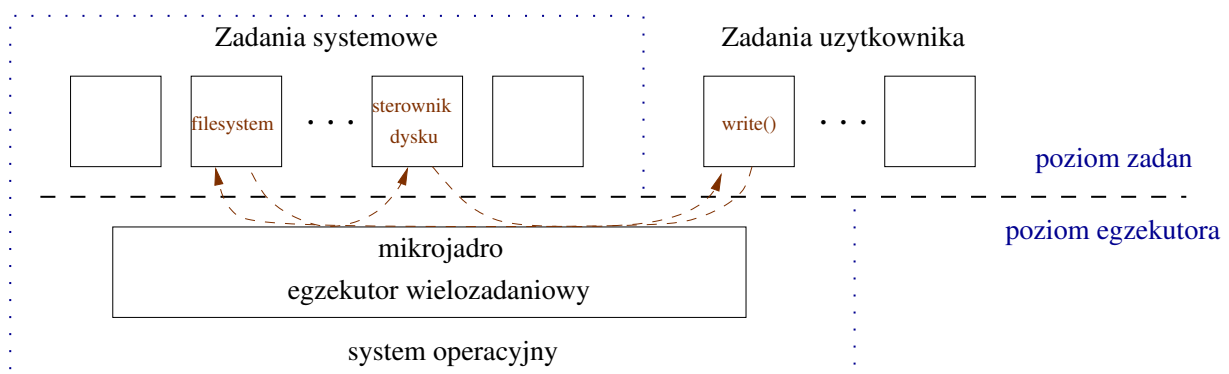
4.1.5 Tryby pracy systemu operacyjnego

- *master* – w tym trybie wykonywane są rozkazy uprzywilejowane, instrukcje obsługi pamięci, procesora, czy operacji we/wy
- *user* – program użytkownika wykonuje rozkazy uprzywilejowane poprzez przerwania programowe, przekazując sterowanie do systemu operacyjnego, który wykona instrukcje i powróci jak ze zwykłej funkcji



Rysunek 14: Tryby pracy SO

4.1.6 Model budowy systemu operacyjnego opartego na mikrojądrze



Rysunek 15: Model budowy systemu operacyjnego opartego na mikrojądrze

Funkcje jądra:

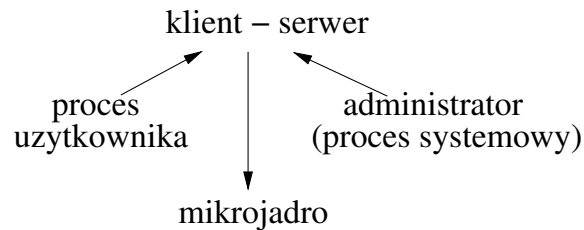
- podział czasu procesora
- elementarna komunikacja zadań

Przykłady zadań systemowych:

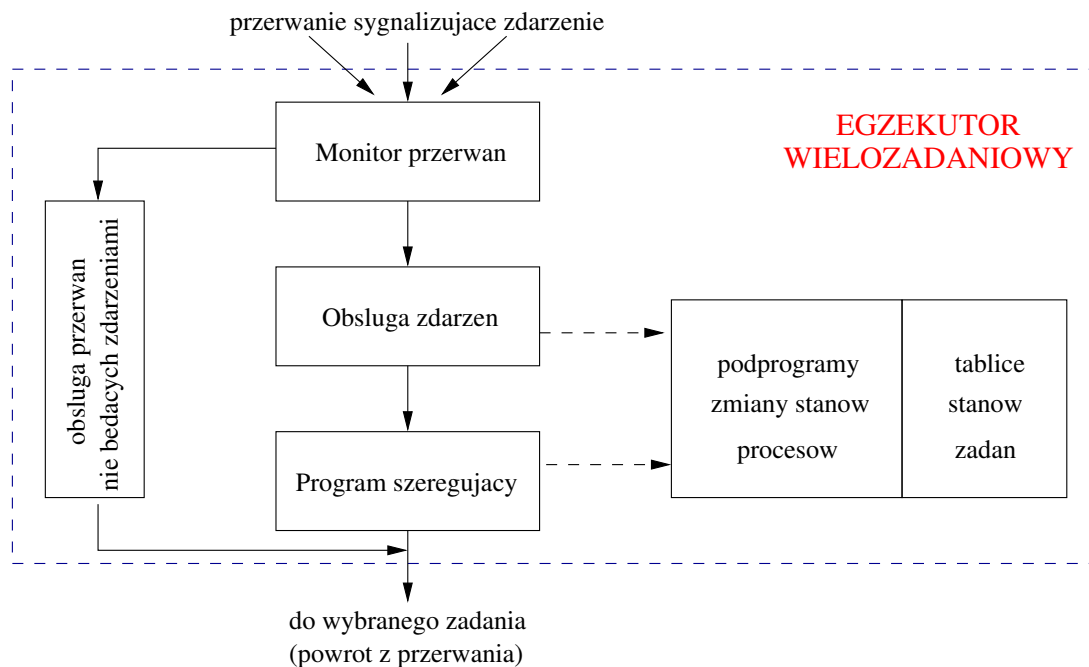
- sterownik dysków
- system plików
- sterownik karty sieciowej
- administrator sieci
- administrator zadań

- administrator narzędzi
- komunikacja zadań

Działanie systemu jest zgodne z modelem:



Rysunek 16: Model budowy systemu operacyjnego opartego na mikrojadrze

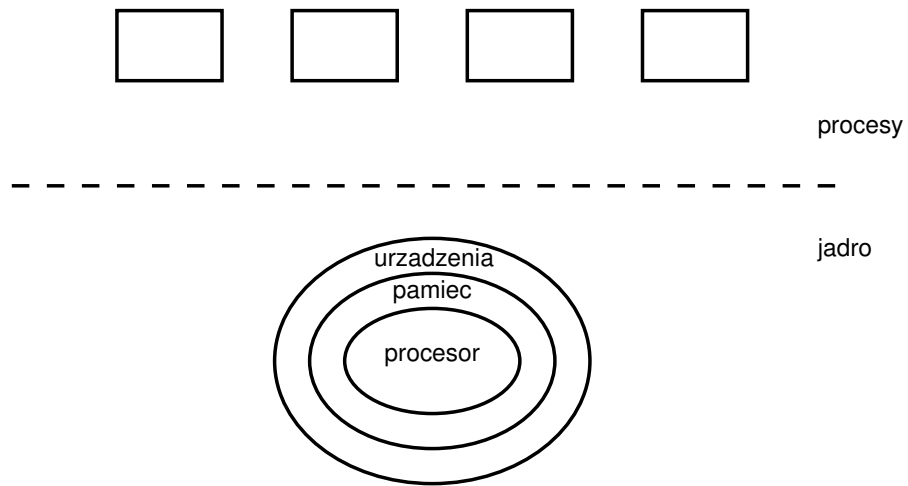


Rysunek 17: Model budowy egzekutora wielozadaniowego

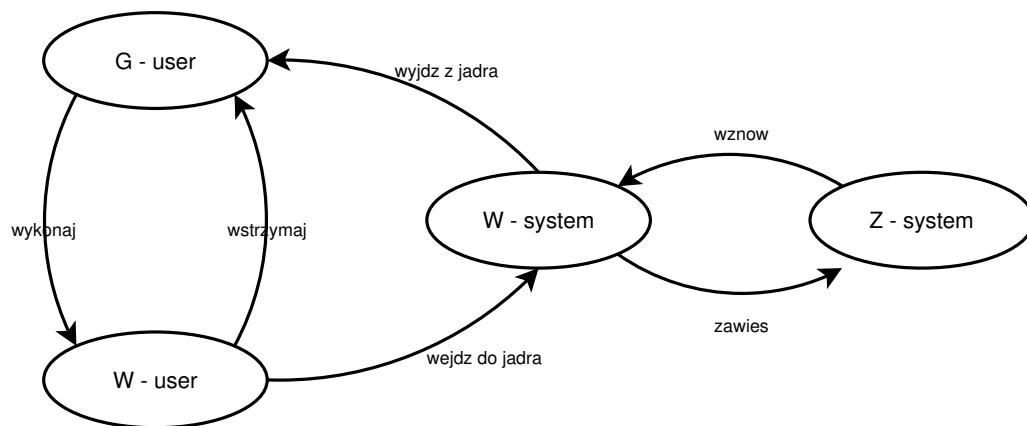
4.1.7 Model budowy systemu operacyjnego opartego jądrze monolitycznym

W systemie operacyjnym opartym na jądrze monolitycznym, jądro jest biblioteką funkcji wykonywanych w ciele zadania (zasada samoobsługi).

- Zalety mikrojadra
 - modularyzacja i bezpieczeństwo
 - krótki czas operacji jądra
 - łatwość rekonfiguracji
- Zalety jądra monolitycznego
 - wysoka przepustowość



Rysunek 18: Model budowy systemu operacyjnego opartego na jądrze monolitycznym



Rysunek 19: Graf stanów zadania w systemie opartym na jądrze monolitycznym

- Wady jądra monolitycznego
 - długi, trudny do określenia czas operacji jądra (z wyłączonymi przerwaniem)

5 Wykład 5 [26.03.2004]

5.1 Tablice systemowe

1. **Tablica stanu procesora** – zwana inaczej blokiem kontrolnym zadania (TCB – Task Control Block), deskryptorem zadania (TD – task descriptor) lub tablicą stanu zadania (TST – Task state table) – zawiera „metryczkę” zadania, czyli wszystkie informacje o nim:
 - priorytet zadania (task priority)
 - stan zadania
 - kontekst zadania (PSW – Program Status World) – zawartość rejestrów procesora
 - opis zależności czasowych
 - opis przydzielonych obszarów pamięci
 - opis zasobów wykorzystywanych przez zadanie
2. **Lista zadań gotowych** (RL - Ready List) – zawierająca adresy tablic stanu zadań gotowych do wykonania
3. **Lista zadań zawieszonych** – każda przyczyna zawieszenia posiada własną, odrębną listę
4. **Identyfikator zadania** (PID – process identifier, TID – task identifier) – wskazuje zadania, pozwala systemowi znaleźć tablicę stanu szukanego zadania
5. **Identyfikator zadania wykonywanego** (RT - Running Task)
6. Inne tablice systemowe:
 - katalog plików
 - mapa zajętości pamięci

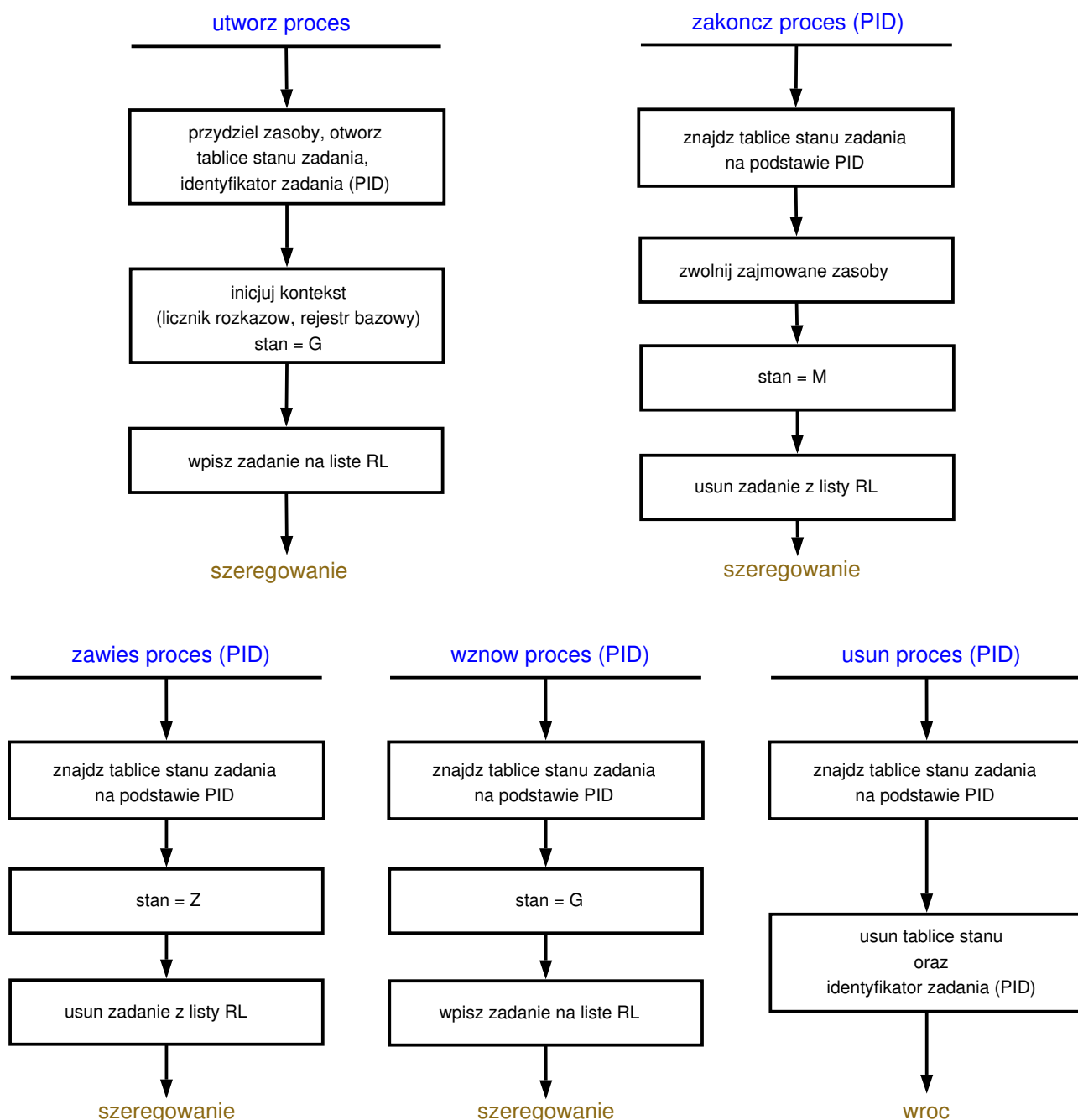
5.2 Wątek a proces

Proces (process, task):

- program
- zasoby

Wątek (thread):

- pojedyncza ścieżka wykonywania programu sekwencyjnego
- wielowątkowość oznacza wykonywanie współbieżne wielu wątków w obrębie jednego procesu
- wątek może być: gotowy-wykonywany-zawieszony



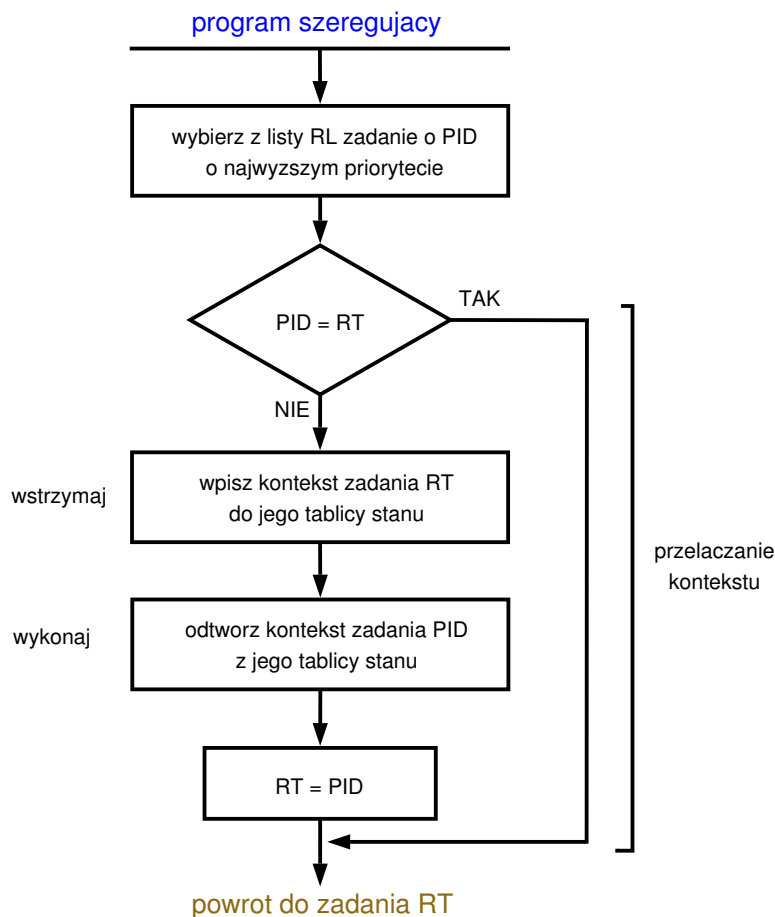
Rysunek 20: Tworzenie, zakańczanie, zawieszanie, wznowianie i usuwanie procesu

5.3 Model działania egzekutora

Tworzenie, zakańczanie, zawieszanie, wznowianie i usuwanie procesu przedstawione są na rysunku nr 20, natomiast program szeregujący na rysunku nr 21.

Istotnym problemem implementacji jest zapewnienie egzekutorowi wyłącznego dostępu do danych systemowych. Problem ten może wystąpić w dwóch aspektach:

1. zadania użytkowe mogą próbować zmieniać tablice lub program egzekutora – skuteczna ochrona przed taką ewentualnością wymaga wykorzystania sprzętowego mechanizmu ochrony pamięci.
2. zgłoszenie przerwania podczas wykonywania programu egzekutora może doprowadzić do wstrzymania bieżącego wykonywania egzekutora i rozpoczęcia nowego wykonania, doprowadzając w efekcie do współbieżnego wykonania dwóch „kopii” egzekutora.



Rysunek 21: Schemat programu szeregującego

W systemie jednoprocessorowym może być skutecznie rozwiązany przez *wyłączenie układu przerwań* podczas wykonania egzekutora.

W systemie wieloprocessorowym ze wspólną pamięcią sytuacja jest bardziej złożona. Poszczególne procesory działają niezależnie i mogą w tym samym czasie przejść do wykonywania programu egzekutora. Istnieją dwa konkurencyjne rozwiązania:

1. INTEL – zajmowanie dostępu do magistrali:

- Bus Lock
- Bus Unlock

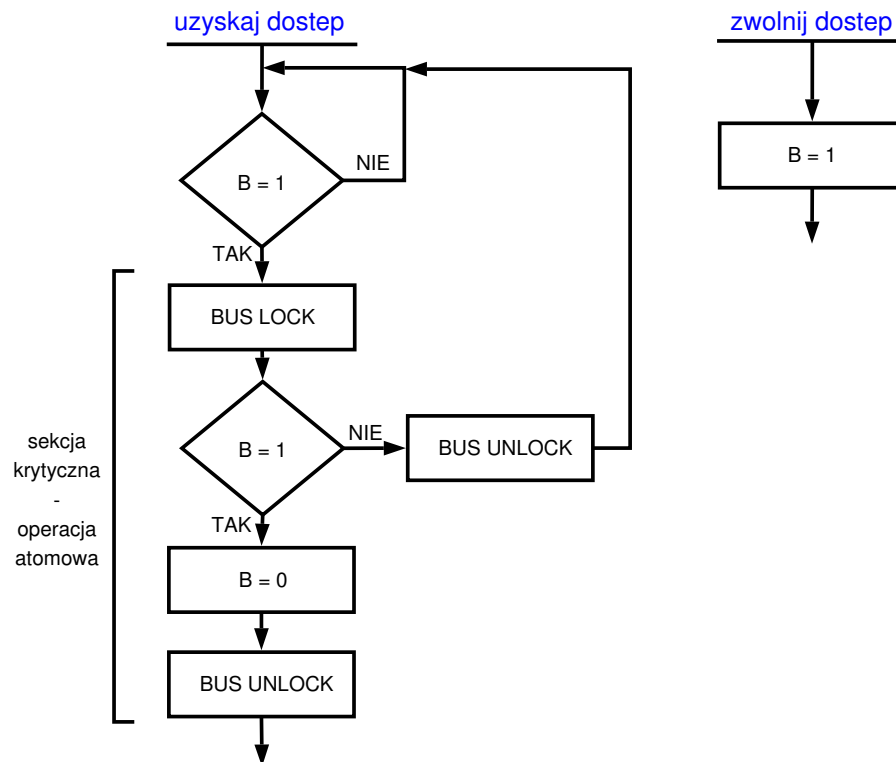
Bezpośrednie wykorzystanie tych operacji do uzyskania wyłącznego dostępu do danych systemowych, prowadziłoby do niepotrzebnego blokowania całej pamięci i znacznego obniżenia wydajności systemu. Dlatego lepszym rozwiązaniem jest implementacja *semaforów binarnych*, zapewniających wzajemne wykluczanie wybranych fragmentów programu bez blokowania magistrali systemowej na długi czas (rysunek nr 22).

Semafor binarny – jest dwuwartościową zmienną przechowywaną w pamięci wspólnej, opi-

sującą stan zasobu lub urządzenia:

$$B = \begin{cases} 1 & - \text{dostęp wolny} \\ 0 & - \text{dostęp zabroniony} \end{cases}$$

2. MOTOROLA – specjalna niepodzielna instrukcja procesora *exchange* zamieniająca zawartość komórki pamięci z zawartością pamięci.



Rysunek 22: Schemat semafora aktywnego

6 Wykład 6 [02.04.2004]

6.1 Tworzenie procesów

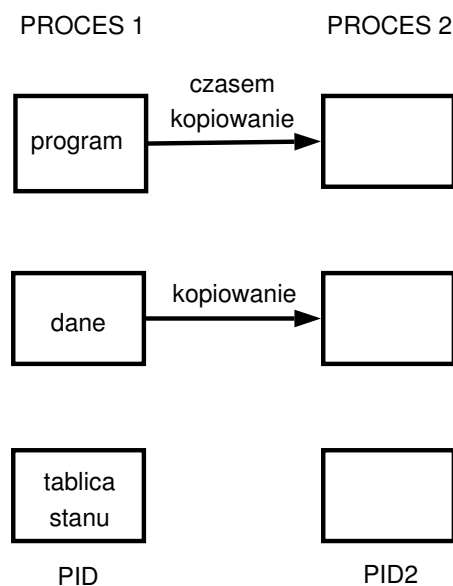
Utworzenie nowego procesu polega na załadowaniu programu tego procesu do pamięci operacyjnej, wstępnym wypełnieniu struktur systemowych opisujących stan procesu i określeniu numerycznego identyfikatora (PID - Proces identifier).

6.1.1 Funkcja FORK

```
#include <sys/types.h>
#include <unistd.h>
```

```
pid_t fork (void)
```

Funkcja *fork* tworzy i uruchamia nowy proces, będący dokładną kopią procesu macierzystego. Utworzony proces potomny wykonuje się współbieżnie z procesem macierzystym. Można powiedzieć, że jest to rozdwojenie procesu, o obszarze programu i danych takim samym, a z innymi PIDem. Program może być dzielony, lub czasem kopiowany.



Rysunek 23: Tworzenie procesu - FORK

Przykładowe użycie funkcji *fork*:

```
x = fork (void)

if (x==0)
{
    /* program procesu potomnego */
}
else
{
    /* program procesu macierzystego */
}
```

Jak widać w powyższym przykładzie, po tym co zwróci *fork* można poznać, czy jest to proces potomny, czy macierzysty:

$$x = \begin{cases} pid2 & - \text{zwraca PID procesu potomnego w procesie macierzystym} \\ 0 & - \text{zwraca w procesie potomnym} \end{cases}$$

6.1.2 Funkcje EXEC...

#include <process.h>

```
int execl (pgm, arg0, arg1, ..., argn, NULL);
int execlp (pgm, arg0, arg1, ..., argn, NULL, envp);
int execlpe (pgm, arg0, arg1, ..., argn, NULL, envp);
int execlpe (pgm, arg0, arg1, ..., argn, NULL, envp);
int execv (pgm, argv);
int execve (pgm, argv, envp);
int execvp (pgm, argv);
int execvpe (pgm, argv, envp);
```

- *pgm* – nazwa pliku
- *arg0, ..., argn* – argumenty
- *argv[]* – tablica argumentów
- *envp[]* – ustawienia środowiska

Funkcje z serii **exec**_{xxx} powodują odczytanie programu *pgm*, z argumentami *arg0, ..., argn* lub *argv[]* i utworzenie procesu, wykonującego ten program. Jeżeli operacja zakończy się sukcesem, to proces macierzysty, który wywołał funkcję **exec** zostanie usunięty, a jego miejsce zajmie proces potomny, wykonujący się z tym samym identyfikatorem PID.

Dokładny opis dziedziczenia parametrów środowiska, przy tworzeniu nowych procesów znajduje się na stronie 27 w „Laboratorium systemu QNX” [2].

6.2 Tworzenie wątków

#include <pthread.h>

```
int pthread_create( pthread_t* thread,
                  const pthread_attr_t* attr,
                  void* (*start_routine)(void* ),
                  void* arg );
```

- *thread* – identyfikator pozwalający odwołać się do wątku
- *attr* – atrybuty, wymagające inicjalizacji oraz ustawienia przed wywołaniem *pthread_create*
- *start_routine* – główna funkcja wątku, ciało wątku
- *arg* – argumenty funkcji *start_routine*

Inne funkcje związane z wątkami:

```
int pthread_join( pthread_t thread ,
                 void** value_ptr );
```

Funkcja *pthread_join* – pozwala w programie głównym oczekiwać na zakończenie wątków.

```
void pthread_exit( void* value_ptr );
```

Funkcja *pthread_exit* – zakończenie wątku.

```
int pthread_detach( pthread_t thread );
```

Funkcja *pthread_detach* – zezwolenie na zwolnienie pamięci, którą wątek zajął.

6.3 Algorytmy szeregowania procesów

Wchodzący w skład jądra egzekutor szereguje procesy *Gotowe* na poziomach priorytetów (w QNX są to 32 poziomy). Procesy o wyższym poziomie priorytetu zawsze wywłaszczają procesy o poziomie niższym. Procesy na tym samym poziomie priorytetów mogą być szeregowane zgodnie z jednym trzech różnych algorytmów.

- **Algorytm karuzelowy (RR - Round Robin)** – procesy wybiera się do wykonywania kolejno, przy czym każdy z nich otrzymuje ograniczony kwant czasu Δt . Jeżeli wykonywany proces nie zakończy się lub nie zawiesi się wcześniej, to w chwili wyczerpania kwantu system operacyjny wywłaszcza go i podejmuje wykonanie następnego procesu.

Cechy charakterystyczne:

- stały priorytet
- stały kwant czasu Δt

- **Algorytm FIFO (First In First Out)** – do wykonania wybiera się proces najdłużej oczekujący w stanie *Gotowy*. Wybrany proces wykonuje się aż do zakończenia lub zawieszenia.

Cechy charakterystyczne:

- stały priorytet
- brak wywłaszczania w ramach poziomu
- główną zaletą jest zminimalizowanie liczby przełączeń procesora

- **Algorytm adaptacyjny** – procesy wybiera się do wykonania kolejno, podobnie jak w algorytmie karuzelowym. Po wyczerpaniu kwantu czasu system operacyjny wywłaszcza wykonywany proces i obniża jego priorytet o 1. Jeżeli wykonanie procesu o obniżonym priorytecie nie zostanie podjęte w ciągu $2s$, jego priorytet podnosi się o 1 (ale nigdy powyżej oryginalnego priorytetu). Proces zawieszony odzyskuje natychmiast swój oryginalny priorytet.

Cechy charakterystyczne:

- zmienny priorytet
- stały kwant czasu Δt
- manipulacja priorytetami aby przyspieszyć działanie

Standard POSIX zakłada konieczność implementacji algorytmów karuzelowego (RR) i FIFO, dodatkowo producent może sobie dodać dowolny inny, najczęściej jest to algorytm adaptacyjny.

W chwili uruchomienia nowy proces dziedziczy priorytet po procesie macierzystym. W systemie QNX, standardowo, procesy uruchamiane przez *shell* otrzymują priorytet 10. Zarówno priorytet, jak i algorytm szeregowania stanowią parametry aktualnie wykonywanego procesu i mogą być dynamicznie zmieniane podczas pracy systemu za pomocą odpowiednich funkcji systemowych. Oznacza to, że decyzja o wyborze algorytmu szeregowania jest lokalna w procesie.

6.4 Mechanizm sygnałów

Sygnały są podstawowym środkiem informowania procesów o błędach i nienormalnych stanach powstających podczas ich wykonywania. Funkcjonalnie, sygnały przypominają więc przerwania programowe, generowane przez programy jądra, procesy systemowe i procesy użytkowe. Wygenerowany sygnał jest dostarczony przez jądro do procesu, dla którego jest przeznaczony. Sposób reakcji procesu na dostarczony sygnał zależy od wybranego sposobu obsługi. Jeżeli nie zostało to określone inaczej, system operacyjny podejmuje akcję domyślną, która polega na ogół na zakończeniu procesu. Alternatywnie, proces może zignorować dostarczony sygnał lub zdefiniować dla niego własną funkcję (podprogram obsługi). W tym ostatnim wypadku, dostarczenie sygnału powoduje przerwanie normalnego toku wykonywania procesu i przekazanie sterowania do funkcji obsługi. Po zakończeniu funkcji obsługi, wykonywanie procesu kontynuuje się od miejsca przerwania.

Generacja sygnałów:

- sprzętowo (system operacyjny)
- *raise(*S*)* – wygenerowanie sygnału *S* w obrębie procesu (np. dla poinformowania procesu o błędzie wynikłym podczas wykonywania podprogramu)
- *kill(*T,S*)* – wygenerowanie sygnału *S* skierowanego do procesu *T*

Obsługa sygnałów:

- domyślnie – zakończenie zadania
- ignorowanie
- przechwycenie:
 - wykonanie funkcji obsługi
 - wznowienie (jeśli zawieszony, np. *pause()*)

Funkcja obsługi sygnału:

```
void handler (int signo)
{
    /* ograniczony zbior funkcji */
}
```

Funkcje systemowe dozwolone wewnątrz funkcji obsługi sygnału są zawarte w Tabeli 2.5 w „Laboratorium systemu QNX” [2].

Funkcja obsługi sygnału jest częścią programu procesu i może używać tych samych zmiennych, zgodnie z ogólnymi zasadami widoczności zmiennych w języku C. Ponieważ jednak wykonanie funkcji obsługi sygnału przebiega asynchronicznie w stosunku do wykonania procesu, przetwarzanie tych

samych zmiennych może prowadzić do błędu. Aby temu zapobiec, wprowadzono specjalny typ całkowity *sig_atomic_t*, w którym podstawowe operacje, np. zwiększania i zmniejszania wartości, są niepodzielne.

Sposoby powrotu z funkcji obsługi:

1. powrót do punktu przerywania
2. skok do zaznaczonego wcześniej miejsca programu
 - *sigsetjmp()* – zapamiętanie adresu powrotu i aktualnej maski sygnałów
 - *siglongjmp()* – powrót z odtworzeniem maski sygnałów

Maskowanie sygnałów:

- inicjowanie zbioru, dodawanie, usuwanie sygnałów:
 - *sigempty(SS)* – zainicjowanie pustego zbioru sygnałów *SS*
 - *sigfillset(SS)* – zainicjowanie zbioru sygnałów *SS*, zawierającego wszystkie sygnały
 - *sigaddset(SS,S)* – dodanie sygnału *S* do zbioru *SS*
 - *sigdelset(SS,S)* – usunięcie sygnału *S* ze zbioru *SS*
- ustalanie i podmienianie maski:
 - *sigprocmask(SS,S)* – ustalenie maski równej zbiorowi sygnałów *SS*
 - *sigsuspend(SS)* – podmienia maskę sygnałową procesu na *SS* i zawiesza proces aż do otrzymania sygnału.
- funkcje pomocnicze:
 - *sigismember(SS,S)* – sprawdzenie, czy sygnał *S* należy do zbioru *SS*
 - *sigpending(SS)* – ustalenie i zapisanie w zbiorze *SS* zgłoszonych, a zamaskowanych sygnałów, oczekujących na dostarczenie procesu

Aby zamaskować sygnał należy:

1. zdefiniować zbiór sygnałów
 - utworzyć zbioru
 - wprowadzić sygnały
2. ustawić maskę

6.5 Przegląd sygnałów

- Błędy sprzętowe:
 - **SIGFPE** [--] błąd operacji arytmetycznej (np. dzielenie przez zero lub nadmiar)
 - **SIGSEGV** [--] naruszenie ochrony pamięci
 - **SIGILL** [--] próba wykonania niepoprawnej instrukcji
- Błędy programowe:

- **SIGALRM** [-] przeterminowanie (np. funkcja *alarm*)
- **SIGPIPE** [-] próba zapisu do potoku (*pipe*), który nie został nigdzie otwarty do odczytu
- Zakończenie procesu:
 - **SIGTERM** [-] normalne zakończenie procesu
 - **SIGABRT** [-] nienormalne zakończenie procesu (np. przez wywołanie funkcji *abort*)
 - **SIGQUIT** [-] zakończenie procesu przez operatora
 - **SIGKILL** [-!] awaryjne zakończenie procesu
- Zdarzenia:
 - **SIGINT** [-] przerwanie z klawiatury (klawisz *Brake*)
 - **SIGHUP** [-] zakończenie procesu wiodącego sesji (z reguły *sh*) lub brak komunikacji z terminalem sesji
 - **SIGCHLD** [] zakończenie procesu potomnego
- Sterowanie procesem:
 - **SIGSTOP** [!] zawieszenie procesu w oczekiwaniu na sygnał **SIGCONT**
 - **SIGCONT** [!] wznowienie procesu zawieszzonego przez sygnał **SIGSTOP**
- Dowolna sygnalizacja:
 - **SIGUSR1** [-] do zdefiniowania przez użytkownika
 - **SIGUSR2** [-] do zdefiniowania przez użytkownika
- QNX niestandardowe:
 - **SIGBUS** [--] błąd parzystości pamięci
 - **SIGPWR** [-] żądanie restartu systemu wywołane przez naciśnięcie klawiszy *Ctrl-Alt-Shift-Del* lub wywołany proces *shutdown*

Wyjaśnienie przyjętych symboli:

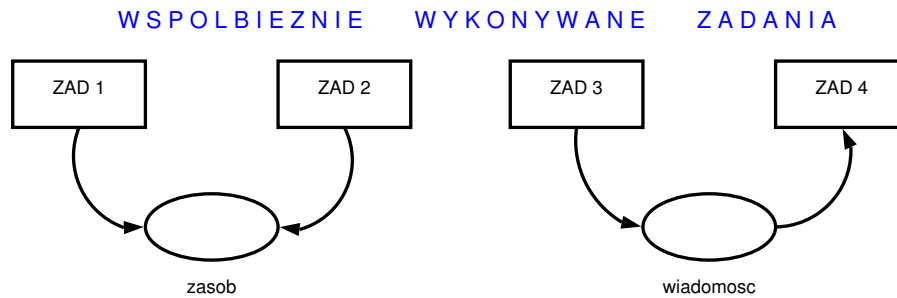
- [--] – pierwsze wystąpienie to błąd, natomiast kolejne kończy proces
- [-] – domyślna akcja – usunięcie procesu
- [-!] – bezwzględne usunięcie z systemu
- [!] – te sygnały nie mogą być przechwytywane przez normalne procesy

7 Wykład 7 [16.04.2004]

7.1 Synchronizacja i komunikacja zadań

W systemie wielozadaniowym współpracujące zadania wykonują się współbieżnie, czyli wchodzi w interakcję. W takiej sytuacji możemy mieć do czynienia z następującymi problemami:

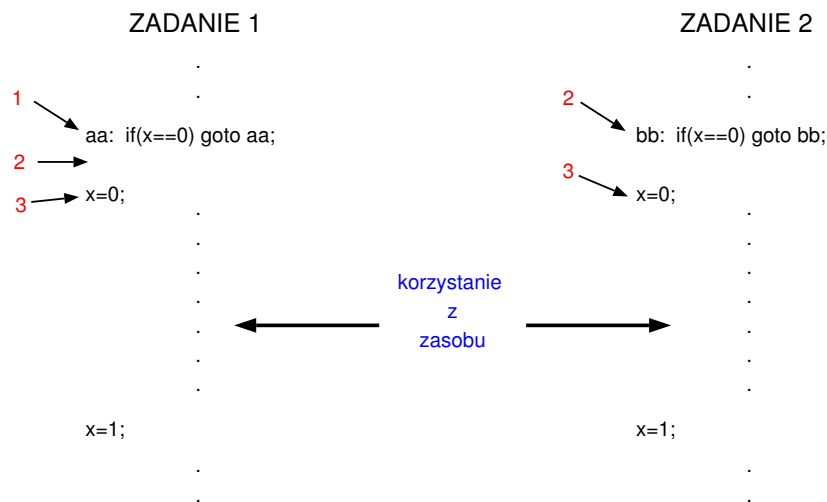
1. konkurencja o dostęp do wspólnych zasobów (synchronizacja)
2. współpraca procesów współbieżnych (komunikacja + synchronizacja)



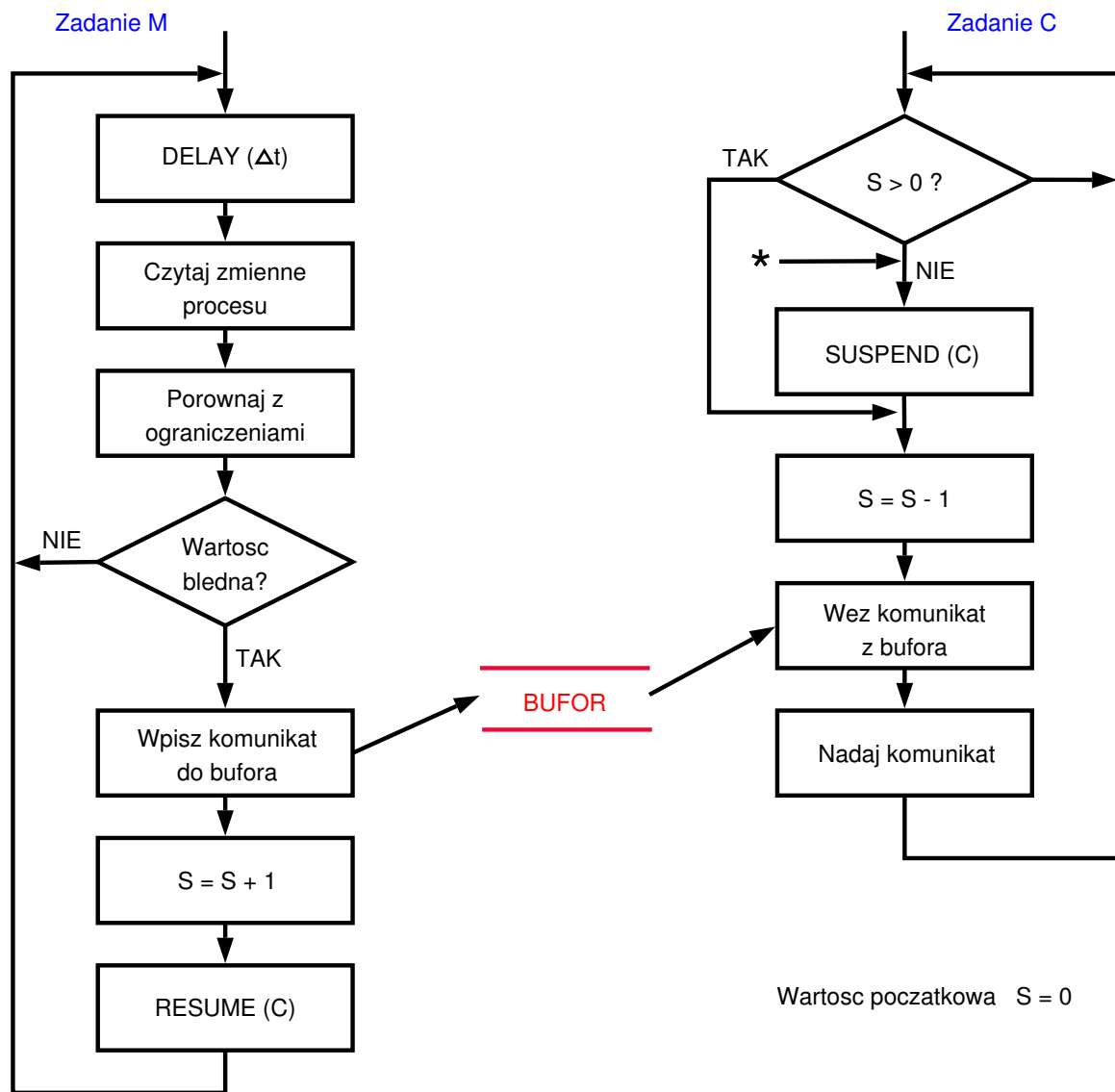
Rysunek 24: Problemy synchronizacji i komunikacji zadań

Przymijmy, że znacznik x przyjmuje następujące wartości:

$$x = \begin{cases} 1 & \text{— wolny} \\ 0 & \text{— zajęty} \end{cases}$$



Rysunek 25: Błędna synchronizacja zadań



Rysunek 26: Błędna synchronizacja zadań w systemie rejestracji

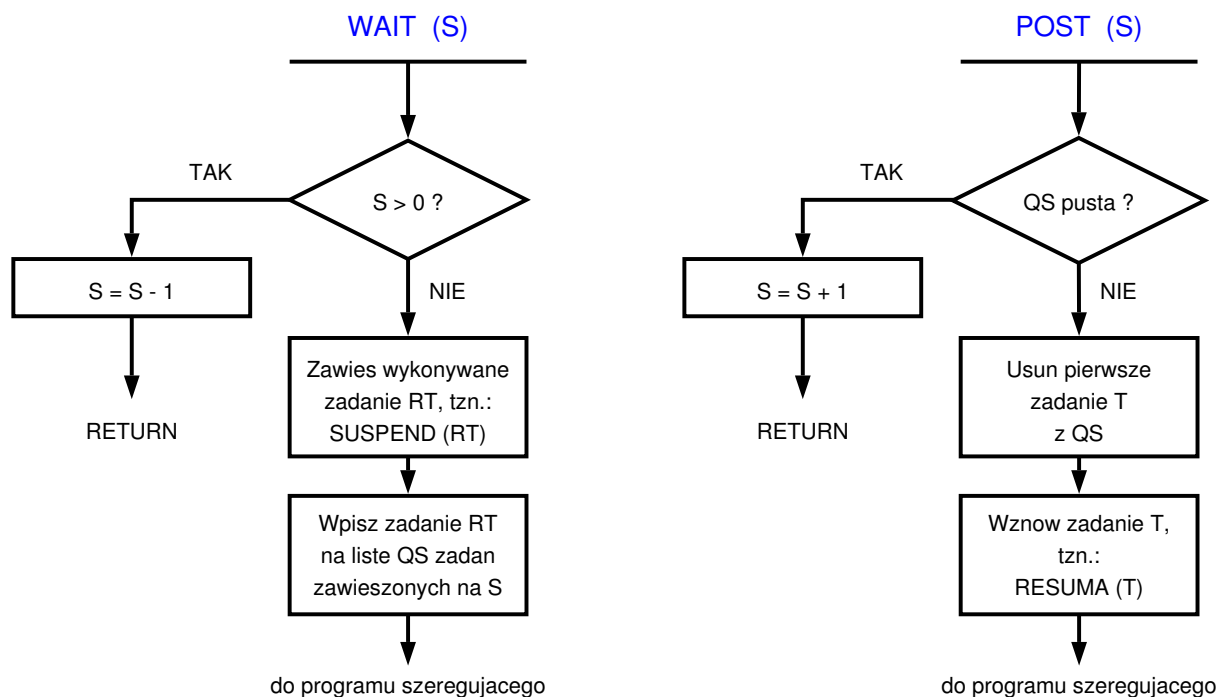
W przykładzie podanym na rysunku nr 26 pokazany jest przykład błędnego rozwiązania problemu współpracy przy pomocy instrukcji *Suspend* (zawieś zadanie) i *Resume* (wznów zadanie). Jeżeli następna wartość nieprawidłowa pojawi się i spowoduje wykonanie instrukcji *Resume* po nadaniu ostatniego komunikatu i sprawdzeniu licznika, a przed wykonaniem instrukcji *Suspend*, to operacja *Resume* zawiesi zadanie wykonanie zadania nadającego pomimo obecności komunikacji w buforze.

7.1.1 Semafor

$$S = \begin{cases} > 0 & \text{— zezwolenie na kontynuację} \\ = 0 & \text{— brak zgody na kontynuację} \end{cases}$$

Semafor (semaphore) S jest zmienną systemową, której wartość może być sprawdzana i zmieniana przez wszystkie zadania za pomocą jednej z dwóch instrukcji:

operacje niepodzielne	[$wait(S) : \text{ if } S > 0$	$\text{ then } S = S - 1$
			$\text{ else } \text{zawieś wykonywanie zadania}$
		$post(S) : \text{ if } \exists \text{ zadanie zawieszone}$	$\text{ then wznów jedno zadanie}$
			$\text{ else } S = S + 1$

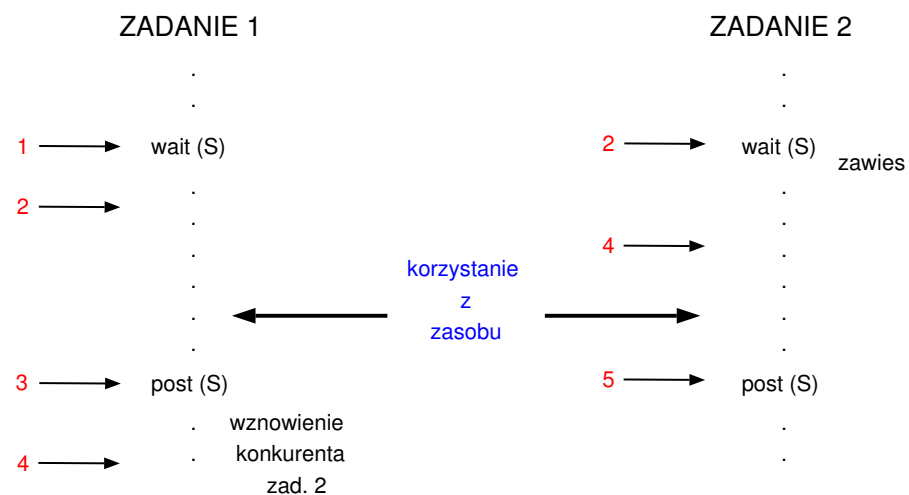


Rysunek 27: Operacje semaforowe

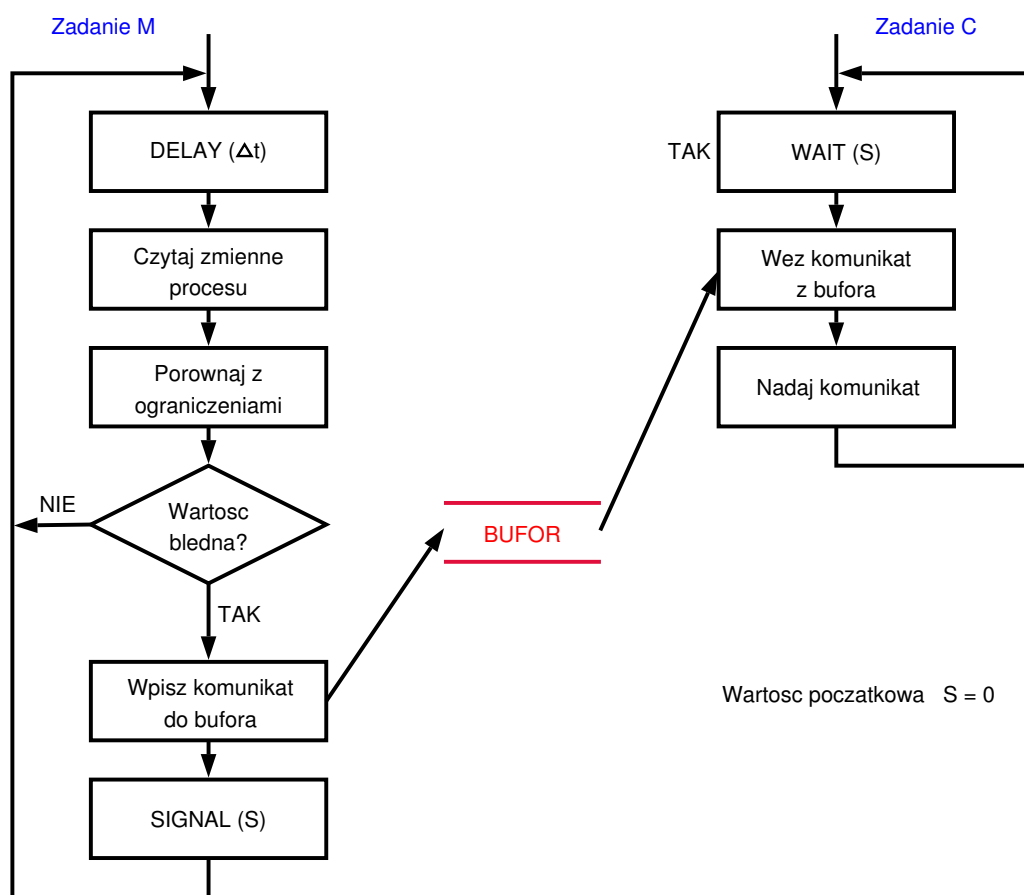
Dopiero standard **POSIX** wprowadził nazwy operacji semaforowych jako *wait* oraz *post*. We wcześniejszych publikacjach można spotkać inne ich oznaczenia. Odpowiednio *P* (*wait*) i *V* (*post*), *wait* i *signal* oraz *wait* i *event*.

Sposób implementacji operacji semaforowych przedstawia rysunek 27. Lista zadań zawieszonych na semaforze może być obsługiwana zgodnie z algorytmem **FIFO** lub też mogą być brane pod uwagę priorytety zadań. Zaletą tego pierwszego sposobu jest prostota i „uczciwość” – tzn. żadne zadanie zawieszone nie będzie w nieskończoność czekać na wznowienie, ustępując miejsca innym wznowianym zadaniom. Wadą jest możliwość wstrzymywania zadania o wyższych priorytetach przez wcześniej zawieszone zadania o niższych priorytetach. Cechy algorytmu priorytetowego są przeciwstawne. W praktyce, za najważniejszą cechę jest uznawana „uczciwość” i najczęściej stosowany jest algorytm FIFO.

Niepodzielność operacji *wait* oraz *post* oznacza niemożliwość jednoczesnego wykonywania dwóch dowolnych operacji na tym samym semaforze przez więcej niż jedno zadanie. Rozwiązanie rozważanych poprzednio problemów współpracy zadań przy pomocy semafora przedstawiają rysunki nr 28 oraz 29. Początkowa wartość semafora *S* wynosi 0.



Rysunek 28: Poprawna synchronizacja zadań



Rysunek 29: Poprawna synchronizacja zadań w systemie rejestracji

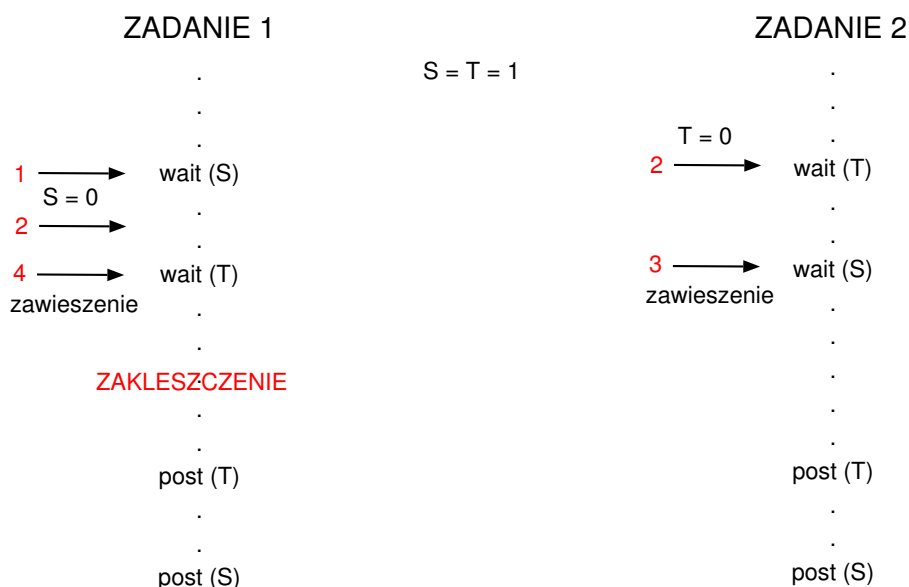
Implementacja semaforów w systemie QNX jest w pełni zgodna z opisanym schematem. Wszystkie operacje na semaforach są realizowane przez następujące funkcje systemowe:

- *sem_init* (*S*, *V*) – utworzenie semafora *S* z pustą kolejką procesów zawieszonych i wartością początkową *V*
- *sem_destroy* (*S*) – usunięcie semafora *S*
- *sem_post* (*S*) – wykonanie operacji *post* na semaforze *S*
- *sem_wait* (*S*) – wykonanie operacji *wait* na semaforze *S*
- *sem_trywait* (*S*) – wykonanie operacji *wait* bez zawieszania procesu; jeżeli wartość semafora *S* nie jest większa od zera, funkcja wraca natychmiast zwracając kod błędu

Wykonanie funkcji *sem_wait* może być przerwane przez dostarczenie sygnału, tzn. dostarczenie sygnału do procesu zawieszonego po wykonaniu funkcji *sem_wait* powoduje wznowienie tego procesu i zakończenie funkcji ze zwróceniem odpowiedniego kodu błędu.

7.1.2 Anomalia synchronizacji

Zakleszczenie (deadlock) jest sytuacją w której zawieszone zadania oczekują na wznowienie przez inne zadania, które z kolei są zawieszone w oczekiwaniu na wznowienie przez zadania pierwszej grupy.



Rysunek 30: Zakleszczenie

Antidotum

1. Numerowanie semaforów – zajmowanie w kolejności rosnącej
2. Analiza projektu (graf zadań – procesu + strzałki określające kolejność czekania „*kto na kogo czeka*” – jeśli w grafie istnieje pętla to może wystąpić zakleszczenie)
3. Jeśli System Operacyjny podejrzewa możliwość wystąpienia zakleszczenia – usuwa jeden z procesów (metoda bardzo brutalna, mogąca doprowadzić do błędów w obliczeniach)

8 Wykład 8 [23.04.2004]

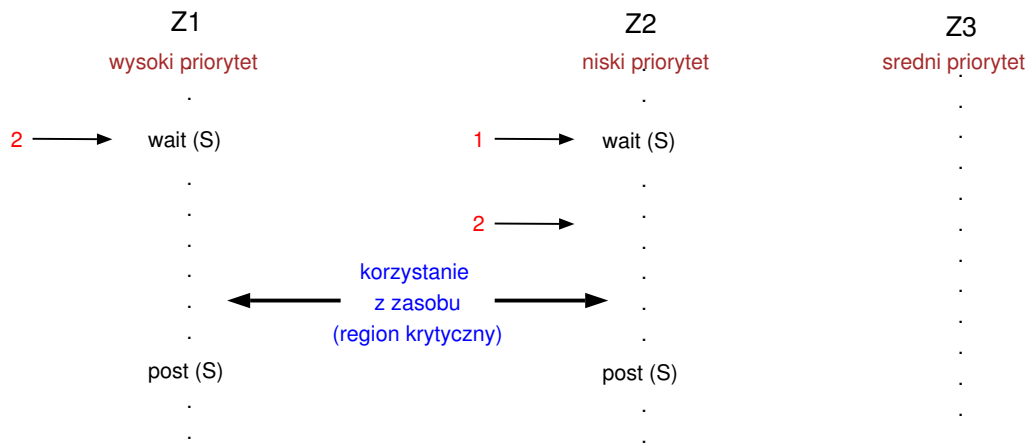
8.1 Synchronizacja i komunikacja zadań - cd

8.1.1 Anomalia synchronizacji

Inwersja priorytetów (priority inversion) – sytuacja, gdy zadanie o niższym priorytecie wstrzymuje wykonywanie zadania o priorytecie wyższym.

Dla przykładu rozważmy problem wykluczania zadań (rysunek nr 31) i przypuścimy, że równoległe z zadaniem $Z1$ o priorytecie wysokim i zadaniem $Z2$ o priorytecie niskim wykonuje się zadanie $Z3$ o priorytecie średnim. W takiej sytuacji możliwa jest następująca sekwencja zdarzeń:

1. Wykonywane zadanie $Z2$ dochodzi do instrukcji *wait* i wchodzi do regionu krytycznego.
2. Startuje zadanie $Z1$, dochodzi do instrukcji *wait* i zostaje zawieszone w oczekiwaniu na zwolnienie regionu.
3. Startuje zadanie $Z3$ i ze względu na wyższy priorytet powoduje wstrzymanie zadania $Z2$



Rysunek 31: Inwersja priorytetów

W tym stanie zadanie $Z3$ uniemożliwia wykonywanie zadania $Z2$, które zajmuje region i wstrzymuje zadanie $Z1$. W konsekwencji zadanie $Z1$ o wysokim priorytecie jest wstrzymywane przez zadanie $Z3$ o priorytecie niższym.

Nie ma na to ogólnego rozwiązania. Jednym ze sposobów uniknięcia opisanej sytuacji jest sposób wykorzystany w systemie **iRMX86**, polegający na wykorzystaniu **regionów** wzajemnego wykluczania zadań.

Każdy region krytyczny powinien zaczynać się instrukcją *receive-control* i kończyć instrukcją *send-control* o następującej interpretacji:

receive-control(R) – sprawdź stan regionu i kontynuuj wykonanie, jeżeli region jest wolny, lub zawieś zadanie, jeżeli region jest zajęty (tzn. inne zadanie wykonało wcześniej instrukcję *receive-control(R)*). Priorytet zadania znajdującego się wewnątrz regionu jest podnoszony do poziomu priorytetu zadania zawieszonego.

send-control(R) – zwolnij region i wznów jedno z zadań oczekujących na wejście do regionu (jeżeli takie zadania istnieją).

Semantyka instrukcji *receive-control* i *send-control* jest analogiczna do semantyki instrukcji *wait* i *post* w przypadku semaforów. Zmiana priorytetu zadania znajdującego się wewnątrz regionu wyklucza jednak możliwość powstania inwersji priorytetów. Rozważana wcześniej sytuacja będzie miała teraz postać:

1. Wykonanie zadania $Z2$ dochodzi do instrukcji *receive-control* i zajmuje region.
2. Startuje zadanie $Z1$, dochodzi do instrukcji *receive-control* i zostaje zawieszone w oczekiwaniu na zwolnienie regionu. Priorytet zadania $Z2$ zostaje podniesiony do poziomu priorytetu zadania $Z1$
3. Startuje zadanie $Z3$, lecz ze względu na wyższy priorytet zadania $Z2$ oczekuje na wykonywanie w stanie gotowości.

Zadanie $Z2$ biegnie dalej, aż do opuszczenia regionu. Wykonywanie instrukcji *send-control* zwalnia region, powoduje obniżenie poziomu priorytetu zadania $Z2$ do początkowego, niskiego poziomu i wznawia zadanie $Z1$. Wszystkie trzy zadania są w stanie *Gotowe* i do wykonania jest wybierane zadanie $Z1$.

8.1.2 Kolejki wiadomości

W wielu zastosowaniach dane przekazywane między procesami poprzez zbiór pośredniczący mają postać wiadomości o ustalonej i znanej obydwu procesom długości i strukturze wewnętrznej. Zbiór pośredniczący staje się w takim wypadku **kolejką wiadomości** (*message queue*), operacje zapisania i odebrania danych przybierają postać operacji nadania i odebrania wiadomości. Często spotykaną formą współpracy procesów jest wymiana danych poprzez zbiór pośredniczący, który magazynuje nadane wiadomości, zwalniając nadawcę od obowiązku oczekiwania na ich odebranie. Podstawowe operacje, związane z takim schematem komunikacji, obejmują utworzenie zbioru przechowującego kolejkę wiadomości, nadanie wiadomości i odebranie wiadomości. Operacja nadania powoduje umieszczenie wiadomości w kolejce i nie zatrzymuje wykonywania procesu nadającego. Operacja odebrania wiadomości powoduje pobranie wiadomości z kolejki lub zawieszenie procesu odbierającego, jeżeli kolejka jest pusta. Późniejsze nadanie wiadomości do kolejki powoduje automatyczne wznowienie zatrzymanego procesu.

Dokładna definicja kolejki wiadomości, wprowadzona w standardzie POSIX, obejmuje również sprawdzenie możliwości przepełnienia kolejki. Podstawowe definicje operacji nadania i odbioru wiadomości *w* do i z kolejki *K* można zapisać następująco:

send(K,w) :

jeżeli kolejka *K* jest pełna, to zawieś wykonywanie procesu,
w przeciwnym razie wpisz wiadomość *w* do kolejki *K* oraz jeżeli istnieją procesy oczekujące na wiadomość,
to wznów wykonywanie jednego z nich

receive(K,W) :

jeżeli kolejka *K* jest pusta, to zawieś wykonywanie procesu,
w przeciwnym razie pobierz wiadomość *w* z kolejki *K* oraz
jeżeli istnieją procesy oczekujące na miejsce w kolejce,
to wznów wykonywanie jednego z nich

Wiadomości zgromadzone w kolejce i oczekujące na odebranie mogą zawierać różne treści, o mniejszej lub większej ważności dla działania aplikacji. Miarą ważności jest priorytet, nadawany jej przez nadawcę, podczas operacji nadawania. Podczas odbierania wiadomości z kolejki najpierw

przekazywana jest zawsze wiadomość o najwyższym priorytecie.

Ta sama kolejka może być wykorzystywana przez wiele różnych procesów. Jeśli wiele procesów wykona operację *mq_send*, próbując nadać wiadomość do pełnej kolejki, to wszystkie zostaną zawieszone w stanie Z w oczekiwaniu na zwolnienie miejsca w kolejce. Podobnie, jeżeli wiele procesów wykona operację *mq_receive*, próbując odebrać wiadomości z kolejki pustej, to wszystkie zostaną zawieszone w stanie Z w oczekiwaniu na pojawienie się wiadomości. W ten sposób, z każdą kolejką wiadomości może być związana kolejka procesów zawieszonych w oczekiwaniu na zmianę stanu tej kolejki.

Obsługa kolejek [POSIX]:

- Kolejka wiadomości:
 - wiadomości mają priorytety
 - z kolejki odbierana jest najstarsza wiadomość o najwyższym priorytecie, a przy równych priorytetach – proces, który najdłużej oczekuje na wznowienie.
- Kolejka procesów zawieszonych:
 - zmienna **POSIX_PRIORITY_SCHEDULING** jest ustawiona – kolejność wznowiania zgodna jest z priorytetem zadań
 - zmienna **POSIX_PRIORITY_SCHEDULING** nie jest ustawiona – kolejność wznowiania jest nieustalona

Rozmiar kolejki oraz maksymalna długość przekazywanych wiadomości są ustalane podczas operacji utworzenia kolejki i nie mogą ulegać zmianom podczas jej istnienia.

Organizacja kolejek wiadomości jest podobna do organizacji plików i podlega takim samym mechanizmom ochrony. Każda kolejka ma swoją nazwę, zbudowaną według reguł semantyki nazw ścieżkowych, a w operacjach nadania i odbierania wiadomości jest reprezentowana przez deskryptor określony podczas operacji otwarcia. Z każdą kolejką wiadomości są związane atrybuty, określające prawa dostępu do tej kolejki, identyczne z atrybutami plików.

Podstawowe atrybuty kolejek:

- *mq_maxmsg* – maksymalna liczba komunikatów w kolejce
- *mq_msgsize* – maksymalna wielkość pojedynczego komunikatu
- *mq_flags* – flagi:
 - O_RDONLY – tylko odczyt z kolejki
 - O_WRONLY – tylko zapis do kolejki
 - O_RDWR – odczyt i zapis
 - O_CREAT – utwórz kolejkę o ile nie istnieje
 - O_NONBLCK – domyślnie flaga jest wyzerowana co powoduje, że operacje odczytu (*mq_receive*) i zapisu (*mq_send*) mogą być blokujące; gdy flaga jest ustawiona operacje te nie są blokujące i kończą się błędem
- *mq_curmsg* – aktualna liczba komunikatów w kolejce

Podstawowe operacje na kolejkach wiadomości są realizowane przez następujące funkcje:

- *mq_open* – otwarcie kolejki wiadomości

`mqd_t mq_open (char *name, int oflag, int mode, mq_attr *attr)`

gdzie:

- *name* – nazwa kolejki
- *oflag* – tryb tworzenia kolejki
- *mode* – atrybuty dostępu (analogicznie jak w przypadku plików)
- *attr* – atrybuty kolejki wiadomości

- *mq_send* – nadanie wiadomości do kolejki

`int mq_send (mqd_t mq, char *msg, size_t len, unsigned int mprio)`

gdzie:

- *mq* – identyfikator kolejki komunikatów
- **msg* – adres bufora wysłanego komunikatu
- *len* – długość wysłanego komunikatu
- *mprio* – priorytet komunikatu

- *mq_receive* – odebranie wiadomości z kolejki

`int mq_receive (mqd_t mq, char *msg, size_t len, unsigned int *mprio)`

gdzie:

- *mq* – identyfikator kolejki komunikatów
- **msg* – adres bufora na odebrany komunikat
- *len* – maksymalna długość odebranego komunikatu
- *mprio* – priorytet odebranego komunikatu

- *mq_close* – zamknięcie kolejki

`int mq_close (mqd_t mq)`

- *mq_unlink* – usunięcie kolejki

`int mq_unlink (char *name)`

- *mq_getattr* – odczytanie atrybutów kolejki

`int mq_getattr (mqd_t mq, struct mq_attr *attr)`

gdzie:

- *mq* – identyfikator kolejki komunikatów
- **attr* – adres bufora ze strukturą zawierającą atrybuty kolejki komunikatów

- *mq_setattr* – zmiana atrybutów kolejki

`int mq_setattr (mqd_t mqdes, const struct mq_attr *mqstat, struct mq_attr *omqstat);`

- *mq_notify* – żądanie zgłoszenia sygnału w chwili zapisania wiadomości do pustej kolejki

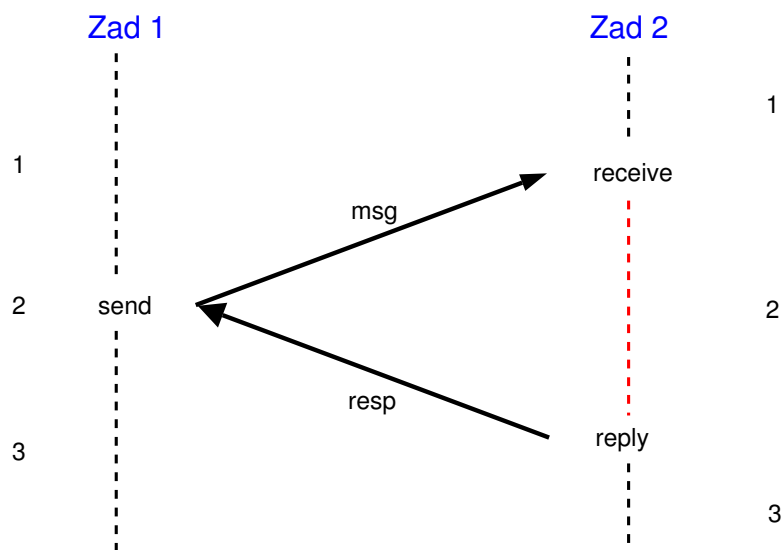
```
int mq_notify(mqd_t mqdes, const struct sigevent *notif);
```

gdzie:

- *mq* – identyfikator kolejki komunikatów
- **notif* – adres struktury specyfikującej sposób zawiadomienia

8.1.3 Spotkanie (rendez-vous)

Spotkanie jest typowym sposobem synchronizacji w języku Ada (programowanie współbieżne).



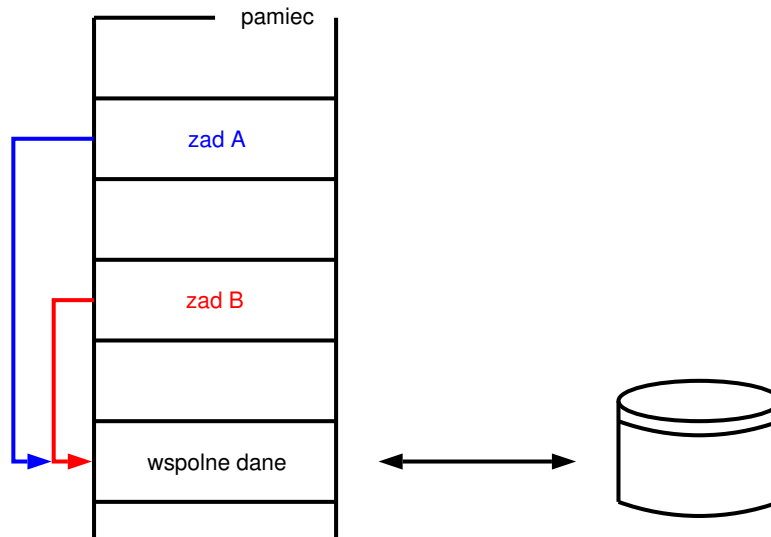
Rysunek 32: Spotkanie (rendez-vous)

Schemat operacji spotkania pokazuje rysunek 32. W chwili czasowej **1** oba zadania wykonują się współbieżnie. W chwili **2** następuje synchronizacja procesów. Zadanie wywołujące (**Zad 1**) pozostaje zawieszone, natomiast zadanie przyjmujące (**Zad 2**) wykonuje zaprogramowane operacje. Po zakończeniu operacji zadanie wywołujące jest wznawiane i obydwa zadania będą dalej współbieżnie.

9 Wykład 9 [07.05.2004]

9.1 Synchronizacja i komunikacja zadań - cd

9.1.1 Komunikacja przez obszary pamięci wspólnej (shared memory object)



Rysunek 33: Obszary pamięci wspólnej (shared memory object)

Udostępnienie segmentu danych wymaga wykonania przez proces dwóch operacji: otwarcia segmentu i alokowania go w przestrzeni pamięci procesu. Operacja otwarcia powoduje otwarcie pliku, w którym ten segment rezyduje lub utworzenie takiego pliku, jeżeli segment wcześniej nie istniał. Jednocześnie z utworzeniem segmentu można określić dla niego wskaźniki praw dostępu, podobnie jak dla innych plików. Operacja alokowania segmentu danych powoduje umieszczenie segmentu (lub jego wskazanego fragmentu) w przestrzeni pamięci procesu i określenie dla niego wymaganych znaczników ochrony. Wszystkie operacje na segmentach danych są wykonywane przez następujące funkcje systemowe:

- *shm_open* (*N*) – otwarcie segmentu danych o nazwie *N*; funkcja zwraca deskryptor pliku *FD*, używany jako parametr innych funkcji
- *ltruncate* (*FD,L*) – określenie długości nowo utworzonego segmentu danych
- *mmap* (*FD*) – alokowanie otwartego segmentu danych w przestrzeni pamięci; funkcja zwraca wskaźnik *A* na początek segmentu
- *mprotect* (*A,P*) – zmiana znaczników ochrony segmentu pamięci
- *munmap* (*A*) – usunięcie segmentu z przestrzeni pamięci procesu
- *close* (*FD*) – zamknięcie pliku segmentu danych

Operacja usunięcie procesu segmentu jest opóźniana do chwili zamknięcia segmentu we wszystkich procesach, które z tego segmentu korzystają.

Przykładowa implementacja komunikacji przez obszary pamięci wspólnej:

```
fd = shm_open ("nazwa", O_RDWR | O_CREAT, 0777);
    // "nazwa" – obiekt pamięciowy
    // 0777 – prawa (takie jak w chmod)

size = ltrunc (fd, size, SEEK_SET);

addr = mmap (start, lenght, PROT_READ | PROT_WRITE, MAP_SHARED, fd, offset);
    /* Adres start jest jednak tylko propozycja i zazwyczaj jest
       przekazywany jako 0. Rzeczywiste miejsce zmapowania obiektu
       jest zwracane przez mmap i nigdy nie jest zerem.*/
    // lenght – wielkość pliku lub innego obiektu (w bajtach)
    // PROT_READ, PROT_WRITE – określają sposoby ochrony pamięci
    /* MAP_SHARED – polecenie współdzielenia mapowania ze wszystkimi
       innymi procesami, które mapują ten obiekt. */
    // fd – dekryptor pliku

sem = (sem_t *) addr;    // ustawienie na początku obszaru

sem_init (sem, 1, value);
    // 1 – semafor dzielony przez wiele procesów
    // value – ustawienie początkowe semafora

...

sem_wait (sem);

...

sem_post (sem);

...

sem_destroy (sem);

munmap (addr, lenght);
close (fd);
shm_unlink ("nazwa");
```

Inny proces:

- możliwość pierwsza (*fork*):

proces dziedziczy:

- obszar pamięci
- semafor

...

```
sem_post (sem);
// może odwieść proces
```

- możliwość druga:

```

fd = shm_open ("nazwa", ...);
addr = mmap (fd, ...);
sem = (sem_t *) addr;
...

sem_post (sem);

```

9.2 Rozszerzenia czasu rzeczywistego

9.2.1 Problemy

Uzależnienie wykonywania zadań od czasu wiąże się z następującymi problemami:

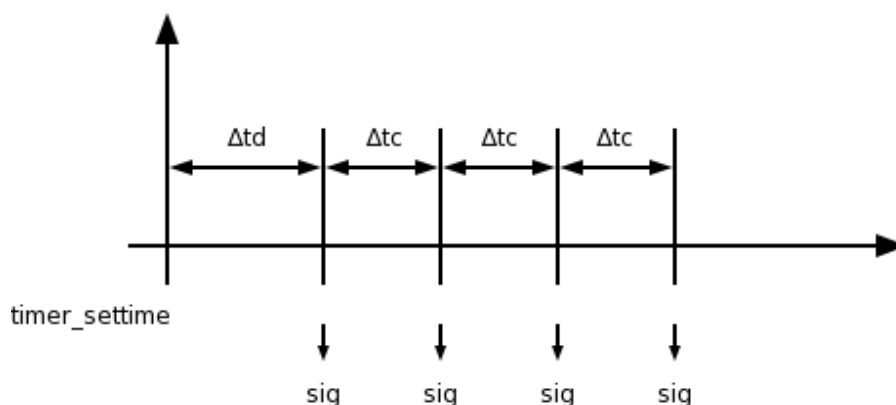
- opóźnienia
 - *sleep* (Δt) – gdzie Δt to struktura *long tv_sec* określająca liczbę sekund
 - *nanosleep* (Δt) – gdzie Δt to struktura *long tv_nsec* określająca liczbę nanosekund
 - *delay* (Δt) – czas w mikrosekundach
- cykliczność
Funkcje nie mogą czekać w nieskończoność (dostęp do zasobów, synchronizacja)
- przeterminowanie

```

while (1)
{
    sleep ($\Delta t$);
    ...
    ...
}

```

9.2.2 Programowe liczniki czasu



Rysunek 34: Programowalny licznik czasu (timer)

Operacje dostępne na programowych licznikach czasu (*timer*)

- *timer_create*(*tim*, *sig*) – utworzenie licznika o identyfikatorze *tim* i numerowi licznika *sig*, z którym chcemy połączyć licznik
- *timer_settime*(*tim*, Δt_d , Δt_c) – uruchomienie licznika z zadany czas zwłoki Δt_d o długości cyklu Δt_c lub zatrzymanie działającego licznika

- *timer_gettime ()* – odczytanie zawartości licznika
- *timer_delete ()* – usunięcie licznika

Przypadki szczególne:

- $\Delta t_d = 0$
brak początkowego opóźnienia
- $\Delta t_c = 0$
tylko opóźnienie
brak akcji

9.3 System QNX

QNX jest wielozadaniowym systemem operacyjnym czasu rzeczywistego, opracowanym przez firmę [Quantum Software](#) dla lokalnych sieci komputerów zgodnych programowo z komputerami rodziny IBM PC.

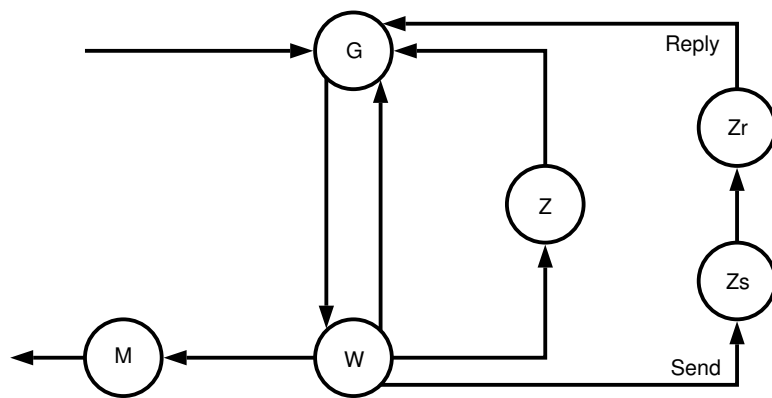
System składa się z jądra, pełniącego rolę egzekutora wielozadaniowego i zespołu zadań systemowych, współpracujących ze sobą i z zadaniami użytkowymi zgodnie z modelem wymiany usług (rysunek 15).

Zakres funkcji jądra obejmuje tylko:

- szeregowanie zadań,
- implementację komunikacji między zadaniami
- przyjmowanie zgłoszeń przerwania

Pozostałe funkcje wypełniają zadania systemowe:

1. Administrator zadań (*Process Manager*) – odpowiedzialny za tworzenie innych zadań oraz inicjowanie funkcji odmierzenia czasu.
 2. Administrator sieci (*Network Manager*) – obsługujący łącza sieciowe i komunikację węzłów sieci.
 3. Administrator zbiorów (*Filesystem Manager*) – organizujący system zbiorów w pamięciach dyskowych i taśmowych.
 4. Administrator urządzeń (*Device Manager*) – formatujący strumienie danych przekazywane między zadaniami i urządzeniami zewnętrznymi.
 5. Zadania sterujące urządzeniami (*Device Drivers*) – inicjujące i kończące ich działanie i obsługujące zgłaszane przerwania.
- poziomy priorytetu numerowany od 0 (min) do 31 (max)
 - zadanie o wyższym prioryecie zawsze wywłaszcza zadanie o prioryecie niższym
 - priorytety mogą być zmieniane dynamicznie
 - szeregowanie zadań na tym samym poziomie priorytetów:



Rysunek 35: Graf stanów zadań w systemie QNX

- algorytm FIFO
- algorytm karuzelowy (*time sharing*)
- algorytm adaptacyjny - dyskryminacja zadań o długich priorytetach

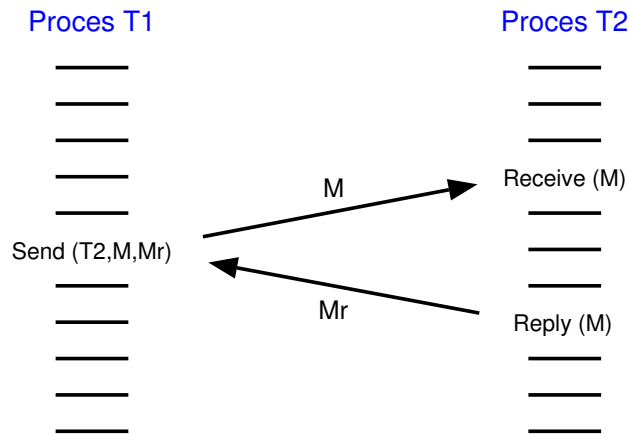
10 Wykład 10 [21.05.2004]

10.1 System QNX - cd

10.1.1 Komunikacja i synchronizacja w systemie QNX

W systemie QNX dostępne są następujące mechanizmy do komunikacji i synchronizacji zadań:

1. Spotkanie (*rendez-vous*)



Rysunek 36: Implementacja spotkania w systemie QNX

Podstawowe etapy wymiany wiadomości są realizowane przez trzy funkcje systemowe:

- *Send (T2,M,Mr)* – wysłanie wiadomości *M* do procesu *T2* i zawieszenie procesu nadającego do czasu otrzymania odpowiedzi *Mr*; proces nadający wchodzi w stan *Zs*;
- *Receive (T1,M)* – odebranie wiadomości *M* od procesu *T1*; proces, który nadał wiadomość, jest przenoszony do stanu *Zr*;
- *Reply (T1,Mr)* – przekazanie odpowiedzi *Mr* do procesu *T1* i wznowienie tego procesu

Dodatkowo istnieje funkcja, która przy wywołaniu nie zawiesza zadania:

- *Creceive (T,M)* – jeśli przy próbie odebrania wiadomości okazuje się, że nie została ona nadana, funkcja zwraca kod błędu.

Jeśli przesyłane wiadomości są za długie można wysyłać je porcjami, a służą do tego następujące funkcje systemowe:

- *Readmsg ()* – pobranie fragmentu oczekującej wiadomości; wykonanie tej funkcji nie zmienia stanu żadnego procesu;
- *Writemsg ()* – dopisanie fragmentu odpowiedzi; wykonanie tej funkcji nie zmienia stanu żadnego procesu.

W systemie QNX problem **inwersji priorytetów** rozwiązany jest w ten sposób, że operacja *send-receive* podnosi priorytet serwera do priorytetu klienta.

2. Depozyty (*proxy*)

Depozyty można uważać za rozszerzenie tradycyjnego mechanizmu semaforów, służących do przekazywania impulsów synchronicznych między współpracującymi procesami. Rozszerzenie polega na stworzeniu impulsów z pewną stałą i niezmienną wartością, przekazywaną do odbiorcy od normalnej akcji synchronicznej. Podstawowe operacje związane z wykorzystaniem tego mechanizmu realizuje funkcja *Receive* i trzy dalsze funkcje systemowe:

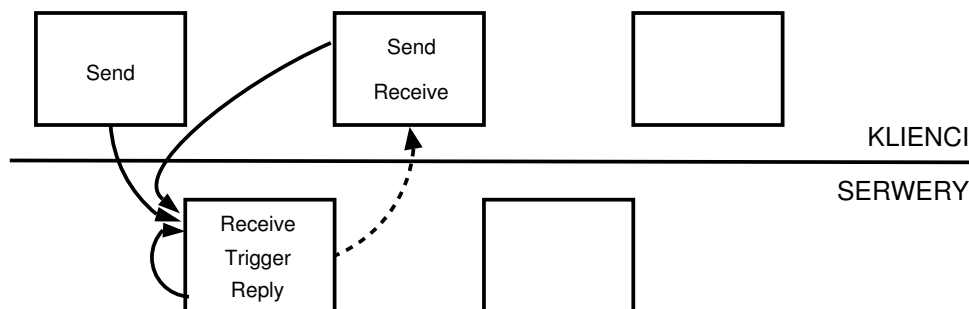
- *qnx_proxy_attach* (T, M) – utworzenie procesu depozytowego, przechowującego wiadomości M i stanowiącego własność procesu T
- *qnx_proxy_detach* (P) – usunięcie procesu depozytowego P
- *Trigger* (P) – pobudzenie procesu depozytowego P i wysłanie zdeponowanej wiadomości do procesu właściciela.

Wywołanie funkcji *Trigger*:

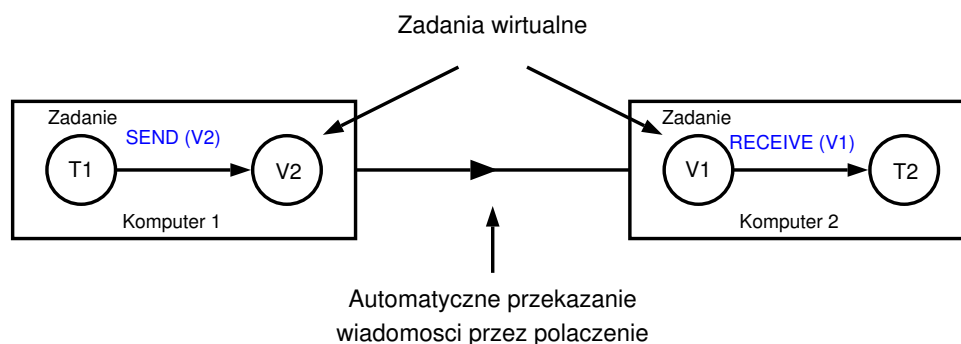
- przez zadanie
- przez funkcję obsługi przerwania

Depozyty w aplikacjach sieciowych:

- Utworzenie zadania depozytowego w węźle właściciela
- Utworzenie reprezentanta w innym węźle
- Funkcje utworzenia i usunięcia lokalnego reprezentanta procesu depozytowego są realizowane przez dwie funkcje systemowe:
 - *qnx_proxy_rem_attach* (N, P) – utworzenie reprezentanta procesu depozytowego P w węźle N ;
 - *qnx_proxy_rem_detach* (N, V) – usunięcie reprezentanta V odległego procesu depozytowego w węźle N .

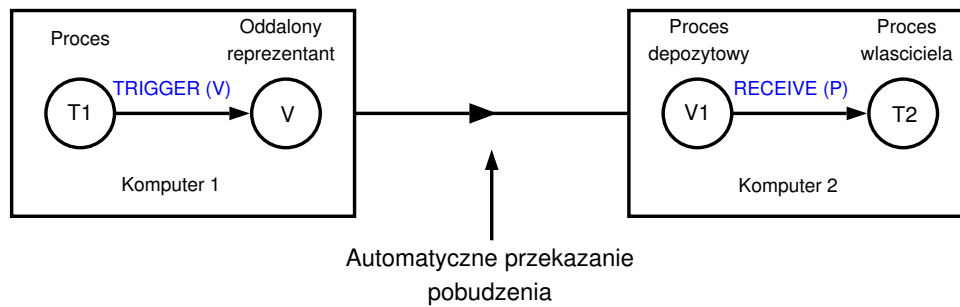


Rysunek 37: Architektura klient-serwer i spotkanie



Rysunek 38: Przekazywanie wiadomości przez połączenie

3. **Sygnały** (*signal*) [POSIX] – obsługa zgodna ze standardem POSIX
4. **Kolejki wiadomości** (*message queue*) [POSIX] – obsługa zgodna ze standardem POSIX
5. **Potoki** (*pipe*) [POSIX] – obsługa zgodna ze standardem POSIX
6. **Semaforey** (*semaphore*) [POSIX] – obsługa zgodna ze standardem POSIX



Rysunek 39: Przekazywanie depozytów między węzłami sieci

10.1.2 Komunikacja ze sprzętem

Komunikacja z rejestrami urządzeń:

- *inp (port)* – odczytanie jednego bajtu danych z rejestru o numerze *port*
- *inpw (port)* – odczytanie słowa z rejestru o numerze *port*
- *inpd (port)* – odczytanie drugiego słowa z rejestru o numerze *port*
- *outp (port,data)* – zapisanie bajtu danych *data* do rejestru o numerze *port*
- *outpw (port,data)* – zapisanie słowa *data* do rejestru o numerze *port*
- *outpd (port,data)* – zapisanie drugiego słowa *data* do rejestru *port*

Wszystkie wymienione funkcje zawierają maszynowe instrukcje wejścia-wyjścia, które mogą być wykonane tylko przez programy uprzywilejowane, zapisane w plikach stanowiących własność administratora systemu. Nadanie statusu uprzywilejowanego może nastąpić podczas konsolidacji programu (parametr *-T1*).

10.1.3 Przerwania

System QNX umożliwia procesom definiowanie własnych funkcji obsługi przerwania zewnętrznych, działających podobnie jak funkcje obsługi sygnałów. Dodatkowo, zakończenie funkcji obsługi może powodować wzbudzenie procesu depozytowego i przekazanie wiadomości o wystąpieniu przerwania do procesu użytkowego.

Podstawowe operacje , związane z instalowaniem funkcji obsługi i maskowaniem przerwania, są realizowane przez następujące funkcje systemowe:

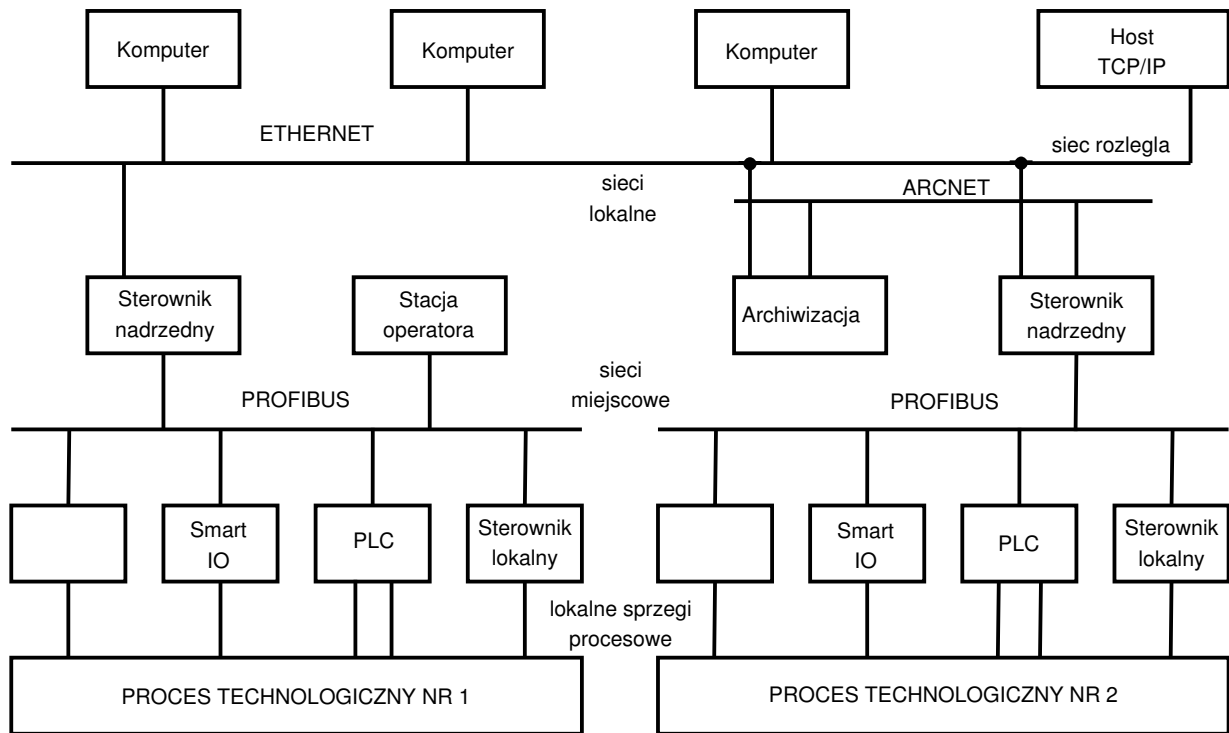
- *qnx_hint_attach (I,H)* – zainstalowane funkcji *H* obsługujące przerwanie numer *I*
- *qnx_hint_detach (I)* – usunięcie funkcji obsługi przerwania *I*
- *qnx_hint_mask (I,M)* – określenie maski *M* przerwania *I*

10.2 Przemysłowe sieci miejscowe

10.2.1 Wprowadzenie

Profibus jest jedną z najbardziej popularnych sieci przemysłowych na całym świecie (została zaprojektowana przez SIEMENS).

Warstwy funkcji systemu:



Rysunek 40: Struktura funkcjonalna systemu sterującego

- sterowania (bezpośrednio połączona z procesem)
- nadrzędna (sterowania operacyjnego)
- zarządzania (planowania, harmonogramowania)

Struktura komunikacyjna systemu sterującego:

- sprzęg procesowy, łączący czujniki i elementy wykonawcze z urządzeniami warstwy sterowania:
 - sygnały analogowe:
 - * sygnał prądowy 4 – 20mA – najbardziej popularny zakres
 - * sygnał prądowy 0 – 20mA – prostszy w realizacji, ale gorszy bo czasem płynie prąd
 - * sygnał napięciowy 0 – 10V – rzadko, bo sygnał napięciowy łatwo zakłócić
 - * rezystancyjne (pomiar temperatury) – zmiana rezystancji wraz ze zmianą temperatury
 - sygnały dwustanowe
dominuje 0 / 24V
- sieci miejscowe, łączące sterowniki warstwy nadrzędnej i inteligentne czujniki
- sieci lokalne i rozległe, łączące komputery warstwy nadrzędnej i warstwy zarządzania

Warunki pracy przez sieć:

- Sieci miejscowe:
 - ostre ograniczenia czasu rzeczywistego (determinizm przekazywania komunikatów)

- efektywne przenoszenie krótkich wiadomości
 - możliwość dołączenia prostych urządzeń
 - praca na poziomie hali fabrycznej
- Sieci lokalne
 - łagodne ograniczenia czasowe

11 Wykład 11 [28.05.2004]

11.1 Przemysłowe sieci miejscowe - cd

11.1.1 Warstwa linowa (łącza danych)

- **Podwarstwa dostępu do łącza**

Warunkiem poprawnego przekazania komuniktu od nadawcy do odbiorcy jest uniknięcie kolizji, która powstałaby w razie jednoczesnego nadawania przez co najmniej dwa węzły sieci. Zadaniem podwarstwy dostępu do kabla jest uniknięcie lub rozwiązanie kolizji i umożliwienie węzłowi uzyskania wyłącznego prawa nadawania. Informacje z wykładów i skryptów do wykładu [1, 2] wzbogacone zostały o informacje z książki prof. nzw. dr hab. Krzysztofa Sachy „Sieci miejscowe PROFIBUS” [3].

1. **Algorytm losowy CSMA/CD (*carrier sense multiple access with collision detection*)** dopuszcza możliwość kolizji i przewiduje procedurę jej rozwiązania. Proces nadania komunikatu dzieli się na dwie fazy. W pierwszej fazie węzeł obserwuje stan kabla i powstrzymuje transmisję aż do stwierdzenia, że kabel jest wolny. Nie zabezpiecza to przed kolizją z innym węzłem, który również mógł obserwować stan kabla i rozpocząć nadanie w tym samym czasie. Dlatego w drugiej fazie (po rozpoczęciu nadawania) węzeł nadal obserwuje stan kabla i wykrywa ewentualną kolizję. W razie kolizji przerywa nadawanie, odczekuje pewnien losowo wybrany czas, po czym ponawia próbę nadawania. Dla obniżenia prawdopodobieństwa ponownej kolizji każdy węzeł losuje czas oczekiwania z przedziału $0... \Delta t$. W razie kolizji postępowanie węzłów jest podobne, ale wartość czasu oczekiwania jest wybierana losowo z dwukrotnie dłuższego przedziału $0... 2\Delta t$, itd.

Warunkiem działania algorytmu jest możliwość wykrywania kolizji przez węzły nadające. Narzuca to ograniczenie na minimalną długość przekazywanych komunikatów – czas nadawania musi być zawsze dłuższy od podwójnego czasu propagacji sygnału przez największą dopuszczalną odległość w sieci. Krótsze wiadomości mogą być przekazane dopiero po dopełnieniu komunikatu nieistotnymi bitami.

Zalety algorytmu CSMA/CD:

- prostota realizacji
- bardzo wysoka średnia przepustowość sieci

Wady algorytmu:

- niedeterministyczny czas nadania wiadomości
- narzucona minimalna długość komunikatu
- nieefektywne przekazywanie krótkich komunikatów

Zastosowanie:

- powszechnie stosowany w sieciach biurowych (np. Ethernet)
- nie nadaje się dla sieci przemysłowych

2. **Algorytm z przekazywaniem znacznika (*token passing*)** dąży do uniknięcia kolizji w normalnym stanie sieci. W każdej chwili prawo nadawania – utożsamiane z umownym znacznikiem – ma tylko jeden węzeł. Węzeł ten może dowolnie wykorzystywać magistralę przez pewnien czas Δt , po upływie którego specjalnym komunikatem przekazuje znacznik do następnego węzła. Kolejność przekazywania znacznika następuje przez wysłanie do następcy specjalnego komunikatu sterującego, o określonej zawartości.

Prosty schemat przekazywania znacznika ulega zakłóceniu w stanach awaryjnych, w których znacznik ulegnie nieoczekiwanemu rozmnożeniu, np. na skutek odebrania znacznika przez dwa węzły w wyniku przekłamania adresu, lub zostanie zgubiony, np. na skutek awarii węzła, który go w danej chwili posiadał. W obu przypadkach konieczne jest wykonanie akcji korekcyjnej, która znacznie podnosi złożoność algorytmu.

Zalety algorytmu z przekazywaniem znacznika:

- deterministyczny czas przekazywania komunikatu, równy co najwyżej czasowi obiegu znacznika w sieci
- brak ograniczeń na minimalną długość komunikatu
- bardzo małe wahania przepustowości sieci w zależności od obciążenia

Wady algorytmu:

- narzut czasowy wynikający z konieczności częstego przekazywania komunikatów zawierających znacznik
- skomplikowany algorytm odtwarzania zgubionego znacznika
- narzucona maksymalna długość komunikatu

Zastosowanie:

- rzadko używany w sieciach biurowych
- jest dominującym sposobem organizacji pracy sieci przemysłowych

3. **Algorytm podziału czasu TDMA (*time division multiple access*)** nie dopuszcza ani możliwości wystąpienia kolizji ani wahan natężenia ruchu w sieci. Cała sieć pracuje w stałym cyklu czasowym o dokładnie określonej długości. W ramach cyklu każdy węzeł ma na stałe przydzielony odcinek czasu, w którym musi nadać swój komunikat. Brak komunikatu świadczy o awarii węzła.

Zalety algorytmu TDMA:

- prostota
- deterministyczny czas nadania komunikatu
- wysoka efektywność

Wady algorytmu:

- algorytm może działać tylko w sieciach o stałej liczbie węzłów i stałym powtarzalnym rytmie pracy

Zastosowanie:

- Sieci o ostrych ograniczeniach czasowych, o stałej liczbie węzłów (np. sieci w nowoczesnych samochodach)

4. **Priorytetowe algorytmy CSMA (*carrier sense multiple access*)** zostały opracowane z myślą o sieciach stosunkowo wolnych, w których czas nadawania bitu jest dłuższy od podwójnego czasu propagacji sygnału przez sieć. Rozstrzygnięcie kolizji w takiej sieci może nastąpić podczas transmisji komunikatu. Pierwszym elementem komunikatu jest adres nadawcy. Jeżeli w tym samym czasie rozpoczną nadawanie dwa węzły, to adresy obu węzłów „nałożą” się na siebie. W przypadku niezgodności bitów (0 i 1) w sieci ustali się wartość 0. Dzięki temu węzeł, który nadał 1 (a więc węzeł o wyższym adresie), może wykryć kolizję i przerwać nadawanie. Węzeł o niższym adresie może nadać resztę komunikatu bez przeszkód.

Zastosowanie:

- sieci stosunkowo wolne, w których czas nadawania bitu jest dłuższy od podwójnego czasu propagacji sygnału przez sieć
5. **Odpytywanie (*polling*)** jest stosowane w systemach o scentralizowanym przetwarzaniu. Węzeł nadrzędny (*master*) regularnie wysyła do wszystkich węzłów podporządkowanych (*slave*) komunikaty z zapytaniem o dane. Po otrzymaniu zapytania węzeł podporządkowany wysyła odpowiedź zawierającą dane lub informację o ich braku. Komunikacja odbywa się tylko między węzłem nadrzędnym a węzłami podporządkowanymi i zawsze ma charakter transakcji: zapytanie – odpowiedź.

Zalety algorytmu odpytywania:

- prostota sprzęgu sieciowego po stronie węzłów podporządkowanych

Wady algorytmu:

- brak możliwości komunikowania się węzłów podporządkowanych
- rozpad sieci po awarii węzła nadrzędnego – co nie ma jednak dużego znaczenia, gdyż węzeł ten jest i tak niezbędny do pracy systemu

Zastosowanie:

- systemy o scentralizowanym przetwarzaniu, np. w sieci łączącej komputer lub sterownik PLC z czujnikami
 - sieci o topologii magistralowej i gwiazdowej
6. **Pierścień znacznikowy (*token ring*)** jest stosowany w sieci pierścieniowej, po której krąży umowny znacznik, z którym związane jest prawo nadawania. Po otrzymaniu znacznika węzeł może zatrzymać go i rozpocząć nadawanie. Po zakończeniu komunikatu przekazuje znacznik do kolejnego węzła. Nadany komunikat przechodzi kolejno przez wszystkie węzły pierścienia, m.in. przez węzeł odbiorcy, po czym wraca do węzła nadającego, który nie przesyła go dalej.

Wady:

- realizacja pierścienia znacznikowego wymaga skomplikowanego układu nadawania i odbioru komunikatów oraz złożonego algorytmu odzyskiwania znacznika i konfigurowania sieci.

• Podwarstwa łącza logicznego

Definicja podwarstwy łącza logicznego nie zależy od rodzaju kabla i algorytmu dostępu. Jedynym wymaganiem stawianym niższym warstwom sieci jest możliwość wysyłania komunikatu do określonego węzła-odbiorcy, przy czym nie wymaga się niezawodnego działania sieci. Funkcje podwarstwy łącza logicznego obejmują zapewnienie bezbłędnego przekazywania wiadomości oraz organizację systemu adresowania komunikatów przeznaczonych dla różnych programów wykonywanych w tym samym węźle sieci. Standard dopuszcza dwa rodzaje protokołów tej podwarstwy: zawodną komunikację bezpołączeniową i niezawodną komunikację połączeniową.

- **Protokół bezpołączeniowy (*datagram*)** określa najprostszy rodzaj komunikacji. Przekazywana wiadomość jest opatrywana odpowiednim nagłówkiem i zakończeniem i wysyłana w postaci komunikatu do odbiorcy. Nadawca wiadomości nie podejmuje żadnych działań dla sprawdzenia, czy przesyłka dotarła do odbiorcy i czy nie została po drodze przekłamana. Błędy transmisji lub brak gotowości odbiorcy do odebrania komunikatu prowadzą nieuchronnie do utraty wiadomości.

Zalety:

- * prostota realizacji
- * duża szybkość komunikacji
- * łatwość rozgłaszania (*broadcast*)

Wady:

- * zawodność przekazu – brakujące funkcje kontrolne muszą być realizowane przez wyższe warstwy, najczęściej warstwę transportową (jeżeli istnieje) lub aplikacyjną.
- **Protokół połączeniowy (*connection-orientred*)** zapewnia niezawodny przekaz komunikatów między dwoma dowolnymi węzłami sieci. Komunikacja ma postać sesji, w trakcie których węzły przekazują komunikaty zawierające dane oraz informacje sterujące. Sesja komunikacji węzłów przebiega w trzech fazach. W pierwszej fazie następuje nawiązanie połączenia, w wyniku czego współpracujące węzły mogą przygotować się do wymiany danych. W ostatniej fazie sesji połączenie jest zamykane. W fazie środkowej węzły mogą wymieniać dowolne komunikaty danych.

Kolejne komunikaty są numerowane liczbami naturalnymi modulo 128. Odbiornik kontroluje poprawność odbioru i przesyła potwierdzenia, nie rzadziej niż co n odebranych komunikatów. Liczba $N < 128$ jest uzgodnionym parametrem połączenia. Nadajnik może wysłać bez potwierdzenia co najwyżej n komunikatów, po czym musi przerwać nadawanie i oczekiwać na potwierdzenia. Potwierdzenie zawiera numer komunikatu i oznacza potwierdzenie wszystkich wcześniejszych komunikatów, aż do tego numeru włącznie. Otrzymanie potwierdzenia zezwala nadawcy na wznowienie nadawania. W razie błędu odbiornik przesyła, zamiast potwierdzenia, żądanie retransmisji komunikatów przekłamanych. W razie długotrwałego braku odpowiedzi nadajnik powtarza ostatni komunikat. Brak odpowiedzi po kilku próbach oznacza awarię odbiorcy.

Adresowanie nadawców i odbiorców komunikatów za pomocą węzłów sieci jest zbyt mało selektywne. Każdy węzeł może wykonywać wiele różnych programów i rzeczywistymi partnerami komunikacji są nie węzły lecz wykonywane programy. Z tego powodu konieczny jest dodatkowy mechanizm adresowy, umożliwiający logiczne podzielenie sieci na odrębne kanały łączące współpracujące programy. Taki mechanizm tworzą **porty (*SAP – Service Access Point*)**, czyli indywidualnie adresowane „skrzynki pocztowe”, poprzez które programy mogą nadawać i odbierać komunikaty poprzez różne porty. Numery portów nadawcy i odbiorcy są zapisane w treści przekazywanych komunikatów.

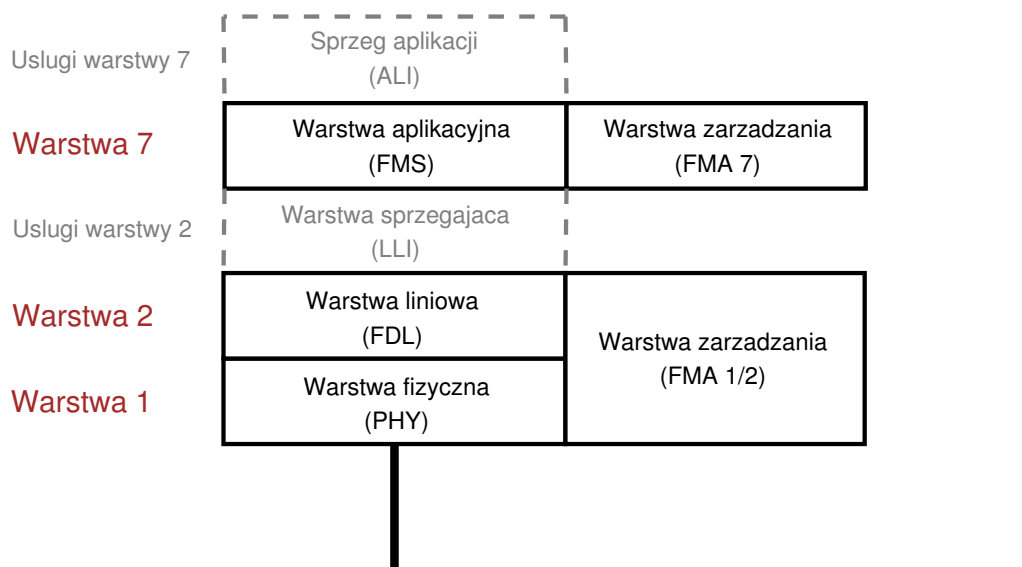
Zalecenia do sieci miejscowych:

- Protokół znacznikowy
 - deterministyczny czas przekazu
 - efektywny przekaz krótkich wiadomości
 - zdecentralizowane sterowanie sieci
- Odpytywanie węzłów podporządkowanych
 - prosty sprzęg węzłów podporządkowanych
- Protokół komunikacyjny połączeniowy
 - duża pewność przekazu

11.2 Sieć PROFIBUS

PROFIBUS (*Process Field Bus*) jest popularną siecią miejscową, przeznaczoną do wykorzystania w rozproszonych systemach sterowania i nadzoru.

- przeznaczenie: systemy sterowania i nadzoru
- typowe węzły sieci:
 - inteligentne czujniki i elementy wykonawcze
 - regulatory
 - sterowniki programowalne i numeryczne
 - lokalne stacje operatorskie



Rysunek 41: Warstwy modelu sieci PROFIBUS

Protokoły komunikacyjne sieci PROFIBUS definiuje norma DIN 19245. Definicja obejmuje:

- warstwę fizyczną – sprzęg RS 485
- warstwę liniową (łącza danych)
Warstwą liniową implementuje dwa rodzaje usług: niezawodne przekazywanie komunikatu z odpowiedzią lub potwierdzeniem odbioru oraz przekazanie komunikatu bez potwierdzenia, w tym rozgłaszanie (*broadcast*).
- warstwę aplikacyjną
Warstwa aplikacyjna jest opcjonalna: użytkownicy, tzn. wykonywane programy, mogą korzystać z sieci wywołując albo usługi warstwy aplikacyjnej, albo warstwy liniowej

11.2.1 Warstwa fizyczna: sprzęg RS-485

Definicja warstwy fizycznej opiera się na specyfikacji sprzęgu RS-485. Podstawową strukturą sieci jest liniowy segment kabla – skrętki, zakończony na obydwu krańcach terminatorami. Maksymalna długość segmentu zależy od szybkości transmisji i jakości kabla. Zalecane wartości tych parametrów podaje tabela poniżej. Norma dopuszcza jednak zwiększenie długości segmentu, pod warunkiem, że jego tłumienie nie przekroczy 6dB. W praktyce podaną w tabeli długość segmentu

można podwoić stosując skrętkę o przekroju przewodnika nie mniejszym niż $0.5mm^2$.

Maksymalna liczba węzłów sieci, które mogą być dołączone, do tego samego segmentu kabla wynika z elektrycznych specyfikacji sprzęgu RS-485, przytoczonych w tabeli poniżej. Liczba ta nie może przekraczać 32 standardowych węzłów, wnoszących maksymalnie obciążenie określone przez dopuszczalne wartości rezystancji nadajnika i odbiornika. Przy napięciu $12V$ jeden nadajnik może obciążyć linię prądem co najwyżej $0.1mA$, a odbiornik prądem co najwyżej $1mA$. Jeżeli zastosowane zostaną układy nadajników i odbiorników wnoszące mniejsze obciążenie, to możliwe jest dołączenie do segmentu sieci większej liczby węzłów. Obecnie dostępne układy nadajników i odbiorników wnoszą obciążenie czterokrotnie mniejsze od maksymalnego, co umożliwia dołączenie do segmentu sieci nawet 128 węzłów. Układy nadajników i odbiorników mogą, lecz nie muszą gwarantować galwaniczną izolację węzłów i kabla.

Rodzaj kabla	ekranowana skrętka o impedancji falowej $100...130\Omega$, pojemność między przewodami nie przekraczająca $60pF/m$ i przekroju przewodnika co najmniej $0.22mm^2$
Topologia	magistralowa, długość doprowadzeń węzłów $\leq 0.3m$
Szybkość transmisji	9.6; 19.2; 93.75; 187.5; 500 lub 1500 $Kbit/s$
Długość segmentu	zależy od szybkości transmisji i wynosi: <ul style="list-style-type: none">• $\leq 1200m$ dla szybkości $\leq 93.75 Kbit/s$• $\leq 600m$ dla szybkości $\leq 187.5 Kbit/s$• $\leq 200m$ dla szybkości $\leq 1500 Kbit/s$
Liczba węzłów	co najwyżej 32 węzły (lub powtarzacze) w obrębie segmentu

Sieć można budować z wielu oddzielnych segmentów, łącząc je ze sobą za pomocą powtarzaczy. Liczbę połączonych segmentów ogranicza warunek, że pomiędzy dwoma dowolnymi węzłami nie mogą znajdować się więcej niż trzy powtarzacze. Maksymalnie sieć może się więc składać z czterech segmentów połączonych w łańcuch albo z większej liczby segmentów połączonych gwiazdźdźście. Niezależnie od liczby segmentów sieć może zawierać co najwyżej 127 węzłów. Ograniczenie to wynika z liczby bitów przeznaczonych na adres węzła sieci w polu adresowym komunikatu. Norma DIN 19245 przewiduje co prawda możliwość rozszerzenia systemu adresowania i wprowadzenie adresów segmentowych, jednak komplikuje to budowę sprzęgu sieciowego i w praktyce jest wykorzystane bardzo rzadko.

11.2.2 Warstwa liniowa (łącza danych)

Węzły dzielą się na:

- nadrzędne (*master*) – które mogą nadawać komunikaty z własnej inicjatywy
- podrzędne (*slave*) – które mogą tylko odpowiadać na zapytania węzłów nadrzędnych.

W każdej chwili sieć jest nadzorowana przez jeden z węzłów nadrzędnych, który może w tym czasie wymieniać dane z innymi węzłami. Prawo nadzorowania sieci, utożsamiane z umownym znacznikiem, jest przekazywane cyklicznie między wszystkimi węzłami nadrzędnymi. Każdy węzeł może

przetrzymanywać znacznik tylko przez ograniczony odcinek czasu.

Wymiana danych w sieci jest zorganizowana w formie transakcji rozpoczynających się komunikatem akcji, wysłanym przez węzeł nadrzędny posiadający znacznik, a kończących się komunikatem odpowiedzi, wysłanym przez węzeł, który odebrał komunikat akcji. Rytm wykonywania transakcji nie jest związany z rytmem pracy przetwarzania danych przez węzeł odpowiadający. Odpowiedź na komunikat akcji musi być natychmiastowa. Jeśli komunikat odpowiedzi ma zawierać dane, to są one pobierane i wysyłane z bufora komunikacyjnego, wypełnionego wcześniej przez program użytkownika. Dane zawarte w komunikacie akcji nie mogą być więc traktowane jako odpowiedź na zapytanie zawarte w komunikacie akcji – taka odpowiedź może pojawić się dopiero w następnej transakcji.

Protokół dostępu do kabla

- wszystkie węzły sieci są identyfikowane adresami z zakresu 0...126
- adres 127 jest zarezerwowany jako adres rozgłaszania
- umowny znacznik jest przekazywany pomiędzy węzłami nadrzędnymi w kolejności rosnących adresów (tylko węzeł o najwyższym adresie przekazuje znacznik węzłowi o adresie najniższym)

12 Wykład 12 [04.06.2004]

12.1 Sieć PROFIBUS - cd

12.1.1 Warstwa liniowa (łącza danych) - cd

Protokół dostępu do kabla

Wszystkie węzły sieci są identyfikowane numerycznymi adresami z zakresu 0...126. Adres 127 jest zarezerwowany jako adres rozgłaszania. Węzły nadrzędne przekazują sobie znacznik w kolejności rosnących adresów.

Pierścień obiegu znacznika. Podstawą realizacji algorytmu przekazywania znacznika w każdym węźle jest zespół trzech parametrów:

- TS – adres własny węzła (ten parametr, jako jedyny jest określany w czasie konfiguracji sieci, resztę węzeł sam sobie określa)
- PS – adres poprzednika, tzn. węzła, od którego jest otrzymywany znacznik
- NS – adres następnika, tzn. węzła, do którego jest przekazywany znacznik

Przekazanie znacznika polega na nadaniu specjalnego komunikatu sterującego, zawierającego adresy TS i NS. Węzeł odbierający znacznik porównuje adres nadawcy z adresem poprzednika PS i jeśli adresy te są równe, to przyjmuje znacznik i rozpoczyna wykonywanie transakcji. Jeśli znacznik został nadany przez innego nadawcę, to za pierwszym razem węzeł odbierający nie przyjmuje go. Jeżeli nadawca ponowi transmisję i węzeł otrzyma znacznik po raz drugi, to przyjmuje go, uznając, że nastąpiła rekonfiguracja sieci.

Węzeł, który przekazał znacznik, oczekuje teraz przez pewien czas na pojawienie się transmisji w sieci. Jeśli transmisja nastąpi, to uznaje, że znacznik został przekazany i inny węzeł przejął nadzór nad pracą sieci. W przeciwnym wypadku powtarza próbę i ponownie oczekuje na rozpoczęcie transmisji. Próba przekazania znacznika jest powtarzana 3-krotnie. Jeżeli w tym czasie znacznik nie zostanie odebrany, to węzeł nadający próbuje przekazać go do węzła następnego po NS. Procedura poszukiwania następnika jest kontynuowana, aż do pomyślnego przekazania znacznika lub do wyczerpania węzłów sieci. W pierwszym przypadku węzeł modyfikuje adres swojego następnika NS. W drugim – zatrzymuje znacznik i zachowuje się, jakby otrzymał go od poprzednika.

Wykonanie transmisji. Dotrzymanie zadanego czasu oczekiwania węzła nadrzędnego na prawo nadawania wymaga stałej kontroli długości cyklu obiegu znacznika i ograniczenia przedziału czasu, przez który węzeł może zatrzymać znacznik w swoim posiadaniu. W tym celu każdy węzeł nadrzędny mierzy czas, jaki upłynął od chwili ostatniego otrzymania znacznika, i dostosowuje działanie do obciążenia sieci. Procesem tym sterują wartości dwu parametrów czasowych:

T_{TR} (*target rotation*) – stała określająca zadaną długość cyklu obiegu znacznika w sieci (wartość ustalana podczas konfiguracji sieci – taka sama dla wszystkich węzłów)

T_{RR} (*real rotation*) – zmierzona długość rzeczywistego obiegu znacznika w sieci (wartość ta jest mierzona przez węzeł w każdym cyklu obiegu znacznika)

Różnica tych dwóch parametrów:

$$T_{TH} = T_{TR} - T_{RR}$$

określa przedział czasu, który węzeł może wykorzystać na realizację transakcji bez naruszenia zadanej długości cyklu obiegu znacznika.

Po otrzymaniu znacznika węzeł oblicza czas T_{TH} . Niezależnie od wyniku ma prawo do nadania jednej transakcji priorytetowej. Dalsze transakcje może wykonywać tylko w obrębie czasu T_{TH} . Każda transakcja rozpoczyna się komunikatem akcji, po wysłaniu którego węzeł oczekuje pewnego czasu na pojawienie się odpowiedzi. Odebranie odpowiedzi oznacza zakończenie transakcji. Brak odpowiedzi powoduje powtórzenie komunikatu akcji i ponowne oczekiwanie na odpowiedź. Liczba powtórzeń jest parametrem konfiguracji sieci. Ponadto jeśli w czasie wykonywania transakcji zostanie przekroczony czas T_{TH} , to pomimo opóźnienia węzeł kontynuuje wykonywanie transakcji, aż do jej pełnego zakończenia, wliczając w to ewentualne powtarzanie komunikatów, które pozostały bez odpowiedzi.

Rodzaje transakcji. Podstawowym trybem pracy każdego węzła nadrzędnego jest cykliczne odpytywanie współpracujących z nim węzłów podrzędnych i nadrzędnych. Steruje tym zdefiniowana przez użytkownika lista odpytywania (*poll list*) zawierająca adresy węzłów i numery portów podlegających odpytywaniu. Każda pozycja listy określa jedną transakcję. Wszystkie transakcje związane z odpytywaniem są transakcjami niskiego priorytetu. Oprócz odpytywania, węzeł posiadający znacznik może wykonywać zlecane przez użytkownika transakcje sporadyczne, które mogą mieć priorytet wysoki lub niski.

Dodawanie i usuwanie węzłów. Dodawanie węzła nadrzędnego do pracującej sieci sprowadza się do włączenia go do pierścienia obiegu znacznika. Procedura włączania nowych węzłów polega na wysyłaniu w kolejnych obiegach znacznika zapytań pod kolejne niewykorzystane adresy sieci. Zapytanie jest transakcją niepriorytetową, wykonywaną w miarę wolnego czasu. Pozytywna odpowiedź na zapytanie oznacza pojawienie się nowego węzła. Węzeł posiadający znacznik przyjmuje adres nowego węzła jako adres następnika NS. Usunięcie węzła nie wymaga żadnych specjalnych działań. Węzeł, który nie przyjmuje znacznika jest automatycznie usuwany z pierścienia obiegu.

Odzyskiwanie znacznika. Podczas normalnej pracy w sieci nieustannie krąży znacznik. Stan beczynności może wystąpić w przypadku zniknięcia znacznika na skutek awarii lub przekłamania komunikatu. Wznowienie pracy wymaga odzyskania znacznika i ustalenia węzła sprawującego nadzór nad siecią. W tym celu każdy węzeł nadrzędny obserwuje stan sieci i mierzy czas jej beczynności. Jeśli długość tego czasu osiągnie pewną wartość, proporcjonalną do adresu węzła, to węzeł uznaje się za posiadacza znacznika i rozpoczyna normalne wykonywanie transakcji.

Planowanie obciążenia sieci. Realna długość cyklu obiegu znacznika zależy od szybkości sieci i od objętości danych, które powinny być przekazane w każdym cyklu. Parametry te należy określić podczas projektowania sieci, wyznaczając odpowiednie wartości:

n – liczba węzłów nadrzędnych

k – przewidywalna liczba transakcji niepriorytetowych w każdym cyklu

m – przewidywalna liczba powtórzeń w każdym cyklu

T_{TC} – czas przekazania znacznika

T_{MC} – czas transakcji priorytetowej (h), niepriorytetowej (l) i powtórzenia (r)

Po określeniu tych wartości możemy wyliczyć minimalną długość cyklu obiegu znacznika, możliwą do utrzymania w sieci, korzystając z następującego wzoru:

$$T_{TR} = n(T_{TC} + T_{MCh}) + kT_{MCl} + mT_{MCr}$$

Wartości czasów T_{TC} i T_{MC} zależą od szybkości transmisji i długości przekazywanych komunikatów. Przewidywana liczba powtórzeń zależy od stopy błędów. Prawidłowe oszacowanie tych parametrów i obliczenie okresu czasu T_{TR} stwarzają gwarancję przekazania zaplanowanych danych w każdym obiegu znacznika.

Protokół komunikacyjny

Warstwa liniowa przekazuje komunikaty nadawane i odbierane przez użytkowników różnych węzłów za pośrednictwem **portów**, czyli odrębnych „skrzynek pocztowych” określonych w poszczególnych węzłach sieci.

- Adresowanie: numery węzłów – numery portów
- Usługi:
 1. **Wysyłanie danych z potwierdzeniem (*SDA – Send Data with Acknowledge*)**
Wykonanie tej usługi polega na wysłaniu komunikatu akcji zawierającemu dane i odebraniu potwierdzenia odbioru. Potwierdzenie wysyła warstwa liniowa węzła adresata po odebraniu komunikatu akcji. Brak potwierdzenia lub błąd w jego odbiorze powodują retransmisję komunikatu akcji przez warstwę liniową węzła inicjującego.
Usługa może mieć priorytet wysoki lub niski.
 2. **Wysyłanie danych bez potwierdzenia (*SDN – Send Data with No acknowledge*)**
Wykonanie tej usługi polega na wysłaniu komunikatu akcji zawierającego dane do jednego lub wielu użytkowników sieci. Transakcja nie zawiera komunikatu odpowiedzi.
Usługa może mieć priorytet wysoki lub niski.
 3. **Wysyłanie danych i odebranie odpowiedzi (*SRD – Send and Request Data with reply*)**
Wykonanie usługi polega na wysłaniu komunikatu akcji i odebraniu komunikatu odpowiedzi. Komunikat akcji może zawierać dane i żądanie odpowiedzi, albo tylko żądanie odpowiedzi. Komunikat odpowiedzi może zawierać albo dane odczytane z bufora portu, albo potwierdzenie negatywne, informujące o braku danych w buforze. Brak odpowiedzi lub błąd w odbiorze odpowiedzi powodują retransmisję komunikatu akcji przez warstwę liniową węzła inicjującego.
Usługa może mieć priorytet wysoki lub niski.
 4. **Cykliczne wysyłanie danych i odbieranie odpowiedzi (*CSRD – Cyclic Send and Request Data with reply*)**
Usługa umożliwia użytkownikowi inicjującemu cykliczne odpytywanie jednego lub wielu użytkowników docelowych, których adresy węzłów i numery portów zostaną umieszczone na tzw. liście odpytywania (*poll list*). Proces odpytywania polega na wysłaniu komunikatów akcji do kolejnych użytkowników wymienionych na tej liście i odebraniu od nich komunikatów odpowiedzi. Po wyczerpaniu listy proces odpytywania powtarza się od początku. Dla zwiększenia częstotliwości odpytywania, czyli pewnego uprzywilejowania wybranych użytkowników docelowych, ten sam użytkownik może być wymieniony na liście wielokrotnie. Zarówno budowa komunikatów, jak i sposób ich wymiany z każdym użytkownikiem docelowym są takie same, jak podczas realizacji usługi *SRD*.
Wszystkie transakcje mają niski priorytet.

Sprzęg programu (API)

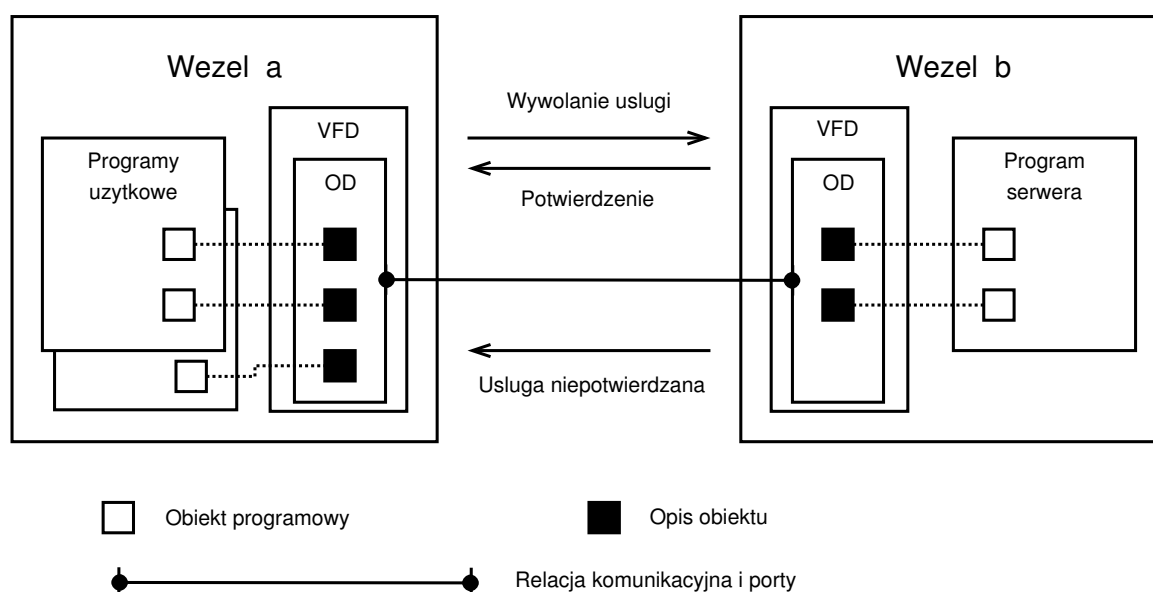
Standard sieci Profibus definiuje abstrakcyjne usługi sieciowe, natomiast nie określa sposobu wywoływania tych usług przez programy użytkowe. Sprzęg programu z usługami sieciowymi (API – *Application Program Interface*) zależy od wytwórcy oprogramowania i charakteru systemu operacyjnego i może być w różnych systemach różny.

Sprzęg komputera z siecią składa się z następujących elementów:

- układy nadajników i odbiorników sprzęgu RS-485 sterownika sieciowego (karty sieciowej), realizujące funkcje warstwy fizycznej;
- oprogramowanie mikrokomputera wbudowanego w sterownik sieciowy, realizujące protokół warstwy liniowej (FDL);
- proces serwera komunikacyjnego, realizujący protokół warstwy aplikacyjnej (FMS) i pośredniczący w komunikacji programu ze sterownikiem sieciowym;
- biblioteka funkcji języka C, realizująca funkcje sprzęgu programu (API) i umożliwiająca wywoływanie usług sieciowych warstw FDL i FMS;
- program konfiguratora, realizujący interakcyjne usługi warstw zarządzania.

12.1.2 Warstwa aplikacyjna

Usługi warstwy liniowej umożliwiają przekazywanie komunikatów o nieokreślonej strukturze wewnętrznej, nie rozpoznawanej przez oprogramowanie komunikacyjne. Ogranicza to automatyczną kontrolę poprawności komunikatów i utrudnia współpracę węzłów sieci, posługujących się różnymi formatami danych. Usługi warstwy aplikacyjnej zdefiniowane są w specyfikacji **FMS (*Fieldbus Message Specification*)** umożliwiają dostęp do obiektów programowych, takich jak: zmienne, tablice lub rekordy, istniejących w innych węzłach sieci. Udostępnione do komunikacji obiekty mogą przyjmować wartości należące do ściśle określonych typów, a format transmisji wartości typów jest szczegółowo określony.



Rysunek 42: Wirtualne urządzenia sieciowe, relacja komunikacyjna i wywołanie usług

Model komunikacyjny przyjęty w warstwie aplikacyjnej jest zgodny ze schematem klient-serwer, w którym serwer udostępnia swoje obiekty dla działań klientów (rys. 42). Wszystkie obiekty, zdefiniowane w danym węźle i przeznaczone do udostępnienia w sieci, muszą być opisane w **słowniku obiektów (OD – object directory)**, utrzymywanym przez warstwę aplikacyjną tego węzła. Większość usług warstwy aplikacyjnej stanowią usługi potwierdzone, których wywołanie przez klienta przekazuje do serwera komunikat zawierający żądanie wykonania operacji, np. odczytu lub zapisu wskazanego obiektu, a potwierdzenie wysłane przez serwer przynosi do klienta rezultat wykonania tej operacji. Nieliczną grupę stanowią usługi niepotwierdzone, których wykonanie przynosi komunikat informujący o stanie lub wartości jakiegoś obiektu. Te same programy mogą wywoływać różne usługi, pełniąc zarówno rolę klienta, jak i serwera.

Relacje komunikacyjne i połączenia

Definicja warstwy aplikacyjnej nie uwzględnia struktury oprogramowania narzuconej przez system operacyjny węzła i nie rozróżnia ani procesów, ani użytkowników zarejestrowanych w systemie. Każdy wykonywany program ma te same prawa dostępu do obiektów udostępnianych w sieci.

Warstwa liniowa implementuje w węźle sieci pewien zestaw portów, przez które przepływają wszystkie komunikaty wysyłane lub odbierane przez ten węzeł. Para portów – po jednym w każdym z dwóch współpracujących węzłów – tworzy **relację komunikacyjną** dostępną dla wykonywanych w tych węzłach programów. Definicja relacji komunikacyjnej obejmuje: numer portu własnego węzła, adres i numer portu węzła docelowego oraz listę usług, które mogą być w tej relacji realizowane. Poszczególne relacje można traktować jako odrębne kanały komunikacyjne, poprzez które programy mogą wywoływać usługi adresowane do innych węzłów. Wykonanie usługi warstwy aplikacyjnej polega na przesłaniu lub wymianie pojedynczych komunikatów w sieci.

Relacje komunikacyjne definiuje się w każdym węźle oddzielnie podczas konfigurowania węzła. Zestaw wszystkich relacji zdefiniowanych w danym węźle tworzy **listę relacji komunikacyjnych** tego węzła. Numer na tej liście (*communication reference – CREF*) jest identyfikatorem relacji używanym jako parametr adresowy w wywołaniach usług warstwy aplikacyjnej.

Niemal wszystkie relacje komunikacyjne są relacjami połączeniowymi. Raz nawiązanie połączenie jest symetryczne i dwukierunkowe, tzn. każdy ze współpracujących programów może pełnić rolę klienta lub serwera usług wywoływanych w tym połączeniu. Relacje bezpołączeniowe (*connectionless*) wykorzystuje się wyłącznie do przekazywania danych skierowanych do odbiorców w wielu węzłach sieci.

Obiekty

Warstwa aplikacyjna separuje użytkowników od techniki komunikacji sieciowej i nie odwołuje się do pojęcia komunikatu, lecz określa działania na obiektach w dziedzinie programu. Podstawowymi obiektami są:

- zmienna: prosta, tablica lub rekord
- zdarzenie sygnalizujące stan szczególnie we współpracującym programie
- domena (*domain*) – segment pamięci, do którego można załadować np. nowy program lub zestaw parametrów konfiguracyjnych
- program (załadowany do domeny), który można uruchomić, zatrzymać, itp.

- obiekt fizyczny (*physical access object*), charakteryzowany przez adres i długość, natomiast o nieznannej strukturze wewnętrznej.

Opis obiektu, przechowywany jest w słowniku obiektów, obejmuje następujące elementy:

- rodzaj obiektu
- typ danych związanych z tym obiektem
- adres miejsca zapisania danych
- długość danych
- atrybuty praw dostępu

Obiekty można wskazywać poprzez podanie indeksu w słowniku obiektów lub przez podanie nazwy.

Typy danych. Specyfikacja FMS definiuje 14 standardowych typów danych wyliczonych w tabeli poniżej.

Nazwa	Indeks	Długość		Nazwa	Indeks	Długość
Boolean	1	1		Floating Point	8	4
Integer8	2	1		Visible String	9	1, 2, 3, ...
Integer16	3	2		Octet String	10	1, 2, 3, ...
Integer32	4	4		Date	11	7
Unsigned8	5	1		Time Of Day	12	4 lub 6
Unsigned16	6	2		Time Difference	13	4 lub 6
Unsigned32	7	4		Bit String	14	1, 2, 3, ...

Usługi warstwy aplikacyjnej

Usługi warstwy aplikacyjnej udostępniają obiekty zdefiniowane w dowolnym węźle sieci programom wykonywanym w innych węzłach. Wykonywanie większości usług sprowadza się do przesyłania wartości przypisanych do obiektów lub poleceń zmieniających stan tych obiektów. Specyfikacja FMS definiuje każdą usługę za pomocą czterech operacji, których wykonywanie przez programy warstwy aplikacyjnej składa się na wykonywanie usługi.

Operacje:

request – wywołanie usługi przez klienta

indication – sygnalizacja odebrania wywołania

response – udzielenie odpowiedzi

confirm – sygnalizacja odebrania potwierdzenia

Klient	Serwer	Rodzaj usługi
<i>request</i> → <i>indication</i>		usługa potwierdzana
<i>confirm</i> ← <i>response</i>		
<i>indication</i> ← <i>request</i>		usługa niepotwierdzana

Lista wszystkich usług warstwy aplikacyjnej jest dość długa. W praktycznych realizacjach jest na ogół tylko pewien podzbiór usług, obejmujący wszystkie usługi zarządzające, usługi umożliwiające przekazywanie danych *Read* i *Write* oraz usługi związane z sygnalizowaniem zdarzeń.

Usługi zarządzające

Usługi tej grupy umożliwiają nawiązywanie i usuwanie połączeń oraz identyfikację węzłów współpracujących w ramach połączenia.

Initiate – nawiązanie połączenia

Wywołanie usługi powoduje nawiązanie połączenia we wskazanej relacji komunikacyjnej oraz uzgodnienie warunków współpracy: zestawu usług opcjonalnych, atrybutów identyfikujących prawa dostępu do obiektów i maksymalnej długości komunikatów sieciowych.

Abort – usunięcie połączenia

Wywołanie usługi powoduje natychmiastowe usunięcie połączenia. Usługa może być wywołana przez jeden z programów uczestniczących w połączeniu lub przez oprogramowanie komunikacyjne.

Reject – odrzucenie usługi

Usługa *Reject* jest wywoływana przez warstwę FMS w celu odrzucenia błędnego komunikatu.

Identify – identyfikacja współpracującego węzła

Wywołanie usługi powoduje odczytanie i przekazania do klienta parametrów identyfikujących rodzaj urządzenia znajdującego się po stronie serwera. Wywołanie usługi nie wymaga żadnych argumentów, a jej wykonanie przekazuje klientowi następujące dane: nazwa producenta, nazwa modelu i numer wersji urządzenia.

Status – odczytanie stanu współpracującego węzła

Wywołanie usługi powoduje przekazanie do klienta opisu stanu urządzenia znajdującego się po stronie serwera. Wywołanie usługi nie wymaga żadnych argumentów.

UnsolicitedStatus – raport stanu urządzenia. Wywołanie usługi powoduje wysłanie raportu zawierającego opis stanu własnego urządzenia. Treść opisu jest identyczna z treścią opisu przekazywanego za pomocą usługi *Status*. Usługa niepotwierdzana, wywoływana z priorytetem wysokim lub niskim.

Obsługa słownika obiektów (OD)

Obiekty udostępniane w sieci są opisane w słowniku obiektów serwera. Usługi tej grupy umożliwiają klientom odczytywanie opisów obiektów ze słownika, a także zapisywanie ich w słowniku.

Indeks	Rodzaj	Typ	Adres	...
20	zmienna	2	2300fa34	
21	zmienna	10	2300fa36	15-bajtów
23	tablica	8	23010000	6
30	rekord	16	23010018	

GetOD – odczytanie opisu słownika

Wywołanie usługi powoduje odczytanie ze słownika obiektów serwera opisu obiektu wskazanego przez indeks lub nazwę. Usługa możliwa również odczytanie ze słownika serii kolejnych opisów obiektów, określonych przez indeks pierwszego obiektu, od którego ma się rozpocząć odczytywanie. Liczba odczytanych opisów zależy od ich długości i od maksymalnej długości komunikatów przekazywanych w danym połączeniu.

InitiatePutOD – otwarcie sesji modyfikacji słownika obiektów

Wywołanie usługi przygotowuje serwer na przyjęcie i zapisanie w słowniku OD sekwencji opisów obiektów nadesłanych przez program klienta za pomocą usługi *PutOD*. Zamknięcie sesji modyfikacji następuje po wykonaniu usługi *TerminatePutOD*.

PutOD – zapisanie opisu obiektu lub zestawu opisów obiektów

Wykonanie usługi zapisuje w słowniku obiektów serwera sekwencję opisów obiektów nadesłanych przez program klienta. Obiekty, których opis składa się wyłącznie z indeksu są ze słownika usuwane. Wykonanie usługi nie wymaga żadnych argumentów, a jej poprawne wykonanie przekazuje klientowi kod potwierdzenia.

TerminatePutOD – zamknięcie sesji modyfikacji słownika obiektów

Wykonanie usługi zamyka sesję modyfikacji słownika OD i powoduje utworzenie wszystkich nowych obiektów. Jeżeli utworzenie jakiegoś obiektu jest niemożliwe, to zostaje odtworzony poprzedni stan słownika.

13 Wykład 13 [15.06.2004]

13.1 Sieć PROFIBUS - cd

13.1.1 Warstwa aplikacyjna - cd

Dostęp do zmiennych

Usługi działające na zmiennych umożliwiają odczytywanie i zapisywanie wartości zmiennych zdefiniowanych w innych węzłach sieci oraz komunikowanie wartości własnych zmiennych innym programom. Dostęp do obiektów złożonych (tablic lub rekordów) może dotyczyć zarówno całych obiektów – adresowanych przez podanie indeksu (lub nazwy), jak i poszczególnych elementów.

Read – odczytanie wartości zmiennej prostej lub złożonej

Wykonanie usługi powoduje odczytanie przez program klienta wartości zmiennej prostej lub złożonej, albo wartości elementu zmiennej złożonej, zdefiniowanych po stronie serwera w odległym węźle sieci. Argumentami wywołania usługi są: indeks (lub nazwa) zmiennej oraz podindeks elementu zmiennej złożonej.

Write – podstawienie wartości na zmienną prostą lub złożoną

Wykonanie usługi powoduje zapisanie przez program klienta nowej wartości zmiennej prostej lub złożonej, albo wartości elementu zmiennej złożonej, zdefiniowanych po stronie serwera w odległym węźle sieci. Argumentami wywołania usługi są: indeks (lub nazwa) zmiennej oraz podindeks elementu zmiennej złożonej.

InformationReport – wysłanie wartości zmiennej prostej lub złożonej

Wykonanie usługi powoduje wysłanie raportu zawierającego wartość zmiennej prostej lub złożonej, albo wartość elementu zmiennej złożonej. Usługa niepotwierdzana, wywoływana z priorytetem wysokim lub niskim.

ReadWithType – odczytanie wartości i opisu typu zmiennej

Wykonanie usługi powoduje odczytanie przez program klienta wartości zmiennej prostej lub złożonej, albo wartości elementu zmiennej złożonej, zdefiniowanych po stronie serwera w odległym węźle sieci. Wraz z wartością zmiennej do klienta jest przekazywany także opis typu odczytanej wartości. Argumentami wywołania usługi są: indeks (lub nazwa) zmiennej oraz podindeks elementu zmiennej złożonej.

WriteWithType – podstawienie wartości i opisu typu zmiennej

Wykonanie usługi powoduje zapisanie przez program klienta nowej wartości zmiennej prostej lub złożonej, albo wartości elementu zmiennej złożonej, zdefiniowanych po stronie serwera w odległym węźle sieci. Zapisywana wartość jest przekazywana przez program klienta wraz z opisem typu. Argumentami wywołania usługi są: indeks (lub nazwa) zmiennej oraz podindeks elementu zmiennej złożonej, wartość oraz opis typu danych przekazywanej wartości

InformationReportWithType – wysłanie wartości zmiennej prostej lub złożonej

Wykonanie usługi powoduje wysłanie raportu zawierającego wartość zmiennej prostej lub złożonej, albo wartość elementu zmiennej złożonej, oraz opis typu wysłanej wartości. Usługa nie jest potwierdzana.

PhysRead – odczytanie obiektu fizycznego

Wykonanie usługi powoduje odczytanie przez program klienta wartości obiektu fizycznego, zdefiniowanego po stronie serwera w odległym węźle sieci. Wartością obiektu jest ciąg bajtów, nie interpretowany przez oprogramowanie komunikacyjne. Argumentami wywołania usługi są adres i długość obiektu w pamięci serwera.

PhysWrite – zapisanie obiektu fizycznego

Wykonanie usługi powoduje zapisanie przez program klienta nowej wartości obiektu fizycznego, zdefiniowanego przez program serwera w odległym węźle sieci. Wartością obiektu jest ciąg bajtów, nie interpretowany przez oprogramowanie komunikacyjne. Argumentami wywołania usługi są adres i długość obiektu w pamięci serwera oraz wartość obiektu (ciąg bajtów).

DefineVariableList – utworzenie listy zmiennych

Wykonanie usługi powoduje utworzenie listy zawierającej wskazane zmienne i zapisanie opisu tej listy, jako nowego obiektu, w słowniku OD serwera. Wszystkie zmienne łączone w listę muszą być wcześniej zdefiniowane i opisane w słowniku OD.

DeleteVariableList – usunięcie listy zmiennych

Wykonanie usługi powoduje usunięcie opisu listy zmiennych ze słownika obiektów serwera. Usunięcie opisu listy zmiennych nie wpływa na istnienie i wartość samych zmiennych.

Dostęp do zmiennych

Zdarzenie jest sytuacją szczególną, o której muszą być powiadomione programy wykonywane w innych węzłach sieci. Lista zdarzeń jest ustalona, a zdarzenia są obiektami opisanymi w słowniku OD. Ze zdarzeniem mogą być związane dane, a kolejne wystąpienia zdarzenia mogą być numerowane liczbami naturalnymi. Powiadomienie o zdarzeniu polega na wysłaniu raportu zawierającego indeks zdarzenia, wartość danych związanych ze zdarzeniem oraz numer identyfikujący wystąpienie zdarzenia. Odbiorca zawiadomienia może potwierdzić zwrotnie fakt jego otrzymania. Odbiorca może też na bieżąco włączać i wyłączać możliwość wysyłania zawiadomień. Sposób obsługi zdarzeń odpowiada koncepcyjnie sieciowej implementacji przerwań, obejmującej możliwość zgłoszenia, potwierdzania odbioru zgłoszenia oraz maskowania zgłoszeń.

EventNotification – zawiadomienie o zdarzeniu

Wykonanie usługi powoduje wysłanie raportu zawierającego zawiadomienie o wystąpieniu zdarzenia. Usługa niepotwierdzana, wywoływana z priorytetem wysokim lub niskim.

AcknowledgeEventNotification – potwierdzenie zawiadomienia

Wykonanie usługi potwierdza odebranie zawiadomienia o wystąpieniu zdarzenia. Potwierdzenie dotyczy konkretnego wystąpienia zdarzenia o podanym numerze.

AlterEventConditionMonitoring – włączenie zawiadomień

Wykonanie usługi powoduje ustawienie lub wyzerowanie znacznika zezwalającego na wysłanie zawiadomień o wystąpieniu zdarzenia.

EventNotificationWithType – wysłanie zawiadomienia o zdarzeniu

Wykonanie usługi powoduje wysłanie do partnera komunikacyjnego zawiadomienia o wystąpieniu zdarzenia, zawierającego dane związane ze zdarzeniem oraz opis typu danych. Usługa nie jest potwierdzana.

Obsługa domen

Wiele systemów rozproszonych zawiera centralną stację nadzorującą zdalnie pracę wielu stacji podporządkowanych. Elementem tego nadzoru może być odczytywanie lub zapisywanie pamięci stacji podporządkowanej przez stację nadzorującą (np. zapis nowej konfiguracji lub programu). Dostępny dla stacji nadzorującej obszar pamięci stacji podporządkowanej nazywa się **domeną**. Usługi działające na domenach umożliwiają odczyt, zapis i przygotowanie stacji podporządkowanej do operacji dostępu do domeny.

InitiateDownloadSequence – rozpoczęcie zapisywania domeny

Wykonanie usługi przygotowuje serwer na przyjęcie od klienta zawartości domeny (ładowanie pamięci serwera przez program klienta).

DownloadSegment – zapisanie danych do domeny

Wykonanie usługi zapisuje do domeny serwera segment danych nadesłany przez program klienta.

TerminateDownloadSequence – zakończenie zapisywania domeny

Wykonanie usługi informuje serwer o zakończeniu zapisywania domeny przez program klienta.

RequestDomainDownload – żądanie zapisu domeny

Wykonanie usługi przekazuje partnerowi żądanie przysłania nowego segmentu danych do domeny.

InitiateUploadSequence – rozpoczęcie odczytywania domeny

Wykonanie usługi przygotowuje serwer na odczytanie przez klienta zawartości domeny.

UploadSegment – odczytanie danych z domeny

Wykonanie usługi odczytuje segment danych z domeny serwera i przekazuje go do programu klienta.

TerminateUploadSequence – zakończenie odczytywania domeny

Usługa informuje serwer o zakończeniu odczytywania domeny przez program klienta.

RequestDomainUpload – żądanie odczytu domeny

Wykonanie usługi przekazuje partnerowi żądanie odczytania segmentu danych z domeny.

Obsługa programów

Zawartością domeny, lub kilku domen, może być program. Każdy program jest traktowany jako odrębny obiekt, który musi być opisany w słowniku obiektów OD. Usługi warstwy aplikacyjnej umożliwiają zdalne definiowanie programu, uruchamianie, zawieszanie, wznowianie i kończenie tego wykonania.

CreateProgramInvocation – utworzenie programu

Wykonanie usługi definiuje nowy program i zapisuje opis tego obiektu w słowniku OD serwera.

DeleteProgramInvocation – usunięcie programu

Wykonanie usługi usuwa opis programu ze słownika obiektów po stronie serwera.

Start – uruchomienie programu

Wykonanie usługi powoduje rozpoczęcie wykonania programu od początku.

Stop – zawieszenie programu

Wykonanie usługi powoduje zapamiętanie stanu programu i zawieszenie jego wykonywania (z możliwością wznowienia).

Resume – wznowienie programu

Wykonanie usługi powoduje odtworzenie zapamiętanego stanu programu i wznowienie jego wykonywania.

Reset – wyzerowanie stanu programu

Wykonanie usługi powoduje skasowanie stanu zawieszonego programu i ustawienie stanu początkowego.

Kill – zablokowanie programu

Wykonanie usługi powoduje zatrzymanie programu i trwałe zablokowanie możliwości ponownego uruchomienia. Zablokowany program można usunąć za pomocą usługi *DeleteProgramInvocation*.

Wykonywanie usług

Domunującym trybem pracy urządzeń wchodzących w skład rozproszonego systemu sterującego jest cykliczne, powtarzalne wykonywanie stale tych samych zadań, takich jak pomiar stanu instalacji, obliczenie nowych wartości sterujących i przekazanie sterowań do urządzeń wykonawczych. W każdym cyklu pracy zmierzone wartości parametrów procesu oraz obliczone wartości sterowań mogą być przekazywane do innych, współpracujących, urządzeń systemu. Oprócz zadań cyklicznych urządzenia wykonują również pewne działania okazjonalne (sporadyczne), których wykonanie może również wymagać wymiany danych z urządzeniami współpracującymi.

Różne tryby pracy i komunikowania się węzłów sieci – cykliczny lub niecykliczny – stwarza różne wymagania na sposób zorganizowania komunikacji, co znajduje wyraz w występowaniu różnych typów relacji komunikacyjnych. W relacjach tych mogą być wywoływane te same usługi, jednak sposób ich wykonywania może być w różnych realizacjach różny.

Specyfikacja FMS definiuje następujące typy realizacji komunikacyjnych:

MSCY, MSCY_SI – relacje cykliczne, łączące węzeł nadrzędny z podrzędnym

MSAC, MSAC_SI – relacje niecykliczne, łączące węzeł nadrzędny z podrzędnym

MMAC – relacje niecykliczne, łączące dwa węzły nadrzędne

MULT, BRCT – relacje niecykliczne, używane do rozgłaszania w sieci

W relacjach typu MS... wszystkie usługi potwierdza oraz usługi niepotwierdzone o niskim priorytecie są realizowane za pomocą usługi CSRD warstwy liniowej. Usługi niepotwierdzone o wysokim priorytecie są realizowane za pomocą usługi SRD. Adresy węzłów podrzędnych i numery portów wykorzystywanych w tych węzłach we wszystkich relacjach tego typu muszą być umieszczone na liście odpytywania węzła nadrzędnego. Cała lista odpytywania musi być zdefiniowana w jednym porcie węzła, który musi być wykorzystywany przez wszystkie relacje tego typu w danym węźle.

W relacjach typu MMAC wszystkie usługi FMS, zarówno potwierdzone, jak niepotwierdzone, realizuje się za pomocą usługi SDA warstwy liniowej. Warto zauważyć, że dzięki temu wykonanie usług niepotwierdzonych odbywa się ze zwrotnym potwierdzeniem poprawności odebrania komunikatu sieciowego przez warstwę liniową.

W relacjach typu MULT i BRCT dozwolone są tylko niepotwierdzone usługi FMS, które realizuje się za pomocą usługi SDN warstwy liniowej.

Relacje cykliczne **MSCY** i **MSCY_SI**

Relacje **MSCY** (*Master Slave Cyclic*) i **MSCY_SI** (*Master Slave Cyclic with Slave Initiative*) są relacjami połączeniowymi. Wymiana danych między współpracującymi węzłami jest możliwa w takiej relacji dopiero po nawiązaniu połączenia za pomocą usługi *Initiate*. Po nawiązaniu połączenia program wykonywany w węźle nadrzędnym może wywołać usługi potwierdzone *Read* i *Write* oraz wszystkie usługi niepotwierdzone. Kolejna usługa potwierdzana może być wywołana dopiero po zakończeniu wykonywania usługi poprzedniej. Program wykonywany w węźle podrzędnym może w

realacji MSCY jedynie usunąć połączenie za pomocą usługi *Abort*, w e relacji MSCY_SI może ponadto wywoływać wszystkie usługi niepotwierdzone.

Usługę *Initiate* może wywołać tylko program wykonywany w węźle nadrzędnym. Wykonanie tej usługi uruchamia proces odpytywania współpracującego węzła podrzędnego (usługa CSRD warstwy liniowej), kontynuowany aż do chwili usunięcia połączenia. Dzięki ciągłemu odpytywaniu węzeł podrzędny może w relacji MSCY_SI przekazywać do węzła nadrzędnego komunikaty przenoszące dane usług niepotwierdzanych. W relacji MSCY wywołanie usług niepotwierdzanych w węźle podporządkowanym jest zabronione.

Węzeł nadrzędny (<i>master</i>)		sieć	Węzeł podrzędny (<i>slave</i>)	
FMS	FDL		FDL	FMS
→ Read.req	Fdl_Send_Update.req	→SRD (dane)→ ← SRD (ϕ) ←	Fdl_Data_Reply.ind	Read.ind → Read.res ←
			Fdl_Reply_Update.req	
← Read.con	Fdl_Cyc_Data_Reply.con	→ SRD (ϕ) → ←SRD (dane)←	Fdl_Data_Reply.ind	Read.ind → Read.res ←
→ Read.req ← Read.con		→ SRD (ϕ) → ←SRD (dane)←	Fdl_Reply_Update.req	
	Fdl_Cyc_Data_Reply.con		Fdl_Data_Reply.ind	Read.ind →
→ Read.req ← Read.con		→ SRD (ϕ) → ← SRD (ϕ) ←		

Harmonogram realizacji usługi *Read* w relacji cyklicznej MSCY jest pokazany w tablicy powyżej. Pierwsze wywołanie operacji *Read.req* przez program klienta przenosi do węzła podrzędnego indeks żadanego obiektu. Odpowiedź węzła podrzędnego, przekazana w tym samym cyklu odpytywania, nie zawiera żadnych danych. Oprogramowanie węzła podrzędnego zapamiętuje indeks w obszarze buforowym (*Image Data Memory – IDM*) i za pomocą operacji *Read.ind* zawiadamia program użytkownika o wywołaniu usługi. Wartość odczytywanego obiektu, przekazana przez program użytkownika za pomocą operacji *Read.res*, jest zapisywana w buforze wyjściowym portu (operacja *Fdl_Reply_Update.req*) i przenoszona w sieci w następnym cyklu odpytywania. Oprogramowanie węzła nadrzędnego zapamiętuje odebraną wartość w obszarze buforowym IDM i przekazuje ją do programu klienta za pomocą operacji *Read.con*.

Wysłanie do węzła nadrzędnego danych, przekazanych przez operację *Read.res*, ponownie inicjuje w węźle podrzędnym operację *Read.ind*, w odpowiedzi na którą program użytkownika może – za pomocą operacji *Read.res* – przekazać nową wartość obiektu. Tak jak poprzednio, nowe dane są

przenoszone w sieci w najbliższym cyklu odpytywania i zapisywane w obszarze IDM węzła nadrzędnego. Proces ten jest powtarzany permanentnie bez względu na działania programu klienta.

Kolejne wywołania operacji *Read.req* przez program klienta nie są już bezpośrednio związane z przesyłaniem jakichkolwiek komunikatów sieciowych. Każde następne wywołanie powoduje odczytanie danych zapisanych w obszarze IDM własnego węzła i przekazanie ich do programu za pomocą operacji *Read.con*.

Sposób wykonania usług niepotwierdzanych wywołanych w węźle nadrzędnym jest analogiczny do sposobu wykonania operacji *.req* i *.ind* pierwszego wywołania usługi *Read*. Oczywiście, nie występują tu operacje *.res* i *.con*. Każde wywołanie usługi powoduje przekazanie danych w sieci i zapisanie ich w buforze wejściowym portu węzła odbierającego. Priorytet transakcji odpytywania realizującej usługę niepotwierdzaną jest równy priorytetowi wywołania usługi.

Sposób wykonania usługi niepotwierdzanych wywołanych w węźle podrzędnym jest analogiczny do sposobu wykonania operacji *.res* lub *.con* pierwszego wywołania usługi *Read*. To znaczy, wywołanie usługi powoduje zapisanie danych w odpowiednim buforze wyjściowym portu. Dane te będą przesłane do węzła nadrzędnego w jednym z następnych cykli odpytywania. Zgodnie z regułami odpytywania dane zapisane w buforze wyjściowym o wysokim priorytecie (związane z usługami o wysokim priorytecie) są przekazywane przed danymi zapisanymi w buforze o priorytecie niskim.

Sposób wykonania usług potwierdzanych w relacjach cyklicznych odpowiada koncepcyjnie sieciowej implementacji pamięci wspólnej, w której proces odpytywania zapewnia spójność kopii, przechowywanej w węźle nadrzędnym, z wartościami zmiennych istniejących w węźle podrzędnym. Definicja usług stwarza jednak niebezpieczeństwo niewykywalnej dezaktualizacji kopii w wypadku przerwania komunikacji. Dla uniknięcia tego niebezpieczeństwa standard narzuca obowiązek monitorowania połączeń w relacjach cyklicznych, z zadaniem okresem kontroli (*Control Interval – ci*). W tym celu oprogramowanie węzła nadrzędnego musi zagwarantować wykonywanie co najmniej jednego odpytania w każdym okresie *ci*. Niedotrzymanie tego warunku jest wykrywalne przez obydwa węzły i powoduje usunięcie połączenia. Czas *ci* jest jednym z parametrów konfiguracyjnych relacji komunikacyjnej. Wartość tego parametru musi być na obu końcach relacji taka sama.

Relacje MSAC i MSAC_SI

Relacje MSAC (*Master Slave Acyclic*) i MSAC_SI (*Master Slave Acyclic with Slave Initiative*) są relacjami połączeniowymi. Wymiana danych między współpracującymi węzłami musi być w takiej relacji poprzedzona nawiązaniem połączenia za pomocą usługi *Initiate*. Po nawiązaniu połączenia program wykonywany w węźle nadrzędnym może wywoływać wszystkie usługi potwierdzane i niepotwierdzane. Usługi mogą być wykonywane równolegle, tzn. program może wywołać następną usługę zanim otrzyma potwierdzenie wykonania poprzedniej. Program wykonywany w węźle podrzędnym może w relacji MSAC jedynie usunąć połączenie za pomocą usługi *Abort*, w relacji MSAC_SI może ponadto wywoływać usługi niepotwierdzane.

Usługę *Initiate* może wywołać tylko program wykonywany w węźle nadrzędnym. Wykonywanie tej usługi w relacji MSAC_SI uruchamia proces odpytywania współpracującego węzła podrzędnego (usługa CSRD warstwy liniowej), który może dzięki temu przekazywać do węzła nadrzędnego komunikaty przenoszące dane usług niepotwierdzanych. W relacji MSAC proces odpytywania jest uruchamiany w chwili wywołania dowolnej usługi potwierdzanej (w tym również usługi *Initiate*) i jest kontynuowany tylko przez czas niezbędny do wykonania tej usługi.

Wykonanie usług potwierdzanych przebiega podobnie do wykonania pierwszego wywołania usługi *Read* w realcji MSCY, tzn. każdemu wywołaniu usługi odpowiada w sieci odrębna para przesłań: zadanie – potwierdzenie. Z każdym wywołaniem usługi potwierdzanej jest związany indywidualny identyfikator (*Invoke ID*), przekazywany w każdym komunikacie sieciowym. Identyfikator ten umożliwia jednoznaczne przyporządkowanie komunikatów do równolegle realizowanych usług potwierdzanych.

Węzeł nadrzędny (<i>master</i>)			Węzeł podrzędny (<i>slave</i>)	
FMS	FDL	sieć	FDL	FMS
← InfoR.ind	Fdl_Cyc_Data_Reply.con	→ SRD (ϕ) → ← SRD (dane) ←	Fdl_Reply_Update.req	InfoR.req ←
	Fdl_Send_Update.req Fdl_Send_Update.con	→ SRD (ack) → ← SRD (ϕ) ←		
	Fdl_Data_Reply.con		Fdl_Data_Reply	

Usługi niepotwierdzone są realizowane tak samo jak usługi niepotwierdzone w relacjach MSCY i MSCY_SI. Harmonogram realizacji usługi *InformationReport* (relacja MSAC_SI) wywołanej w węźle podrzędnym jest pokazana w tabeli powyżej. Wywołanie usługi powoduje zapisanie danych, które mają być przekazane do partnera, w buforze wyjściowym portu (operacja *Fdl_Reply_Update.req* warstwy liniowej). Najbliższy cykl odpytywania przenosi dane do węzła docelowego, zapisuje je w buforze wejściowym portu i za pomocą operacji *InformationReport.ind* zawiadamia program użytkownika o odebraniu wywołania usługi. Po odczytaniu przez użytkownika danych z bufora wejściowego portu oprogramowanie komunikacyjne przekazuje do węzła nadawcy komunikat, który zwalnia bufor wyjściowy portu w tym węźle. Operacja przekazania potwierdzenia jest wykonywana automatycznie i w żaden sposób nie angażuje użytkownika.

Relacja MMAC

Relacja MMAC (*Master Master Acyclic*) jest relacją połączeniową, łączącą dwa węzły nadrzędne. Wymiana danych między współpracującymi węzłami jest możliwa dopiero po nazwianiu w danej relacji połączenia za pomocą usługi *Initiate*. Po nawiązaniu połączenia programy wykonywane w obydwu węzłach mogą być wykonywane równolegle, tzn. program może wywołać następną usługę zanim otrzyma potwierdzenie wykonywania poprzedniej.

Wykonanie usługi potwierdzanej sprowadza się do wymiany w sieci pary komunikatów, przekazywanych za pomocą usługi SDA warstwy liniowej. Z każdym wywołaniem usługi potwierdzanej jest związany indywidualny identyfikator (*Invoke ID*), przekazywany w każdym komunikacie sieciowym. Identyfikator ten umożliwia jednoznacznie przyporządkowanie komunikatów do równolegle realizowanych usług potwierdzanych. Harmonogram realizacji usługi *Read* w relacji MMAC jest przedstawiony w poniższej tabeli.

Usługi niepotwierdzone są wykonywane również za pomocą usługi SDA warstwy liniowej. Wykonanie usługi wywołanej w węźle nadrzędnym przebiega podobnie do wykonania pary operacji *.req* i *.ind* w tabeli poniżej, a wykonanie usługi wywołanej w węźle podrzędnym – podobnie do wykonania pary operacji *.res* i *.con*. Pomimo braku komunikatu usługi niepotwierdzonej jest związane ze

Węzeł nadrzędny (<i>master</i>)			Węzeł podrzędny (<i>slave</i>)	
FMS	FDL	sieć	FDL	FMS
→ Read.req	Fdl_Data_Ack.req	→SDA (dane)→ ← SDA (ack) ←	Fdl_Data_Ack.ind	Read.ind → Read.res ←
	Fdl_Data_Ack.req		Fdl_Data_Ack.req	
		← SRD (data) ← → SRD (ack) →		
← Read.con	Fdl_Data_Ack.req		Fdl_Data_Ack.con	

zwrotnym potwierdzeniem odbioru na poziomie warstwy liniowej.

Relacje **MULT** i **BRCT**

Relacje **MULT** i **BRCT** są relacjami bezpołączeniowymi, z których każda może łączyć wiele węzłów nadrzędnych. Węzły te mogą wywoływać tylko usługi niepotwierdzone. Wykonanie usługi sprowadza się do wysłania komunikatu rozgłaszanego w sieci za pomocą usługi SDN warstwy liniowej.

14 Wykład 14 [18.06.2004]

14.1 Rozproszony system sterujący w praktyce

Przykład: Automatyizacja budynku (BMS)

1. Instalacje klimatyzacyjne budynku:

- centrale nawiewne (*Air Handler Unit – AHU*)
- węzeł cieplny:
 - lokalne piece gazowe/olejowe
 - wymienniki ciepła z sieci miejskiej
 - agregaty chłodzące (*chiller*)
- wentylatory wyciągowe

2. Funkcje systemu:

- sterowanie bieżące
- realizacje harmonogramu:
 - dzienny
 - świąteczny
 - wakacyjny
- alarmy
- zobrazowanie sytuacji
- kontrola dyspozytorów
- współpraca z innymi systemami (pożarowe, itp.)

3. Architektura systemu (Bacnet)

4. Usługi sieciowe

- odpytywanie i monitorowanie instalacji
- zdarzenia i przekazywanie alarmów
- odczyt/zapis danych i komendy operatorskie
- download programu i programowanie
- upload – odczyt trendlogów

Indeks

- Abort, 69
- abort, 35
- AcknowledgeEventNotification, 72
- administrator sieci, 49
- administrator urządzeń, 49
- administrator zadań, 49
- administrator zbiorów, 49
- adresowanie, 59
- AHU, 79
- alarm, 35
- algorytm adaptacyjny, 32, 50
- algorytm FIFO, 32, 50
- algorytm karuzelowy, 32, 50
- AlterEventConditionMonitoring, 72
- API, 66
- atrybuty kolejek, 43
- Background System, 17
- BMS, 79
- BRCT, 74, 78
- broadcast, 59, 60
- Bus Lock, 28
- Bus Unock, 28
- Buttazzo, 12
- ci, 76
- close, 46
- communication reference, 67
- confirm, 68
- connection-orientred, 59
- connectionless, 67
- CreateProgramInvocation, 73
- Creceive, 51
- CREF, 67
- CSMA, 57
- CSMA/CD, 56
- CSRD, 65
- cycle, 10
- cykliczność, 48
- Cykliczny program sekwencyjny, 13
- datagram, 58
- deadline, 10
- deadlock, 40
- DefineVariableList, 72
- delay, 48
- DeleteProgramInvocation, 73
- DeleteVariableList, 72
- depozyty, 51
- Device Drivers, 49
- Device Manager, 49
- domain, 67
- domena, 67, 72
- DownloadSegment, 73
- Earliest Deadline First, 10
- EDF, 10, 11
- event, 38
- event-triggered system, 8
- EventNotification, 72
- EventNotificationWithType, 72
- exchange, 28
- exec, 31
- execl, 31
- execle, 31
- execlp, 31
- execlpe, 31
- execv, 31
- execve, 31
- execvp, 31
- execvpe, 31
- Fieldbus Message Specification, 66
- FIFO, 32, 38, 50
- Filesystem Manager, 49
- First In First Out, 32
- FMS, 66
- Foreground System, 17
- fork, 30
- funkcje jądra, 23
- GetOD, 69
- gotowy, 21
- Hard Real-Time System, 5
- Identify, 69
- IDM, 75
- indication, 68
- InformationReport, 71
- InformationReportWithType, 71
- Initiate, 69
- InitiateDownloadSequence, 73
- InitiatePutOD, 70
- InitiateUploadSequence, 73
- inp, 53
- inpd, 53

inpw, 53
 Intel, 28
 Invoke ID, 77
 inwersja priorytetów, 41, 51
 iRMX86, 41

 Kill, 74
 kill, 33
 kolejki wiadomości, 42, 52
 komunikacja zadań, 36

 Layland, 10
 Liu, 10
 Liu&Layland, 10
 ltrunc, 46

 martwy, 21
 maskowanie sygnałów, 34
 master, 23, 58
 message queue, 42, 52
 MMAC, 74, 77
 mmap, 46
 Motorola, 28
 mprotect, 46
 mq_close, 44
 mq_curmsg, 43
 mq_flags, 43
 mq_getattr, 44
 mq_maxmsg, 43
 mq_msgsize, 43
 mq_notify, 45
 mq_open, 44
 mq_receive, 43, 44
 mq_send, 43, 44
 mq_setattr, 44
 mq_unlink, 44
 MSAC, 74, 76
 MSAC_SI, 74, 76
 MSCY, 74
 MSCY_SI, 74
 MULT, 74, 78
 multitasking, 18
 munmap, 46

 nanosleep, 48
 Network Manager, 49

 O_CREAT, 43
 O_NONBLCK, 43
 O_RDONLY, 43
 O_RDWR, 43
 O_WRONLY, 43

 obiekt fizyczny, 68
 object directory, 67
 obsługa sygnałów, 33
 OD, 67
 odpytywanie, 58
 Ograniczenia czasowe - łagodne, 6
 ograniczenia czasowe - łagodne, 19
 ograniczenia czasowe - ostre, 5, 19
 opóźnienia, 48
 outp, 53
 outpd, 53
 outpw, 53

 pamięć wspólna, 46
 pause, 33
 physical access object, 68
 PhysRead, 71
 PhysWrite, 72
 PID, 26, 30
 pierścień znacznikowy, 58
 pipe, 35, 52
 podwarstwa łącza logicznego, 58
 podwarstwa dostępu do kabla, 56
 poll list, 64, 65
 polling, 58
 porty, 59, 65
 POSIX, 32, 38
 POSIX_PRIORITY_SCHEDULING, 43
 post, 38
 potoki, 52
 pre-scheduling, 15
 priority inversion, 41
 proces, 21, 26
 Proces identyfikator, 30
 proces macierzysty, 30, 31
 proces potomny, 30, 31
 process, 21, 26
 Process Field Bus, 60
 Process identifier, 26
 Process Manager, 49
 PROFIBUS, 60
 program, 21
 Program Status World, 26
 protokół bezpołączeniowy, 58
 proxy, 51
 przerwanie, 53
 przeterminowanie, 48
 PSW, 26
 pthread, 31
 pthread_create, 31
 pthread_detach, 31

pthread_exit, 31
 pthread_join, 31
 PutOD, 70

 QNX, 49
 qnx_hint_attach, 53
 qnx_hint_detach, 53
 qnx_hint_mask, 53
 qnx_proxy_attach, 52
 qnx_proxy_detach, 52
 qnx_proxy_rem_attach, 52
 qnx_proxy_rem_detach, 52

 raise, 33
 Rate Monotonic Scheduling, 10, 11
 reactive systems, 8
 Read, 71
 Readmsg, 51
 ReadWithType, 71
 Ready List, 26
 real rotation, 63
 Real Time System, 5
 Receive, 51
 receive-control, 41
 region, 41
 Reject, 69
 relacja komunikacyjna, 67
 rendez-vous, 45, 51
 Reply, 51
 request, 68
 RequestDomainDownload, 73
 RequestDomainUpload, 73
 Reset, 73
 response, 68
 Resume, 73
 resume, 37
 RL, 26
 RMS, 10, 11
 Round Robin, 32
 rozgłaszanie, 59
 RR, 32
 RT, 26
 Running Task, 26

 słownik obiektów, 67
 SAP, 59
 SDA, 65
 SDN, 65
 sem_destroy, 40
 sem_init, 40
 sem_post, 40
 sem_trywait, 40
 sem_wait, 40
 semafor, 37, 52
 semaphore, 37, 52
 Send, 51
 send-control, 41
 Sensini, 12
 Service Access Point, 59
 shared memory object, 46
 shell, 33
 shm_open, 46
 shutdown, 35
 sieci miejscowe, 53
 sig_atomic_t, 34
 SIGABRT, 35
 sigaddset, 34
 SIGALRM, 35
 SIGBUS, 35
 SIGCHLD, 35
 SIGCONT, 35
 sigdelset, 34
 sigempty, 34
 sigfillset, 34
 SIGFPE, 34
 SIGHUP, 35
 SIGILL, 34
 SIGINT, 35
 sigismember, 34
 SIGKILL, 35
 siglongjmp, 34
 signal, 38, 52
 sigpending, 34
 SIGPIPE, 35
 sigprocmask, 34
 SIGPWR, 35
 SIGQUIT, 35
 SIGSEGV, 34
 sigsetjmp, 34
 SIGSTOP, 35
 sigsuspend, 34
 SIGTERM, 35
 SIGUSR1, 35
 SIGUSR2, 35
 slave, 58
 sleep, 48
 Soft Real-Time System, 6
 spotkanie, 45, 51
 sprzęg RS-485, 60
 Spuri, 12
 SRD, 65

standard POSIX, 21
 stany procesu, 21
 Start, 73
 Status, 69
 stimulus response path, 8
 Stop, 73
 suspend, 37
 sygnały, 33, 52
 synchronizacja zadań, 36
 System Czasu Rzeczywistego, 5
 system czasu rzeczywistego, 49
 system dwuplanowy, 17
 system plików, 21
 system wielozadaniowy, 21
 systemy wielozadaniowe, 18
 szeregowanie priorytetowe, 10
 szeregowanie procesów, 32
 szeregowanie wg terminów wykonywania, 10

 tablice systemowe, 26
 target rotation, 63
 task, 26
 Task Control Block, 26
 Task descriptor, 26
 Task identifier, 26
 task priority, 26
 Task state table, 26
 TCN, 26
 TD, 26
 TDMA, 57
 terminal, 21
 TerminateDownloadSequence, 73
 TerminatePutOD, 70
 TerminateUploadSequence, 73
 thread, 26
 TID, 26
 time sharing, 50
 time-triggered system, 8
 timer, 48
 timer_create, 48
 timer_delete, 49
 timer_gettime, 49
 timer_settime, 48
 token passing, 56
 token ring, 58
 Trigger, 52
 tryb master, 23
 tryb user, 23
 TST, 26
 tv_nsec, 48
 tv_sec, 48

 Twierdzenie Liu&Layland'a, 10
 twierdzenie Spuri, Buttazzo, Sensini, 12
 tworzenie wątków, 31

 UnsolicitedStatus, 69
 UploadSegment, 73
 user, 23

 wątek, 26, 31
 wait, 38
 warstwa łącza danych, 56, 61
 warstwa fizyczna, 60
 warstwa liniowa, 56, 61
 wielodostęp, 21
 wielozadaniowość, 21
 Write, 71
 Writemsg, 51
 WriteWithType, 71
 wykonywany, 21

 zadania sterujące urządzeniami, 49
 zakleszczenie, 40
 zawieszony, 21
 zdarzenia, 21
 zdarzenia czasowe, 22
 zdarzenia zewnętrzne, 22

Spis rysunków

1	Przejazd kolejowy	6
2	Regulacja Temperatury	7
3	Transmisja pakietów	8
4	Sterowanie zdarzeniami	9
5	Szeregowania algorytmem RMS	12
6	Szeregowania algorytmem EDF	12
7	Cykliczny program sekwencyjny	14
8	Cykliczny program sekwencyjny (2 wersja)	16
9	Obsługa zdarzenia	16
10	System Dwuplanowy - Foreground	17
11	System Dwuplanowy - Background	18
12	Współrzędne wykonywanie zadań przez podział czasu procesora	18
13	Graf stanów zadania	22
14	Tryby pracy SO	23
15	Model budowy systemu operacyjnego opartego na mikrojądrze	23
16	Model budowy systemu operacyjnego opartego na mikrojądrze	24
17	Model budowy egzekutora wielozadaniowego	24
18	Model budowy systemu operacyjnego opartego na jądrze monolitycznym	25
19	Graf stanów zadania w systemie opartym na jądrze monolitycznym	25
20	Tworzenie, zakańczanie, zawieszanie, wznowianie i usuwanie procesu	27
21	Schemat programu szeregującego	28
22	Schemat semafora aktywnego	29
23	Tworzenie procesu - FORK	30
24	Problemy synchronizacji i komunikacji zadań	36
25	Błędna synchronizacja zadań	36
26	Błędna synchronizacja zadań w systemie rejestracji	37
27	Operacje semaforowe	38
28	Poprawna synchronizacja zadań	39
29	Poprawna synchronizacja zadań w systemie rejestracji	39
30	Zakleszczenie	40
31	Inwersja priorytetów	41
32	Spotkanie (rendez-vous)	45
33	Obszary pamięci wspólnej (shared memory object)	46
34	Programowalny licznik czasu (timer)	48
35	Graf stanów zadań w systemie QNX	50
36	Implementacja spotkania w systemie QNX	51
37	Architektura klient-serwer i spotkanie	52
38	Przekazywanie wiadomości przez połączenie	52
39	Przekazywanie depozytów między węzłami sieci	53
40	Struktura funkcjonalna systemu sterującego	54
41	Warstwy modelu sieci PROFIBUS	60
42	Wirtualne urządzenia sieciowe, relacja komunikacyjna i wywołanie usług	66

Literatura

- [1] Krzysztof Sacha. „*Systemy Czasu Rzeczywistego*”. Oficyna wydawnicza Politechniki Warszawskiej, Warszawa, 1999.
- [2] Krzysztof Sacha. „*Laboratorium systemu QNX*”. Oficyna wydawnicza Politechniki Warszawskiej, Warszawa, 2001.
- [3] Krzysztof Sacha. „*Sieci miejscowe PROFIBUS*”. Wydawnictwo MIKOM, Warszawa, 1998.