



Politechnika Gdańskia
Wydział Elektrotechniki i Automatyk
Katedra Automatyki

Systemy Czasu Rzeczywistego

...
(ang. Real Time Operating Systems - RTOS)

Wykład w ramach przedmiotu:

Komputerowe Systemy Sterowania

dr inż. Tomasz Rutkowski



Zespół Inteligentnych Systemów
Wspomagania Decyzji i Sterowania

21.05.2008



Trochę Historii



Wszystko zaczęło się...

Lata 60 i 70 XX wieku – maszyny wolnostojące z dedykowanym oprogramowaniem (silna więź pomiędzy sprzętem a oprogramowaniem)

Rok 1992 – powstaje konsorcjum o nazwie PC/104, jego działania związane głównie z rozwojem aplikacji do zastosowań militarnych i medycznych, opracowanie standardu mikroprocesorowego (płytki mniejsza niż 1 dm²)

Rok 1993 – opracowanie standardu POSIX 1003.1b dotyczącego systemów czasu rzeczywistego



Standard POSIX - historia

(ang. Portable Operating System Interface)



Prace nad POSIX-em rozpoczęto w 1985 roku – próba standaryzacji różnych odmian systemu UNIX

Początkowo pracami kierowało IEEE (ang. Institute of Electrical and Electronics Engineers), stąd: **POSIX = IEEE 1003**

Od 1996 roku pieczę nad rozwojem standardu POSIX trzyma The Open Group (konsorcjum przemysłowe sponsorowane przez IBM, Sun, HP, Hitachi, Fujitsu, NEC) ze współpracą z IEEE

Kolejne wydania standardu POSIX noszą nazwę:

Single UNIX Specification, Version x

Aktualna wersja POSIX-a : 3 (od 2001 roku)

Od 2003 roku POSIX jest normą międzynarodową:
ISO/IEC 9945:2003

3



Standard POSIX – P1003



P1003.1

Definiuje interfejs aplikacji tak, aby była ona w pełni przenośna pomiędzy różnymi.

P1003.1a

Zestaw różnych interpretacji, wyjaśnień i rozszerzeń.

P1003.1b

Rozszerzenia dotyczące systemów czasu rzeczywistego.

P1003.1c

Dodanie funkcji wspierających wątki.

P1003.1d

Kolejne rozszerzenia wspierające systemy czasu rzeczywistego.

P1003.1e

Rozszerzenia dotyczące bezpieczeństwa systemu spełniające kryteria bezpieczeństwa opublikowane przez Departament Obrony USA w 'Trusted Computer System Evaluation Criteria' (TCSEC).

Itd. P1003.1f, P1003.1g, P1003.1h, P1003.2, P1003.2b, P1003.2c, P1003.2d, P1003.3, P1003.5, P1387, P1003.9, P1003.10, P1003.11, P1003.13, P1003.14, P1003.16, P1224.2, POSIX.18, POSIX.19, POSIX.20, POSIX.21, POSIX.0

4



Standard POSIX – P1003.1b



P1003.1b

Rozszerzenia dotyczące systemów czasu rzeczywistego:

- semafory,
- proces blokowania pamięci,
- pliki mapowane w pamięci i współdzielona pamięć,
- kolejkowanie priorytetowe,
- obsługa sygnałów w czasie rzeczywistym,
- liczniki czasu,
- komunikacja międzyprocesowa
- synchroniczne operacje wejścia/wyjścia (I/O),
- asynchroniczne operacje wejścia/wyjścia (I/O).

(więcej o tych rozszerzeniach w dalszej części wykładu...)

5



Standard POSIX – dalsze rozszerzenia



POSIX wprowadza:

- funkcje wspierające wielowątkowość w ramach jednego procesu,
- funkcje kontrolujące wątki i ich atrybuty,
- muteksy (specjalnego rodzaju semafory binarne),
- zmienne warunkowe,
- funkcje umożliwiające monitorowanie czasu wykonywania procesów i wątków oraz określające limity czasów oczekiwania dla funkcji blokujących,
- normy dotyczące rozproszonej komunikacji czasu rzeczywistego, obsługi urządzeń z buforami, wysyłania bloków kontrolnych,
- usługi dla niezawodności, dostępności oraz użyteczności.

(więcej o tych rozszerzeniach w dalszej części wykładu...)

6



Standard POSIX – podsumowanie



POSIX daje pewną dowolność twórcom systemów czasu rzeczywistego, **nie muszą** oni implementować całości, a tylko wybrane przez siebie funkcje opisane w standardzie.

UWAGA:

Wymagane jest jasne określenie co jest, a czego nie ma w systemie – odnośnie implementowanych funkcji ze standardu POSIX.

W praktyce, zgodność systemu ze standardem 1003.1b gwarantuje poprawne wykonywanie zadań czasu rzeczywistego o charakterze miękkim (Soft RTOS).

7



Ogólna Klasyfikacja Systemów Operacyjnych



Systemy Operacyjne sterowane zdarzeniami:

- ryzyko „przeciążenia” systemu w wyniku napływania dużej liczby danych i konieczności ich przetworzenia.

Systemy Operacyjne sterowane czasem:

- działają „w rytm” zegara czasu rzeczywistego,
- nie są podatne na „przeciążenia” wynikające z potrzeby przetworzenia dużej liczby danych.

Protokół zarządzania obciążeniem – przydzielanie zadań i zasobów do wykonujących je jednostek tak aby w pełni wykorzystać możliwości systemu.

8



Przykłady zastosowań RTOS



Sterowanie: elektrownie, fabryki chemiczne, linie technologiczne (np. przygotowywanie żywności), systemy kontroli ruchu lotniczego, roboty, sterowanie silnikami odrzutowymi, sterowanie silnikami samochodowymi, ABS, ...

Urządzenia telekomunikacyjne i sieciowe: routery...

Urządzenia biurowe: fax'y, kopiarki ...

Peryferia komputerowe: drukarki, terminale, skanery, modemy ...

Militaria: systemy uzbrojenia (ofensywne i defensywne)...

Medycyna: aparaty USG, tomografy komputerowe ...

Sprzęt AGD: kuchenki mikrofalowe, zmywarki, pralki, odtwarzacze DVD, sterowanie ogrzewaniem, klimatyzacja ...

9



Czym jest system czasu rzeczywistego ? - definicje cz. I -



- System czasu rzeczywistego to taki, w którym wynik przetwarzania nie zależy tylko i wyłącznie od jego logicznej poprawności, ale również od czasu, w jakim został osiągnięty. Jeśli nie są spełnione ograniczenia czasowe, mówi się, że nastąpił błąd systemu.
- Tryb przetwarzania w czasie rzeczywistym jest takim trybem, w którym programy przetwarzające dane napływające z zewnątrz są zawsze gotowe, a wynik ich działania jest dostępny nie później niż po zadanym czasie. Momentadejścia kolejnych danych może być losowy (asynchroniczny) lub ścisłe określony (synchroniczny).
- System czasu rzeczywistego jest systemem interaktywnym, który utrzymuje ciągły związek z asynchronicznym środowiskiem, np. środowiskiem, które zmienia się bez względu na system, w sposób niezależny.

10



Czym jest system czasu rzeczywistego ?

- definicje cz. II -



- Oprogramowanie czasu rzeczywistego odnosi się do systemu lub trybu działania, w którym przetwarzanie jest przeprowadzane na bieżąco, w czasie wystąpienia zewnętrznego zdarzenia, w celu użycia rezultatów przetwarzania do kontrolowania lub monitorowania zewnętrznego procesu.
- System mikrokomputerowy działa w czasie rzeczywistym, jeżeli wypracowane przez ten system decyzje są realizowane w tempie obsługiwanej procesu. Inaczej mówiąc, system działa w czasie rzeczywistym, jeżeli czas reakcji systemu jest niezauważalny przez proces (decyzja jest wypracowana we właściwym czasie).
- **System czasu rzeczywistego odpowiada w sposób przewidywalny (w określonym czasie) na bodźce zewnętrzne napływające w sposób nieprzewidywalny.**

11



Analiza przykładowych RTOS



- Odtwarzacz DVD
- System naprowadzania rakiet przeciwlotniczych

Zależnie od roli i przeznaczenia danej grupy aplikacji czasu rzeczywistego, ograniczenia czasowe są koniecznością, której niespełnienie prowadzi w najgorszym przypadku do nieodwracalnych i tragicznych skutków oraz tych, w których czas wykonania nie jest tak krytyczny i dopuszcza się pewne odstępstwa.

**Określenie "czasu rzeczywistego,,
jest pojęciem „względnym”**

12



Klasyfikacja RTOS - ze względu na podejście do czasu -



1) **Hard RTOS** (ang. *hard real-time systems*)

Systemy o ostrych ograniczeniach czasowych.

Zadania **muszą** zakończyć się **prawidłowo** i w **określonym czasie**.

2) **Soft RTOS** (ang. *soft real-time systems*)

Systemy o miękkich lub łagodnych ograniczeniach czasowych.

Zadania wykonywane są **tak szybko jak to możliwe** ale **nie muszą** zakończyć się w określonym czasie.

3) **Nieoficjalna klasa RTOS, Firm RTOS** (ang. *firm real-time systems*)

Systemy sprzętowo-programowe o mocnych ograniczeniach czasowych, systemy zaliczane do tej klasy posiadają cech systemów Hard RTOS i Soft RTOS.

13



Hard RTOS



- Zadania **muszą** zakończyć się **prawidłowo** i w **określonym czasie**.
- Przekroczenie czasu wykonania zadania powoduje poważne, a nawet katastrofalne skutki, jak np. zagrożenie życia lub zdrowia ludzi, uszkodzenie lub zniszczenie urządzeń, przy czym nie jest istotna wielkość przekroczenia terminu a jedynie sam fakt jego przekroczenia.

14



Hard RTOS (cd.)



- W Hard RTOS wymagana jest implementacja odpowiednich mechanizmów:
 - ograniczających wszystkie opóźnienia w systemie (zaczynając od odzyskiwania przechowywanych danych, a kończąc na czasie zużywanym przez system na wypełnienie dowolnego zamówienia),
 - mała ilość szybkiej pamięci operacyjnej w której przechowywanych jest niewiele zadań,
- Systemy Hard RTOS nie mają również większości cech nowoczesnych systemów operacyjnych, które oddalają użytkownika od sprzętu, zwiększając niepewność odnośnie do ilości czasu zużywanego przez operacje, np. prawie nie spotyka się w systemach czasu rzeczywistego pamięci wirtualnej.

15



Soft RTOS



- Zadania wykonywane są **tak szybko jak to możliwe ale nie muszą** zakończyć się w określonym czasie.
- Przekroczenie pewnego czasu powoduje negatywne skutki tym poważniejsze, im bardziej ten czas został przekroczony.
W systemach Soft RTOS krytyczne zadanie do obsługi w czasie rzeczywistym otrzymuje pierwszeństwo przed innymi zadaniami i zachowuje je aż do swojego zakończenia.

16



Soft RTOS (cd.)



- Podobnie jak w Hard RTOS opóźnienia muszą być ograniczone - zadanie czasu rzeczywistego nie może „w nieskończoność” czekać na usługi jądra.
- Zaletą łagodnego traktowanie wymagań dotyczących czasu rzeczywistego w Soft RTOS jest możliwość ich łatwego łączenia z systemami innych rodzajów.
- Użyteczność Soft RTOS jest bardziej ograniczona niż systemów rygorystycznych, ponieważ nie zapewniają one „nieprzekraczalnych terminów” realizacji zadań, co stanowi iż ich zastosowanie w przemyśle jest ryzykowne.
- Systemy Soft RTOS znajdują swoje miejsce wszędzie tam, gdzie istnieje potrzeba systemów o bardziej rozbudowanych możliwościach np. techniki multimedialne, czy zaawansowane projekty badawcze w rodzaju eksploracji podmorskich lub eksploracji kosmosu.

17



Firm RTOS



- Systemy sprzętowo-programowe o mocnych ograniczeniach czasowych. systemy zaliczane do tej klasy posiadają cech systemów Hard RTOS i Soft RTOS,
- Fakt przekroczenia terminu „realizacji zadania” powoduje całkowitą nieprzydatność wypracowanego przez system wyniku (nie ma żadnej korzyści), jednakże nie oznacza to zagrożenia dla ludzi lub sprzętu (nie ma żadnej groźby).

18



Inne kryteria klasyfikacji RTOS



Klasyfikacja RTOS ze względu na sposób tworzenia aplikacji dedykowanych:

- uruchamianie i wykonywanie dedykowanych aplikacji bez możliwości interakcji z systemem operacyjnym,
- embedded RTOS (*wbudowane RTOS*), e-RTOS, wykorzystują platformę sprzętowo-programową (może zawierać oprogramowanie dedykowane wyłącznie dla wykorzystanego urządzenia (*firmware*), lub może to być system operacyjny wraz ze specjalizowanym oprogramowaniem).

Klasyfikacja RTOS ze względu na przeznaczenie do pracy:

- z jednym procesorem,
- z wieloma procesorami,
- rozproszonej (platforma złożona z wielu jednostek obliczeniowych, połączonych ze sobą przez sieć).

19



Klasyfikacja RTOS



Systemy
Wbudowane
(*embedded*)

Systemy
Wbudowane
Czasu
Rzeczywistego
(*embedded*
RTOS)

Systemy
Czasu
Rzeczywistego
(RTOS)

20



RTOS - podsumowanie



System czasu rzeczywistego odpowiada w sposób przewidywalny, deterministyczny (w określonym czasie), na bodźce zewnętrzne napływające w sposób nieprzewidywalny, niedeterministyczny.

**W przypadku systemów RTOS
„real time” nie oznacza „szybki”
lecz „przewidywalny”**

21



Podstawowe różnice pomiędzy RTOS a „zwykłym” OS



Powszechnie wykorzystywane „zwykłe” OS:

- stosują liberalna politykę podziału zasobów,
- priorytety różnych zadań mogą być w tych systemach dynamicznie zmieniane w sposób nieznany użytkownikowi,
- procesy czasu rzeczywistego nie są specjalnie uprzywilejowane, nie są też zachowywane ich priorytety,
- nie można przewidzieć, który z procesów zostanie wykonany w jakim czasie, ani w jakiej kolejności będą one przetwarzane
- faworyzują użytkowników terminalowych i programy interaktywne (np. edytor tekstu),
- procesy wykonywane w trybie systemowym nie mogą zostać wywolaszczone.

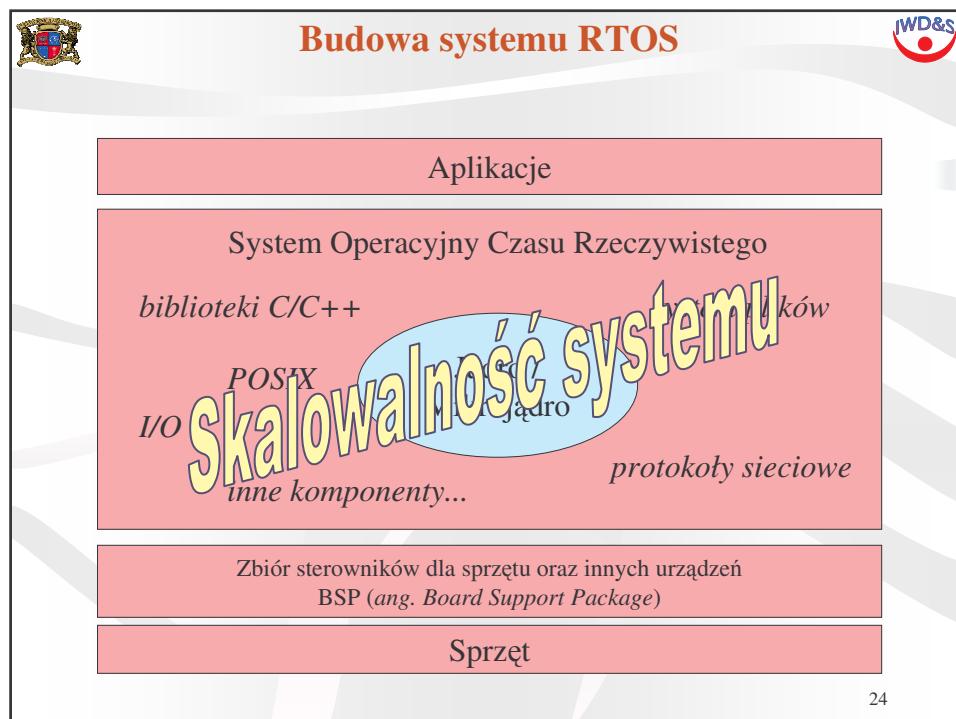
22

Podstawowe różnice pomiędzy RTOS a „zwykłym” OS

W systemach RTOS:

- priorytety przydzielane są do konkretnych zadań i w oparciu o nie przydzielane są zasoby,
- zarządzanie zasobami jest o wiele bardziej restrykcyjne (niż w przypadku „zwykłego” OS) co powoduje skuteczne eliminowanie opóźnień,
- rezygnuje się z pamięci wirtualnej (inaczej niż w przypadku „zwykłego” OS),
- mogą wystąpić niebezpieczne sytuacje (opóźnienia) wynikające z braku odpowiedniego rejestru pamięci (zasobu) w chwili gdy uruchomione programy będą potrzebowaly więcej zasobów niż jest dostępnych w systemie.

23





Podstawowe pojęcia - Zasób



Zasób

Zasób jest elementem systemu wykorzystywanym przez zadania. Zasobem może być urządzenie I/O (klawiatura, wyświetlacz) pamięć (zmienna, stała, macierz itp.), rejesty, interfejsy, czas procesora, stos.

Zasób współdzielony

Zasób, który może być wykorzystywany przez więcej niż jedno zadanie. Każde zadanie powinno uzyskiwać wyłączny dostęp do współdzielonego zasobu, aby uniknąć przekłamania danych - technika **wzajemnego wykluczania** (ang. *Mutual Exclusion*).

25



Podstawowe pojęcia - Zadanie



Zadanie

Zadanie (wątek) jest „małym” niezależnym programem, który sądzi „że ma procesor na swoją wyłączność”.

Poszczególne zadania konkurują między sobą o czas procesora.

Każde uruchomione w systemie zadanie zawiera szereg charakterystycznych dla niego parametrów, takich jak:

- unikalny numer,
- priorytet,
- kontekst (stan zestawu rejestrów procesora),
- obszar stosu,
- kod programu.

26

 **Podstawowe pojęcia – Zadanie (cd.)** 

Podstawowe stany zadania:

- gotowe (zadanie gotowe do wykonania),
- wykonywane,
- zablokowane.

Diagram przedstawiający zmiany stanów zadania:

```
graph TD; Start((Start)) --> Gotowe[GOTOWE]; Gotowe -- "Zostało odblokowane, nie ma najwyższego priorytetu" --> Zablokowane[Zablokowane]; Gotowe -- "Zostało wywłaszczone" --> Wykonywane[Wykonywane]; Zablokowane -- "Zostało zablokowane" --> Zablokowane; Wykonywane -- "Zadanie zostało odblokowane i ma najwyższy priorytet" --> Gotowe;
```

Opis zmian stanów zadania:

- Przejście z **Start** do **GOTOWE** (zaznaczane przez żółty wskazówkę).
- Przejście z **GOTOWE** do **Zablokowane** (zaznaczane przez żółty wskazówkę): *Zostało odblokowane, nie ma najwyższego priorytetu*.
- Przejście z **GOTOWE** do **Wykonywane** (zaznaczane przez żółty wskazówkę): *Zostało wywłaszczone*.
- Przejście z **Zablokowane** do **Zablokowane** (zaznaczane przez żółty wskazówkę): *Zostało zablokowane*.
- Przejście z **Wykonywane** do **GOTOWE** (zaznaczane przez żółty wskazówkę): *Zadanie zostało odblokowane i ma najwyższy priorytet*.

 **Podstawowe pojęcia – Przełączanie kontekstu** 

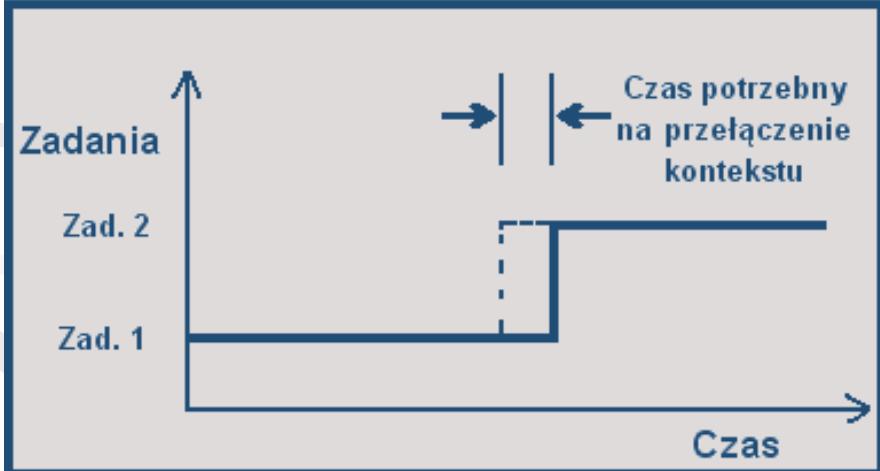
Przełączanie kontekstu (zadania)

Kiedy jądro systemu decyduje o uruchomieniu innego zadania, kontekst obecnego zadania (rejestry CPU) zostaje zapisany w przeznaczonym dla danego zadania obszarze pamięci. Następnie kontekst nowego zadania zostaje odczytany z odpowiedniego obszaru i zostaje wznowione wykonanie nowego zadania.

Proces ten nosi nazwę przełączania kontekstu lub zadania.

Przełączanie zadań stanowi martwy czas programu.

 Podstawowe pojęcia - Przełączanie kontekstu (cd.) 



29

 Podstawowe pojęcia – Priorytety 

Priorytety Statyczne

Priorytety zadań są priorytetami **statycznymi**, wtedy gdy priorytet każdego z zadań **nie zmienia** się podczas wykonania aplikacji. Każdy zapis ma więc przydzielony ustalony priorytet (na podstawie znanych wymagań i ograniczeń czasowych zadania) podczas komplikacji.

Priorytety Dynamiczne

Priorytety zadań są priorytetami **dynamicznymi** jeśli priorytety zadań mogą być zmienianie podczas wykonania, uruchamiania aplikacji.

Priorytety dynamiczne to pożądana właściwość systemów RTOS dla uniknięcia *inwersji priorytetów* (...o tym dalej).

30



Jądro systemu



- Jądro jest częścią systemu wielozadaniowego odpowiedzialną za zarządzanie zadaniami (a więc zarządzanie czasem procesora) oraz komunikację pomiędzy zadaniami.
- Podstawową funkcją dostarczaną przez jądro jest przełączanie kontekstu/zadania.
- Wykorzystanie jądra czasu rzeczywistego upraszcza projektowanie systemów dzięki umożliwieniu podziału problemu na wiele zadań zarządzanych przez jądro (jednocześnie lub współbieżnie).

31



Szeregowanie zadań – Scheduler / Dispatcher



- Integralną i najważniejszą częścią jądra systemu RTOS jest program szeregujący – scheduler.
- Scheduler jest to zestaw algorytmów określających jakie zadanie będzie wykonywane jako następne oraz kiedy to nastąpi.
- Większość systemów RTOS oparta jest na priorytetach – każde zadanie ma przyporządkowany priorytet zależny od ważności zadania.
- W takim przypadku czas CPU zostaje przydzielony do zadania, które jest gotowe i ma najwyższy priorytet.

32



Typy jąder opartych na priorytetach



Typy jader systemu RTOS oparte na priorytetach:

- jądro **bez wywłaszczenia** (ang. *non-preemptive*)
- jądro **z wywłaszczeniem** (ang. *preemptive*)

33



Jądro bez wywłaszczenia



W tym typie jądra wymagane jest aby każde zadanie samodzielnie oddało kontrolę nad procesorem.

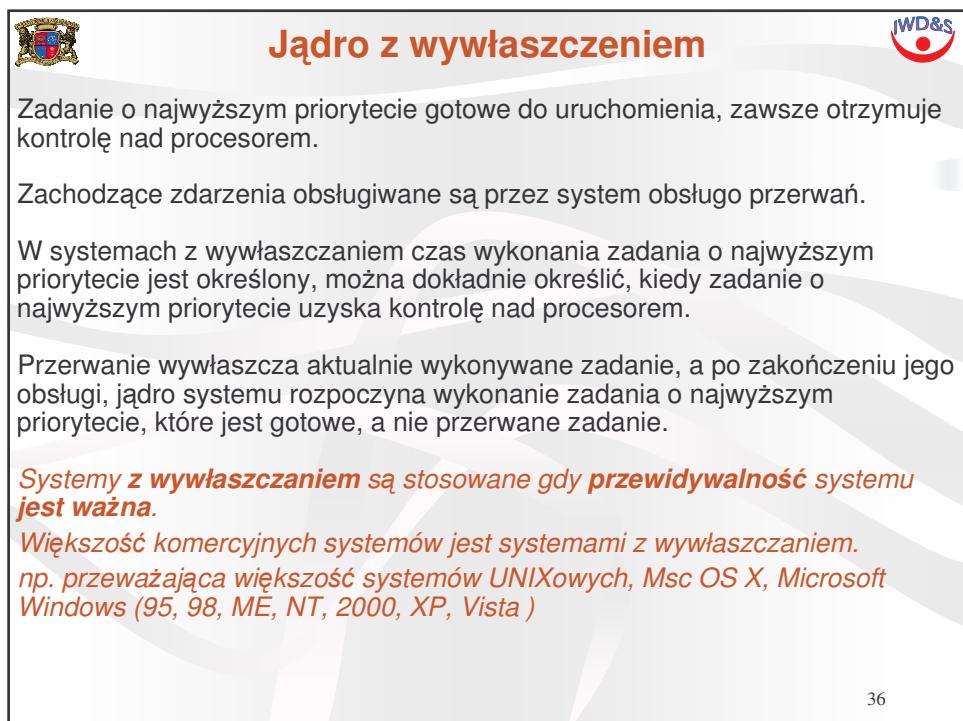
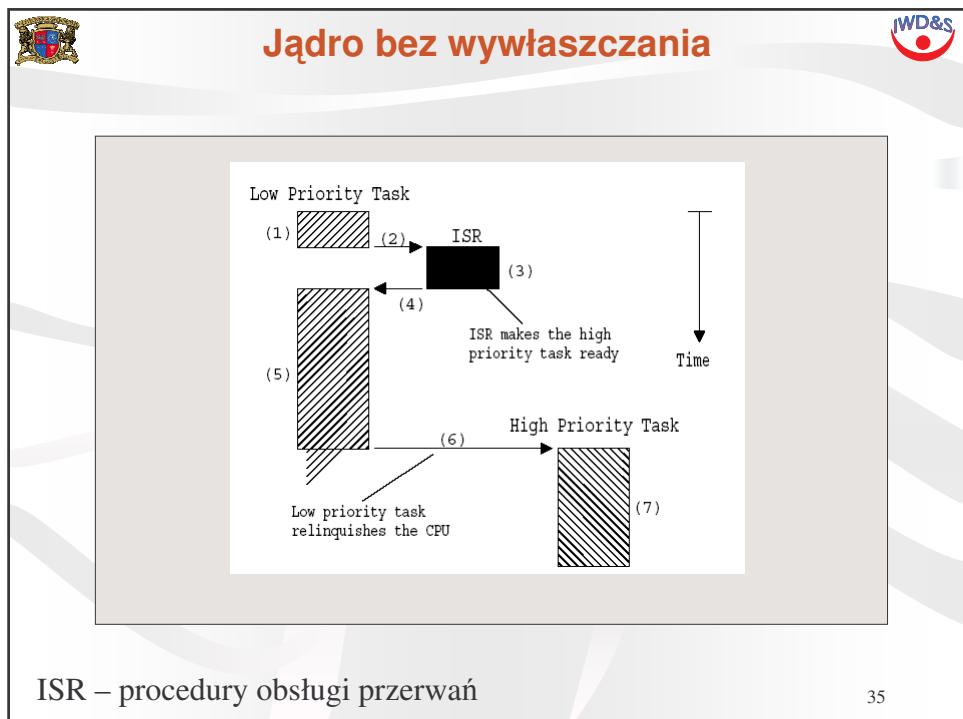
Zachodzące zdarzenia obsługiwane są przez system obsługi przerwań.

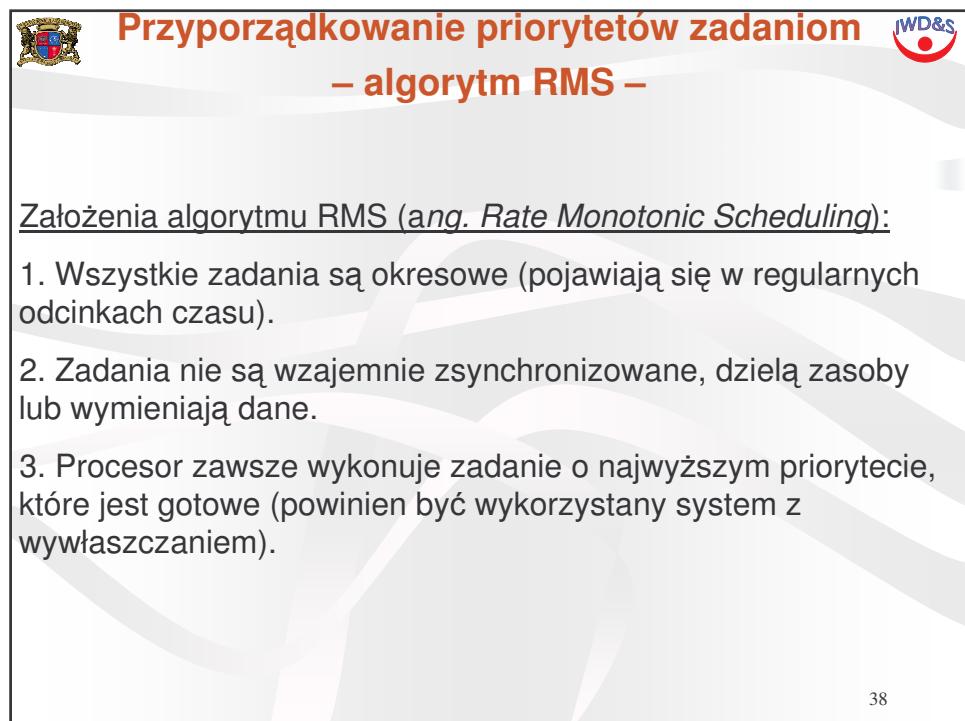
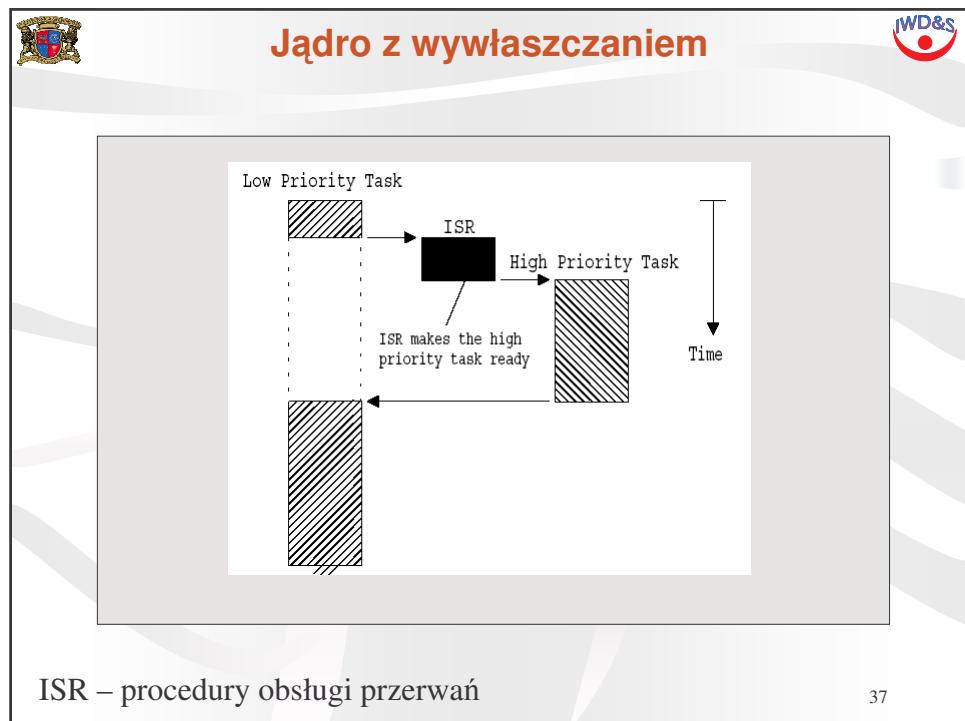
Procedura obsługi przerwania może ustawić zadanie o wyższym priorytecie jako gotowe, ale i tak następuje powrót do aktualnego zadania.

Zadanie o wyższym priorytecie uzyska kontrolę nad procesorem, gdy zadanie o niższym priorytecie ją odda.

*Niewiele systemów komercyjnych jest wykonywanych bez mechanizmu wywłaszczenia.
np. Microsoft Windows 3.11, Mac OS 9 (i wcześniejsze).*

34







Przyporządkowanie priorytetów zadaniom – algorytm RMS – (cd.)



Podstawowe twierdzenie RMS:

wszystkie limity czasowe systemu Hard RTOS będą zawsze spełnione jeżeli:

$$\sum_i \frac{E_i}{T_i} \leq n \cdot \left(2^{\frac{1}{n}} - 1 \right)$$

gdzie:

n – dany zestaw zadań z przypisanymi priorytetami RMS,

E_i – maksymalny czas wykonania zadania i ,

T_i – okres wykonania zadania i ,

E_i/T_i – część czasu procesora wymagana do wykonania zadania i .

Zgodnie ze statycznym doborem priorytetów algorytmem RMS, im krótszy okres zadania, tym wyższy jego priorytet.

39



Przyporządkowanie priorytetów zadaniom – algorytm RMS – (cd.)



Liczba Zadań	$n \cdot \left(2^{\frac{1}{n}} - 1 \right)$
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
...	...
∞	0.693

**Aby spełnić wszystkie limity czasowe systemu HARD opartego o RMS,
wykorzystanie CPU przez wszystkie zadania krytyczne czasowo
powinno być mniejsze niż 70 procent !!!**

**Nadal można mieć zadania niekrytyczne czasowo, a więc wykorzystać
100 procent czasu CPU.**

40



Problemy związane z szeregowaniem zadań



W systemach wielozadaniowych, prawie zawsze istnieje pewien podzbiór procesów uruchomionych w danej chwili, które oddziałują na siebie.

Źle zorganizowany współbieżny dostęp do tych samych zasobów, prowadzi do powstawania niespójności danych, w efekcie którego efektywność systemu może być dalece niezadowalająca.

- zjawisko zakleszczenia (*ang. impasse lub deadlock*)
- zjawisko inwersji priorytetów (*ang. priority inversion*)

41

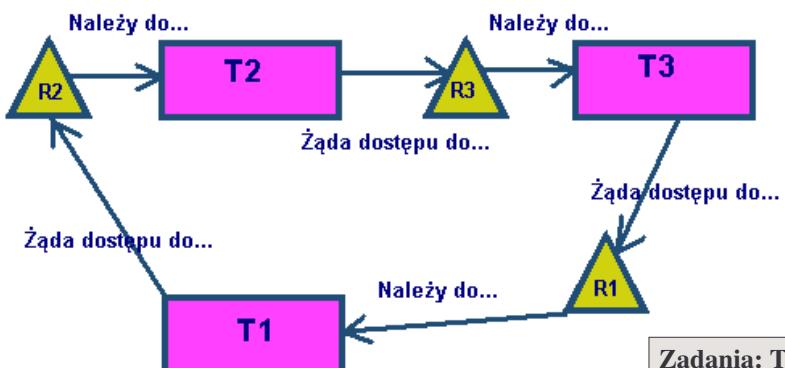


Zakleszczenie procesów



Zbiór procesów znajduje się w stanie blokady (zakleszczenia), jeśli każdy z nich jest wstrzymany w oczekiwaniu na zdarzenie, które może być wywołane przez jakiś inny proces z tego zbioru.

(*np. klasyczna sytuacja cyklicznego oczekiwania*).





Zakleszczenie procesów (cd.)



Algorytmy wykrywające zakleszczenie oparte są głównie o teorię grafów.

W trakcie projektowania aplikacji należy zadbać o to by zadania sprawdzały czy to czego oczekują jest w danej chwili dostępne, a kiedy już nie korzystają z jakiś zasobów to powinny je zwolnić.

Można również zasoby zorganizować w hierarchiczną strukturę i zaimplementować prosty model ich przydziału (protokół), np.:

zadanie które uzyska dostęp do jakiegoś zasobu, w kolejnych krokach albo musi ten zasób zwolnić lub starać się tylko i wyłącznie o zasoby znajdujące się wyżej w hierarchii

43



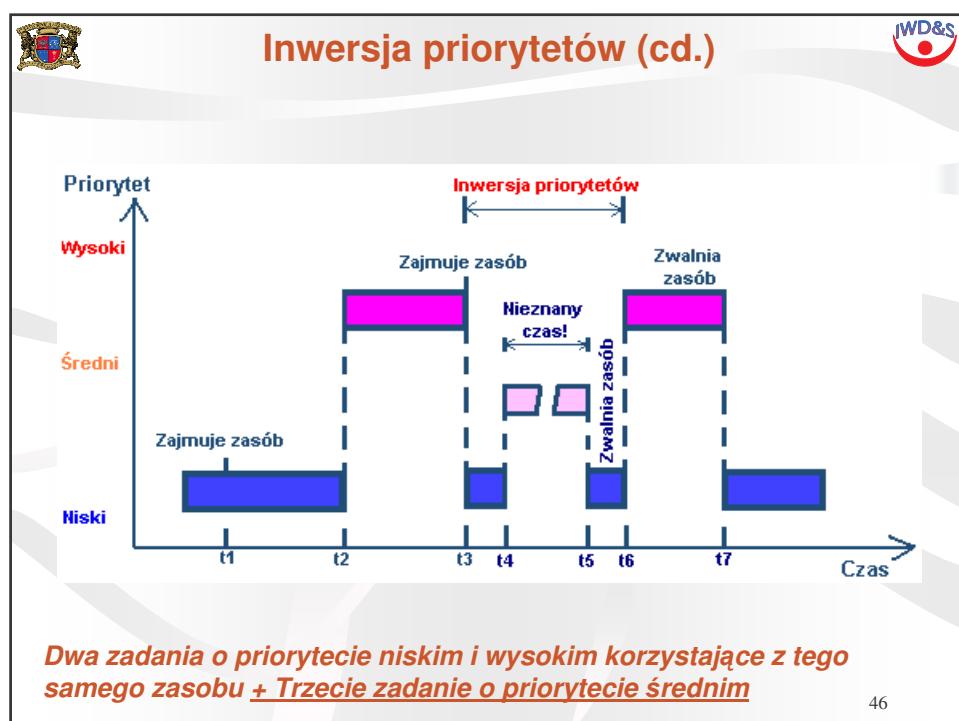
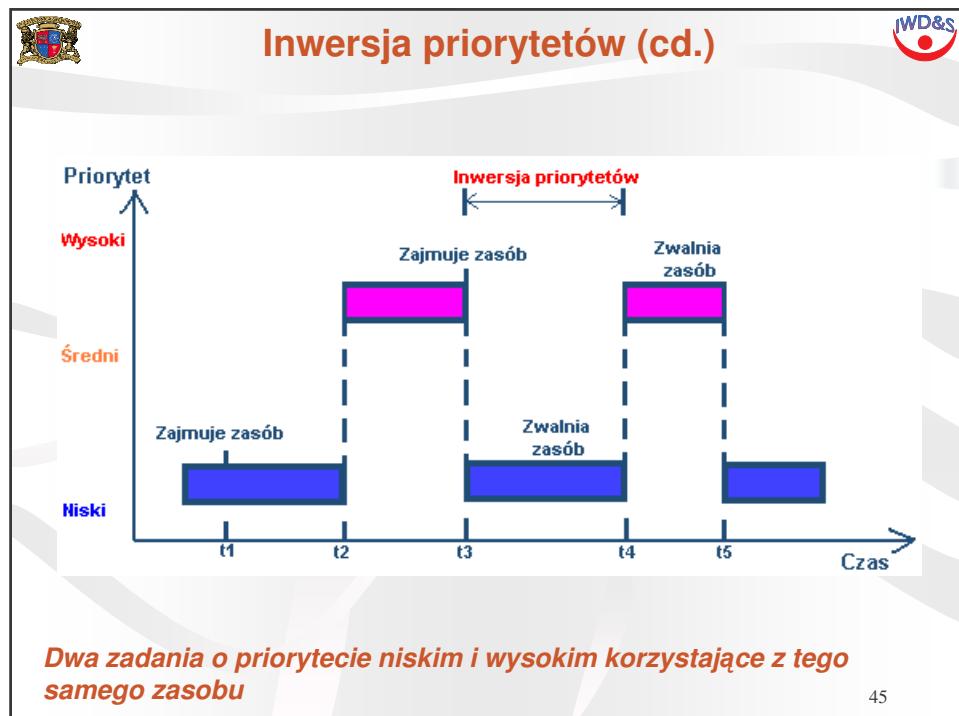
Inwersja priorytetów



Inwersja priorytetów występuje w przypadkach gdy zadania o różnych priorytetach wykorzystują te same zasoby (zasoby współdzielone).

Ogólnie zjawisko inwersji priorytetów polega na tym, że zadanie o niskim priorytecie, zajmując zasób dzielony, zmusza zadanie o wyższym priorytecie do przejścia w stan „blokowane” i oczekiwania na zwolnienie tego zasobu.

44





Inwersja priorytetów (cd.)



Inwersja priorytetów prowadzi do wielu niebezpieczeństw, które w najgorszym wypadku mogą prowadzić do powstania w systemie „sporych” anomalii czasowych.

Całkowite wyeliminowanie zjawiska inwersji priorytetów jest praktycznie niemożliwe.

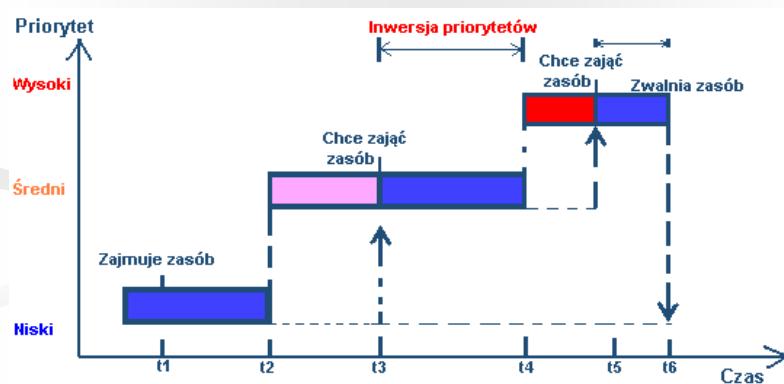
Metody eliminacji tego zjawiska wykorzystują różne modele sterowania dostępem do zasobów (protokoły), np.:

- protokół wykorzystujący dziedziczenie priorytetów (może powodować zakleszczenia),
- protokół pułapów priorytetu (całkowita eliminacja wystąpienia blokad),
- różne modyfikacje powyższych.

47



Inwersja priorytetów – rozwiązywanie przez dziedziczenie priorytetów –



Zasada jego działania polega na podnoszeniu priorytetu zadania, które zajmuje zasób dzielony, do poziomu priorytetu tego zadania, które zgłasza żądanie dostępu do tego zasobu.

48

**Inwersja priorytetów
– rozwiązywanie przez pułap priorytetów –**

Każdy zasób ma przypisany tzw. pułap priorytetów.

Jego wartość jest określana przez najwyższy priorytet spośród zadań, które mogą o niego konkurować.

Gdy zadanie zarygluje dostęp do takiego zasobu, jego priorytet jest zwiększany do wartości pułapu tego zasobu.

Po zwolnieniu zasobu, zadanie ma przywracany priorytet do wartości początkowej.

49

Semafony (binarny, ogólny)

To jedne z obiektów umożliwiających wzajemne wykluczanie oraz synchronizację zadań.

Semafor, jest obiektem jądra, który może zostać zajęty przez jeden lub kilka wątków w celu sterowania dostępem do np. jakiegoś wspólnego zasobu.

Operacje na semaforach, umożliwiające wstrzymanie i wznowienie procesów:

- czekaj: to operacja opuszczająca semafor, zmniejsza jego wartość o jeden (semafor może mieć wartości nieujemne),
- sygnalizuj: jest operacją zwiększającą wartość semafora o jeden, logicznie odpowiada operacji podniesienia semafora

Semafony są globalne : dowolny proces może je opuszczać albo podnosić

50



Muteksy



Muteksy (ang. *mutual exclusion semaphores*), stanowią szczególny rodzaj semaforów binarnych.

Muteks może być zablokowany (ma wartość 1) lub odblokowany (ma wartość 0).

Jedną z cech muteksów jest zasada posiadania.

Zasada posiadania polega na tym, że jeśli jakieś zadanie zablokuje muteks (nada mu wartość 1), to tylko ono może ten muteks odblokować (nadać mu wartość 0).

Zapobiega to sytuacji, w której jedno z zadań wykona operację *czekaj* aby synchronizować dostęp do jakiegoś zasobu, a później inne zadanie niezwiązane logicznie z tym zasobem, podniesie semafor (potencjalny problem niespójności danych przy wykorzystywaniu jedynie semaforów)

51



Zmienne warunkowe



Zmienne warunkowe również służą do synchronizacji zadań.

Zmienne warunkowe pozwalają wielokrotnie wstrzymywać wykonanie jakiegoś zadania, aż żądany warunek zostanie spełniony.

52

Komunikacja między procesami – kolejki komunikatów –

Kolejki komunikatów, to inteligentne bufory, które zazwyczaj działają w oparciu o algorytm „pierwszy przyszedł, pierwszy został obsłużony” (FIFO).

Proces przesyłania komunikatu przy pomocy kolejki:

Rozmiar kolejki, to ilość komunikatów, jakie może ona pomieścić.

53

Komunikacja między procesami – kolejki komunikatów – (cd.)

Dwukierunkowa komunikacja z użyciem kolejek:

54

**Komunikacja między procesami
– kolejki komunikatów – (cd.)**

Rozsyłanie wiadomości z jednego zadania do wielu:

The diagram illustrates a broadcast communication pattern using a message queue. A process labeled "ZAD" (Zadanie) is shown sending a message to a "Kolejka" (Queue). The queue is represented as a rectangular container divided into five vertical slots. From the right side of the queue, three arrows point to three separate processes, each also labeled "ZAD". These three receiving processes are collectively labeled "Odbiorcy" (Recipients).

ZAD → Kolejka → ZAD → ZAD → Odbiorcy

55

**Komunikacja między procesami
– potoki –**

Potoki (ang. *pipes*), służą do przechowywania danych jako strumień pozbawiony jakiekolwiek struktury.

Potoki są odczytywane według algorytmu „pierwszy przyszedł, pierwszy zostanie obsłużony” (FIFO).

The diagram illustrates a pipe communication pattern. It shows two processes, "ZAD" (Zadanie piszące) and "ZAD" (Zadanie czytające), connected to a "Potok" (Pipe). The pipe is depicted as a horizontal cylinder containing a blue rectangular area labeled "Dane przechowywane w potoku" (Data stored in the pipe). Two "Deskryptor" (Descriptor) symbols, which look like small circles with arrows, point to the pipe from both ends. These descriptors indicate the starting and ending points for reading and writing data from the pipe.

ZAD → Potok → ZAD

Dane przechowywane w potoku

56

Komunikacja między procesami
– rejesty zdarzeń –

The diagram shows three tasks (ZAD. 1, ZAD. 2, ZAD. 3) interacting with a shared event register. Task ZAD. 1 and ZAD. 2 both point to the same event (Zdarzenie). Task ZAD. 3 points to a decision diamond labeled AND / OR. The output of this diamond also points to the event. The event triggers an update to the event register (Rejestr zdarzeń), which is represented as a sequence of bits: 0 0 1 0 0 1 0 0 0 0 0 0. The bit at index 2 is highlighted in purple, indicating an event has occurred.

Bardzo często występuje konieczność, aby zadanie miało możliwość śledzenia wystąpienia jakiś konkretnych zdarzeń oraz podjęcia odpowiednich kroków w celu wypracowania jakiejś odpowiedzi na nie. Obiektami umożliwiającymi takie zachowanie są właśnie rejesty zdarzeń - obiekty, które składają się z szeregu bitów, interpretowanych jako flagi.

57

Komunikacja między procesami
– sygnały –

The diagram illustrates signal generation and handling. Two tasks (Zad. 1 and Zad. 2) generate a signal (Sygnal). This signal triggers a procedure in Task 3 (run1(), odbierz_sygn(); ...). Task 3 also contains a vector table (Tablica wektorów) with address 1 0 0 0 0. A separate procedure (Procedura obsługi sygnału) handles the signal with code: sgn_run(), ..., sgn_return(); .

Sygnał, to przerwanie programowe, generowane w odpowiedzi na zaistnienie jakiegoś zdarzenia. Przerywa normalny tok wykonywania u jego odbiorcy oraz wymusza wykonanie jakiegoś określonego zadania.

58



Wyjątki



Wyjątek (ang. exception) – jest to dowolne zdarzenie, które przerywa normalny tok obliczeń procesora i wymusza wykonanie określonego zbioru instrukcji w trybie uprzywilejowanym. Najogólniej można je podzielić na dwie grupy: synchroniczne i asynchroniczne.

Synchroniczne są generowane przez tzw. zdarzenia wewnętrzne jak np. efekt wykonania jakiejś instrukcji procesora. Przykładami mogą być dzielenie przez zero lub niepoprawny odczyt z pamięci.

Asynchroniczne (przerwania) są generowane przez tzw. zdarzenia zewnętrzne.

59



Przerwania



Przerwanie (ang. interrupt, external interrupt) – to tzw. wyjątki asynchroniczne i nie są powiązane z instrukcjami wykonywanymi przez procesor. Ich źródłem są wszelkie zdarzenia zewnętrzne, czyli odnoszą się do różnych sygnałów generowanych przez sprzęt. Przykładami mogą być wciśnięcie przycisku *reset* na płycie głównej lub sygnał urządzenia komunikacyjnego, które właśnie otrzymało pakiet z danymi.

Mожно je podzielić na maskowalne, czyli takie, które można wyłączyć programowo oraz niemaskowalne, których nie da się zablokować.

Przerwania niemaskowalne, są zazwyczaj połączone z procesorem przy pomocy specjalnego kanału komunikacyjnego i są obsługiwane natychmiast po ich wystąpieniu.

60



Czas



Aby poprawnie mogły działać proces szeregujący zadania oraz same zadania czasu rzeczywistego, bardzo ważne jest precyzyjne odmierzanie czasu.

Większość systemów wbudowanych dostarcza dwa rodzaje mechanizmów odmierzających czas (*ang. timer*):

- timery oparte o rozwiązania sprzętowe (programowalne kontrolery czasu),
- typowe rozwiązania programowe.

61



Popularne systemy RTOS – QNX Neutrino –



QNX, to zdaniem wielu (np. AMD, IBM, Cisco Systems) najlepszy i jednocześnie najbardziej zaawansowany oraz przyszłościowy, rygorystyczny (Hard RTOS) system operacyjny czasu rzeczywistego.

Jest pierwszym w historii systemem wielozadaniowym i wielodostępnym przeznaczonym dla mikrokomputerów IBM PC.

Wykorzystuje architekturę mikrojądra, które od wersji 6.0 systemu zajmuje 8kB (jądro systemu UNIX to co najmniej 700kB).

QNX ma strukturę modułową oraz architekturę opartą o przesyłanie komunikatów (model klient – serwer).

Wysoka posunięta modularność i skalowalność systemu.

Komunikacja pomiędzy procesami znajdującymi się na odległych węzłach sieci jest tak samo prosta, jak w obrębie jednego komputera.

62



Popularne systemy RTOS – QNX Neutrino – (cd.)



QNX daje możliwość zdeterminowania czasu reakcji na zdarzenia występujące w systemie.

QNX dzięki rozbudowanym możliwościom definiowania priorytetów, jest stosowany jako system służący do sterowania automatyką przemysłową, gdzie pewne zdarzenia są krytyczne (np. otwarcie zaworu bezpieczeństwa w zbiorniku kiedy gwałtownie wzrasta ciśnienie) i muszą być zawsze obsłużone na czas.

Na bazie QNX opracowywane są również systemy SCADA.

QNX jest również wykorzystywany jako platforma dla baz danych.

63



Popularne systemy RTOS – RTLinux –



RTLinux, to rygorystyczny (Hard RTOS) system czasu rzeczywistego.

Występuje w dwóch wersjach, komercyjnej RT Linux PRO i w ogólnodostępnej GPL – RTLinux (darmowy i udostępniany wraz z całym kodem źródłowym).

Jego cechą charakterystyczną jest to, że współistnieją w nim: jądro czasu rzeczywistego RTCore i jądro Linuksa.

RTLinux „szybko” potrafi obsługiwać przerwania w warunkach dużego obciążenia obliczeniami, przy jednoczesnych znikomych operacjach dyskowych.

Architektura RTLinux wymusza jednak pewien styl programowania (podział na dwie części: obliczeniową i operacje dyskowe, sieciowe).

64



Popularne systemy RTOS



– Windows CE –

Jedyny systemem czasu rzeczywistego (rygorystyczny – Hard RTOS), jaki stworzył gigant z Redmont.

Posiada architekturę modułową.

Zoptymalizowany dla urządzeń o niewielkiej ilości pamięci – jądro systemu wymaga do uruchomienia około 1MB RAM.

Microsoft opracował dedykowaną wersję systemu do układów elektronicznych urządzeń instalowanych w samochodach (na razie zarządza sprzętem audio auta)

Szerokie pole do popisu wszędzie tam, gdzie konieczne jest wykorzystanie wszelkich najnowszych technologii, a zwłaszcza tych związanych z multimediami. Można tworzyć bardzo wydajne konsole służące do gier oraz instalować w nich różne pakiety biurowe.

65



Popularne systemy RTOS

– Windows XP Embedded –



Windows XP Embedded, nie jest zasadniczo przeznaczony do przetwarzania w czasie rzeczywistym. Jego budowa jest jednak tak zorganizowana, że łatwo można go wzbogacić o takie możliwości. Służą do tego specjalne komponenty oraz rozwiązania firm trzecich.

Windows XP Embedded składa się dokładnie z tych samych plików binarnych, co jego brat, przeznaczony na komputery typu desktop, tyle że niektóre zostały nieznacznie zmodyfikowane lub uproszczone.

Posiada architekturę modułową.

Znajduje zastosowanie u producentów bankomatów, terminale graficzne, urządzeń przenośnych, konsoli do gier oraz tych którzy wykorzystują najnowsze technologie multimedialne.

66



Systemy RTOS - obecnie



Obecnie dostępnych jest około 90 systemów RTOS lub mających znamiona systemów operacyjnych czasu rzeczywistego.

Część z nich to kompletne systemy operacyjne, w skład których wchodzą jądro, sterowniki I/O, systemy plików, usługi sieciowe ...

Niektóre dostępne są na licencji GPL a niektóre odpłatnie.

Ceny wahają się od kilkuset do kilkuset tysięcy zł, przy czym cena zależy od dodatkowych modułów

67



Dziękuję za uwagę !!!

68