Oprogramowanie systemowe

Dariusz Wawrzyniak

✉ Politechnika Poznańska
Instytut Informatyki
ul. Piotrowo (CW, pok. 5)
60-965 Poznań

Dariusz.Wawrzyniak@cs.put.poznan.pl

www.cs.put.poznan.pl/dwawrzyniak

☎ +48 (61) 665 29 63

Program przedmiotu

1. Wprowadzenie — wielowarstwowa organizacja systemu komputerowego
2. Procesor — rozkazy, cykl rozkazowy, rejestry, adresowanie pamięci
3. Wielozadaniowość — koncepcja procesu, przełączanie kontekstu, szeregowanie zadań,
4. Pamięć — alokacja pamięci, organizacja przestrzeń adresowej, wiązanie adresu, wirtualizacja, cache
5. Synchronizacja procesów i zakleszczenie
6. Urządzenia wejścia-wyjścia — klasyfikacja, interakcja z jednostką centralną, buforowanie i spooling
7. Zarządzanie informacją — organizacja systemu plików

Literatura (1)

1. G. Nutt: *Operating Systems. A Modern Perspective*, wyd. 2, Addison Wesley Longman, Inc., 2002
2. L. Bic, A. C. Shaw: *The Logical Design of Operating Systems*, Prentice-Hall, Inc, 1988
3. W. Stallings: *Systemy operacyjne*, Wydawnictwo Naukowe PWN, Warszawa, 2006.
4. A. S. Tanenbaum: *Strukturalna organizacja systemów komputerowych*, wyd. 5, Helion, Gliwice, 2006.
5. A. Silberschatz, J. L. Peterson , G. Gagne : *Podstawy systemów operacyjnych*, WNT, Warszawa 2005.

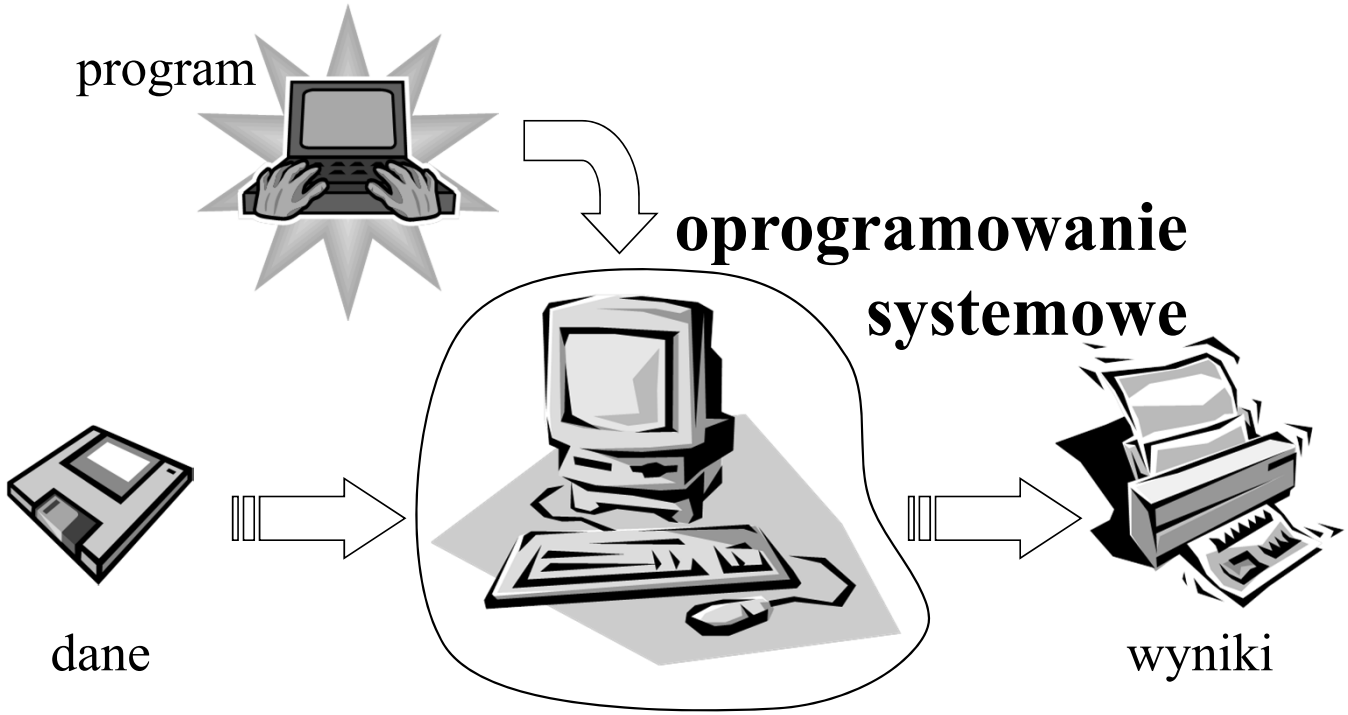
Literatura

6. L. Null, J. Lobur: *Struktura organizacyjna i architektura systemów komputerowych*, HELION, 2004
7. W. Stallings: *Organizacja i architektura systemu komputerowego*, WNT, Warszawa 2000
8. A. S. Tanenbaum: *Systemy operacyjne*, wyd. 3, Helion, 2010

Wprowadzenie

1. Organizacja przetwarzania danych
2. Struktura systemu komputerowego
3. Oprogramowanie systemu komputerowego
4. Podejście wielowarstwowe
5. Translacja i interpretacja
6. Pojęcie organizacji i architektury systemu komp.
7. Wielowarstwowa struktura systemu komputerowego

Organizacja przetwarzania danych



SUM SKiSR Oprogramowanie systemowe 6

Struktura systemu komputerowego

oprogramowanie aplikacyjne

oprogramowanie użytkowe

oprogramowanie systemowe

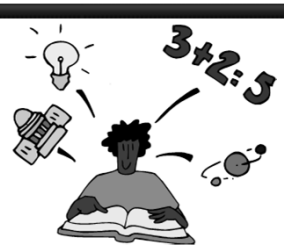
oprogramowanie (ang. software)

sprzęt (ang. hardware)

Oprogramowanie

- aplikacyjne — zbiór programów do przetwarzania danych
- użytkowe — zbiór programów ułatwiających pracę i poruszanie się użytkownika w systemie komputerowym (edytory, eksploratory, kompilatory, debuggery, profilery itp.)
- oprogramowanie systemowe — zbiór narzędzi do automatycznego lub „ręcznego” zarządzania zasobami systemu komputerowego (np. system operacyjny)

Wielowarstwowe ujęcie struktury systemu komputerowego



program w języku „wygodnym”
dla człowieka

przekład na język L_n

⇩

przekład na język L_{n-1}

⇩

...

⇩

przekład na język L_1

program w języku „wygodnym”
dla komputera (L_0)

Maszyna wirtualna

- Maszyna wirtualna warstwy k — M_k — rozumiana jest jako logiczny komputer, którego językiem maszynowym jest język L_k, a realizacja programu w tym języku polega na zleceniu wykonania odpowiednich instrukcji w języku $L_{k\text{-}1}$ maszynie $M_{k\text{-}1}$.

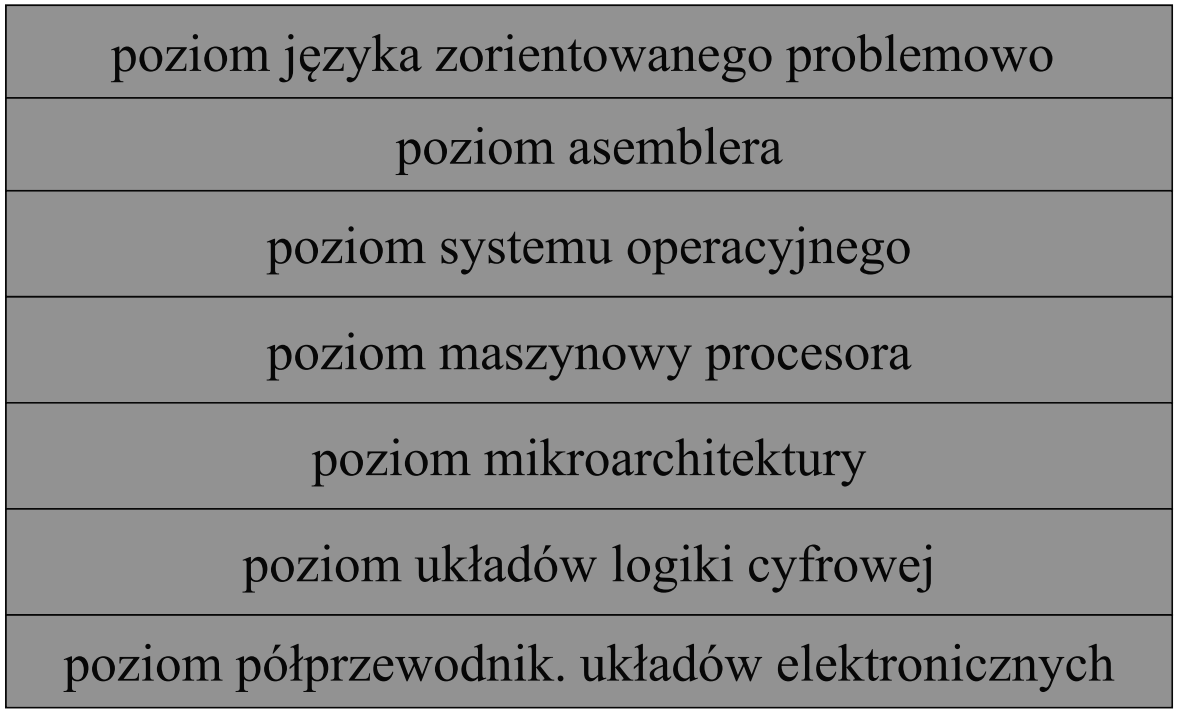
Translacja i interpretacja

- translacja — przełożenie całego ciągu instrukcji w języku warstwy wyższej — L_k — na równoważny ciąg instrukcji w języku warstwy niższej — $L_{k\text{-}1}$, a następnie realizacja uzyskanego programu w języku $L_{k\text{-}1}$ przez maszynę $M_{k\text{-}1}$.
- interpretacja — sukcesywne pobieranie instrukcji programu w języku warstwy wyższej — L_k, zamiana pobranej instrukcji na odpowiadający jej ciąg instrukcji w języku warstwy niższej — $L_{k\text{-}1}$, a następnie realizacja uzyskanego ciągu przez maszynę $M_{k\text{-}1}$.

Pojęcie organizacji i architektury systemu komputerowego

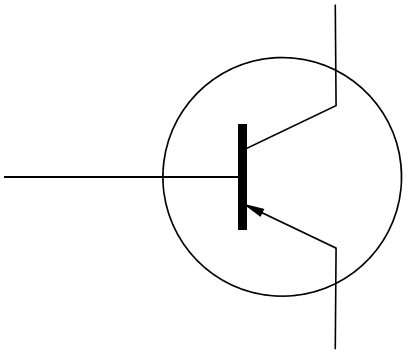
- **Architektura** systemu komputerowego obejmuje te jego elementy, które są istotne dla programisty (wpływają bezpośrednio na logiczne wykonywanie programu i tym samym sposób jego konstrukcji), np. lista instrukcji, typy danych, sposób adresowania pamięci itp.
- **Organizacja** systemu komputerowego obejmuje powiązania jednostek funkcjonalnych, które mogą wpływać na sposób wykonania instrukcji, ale nie mają bezpośredniego wpływu na logikę działania programu lub wynik jego działania, np. technologia wykonania, częstotliwość pracy procesora itp.

Wielowarstwowa struktura systemu komputerowego



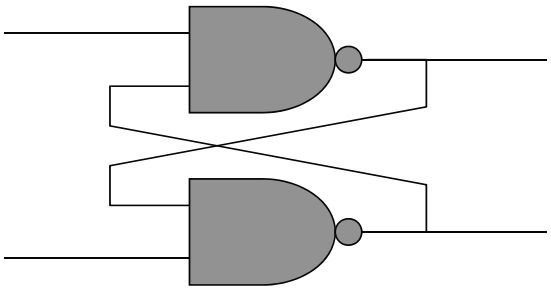
Poziom półprzewodnikowych układów elektronicznych

- Podstawowy element — tranzystor
- Informacja reprezentowana jest przez sygnał elektryczny



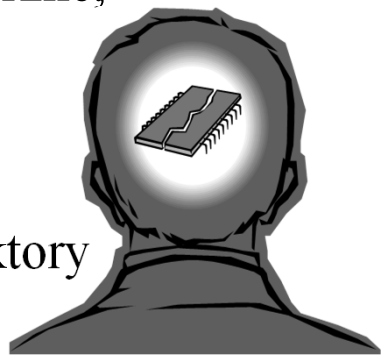
Poziom układów logiki cyfrowej

- Podstawowy element — bramka logiczna (AND, NAND, OR, NOR)
- Informacja reprezentowana jest przez stan logiczny wejść i wyjść układów cyfrowych



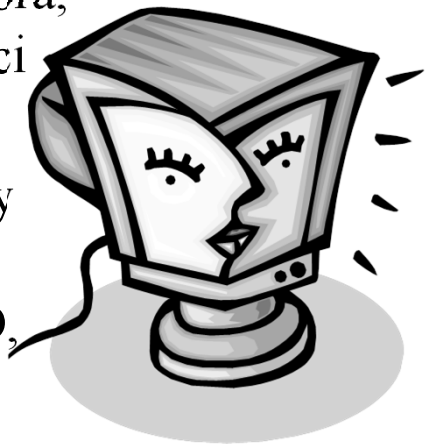
Poziom mikroarchitektury

- Podstawowe elementy — rejestry wewnętrzne, jednostka arytmetyczno-logiczna (ALU), jednostka zmiennoprzecinkowa (FPU), jednostka zarządzania pamięcią (MMU)
- Informacja reprezentowana jest przez wektory bitów (bajty, słowa)
- Poziom mikroarchitektury może być zrealizowany sprzętowo lub programowo (mikroprogram)



Poziom maszynowy procesora

- Podstawowe elementy — rejestry procesora, lista rozkazów, tryby adresowania pamięci
- Informacja reprezentowana jest przez niskopoziomowe struktury danych (liczby całkowite ze znakiem i bez znaku, liczby zmiennopozycyjne, liczby w kodzie BCD, adresy, łańcuchy znaków)
- Poziom maszynowy definiuje niskopoziomową architekturę systemu komputerowego



Poziom systemu operacyjnego

- Poziom hybrydowy — nie ukrywa poziomu maszynowego przed wyższymi warstwami, ale raczej uzupełnia go o dodatkowe mechanizmy
- Podstawowe elementy — elementy poziomu maszynowego procesora uzupełnione o organizację pamięci, interakcję z urządzeniami wejścia-wyjścia, system plików
- Informacja reprezentowana jest przez pliki

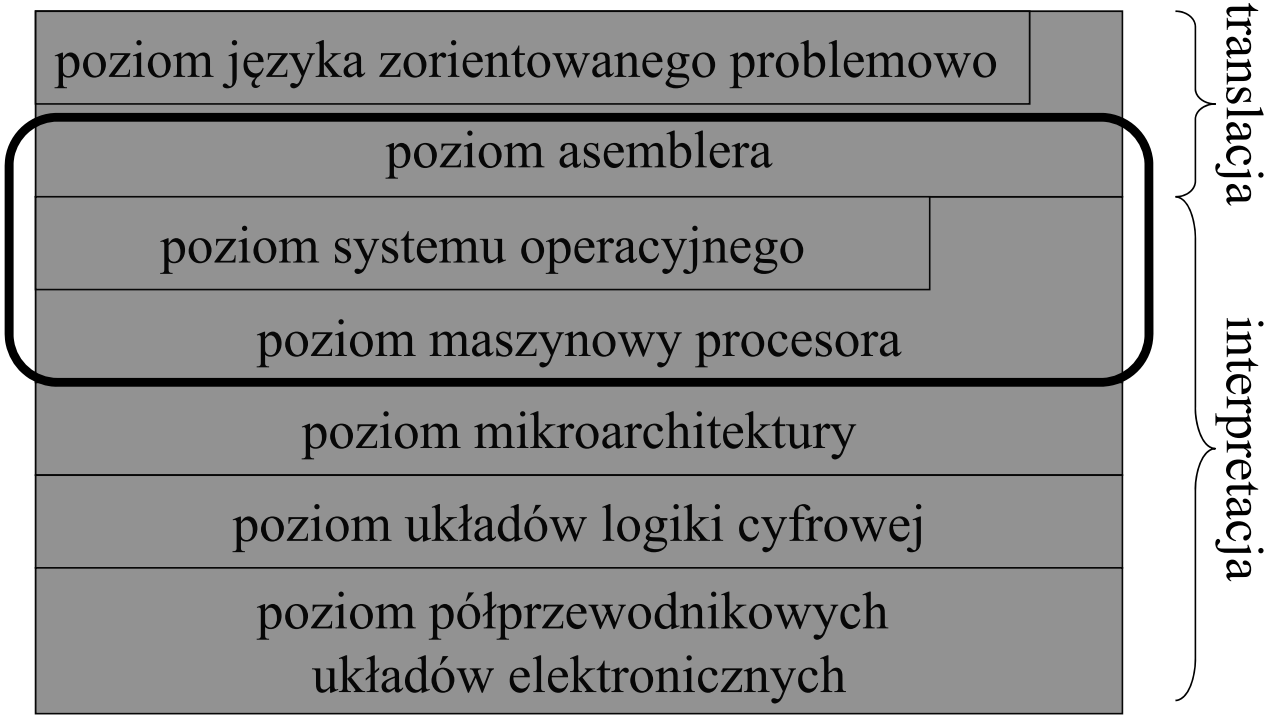
Poziom asemblera

- Udostępnia funkcjonalność poziomu maszynowego oraz poziomu systemu operacyjnego w formie symbolicznej, łatwiejszej do opanowania dla programisty (mnemoniki, etykiety)

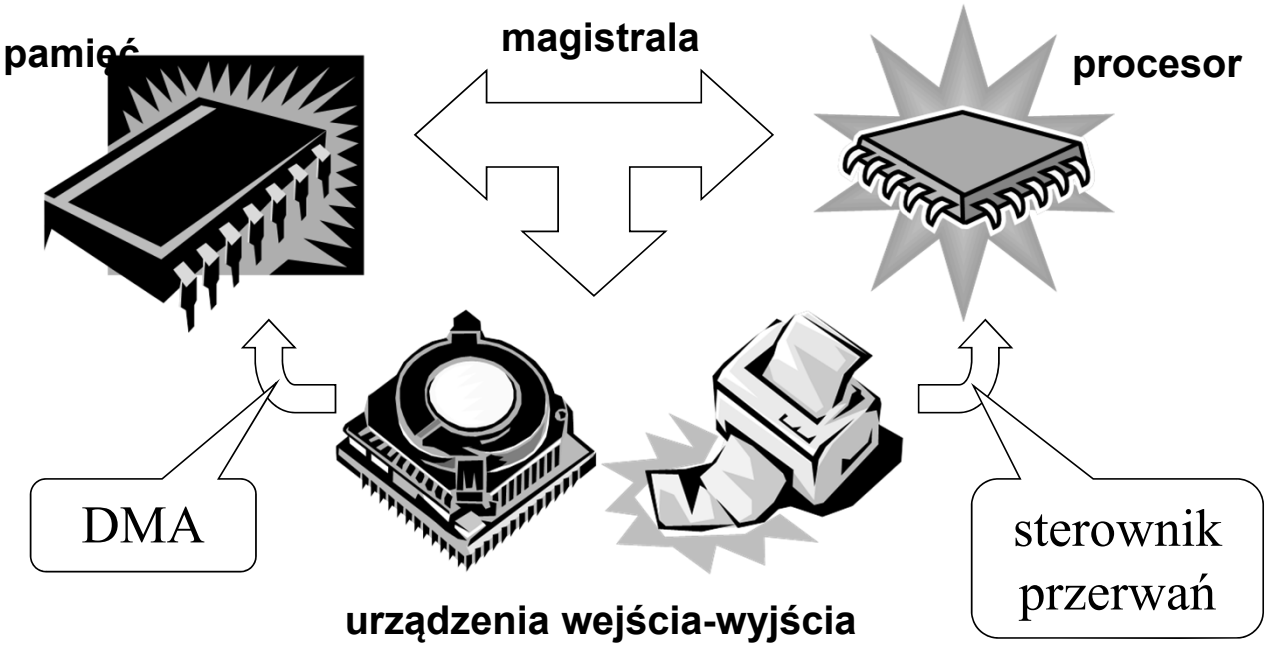
Poziom języka zorientowanego problemowo

- Ukrywa szczegóły architektury procesora oraz redefiniuje interfejs dostępu do niektórych usług systemu operacyjnego w celu uniezależnienia oprogramowania od konkretnego systemu komputerowego
- Udostępnia funkcjonalność (instrukcje, struktury danych) dostosowaną do konkretnych zastosowań (np. dostęp do baz danych, zastosowania numeryczne itp.)

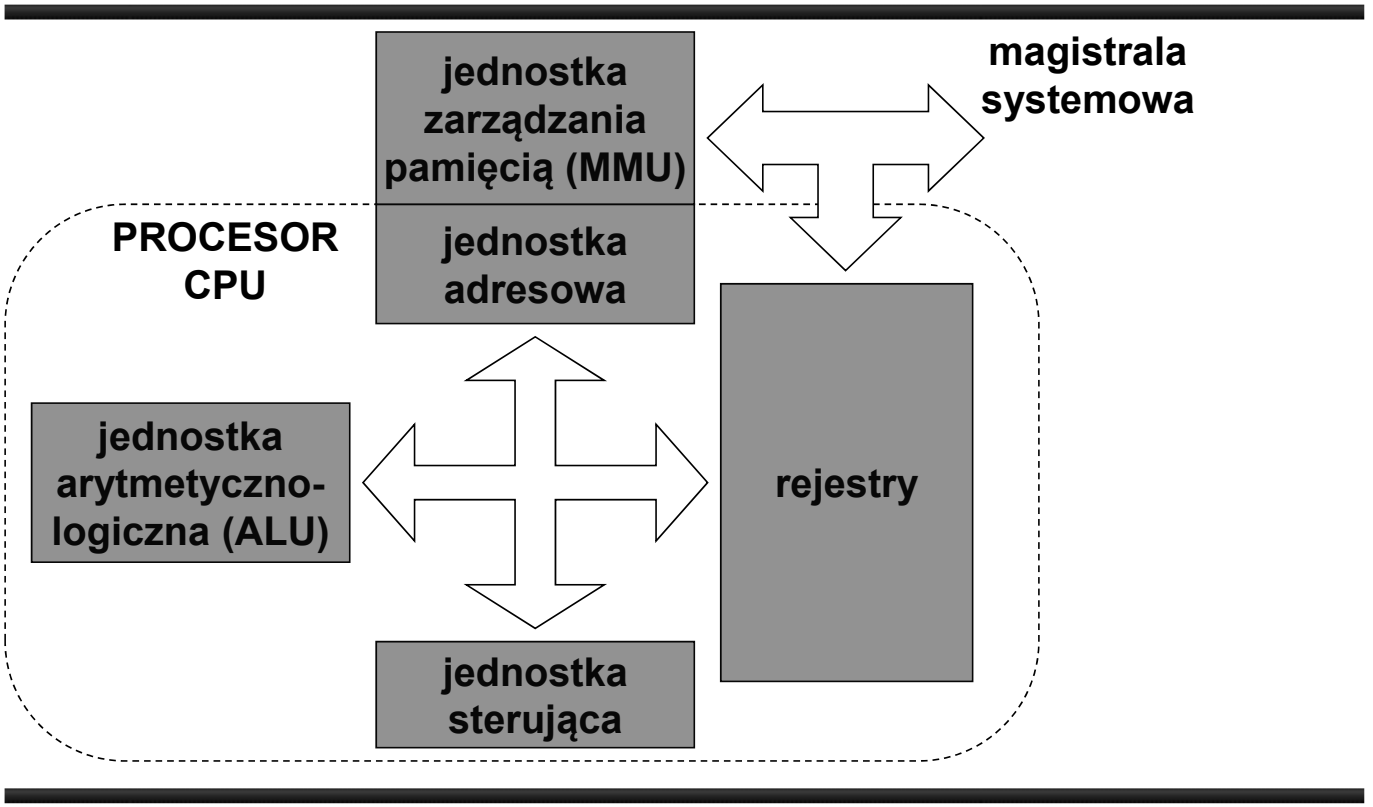
Wielowarstwowa struktura systemu komputerowego — podsumowanie



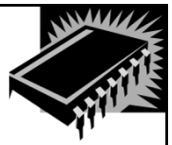
Architektura komputera na poziomie maszynowym



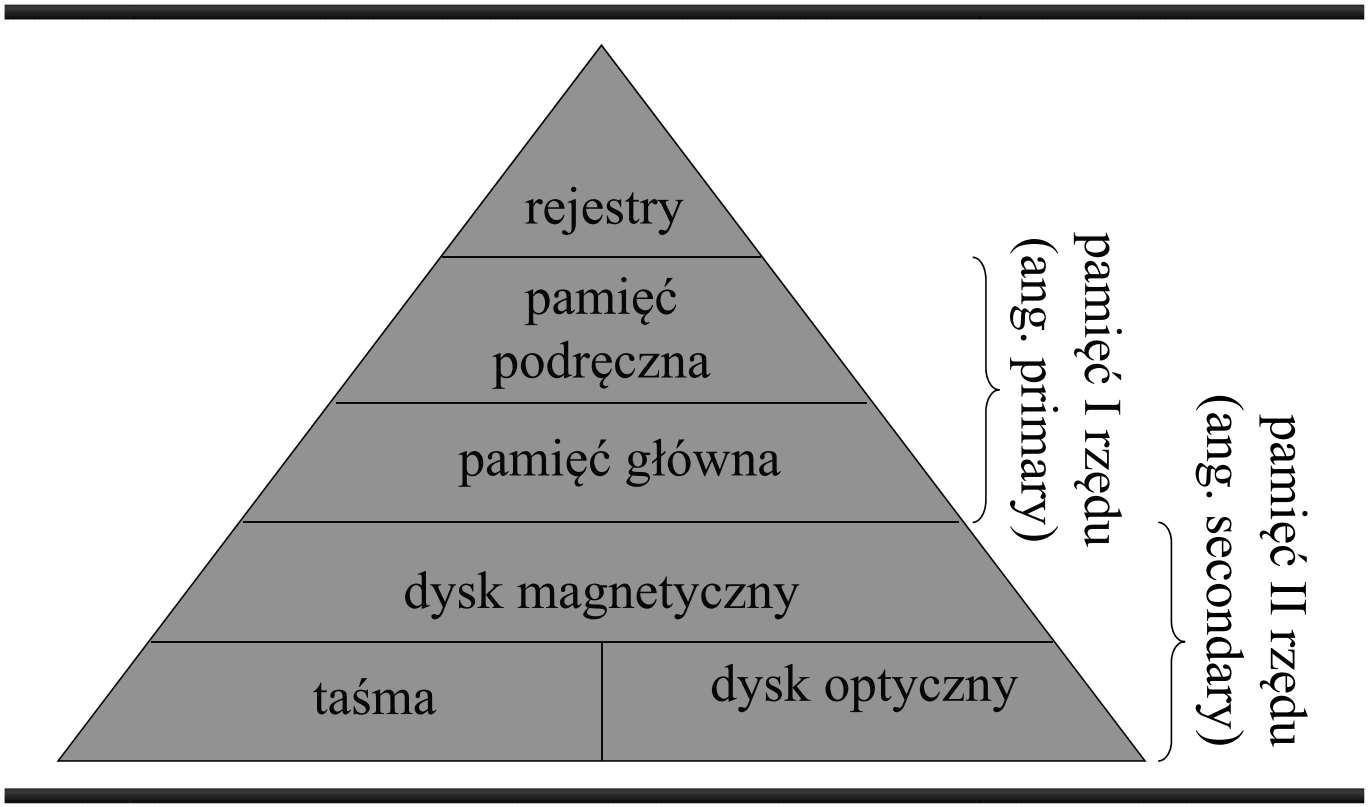
Organizacja procesora



SUM SKiSR Oprogramowanie systemowe 23



Hierarchia pamięci



SUM SKiSR Oprogramowanie systemowe 24

Definicja systemu operacyjnego (1)

- System operacyjny jest zbiorem ręcznych i automatycznych procedur, które pozwalają grupie osób na efektywne współdzielenie urządzeń maszyny cyfrowej.

  Per Brinch Hansen

- System operacyjny (nadzorczy, nadrzędny, sterujący) jest to zorganizowany zespół programów, które pośredniczą między sprzętem a użytkownikami, dostarczając użytkownikom zestawu środków ułatwiających projektowanie, kodowanie, uruchamianie i eksploatację programów oraz w tym samym czasie sterują przydziałem zasobów dla zapewnienia efektywnego działania.

  Alen Shaw

Definicja systemu operacyjnego (2)

- System operacyjny jest programem, który działa jako pośrednik między użytkownikiem komputera a sprzętem komputerowym. Zadaniem systemu operacyjnego jest tworzenie środowiska, w którym użytkownik może wykonywać programy w sposób wygodny i wydajny.

  Abraham Silberschatz

- System operacyjny jest warstwą oprogramowania operującą bezpośrednio na sprzęcie, której celem jest zarządzanie zasobami systemu komputerowego i stworzenie użytkownikowi środowiska łatwiejszego do zrozumienia i wykorzystania.

  Andrew Tanenbaum

Zasada działania systemu operacyjnego

- Odwołania do jądra systemu przez system przerwań lub specjalne instrukcje (przerwanie programowe)
- Dualny tryb pracy — tryb użytkownika (ang. user mode) i tryb systemowy (ang. system mode)
- Wyróżnienie instrukcji uprzywilejowanych, wykonywanych tylko w trybie systemowym
- Sprzętowa ochrona pamięci
- Uprzywilejowanie instrukcji wejścia-wyjścia
- Przerwanie zegarowe

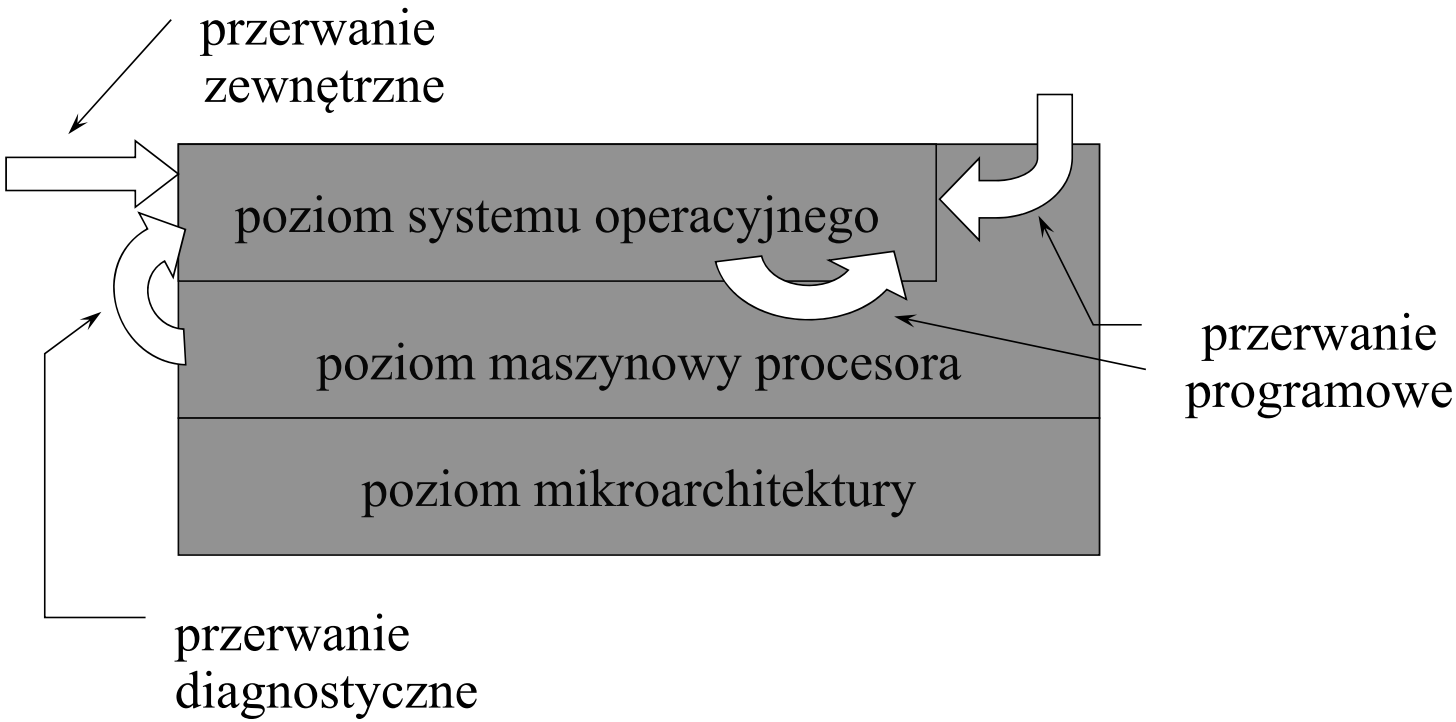
Przerwania w systemie komputerowym

Przerwanie jest reakcją na asynchroniczne zdarzenie, polegającą na automatycznym zapamiętaniu bieżącego stanu procesora w celu późniejszego odtworzenia oraz przekazaniu sterowania do ustalonej procedury obsługi przerwania.

Podział przerwań ze względu na źródło:

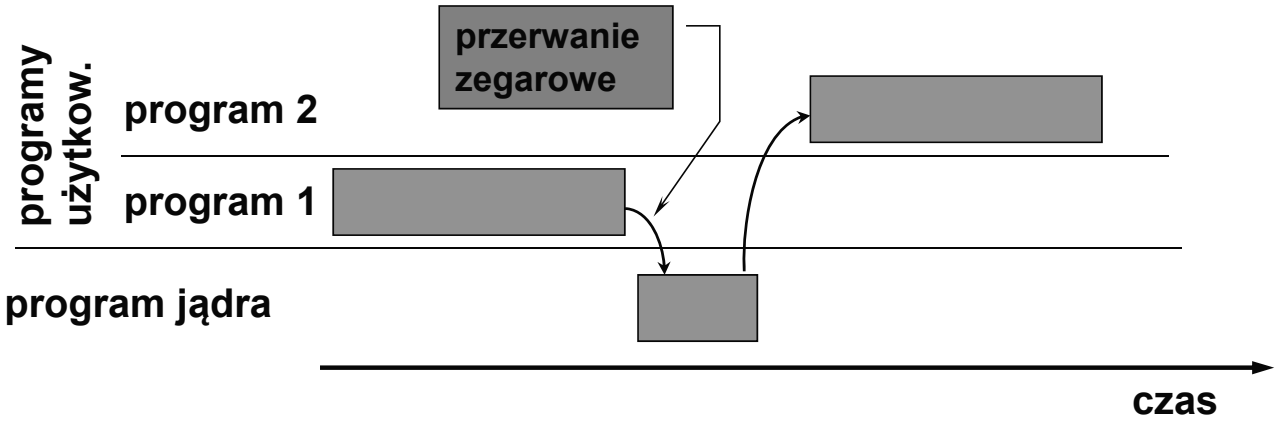
- przerwania zewnętrzne — od urządzeń zewnętrznych
- przerwania programowe — wykonanie specjalnej instrukcji
- przerwania diagnostyczne — pułapki, błędy programowe i sprzętowe

Przerwania w ujęciu wielowarstwowym



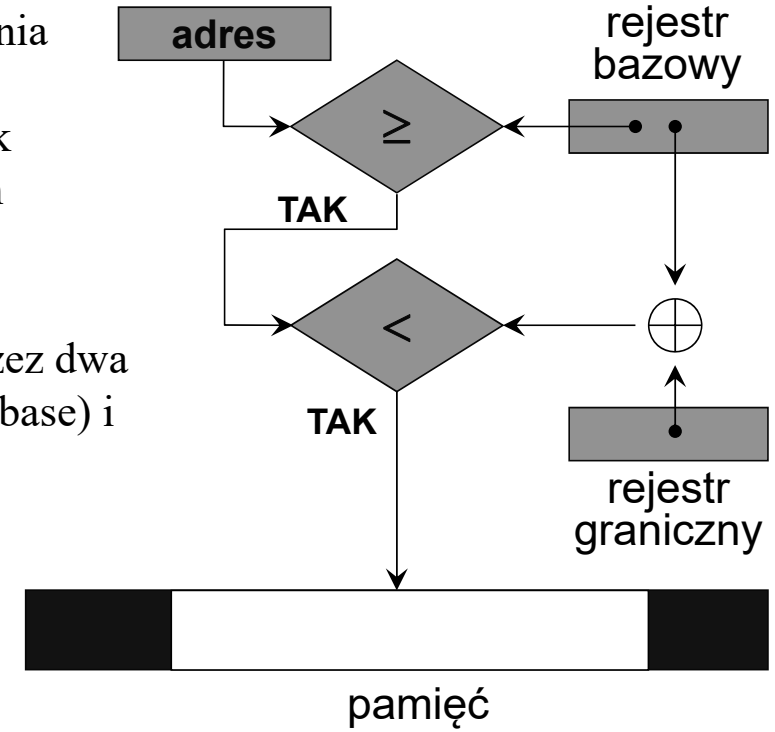
Przerwanie zegarowe

- Przerwanie zegarowe generowane jest przez czasomierz (ang. timer) po wyznaczonym okresie czasu.
- Obsługa przerwania zegarowego oznacza przekazanie sterowania do jądra systemu operacyjnego, umożliwiając w ten sposób wykonanie pewnych zdań okresowych oraz uniemożliwiając zawłaszczenie procesora przez program użytkownika.



Zasady ochrony pamięci

- W wyniku wykonywania programu następuje odwołanie do komórek pamięci o określonych adresach
- Zaalokowane obszary pamięci opisane są przez dwa parametry: bazę (ang. base) i granicę (ang. limit)



Procesor

1. Elementy architektury procesora
2. Klasyfikacja rejestrów procesora
3. Podział rozkazów procesora
4. Tryby adresowania pamięci
5. Typy danych
6. Realizacja cyklu rozkazowego
7. Architektura RISC i CISC
8. Elementy programowania niskopoziomowego

Elementy architektury procesora

- model pamięci — określa jednostki, jakie można adresować i w jakich można wymieniać dane pomiędzy procesorem a pamięcią (np. bajty, słowa) oraz sposób ich ustawienia (ang. alignment)
- rejestry procesora dostępne dla programisty — liczba rejestrów, ich rozmiar oraz przeznaczenie
- lista rozkazów procesora — zbiór instrukcji, które procesor potrafi wykonać oraz wymaganych operandów dla tych instrukcji
- tryby adresowania — określają sposób specyfikowania adresu operandów
- typy danych — określają sposób interpretacji wartości przechowywanych w rejestrach lub pamięci i tym samym zbiór dostępnych operacji na tych wartościach

Klasyfikacja rejestrów procesora

- Rejestry ogólnego przeznaczenia (ang. general purpose registers) — mogą przechowywać dowolne wartości (np. tymczasowe wyniki), a ich zawartość nie ma bezpośredniego wpływu na sposób przetwarzania lub wybór danych
- Rejestry specjalnego przeznaczenia (ang. special purpose registers) — rejestry, które pełnią szczególną rolę w czasie wykonywania programu (np. licznik rozkazów, wskaźnik stosu)
- Rejestry ogólnego przeznaczenia ze specyficzną funkcjonalnością — mogą na ogół przechowywać dowolne wartości, ale spełniają szczególną funkcję w czasie wykonywania niektórych instrukcji (np. licznik pętli, rejestr bazowy)

Typowe rejestry procesora

- Akumulator — przechowuje operand lub wynik operacji arytmetycznej
- Licznik rozkazów — zawiera adres następnej instrukcji do wykonania
- Rejestr flagowy — zawiera bity, które są ustawiane w celu zasygnalizowania specyficznego stanu przetwarzania (np. przeniesienie, przepełnienie, bit znaku, bit przerwania itp.)

Podział rozkazów procesora

- Rozkazy transferu (kopiowania) danych
  - wewnętrznie pomiędzy rejestrami
  - pomiędzy rejestrami a pamięcią
  - pomiędzy rejestrami a portami wejścia-wyjścia
- Rozkazy przetwarzania danych
  - rozkazy arytmetyczne — dodawanie, odejmowanie, mnożenie, dzielenie, inkrementacja itp.
  - bitowe rozkazy logiczne — AND, OR, XOR, NOT itp.
- Rozkazy porównania
- Rozkazy przekazywania sterowania
  - rozkazy skoków i rozgałęzień warunkowych
  - rozkazy wywołania podprogramu i powrotu z podprogramu

Typy danych

- numeryczne
  - typ całkowity ze znakiem lub bez znaku różnej długości (np. 8 bitów, 32 bity, 64 bity itp.)
  - typ zmiennopozycyjny
- nienumeryczne
  - znaki (np. ASCII, UNICODE)
  - łańcuchy znaków (ang. strings)
  - wartości logiczne
  - mapa bitowa
  - wskaźnik, czyli adres w pamięci

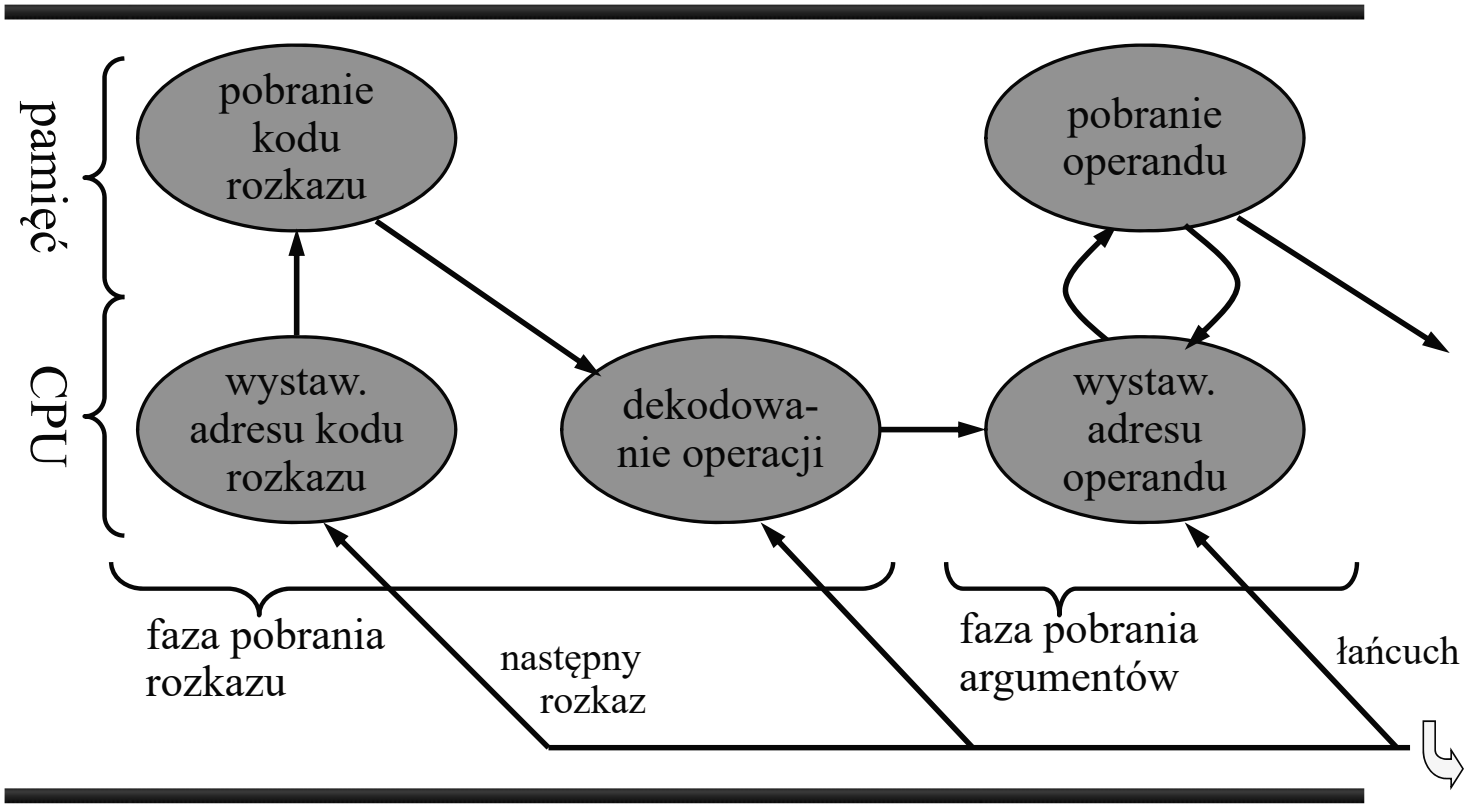
Architektura von Neumana — założenia

- Zarówno program (kody instrukcji), jak i dane (argumenty instrukcji, operandy) znajdują się w pamięci operacyjnej.
- Rozkazy wykonywane są w kolejności, w jakiej zostały umieszczone w programie (i tym samym w pamięci), a zmiana tej kolejności może nastąpić w wyniku wykonania specjalnej instrukcji, np. instrukcji skoku, wywołania podprogramu, powrotu z podprogramu itp., (jest więc zdefiniowana przez sam program).
- W celu pobrania rozkazu z pamięci procesor wystawia odpowiedni adres na magistrali adresowej.

Cykl rozkazowy

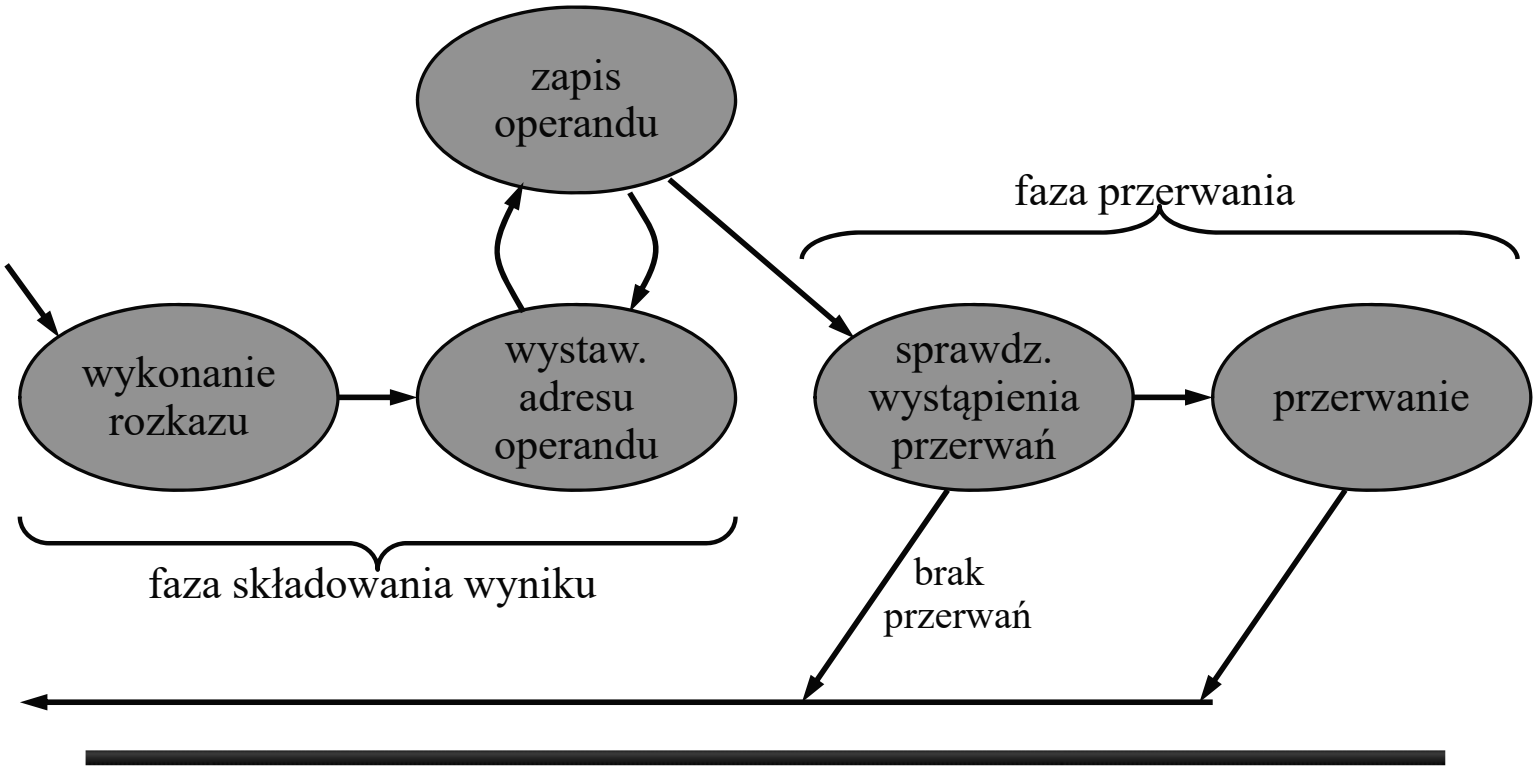
- Cykl rozkazowy — cykl działań procesora i jego interakcji z pamięcią operacyjną związanych z realizacją rozkazu.
- Cykl rozkazowy składa się z faz, zwanych cyklami maszynowymi.
- Typowe fazy cyklu rozkazowego:
  - pobranie kodu rozkazu — odczyt pamięci
  - pobranie operandu — odczyt pamięci
  - składowanie wyniku — zapis pamięci

Cykl rozkazowy — graf stanów procesora (1)



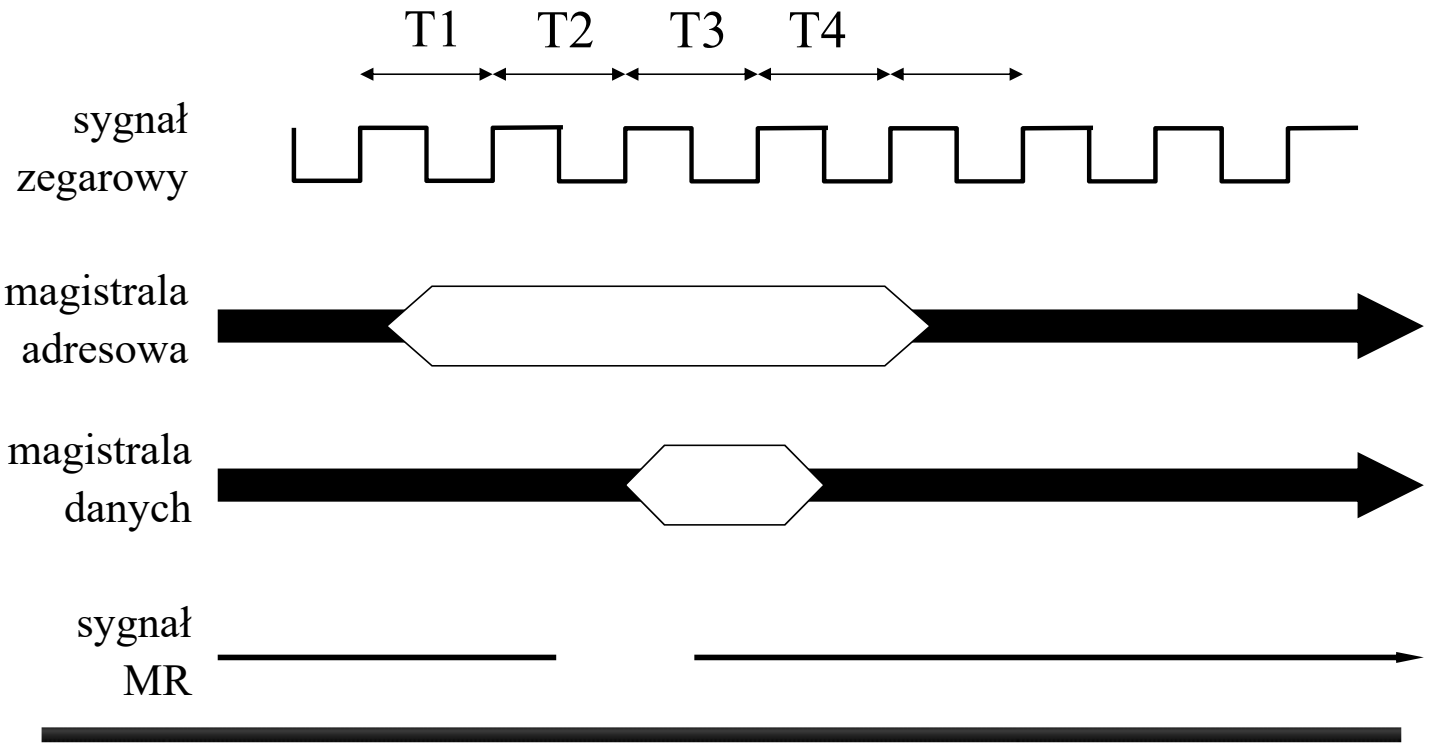
SUM SKiSR Oprogramowanie systemowe 48

Cykl rozkazowy — graf stanów procesora (2)



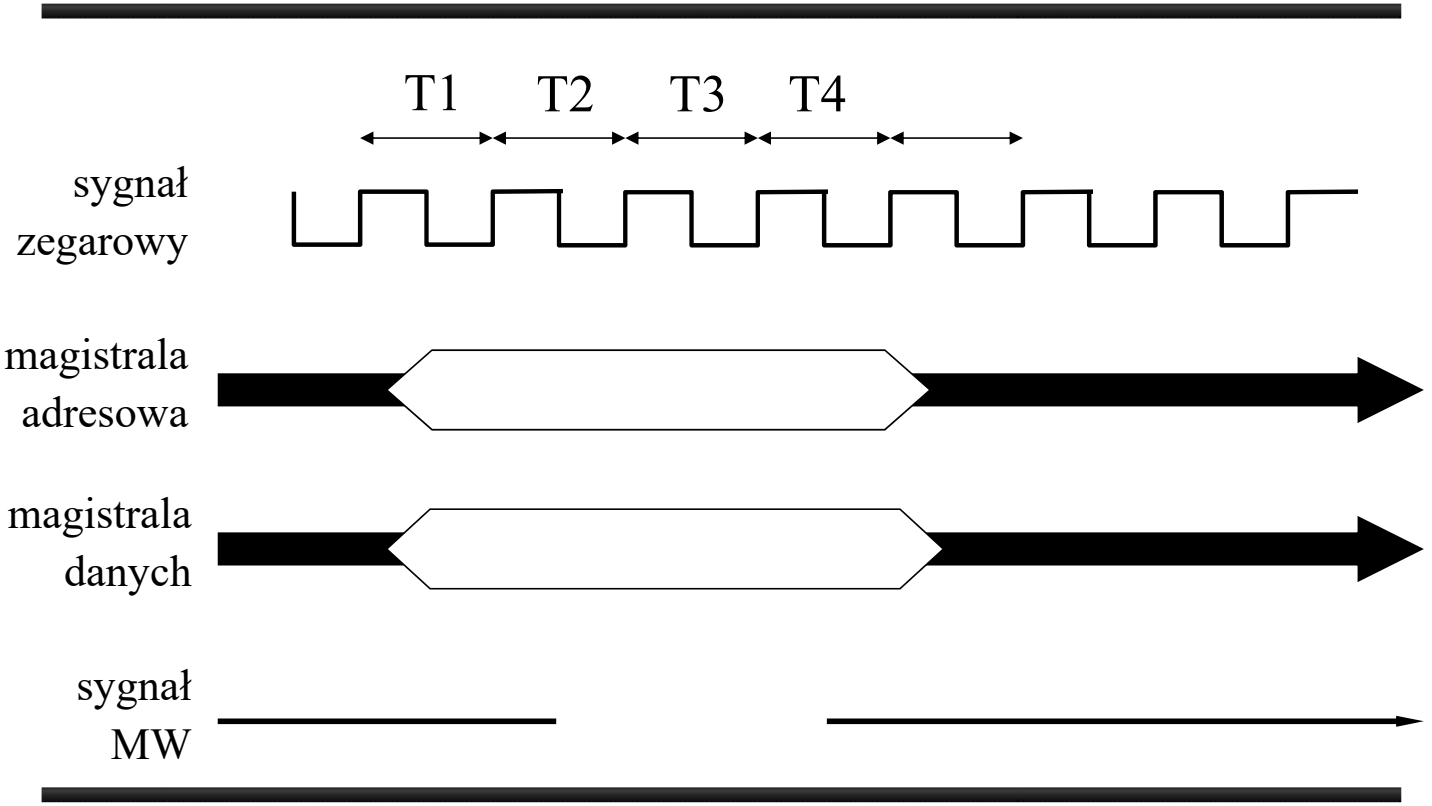
SUM SKiSR Oprogramowanie systemowe 49

Cykl odczytu z pamięci — przebieg czasowy sygnałów na magistrali



SUM SKiSR Oprogramowanie systemowe 50

Cykl zapisu w pamięci — przebieg czasowy sygnałów na magistrali



SUM SKiSR Oprogramowanie systemowe 51

CISC i RISC

Complex instruction set computers

- rozbudowana, uniwersalna lista rozkazów
- zróżnicowany format rozkazów
- różnorodność trybów adresowania pamięci

Reduced instruction set computers

- lista rozkazów zredukowana do najczęściej występujących, o stałym formacie i trwających krótko
- dostęp do pamięci tylko za pomocą rozkazów przesłań
- operandy tylko w rejestrach
- ograniczona liczba trybów adresowania pamięci
- rozbudowany zestaw rejestrów

Przesłanki rozwoju architektury typu RISC

- Częstość występowania rozkazów
  - transfer — 33%
  - skoki — 15%
  - rozkazy arytmetyczne i logiczne — 15%
  - rozkazy porównań — 10%
- Liczba parametrów podprogramów — średnio 5
- Liczba zmienny lokalnych podprogramów — średnio 7
- Poziom zagnieżdżenia wywołań podprogramów — 6

Wielozadaniowość

1. Koncepcja procesu
2. Blok kontrolny procesu i przełączanie kontekstu
3. Kolejki procesów
4. Wątki
5. Szeregowanie (planowanie przydziału procesora)
   - Planiści
   - Kryteria planowania
   - Algorytmy planowania

Klasyfikacja sys. op. ze względu na liczbę wykonywanych zadań

- Systemy jednozadaniowe (jednoprogramowe) — niedopuszczalne jest rozpoczęcie wykonywania następnego zadania użytkownika przed zakończeniem poprzedniego.
- Systemy wielozadaniowe (wieloprogramowe) — dopuszczalne jest istnienie jednocześnie wielu zadań (procesów), którym kolejno przydzielany jest procesor.

Koncepcja procesu

- Proces jest elementarną jednostką pracy (aktywności) zarządzaną przez system operacyjny, która może ubiegać się o zasoby systemu komputerowego
- Proces = wykonujący się program
- Zasoby potrzebne do wykonania procesu
  - czas procesora
  - pamięć operacyjna
  - pliki i i urządzenia wejścia-wyjścia
- Elementy składowe procesu:
  - program — definiuje zachowanie procesu,
  - dane — zbiór wartości przetwarzanych oraz wyniki,
  - zbiór zasobów tworzących środowisko wykonawcze,
  - blok kontrolny procesu (PCB, deskryptor)

Stany procesu

- Nowy (ang. new) — proces został utworzony
- Wykonywany (ang. running) — wykonywane są instrukcje programu
- Zawieszony (oczekujący, ang. suspanded, waiting) — proces oczekuje na jakieś zdarzenie, np. na zakończenie operacji wejścia-wyjścia, na przydział dodatkowego zasobu, synchronizuje się z innymi procesami
- Gotowy (ang. ready) — proces czeka na przydział procesora
- Zakończony (ang. terminated) — proces zakończył działanie i zwalnia zasoby

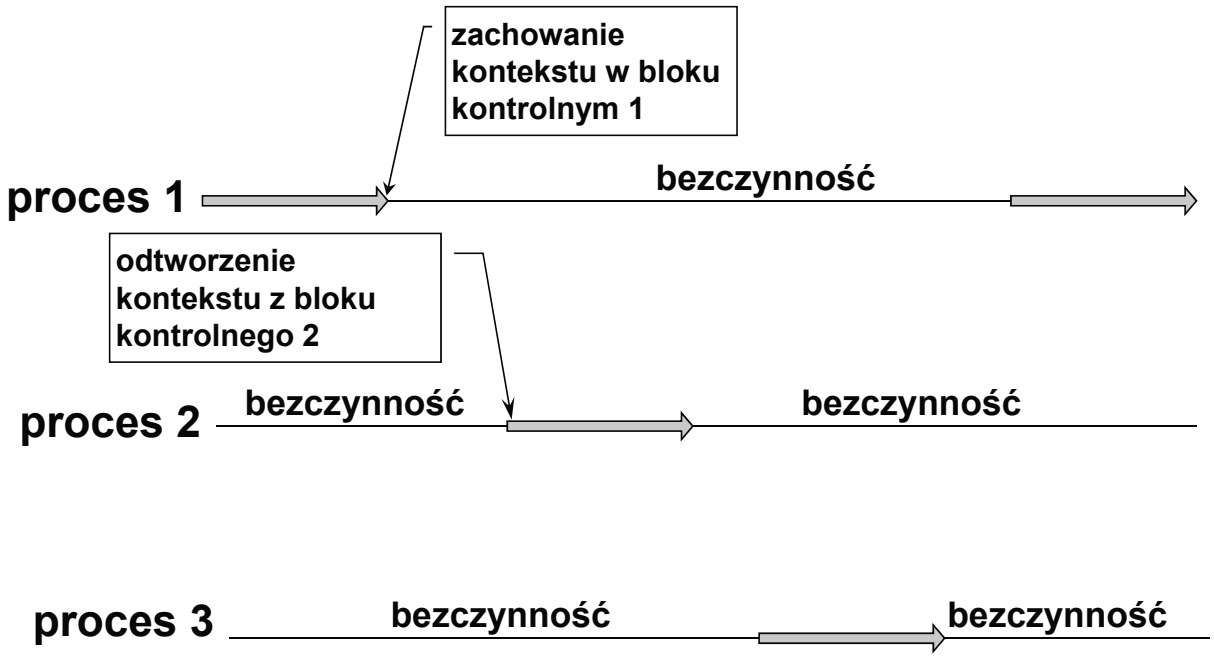
Cykl zmian stanów procesu

SUM SKiSR Oprogramowanie systemowe 69

Deskryptor procesu

- Identyfikator procesu
- Stan procesu (nowy, gotowy, oczekujący, itd.)
- Identyfikator właściciela
- Identyfikator przodka
- Lista przydzielonych zasobów
- Zawartość rejestrów procesora
- Prawa dostępu (domena ochrony)
- Informacje na potrzeby zarządzania pamięcią
- Informacje na potrzeby planowaniu przydziału procesora (priorytet, wskaźniki do kolejek porządkujących itp.)
- Informacje do rozliczeń

Przełączanie kontekstu (ang. context switch)



Kolejki procesów

- Kolejka zadań (ang. job queue) — wszystkie procesy systemu
- Kolejka procesów gotowych (ang. ready queue) — procesy gotowe do działania, przebywające w pamięci głównej
- Kolejka do urządzenia (ang. device queue) — procesy czekające na zakończenie operacji wejścia-wyjścia
- Kolejka procesów oczekujących w wyniku synchronizacji z innymi procesami (np. kolejka procesów na semaforze)

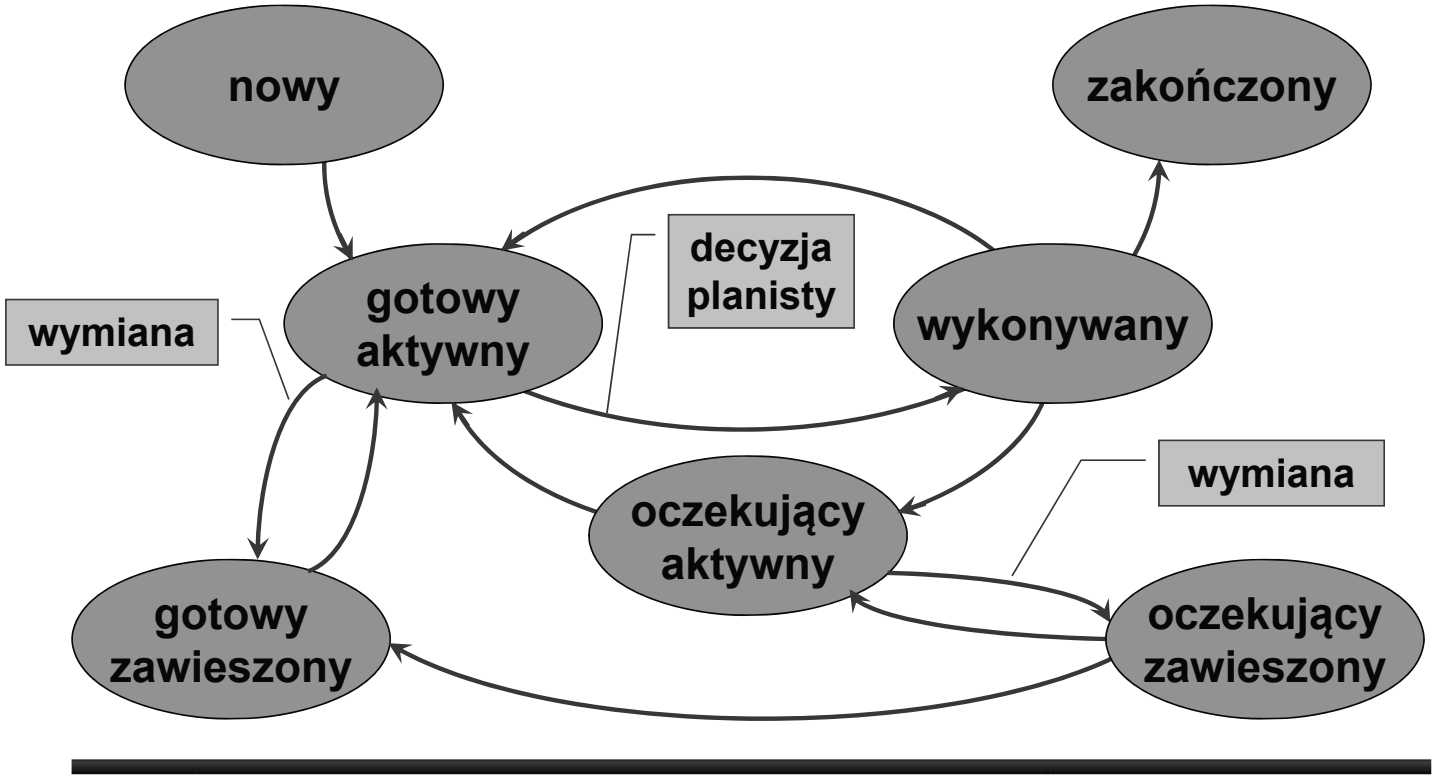
Planista (ang. scheduler)

- Planista krótkoterminowy, planista przydziału procesora (ang. CPU scheduler) — zajmuje się przydziałem procesora do procesów gotowych.
- Planista średnioterminowy (ang. medium-term scheduler) — zajmuje się wymianą procesów pomiędzy pamięcią główną a pamięcią zewnętrzną (np. dyskiem).
- Planista długoterminowy, planista zadań (ang. long-term scheduler, job scheduler) — zajmuje się ładowaniem nowych programów do pamięci, kontroluje stopień wieloprogramowości, dąży do zrównoważenia procesora.

Klasyfikacja systemów operacyjnych ze względu na sposób przetwarzania

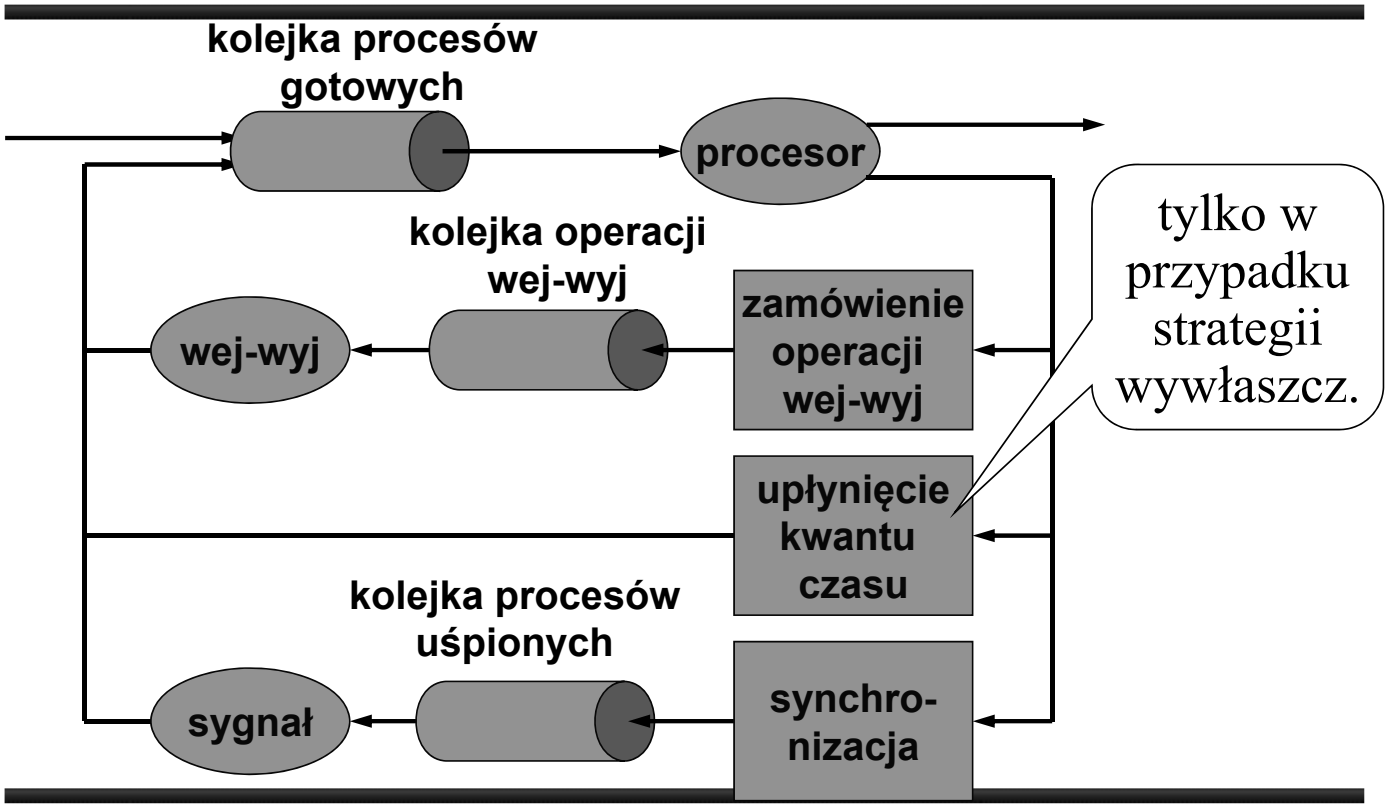
- Systemy przetwarzania bezpośredniego (ang. on-line processing systems) — systemy interakcyjne
  - występuje bezpośrednia interakcja pomiędzy użytkownikiem a systemem
  - wykonywanie zadania użytkownika rozpoczyna się zaraz po przedłożeniu
- Systemy przetwarzania pośredniego (ang. off-line processing systems) — systemy wsadowe
  - występuje istotna, nieznana zwłoka czasowa między przedłożeniem zadania a rozpoczęciem jego wykonywania
  - niemożliwa jest ingerencja użytkownika w wykonywanie zadania

Cykl zmian stanów procesu z uwzględnieniem wymiany



SUM SKiSR Oprogramowanie systemowe 75

Diagram kolejek w planowaniu procesora



SUM SKiSR Oprogramowanie systemowe 76

Wątki

- Wątek (lekki proces, ang. lightweight process — LWP) jest obiektem w obrębie procesu ciężkiego (heavyweight), posiadającym własne sterowanie i współdzielącym z innymi wątkami tego procesu przydzielone (procesowi) zasoby:
  - segment kodu i segment danych w pamięci
  - tablicę otwartych plików
  - tablicę sygnałów

Realizacja wątków

- Realizacja wątków na poziomie jądra systemu operacyjnego — jądro tworzy odpowiednie struktury (blok kontrolny) do utrzymywania stanu wątku.
- Realizacja wątków na poziomie użytkownika — struktury związane ze stanami wątków tworzone są w przestrzeni adresowej procesu.

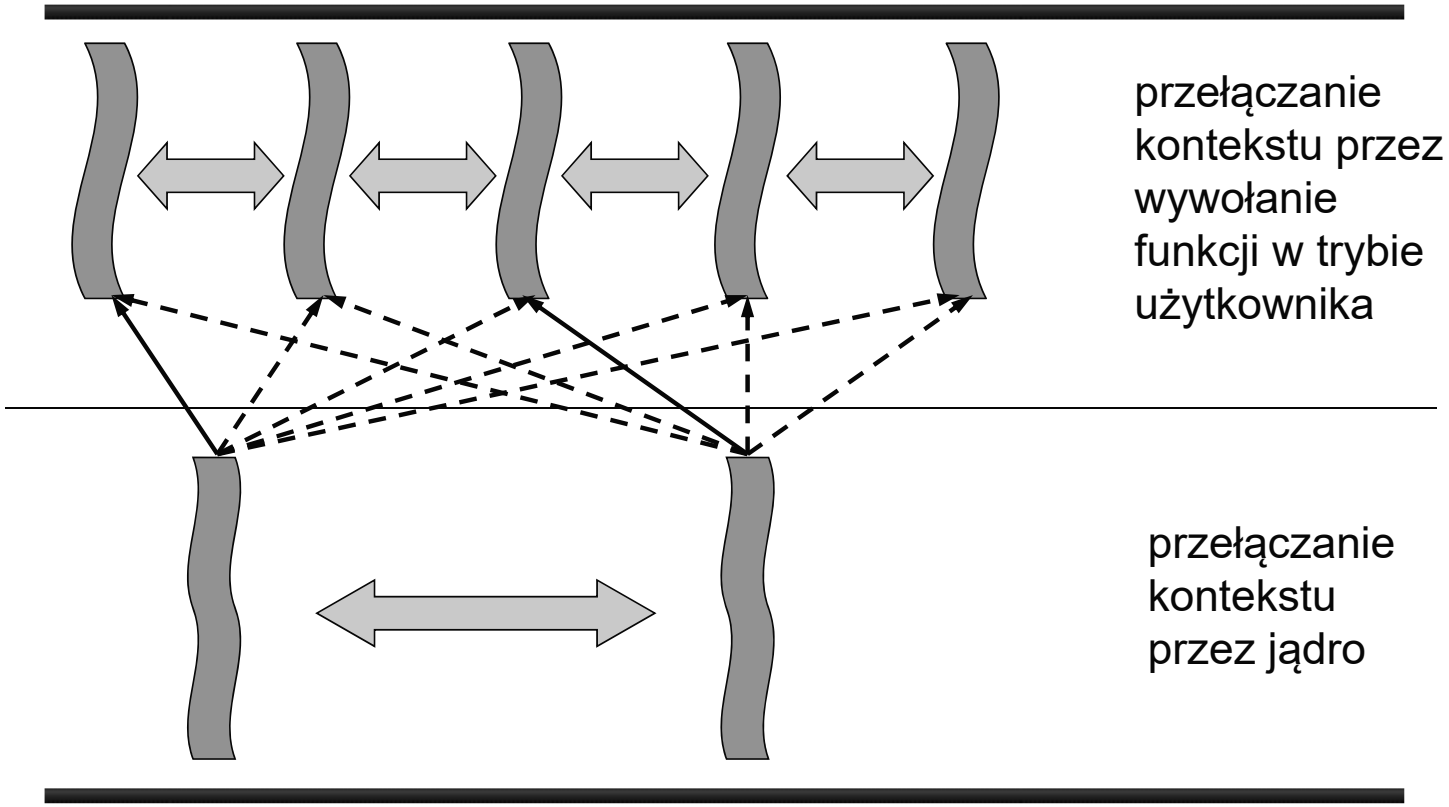
Realizacja wątków na poziomie jądra systemu operacyjnego

- Wątek posiada własny blok kontrolny w jądrze systemu operacyjnego, obejmujący:
  - stan licznika rozkazów,
  - stan rejestrów procesora,
  - stan rejestrów związanych z organizacją stosu.
- Własności realizacji wątków na poziomie jądra:
  - przełączanie kontekstu pomiędzy wątkami przez jądro,
  - większy koszt przełączanie kontekstu,
  - bardziej sprawiedliwy przydział czasu procesora.

Realizacja wątków w trybie użytkownika

- Deskryptor wątku znajduje się w tablicy wątków w pamięci danego procesu (jądro nie wie nic o wątkach).
- Własności realizacji na poziomie użytkownika:
  - przydział czasu procesora dla procesu (nie dla wątku)
  - przełączanie kontekstu pomiędzy wątkami przez jawne odwołania do mechanizmu obsługi wątków
  - mniejszy koszt przełączanie kontekstu (bez angażowania jądra systemu operacyjnego)
  - możliwość „głodzenia” wątków tego samego procesu, gdy jeden z nich spowoduje przejście procesu w stan oczekiwania

Przełączanie kontekstu wątków



SUM SKiSR Oprogramowanie systemowe 81

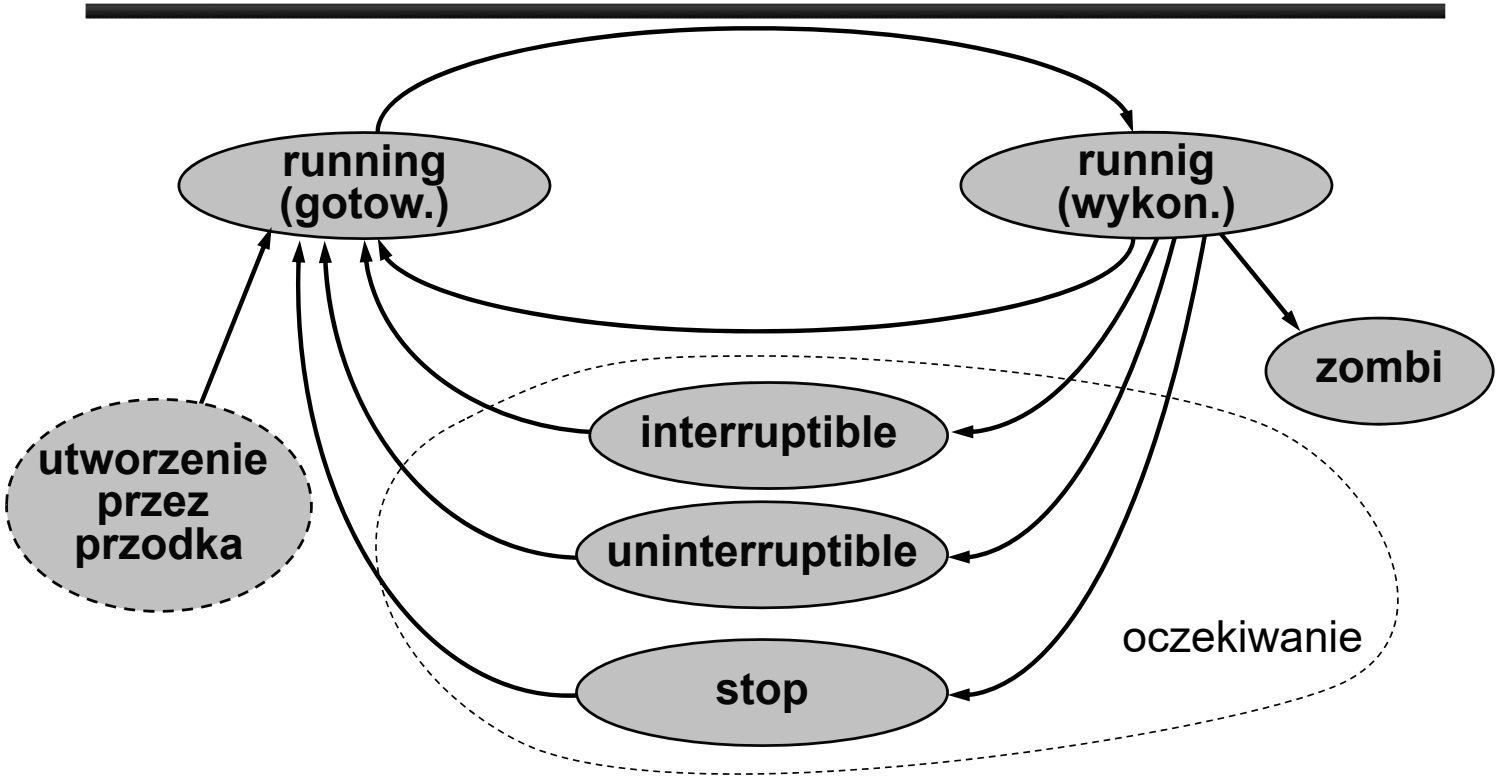
Realizacja procesów/wątków w systemie Linux

- W jądrze systemu Linux nie odróżnia się pojęcia wątku od procesu.
- Procesy mogą współdzielić takie zasoby, jak:
  - przestrzeń adresowa,
  - otwarte pliki,
  - informacje o systemie plików,
  - procedury obsługi sygnałów.
- Deskryptory procesów (o strukturze `struct task_struct`) przechowywane są na dwukierunkowej, cyklicznej liście zadań.

Stany procesu (wątku) w systemie Linux

- TASK_RUNNING — wykonywanie lub gotowość (do wykonania)
- TASK_INTERRUPTIBLE — oczekiwanie na zajście zdarzenia lub sygnał
- TASK_UNINTERRUPTIBLE — oczekiwanie na zajście zdarzenia, przy czym sygnały są ignorowane
- TASK_ZOMBI — stan zakończenia utrzymywany w celu przechowania deskryptora procesu
- TASK_STOP — zatrzymanie w wyniku otrzymania sygnału (np. SIGSTOP)

Cykl zmian stanów procesu (wątku) w systemie Linux



SUM SKiSR Oprogramowanie systemowe 84

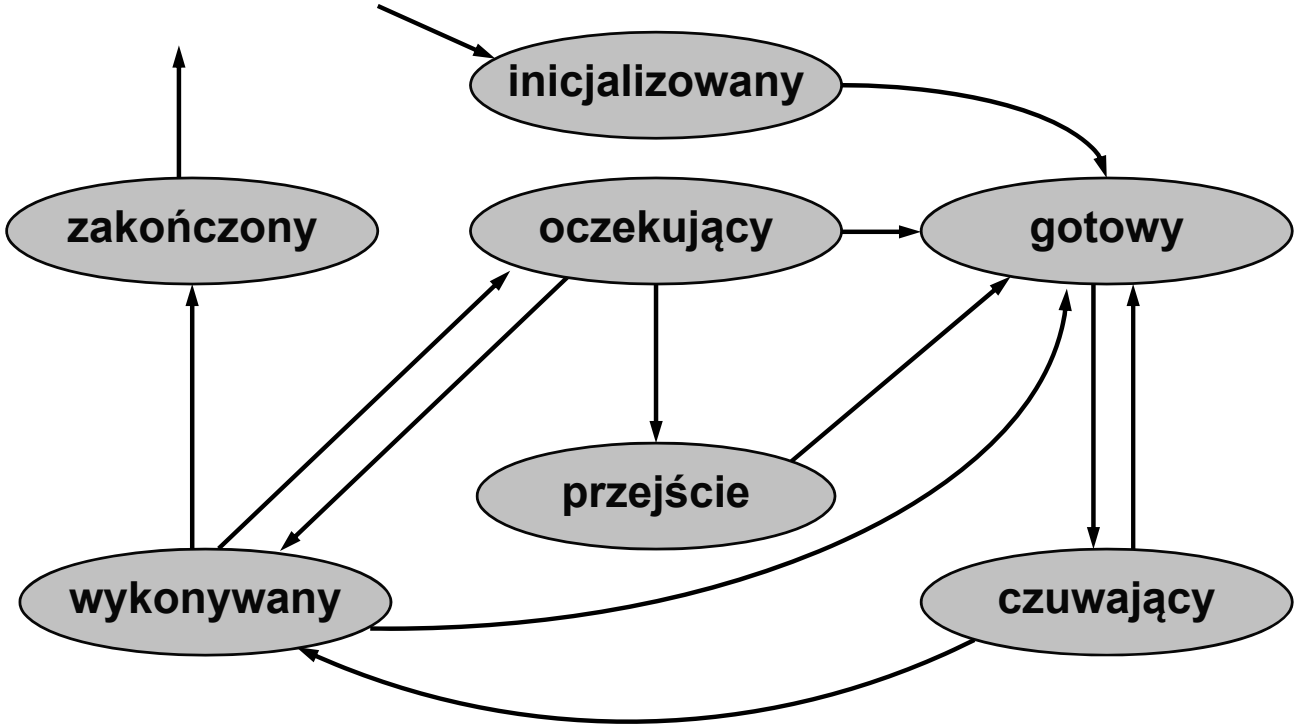
Proces i wątki w systemie Windows 2000/XP

- Proces stanowi środowisko do wykonywania wątków.
- Wątki korzystają z zasobów przydzielonych procesom (proces tworzy środowisko dla wykonywania wątków).
- Wątki (nie procesy) ubiegają się o przydział procesora i są szeregowane przez planistę krótkoterminowego.

Stany wątku w systemie Windows 2000/XP

- Inicjalizowany (initialized, wartość 0) — stan wewnętrzny w trakcie tworzenia wątku,
- Gotowy (ready, wartość 1) — oczekuje na przydział proc.,
- Wykonywany (running, wartość 2),
- Czuwający (standby, wartość 3) — wybrany do wykonania jako następny,
- Zakończony (terminated, wartość 4),
- Oczekujący (waiting, wart. 5) — oczekuje na zdarzenie,
- Przejście (transition, wartość 6) — oczekuje na sprowadzenie swojego stosu jądra z pliku wymiany,
- Unknown (wart. 7)

Cykl zmian stanów wątku w systemie Windows 2000/XP

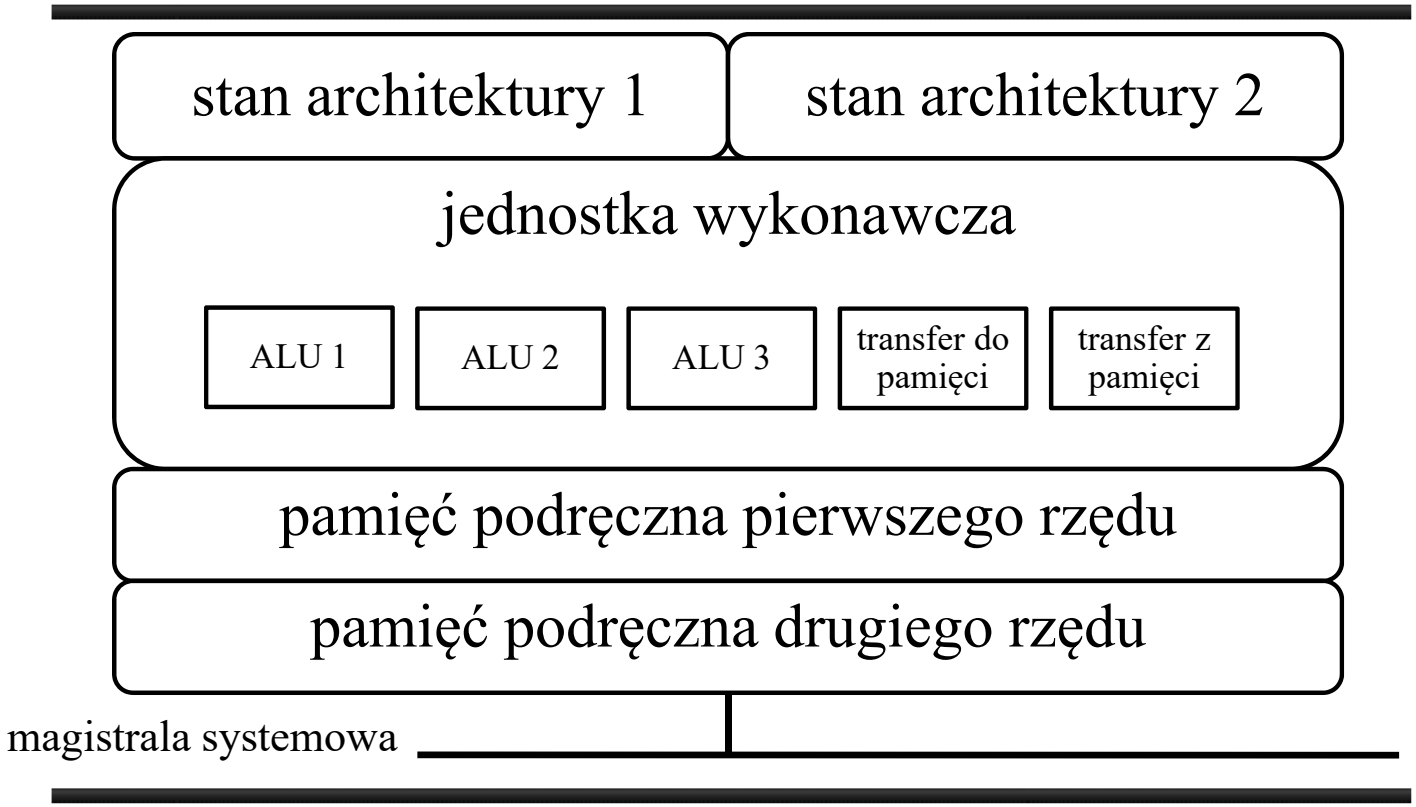


SUM SKiSR Oprogramowanie systemowe 87

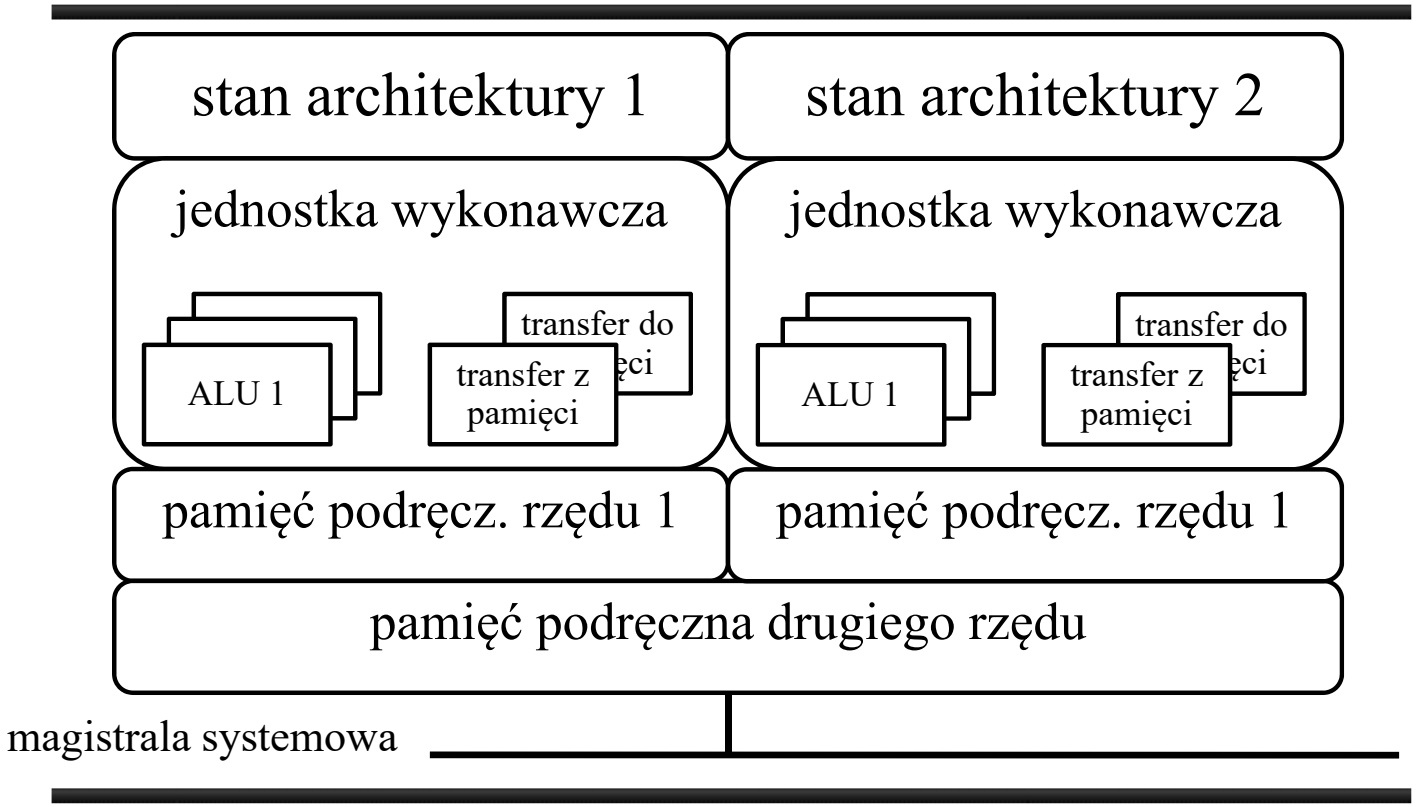
Wsparcie dla wielowątkowości na poziomie architektury procesora

- Hiperwątkowość — utrzymywanie w procesorze wielu (dwóch) kontekstów przetwarzania (tzw. stanu architektury) w celu efektywnego współdzielenia zasobów jednostki wykonawczej
- Wielordzeniowość — realizacja w jednym układzie dwóch jednostek wykonawczych, posiadających wspólny interfejs magistrali systemowej, współdzielących (w rozwiązaniach Intela) lub udostępniających (w rozwiązaniach AMD) pamięć podręczną drugiego poziomu

Hiperwątkowość



Wielordzeniowość



Klasyfikacja strategii planowanie przydziału procesora

- Strategie nie wywłaszczające — proces może kontynuować swoje działanie aż do zakończenia wykonywania ciągu instrukcji lub zgłoszenia do systemu żądania, którego spełnienie nie jest możliwe natychmiast.
- Strategie wywłaszczające — dopuszcza się odebranie procesowi procesora (przełączenie kontekstu) w wyniku decyzji podjętej poza procesem.

Ogólna koncepcja planowania

- Tryb decyzji — określa moment czasu, w którym oceniane i porównywane są priorytety procesów i dokonywany jest wybór procesu do wykonania.
- Funkcja priorytetu — funkcja zwracająca aktualny priorytet procesu na podstawie parametrów procesu i stanu systemu.
- Reguła arbitrażu — reguła rozwiązywania konfliktów pomiędzy procesami o tym samym priorytecie (np. losowo, w kolejności FIFO itp.).

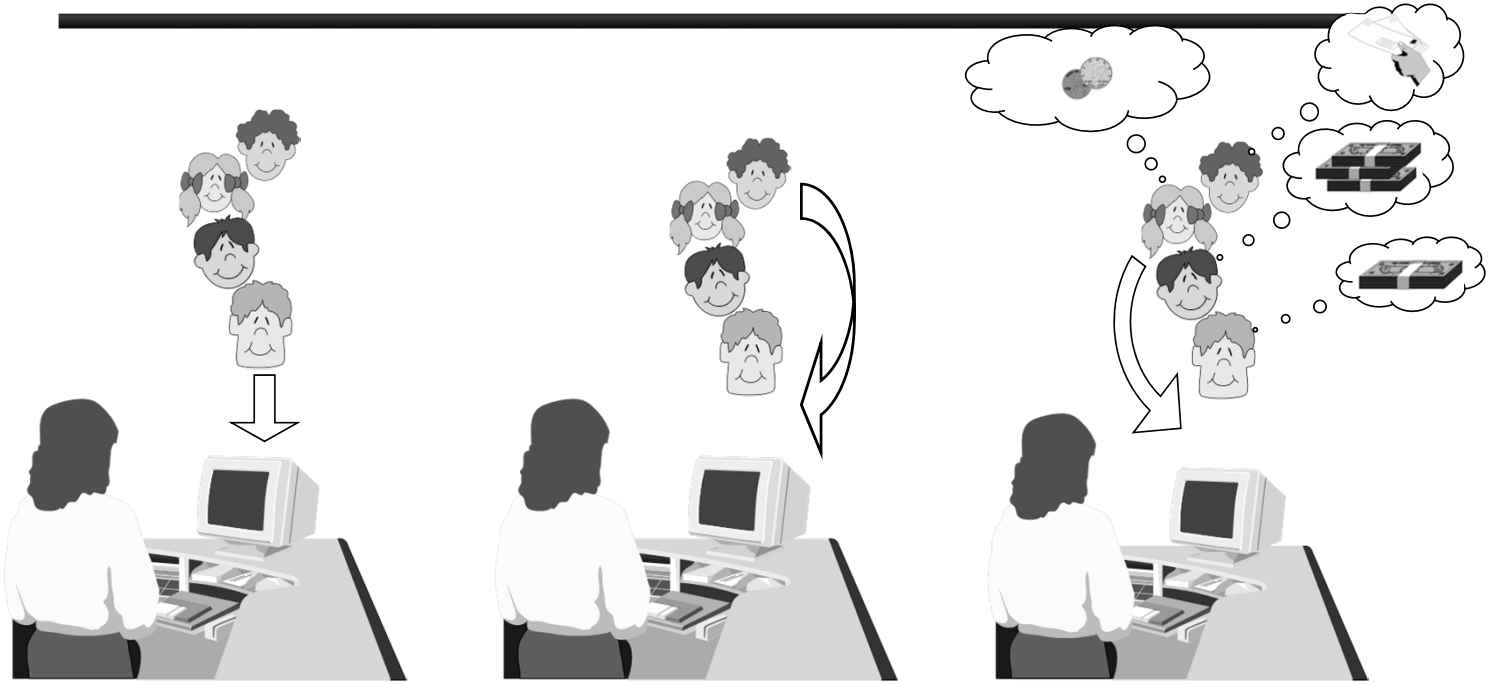
Kryteria planowania przydziału procesora

- Wykorzystanie procesora — procent czasu, przez który procesor jest zajęty pracą
- Przepustowość — liczba procesów kończonych w jednostce czasu
- Czas cyklu przetwarzania — czas pomiędzy przedłożeniem zadania, a zakończeniem jego wykonywania
- Czas oczekiwania — łączny czas spędzony przez proces w kolejce procesów gotowych
- Czas odpowiedzi — czas pomiędzy przedłożeniem zadania, a uzyskaniem pierwszej odpowiedzi

Algorytmy planowania niewywłaszczającego (1)

- FCFS (First Come First Served) — pierwszy zgłoszony, pierwszy obsłużony
- LCFS (Last Come First Served) — ostatni zgłoszony, pierwszy obsłużony
- SJF (SJN, SPF, SPN, Shortest Job/Process First/Next) — najpierw najkrótsze zadanie

Algorytmy planowania niewywłaszczającego (2)

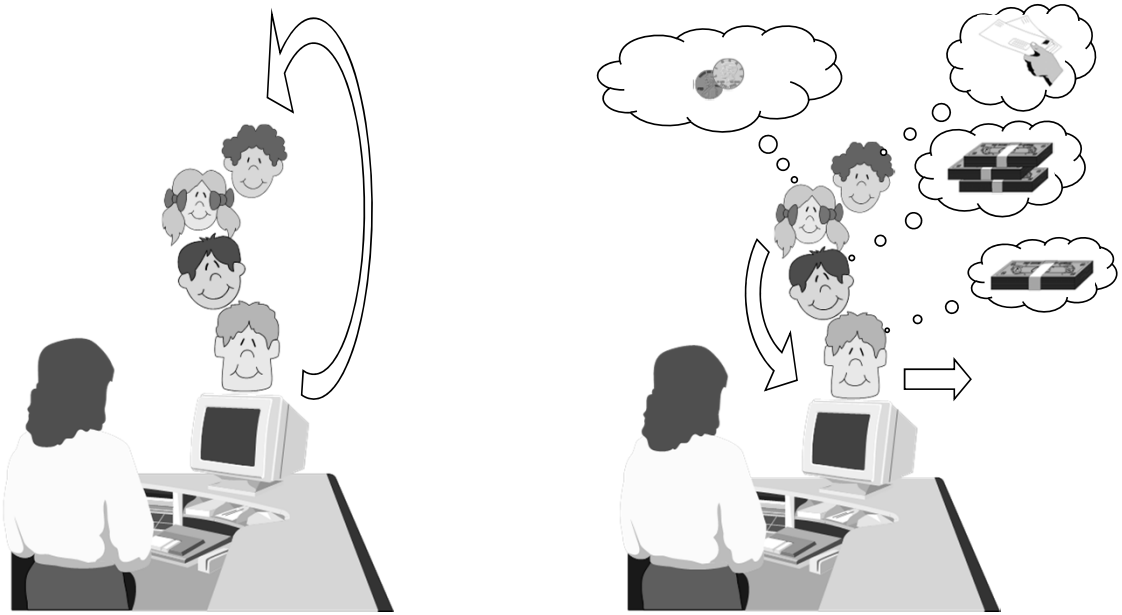


SUM SKiSR Oprogramowanie systemowe 95

Algorytmy planowania wywłaszczającego (1)

- Planowanie rotacyjne (ang. Round Robin, RR) — po ustalonym kwancie czasu proces wykonywany jest przerywany i trafia do kolejki procesów gotowych.
- SRT (Shortest Remaining Time) — najpierw zadanie, które ma najkrótszy czas do zakończenia.

Algorytmy planowania wywłaszczającego (2)

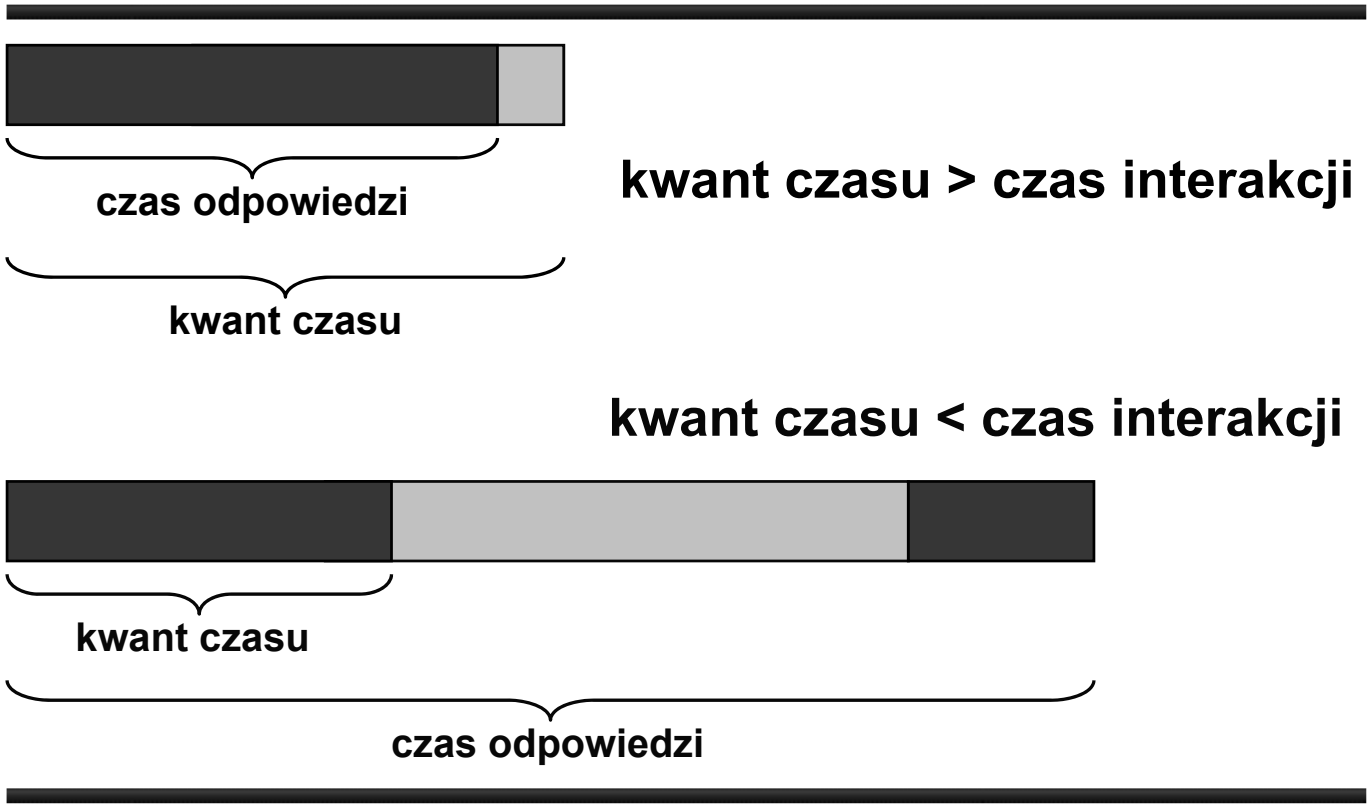


RR SRT

Algorytm RR — dobór kwantu czasu

- Krótki kwant czasu oznacza zmniejszenie czasu cyklu przetwarzania procesów krótkich, ale zwiększa narzut czasowy związany z obsługą przerwań i przełączaniem kontekstu.
- Z punktu widzenia interakcji z użytkownikiem kwant czasu powinien być trochę większy, niż czas potrzebny na typową interakcję.

Dobór kwantu czasu, a czas odpowiedzi systemu



Algorytmy SJF/SRT — estymacja czasu obsługi

- Średnia arytmetyczna
- Średnia wykładnicza

Definicja funkcji priorytetu

- Zdefiniować funkcję priorytetu dla algorytmów
  - FCFS
  - LCFS
  - SJN
  - SRT

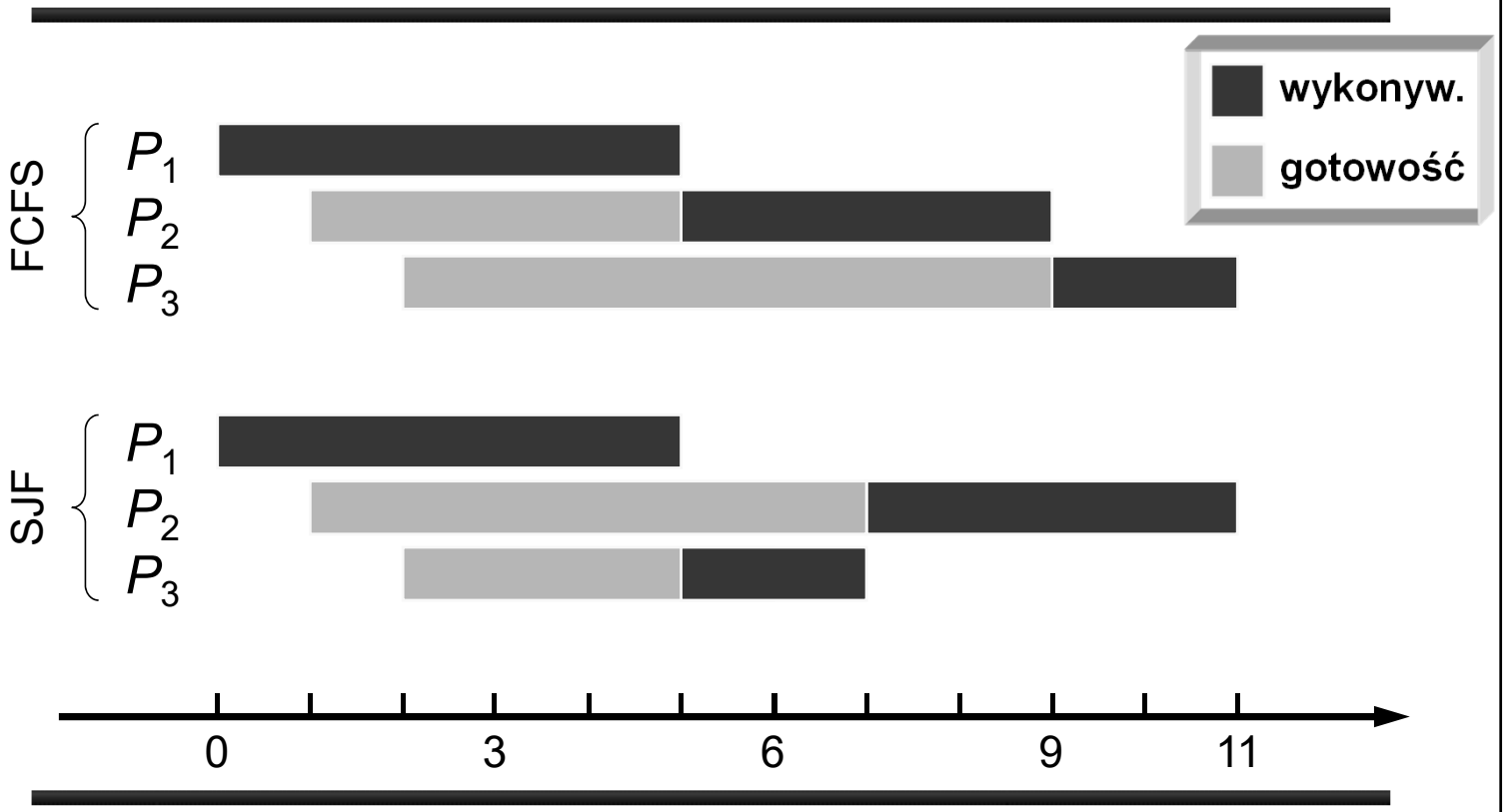
jako funkcję zmiennych (parametrów procesu)

- a — bieżący (dotychczasowy) czas obsługi
- r — rzeczywisty czas w systemie
- t — całkowity (do momentu zakończenia) czas obsługi

Własności algorytmów planowania

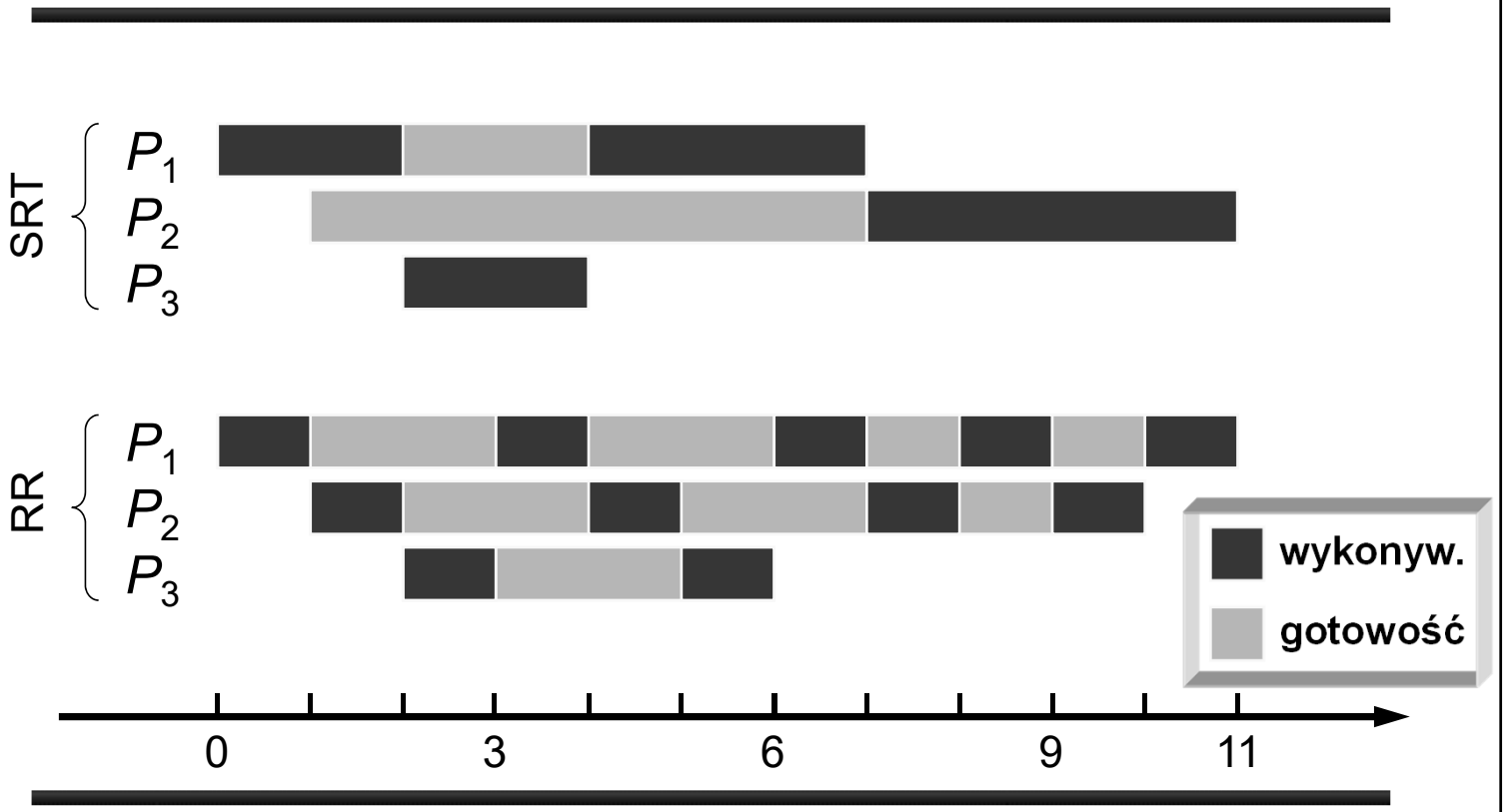
algorytm	priorytet	tryb decyzji	arbitraż
FIFO	*r*	niewywłaszczalny	losowy
LIFO	*-r*	niewywłaszczalny	losowy
SJN	*-t*	niewywłaszczalny	losowy lub chronologiczny
SRT	*a - t*	wywłaszczalny (przyjęcie proc.)	losowy lub chronologiczny
RR	stały	wywłaszczalny (kwant czasu)	cykliczny

Przykłady uszeregowania bez wywłaszczeń



SUM SKiSR Oprogramowanie systemowe 103

Przykłady uszeregowania z wywłaszczaniem



SUM SKiSR Oprogramowanie systemowe 104

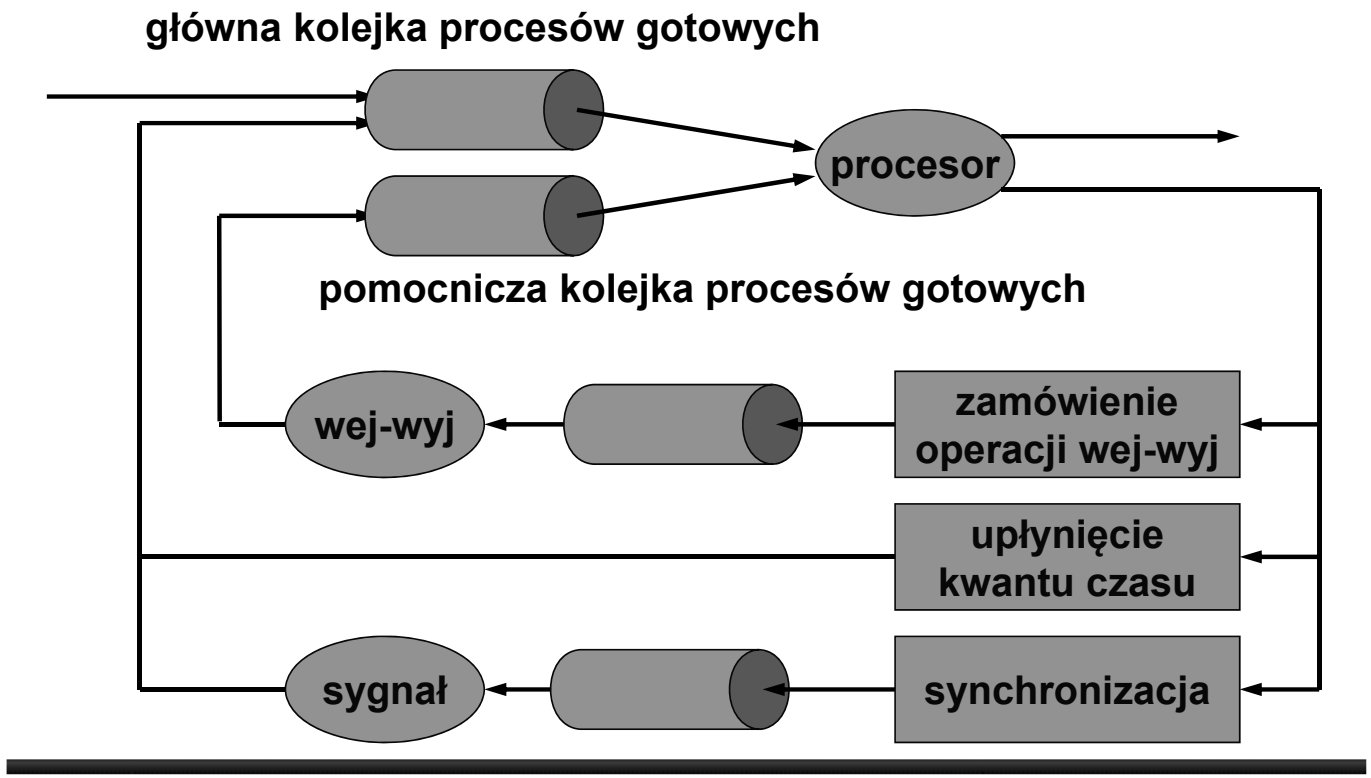
Planowanie wielokolejkowe

- Podział procesów na grupy (np. procesy interaktywne i procesy wsadowe) i wynikający z tego przydział do różnych kolejek procesów gotowych
- Możliwość przydziału różnych priorytetów oraz różnych algorytmów szeregowania do poszczególnych kolejek

Szeregowanie procesów, ograniczonych wejściem-wyjściem

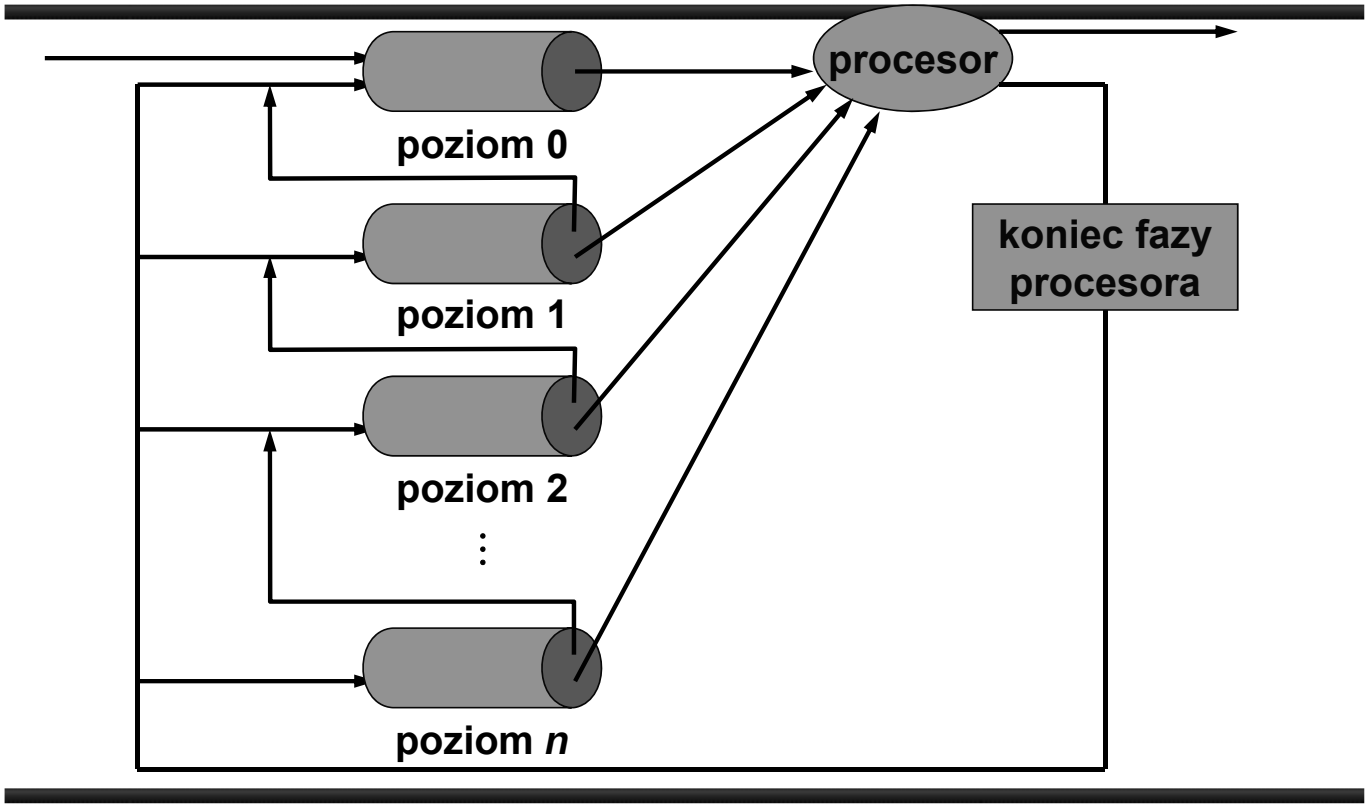
- Procesy ograniczone wejściem-wyjściem potrzebują niewiele czasu procesora, większość czasu w systemie spędzając na oczekiwaniu na urządzenia zewnętrzne.
- Opóźnianie przydziału procesora dla tego typu procesów powoduje zmniejszenie wykorzystania urządzeń zewnętrznych, a przydział — ze względu na nie długą fazę procesora — nie powoduje istotnego zwiększenia czasu oczekiwania innych procesów.
- Właściwym algorytmem byłby SJF lub SRT.
- Bezwzględna preferencja dla procesów oczekujących na gotowość urządzeń może spowodować głodzenie procesów ograniczonych procesorem.

Wirtualne planowanie rotacyjne (VRR)



SUM SKiSR Oprogramowanie systemowe 107

Wielopoziomowe kolejki ze sprzężeniem zwrotnym



Implementacja algorytmów planowania

- Z punktu widzenia przetwarzania użytkowego przełączanie kontekstu jest marnotrawstwem czasu procesora.
- Decyzja planisty krótkoterminowego musi zapaść w możliwie krótkim czasie.
- Struktury danych muszą dostarczyć informacji niezbędnych do dokonania szybkiego wyboru procesu o najwyższym priorytecie zgodnie z polityką planowania przydziału procesora (modelem matematycznym).

Synchronizacja procesów

- Abstrakcja programowania współbieżnego
- Instrukcje atomowe i ich przeplot
- Istota synchronizacji
- Kryteria poprawności programów współbieżnych
- Klasyfikacja mechanizmów synchronizacji
- Wzajemne wykluczania
- Semafory
- Mechanizmy synchronizacji wątków w standardzie POSIX
- Strukturalne mechanizmy synchronizacji

Wprowadzenie do abstrakcji przetwarzania współbieżnego

- Realizacja przetwarzania polega na wykonywaniu instrukcji przez jednostkę przetwarzającą (procesor).
- Instrukcje wykonywane są w kontekście jakiegoś procesu.
- Wykonanie instrukcji (akcja) oznacza zajście zdarzenia w procesie, skutkiem czego jest zmiana stanu procesu.
- Zajście zdarzenia jest zdeterminowane poprzedzającym je stanem procesu.
- Na stan procesu wpływają pośrednio akcje innych procesów w systemie, przejawiające się w stanie współdzielonych zasobów.

Podstawowe definicje i oznaczenia

- C — zbiór instrukcji (operacji), możliwych do wykonania przez jednostkę przetwarzającą (np. dodawanie, odejmowanie, skok, rozgałęzienie warunkowe)
- D — zbiór danych, przetwarzanych (modyfikowanych lub czytanych) w ramach wykonywania operacji przez jednostkę przetwarzającą
- $e_i(c, O)$ — wykonanie w ramach procesu P_i instrukcji atomowej $c \in C$ na operandach ze zbioru $O \subseteq D$

Stan procesu i zdarzenie

- Stan procesu obejmuje te elementy (wartości zmiennych, rejestrów, stan zasobów), które między innymi determinują następną instrukcję do wykonania.
- Wykonanie instrukcji określane jest jako *zdarzenie* lub *akcja*.
- Zbiór instrukcji do wykonania oraz zależności pomiędzy nimi określone są przez program procesu.
- Zajście zdarzenia zdeterminowane jest zatem przez program oraz bieżący stan procesu.

Proces sekwencyjny

- Proces (wątek) sekwencyjny jest wykonaniem ciągu instrukcji, opisanych przez program dla tego procesu (procedurę dla wątku), w taki sposób, że następna akcja nie rozpocznie się, zanim nie skończy się poprzednia.
- Zdarzenie (akcja) w procesie P_i oznacza zmianę stanu tego procesu, co formalnie opisane jest poprzez odwzorowanie (przejście, tranzycję):
  L: $S_i \times E_i \rightarrow S_i$, gdzie
  S_i — zbiór stanów procesu P_i
  E_i — zbiór zdarzeń w procesie P_i

Relacja lokalnego porządku

- Proces P_i jest ciągiem (skończonym lub nieskończonym) następujących po sobie naprzemiennie stanów i zdarzeń $s_i^0, e_i^1, s_i^1, e_i^2, s_i^2, ...,$ gdzie:
  s_i^0 — stan początkowy procesu P_i
  $\forall_{k\geq 0}\ s_i^k \in S_i$
  $\forall_{k>0}\ e_i^k \in E_i$
  $\forall_{k\geq 0}\ \mathrm{L}(s_i^k, e_i^{k+1}) = s_i^{k+1}$
- Relację porządku w procesie P_i — odzwierciedlająca kolejność stanów i zdarzeń w ciągu — nazywana będzie lokalnym porządkiem i oznaczana symbolem $\rightarrow_i$

Współbieżna realizacja zbioru procesów

- Zdarzenie w systemie współbieżnym, złożonym z procesów sekwencyjnych P_1, P_2, ..., P_n, oznacza zajście zdarzenia w jednym z procesów.
- Zdarzenie, zmieniając stan jednego procesu, zmienia stan całego systemu, co formalnie opisane jest poprzez odwzorowanie (przejście, tranzycję):
  G: $\Sigma \times \Lambda \rightarrow \Sigma$, gdzie
  $\Sigma \subseteq S_1 \times S_2 \times ... \times S_n$ — zbiór stanów systemu
  $\Lambda = E_1 \cup E_2 \cup ... \cup E_n$ — zbiór zdarzeń w systemie

Relacja globalnego porządku

- Przetwarzanie współbieżne zbioru procesów sekwencyjnych $P_1, P_2, ..., P_n$, jest ciągiem (skończonym lub nieskończonym) następujących po sobie naprzemiennie stanów systemu i zdarzeń
  $\sigma^0, e^1, \sigma^1, e^2, \sigma^2, ...,$ gdzie:
  σ^0 — stan początkowy $\langle s_1{}^0, s_2{}^0, ... , s_n{}^0 \rangle$
  $\forall_{k\geq 0}\ \sigma^k \in \Sigma$
  $\forall_{k>0}\ e^k \in \Lambda$
  $\forall_{k \geq 0}\ \mathrm{G}(\sigma^k, e^{k+1}) = \sigma^{k+1}$
- Relacja porządku w systemie — odzwierciedlająca kolejność stanów i zdarzeń w ciągu — nazywana będzie globalnym porządkiem i oznaczana symbolem $\rightarrow$

Niedeterminizm przetwarzania

- Zdarzenie, które może pojawić się w określonym stanie przetwarzania, określane jest jako zdarzenie dopuszczalne.
- W przetwarzaniu współbieżnym z każdym sekwencyjnym procesem w danym stanie związane jest jedno zdarzenie. W stanie całego przetwarzania jest zatem zbiór zdarzeń dopuszczalnych.
- W zależności od dostępności zasobów (procesora, magistrali systemowej) oraz decyzji planisty, wykonywany będzie jeden z procesów gotowych. Należy więc przyjąć, że zajście jednego ze zdarzeń dopuszczalnych ma charakter losowy.

Przeplot i osiągalność stanu

- Przeplotem jest zbiór zdarzeń Λ, uporządkowany przez relację globalnego porządku $\rightarrow$, spełniającą następujący warunek:

$$\forall_{e,\, e' \in \Lambda}\ \exists_i\ e \rightarrow_i e' \Rightarrow e \rightarrow e'$$

- Stan σ' systemu jest osiągalny ze stanu σ, co będzie oznaczone przez $\sigma \rightsquigarrow \sigma'$, jeśli zachodzi jeden z warunków:
  1) są to te same stany, czyli $\sigma \equiv \sigma'$
  2) $\exists_{e \in \Lambda}\ \mathrm{G}(\sigma, e) = \sigma'$
  3) $\exists_{\sigma'' \in \Sigma}\ \sigma \rightsquigarrow \sigma'' \wedge \sigma'' \rightsquigarrow \sigma'$

Procesy niezależne a procesy współpracujące

- Niech $D_i \subseteq D$ oznacza zbiór danych przetwarzanych przez proces P_i, czyli dla ciągu instrukcji procesu P_i: $e_i^1(c_i^1, O_i^1), e_i^2(c_i^2, O_i^2), \ldots$

$$D_i = \bigcup_k O_i^k$$

- Dwa procesy, P_i i P_j, są **niezależne**, jeśli

$$D_i \cap D_j = \varnothing$$

- Dwa procesy, P_i i P_j, **współpracują** (są w interakcji), jeśli

$$D_i \cap D_j \neq \varnothing$$

Dane współdzielone a lokalne

- Dane lokalne (prywatne) procesu P_i to takie, które są przetwarzane wyłącznie przez P_i, formalnie zatem jest to zbiór:

$$D_i \setminus \bigcup_{1 \le j \le n, j \neq i} D_j$$

- Dane współdzielone przez proces P_i to takie, które przetwarzane są oprócz P_i przez co najmniej jeszcze jeden inny proces, formalnie zatem jest to zbiór:

$$D_i \cap \bigcup_{1 \le j \le n, j \neq i} D_j$$

Dane wejściowy i wyjściowe

- Niech $D_i^R \subseteq D_i$ oznacza zbiór danych czytanych przez proces P_i, a $D_i^M \subseteq D_i$ oznacza zbiór danych modyfikowanych przez proces P_i
- Dane wejściowe procesu P_i to takie dane współdzielone, które są czytane przez proces P_i, formalnie zatem jest to zbiór:

$$D_i^R \cap \bigcup_{1 \le j \le n, j \neq i} D_j$$

- Dane wyjściowe procesu P_i to takie dane współdzielone, które są modyfikowane przez proces P_i, formalnie zatem jest to zbiór:

$$D_i^M \cap \bigcup_{1 \le j \le n, j \neq i} D_j$$

Przykład przetwarzania współbieżnego

```
n: integer := 0; /* zmienna współdzielona */
```

instrukcje \ wątki	wątek A	wątek B
wysokopoziomowe	`n := n + 1`	`n := n + 1`

RISC	`load RA, n`
	`add RA, 1`
	`store RA, n`
		`load RB, n`
		`add RB, 1`
		`store RB, n`

CISC	`inc n`	`inc n`

Przykład przeplotu instrukcji RISC

	przeplot 1	przeplot 2

przepływ sterowania	`{A} load` R_A`, n`
	`{A} add` R_A`, 1`
	`{A} store` R_A`, n`
	`// n = 1`
	`{B} load` R_B`, n`
	`{B} add` R_B`, 1`
	`{B} store` R_B`, n`
		`{A} load` R_A`, n`
		`{A} add` R_A`, 1`
		`// n = 0`
		`{B} load` R_B`, n`
		`{B} add` R_B`, 1`
		`{A} store` R_A`, n`
		`{B} store` R_B`, n`

wartość *n*	2	1

Przykład przeplotu instrukcji CISC

	przeplot 1	przeplot 2
przepływ sterowania	`{A} inc n` `{B} inc n`	`{B} inc n` `{A} inc n`
wartość *n*	2	2

Istota synchronizacji

- Celem synchronizacji jest kontrola przepływu sterowania pomiędzy procesami tak, żeby dopuszczalne stały się tylko przeploty instrukcji zgodne z intencją programisty.
- Synchronizacja na najniższym poziomie polega na wykonaniu specjalnych instrukcji, które powodują zatrzymanie postępu przetwarzania.
- Synchronizacja na wyższym poziomie polega na użyciu specjalnych konstrukcji programotwórczych lub odpowiednich definicji struktur danych.

Poprawność programów współbieżnych

- Własność bezpieczeństwa (ang. safety) — w każdym stanie przetwarzania muszą być spełnione pewne warunki.
- Własność żywotności (ang. liveness) — w wyniku przetwarzania muszą w końcu zajść pewne warunki.

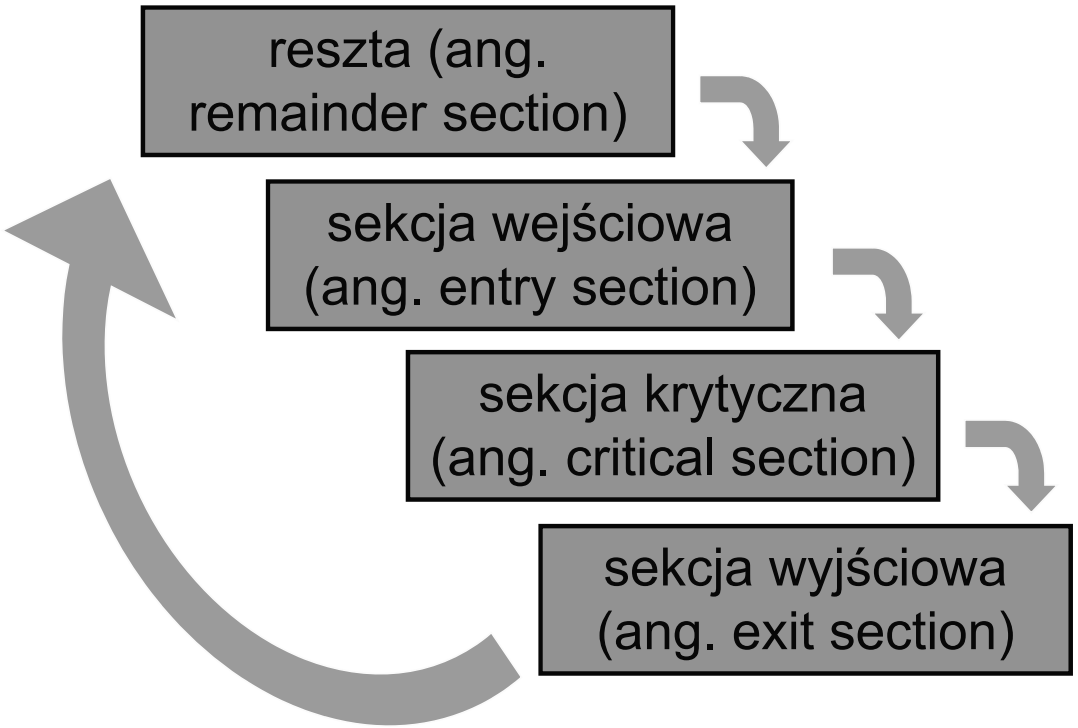
Klasyfikacja mechanizmów synchronizacji

- Zapis lub odczyt współdzielonych danych
- Złożone operacje atomowe na współdzielonych danych (np. **`test&set`**, **`exchange`**)
- Mechanizmy wspierane przez system operacyjny
  - semafory
  - mechanizmy POSIX (zamki oraz zmienne warunkowe)
- Mechanizmy strukturalne (wspierane przez wysokopoziomowe języki programowania)
  - monitory
  - regiony krytyczne

Wzajemne wykluczanie — sformułowanie problemu

- W systemie działa n procesów P_0, P_1, ..., P_{n-1}.
- W programie każdego procesu znajduje się fragment kodu zwany sekcją krytyczną (ang. critical section).
- Sekcja krytyczna wykonywana jest w danej chwili przez co najwyżej jeden proces.

Ogólna postać algorytmu wzajemnego wykluczania



SUM SKiSR Oprogramowanie systemowe 131

Poprawność rozwiązania problemu wzajemnego wykluczania

- Wzajemne wykluczanie — warunek bezpieczeństwa,
- Postęp (ang. progress) — warunek żywotności z punktu widzenia systemu,
- Ograniczone czekanie (ang. lockout-freedom) — warunek żywotności z punktu widzenia procesu.

Wzajemne wykluczanie 2 procesów — podejście 1

Algorytm dla procesu P_i przy dwóch procesach P_i i P_j

```
shared numer: Integer := i;

while numer ≠ i do
   nic;                          } sekcja wejściowa
sekcja krytyczna_i;
numer := j;                      ← sekcja wyjściowa
reszta_i;
```

Wzajemne wykluczanie 2 procesów — podejście 2

Algorytm dla procesu P_i przy dwóch procesach P_i i P_j
(dla uproszczenia prezentacji założono, że $i = 0$, a $j = 1$)

```
shared znacznik: array [0..1] of
                 Boolean := false;

znacznik[i] := true;
while znacznik[j] do
   nic;
sekcja krytyczna_i;
znacznik[i] := false;
reszta_i;
```

sekcja wejściowa

← *sekcja wyjściowa*

Wzajemne wykluczanie 2 procesów — podejście 3

```
shared znacznik: array [0..1] of
                    Boolean := false;
znacznik[i] := true;
while znacznik[j] do
begin
   znacznik[i] := false;
   znacznik[i] := true;
end;
sekcja krytyczna_i;
znacznik[i] := false;
reszta_i;
```

sekcja wejściowa

← *sekcja wyjściowa*

Wzajemne wykluczanie 2 procesów — podejście 4

```
shared znacznik: array [0..1] of Boolean;
shared numer:    Integer;

znacznik[i] := true;
numer := j;
while znacznik[j] and numer ≠ i do
   nic;
sekcja krytyczna_i;
znacznik[i] := false;
reszta_i;
```

sekcja wejściowa

← *sekcja wyjściowa*

Operacja `test&set`

```
function test&set(var l: Boolean):
  Boolean;
   begin
      test&set := l;
      l := true;
   end;
```

Wzajemne wykluczanie z użyciem instrukcji `test&set`

```
shared zamek: Boolean;

while test&set(zamek) do
    nic;                          } sekcja wejściowa
sekcja krytyczna;
zamek := false;                   ← sekcja wyjściowa
reszta;
```

Semafory

- Semafor jest zmienną całkowitą nieujemną lub — w przypadku semaforów binarnych — zmienną typu logicznego.
- Na semaforze można wykonywać dwa rodzaje operacji:
  - P — opuszczanie semafora (hol. proberen)
  - V — podnoszenie semafora (hol. verhogen)
- Synchronizacja polega na blokowaniu procesu w operacji opuszczania semafora, jeśli semafor jest już opuszczony.

Rodzaje semaforów (1)

- Semafor binarny — zmienna semaforowa przyjmuje tylko wartości true (stan podniesienia, otwarcia) lub false (stan opuszczenia, zamknięcia).
- Semafor ogólny (zliczający) — zmienna semaforowa przyjmuje wartości całkowite nieujemne, a jej bieżąca wartość jest zmniejszana lub zwiększana o 1 w wyniku wykonania odpowiednio operacji opuszczenia lub podniesienia semafora.

Rodzaje semaforów (2)

- Semafor uogólniony — semafor zliczający, w przypadku którego zmienną semaforową można zwiększać lub zmniejszać o dowolną wartość, podaną jako argument operacji.
- Semafor dwustronnie ograniczony — semafor ogólny, w przypadku którego zmienna semaforowa, oprócz dolnego ograniczenia wartością 0, ma górne ograniczenie, podane przy definicji semafora.

Implementacja semafora ogólnego na poziomie maszynowym

- Opuszczanie semafora

```
procedure P(var s: Semaphore)
   begin
      while s = 0 do nic;
      s := s - 1;
   end;
```

instrukcje muszą być wykonane atomowo

- Podnoszenie semafora

```
procedure V(var s: Semaphore)
   begin
      s := s + 1;
   end;
```

instrukcja musi być wykonana atomowo

Implementacja semafora binarnego na poziomie maszynowym

- Opuszczanie semafora

```
procedure P(var s: Binary_Semaphore)
   begin
      while not s do nic;
      s := false;
   end;
```

instrukcje muszą być wykonane atomowo

- Podnoszenie semafora

```
procedure V(var s: Binary_Semaphore)
   begin
      s := true;
   end;
```

instrukcja musi być wykonana atomowo

Implementacja semafora ogólnego na poziomie systemu operacyjn. (1)

- Struktury danych

```
type Semaphore = record
    wartość: Integer;
    L: list of Proces;
end;
```

- Opuszczanie semafora

```
procedure P(var s: Semaphore) begin
    s.wartość := s.wartość - 1;
    if s.wartość < 0 then begin
        dołącz dany proces do  s.L
        zmień stan danego procesu na „oczekujący”
    end;
end;
```

Implementacja semafora ogólnego na poziomie systemu operacyjn. (2)

☞ Podnoszenie semafora

```
procedure V(var s: Semaphore)
  begin
    s.wartość := s.wartość + 1;
    if s.wartość ≤ 0 then begin
      wybierz i usuń jakiś/kolejny proces z kolejki s.L
      zmień stan wybranego procesu na „gotowy”
    end;
 end;
```

Wzajemne wykluczanie z użyciem semaforów

```
shared mutex: Semaphore := 1;
P(mutex);          ← sekcja wejściowa
sekcja krytyczna;
V(mutex);          ← sekcja wyjściowa
reszta;
```

Zakleszczenie (ang. deadlock)

1. Zasoby i model systemu
2. Warunki konieczne wystąpienia zakleszczenia
3. Graf przydziału zasobów
4. Przeciwdziałanie zakleszczeniu
   - ignorowanie problemu zakleszczenia
   - zapobieganie zakleszczeniom
   - unikanie zakleszczeń
   - wykrywanie zakleszczeń
   - likwidowanie zakleszczeń

Model systemu

- System składa się z zasobów *m* różnych typów (rodzajów) ze zbioru $Z = \{Z_1, Z_2, ..., Z_m\}$.
- Zasób każdego typu może być reprezentowany przez wiele jednorodnych jednostek (egzemplarzy).
- O zasoby rywalizują procesy ze zbioru $P = \{P_1, P_2, ..., P_n\}$
- Klasyfikacja zasobów z punktu widzenia problemu zakleszczenia:
  - zasoby odzyskiwalne (zwrotne, trwałe, ang. reusable resources)
  - zasoby nieodzyskiwalne (zużywalne, niezwrotne, ang. consumable resources)

Zasoby odzyskiwalne

- Liczba jednostek zasobów odzyskiwalnych jest ustalona.
- Zasoby odzyskiwalne po ich zwolnieniu przez jakiś proces mogą zostać ponownie użyte przez inny proces.
- Proces ubiega się o dowolny egzemplarz zasobu odzyskiwalnego według następującego schematu:
  1. zamówienie (ewentualnie oczekiwanie na realizację)
  2. użycie — korzystanie zasobu (jego przetrzymywanie)
  3. zwolnienie — oddanie zasobu do systemu
- Przykłady zasobów odzyskiwalnych: procesor, pamięć, kanał wejścia-wyjścia.

Zasoby nieodzyskiwalne

- Jednostki zasobu nieodzyskiwalnego są tworzone przez jakiś proces, a następnie zużywane (tym samym usuwane) przez inny proces.
- Nie ma ograniczenia na liczbę tworzonych jednostek zasobu.
- Liczba aktualnie dostępnych jednostek jest skończona i może się zmieniać w czasie w wyniku zmian stanu systemu.
- Przykłady zasobów nieodzyskiwalnych: kod znaku z klawiatury, sygnał lub komunikat przekazany do procesu.

Korzystanie z zasobów nieodzyskiwalnych

- Proces ubiega się o dowolny egzemplarz zasobu nieodzyskiwalnego według następującego schematu:
  - 1. zamówienie (ewentualnie oczekiwanie na realizację)
  - 2. zużycie — wykorzystanie zasobu (jego usunięcie)
- Proces może wyprodukować i przekazać zasób do systemu.

Warunki konieczne wystąpienia zakleszczenia

- Wzajemne wykluczanie — przynajmniej jeden zasób musi być niepodzielny, czyli niedopuszczalne jest współbieżne używanie egzemplarza tego zasobu przez dwa procesy.
- Przetrzymywanie i oczekiwanie — proces, któremu przydzielono jakieś jednostki, oczekuje na dodatkowe jednostki blokowane przez inny proces.
- Brak wywłaszczeń — jednostki zasobu zwalniane są tylko z inicjatywy odpowiednich procesów.
- Cykl w oczekiwaniu — istnieje podzbiór $\{P_1, ..., P_k\} \subseteq P$ taki, że P_1 czeka na jednostkę zasobu przetrzymywaną przez P_2, P_2 na jednostkę przetrzymywany przez P_3, ..., P_k czeka na jednostkę przetrzymywany przez P_1.

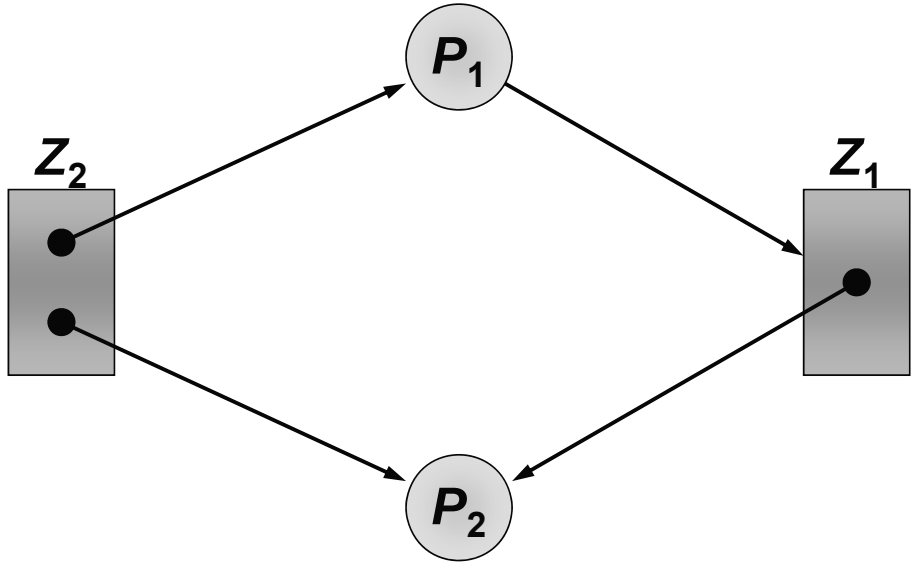
Warunki konieczne w odniesieniu do zasobów nieodzyskiwalnych

- Wzajemne wykluczanie — jednostka zasobu może być zużyta przez jeden proces.
- Przetrzymywanie i oczekiwanie — w stanie oczekiwania proces nie produkuje jednostek zasobów.
- Brak wywłaszczeń — nie można zmusić procesu do wyprodukowania jednostki zasobu lub zrobić to za niego.
- Cykl w oczekiwaniu — istnieje podzbiór $\{P_1, ..., P_k\} \subseteq P$ taki, że P_1 czeka na wyprodukowanie jednostki zasobu przez P_2, P_2 czeka wyprodukowanie jednostki przez P_3, ..., P_k czeka na wyprodukowanie jednostki przez P_1.

Reprezentacja stanu systemu — graf przydziału zasobów odzyskiwalnych

- Zbiór wierzchołków obejmuje procesy (reprezentowane przez kółka) i zasoby (reprezentowane przez prostokąty) czyli $W = P \cup Z$.
- Egzemplarze danego zasobu reprezentowane przez kropki wewnątrz prostokąta.
- Zbiór skierowanych krawędzi (łuków) obejmuje
  - krawędzie zamówienia (ang. request edge) $P_i \rightarrow Z_j$
  - krawędzie przydziału (ang. assignment edge) $Z_j \rightarrow P_i$

Przykład grafu zasobów odzyskiwalnych



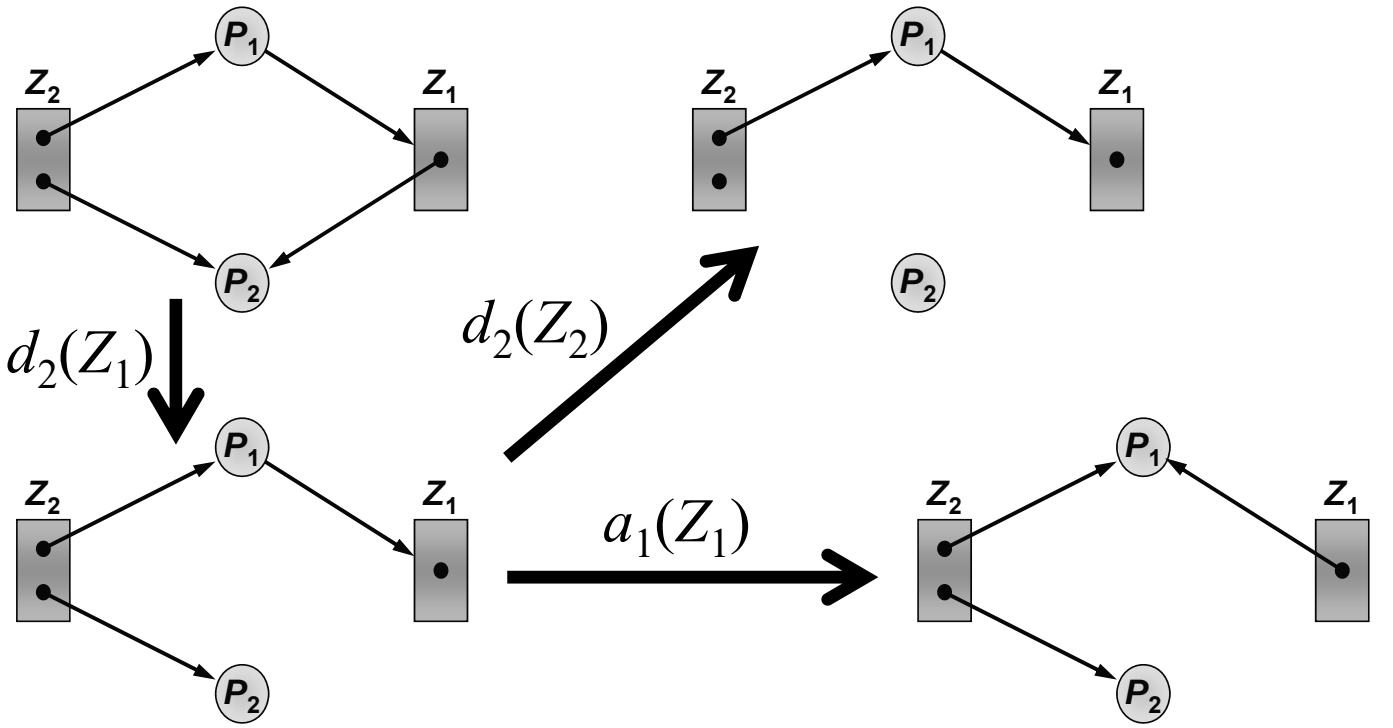
Zdarzenia w systemie z zasobami odzyskiwalnymi

- Zamówienie (ang. request) jednostki zasobu przez procesu P_i — r_i
- Nabycie (ang. acquisition) jednostki zasobu przez proces P_i — a_i
- Zwolnienie (ang. release) jednostki zasobu przez proces P_i — d_i

Zmiana stanu systemu a graf zasobów odzyskiwalnych

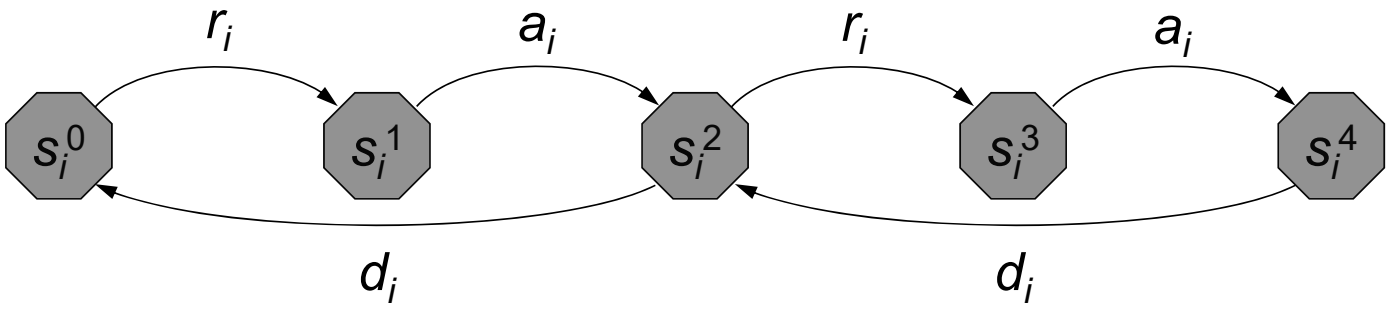
- W wyniku zamówienia jednostki zasobu Z_j przez proces P_i w grafie pojawia się krawędź zamówienia $P_i \rightarrow Z_j$.
- Realizacja zamówienia może nastąpić wówczas, gdy są wolne jednostki żądanego zasobu, a jej wynikiem jest zmiana kierunku krawędzi żądania, tym samym zamiana na krawędź przydziału $Z_j \rightarrow P_i$.
- W wyniku zwolnienia jednostka zasobu jest odzyskiwana przez system a krawędź przydziału znika.

Przykład przejść pomiędzy stanami w przypadku zasobów odzyskiwalnych

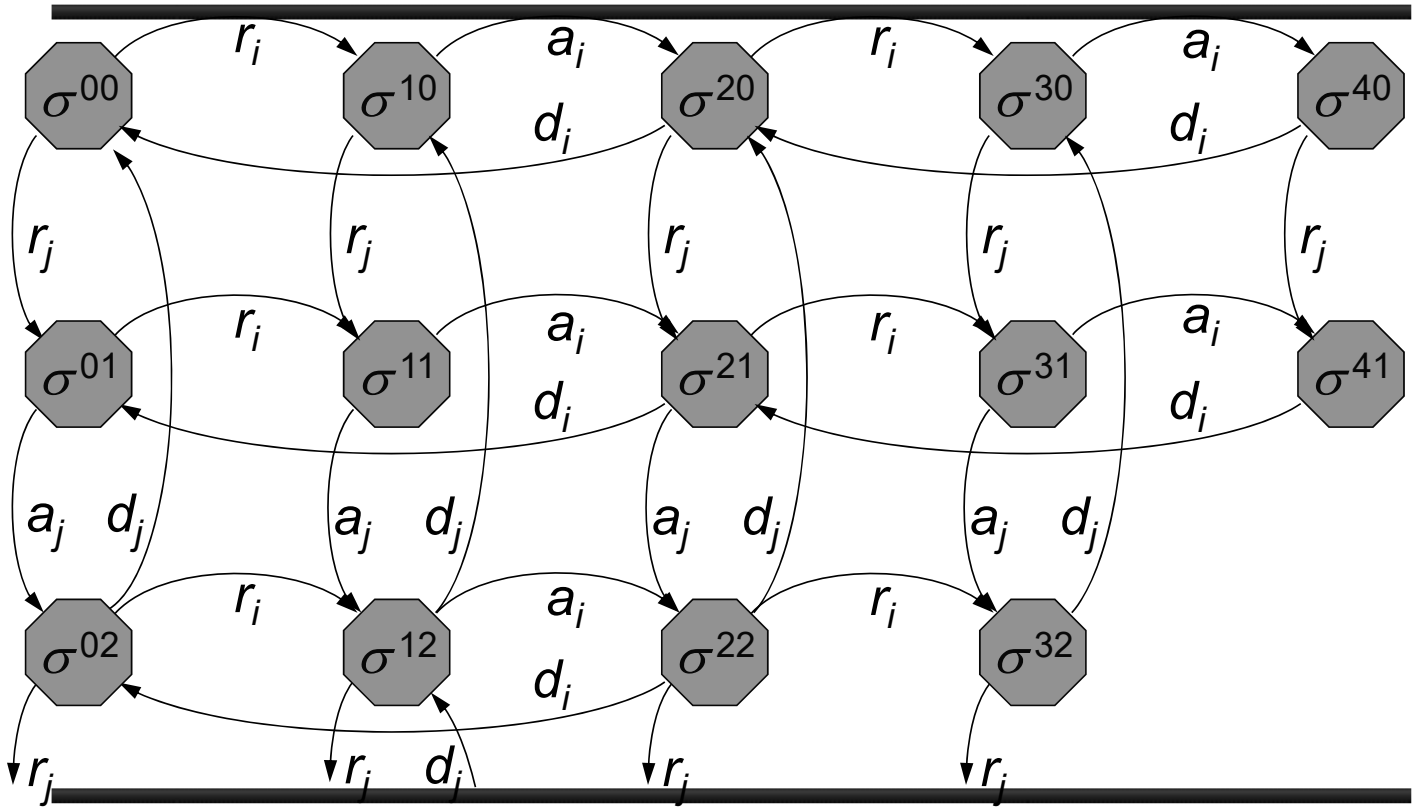


SUM SKiSR Oprogramowanie systemowe 201

Przykład przejść procesu w systemie z dwoma jednostkami zasobu

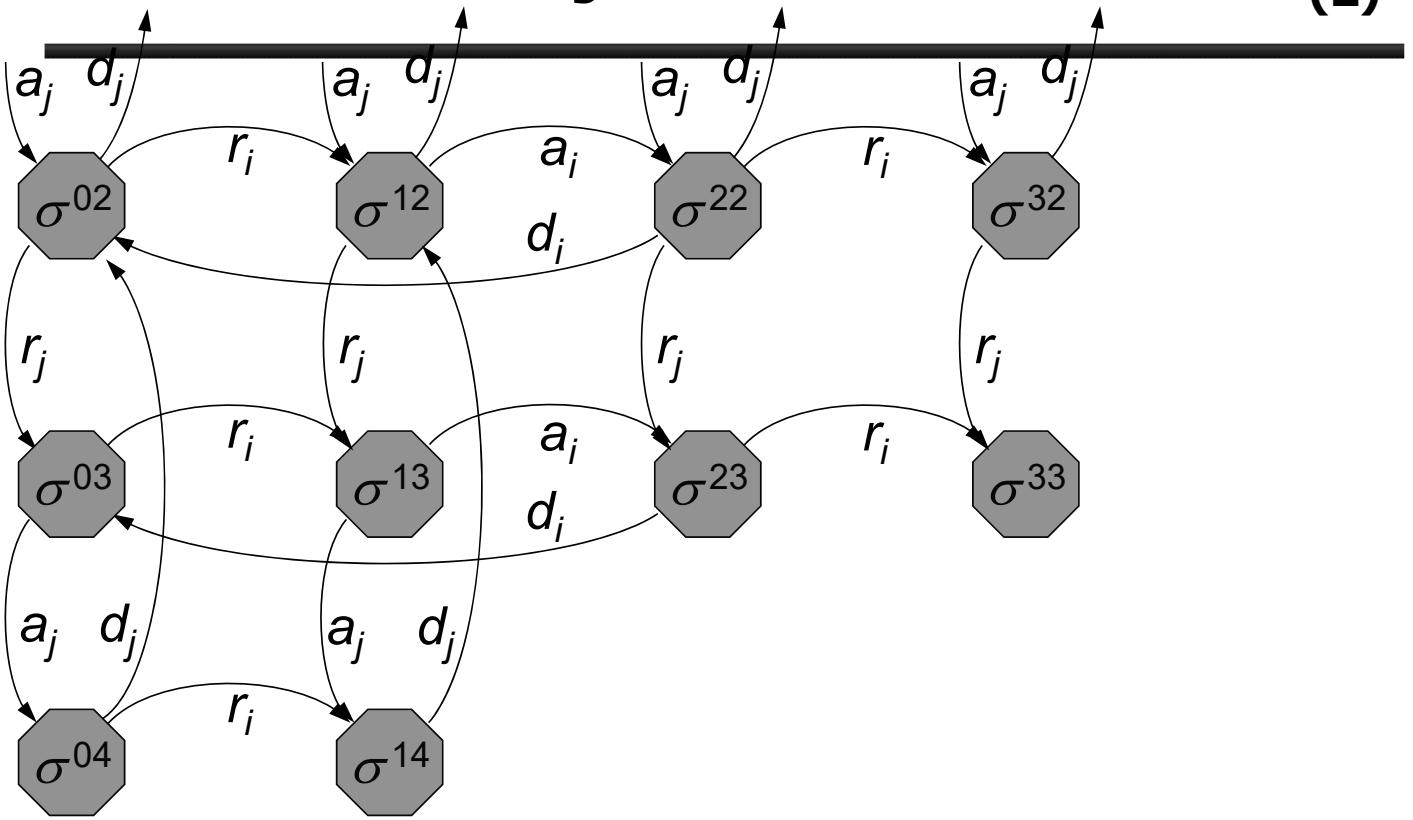


Przykład przejść 2 procesów w sys. z dwoma jednostkami zasobu (1)



SUM SKiSR Oprogramowanie systemowe 203

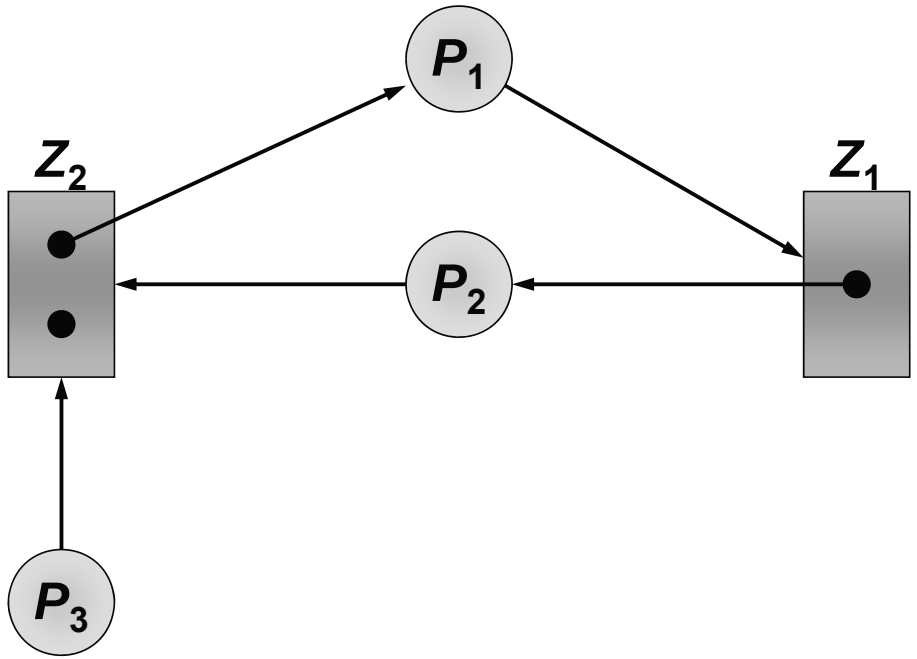
Przykład przejść 2 procesów w sys. z dwoma jednostkami zasobu (2)



Definicja zakleszczenia procesu

- Proces P_i jest wstrzymany (zablokowany) w stanie systemu σ, jeśli wszystkie dopuszczalne zdarzenia w systemie mają miejsce w innych procesach niż P_i.
- Proces P_i jest zakleszczony w stanie σ, jeśli jest wstrzymany w stanie σ i w każdym stanie osiągalnym ze stanu σ.

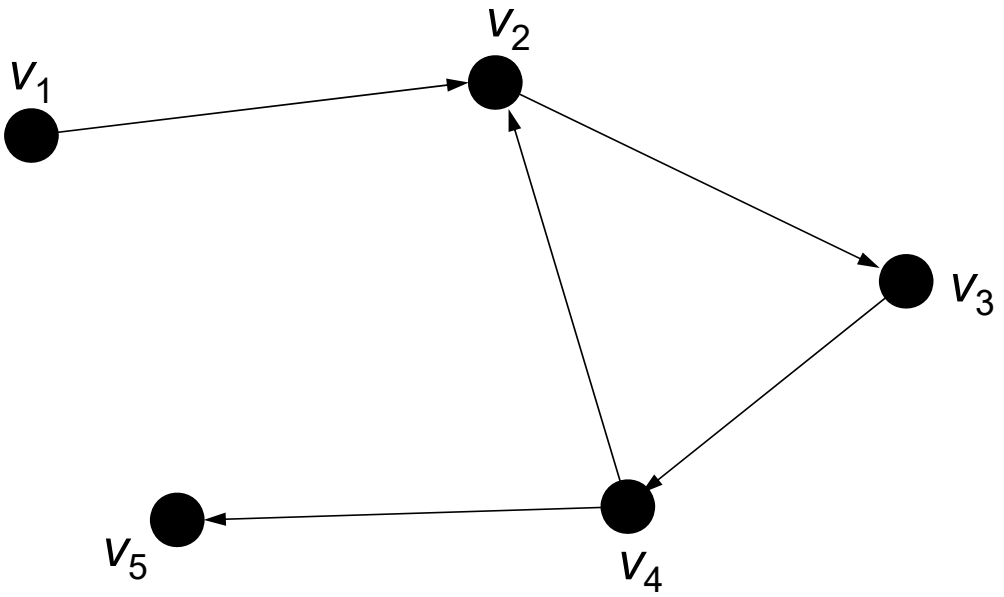
Przydział natychmiastowy



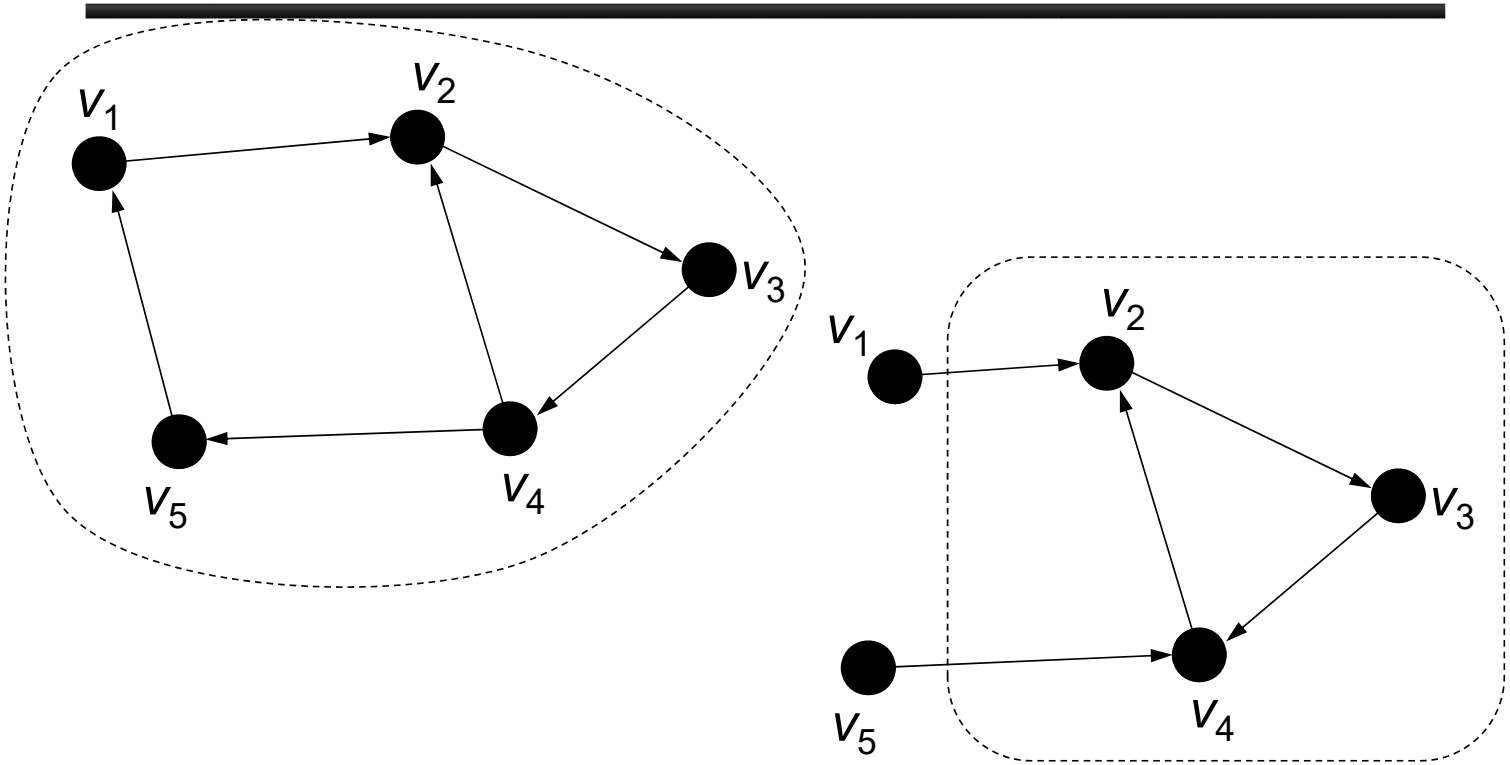
Własności grafów

- Dany jest graf skierowany $G = (N, E)$, gdzie
  N — zbiór wierzchołków,
  $E \subseteq N \times N$ — zbiór łuków (skierowanych krawędzi).
- Niech dla $v \in N$ zdefiniowany będzie zbiór
  $O(v) = \{u \in N: (v,u) \in E\} \cup \{u \in N: \exists_{w \in N} (v,w) \in E \wedge u \in O(w)\}$
- W grafie G występuje *cykl*, gdy:
  $\exists_{v \in N} v \in O(v)$
- W grafie występuje *supeł* (węzeł, zatoka, ang. knot), gdy:
  $\exists_{N' \subseteq N} \forall_{v \in N'} (\forall_{u \in N'} u \in O(v) \wedge \forall_{u \in N \setminus N'} u \notin O(v))$

Przykład cyklu w grafie

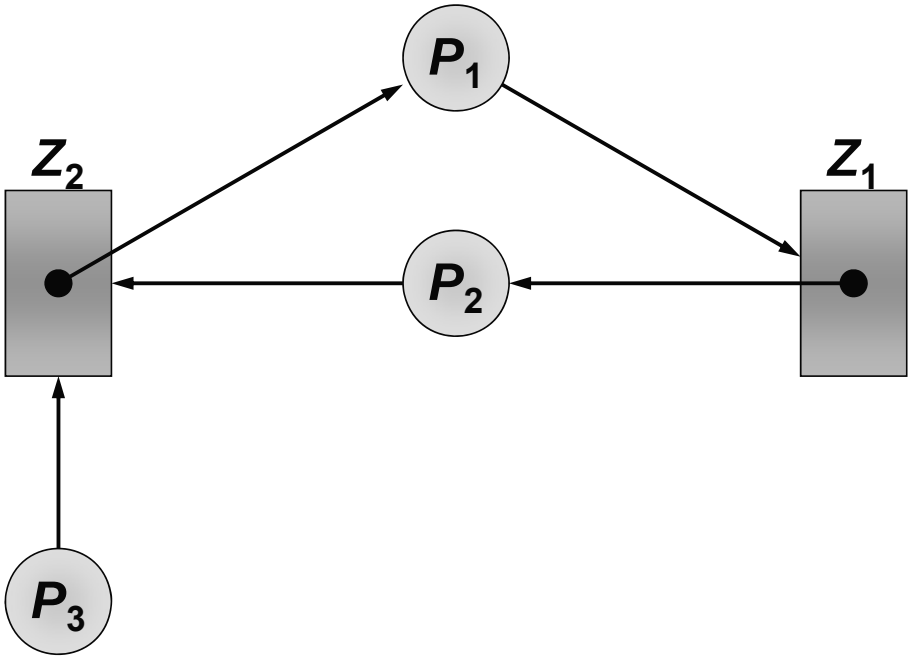


Przykłady supła w grafie

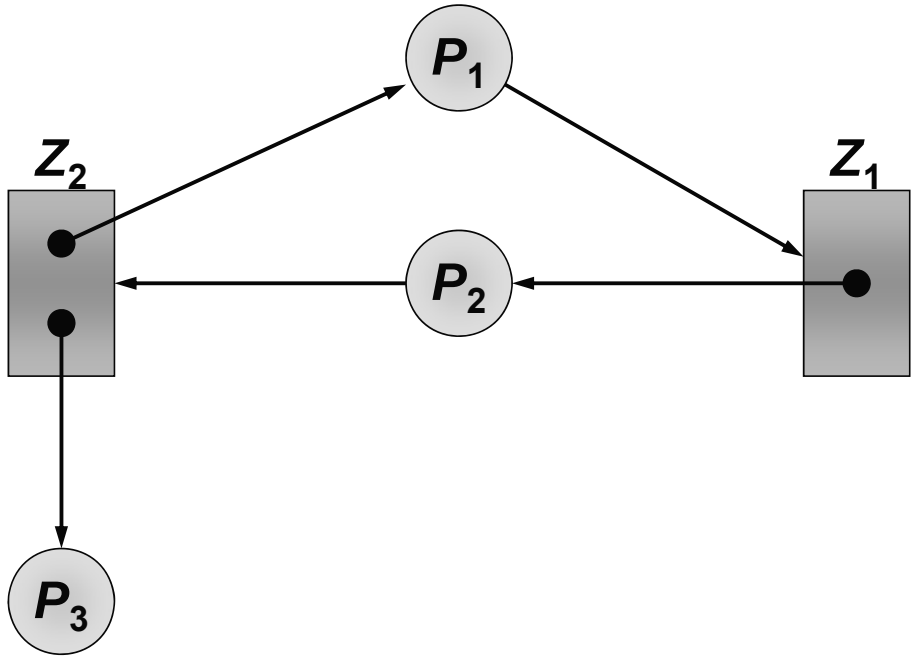


SUM SKiSR Oprogramowanie systemowe 209

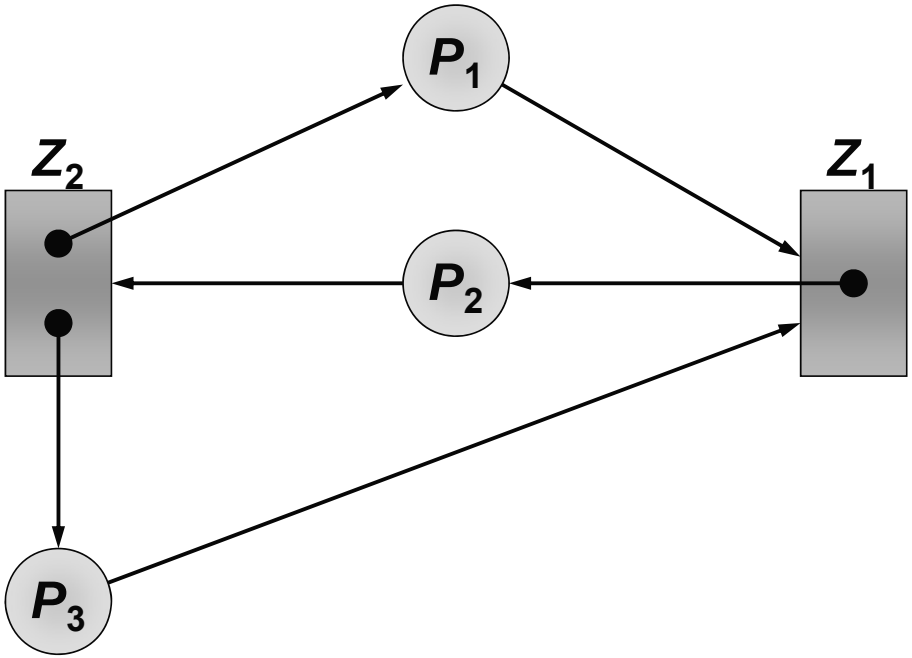
Cykl w grafie przydziału — zakleszczenie



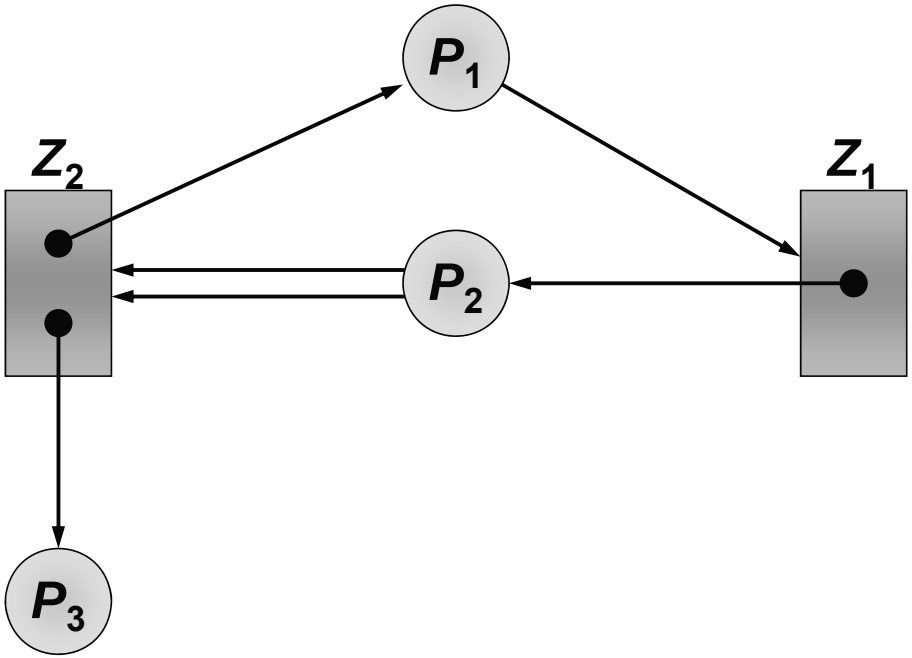
Cykl w grafie przydziału — brak zakleszczenia



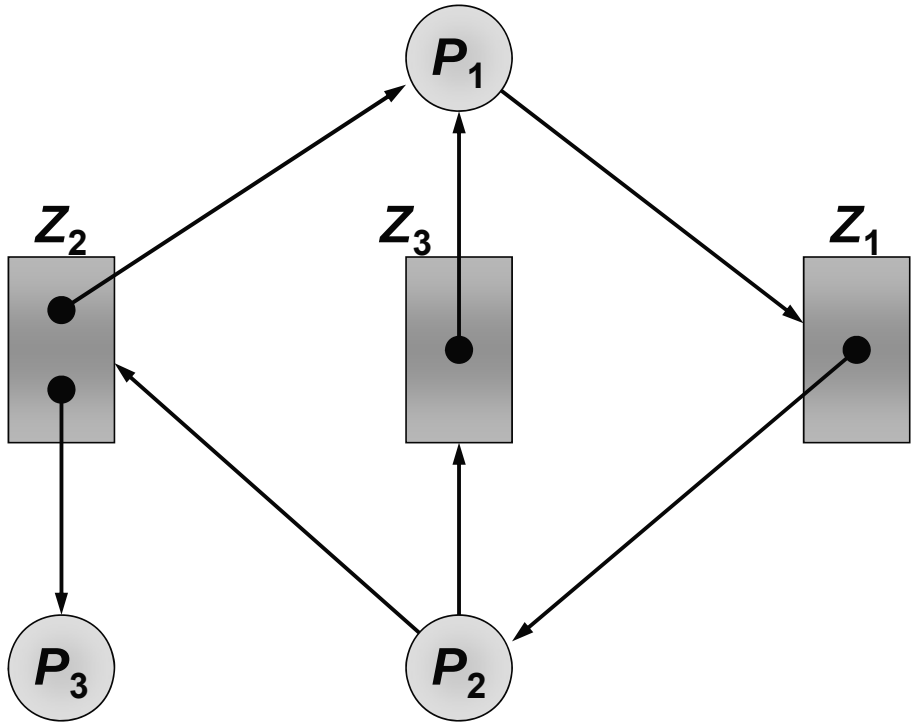
Supeł w grafie przydziału — zakleszczenie



Brak supła w grafie przydziału — zakleszczenie



Brak supła w grafie przydziału — zakleszczenie



SUM SKiSR Oprogramowanie systemowe 214

Własności grafu zasobów odzyskiwalnych a stan zakleszczenia

- Zasoby pojedyncze:
  cykl $\Leftrightarrow$ zakleszczenie
- Przydział natychmiastowy (stan zupełny):
  supeł $\Rightarrow$ zakleszczenie
- Zasoby reprezentowane przez wiele egzemplarzy w systemie z przydziałem natychmiastowym (w stanie zupełnym), dopuszczającym pojedyncze żądania:
  supeł $\Leftrightarrow$ zakleszczenie

Macierzowa reprezentacja stanu systemu

- C — m-elementowy wektor liczebności zasobów systemu
  - $C[j]$ — całkowita liczba jednostek zasobu Z_j, zarządzanych przez system
- R — macierz $n \times m$ zamówień procesów
  - $R[i,j]$ — liczba jednostek zasobu Z_j zamówiona i oczekiwana przez proces P_i
- A — macierz $n \times m$ przydzielonych jednostek zasobów
  - $A[i,j]$ — liczba jednostek zasobu Z_j przydzielona procesowi P_i
- F — m-elementowy wektor wolnych jednostek
  - $F[j]$ — liczba jednostek zasobu Z_j pozostających w dyspozycji systemu (nie przydzielona procesom)

Integralność macierzowej reprezentacji stanu systemu

$$\bigvee_{1 \le j \le m} C[j] \ge \sum_{i=1}^{n} A[i,j]$$

$$\bigvee_{1 \le j \le m} F[j] \ge C[j] - \sum_{i=1}^{n} A[i,j]$$

$$\bigvee_{1 \le i \le n} \bigvee_{1 \le j \le m} A[i,j] + R[i,j] \le C[j]$$

Macierzowa reprezentacja stanu — zakleszczenie

$$\exists_{\substack{P' \subseteq P \\ P' \neq \emptyset}} \; \forall_{P_i \in P'} \; \exists_{1 \le j \le m} \; R[i,j] > F[j] + \underbrace{\sum_{P_k \notin P'} A[k,j]}$$

zasoby zwolnione
przez nie zakleszczone
procesy

Macierzowa reprezentacja stanu — wykrywanie zakleszczenia (1)

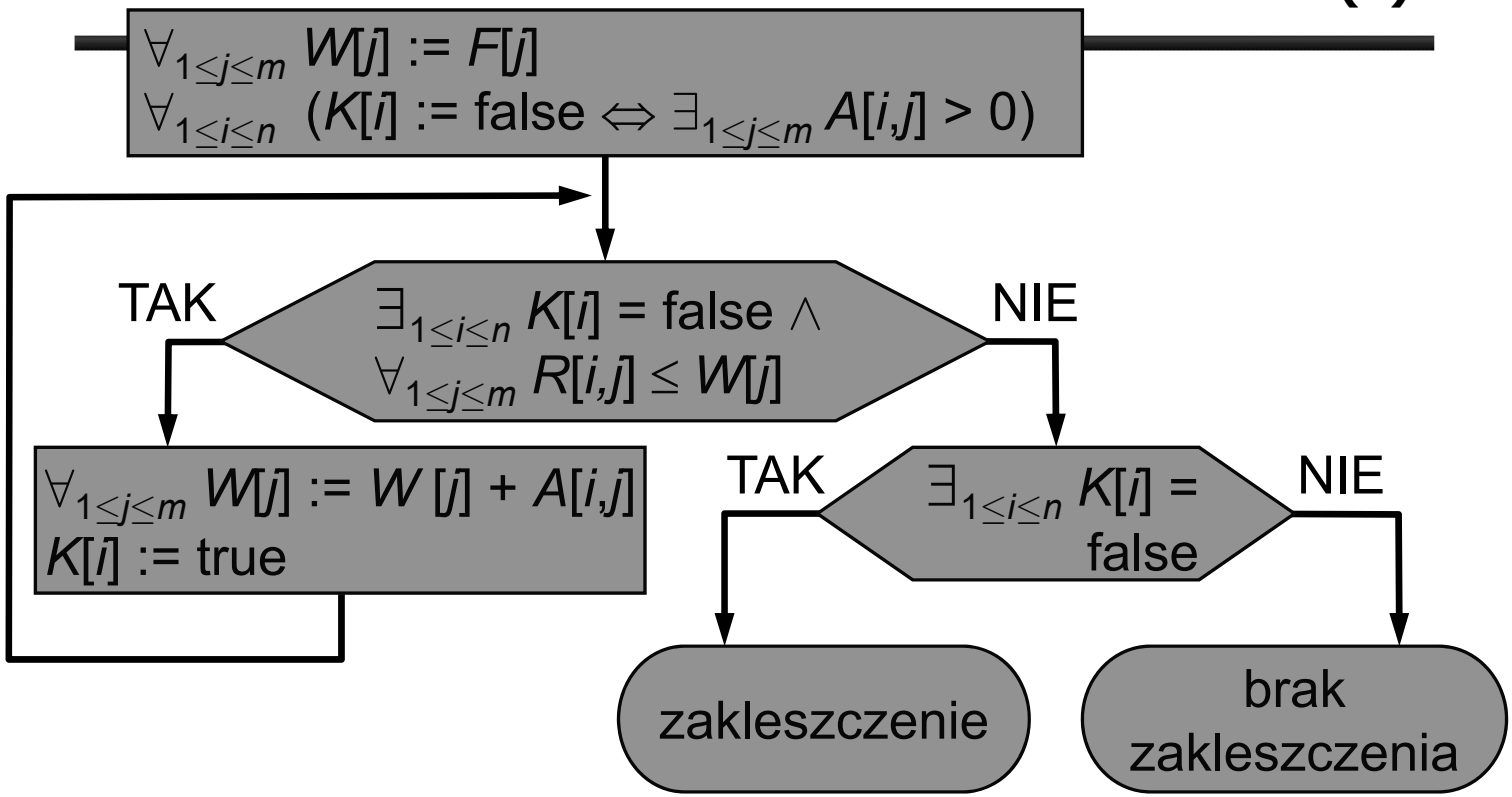
- W — m-elementowy wektor liczby wolnych jednostek zasobów, uwzględniający jednostki zwrócone do systemu przez procesy, które mogą się zakończyć

  $W[j]$ — liczba jednostek zasobu Z_j do rozdysponowania

- K — n-elementowy wektor wartości logicznych, informujący o odzyskaniu zasobów procesu

  $K[i]$ — wartość logiczna informująca, że proces P_i zwrócił do systemu przydzielone mu jednostki zasobów

Macierzowa reprezentacja stanu — wykrywanie zakleszczenia (2)



Przykład działania algorytmu wykrywania zakleszczenia (1)

- System dysponuje 3 typami zasobów: Z_1, Z_2, Z_3 o liczebności odpowiednio 6, 5, 4.
- W systemie współpracują 4 procesy: P_1, P_2, P_3 i P_4.
- Stan systemu:

	$A[1]$	$A[2]$	$A[3]$	$R[1]$	$R[2]$	$R[3]$
P_1	1	0	2	3	1	0
P_2	3	1	0	0	1	3
P_3	1	1	1	1	0	0
P_4	0	2	1	0	1	1

Przykład działania algorytmu wykrywania zakleszczenia (2)

- Wolne zasoby w systemie $F = [1, 1, 0]$.
- Zmiana wektora W oraz K w kolejnych krokach

	$W[1]$	$W[2]$	$W[3]$	$K[1]$	$K[2]$	$K[3]$	$K[4]$
początkowo	1	1	0	F	F	F	F
po zak. P_3	2	2	1	F	F	T	F
po zak. P_4	2	4	2	F	F	T	T

Unikanie zakleszczeń

- Wymagana jest dodatkowa informacja o tym, jakie zasoby będą zamawiane przez proces.
  - W najprostszym przypadku jest to maksymalna liczba jednostek poszczególnych zasobów, niezbędna do zakończenia zadania przez proces.
- Przy każdym zamówieniu zarządca decyduje, czy można je zrealizować, czy należy wstrzymać realizację, biorąc pod uwagę aktualny stan zasobów.
  - W przypadku zadeklarowania maksymalnej liczby jednostek zasobów system musi zapewnić, że nie dojdzie do cyklu w oczekiwaniu na zasoby.

Stan bezpieczny

- Stan systemu jest bezpieczny, jeśli istnieje porządek przydziału zasobów żądającym tego procesom (nawet w stopniu maksymalnym), gwarantujący uniknięcie zakleszczenia.
- Formalnie: system jest w stanie bezpiecznym, jeśli istnieje ciąg bezpieczny, czyli taki ciąg procesów $\langle P_1, P_2, ..., P_n \rangle$, że w danym stanie przydziału zasobów zapotrzebowanie procesu P_i może być zaspokojone przez bieżąco dostępne zasoby oraz zasoby użytkowane przez wszystkie proces poprzedzające go w ciągu, czyli procesy P_j, gdzie $j < i$.

Przykład stanu i ciągu bezpiecznego

- Procesy P_1, P_2, P_3 ubiegają się o jednostki zasobu Z_1, których łączna liczba jest 12.
- Maksymalne zapotrzebowanie oraz bieżący przydział jest następujący:

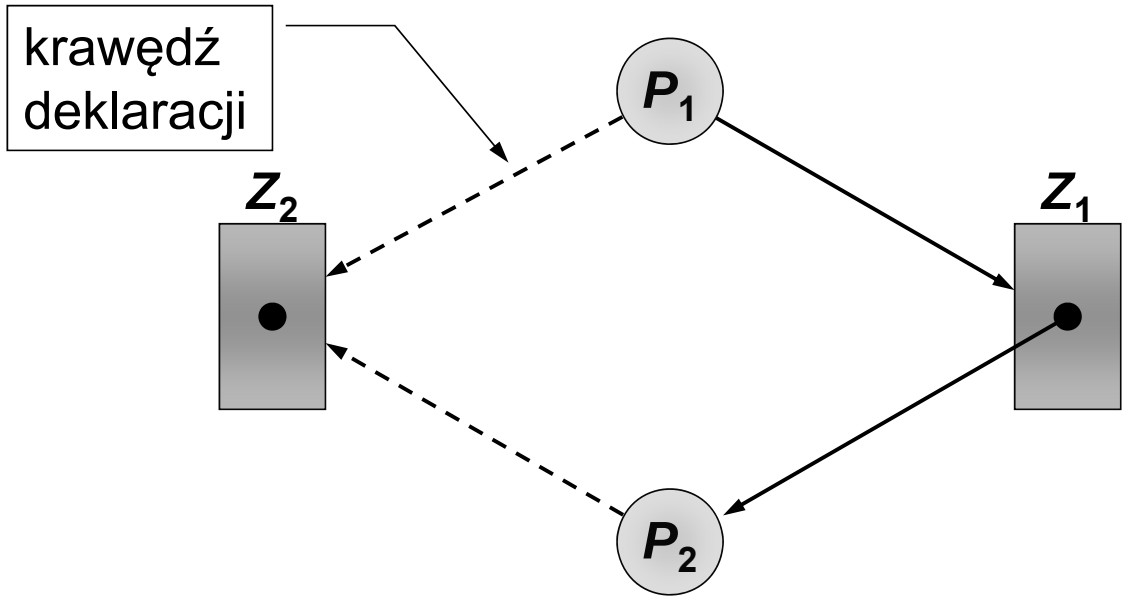
proces	max zapot.	bieżący przydział
P_1	10	5
P_2	4	2
P_3	9	2

- Czy istnieje ciąg bezpieczny?
- Czy można zrealizować żądanie przydziału 1 jednostki zasobu Z_1 procesowi P_3?

Grafowa reprezentacja stanu — unikanie zakleszczenia

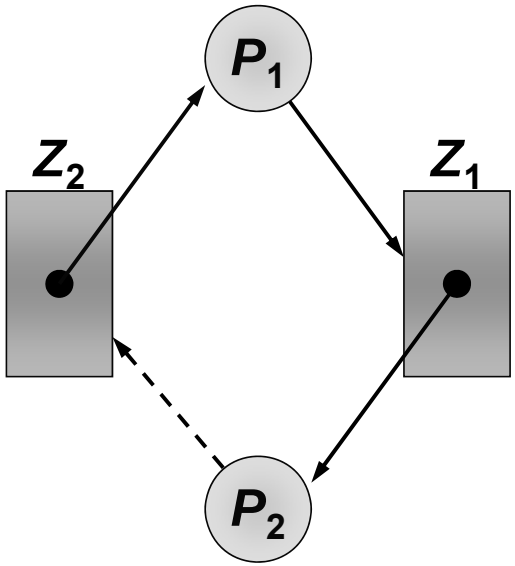
- W celu uwzględnienia potencjalnych żądań, w grafie przydziału zasobów wprowadza się dodatkową krawędź deklaracji, wskazującą, że proces może zamówić egzemplarz zasobu do realizacji zadania.
- Gdy proces zamawia zasób, krawędź deklaracji zamieniana jest na krawędź zamówienia, a gdy go zwalnia, ale się nie kończy, krawędź przydziału zamieniana jest na krawędź deklaracji.
- Zamówienie może być zrealizowane, gdy zamiana krawędzi zamówienia na krawędź przydziału nie spowoduje cyklu lub supła (zależnie od charakterystyki systemu) w grafie oczekiwania, uwzględniającym krawędzie deklaracji.

Graf przydziału z krawędziami deklaracji

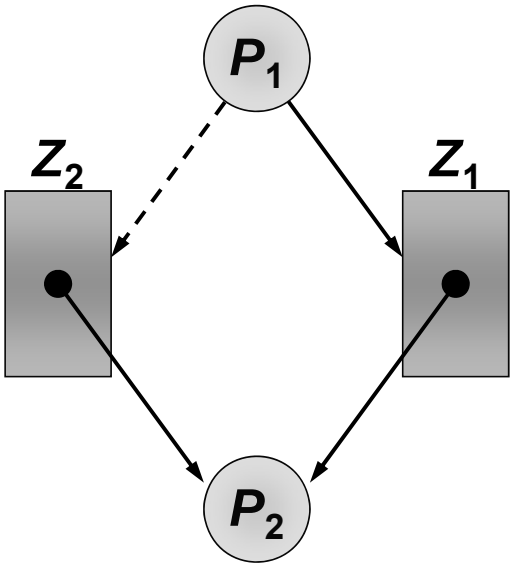


Grafowa reprezentacja stanu — zagrożenie

zagrożenie



brak zagrożenia



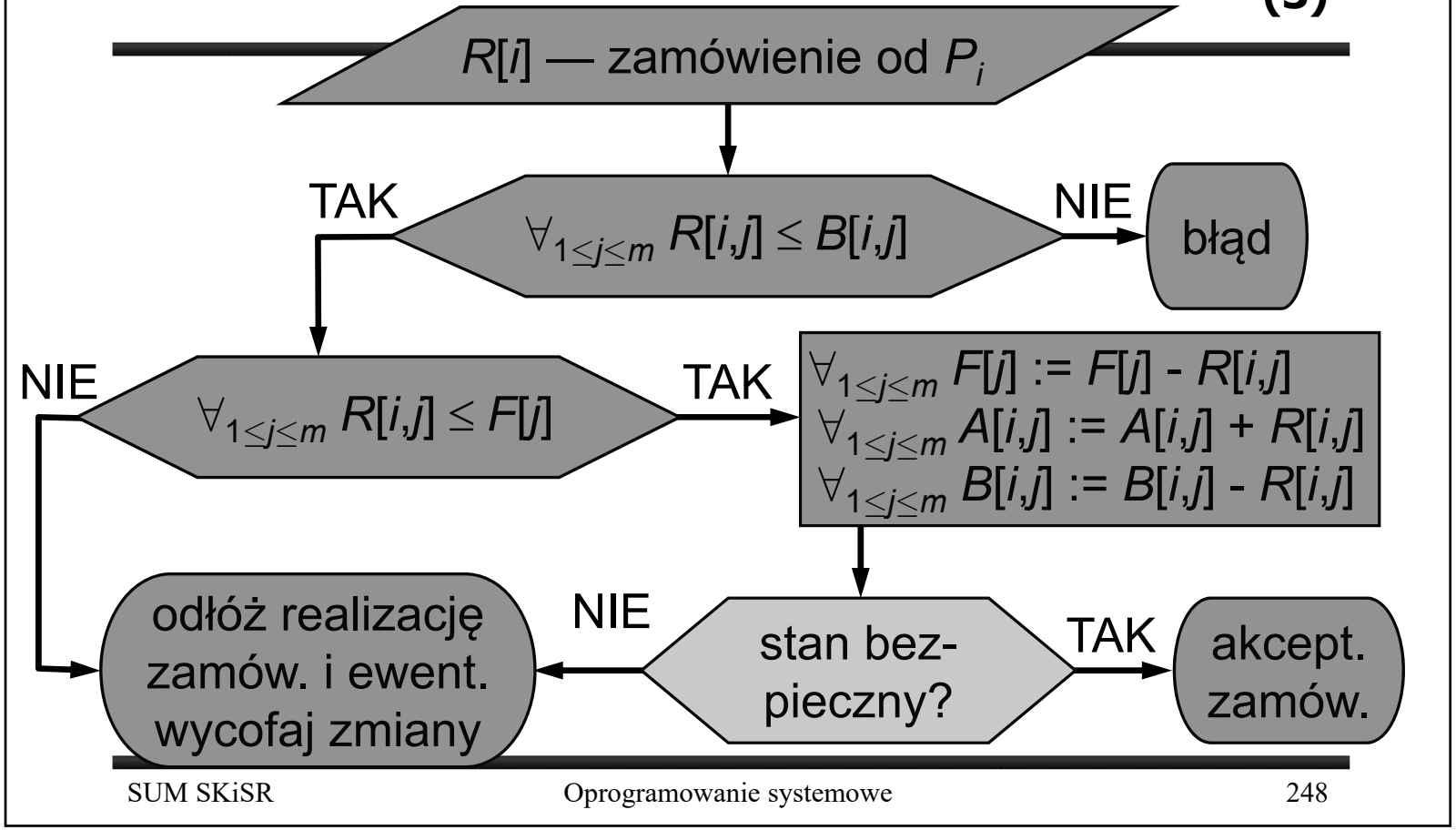
Macierzowa reprezentacja stanu — unikanie zakleszczenia (1)

- Przed rozpoczęciem realizacji zadania proces musi zadeklarować maksymalną liczbę jednostek poszczególnych typów zasobów, których może potrzebować.
- Na podstawie deklaracji i bieżącego stanu systemu zarządca musie rozstrzygnąć, czy przydział zasobów pozostawi system w stanie bezpiecznym. Jeśli tak, to zasoby mogą zostać przydzielone, a jeśli nie, to proces musi poczekać.

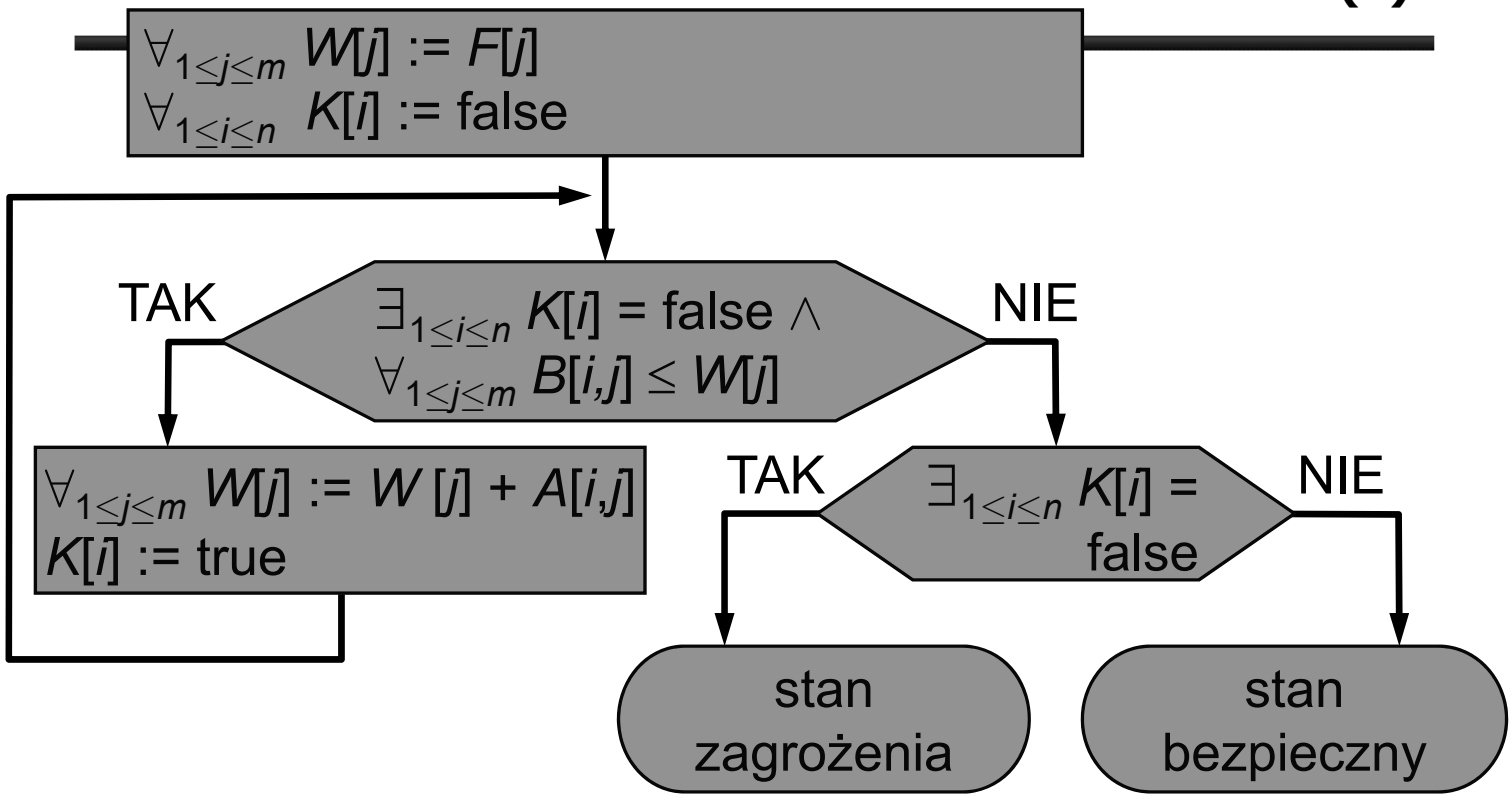
Macierzowa reprezentacja stanu — unikanie zakleszczenia (2)

- D — macierz o wymiarach $n \times m$ określająca maksymalne zapotrzebowanie poszczególnych procesów na poszczególne zasoby
- B — macierz o wymiarach $n \times m$ określająca zapotrzebowanie poszczególnych procesów na poszczególne zasoby, które jest jeszcze do zrealizowania (nie wykorzystana jeszcze część deklaracji)

Macierzowa reprezentacja stanu — unikanie zakleszczenia (3)



Macierzowa reprezentacja stanu — unikanie zakleszczenia (4)



Przykład działania algorytmu (1)

- System dysponuje zasobami Z_1, Z_2, po 8 jednostek.
- W systemie współpracuje 5 procesów: P_1, P_2, P_3 , P_4, P_5.
- Stan systemu:

	$A[1]$	$A[2]$	$D[1]$	$D[2]$	$B[1]$	$B[2]$
P_1	2	2	6	4	4	2
P_2	2	0	7	2	5	2
P_3	0	1	2	3	2	2
P_4	1	0	3	4	2	4
P_5	0	2	2	6	2	4

Przykład działania algorytmu (2)

- Wolne zasoby w systemie $F = [3, 3]$.
- Zmiana wektora W oraz K w kolejnych krokach

	$W[1]$	$W[2]$	$K[1]$	$K[2]$	$K[3]$	$K[4]$	$K[5]$
początkowo	3	3	F	F	F	F	F
po zak. P_3	3	4	F	F	T	F	F
po zak. P_4	4	4	F	F	T	T	F
po zak. P_5	4	6	F	F	T	T	T
po zak. P_1	6	8	T	F	T	T	T
po zak. P_2	8	8	T	T	T	T	T

Przykład działania algorytmu (3)

- Stan systemu po zrealizowaniu żądania [1,0] procesu P_2:

	$A[1]$	$A[2]$	$D[1]$	$D[2]$	$B[1]$	$B[2]$
P_1	2	2	6	4	4	2
P_2	3	0	7	2	4	2
P_3	0	1	2	3	2	2
P_4	1	0	3	4	2	4
P_5	0	2	2	6	2	4

Przykład działania algorytmu (4)

- Wolne zasoby w systemie $F = [2, 3]$.
- Zmiana wektora W oraz K w kolejnych krokach

	$W[1]$	$W[2]$	$K[1]$	$K[2]$	$K[3]$	$K[4]$	$K[5]$
początkowo	2	3	F	F	F	F	F
po zak. P_3	2	4	F	F	T	F	F
po zak. P_4	3	4	F	F	T	T	F
po zak. P_5	3	6	F	F	T	T	T