

# Systemy Operacyjne - wprowadzenie

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 11 października 2020

# Plan wykładu

- 1 Wykładowca
- 2 Bibliografia
- 3 Czym jest System Operacyjny?
  - System Komputerowy
  - System Operacyjny
- 4 Kategorie systemów operacyjnych
  - Początki systemów operacyjnych
  - Systemy wsadowe - monitory
    - Przetwarzanie pośrednie i satelitarne
    - Buforowanie
    - Spooling
  - Systemy z podziałem czasu
  - Systemy jednostanowiskowe
  - Systemy z wieloma procesorami
  - Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy

- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych

- Systemy wsadowe - monitory

- Przetwarzanie pośrednie i satelitarne

- Buforowanie

- Spooling

- Systemy z podziałem czasu

- Systemy jednostanowiskowe

- Systemy z wieloma procesorami

- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Plan wykładu

1 Wykładowca

2 Bibliografia

3 Czym jest System Operacyjny?

- System Komputerowy
- System Operacyjny

4 Kategorie systemów operacyjnych

- Początki systemów operacyjnych
- Systemy wsadowe - monitory
  - Przetwarzanie pośrednie i satelitarne
  - Buforowanie
  - Spooling
- Systemy z podziałem czasu
- Systemy jednostanowiskowe
- Systemy z wieloma procesorami
- Systemy czasu rzeczywistego

# Wykładowca

- dr inż. Arkadiusz Chrobot
- pokój: 3.23 D
- telefon: 41 34-24-185
- e-mail: [a.chrobot@tu.kielce.pl](mailto:a.chrobot@tu.kielce.pl)
- WebEx: <https://tu-kielce.webex.com/meet/a.chrobot>
- termin konsultacji: poniedziałki, 18:00 – 19:30 (przez WebEx)
- www:<https://achilles.tu.kielce.pl>

## Bibliografia - wykład

- ① Abraham Silberschatz, James L.Peterson, Peter B.Galvin, *Podstawy systemów operacyjnych*, WNT, Warszawa 1993
- ② Abraham Silberschatz, Peter B. Galvin, Greg Gagne, *Podstawy systemów operacyjnych*, WNT, Warszawa 2005
- ③ William Stallings, *Systemy operacyjne Struktura i zasady budowy*, PWN, Warszawa 2006
- ④ Andrew S. Tanenbaum, *Systemy operacyjne*, Helion, Gliwice 2010
- ⑤ Andrew S. Tanenbaum, Albert S. Woodhull, *Operating Systems Design and Implementation*, Pearson Education International, Upper Saddle River, 2009

## Bibliografia - laboratorium

- ① W.Richard Stevens, *Programowanie zastosowań sieciowych w systemie Unix*, WNT, Warszawa 1995
- ② Neil Matthew, Richard Stones, *Linux Programowanie*, Wydawnictwo RM, Warszawa 1999
- ③ Keith Haviland, Dina Gray, Ben Salama, *Unix Programowanie systemowe*, Wydawnictwo RM, Warszawa 1999

# System komputerowy

## Definicja

*System Komputerowy* jest to zespół sprzętu i oprogramowania, którego zadaniem jest przetwarzanie danych. Częścią systemu komputerowego jest również jego *użytkownik*.

Przykładami systemu komputerowego są: bankomat, komputer domowy, telefon komórkowy. Sieci komputerowe mogą być traktowane jako jeden spójny system komputerowy lub zespół systemów komputerowych. Elementy systemu komputerowego, zarówno te fizyczne, jak i logiczne nazywamy *zasobami*. Do zasobów fizycznych możemy zaliczyć: procesor, pamięć i urządzenia wejścia-wyjścia, do zasobów logicznych dane komputerowe, w postaci np. plików.

# System operacyjny - definicja

## Definicja

*System Operacyjny* jest częścią systemu komputerowego. Jest programem komputerowym, którego zadaniem jest zarządzanie wszystkimi zasobami systemu komputerowego. Stanowi on kluczowy element oprogramowania, wykonujący takie podstawowe zadania, jak: kontrola i alokacja pamięci, określanie kolejności wykonania programów, sterowanie urządzeniami wejścia-wyjścia, obsługa sieci i zarządzanie plikami.

# System operacyjny - opis beletrystyczny

Neal Stephenson *Zamień*

„Kiedy po raz pierwszy włączasz komputer, masz do czynienia z nienowym zbiorowiskiem obwodów, które same z siebie nic nie potrafią. Żeby komputer działał, musisz włączyć do obwodów zbiór zasad, które powiedzą im, co należy zrobić. Jak być komputerem.”

Hiro Protagonista do Bibliotekarza

# Co należy do zadań Systemu Operacyjnego?

- Usprawnienie pracy programisty.

# Co należy do zadań Systemu Operacyjnego?

- Usprawnienie pracy programisty.
- Sprawiedliwe zarządzanie zasobami komputera.

# Co należy do zadań Systemu Operacyjnego?

- Usprawnienie pracy programisty.
- Sprawiedliwe zarządzanie zasobami komputera.
- Nadzór nad programami użytkownika.

# Co należy do zadań Systemu Operacyjnego?

- Usprawnienie pracy programisty.
- Sprawiedliwe zarządzanie zasobami komputera.
- Nadzór nad programami użytkownika.
- Świadczenie usług programom użytkownika, które są wykonywane przez system komputerowy.

# Co należy do zadań Systemu Operacyjnego?

- Usprawnienie pracy programisty.
- Sprawiedliwe zarządzanie zasobami komputera.
- Nadzór nad programami użytkownika.
- Świadczenie usług programom użytkownika, które są wykonywane przez system komputerowy.
- Ułatwienie użytkownikowi posługiwania się systemem komputerowym.

# Proste oprogramowanie

Pierwsze systemy komputerowe *w ogóle nie posiadały* systemów operacyjnych. Z czasem stworzono dla nich oprogramowanie, które przyczyniło się do powstania lub weszło w skład systemów operacyjnych. Niewątpliwie należy do niego zaliczyć biblioteki procedur obsługi urządzeń wejścia-wyjścia. Pozwalały one odciążyć programistę od kodowania powtarzających się fragmentów programu, tym samym zmniejszając prawdopodobieństwo popełnienia przez niego błędów. Do oprogramowania systemowego można zaliczyć również program ładowający (ang. *loader*), który stanowił część oprogramowania podstawowego. W skład tego oprogramowania wchodziły również kompilatory ówczesnych języków programowania (BASIC, COBOL, FORTRAN) i konsolidatory (ang. *linker*).

# Charakterystyka pierwszych systemów komputerowych:

Zalety:

- + bezpośrednia styczność programisty z komputerem,

Wady:

# Charakterystyka pierwszych systemów komputerowych:

Zalety:

- + bezpośrednia styczność programisty z komputerem,
- + programista miał całkowitą kontrolę nad systemem komputerowym.

Wady:

# Charakterystyka pierwszych systemów komputerowych:

Zalety:

- + bezpośrednia styczność programisty z komputerem,
- + programista miał całkowitą kontrolę nad systemem komputerowym.

Wady:

- duża cena,

# Charakterystyka pierwszych systemów komputerowych:

## Zalety:

- + bezpośrednia styczność programisty z komputerem,
- + programista miał całkowitą kontrolę nad systemem komputerowym.

## Wady:

- duża cena,
- brak jakiegokolwiek oprogramowania wspomagającego,

# Charakterystyka pierwszych systemów komputerowych:

## Zalety:

- + bezpośrednia styczność programisty z komputerem,
- + programista miał całkowitą kontrolę nad systemem komputerowym.

## Wady:

- duża cena,
- brak jakiegokolwiek oprogramowania wspomagającego,
- czasochłonna i skomplikowana obsługa,

# Charakterystyka pierwszych systemów komputerowych:

Zalety:

- + bezpośrednia styczność programisty z komputerem,
- + programista miał całkowitą kontrolę nad systemem komputerowym.

Wady:

- duża cena,
- brak jakiegokolwiek oprogramowania wspomagającego,
- czasochłonna i skomplikowana obsługa,
- skomplikowane usuwanie błędów w oprogramowaniu,

# Charakterystyka pierwszych systemów komputerowych:

## Zalety:

- + bezpośrednia styczność programisty z komputerem,
- + programista miał całkowitą kontrolę nad systemem komputerowym.

## Wady:

- duża cena,
- brak jakiegokolwiek oprogramowania wspomagającego,
- czasochłonna i skomplikowana obsługa,
- skomplikowane usuwanie błędów w oprogramowaniu,
- praca oparta na harmonogramach.

# Systemy wsadowe

Aby zwiększyć efektywność pierwszych systemów komputerowych zatrudniano specjalnie przeszkołonych *operatorów*, których zadaniem była obsługa komputera, polegająca na komplikowaniu i uruchamianiu programów. Programiści przekazywali swoje programy, w postaci kodu źródłowego wydrukowanego na kartach perforowanych operatorowi, a ten uruchamiał je zgodnie z ich instrukcjami. Operator z reguły **nie był programistą**, a więc jeśli program zawierał błędy, mógł tylko przekazać informacje o nich programiście i zająć się uruchamianiem innego programu. W celu usprawnienia swojej pracy operatorzy segregowali programy uwzględniając ich wymagania. Wszystkie programy o podobnych wymaganiach (np. komplikowanych tym samym kompilatorem) organizowali w jeden zbiór zwany wsadem (ang. *batch*). Stąd powstała nazwa dla tych systemów komputerowych — systemy wsadowe.

# Systemy wsadowe — charakterystyka

Zalety:

- + lepsza organizacja pracy komputera,

Wady:

# Systemy wsadowe — charakterystyka

Zalety:

- + lepsza organizacja pracy komputera,
- + efektywniejsze wykorzystanie systemu komputerowego.

Wady:

# Systemy wsadowe — charakterystyka

Zalety:

- + lepsza organizacja pracy komputera,
- + efektywniejsze wykorzystanie systemu komputerowego.

Wady:

- konieczność zatrudnienia i/lub przeszkołenia operatora,

# Systemy wsadowe — charakterystyka

Zalety:

- + lepsza organizacja pracy komputera,
- + efektywniejsze wykorzystanie systemu komputerowego.

Wady:

- konieczność zatrudnienia i/lub przeszkolenia operatora,
- odsunięcie programisty od sprzętu.

# Prosty monitor

Z biegiem czasu część czynności, które wykonywał operator została zautomatyzowana. Ich wykonaniem zajął się program o nazwie *monitor rezydujący*. Składał się on z interpretera kart sterujących, modułu porządkującego zadania i programu ładowającego. Monitor od chwili uruchomienia, do chwili zakończenia działania systemu komputerowego zawsze pozostawał w pamięci operacyjnej. Można go więc uznać za protoplastę *jądra systemu operacyjnego*.

# Praca pośrednia

Po usprawnieniu pracy operatora kolejnym wąskim gardłem obniżającym wydajność systemów komputerowych okazały się urządzenia wejścia-wyjścia, do których należały przede wszystkim czytniki kart perforowanych i drukarki. Szybszym nośnikiem danych od kart były taśmy magnetyczne, ale posiadały one podstawową wadę - nie można było na nich bezpośrednio zapisywać, tak jak na kartach perforowanych. Rozwiążanie problemu polegało na zakupie specjalnych urządzeń, które przepisywały zawartość kart perforowanych na taśmę magnetyczną lub dane z taśmy magnetycznej drukowały na drukarce. Podczas trwania tych czynności komputer mógł pracować wczytując programy i dane z taśm wcześniej przygotowanych przez te urządzenia i zapisując wynik swej pracy na innych taśmach. Taki sposób obsługi czytników kart i drukarek nazywamy pracą pośrednią (ang. *off-line*).

# Przetwarzanie satelitarne

Odmianą opisanego wcześniej rozwiązania były systemy komputerowe, składające się z głównego komputera, korzystającego wyłącznie z napędów taśm magnetycznych, jako jednostek wejścia-wyjścia i z szeregu „mniejszych” komputerów, które spełniały funkcję opisanych wcześniej urządzeń. Konsekwencją wprowadzenia pracy pośredniej było uniezależnienie działania programów użytkownika, od rodzaju urządzeń wejścia-wyjścia z jakimi pracowały. Uruchomiony program (proces) wykonywał odczyt lub zapis na *urządzeniu logicznym*, natomiast monitor, którego częścią stały się biblioteki podprogramów wejścia-wyjścia odwzorowywał to urządzenie na *urządzenie fizyczne*.

# Praca pośrednia - charakterystyka

Zalety:

- + lepsze wykorzystanie jednostki obliczeniowej centralnego komputera,

Wady:

# Praca pośrednia - charakterystyka

Zalety:

- + lepsze wykorzystanie jednostki obliczeniowej centralnego komputera,
- + „wirtualizacja” urządzeń wejścia-wyjścia.

Wady:

# Praca pośrednia - charakterystyka

Zalety:

- + lepsze wykorzystanie jednostki obliczeniowej centralnego komputera,
- + „wirtualizacja” urządzeń wejścia-wyjścia.

Wady:

- koszt zakupu dodatkowych urządzeń,

# Praca pośrednia - charakterystyka

Zalety:

- + lepsze wykorzystanie jednostki obliczeniowej centralnego komputera,
- + „wirtualizacja” urządzeń wejścia-wyjścia.

Wady:

- koszt zakupu dodatkowych urządzeń,
- długi czas przetwarzania zadania.

# Buforowanie wejścia-wyjścia

Postęp w dziedzinie technologii umożliwił jednoczesną pracę procesora i urządzeń wejścia-wyjścia w obrębie jednego systemu komputerowego. W czasie, kiedy procesor realizował obliczenia urządzenia wejściowe odczytywały dane, które były potrzebne programowi w przyszłości i umieszczały je w odpowiednich miejscach w pamięci operacyjnej komputera. Te miejsca określono mianem *buforów*, a samą technikę *buforowaniem*. Stosowano ją również w operacjach wyjścia. Wyniki swojej pracy program nie wysyłał bezpośrednio do urządzenia wyjściowego, lecz umieszczał je w odpowiednich buforach. Udostępnianie buforów wejściowych i opróżnianie wyjściowych nadzorował system operacyjny. Buforowanie ma na celu zrównoważenie obciążenia procesora i urządzeń wejścia-wyjścia.

# Efektywność buforowania

W rzeczywistym systemie komputerowym dosyć rzadko występują programy, które w równym stopniu korzystają z procesora i jednostek wejścia-wyjścia. Najczęściej występują dwie odmiany zadań: *uzależnione od wejścia-wyjścia* lub *uzależnione od procesora*. Zadania uzależnione do wejścia-wyjścia wykonują więcej operacji pobrania danych, niż obliczeń, co powoduje, że procesor czeka na zakończenie pracy przez urządzenia wejścia-wyjścia. W przypadku zadań uzależnionych od procesora sytuacja jest odwrotna. W obu przypadkach buforowanie, jeśli nie jest wspierane dodatkowymi rozwiązaniami może się nie sprawdzić.

# Buforowanie - charakterystyka

Zalety:

- + zrównoważenie (a przynajmniej próba) obciążenia procesora i jednostek wejścia-wyjścia,

Wady:

# Buforowanie - charakterystyka

Zalety:

- + zrównoważenie (a przynajmniej próba) obciążenia procesora i jednostek wejścia-wyjścia,
- + mały koszt rozwiązania.

Wady:

# Buforowanie - charakterystyka

Zalety:

- + zrównoważenie (a przynajmniej próba) obciążenia procesora i jednostek wejścia-wyjścia,
- + mały koszt rozwiązania.

Wady:

- mała efektywność (w przypadku pierwszych rozwiązań).

# Spooling

Ulepszeniem techniki buforowania, które pojawiło się wraz z upowszechnieniem pamięci dyskowych o dostępnie swobodnym był spooling (ang. simultaneous peripheral operation on-line). Umożliwiał on „równoczesne” buforowanie na dysku danych wejściowych i wyników pracy wielu zadań. Możliwe również stało się umieszczanie w pamięci dyskowej pewnej liczby zadań (programów użytkowników) i **dynamiczne planowanie kolejności ich wykonania**. System operacyjny stał się odpowiedzialny za obsługę pamięci dyskowej, nadzorowanie spoolingu i za wspomniane planowanie.

# Systemy wielodostępne i wielozadaniowe

Kolejna generacja systemów komputerowych dysponowała na tyle dużą pamięcią operacyjną, że mogła utrzymywać w niej równocześnie do kilku-dziesięciu procesów użytkownika. Procesor mógł je wykonywać w dowolnej kolejności. W chwili, gdy bieżące zadanie musiało pobrać dane z urządzenia wejściowego, procesor był przełączany do innego zadania, które oczekiwano na wykonanie. Za przełączanie procesora między zadaniami i określanie kolejności ich wykonania odpowiedzialny stał się system operacyjny. Do jego obowiązków należała również ochrona obszarów pamięci operacyjnej przydzielonych poszczególnym procesom (zadaniom). Kiedy systemy komputerowe zaczęto wyposażyć w terminale, składające się z monitora CRT i klawiatury, stało się możliwe użytkowanie komputera przez kilkunastu lub kilkudziesięciu użytkowników równocześnie.

# Systemy wielodostępne i wielozadaniowe

Aby ich praca mogła przebiegać „równocześnie” i w sposób interaktywny procesor musiał być przełączany pomiędzy zadaniami poszczególnych użytkowników, co pewien krótki odcinek czasu. Systemy tego typu mogły również wykonywać zadania w trybie wsadowym. Systemy operacyjne działające na takich systemach komputerowych (które określa się mianem wielozadaniowych i wielodostępnych) są skomplikowanym oprogramowaniem. Do ich zadań należy nie tylko zarządzanie i ochrona programów, ale również ochrona i zarządzanie danymi użytkowników zgromadzonymi w pamięciach dyskowych oraz interaktywna komunikacja z użytkownikiem. W nadzorowanych przez nie systemach komputerowych nie tylko jest ważny *czas przetwarzania* zadań, ale również *czas odpowiedzi* systemu, który jest wyznacznikiem stopnia jego interaktywności i wygody użytkowania.

# Systemy wielodostępne i wielozadaniowe - charakterystyka

## Zalety:

- + możliwość jednoczesnej pracy wielu użytkowników,

## Wady:

# Systemy wielodostępne i wielozadaniowe - charakterystyka

## Zalety:

- + możliwość jednoczesnej pracy wielu użytkowników,
- + bezpośredni kontakt programisty z systemem komputerowym,

## Wady:

# Systemy wielodostępne i wielozadaniowe - charakterystyka

## Zalety:

- + możliwość jednoczesnej pracy wielu użytkowników,
- + bezpośredni kontakt programisty z systemem komputerowym,
- + wygoda użytkowania,

## Wady:

# Systemy wielodostępne i wielozadaniowe - charakterystyka

## Zalety:

- + możliwość jednoczesnej pracy wielu użytkowników,
- + bezpośredni kontakt programisty z systemem komputerowym,
- + wygoda użytkowania,
- + możliwość wykonywania zadań wsadowych „w tle” ,

## Wady:

# Systemy wielodostępne i wielozadaniowe - charakterystyka

## Zalety:

- + możliwość jednoczesnej pracy wielu użytkowników,
- + bezpośredni kontakt programisty z systemem komputerowym,
- + wygoda użytkowania,
- + możliwość wykonywania zadań wsadowych „w tle” ,
- + efektywność.

## Wady:

# Systemy wielodostępne i wielozadaniowe - charakterystyka

## Zalety:

- + możliwość jednoczesnej pracy wielu użytkowników,
- + bezpośredni kontakt programisty z systemem komputerowym,
- + wygoda użytkowania,
- + możliwość wykonywania zadań wsadowych „w tle” ,
- + efektywność.

## Wady:

- względnie duża cena.

# Systemy jednostanowiskowe

Rozwój w dziedzinie sprzętu doprowadził do powstania tanich komputerów osobistych, które stanowiły konkurencję dla dużych systemów komputerowych, a obecnie dominują na rynku informatycznym. Dla tych komputerów powstały specjalne wersje systemów operacyjnych. Najpierw były to dosyć proste systemy, jak MS-DOS, z czasem zaczęły jednak ewoluować i stawać się skomplikowanymi systemami wielozadaniowymi, z możliwością obsługi (niekoniecznie równoczesnej) wielu użytkowników, takimi jak MS-Windows i Mac OS. Część „dużych” systemów operacyjnych została przystosowana do pracy na takich komputerach. Tutaj sztandarowym przykładem są różne odmiany systemu Unix.

# Systemy z wieloma procesorami

Wraz z postępem w dziedzinie sprzętu pojawiły się systemy komputerowe zawierające więcej niż jeden procesor do przetwarzania danych. Te systemy możemy podzielić na trzy grupy:

- ① *wieloprocesory* - komputery z określona liczbą procesorów mających wspólną pamięć,
- ② *wielokomputery* - komputery jedno lub wieloprocesorowe połączone lokalną siecią komputerową,
- ③ *systemy rozproszone* - tak jak wyżej, ale połączenie jest realizowane za pomocą sieci rozległej.

Rozróżniamy dwa rodzaje systemów operacyjnych współpracujących z systemami z wieloma procesorami, opartymi o sieć:

- ① *rozproszone systemy operacyjne*, np.: Linux z rozszerzeniem Kerberos, Windows 2003,
- ② *sieciowe systemy operacyjne*, np.: Unix, Windows, Mac OS.

# Systemy czasu rzeczywistego

Systemy czasu rzeczywistego mają za zadanie zapewnienie wykonania zadań w ścisłe określonych ramach czasowych. Najczęściej są one stosowane wszędzie tam, gdzie trzeba zapewnić zakończenie zadania w określonym czasie np.: w elektrowniach atomowych, samolotach. Przykładem takiego systemu jest system QNX. Rozróżniamy dwie podstawowe kategorie systemów czasu rzeczywistego:

- miękkie systemy czasu rzeczywistego (ang. *soft real-time systems*), są to systemy, w których przekroczenie czasu realizacji zadania nie skutkuje katastrofalnymi następstwami, a jedynie pogorszeniem jakości świadczonych przez nie usług (ang. *Quality of Service*),
- twardé systemy czasu rzeczywistego (ang. *hard real-time systems*), są to systemy, w których przekroczenie czasu realizacji zadania skutkuje katastrofalnymi następstwami.

# Pytania

?

# Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne — sprzęt

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 10 października 2020

# Plan wykładu

## ① Scenariusze obsługi wejścia-wyjścia

### ② System przerwań

- Sposób działania
- Realizacja
- DMA
- Pułapki
- Zagadnienia pokrewne

### ③ Ochrona sprzętowa

- Dualny tryb pracy procesora
- Ochrona pamięci
- Timer

### ④ Wywołania systemowe

### ⑤ Architektury wieloprocesorowe

### ⑥ Komputery osobiste

# Plan wykładu

## ① Scenariusze obsługi wejścia-wyjścia

## ② System przerwań

- Sposób działania
- Realizacja
- DMA
- Pułapki
- Zagadnienia pokrewne

## ③ Ochrona sprzętowa

- Dualny tryb pracy procesora
- Ochrona pamięci
- Timer

## ④ Wywołania systemowe

## ⑤ Architektury wieloprocesorowe

## ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

## ① Scenariusze obsługi wejścia-wyjścia

## ② System przerwań

- Sposób działania
- Realizacja
- DMA
- Pułapki
- Zagadnienia pokrewne

## ③ Ochrona sprzętowa

- Dualny tryb pracy procesora
- Ochrona pamięci
- Timer

## ④ Wywołania systemowe

## ⑤ Architektury wieloprocesorowe

## ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

# Plan wykładu

- ① Scenariusze obsługi wejścia-wyjścia
- ② System przerwań
  - Sposób działania
  - Realizacja
  - DMA
  - Pułapki
  - Zagadnienia pokrewne
- ③ Ochrona sprzętowa
  - Dualny tryb pracy procesora
  - Ochrona pamięci
  - Timer
- ④ Wywołania systemowe
- ⑤ Architektury wieloprocesorowe
- ⑥ Komputery osobiste

## Obsługa wejścia-wyjścia

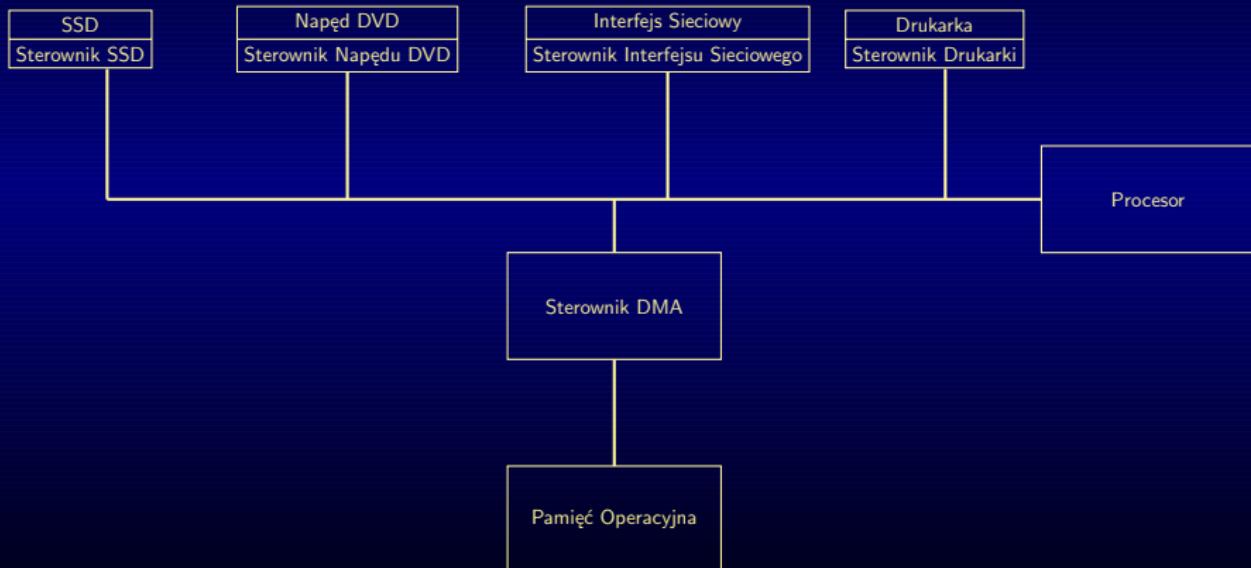
### Aktywne oczekiwanie (ang. *busy waiting*)

Procesor nadzoruje pracę urządzenia przez cały czas trwania transmisji. Nadzór ten obejmuje takie czynności, jak: sprawdzenie gotowości urządzenia, wprowadzenie danych niezbędnych do wykonania komunikacji, oczekивание на odbiór lub wysłanie danych, w przypadku odbioru skopiowanie danych do pamięci operacyjnej. W czasie oczekiwania na zakończenie operacji wejścia-wyjścia procesor nie wykonuje żadnych innych zadań.

### Przerwania (ang. *interrupts*)

Urządzenie możemy wyposażyć w układ sterownika, który może nadzorować jego pracę, zwalniając tym samym procesor z tego obowiązku. W szczególności CPU nie musi czekać na zakończenie realizacji komunikacji. Sterownik powiadamia procesor o zakończeniu transmisji generując sygnał nazywany *przerwaniem*.

# Przerwania - schemat sprzętu



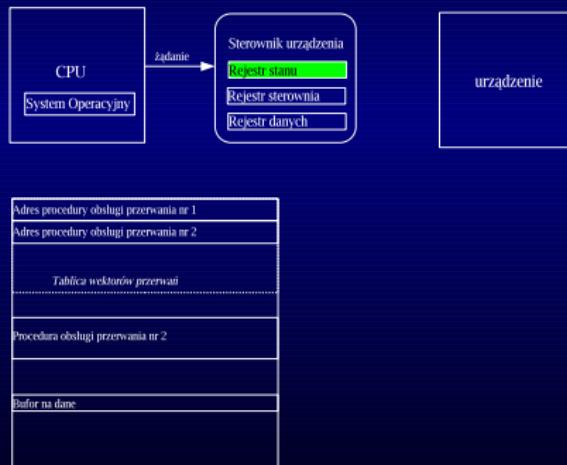
## Przerwania - schemat działania



Adres procedury obsługi przerwania nr 1
Adres procedury obsługi przerwania nr 2
Tablica wektorów przerwań
Procedura obsługi przerwania nr 2
Dufor na dane

Proces (program) użytkownika żąda od systemu operacyjnego wykonania komunikacji z urządzeniem peryferyjnym, polegającej na przesłaniu danej z urządzenia.

## Przerwania - schemat działania



System operacyjny sprawdza, czy urządzenie jest gotowe do transmisji badając zawartość rejestru stanu sterownika urządzenia.

## Przerwania - schemat działania



Adres procedury obsługi przerwania nr 1
Adres procedury obsługi przerwania nr 2
.....
Tablica wektorów przerwań
.....
Procedura obsługi przerwania nr 2
.....
Dufor na dane
.....

System operacyjny programuje sterownik urządzenia wpisując odpowiedni rozkaz do jego rejestru sterowania.

## Przerwania - schemat działania



Adres procedury obsługi przerwania nr 1
Adres procedury obsługi przerwania nr 2
.....
Tablica wektorów przerwań
.....
Procedura obsługi przerwania nr 2
.....
Dufor na dane
.....

System operacyjny oddaje procesor procesowi użytkownika. W zależności od rodzaju wykonywanej operacji wejścia-wyjścia może to być ten sam proces, który zażądał wykonania transmisji, lub inny. W tym samym czasie sterownik nadzoruje pracę urządzenia, bez interwencji procesora.

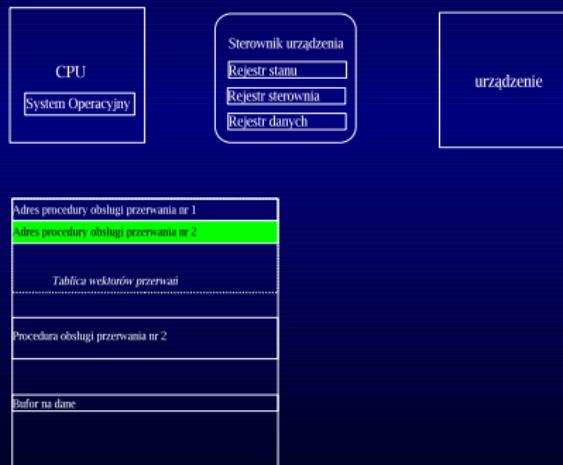
## Przerwania - schemat działania



Adres procedury obsługi przerwania nr 1
Adres procedury obsługi przerwania nr 2
.....
Tаблицa wektorów przerwań
Procedura obsługi przerwania nr 2
.....
Bufor na dane
.....

Kończy się proces transmisji, odebrana informacja jest umieszczona w rejestrze danych sterownika. W rejestrze stanu sterownik ustawia flagę oznaczającą zakończenie komunikacji, a następnie zgłasza przerwanie. Powoduje to automatyczne zapamiętanie bieżącego stanu procesora i adresu powrotu oraz przekazanie sterowania do systemu operacyjnego (objaśnione na kolejnych slajdach).

# Przerwania - schemat działania



Następuje identyfikacja źródła przerwania - w tym przypadku dzięki wektorowemu systemowi przerwań. Z tablicy wektorów przerwań pobierany jest adres procedury obsługi tego przerwania.

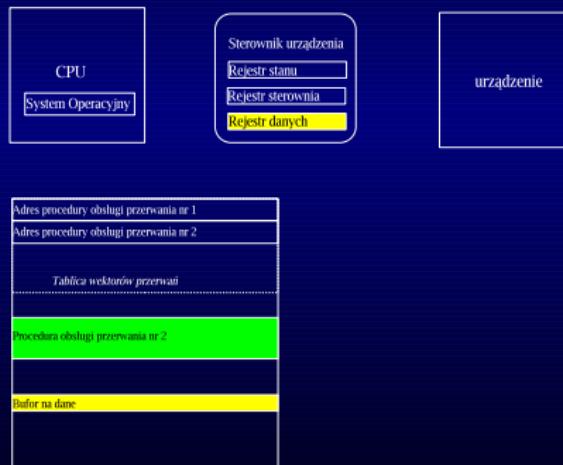
## Przerwania - schemat działania



Adres procedury obsługi przerwania nr 1
Adres procedury obsługi przerwania nr 2
.....
Tablica wektorów przerwań
.....
procedura obsługi przerwania nr 2
.....
Dufor na dane
.....

Sterowanie zostaje przekazane do procedury obsługi przerwania.

# Przerwania - schemat działania



Podprogram obsługi przerwania kopiuje zawartość rejestrów danych sterownika do bufora w pamięci i kończy swoją pracę.

## Przerwania - schemat działania



Adres procedury obsługi przerwania nr 1
Adres procedury obsługi przerwania nr 2
.....
Tablica wektorów przerwań
.....
Procedura obsługi przerwania nr 2
.....
Dufor na dane
.....

Procesor wraca do prze-  
rwanego zadania.

# System przerwań-implementacja.

Podstawowym problemem, po wystąpieniu przerwania, jest zidentyfikowanie jego źródła. Pierwsze procesory, w których zastosowano przerwania miały tylko jedną linię zgłoszenia przerwania (*IRQ*), do której podpięte były wszystkie urządzenia peryferyjne. Jeśli wystąpił sygnał przerwania, to procesor sprawdzał rejesty stanu wszystkich urządzeń, celem znalezienia tego, które zgłosiło żądanie obsługi i uruchamiał odpowiadającą mu procedurę obsługi. Ten tryb ustalania źródła przerwania nazywamy odpytywaniem (ang. *polling*). Procesor musiał również rozstrzygnąć sytuację, w której więcej niż jedno urządzenie zgłosiło konieczność obsługi.

# System przerwań-implementacja.

Bardziej wydajne rozwiązanie polega na zwiększeniu linii zgłoszeń przerwań i zastosowaniu łańcucha priorytetów zgłoszeń. Dzięki temu możliwe jest nie tylko natychmiastowe znalezienie źródła przerwania, ale również określenie kolejności obsługi zgłoszeń przerwań. Zgłoszenie określonego przerwania blokuje możliwość zgłoszenia przerwań o niższym lub równym priorytecie, ale możliwe jest zgłaszanie przerwań o wyższym priorytecie. Wadą tego rozwiązania jest to, że priorytety przerwań są przypisane urządzeniom „na sztywno”.

## System przerwań-implementacja.

Doskonalszym rozwiązaniem jest wektorowy system przerwań. Wymaga on użycia osobnego, programowalnego kontrolera przerwań (ang. PIC). Ten kontroler połączony jest z procesorem. Jeśli urządzenie peryferyjne zgłosi przerwanie, to kontroler je identyfikuje i przekazuje jego numer procesorowi. Numer ten jest indeksem w *tablicy wektorów przerwań* (ang. *ivt*). Wartościami tej tablicy są adresy procedur obsługi przerwań. Każde przerwanie może więc mieć własną procedurę obsługi. Tablica ta może być umieszczona na początku lub końcu pamięci operacyjnej. Nowsze komputery pozwalają systemowi operacyjnemu zdecydować o jej położeniu. Tablica wektorów przerwań wraz z procedurami obsługi przerwań stanowi część systemu operacyjnego. Jest ona również używana razem z łańcuchami priorytetów zgłoszeń.

# System przerwań-implementacja.

Nowsze platformy PC stosują rozwiązanie będące połączeniem odpytywania i wektorowego systemu przerwań. Linie żądania przerwań, które są przypisane przerwaniom o niskim priorytecie mogą być współdzielone przez kilka urządzeń równocześnie. Przez odpytywanie ustala się, które urządzenie zgłosiło przerwanie. Przerwania o wysokim priorytecie nie są współdzielone.

## System przerwań-implementacja.

Innym problemem jest zapamiętanie kontekstu procesora, aby mógł on wrócić do zadania które realizował przed wystąpieniem przerwania. Przez kontekst procesora rozumiemy adres w pamięci operacyjnej (*adres powrotu*) spod którego ma być pobrany następny, po zakończeniu procedury obsługi przerwania, rozkaz do realizacji oraz stan rejestrów procesora. Najmniejszy kontekst obejmuje adres powrotu i register stanu maszyny<sup>1</sup>. Wczesne rozwiązania polegały na wyznaczeniu pewnego niewielkiego, ustalonego miejsca w pamięci komputera, gdzie kontekst był składowany. Nowsze polegają najczęściej na zapamiętaniu kontekstu na stosie.

---

<sup>1</sup>W przypadku procesorów bazujących na architekturze x86 jest to register flag.

# System przerwań-implementacja.

W zależności od tego w jaki sposób zapamiętywany jest kontekst procesora i jak rozwiązać zada jest kwestia identyfikacji źródła przerwania, możemy różnie obsługiwać sytuację, w której przerwania następują szybko po sobie. Najprostsze rozwiązanie polega na wyłączeniu systemu przerwań na czas realizacji procedury obsługi pierwszego zgłoszonego przerwania i ponownym jego włączeniu po jej zakończeniu. Jest to jedyny sposób, który możemy stosować w systemach z odpytywaniem, ale jego zastosowanie nie ogranicza się wyłącznie do nich. W systemach ze współdzieleniem przerwań może być wyłączana tylko linia współdzielona. Systemy priorytetowe pozwalają na selektywne wyłączanie przerwań o niższym lub równym priorytecie. Technika ta nazywa się *maskowaniem*. Możliwe jest również odkładanie na później ich realizacji. We współczesnych systemach operacyjnych, celem zminimalizowania czasu, kiedy określona grupa przerwań lub wszystkie przerwania są wyłączone, procedury obsługi przerwań są podzielone na dwie części zwane połówkami. Góra połówka jest procedurą wykonywaną zaraz po otrzymaniu przerwania. Jej działanie jest krótkie i najczęściej sprowadza się do potwierdzenia odebrania przerwania oraz inicjacji dolnej połówki, której uruchomienie może być odroczone i która wykonuje czasochłonne czynności obsługi przerwania.

# Direct Memory Access-DMA

Opisany wcześniej sposób komunikacji z urządzeniami wejścia-wyjścia, sprawdza się w przypadku dosyć wolnych jednostek, jak np. klawiatura. Takie urządzenia nazywane są *urządzeniami znakowymi*. Gdyby zastosować go do szybkich urządzeń, takich jak np. dysk twardy to okazałoby się, że procesor musiałby większość czasu spędzać wykonując podprogramy obsługi przerwań, ze względu na bardzo dużą liczbę ich zgłoszeń. Rozwiązaniem jest zastosowanie dla tych urządzeń bezpośredniego dostępu do pamięci (*DMA*). System komputerowy jest wyposażony w dodatkowy sterownik zwany *kontrolerem DMA*. Kiedy program użytkownika żąda transmisji danych z szybkiego urządzenia to system operacyjny programuje odpowiednio kontroler DMA, co wymaga między innymi wyznaczenia obszaru w pamięci do którego urządzenie będzie zapisywać dane (lub z którego będzie pobierało je w przypadku transmisji w odwrotnym kierunku) oraz określenia wielkości tych danych. Urządzenie przesyła te dane blokami wielkości kilkuset lub nawet kilku tysięcy bajtów. Sterownik DMA powiadamia odpowiednim przerwaniem procesor, że transmisja została zakończona. **Procesor nie nadzoruje przesyłania danych do lub z pamięci, robi to kontroler DMA.** Urządzenia obsługiwane w ten sposób są nazywane *urządzeniami blokowymi*.

# Pułapki

Wektorowy system przerwań można wykorzystać nie tylko do obsługi komunikacji z urządzeniami zewnętrznymi, ale również do obsługi sytuacji wyjątkowych. Najprostszym przykładem takiej sytuacji jest próba dzielenia przez zero. Przerwania obsługujące zdarzenia tego typu nazywamy pułapkami (ang. trap) lub wyjątkami (ang. exception). Takie przerwania mają wysokie priorytety i są najczęściej *niemaskowalne*. Mogą być również przerwaniami *neprecyzyjnymi*, co oznacza że jeśli sygnał takiego przerwania pojawi się w trakcie realizacji dowolnego rozkazu, to wykonanie tego rozkazu **nie jest kończone**. Przerwania związane z operacjami I/O są na ogół precyzyjne, tzn. ich obsługa zaczyna się po zakończeniu realizacji rozkazu. Pułapki są też najczęściej przerwaniami *synchronicznymi*, tzn. pojawiają się podczas realizacji określonych fragmentów programu, podczas gdy przerwania związane z urządzeniami peryferyjnymi mogą pojawić się podczas realizacji dowolnej instrukcji programu. Dlatego te ostatnie nazywane są przerwaniami *asynchronicznymi*. Od reakcji na sytuacje wyjątkowe zależy stabilność systemu komputerowego. Aby był on maksymalnie stabilny, musi reagować poprawnie na każdą możliwą sytuację krytyczną.

## Obsługa wielu żądań dostępu do urządzenia

Ponieważ działanie systemu operacyjnego w dużej mierze zależy od systemu przerwań, łatwo zauważyc, że jest on *oprogramowaniem reagującym na zdarzenia*. Projektując system przerwań należy uwzględnić sytuację, w której urządzenie wejścia-wyjścia nie jest w stanie obsłużyć szybko otrzymywanych od procesora zleceń. Jest to sytuacja do której dochodzi bardzo często. Ten problem rozwiązuje się tworząc kolejki zleceń przypisane dla każdego urządzenia w komputerze. Jeżeli urządzenie jest wolne, to natychmiast wykonuje skierowane do niego zlecenie, w przeciwnym przypadku zlecenie to trafia do kolejki, gdzie oczekuje na swoją realizację.

## Rozkaz oczekiwania

Pozostaje do rozstrzygnięcia kwestia, co powinien zrobić system operacyjny z procesem, którego żądanie operacji wejścia-wyjścia oczekuje na realizację. Jeśli żądane dotyczyło zapisu lub wysłania danych, to może on kontynuować swoją pracę. W przypadku operacji odbioru bądź odczytu danych system operacyjny może mu pozwolić na aktywne oczekiwanie (co jest jawnym marnotrawieniem czasu procesora) lub umieścić go w kolejce oczekiwania, a procesor oddać innemu procesowi. Większość współczesnych procesorów wyposażona jest w rozkaz *wait*, który zawiesza ich pracę do czasu pojawiения się przerwania. Rozkaz ten jest szczególnie użyteczny w przypadku systemów wbudowanych (ang. *embedded*), jeśli w systemie nie ma zadań, które miałyby być natychmiast wykonane.

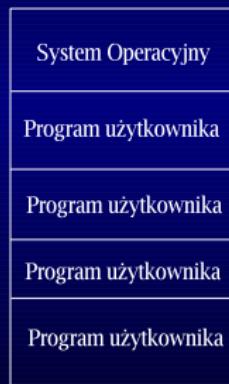
## Prosta ochrona sprzętowa

Zastosowanie pułapek nie gwarantuje samoistnie stabilności systemu. Muszą istnieć dodatkowe mechanizmy, które pozwolą na jej zachowanie. Trzy z nich są konieczne do zbudowania najprostszego systemu ochrony.

# Dualny tryb pracy procesora

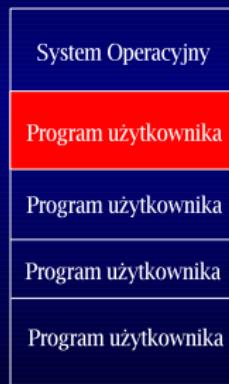
Procesy użytkownika nie powinny mieć możliwości wykonywania pewnych rozkazów. Aby to zapewnić twórcy sprzętu wyposażyli procesory w dwa tryby pracy: tryb jądra (nazywany też trybem monitora, systemu lub nadzorcy) i tryb użytkownika. W pierwszym trybie wykonywany jest oczywiście system operacyjny, w drugim, procesy użytkownika. Przełączenie z trybu użytkownika do trybu monitora następuje między innymi w wyniku wystąpienia przerwania lub pułapki. To w jakim trybie znajduje się w chwili obecnej procesor określone jest ustawieniem odpowiedniego bitu w rejestrze stanu procesora. Lista rozkazów jest podzielona na dwa zbiory: rozkazów uprzywilejowanych, które mogą być wykonywane jedynie w trybie jądra i nieuprzywilejowanych, które mogą być wykonywane zawsze. Rozróżnia je ustawienie odpowiedniego bitu w słowie rozkazu, który jest porównywany z bitem trybu pracy. Jeśli program użytkownika spróbuje wykonać rozkaz uprzywilejowany, to zostanie uruchomiona odpowiednia pułapka, która zakończy jego działanie w sposób krytyczny.

# Prosta ochrona pamięci



W systemach wieloprogramowych system operacyjny i pewna liczba programów użytkownika stale rezyduje w pamięci operacyjnej komputera.

# Prosta ochrona pamięci



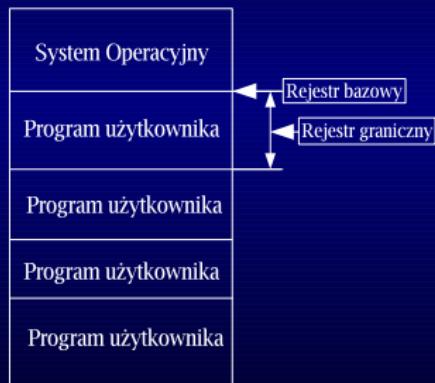
Jeśli wśród tych programów znalazłby się program, który odwoływałby się do pamięci w nieprawidłowy sposób, bądź to na skutek błędu programisty, bądź na skutek celowego, złośliwego działania...

# Prosta ochrona pamięci



... wówczas cały system mógłby być zagrożony.

# Prosta ochrona pamięci

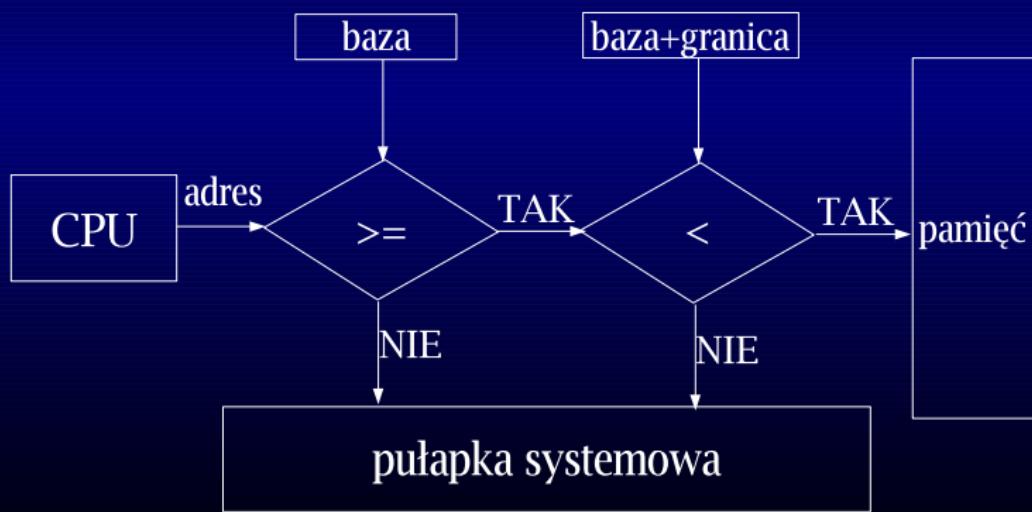


Aby zapobiec takiej sytuacji należy kontrolować każde odwołanie do pamięci bieżąco wykonywanego procesu użytkownika. Najprostsze rozwiązanie polega na zastosowaniu odpowiedniego układu elektronicznego składającego się między innymi z dwóch rejestrów - bazowego i granicznego. Pierwszy zawiera adres początku obszaru pamięci, który system operacyjny przypisał procesowi użytkownika, a drugi jego rozmiar.

# Prosta ochrona pamięci

Każdy adres wygenerowany przez proces użytkownika sprawdzany jest zgodnie ze schematem umieszczonym na następnej planszy. Jeśli test nie powiedzie się uruchamiana jest pułapka i sterowanie wraca do systemu operacyjnego. Rozkazy modyfikujące rejestr bazowy i graniczny muszą być uprzywilejowane.

## Prosta ochrona pamięci



# Czasomierz

Opisane wcześniej mechanizmy nie chronią systemu przed sytuacją, w której proces użytkownika przejmuje procesor i nigdy go nie oddaje (np.: na skutek wystąpienia niekończącej się pętli). Taką ochronę może zagwarantować sprzętowy licznik czasu. Jest on najczęściej implementowany jako licznik zliczający w dół, którego stan zmienia się co takt zegara. Po wyzerowaniu licznika sterownie wraca do systemu operacyjnego. Opisany czasomierz (ang. *timer*) może służyć jako układ typu *watchdog*, który np.: w systemie czasu rzeczywistego pozwalałby sprawdzić, czy zadanie wykonało się w przeznaczonym dla niego czasie. W systemach z podziałem czasu ten układ wyznacza moment, w którym system operacyjny powinien przełączyć procesor na inne zadanie użytkownika. Może on również służyć do (niezbyt dokładnego) wyznaczania pory dnia. Inicjacji czasomierza dokonuje system operacyjny, przed oddaniem sterowania procesowi użytkownika. Rozkaz inicjacji jest rozkazem uprzywilejowanym.

# Odwołania do systemu operacyjnego

Skoro tylko system operacyjny może wykonywać operacje I/O i szereg innych ważnych czynności, to w jaki sposób użytkownika może się komunikować ze światem zewnętrznym lub podejmować inne działania konieczne do jego realizacji? Otóż współczesne komputery udostępniają procesom użytkownika rozkaz<sup>2</sup>, który umożliwia zażądanie od systemu operacyjnego wykonania wymaganych przez nie operacji. Ten rozkaz jest nazywany przerwaniem programowym, ponieważ powoduje przełączenie procesora w tryb monitora, oraz wykonanie odpowiedniej procedury obsługi przerwania. Ta procedura realizuje zamówioną przez proces użytkownika czynność. Nazywana jest ona *wywołaniem systemowym* (ang. *system call*) lub *funkcją systemową*. Proses użytkownika może przekazać jej pewne argumenty, ale zanim zostaną one użyte, muszą być przez procedurę dokładnie sprawdzone. Podsumowując — przerwania programowe umożliwiają procesom użytkownika korzystnie z usług systemu operacyjnego.

---

<sup>2</sup>W procesorach Intel jest to rozkaz INT.

# Systemy wieloprocesorowe

Systemy wieloprocesorowe (równoległe) mają wiele zalet. Do największych z nich należy oczywiście wydajność. Należy jednak pamiętać, że system złożony z  $N$  procesorów nie wykonuje zadań  $N$ -razy szybciej niż system jednoprocesorowy. Przyspieszenie może być proporcjonalne do tej wartości, ale nigdy równe. Inną zaletą jest niezawodność - jeśli system składa się z takich samych procesorów, to w razie awarii jednego z nich, jego obowiązki przejmują pozostałe. Jeśli procesory są różne, takie rozwiązanie nie jest możliwe. Popularnym rozwiązaniem jest symetryczne wieloprzetwarzanie (ang. *SMP*). Polega ono na tym, że w systemie umieszczonych jest kilka jednakowych procesorów, a każdy z nich wykonuje własną kopię oprogramowania systemowego i przydzielone mu procesy użytkowników. Przed systemem operacyjnym stoi zadanie koordynacji pracy tych procesorów. We wczesnych systemach komputerowych popularne było wieloprzetwarzanie asymetryczne. W takim schemacie jeden, uprzywilejowany procesor zajmował się przetwarzaniem danych, pozostałe operacjami I/O. W chwili obecnej każdy komputer jest wyposażony w kontrolery, które zapewniają taki rodzaj wieloprzetwarzania, ale to określenie już nie funkcjonuje.

# Architektury komputerów osobistych

Pierwsze mikroprocesory, takie jak Intel 8086, czy Motorola MC68000 pozbawione były środków sprzętowych wymaganych do zapewnienia choćby minimalnej ochrony systemowwi operacyjnemu i procesom użytkownika. Rzutowało to oczywiście na budowę i sposób pracy pierwszych systemów operacyjnych przeznaczonych na te komputery (MS-DOS, MacOS). Z czasem pojawiły się droższe od komputerów klasy IBM PC stacje robocze (SUN, Apollo, Silicon Graphics), których procesory zapewniały odpowiednie mechanizmy ochrony. Te mechanizmy zaczęły się pojawiać w tańszych rozwiązańach, co umożliwiło pisanie bardziej bezpiecznych i stabilnych systemów operacyjnych lub przenoszenie na komputery osobiste systemów znanych z „dużych” systemów komputerowych. Przykładem rodziny procesorów, która przeszła ewolucję od urządzeń w ogóle pozbawionych środków ochrony do urządzeń z elastycznym systemem ochrony jest rodzina x86. Obecnie procesory te potrafią pracować w czterech trybach nazywanych pierścieniami (ang. *ring*), przy czym najbardziej uprzywilejowany jest pierścień zerowy. Ten mechanizm wzorowany jest na systemie Multics.

# Pytania

?

Koniec

Dziękuję Państwu za uwagę.

# Systemy Operacyjne - struktura

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 19 października 2020

# Plan wykładu

# Plan wykładu

- ① Jeszcze jedna definicja systemu operacyjnego
- ② Elementy (podsystemy) systemu operacyjnego
- ③ Usługi systemu operacyjnego
- ④ Wywołania systemowe
- ⑤ Programy systemowe
- ⑥ Struktury jądra systemu operacyjnego
- ⑦ Projektowanie i implementacja systemu operacyjnego

# Plan wykładu

- ① Jeszcze jedna definicja systemu operacyjnego
- ② Elementy (podsystemy) systemu operacyjnego
- ③ Usługi systemu operacyjnego
- ④ Wywołania systemowe
- ⑤ Programy systemowe
- ⑥ Struktury jądra systemu operacyjnego
- ⑦ Projektowanie i implementacja systemu operacyjnego

# Plan wykładu

- ① Jeszcze jedna definicja systemu operacyjnego
- ② Elementy (podsystemy) systemu operacyjnego
- ③ Usługi systemu operacyjnego
- ④ Wywołania systemowe
- ⑤ Programy systemowe
- ⑥ Struktury jądra systemu operacyjnego
- ⑦ Projektowanie i implementacja systemu operacyjnego

# Plan wykładu

- ① Jeszcze jedna definicja systemu operacyjnego
- ② Elementy (podsystemy) systemu operacyjnego
- ③ Usługi systemu operacyjnego
- ④ Wywołania systemowe
- ⑤ Programy systemowe
- ⑥ Struktury jądra systemu operacyjnego
- ⑦ Projektowanie i implementacja systemu operacyjnego

# Plan wykładu

- ① Jeszcze jedna definicja systemu operacyjnego
- ② Elementy (podsystemy) systemu operacyjnego
- ③ Usługi systemu operacyjnego
- ④ Wywołania systemowe
- ⑤ Programy systemowe
- ⑥ Struktury jądra systemu operacyjnego
- ⑦ Projektowanie i implementacja systemu operacyjnego

# Plan wykładu

- ① Jeszcze jedna definicja systemu operacyjnego
- ② Elementy (podsystemy) systemu operacyjnego
- ③ Usługi systemu operacyjnego
- ④ Wywołania systemowe
- ⑤ Programy systemowe
- ⑥ Struktury jądra systemu operacyjnego
- ⑦ Projektowanie i implementacja systemu operacyjnego

# Plan wykładu

- ① Jeszcze jedna definicja systemu operacyjnego
- ② Elementy (podsystemy) systemu operacyjnego
- ③ Usługi systemu operacyjnego
- ④ Wywołania systemowe
- ⑤ Programy systemowe
- ⑥ Struktury jądra systemu operacyjnego
- ⑦ Projektowanie i implementacja systemu operacyjnego

# System operacyjny-inne spojrzenie

Podobnie jak nie ma jednoznacznej definicji *czym* jest system operacyjny, tak nie ma jednoznacznej definicji *co* nim jest. Termin *system operacyjny* może oznaczać „to co dostarcza producent jako system operacyjny” i obejmować swoim znaczeniem zbiór takich elementów oprogramowania jak: jądro systemu, interpreter poleceń, edytory tekstu itd. Może również określać część oprogramowania systemowego stale rezydującą w pamięci operacyjnej komputera i wykonywaną w trybie monitora procesora, czyli *jądro systemu operacyjnego*. W trakcie tego wykładu system operacyjny będziemy definiować zgodnie z tą drugą możliwością. Wszelkie odstępstwa od tej definicji będą sygnalizowane.

# Elementy systemu operacyjnego

Choć istnieje wiele systemów operacyjnych, to można wyróżnić pewne wspólne elementy, które prawie każdy z nich zawiera. Zaliczają się do nich:

- ① podsystem zarządzania procesami,
- ② podsystem zarządzania pamięcią operacyjną,
- ③ podsystem zarządzania pamięcią pomocniczą,
- ④ podsystem wejścia-wyjścia,
- ⑤ system plików,
- ⑥ podsystem obsługi sieci,
- ⑦ *ochrona*,
- ⑧ *interpretator poleceń*,

# Procesy

Każda praca jest wykonywana w komputerze w ramach procesu. W szczególności każdy uruchomiony program użytkownika jest procesem lub grupą procesów. Aby wykonać swoje zadania procesy muszą dysponować określonymi zasobami. Te zasoby udostępnia im system operacyjny. Do jego zadań należy również ochrona zasobów przed nieprawidłowym użyciem ich przez procesy. Pojedynczy proces jest wykonywany sekwencyjnie, natomiast kilka procesów może być wykonywanych współbieżnie. Koordynacja takiego wykonania jest również zadaniem systemu operacyjnego.

# Obsługa procesów

Czynności, które system operacyjny wykonuje zarządzając procesami obejmują:

- tworzenie i usuwanie procesów użytkowników i systemowych,
- wstrzymywanie i wznowianie wykonania procesów,
- zapewnianie możliwości synchronizacji procesów,
- zapewnianie środków komunikacji między procesami,
- zapewnienie mechanizmów obsługi zakleszczeń (nieobowiązkowe).

# Pamięć operacyjna

Pamięć operacyjna stanowi główny magazyn danych dla procesora. Można ją zobrazować, jako tablicę komórek o wielkości 1 bajta (najpopularniejsze rozwiązanie). Każda z tych komórek posiada swój unikatowy adres. Do pamięci operacyjnej bezpośredni<sup>1</sup> dostęp ma procesor oraz urządzenia obsługiwane w trybie DMA. Ponieważ pamięć operacyjna, jak każda inna ma skończoną wielkość, więc zarządzanie nią jest ważnym zdaniem systemu operacyjnego. Ma to szczególne znaczenie zwłaszcza w systemach wielozadaniowych.

---

<sup>1</sup>W przypadku nowszych komputerów to stwierdzenie nie do końca jest prawdziwe, a to za sprawą pamięci podręcznej (ang. cache).

# Obsługa pamięci operacyjnej

System operacyjny wykonuje następujące czynności w stosunku do pamięci operacyjnej:

- utrzymuje ewidencję obszarów pamięci, które są w danej chwili zajęte, wraz z informacją do kogo one należą,
- decyduje o tym, które procesy zostaną umieszczone w wolnych obszarach pamięci,
- przydziela i zwalnia obszary pamięci, w zależności od zapotrzebowania.

# Pamięć pomocnicza

Pamięć pomocnicza (ang. external memory) realizowana jest w postaci pamięci dyskowej i stanowi uzupełnienie pamięci operacyjnej, która może się okazać niewystarczająca dla procesów użytkownika. Ponieważ dysk twardy jest jednostką wolniejszą od pamięci RAM, to konieczne jest efektywne zarządzanie pamięcią pomocniczą.

# Zarządzanie pamięcią pomocniczą

Do obowiązków systemu operacyjnego, jako zarządcy pamięci pomocniczej należy:

- zarządzanie obszarami wolnymi,
- przydzielanie pamięci pomocniczej,
- planowanie przydziału obszarów pamięci dyskowej.

# System wejścia-wyjścia

Jednym z naczelnych zadań systemu operacyjnego jest ochrona urządzeń peryferyjnych przed nieprawidłowym ich użyciem przez procesy użytkownika. Efektem tej ochrony jest ukrycie przed procesami użytkownika szczegółów obsługi tych urządzeń. Ma to dodatkową zaletę - zwiększa elastyczność systemu. Opisany na poprzednim wykładzie system przerwań pozwala skonstruować wydajny *system wejścia-wyjścia*. Większość ze współczesnych systemów operacyjnych łączy obsługę urządzeń zewnętrznych z usługą plików.

## Zarządzanie urządzeniami wejścia-wyjścia

System operacyjny tworzy warstwę abstrakcji ułatwiającą procesom użytkownika korzystanie z urządzeń zewnętrznych, która może składać się z np.:

- systemu buforowo-notatnikowego,
- interfejsu do podprogramów obsługi urządzeń peryferyjnych,
- podprogramu obsługi urządzeń peryferyjnych.

# Pliki

Zawartość pamięci operacyjnej jest ulotna, tzn. przestaje istnieć wraz z wyłączeniem zasilania. Ważne informacje, w tym dane i programy powinny więc zostać zapamiętane na nośnikach, które pozwalają je przechować w sposób trwały. Istnieje wiele urządzeń, które mogą służyć jako pamięć masowa. Każde z tych urządzeń ma specyficzną budowę i sposób obsługi. Aby ujednolicić dla procesów użytkownika sposób korzystania z tych urządzeń system operacyjny tworzy *system plików*. Plik jest jednostką informacji, która nie jest zależna od specyfiki nośnika na którym jest przechowywana. Struktura plików zależy od ich twórców. Do przechowywania informacji o plikach i ich porządkowania służą *katalogi*. Niektóre systemy operacyjne implementują katalogi jako specjalny rodzaj plików (pliki gromadzące informacje - metadane - o innych plikach).

# Zarządzanie plikami i katalogami

System operacyjny nie tylko tworzy system plików, ale również jest odpowiedzialny za:

- tworzenie i usuwanie plików,
- tworzenie i usuwanie katalogów,
- dostarczanie podstawowych operacji do manipulowania plikami i katalogami,
- odwzorowywanie całości, lub części plików w pamięci operacyjnej,
- umieszczenie plików w pamięci trwałej.

# Sieć

Sieci komputerowe (ang. networks) służą do komunikacji pomiędzy systemami komputerowymi i mogą służyć do budowy tzw. systemów rozproszonych. Sieci mogą mieć różny zasięg i różne topologie. Systemy komputerowe połączone w sieć mogą być jednakowego typu (sieć homogeniczna) lub różnych typów (sieć heterogeniczna).

# Obsługa sieci

W ramach obsługi sieci system operacyjnych może wykonywać następujące czynności:

- wyznaczanie tras pakietów,
- translacja nazw komputerów połączonych w sieć,
- dzielenie i scalanie pakietów,
- nawiązywanie i kończenie połączeń,
- obsługa błędów transmisji.

# System ochrony

Ochrona nie jest jednym spójnym mechanizmem, ale paradoksalnie jest niezbędna do zapewnienia spójności i stabilności działania systemu komputerowego. W skład tego „podsystemu” wchodzą środki pozwalające wykrywać próby nieupoważnionego dostępu do zasobów oraz im zapobiegać.

# Interpreter poleceń

W niektórych systemach operacyjnych (MS-DOS) interpreter poleceń, czyli część systemu umożliwiająca komunikację z użytkownikiem, jest częścią jądra systemu. W większości innych systemów jest to osobny program wchodzący w skład oprogramowania systemowego.

# Usługi systemu operacyjnego

Obok zarządzania zasobami i nadzoru nad procesami system operacyjny dostarcza zarówno procesom użytkowników, jak i samym użytkownikom pewnych usług. Dzięki tym usługom tworzy środowisko w którym mogą się wykonywać procesy użytkownika. To jakie usługi i w jaki sposób dostarcza system operacyjny zależy od wielu czynników, niemniej można wyróżnić kilka grup usług, które są świadczone przez prawie każdy system operacyjny.

# Wykonanie programu

Na życzenie użytkownika system operacyjny powinien załadować określony program do pamięci i umożliwić mu jego wykonanie. Program powinien móc zasygnalizować stan swojego wykonania systemowi operacyjnemu (poprawny/niepoprawny).

# Operacje wejścia-wyjścia

Procesy użytkownika nie powinny mieć możliwości używania urządzeń peryferyjnych bezpośrednio, bo mogłoby to prowadzić do szeregu nadużyć z ich strony. Opracowywanie fragmentów kodu związanego z wejściem-wyjściem byłoby również uciążliwe dla programistów piszących aplikacje. Dlatego to system operacyjny jest wyposażony w odpowiednie elementy umożliwiające procesom użytkownika wykonanie rozważanych operacji.

# Manipulowanie systemem plików.

Ponieważ pliki są tworami kreowanymi przez system operacyjny, to również za jego pośrednictwem muszą być obsługiwane. Usługi związane z plikami obejmują ich tworzenie, usuwanie, otwieranie, odczyt, zapis, jak również przemieszczanie i kopiowanie.

Plan wykładu	Wykonywanie programu
Jeszcze jedna definicja systemu operacyjnego	Operacje wejścia-wyjścia
Podsystemy	Manipulowanie systemem plików
<b>Usługi</b>	<b>Komunikacja</b>
Wypołania systemowe	Wykrywanie wyjątków
Programy systemowe	Przydział zasobów
Struktury jądra	Rozliczanie
Projekt i implementacja	Bezpieczeństwo

# Komunikacja

Możemy wyróżnić dwie kategorie sposobów komunikowania się procesów: *komunikację lokalną* i *komunikację sieciową*. Pierwszy rodzaj komunikacji obejmuje komunikację za pomocą lokalnych łączys lub za pomocą *pamięci dzielonej*. Wszystkie te środki łączności są zapewniane przez system operacyjny.

# Wykrywanie wyjątków

Podczas przetwarzania informacji mogą pojawić się wyjątki. Ich źródłem mogą być nie tylko procesy użytkownika, ale również inne elementy systemu komputerowego. System operacyjny musi zagwarantować wykrywanie wszystkich wyjątków niskopoziomowych i poprawną reakcję na nie.

# Przydział zasobów

Każdy proces do wykonania potrzebuje zasobów. W każdym systemie komputerowym występuje ograniczona liczba zasobów. Zarządzanie zasobami staje się szczególnie ważne w systemach wielozadaniowych i wielodostępnych, gdyż od niego zależy efektywność i wygoda używania komputera. Przydział niektórych rodzajów zasobów może być oprogramowany za pomocą dosyć ogólnego kodu, natomiast przydziały innych rodzajów zasobów będą wymagały szczególnych rozwiązań.

# Rozliczanie

Czas pracy pierwszych systemów komputerowych był cenny, ze względu na wartość materialną tych urządzeń. Należało więc starannie mierzyć czas poświęcony na wykonanie przez system zadania użytkownika, aby móc później przedstawić mu wiarygodny rachunek. Z czasem obowiązek dokonywania pomiaru czasu pracy procesów przejęły systemy operacyjne. W nowszych ich wersjach takie usługi są rzadziej spotykane, ale dały one początek usługom, które pozwalają sporządzać statystyki wykorzystania zasobów komputera i tym samym pozwalają na wprowadzenie do systemu poprawek optymalizacyjnych.

# Bezpieczeństwo

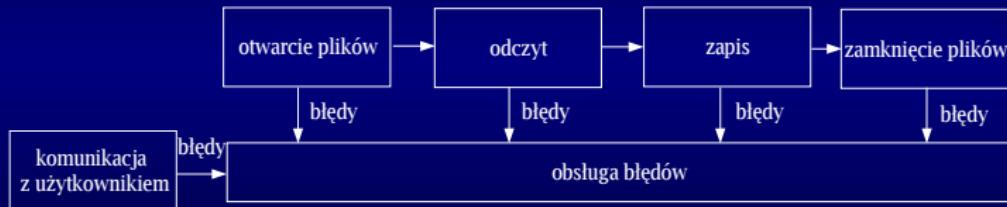
System operacyjny powinien dostarczać swym użytkownikom mechanizmów pozwalających na realizację przyjętej przez nich polityki bezpieczeństwa. Do tych mechanizmów należy zaliczyć prawa dostępu, system uwierzytelniania użytkowników, system rejestracji zdarzeń zachodzących w systemie.

# Wywołania systemowe

Wywołania systemowe (ang. *system calls*) nazywane również *funkcjami systemowymi* są specjalnymi procedurami (lub procedurą) obsługującymi przerwań (lub przerwania), które pozwalają na komunikację między procesami użytkownika, a systemem operacyjnym. Za pomocą wywołań systemowych procesy użytkownika mogą przedstawiać swoje żądania systemowi, a więc tworzą one *interfejs* między tymi dwoma elementami systemu komputerowego. Wywołania systemowe są bezpośrednio dostępne dla programistów piszących aplikacje w języku assemblerowym oraz w niektórych językach wysokiego poziomu (np. C). Częściej jednak mamy do czynienia z pośrednim wywołaniem funkcji systemowych. Języki wysokiego poziomu dostarczają bibliotek podprogramów, które stanowią mniej lub bardziej złożone „opakowania” na wywołania systemowe (np. funkcja printf() w języku C lub procedura write w języku Pascal).

# Przykład

Poniżej przedstawiony jest diagram, który obrazuje z jakich wywołań systemowych może korzystać program kopiujący pliki.



# Argumenty wywołań systemowych

Podobnie jak zwykłe podprogramy wywołania systemowe mogą wymagać pewnych argumentów wywołania. Te argumenty mogą być im przekazywane na trzy różne sposoby:

- ① przez rejesty,
- ② przez stos,
- ③ przez pamięć - adres początku obszaru pamięci zawierającego argumenty umieszczany jest w rejestrach.

# Kategorie wywołań systemowych

Liczba i sposób działania wywołań systemowych jest zależna od usług, jakich system operacyjny dostarcza procesom i użytkownikom. Możemy w związku z tym podzielić funkcje systemowe na kilka kategorii:

- ① wywołania związane z zarządzaniem procesami,
- ② wywołania związane z operacjami na plikach,
- ③ wywołania związane z operacjami na urządzeniach peryferyjnych,
- ④ wywołania związane z utrzymywaniem informacji,
- ⑤ wywołania związane z komunikacją.

# Wywołania związane z nadzorem nad procesami

Do tej kategorii wywołań systemowych należy zaliczyć wywołania służące do tworzenia nowych procesów, ładowania do pamięci programów użytkownika, końca działania procesu, debugowania, profilowania działania procesu, zawieszania działania procesu i synchronizowania procesów.

# Wywołania związane z operacjami na plikach

Ta kategoria obejmuje wywołania związane z tworzeniem, otwieraniem plików, odczytem, zapisem, zmianą pozycji wskaźnika pliku oraz zamknięciem.

# Wywołania związane z operacjami na urządzeniach wejścia-wyjścia

Wiele systemów operacyjnych, na czele z Uniksem łączy system zarządzania urządzeniami zewnętrznymi z systemem plików, dlatego te same wywołania, które służą do obsługi plików są także używane do obsługi urządzeń wejścia-wyjścia. Niektóre funkcje systemowe z tej grupy mogą być specyficzne jedynie dla urządzeń peryferyjnych, np.: montowanie urządzenia w systemie.

# Wywołania związane z utrzymywaniem informacji.

Najprostszymi przykładami wywołań należących do tej kategorii są wywołania pozwalające pobrać bieżący czas i datę. Bardziej skomplikowane pozwalają poznać wszelkie informacje statystyczne związane z systemem, jak: ilość wolnego miejsca na dysku, ilość dostępnej pamięci operacyjnej, liczba użytkowników, itp.

# Wywołania związane z komunikacją między procesami.

W przypadku komunikacji przez sieć lub łącza logiczne muszą istnieć wywołania pozwalające utworzyć połączenie, nadać i odebrać komunikat oraz zamknąć połączenie. W przypadku komunikacji przez pamięć musi istnieć funkcja systemowa pozwalająca zażądać od systemu operacyjnego obszaru pamięci, który będzie współdzielony przez dwa lub większą liczbę procesów równocześnie.

# Programy systemowe

Wraz z niemalże każdym systemem operacyjnym dostarczane są programy, które nie stanowią części jądra systemu, ale należy je zaliczyć do oprogramowania systemowego. Te programy również możemy podzielić na kategorie, w zależności do czego służą:

- manipulowanie plikami,
- informowanie o stanie systemu,
- tworzenie i zmienianie zawartości plików,
- translacja języków programowania,
- komunikacja,
- programy użytkowe.

# Programy do manipulowania plikami

Do tej grupy należy zaliczyć programy kopiące (copy, xcopy, cp), przeknoszące (move, mv), usuwające (rm, erase) oraz tworzące pliki (touch) i podobne działające na katalogach (ls, dir, mkdir, rmdir).

# Programy do uzyskiwania informacji o systemie

Do tej kategorii należą programy pozwalające poznać liczbę użytkowników korzystających z systemu (who, w, users), ilość wolnego miejsca na dysku (df), informacje na temat dostępnej pamięci operacyjnej (vmstat, free, mem), datę i czas (date, time) i inne informacje o stanie systemu.

# Programy do przetwarzania plików.

W tej grupie znajdują się zarówno edytory tekstów typu Notatnik, WordPad, VIM, Emasc, Norton Editor, jak i specjalistyczne narzędzia do przetwarzania plików tekstowych (sed,awk,TEX,troff)

# Translatory języków programowania

Do tej kategorii należy zaliczyć kompilatory i interpretery dostarczane wraz z systemem operacyjnym. Przykładami takich programów są gcc, python, perl, itd.

# Programy systemowe związane z komunikacją.

W tej grupie znajdują się programy związane zarówno z diagnostyką sieci komputerowych (ping, traceroute, tracert), jak również pozwalające na prostą komunikację między użytkownikami (mail, talk, WinPopUp, Windows Messenger), oraz programy udostępniające pewne usługi, zwane demonami lub serwerami (sshd, nfsd). Korzystanie z usług ostatniej kategorii programów komunikacyjnych jest możliwe za pomocą programów klienckich.

# Programy użytkowe

Ta kategoria jest dosyć szeroka. Może obejmować zaawansowane edytory tekstu, arkusze kalkulacyjne, programy graficzne, gry i podobne oprogramowanie. Najważniejszym programem, który należy do tej kategorii, a który jest zawsze dostarczany wraz z systemem operacyjnym jest *interpreter poleceń*, program służący użytkownikowi do komunikacji z systemem operacyjnym. Są co najmniej trzy rodzaje takich programów:

- ① interpreterы tekstowe,
- ② interpreterы graficzne 2D,
- ③ interpreterы graficzne 3D.

# Interpretery tekstowe

Interpretery pracujące w środowisku tekstowym pozwalają komunikować się użytkownikowi z komputerem za pomocą *wiersza poleceń* (ang. command line). Prostym przykładem takiego interpretera jest command.com z systemu MS-DOS. Ładując do pamięci program, który mu został przedłożony do wykonania, nie tworzy nowego procesu lecz usuwa fragment siebie, zwalniając tym samym fragment pamięci operacyjnej, który przeznacza dla programu użytkownika. Po zakończeniu wykonania programu, sterowanie wraca do interpretera, który odbudowuje się. Bardziej wyrafinowaną postacią interpreterów poleceń są powłoki (ang. shell) w systemie Unix (bash,tcsh,ksh). Są one wykonywane jako osobne procesy. Kiedy muszą wykonać inny proces, to tworzą proces potomny, którego program jest zastępowany zleconym zadaniem. W zależności od sposobu uruchomienia nowego procesu sterowanie może wrócić natychmiast do interpretera, lub po zakończeniu procesu. Polecenia powłoki mogą stanowić część jej kodu, bądź być osobnymi programami.

# Interpretery graficzne 2D

Interpretery graficzne 2D tworzą *graficzny interfejs użytkownika* (GUI), pozwalając porozumiewać się z komputerem za pomocą wskaźnika myszy oraz systemu okien i ikon. Interpretery te mogą być na stałe zintegrowane z systemem operacyjnym (GUI systemów Mac OS, explorer w MS-Windows) lub być osobnym rozbudowanym systemem, takim jak X Window, który umożliwia nawet zdalną pracę przez sieć i zmianę interfejsu użytkownika, od prostych zarządców okien (fvwm2, Enlightenment, Window Maker), po całe środowiska graficzne (Gnome, KDE).

# Interpretery 3D

Interpretery graficzne 3D są dosyć nową próbą dodania do znanych środowisk graficznych efektu przestrzennego. Wiąże się to z rosnącym wsparciem sprzętowym dla trójwymiarowych operacji graficznych. Przykładami takich środowisk są Looking Glass, XGL (Compiz i Beryl), AIGLX (Compiz i Beryl), Aero. Pojawiły się również rozwiązania określane mianem interfejsów 2.5D (Metisse).

# Struktury jądra systemu operacyjnego

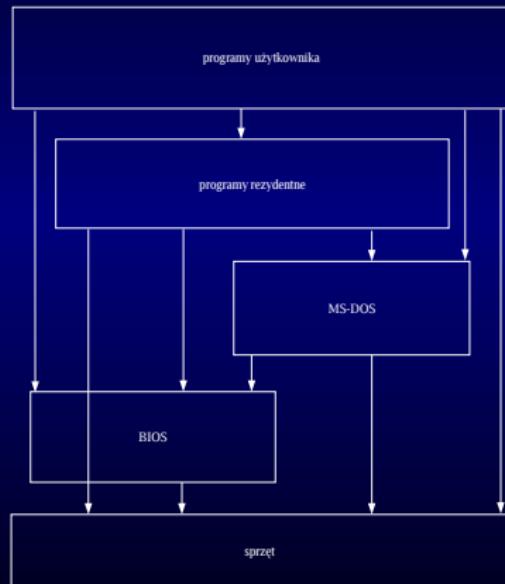
Ponieważ napisanie jądra systemu operacyjnego jest złożonym przedsięwzięciem, to musi je poprzedzić proces przygotowań w wyniku którego zostają podjęte decyzje, co do struktury i funkcjonowania jądra. Na następnych planszach zostaną przedstawione ogólne rozwiązania takiego problemu.

# Jądro monolityczne



Jądro monolityczne jest jednym programem, podzielonym na podprogramy, które wzajemnie są ze sobą powiązane. Brak w nich wyraźnej struktury, lub jest ona dosyć luźna. Przykłady: Unix, MS-DOS, MS-Windows 95, 98, ME.

# Jądro systemu MS - DOS



Jak wynika z zamieszczonego obok diagramu, procesy użytkownika mogą korzystać zarówno z funkcji dostarczanych przez system DOS, jak i z usług dostarczanych przez rezydentne programy systemowe oraz BIOS. Mogą uzyskiwać również bezpośredni dostęp do sprzętu.

# Jądro systemu Unix

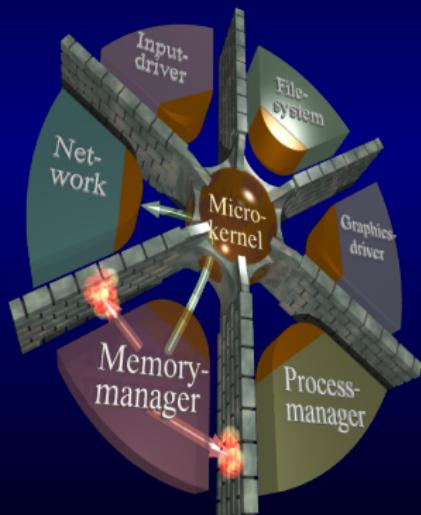


Oryginalne jądro systemu Unix było zaprojektowane dla sprzętu nieposiadającego żadnego mechanizmu ochrony. Mimo to twórcy systemu postanowili dokładnie odseparować procesy użytkownika od sprzętu.

# Jądro monolityczne z modułami

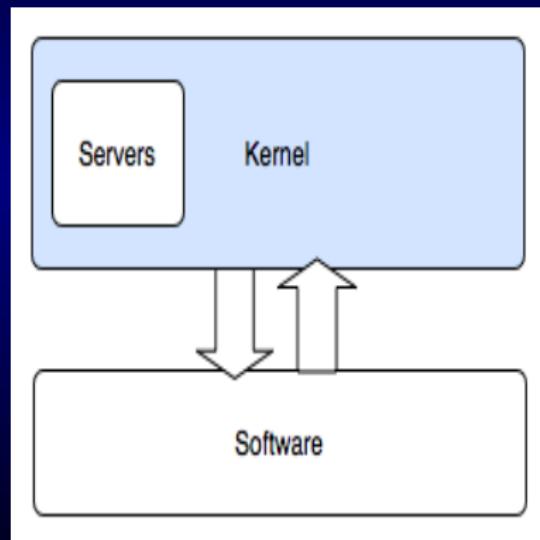
Jest to pewna modyfikacja jądra monolitycznego, pozwalająca na ładowanie w trakcie działania jądra pewnych jego fragmentów (np.: sterowników urządzeń) do pamięci, na podobnej zasadzie, jak programy użytkowników ładują biblioteki współdzielone. Jądro takie musi być wyposażone w dodatkowe elementy: tablicę symboli, mechanizm ładowania modułu i mechanizm śledzenia zależności między modułami. Przykładami systemów z jądrem modularnym są Linux i FreeBSD.

# Mikrojądro (ang. microkernel)



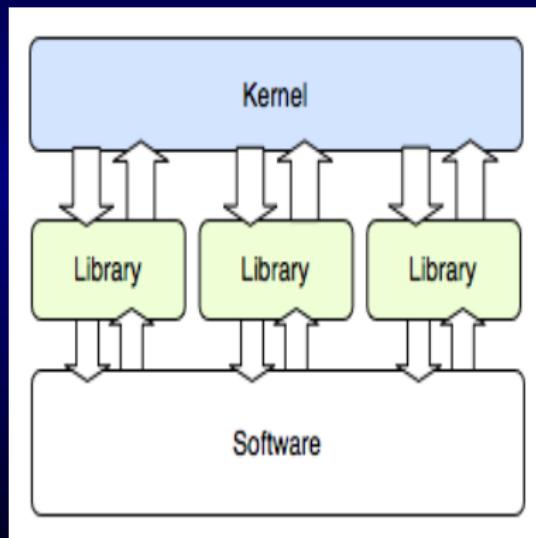
Jądro ma niewielkie rozmiary i jest wykonywane w trybie systemowym procesora. Zadania jądra sprowadzają się do zarządzania procesami, pamięcią operacyjną i zapewnienia komunikacji międzyprocesowej. Inne czynności wykonywane tradycyjnie przez jądro przejmują programy-demony pracujące w trybie użytkownika, lub pośrednim.

# Jądro hybrydowe



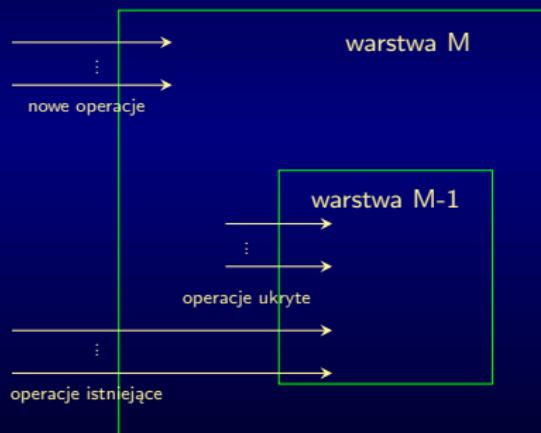
Jest to rozwiązanie pośrednie między jądem monolitycznym, a mikrojądem. Wszystkie serwisy są uruchamiane w trybie jądra. Część ekspertów uważa, że taka kategoria nie istnieje naprawdę i jest tylko chwytem marketingowym, a jądro hybrydowe jest zwykłym jądem monolitycznym, z dobrze określoną strukturą wewnętrzną. Przykłady: Windows NT, 2000, XP.

# Egzojadro (ang. exokernel)



Egzojadro (ang. exokernel) jest nową koncepcją w dziedzinie systemów operacyjnych. Jądro jest jeszcze mniejszych rozmiarów niż w przypadku mikrojadra. Jego jedynym zadaniem jest zapewnienie ochrony zasobów na niskim poziomie. Wszelkie rodzaje abstrakcji, takie jak np.: system plików są dostarczane procesom użytkownika za pomocą zewnętrznych bibliotek. Podobne rozwiązania noszą nazwy pikojąder i nanojaдер.

# Jądro z podziałem na warstwy.



Jądro systemu jest podzielone na współpracujące ze sobą warstwy. Każda warstwa posiada określony interfejs, który udostępnia warstwie następnej. Ta z kolei może udostępniać bezpośrednio część funkcji z warstwy poprzedniej, warstwie która znajduje się nad nią, może ukrywać część funkcji z warstwy poprzedniej i może definiować własne funkcje. Przykłady: THE, Venus, MULTICS.

# Maszyny wirtualne

Maszyny wirtualne są naturalnym rozwinięciem idei podziału jądra na warstwy. Taka maszyna jest po prostu wirtualną kopią komputera, którą otrzymuje każdy z uruchomionych programów. Maszyna ta może pracować w rzeczywistym trybie użytkownika i wykonywać system operacyjny, pracujący w wirtualnym trybie systemowym, który będzie wykonywał procesy działające w wirtualnym trybie użytkownika. Jednym z pierwszych systemów wykorzystujących koncepcję maszyn wirtualnych był IBM VM/370. Obecnie ta idea zyskuje na popularności dzięki pojawienniu się w najnowszych procesorach sprzętowego wsparcia dla wirtualizacji.

# Projekt systemu operacyjnego

Podobnie jak w przypadku innych dużych projektów, nie ma gotowych przepisów na napisanie systemu operacyjnego. Projekt takiego oprogramowania jest kompromisem między wymaganiami narzucanymi przez użytkowników (wygoda obsługi), programistów (wygoda tworzenia nowych aplikacji, utrzymania (ang. maintenance)) i sprzęt (dostępne urządzenia, mechanizmy ochrony). Również jak w przypadku innego oprogramowania przydatne są zasady inżynierii oprogramowania (np.: podział projektu na mniejsze części) i mniej oficjalne reguły (KISS - Keep It Simple, Stupid, DRY - Don't Repeat Yourself).

# Polityka i mechanizmy

Podstawową zasadą przy tworzeniu systemów operacyjnych jest oddzielenie mechanizmu od polityki. Polityka określa *co* ma być zrobione, natomiast mechanizm określa *w jaki sposób*. Mechanizmy powinny być na tyle elastyczne, aby pozwalały zrealizować dowolną politykę. Twórca systemu nie powinien również narzucać jego użytkownikom (głównie administratorom) żadnej polityki. Przykład: w systemie trzeba uwierzytelnić użytkowników (polityka), może to się odbywać za pomocą haseł statycznych, zmiennych w czasie lub za pomocą systemów biometrycznych (mechanizmy).

# Języki programowania

Początkowo wszystkie systemy operacyjne były pisane w języku assemblyowym. Z czasem pojawiły się języki wysokiego poziomu, których można było użyć zamiast języków niskopoziomowych, takimi językami były Bliss, Concurrent Pascal, niektóre dialekty języka Fortran. Prawdziwym przełomem było pojawienie się języka C, który został opracowany specjalnie na potrzeby prac nad systemem Unix. Większość współczesnych systemów jest napisana w tym języku. Obecnie używane są również C++ (Haiku, UnixLite), Java (JavaOS, JNode) i C# (Singularity, OSharp, Cosmos).

# Instalacja

Istnieją dwa skrajne scenariusze instalowania systemów operacyjnych: komplikacja całości systemu, dzięki czemu otrzymujemy system całkowicie dostosowany do sprzętu którym dysponujemy i do naszych potrzeb oraz instalacja wersji binarnej, co jest niewątpliwie szybszym rozwiązaniem. Rozwiążanie pośrednie polega na zainstalowaniu części binarnej i komplikacji elementów systemu, które mogą mieć wpływ na efektywność korzystania z systemu komputerowego.

# Pytania

?

# Źródła

Rysunki monojądra i mikrojądra pochodzą ze strony TU Dresen: Operating System Group. Schematy jądra hybrydowego i egzojądra pochodzą ze stron Wikipedii i są chronione przez licencję Creative Commons.

# Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne - zarządzanie procesami

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 26 października 2020

# Plan wykładu

## 1 Procesy

- Proces sekwencyjny
- Cykl życia procesu
- Deskryptor procesu
- Procesy współbieżne

## 2 Wątki

## 3 Planowanie przydziału procesora

- Motywacja
- Kolejki
- Planiści
- Przełączanie kontekstu

## 4 Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## 5 Szeregowanie w systemach wieloprocesorowych

## 6 Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- Motywacja
- Kolejki
- Planiści
- Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- Motywacja
- Kolejki
- Planiści
- Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- Motywacja
- Kolejki
- Planiści
- Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- Motywacja
- Kolejki
- Planiści
- Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- Motywacja
- Kolejki
- Planiści
- Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

### ③ Planowanie przydziału procesora

- Motywacja
- Kolejki
- Planiści
- Przełączanie kontekstu

### ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

### ⑤ Szeregowanie w systemach wieloprocesorowych

### ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- Algorytm FCFS
- Algorytm SJF
- Algorytm SRT
- Algorytm priorytetowy
- Algorytm rotacyjny
- Wielopoziomowe kolejki
- Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

- ① Procesy
  - ① Proces sekwencyjny
  - ② Cykl życia procesu
  - ③ Deskryptor procesu
  - ④ Procesy współbieżne
- ② Wątki
- ③ Planowanie przydziału procesora
  - ① Motywacja
  - ② Kolejki
  - ③ Planiści
  - ④ Przełączanie kontekstu
- ④ Algorytmy szeregowania procesów
  - ① Algorytm FCFS
  - ② Algorytm SJF
  - ③ Algorytm SRT
  - ④ Algorytm priorytetowy
  - ⑤ Algorytm rotacyjny
  - ⑥ Wielopoziomowe kolejki
  - ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
  - ⑧ Przykłady
- ⑤ Szeregowanie w systemach wieloprocesorowych
- ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- ① Algorytm FCFS
- ② Algorytm SJF
- ③ Algorytm SRT
- ④ Algorytm priorytetowy
- ⑤ Algorytm rotacyjny
- ⑥ Wielopoziomowe kolejki
- ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- ⑧ Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- ① Algorytm FCFS
- ② Algorytm SJF
- ③ Algorytm SRT
- ④ Algorytm priorytetowy
- ⑤ Algorytm rotacyjny
- ⑥ Wielopoziomowe kolejki
- ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- ⑧ Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

- ① Procesy
  - ① Proces sekwencyjny
  - ② Cykl życia procesu
  - ③ Deskryptor procesu
  - ④ Procesy współbieżne
- ② Wątki
- ③ Planowanie przydziału procesora
  - ① Motywacja
  - ② Kolejki
  - ③ Planiści
  - ④ Przełączanie kontekstu
- ④ Algorytmy szeregowania procesów
  - ① Algorytm FCFS
  - ② Algorytm SJF
  - ③ Algorytm SRT
  - ④ Algorytm priorytetowy
  - ⑤ Algorytm rotacyjny
  - ⑥ Wielopoziomowe kolejki
  - ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
  - ⑧ Przykłady
- ⑤ Szeregowanie w systemach wieloprocesorowych
- ⑥ Ocena algorytmów

# Plan wykładu

- ① Procesy
  - ① Proces sekwencyjny
  - ② Cykl życia procesu
  - ③ Deskryptor procesu
  - ④ Procesy współbieżne
- ② Wątki
- ③ Planowanie przydziału procesora
  - ① Motywacja
  - ② Kolejki
  - ③ Planiści
  - ④ Przełączanie kontekstu
- ④ Algorytmy szeregowania procesów
  - ① Algorytm FCFS
  - ② Algorytm SJF
  - ③ Algorytm SRT
  - ④ Algorytm priorytetowy
  - ⑤ Algorytm rotacyjny
  - ⑥ Wielopoziomowe kolejki
  - ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
  - ⑧ Przykłady
- ⑤ Szeregowanie w systemach wieloprocesorowych
- ⑥ Ocena algorytmów

# Plan wykładu

## ① Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## ② Wątki

## ③ Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## ④ Algorytmy szeregowania procesów

- ① Algorytm FCFS
- ② Algorytm SJF
- ③ Algorytm SRT
- ④ Algorytm priorytetowy
- ⑤ Algorytm rotacyjny
- ⑥ Wielopoziomowe kolejki
- ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- ⑧ Przykłady

## ⑤ Szeregowanie w systemach wieloprocesorowych

## ⑥ Ocena algorytmów

# Plan wykładu

- ① Procesy
  - ① Proces sekwencyjny
  - ② Cykl życia procesu
  - ③ Deskryptor procesu
  - ④ Procesy współbieżne
- ② Wątki
- ③ Planowanie przydziału procesora
  - ① Motywacja
  - ② Kolejki
  - ③ Planiści
  - ④ Przełączanie kontekstu
- ④ Algorytmy szeregowania procesów
  - ① Algorytm FCFS
  - ② Algorytm SJF
  - ③ Algorytm SRT
  - ④ Algorytm priorytetowy
  - ⑤ Algorytm rotacyjny
  - ⑥ Wielopoziomowe kolejki
  - ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
  - ⑧ Przykłady
- ⑤ Szeregowanie w systemach wieloprocesorowych
- ⑥ Ocena algorytmów

# Plan wykładu

- ① Procesy
  - ① Proces sekwencyjny
  - ② Cykl życia procesu
  - ③ Deskryptor procesu
  - ④ Procesy współbieżne
- ② Wątki
- ③ Planowanie przydziału procesora
  - ① Motywacja
  - ② Kolejki
  - ③ Planiści
  - ④ Przełączanie kontekstu
- ④ Algorytmy szeregowania procesów
  - ① Algorytm FCFS
  - ② Algorytm SJF
  - ③ Algorytm SRT
  - ④ Algorytm priorytetowy
  - ⑤ Algorytm rotacyjny
  - ⑥ Wielopoziomowe kolejki
  - ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
  - ⑧ Przykłady
- ⑤ Szeregowanie w systemach wieloprocesorowych
- ⑥ Ocena algorytmów

# Plan wykładu

- ① Procesy
  - ① Proces sekwencyjny
  - ② Cykl życia procesu
  - ③ Deskryptor procesu
  - ④ Procesy współbieżne
- ② Wątki
- ③ Planowanie przydziału procesora
  - ① Motywacja
  - ② Kolejki
  - ③ Planiści
  - ④ Przełączanie kontekstu
- ④ Algorytmy szeregowania procesów
  - ① Algorytm FCFS
  - ② Algorytm SJF
  - ③ Algorytm SRT
  - ④ Algorytm priorytetowy
  - ⑤ Algorytm rotacyjny
  - ⑥ Wielopoziomowe kolejki
  - ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
  - ⑧ Przykłady
- ⑤ Szeregowanie w systemach wieloprocesorowych
- ⑥ Ocena algorytmów

# Plan wykładu

- ① Procesy
  - ① Proces sekwencyjny
  - ② Cykl życia procesu
  - ③ Deskryptor procesu
  - ④ Procesy współbieżne
- ② Wątki
- ③ Planowanie przydziału procesora
  - ① Motywacja
  - ② Kolejki
  - ③ Planiści
  - ④ Przełączanie kontekstu
- ④ Algorytmy szeregowania procesów
  - ① Algorytm FCFS
  - ② Algorytm SJF
  - ③ Algorytm SRT
  - ④ Algorytm priorytetowy
  - ⑤ Algorytm rotacyjny
  - ⑥ Wielopoziomowe kolejki
  - ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
  - ⑧ Przykłady
- ⑤ Szeregowanie w systemach wieloprocesorowych
- ⑥ Ocena algorytmów

# Plan wykładu

## 1 Procesy

- ① Proces sekwencyjny
- ② Cykl życia procesu
- ③ Deskryptor procesu
- ④ Procesy współbieżne

## 2 Wątki

## 3 Planowanie przydziału procesora

- ① Motywacja
- ② Kolejki
- ③ Planiści
- ④ Przełączanie kontekstu

## 4 Algorytmy szeregowania procesów

- ① Algorytm FCFS
- ② Algorytm SJF
- ③ Algorytm SRT
- ④ Algorytm priorytetowy
- ⑤ Algorytm rotacyjny
- ⑥ Wielopoziomowe kolejki
- ⑦ Wielopoziomowe kolejki ze sprzężeniem zwrotnym
- ⑧ Przykłady

## 5 Szeregowanie w systemach wieloprocesorowych

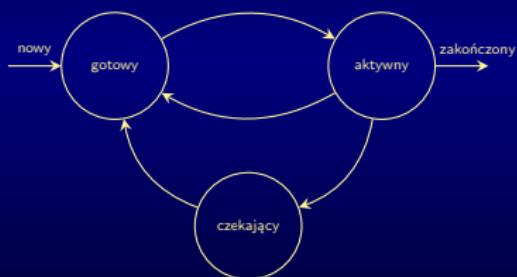
## 6 Ocena algorytmów

# Proces sekwencyjny

## Definicja

Program komputerowy, który został załadowany do pamięci komputera i który jest wykonywany nazywa się *procesem sekwencyjnym*. Wyraz *sekwencyjny* oznacza, że instrukcje programu są wykonywane przez proces w takiej kolejności, w jakiej zostały umieszczone w pliku z kodem wynikowym. Podczas wykonania proces może tworzyć inne procesy sekwencyjne, które mogą być wykonywane *współbieżnie*. Każdy proces posiada swój odrębny fragment pamięci operacyjnej i zestaw adresów przez który może się do niej odwoływać, czyli własną *przestrzeń adresową*. Wewnętrznie pamięć procesu jest podzielona na obszary kodu, danych, stosu, sterty i inne. W rzeczywistości procesy mogą współdzielić fragmenty swej pamięci ze względu na oszczędności (obszar kodu, biblioteki współdzielone) lub z konieczności komunikacji (pamięć dzielona). Wymennie proces nazywany jest *zadaniem*.

# Cykl życia procesu



*Stan* procesu określa etap jego realizacji. Podczas przetwarzania procesu jego stan ulega zmianom. Graf obok ilustruje możliwości zmiany stanu procesu. Jest to diagram uogólniony dla większości systemów operacyjnych. Stan *gotowy* oznacza, że proces *może być* wykonany przez procesor. Stan *aktywny* oznacza, że proces jest w danej chwili wykonywany przez procesor. W tym stanie może znajdować się tylko jeden proces w komputerze jednoprocesorowym. Proces w stanie *czekający* oczekuje na zdarzenie (np.: zakończenie operacji wejścia-wyjścia) i nie może być w chwili bieżącej wykonywany przez procesor.

## Deskryptor procesu

*Deskryptor procesu* zwany również *blokiem kontrolnym procesu* jest to zmienna (w języku Pascal byłby to rekord, w języku C struktura) przechowująca informacje na temat procesu. Jest niezbędna do zarządzania procesami i jest tworzona dla każdego z nich przez system operacyjny. Każdy deskryptor procesu jest więc strukturą należącą do systemu operacyjnego i jest umieszczany w jego przestrzeni adresowej. Blok kontrolny procesu zawiera takie informacje, jak: stan procesu, stan licznika rozkazów i innych rejestrów procesora dla tego procesu, priorytet procesu, informacje o pamięci procesu, itd. Posiada on też pola wskaźnikowe, które pozwalają go łączyć, wraz z innymi blokami w większe struktury (listy, drzewa, itd.).

# Współbieżność

Jeśli w pamięci systemu komputerowego rezyduje równocześnie kilka procesów, to mogą one być wykonywane *współbieżnie*. W przypadku komputerów jednoprocesorowych oznacza to, że procesor co pewien czas jest przydzielany na zmianę różnym procesom. Taką współbieżność nazywamy *pseudorównoległością*. W systemach wieloprocesorowych każdy z procesorów może w danej chwili wykonywać odrębny proces. Takie przetwarzanie nazywamy *przetwarzaniem równoległym*. Za stosowaniem współbieżności przemawia możliwość podziału zasobów fizycznych i logicznych między wielu użytkowników, przyspieszenie przetwarzania (w architekturach wieloprocesorowych), podział programu na niezależnie wykonywane jednostki i zwiększenie stopnia wykorzystania systemu komputerowego.

## Tworzenie procesów

System operacyjny może utworzyć nowy proces na żądanie innego, istniejącego procesu. Proces tworzący nazywany jest *procesem macierzystym*. Tworzenie nowego procesu realizowane jest przez odpowiednie wywołanie systemowe. System operacyjny może przypdzielić nowemu procesowi zasoby z puli zasobów wolnych lub część zasobów należących do procesu macierzystego. Ten drugi sposób zapobiega nadmiernemu rozmnażaniu procesów. Proces macierzysty może również przekazać procesowi potomnemu niezbędne do jego wykonania informacje. Po utworzeniu procesu potomnego proces rodzicielski może wykonywać się z nim współbieżnie, lub czekać na jego zakończenie. Powiązania „rodzinne” między procesami są odnotowywane w ich deskryptorach.

### Przykład

W systemie Unix do tworzenia nowego procesu wykorzystywane jest wywołanie `fork()`, tworzy ono nowy proces, który współdzieli z procesem macierzystym obszar kodu, ale ma odrębny obszar danych. Wywołanie `fork()` zwraca wartość 0 dla procesu potomnego, a procesowi macierzystemu zwraca identyfikator potomka (PID). Jeśli programista chce aby nowy proces wykonywał inny kod niż jego proces macierzysty, to może zastąpić go poprzez użycie wywołania systemowego `execve()`.

## Kończenie procesu

Proces potomny może zakończyć się i przekazać swój kod zakończenia za pomocą odpowiedniego wywołania systemowego (w Uniksie `exit()`) procesowi macierzystemu. Proces macierzysty może również zakończyć działanie procesu potomnego posługując się jego identyfikatorem i odpowiednim wywołaniem systemowym (w Uniksie `kill()`). Za sytuację anormalną uznaje się zwykle zakończenie procesu macierzystego przed procesami potomnymi. Oznacza to, że żaden proces nie czeka na wyniki ich pracy (w Uniksie to oczekiwanie jest realizowane za pomocą wywołania systemowego `wait()`). Systemy operacyjne mogą w różny sposób obsługiwać tę sytuację. Część z nich kończy kaskadowo działanie osieroconych procesów potomnych, inne stosują swoisty mechanizm adopcji takich procesów. Do tej ostatniej grupy należy Unix. Procesy osierocone w tym systemie są przekazywane procesowi, który się nigdy nie kończy (`init`) i który co pewien czas wywołuje funkcję systemową `wait()`.

# Współpraca między procesami współbieżnymi

W zależności od intencji programisty i usług dostarczanych przez system operacyjny procesy współbieżne mogą ze sobą współpracować bądź też nie. Oto porównanie takich procesów:

## Proces niezależny

- na jego stan nie wpływa żaden inny proces,
- jego działanie jest deterministyczne,
- jego działanie daje się powielać,
- jego działanie może być wstrzymywane i wznowiane bez żadnych skutków ubocznych.

## Proces współpracujący

- jego stan jest dzielony z innymi procesami,
- jego wykonanie jest niedeterministyczne,
- wynik jego działania może zależeć od wykonania innych procesów.

# Współpraca między procesami współbieżnymi

W zależności od intencji programisty i usług dostarczanych przez system operacyjny procesy współbieżne mogą ze sobą współpracować bądź też nie. Oto porównanie takich procesów:

## Proces niezależny

- na jego stan nie wpływa żaden inny proces,
- jego działanie jest deterministyczne,
- jego działanie daje się powielać,
- jego działanie może być wstrzymywane i wznowiane bez żadnych skutków ubocznych.

## Proces współpracujący

- jego stan jest dzielony z innymi procesami,
- jego wykonanie jest niedeterministyczne,
- wynik jego działania może zależeć od wykonania innych procesów.

# Współpraca między procesami współbieżnymi

W zależności od intencji programisty i usług dostarczanych przez system operacyjny procesy współbieżne mogą ze sobą współpracować bądź też nie. Oto porównanie takich procesów:

## Proces niezależny

- na jego stan nie wpływa żaden inny proces,
- jego działanie jest deterministyczne,
- jego działanie daje się powielać,
- jego działanie może być wstrzymywane i wznowiane bez żadnych skutków ubocznych.

## Proces współpracujący

- jego stan jest dzielony z innymi procesami,
- jego wykonanie jest niedeterministyczne,
- wynik jego działania może zależeć od wykonania innych procesów.

# Współpraca między procesami współbieżnymi

W zależności od intencji programisty i usług dostarczanych przez system operacyjny procesy współbieżne mogą ze sobą współpracować bądź też nie. Oto porównanie takich procesów:

## Proces niezależny

- na jego stan nie wpływa żaden inny proces,
- jego działanie jest deterministyczne,
- jego działanie daje się powielać,
- jego działanie może być wstrzymywane i wznowiane bez żadnych skutków ubocznych.

## Proces współpracujący

- jego stan jest dzielony z innymi procesami,
- jego wykonanie jest niedeterministyczne,
- wynik jego działania może zależeć od wykonania innych procesów.

## Wątki

Wątek (ang. thread) jest pojęciem pokrewnym pojęciu procesu. Podstawowa różnica polega na tym, że wątki nie mają osobnej przestrzeni adresowej, a współdzielą ten sam obszar pamięci. Pojedynczy proces może być wykonywany jako jeden wątek lub jako grupa wątków. Za stosowaniem wątków przemawia fakt, że przełączanie procesora między nimi jest na ogół mniej kosztowne niż przełączanie między procesami (trzeba zapamiętać mniej informacji). W niektórych systemach operacyjnych różnica w tych kosztach jest duża (Windows), w innych mała lub wręcz niewielka (Linux). Wątki mogą poprawiać interaktywność aplikacji (GUI) lub efektywność ich działania (demony). Mogą być one implementowane całkowicie w trybie użytkownika (biblioteka pth), mogą one być obsługiwane przez system operacyjny (Solaris, Windows, wywołanie clone() w Linuksie) lub można jednocześnie stosować oba podejścia (Java w różnych wersjach i dla różnych platform systemowych). Szeregowanie wątków może przebiegać na podstawie zawartości macierzy szeregowania lub podlegać takim samym mechanizmom jak szeregowanie zwykłych procesów. Wątki nazywane są często procesami lekkimi (ang. lightweight process), a tradycyjne procesy procesami ciężkimi (ang. heavy process). Ze względu na współdzielenie przestrzeni adresowej oprogramowanie wątków może być trudne.

# Motywacja



Wykonanie każdego procesu jest podzielone na fazy. Rozróżnia się dwa rodzaje faz: fazę procesora, w której procesowi jest przydzielony procesor i fazę wejścia-wyjścia, w której na rzecz procesu wykonywana jest operacja wejścia-wyjścia lub inną czynność nieangażującą procesora. Jeśli dwa lub większa liczba procesów byłaby wykonywana szeregowo, tak jak obrazuje to ilustracja obok (linia pogrubiona - faza procesora, linia kropkowana - faza wejścia-wyjścia, linia kreskowana - czas przed lub po wykonaniu procesu), to procesy musiałyby czekać długo na swoje wykonanie, a urządzenia wejścia-wyjścia i procesor byłyby naprzemiennie bezczynne. Takie procesy można jednak wykonywać współbieżnie, tzn. jeśli proces, który do tej pory korzystał z procesora wszedł w fazę korzystania z urządzeń perforejnych, to procesor może zostać przydzielony innemu procesowi, który jest gotów do wykonania. Takie postępowanie zmniejsza czas przetwarzania i czas oczekiwania procesu, jak również zwiększa i równoważy obciążenie procesora i urządzeń zewnętrznych, oraz podnosi przepustowość systemu (ilość pracy wykonaną w jednostce czasu).

# Motywacja

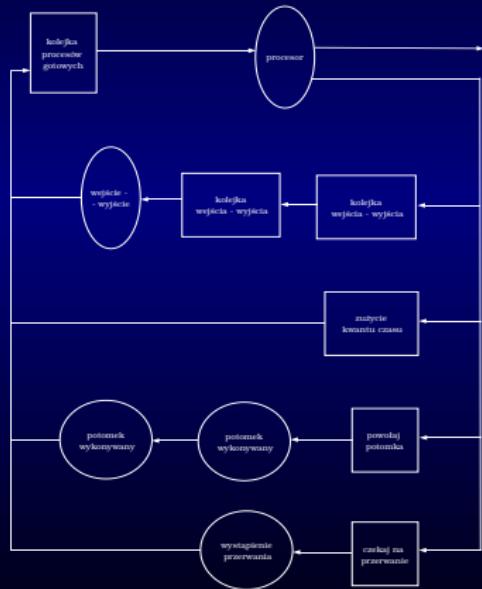


Wykonanie każdego procesu jest podzielone na fazy. Rozróżnia się dwa rodzaje faz: fazę procesora, w której procesowi jest przydzielony procesor i fazę wejścia-wyjścia, w której na rzecz procesu wykonywana jest operacja wejścia-wyjścia lub inną czynność nieangażującą procesora. Jeżeli dwa lub większa liczba procesów byłaby wykonywana szeregowo, tak jak obrazuje to ilustracja obok (linia pogrubiona - faza procesora, linia kropkowana - faza wejścia-wyjścia, linia kreskowana - czas przed lub po wykonaniu procesu), to procesy musiałyby czekać długo na swoje wykonanie, a urządzenia wejścia-wyjścia i procesor byłyby naprzemiennie bezczynne. Takie procesy można jednak wykonywać współbieżnie, tzn. jeśli proces, który do tej pory korzystał z procesora wszedł w fazę korzystania z urządzeń perforejnych, to procesor może zostać przydzielony innemu procesowi, który jest gotów do wykonania. Takie postępowanie zmniejsza *czas przetwarzania i czas oczekiwania procesu*, jak również zwiększa i równoważy obciążenie procesora i urządzeń zewnętrznych, oraz podnosi *przepustowość* systemu (ilość pracy wykonaną w jednostce czasu).

# Kolejki

Podstawową strukturą danych wykorzystywaną w zarządzaniu procesami są opisane wcześniej deskryptory procesów. Bloki kontrolne wszystkich procesów, które są gotowe do wykonania powiązane są w *kolejkę procesów gotowych*. Wyraz kolejka w tym kontekście nie oznacza konkretnej struktury danych, a miejsce w którym procesy oczekują na przydział procesora. Taka kolejka może być stosem, kolejką FIFO lub innym rodzajem listy, a nawet innym rodzajem struktur danych (tablicą, drzewem). System operacyjny utrzymuje również kolejki oczekiwania na realizację operacji wejścia-wyjścia. Te kolejki tworzą deskryptory procesów będących w stanie oczekiwania. Często upraszczając nie mówimy, że deskryptor procesu jest w kolejce lecz, że proces znajduje się w kolejce.

## Diagramy kolejek



Aby zwizualizować migrację procesów między kolejkami stosuje się często tzw. diagramy kolejek. Jeden z nich jest zaprezentowany obok.

## Planisci

Decyzję o tym w jakiej kolejności i które procesy zostaną umieszczone w kolejkach podejmują mechanizmy systemu operacyjnego nazywane *mechanizmami szeregującymi* lub *planistami* (ang. scheduler). Najważniejsze mechanizmy szeregujące odpowiedzialne są za przydział procesora procesom gotowym do wykonania. We współczesnych, głównie interaktywnych systemach operacyjnych istnieje tylko jeden rodzaj takich planistów. Jest to *planista krótkoterminowy*. Taki planista wybiera z kolejki procesów gotowych proces, który jako następny otrzyma procesor. Wywoływany jest on bardzo często i musi krótko działać, aby nie tworzyć zbyt dużych narzutów czasowych. Drugi rodzaj planistów, który był właściwy głównie dla systemów wsadowych, to *planisci długoterminowi*. Taki planista był wywoływany wtedy, kiedy kończył swe działanie jakiś proces. Jego zadaniem było wybrać zadanie z puli zadań do wykonania, które trafiało do kolejki zadań gotowych do wykonania. Planista długoterminowy dbał o to by w tej kolejce znajdowały się zarówno procesy ograniczone przez wejście-wyjście jak i procesy ograniczone przez procesor. Dzięki temu praca tych jednostek była równoważona. Dbał on również o zachowanie stopnia *wieloprogramowości* (stałej w czasie liczby procesów w pamięci). W niektórych systemach operacyjnych stosowano również *planistów średnioterminalowych*. Ich zadanie polegało na podejmowaniu decyzji, który z procesów ma być wycofany z pamięci operacyjnej do pamięci pomocniczej, aby umożliwić wykonanie innym procesom.

## Przełączanie kontekstu

Jeśli proces traci procesor na rzecz innego procesu, lub w wyniku innego zdarzenia (np.: obsługa przerwania, operacja wejścia-wyjścia), to należy zapamiętać informacje niezbędne do kontynuowania jego wykonania w przyszłości. Te informacje nazywane są *kontekstem procesu*. Kontekst procesu, który utracił procesor zastępowany jest kontekstem procesu, który procesor uzyskał. Tę operację nazywa się *przełączaniem kontekstu* i dąży się do tego, aby czas poświęcony na nią był jak najmniejszy. Kontekst procesu tracącego procesor jest zapamiętywany w jego deskryptorze.

# Koordynator

Przekazaniem sterowania do procesu, który wybrał planista krótkoterminowy zajmuje się koordynator (ang. dispatcher). Jego zadanie polega na przełączeniu kontekstu procesów, przełączeniu procesora w tryb użytkownika i wykonaniu skoku do adresu w programie użytkownika, pod którym znajduje się kolejny rozkaz do wykonania przez program.

# Wstęp

Sytuacja, która została przedstawiona we wstępie do opisu zagadnienia planowania procesów jest idealną i w rzeczywistości występuje rzadko (jeśli w ogóle). We wszystkich procesach występują naprzemiennie fazy procesora i wejścia-wyjścia, ale procesy ograniczone przez procesor charakteryzują się małą liczbą faz procesora, które są za to długie, z kolei procesy ograniczone przez wejście-wyjście mają dużo bardzo krótkich faz procesora. Bazując na tych spostrzeżeniach można optymalizować różne wielkości związane z efektywnością pracy systemu komputerowego, np.:

- ① wykorzystanie procesora,
- ② przepustowość (liczbę wykonanych procesów w jednostce czasu),
- ③ czas cyklu przetwarzania (czas od przedłożenia procesu do wykonania, do jego zakończenia),
- ④ czas oczekiwania (czas od przedłożenia procesu do rozpoczęcia jego wykonania),
- ⑤ czas odpowiedzi (czas reakcji procesu na żądania użytkownika - ważny w systemach interaktywnych),

Zostaną zaprezentowane podstawowe strategie używane przez planistę krótkoterminowego, z których większość będzie oceniona pod względem średniego czasu oczekiwania procesów.

# Wstęp

Sytuacja, która została przedstawiona we wstępie do opisu zagadnienia planowania procesów jest idealną i w rzeczywistości występuje rzadko (jeśli w ogóle). We wszystkich procesach występują naprzemiennie fazy procesora i wejścia-wyjścia, ale procesy ograniczone przez procesor charakteryzują się małą liczbą faz procesora, które są za to długie, z kolei procesy ograniczone przez wejście-wyjście mają dużo bardzo krótkich faz procesora. Bazując na tych spostrzeżeniach można optymalizować różne wielkości związane z efektywnością pracy systemu komputerowego, np.:

- ① wykorzystanie procesora,
- ② przepustowość (liczbę wykonanych procesów w jednostce czasu),
- ③ czas cyklu przetwarzania (czas od przedłożenia procesu do wykonania, do jego zakończenia),
- ④ czas oczekiwania (czas od przedłożenia procesu do rozpoczęcia jego wykonania),
- ⑤ czas odpowiedzi (czas reakcji procesu na żądania użytkownika - ważny w systemach interaktywnych),

Zostaną zaprezentowane podstawowe strategie używane przez planistę krótkoterminowego, z których większość będzie oceniona pod względem średniego czasu oczekiwania procesów.

## Wstęp

Sytuacja, która została przedstawiona we wstępie do opisu zagadnienia planowania procesów jest idealną i w rzeczywistości występuje rzadko (jeśli w ogóle). We wszystkich procesach występują naprzemiennie fazy procesora i wejścia-wyjścia, ale procesy ograniczone przez procesor charakteryzują się małą liczbą faz procesora, które są za to długie, z kolei procesy ograniczone przez wejście-wyjście mają dużo bardzo krótkich faz procesora. Bazując na tych spostrzeżeniach można optymalizować różne wielkości związane z efektywnością pracy systemu komputerowego, np.:

- ① wykorzystanie procesora,
- ② przepustowość (liczbę wykonanych procesów w jednostce czasu),
- ③ czas cyklu przetwarzania (czas od przedłożenia procesu do wykonania, do jego zakończenia),
- ④ czas oczekiwania (czas od przedłożenia procesu do rozpoczęcia jego wykonania),
- ⑤ czas odpowiedzi (czas reakcji procesu na żądania użytkownika - ważny w systemach interaktywnych),

Zostaną zaprezentowane podstawowe strategie używane przez planistę krótkoterminowego, z których większość będzie oceniona pod względem średniego czasu oczekiwania procesów.

## Wstęp

Sytuacja, która została przedstawiona we wstępie do opisu zagadnienia planowania procesów jest idealną i w rzeczywistości występuje rzadko (jeśli w ogóle). We wszystkich procesach występują naprzemiennie fazy procesora i wejścia-wyjścia, ale procesy ograniczone przez procesor charakteryzują się małą liczbą faz procesora, które są za to długie, z kolei procesy ograniczone przez wejście-wyjście mają dużo bardzo krótkich faz procesora. Bazując na tych spostrzeżeniach można optymalizować różne wielkości związane z efektywnością pracy systemu komputerowego, np.:

- ① wykorzystanie procesora,
- ② przepustowość (liczbę wykonanych procesów w jednostce czasu),
- ③ czas cyklu przetwarzania (czas od przedłożenia procesu do wykonania, do jego zakończenia),
- ④ czas oczekiwania (czas od przedłożenia procesu do rozpoczęcia jego wykonania),
- ⑤ czas odpowiedzi (czas reakcji procesu na żądania użytkownika - ważny w systemach interaktywnych),

Zostaną zaprezentowane podstawowe strategie używane przez planistę krótkoterminowego, z których większość będzie oceniona pod względem średniego czasu oczekiwania procesów.

## Wstęp

Sytuacja, która została przedstawiona we wstępie do opisu zagadnienia planowania procesów jest idealną i w rzeczywistości występuje rzadko (jeśli w ogóle). We wszystkich procesach występują naprzemiennie fazy procesora i wejścia-wyjścia, ale procesy ograniczone przez procesor charakteryzują się małą liczbą faz procesora, które są za to długie, z kolei procesy ograniczone przez wejście-wyjście mają dużo bardzo krótkich faz procesora. Bazując na tych spostrzeżeniach można optymalizować różne wielkości związane z efektywnością pracy systemu komputerowego, np.:

- ① wykorzystanie procesora,
- ② przepustowość (liczbę wykonanych procesów w jednostce czasu),
- ③ czas cyklu przetwarzania (czas od przedłożenia procesu do wykonania, do jego zakończenia),
- ④ czas oczekiwania (czas od przedłożenia procesu do rozpoczęcia jego wykonania),
- ⑤ czas odpowiedzi (czas reakcji procesu na żądania użytkownika - ważny w systemach interaktywnych),

Zostaną zaprezentowane podstawowe strategie używane przez planistę krótkoterminowego, z których większość będzie oceniona pod względem średniego czasu oczekiwania procesów.

# Wstęp

Sytuacja, która została przedstawiona we wstępie do opisu zagadnienia planowania procesów jest idealną i w rzeczywistości występuje rzadko (jeśli w ogóle). We wszystkich procesach występują naprzemiennie fazy procesora i wejścia-wyjścia, ale procesy ograniczone przez procesor charakteryzują się małą liczbą faz procesora, które są za to długie, z kolei procesy ograniczone przez wejście-wyjście mają dużo bardzo krótkich faz procesora. Bazując na tych spostrzeżeniach można optymalizować różne wielkości związane z efektywnością pracy systemu komputerowego, np.:

- ① wykorzystanie procesora,
- ② przepustowość (liczbę wykonanych procesów w jednostce czasu),
- ③ czas cyklu przetwarzania (czas od przedłożenia procesu do wykonania, do jego zakończenia),
- ④ czas oczekiwania (czas od przedłożenia procesu do rozpoczęcia jego wykonania),
- ⑤ czas odpowiedzi (czas reakcji procesu na żądania użytkownika - ważny w systemach interaktywnych),

Zostaną zaprezentowane podstawowe strategie używane przez planistę krótkoterminowego, z których większość będzie oceniona pod względem średniego czasu oczekiwania procesów.

Plan wykładu	Algorytm FCFS
Procesy	Algorytm SJF
Wątki	Algorytm SRT
Planowanie przydziału procesora	Algorytm priorytetowy
<b>Algorytmy szeregowania</b>	Algorytm rotacyjny
Planowanie w systemach wieloprocesorowych	Kolejki wielopoziomowe
Ocena algorytmów	Kolejki wielopoziomowe ze sprzężeniami zwrotnymi
	Przypadły

## Wywłaszczanie i jego brak

W zależności od tego, czy proces wyłącznie dobrowolnie oddaje procesor (np.: po zgłoszeniu zapotrzebowania na operację wejścia-wyjścia), czy też może mu on zostać odebrany (np.: w wyniku działania czasomierza), rozróżniamy dwa rodzaje strategii szeregowania procesów: szeregowanie *bez wywłaszczzeń* i szeregowanie z *wywłaszczaniem* (ang. preemptive). Systemy operacyjne stosujące pierwszą strategię nazywamy systemami z *kooperacją*, a systemy stosujące drugą po prostu systemami z *wywłaszczaniem*. Większość współczesnych systemów należy do drugiej kategorii.

# FCFS

Algorytm FCFS (ang. First-Come First-Served), nazywany także FIFO jest najprostszym algorytmem szeregowania. Procesor jest przyznawany procesom w takiej kolejności w jakiej są one umieszczone w kolejce procesów gotowych. Brak jest wywłaszczenia. FCFS może prowadzić do *efektu konwoju*, tzn. oczekiwania procesów ograniczonych przez *wejście-wyjście* na zakończenie realizacji długich faz procesora procesu ograniczonego przez procesor.

## Średni czas oczekiwania

Proces	Czas trwania fazy
P <sub>1</sub>	24
P <sub>2</sub>	3
P <sub>3</sub>	3



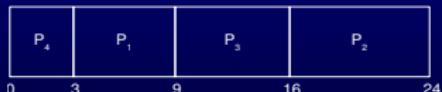
Średni czas oczekiwania można policzyć posługując się diagramem Gantta. Rysunek obok przedstawia taki diagram (w jego skład nie wchodzi linia wymiarowa). Obrazuje on kolejność wykonania procesów. Na jego dole znajdują się czasy oczekiwania kolejnych procesów na wykonanie. Są one sumą czasów wykonania ich poprzedników. Średni czas oczekiwania procesów jest w przypadku algorytmu FCFS średnią arytmetyczną czasów oczekiwania poszczególnych procesów w kolejce, czyli  $\frac{1}{n} \cdot \sum_{k=1}^n t_k = (0 + 24 + 27)/3 = 17ms$

## Algorytm SJF

**Algorytm SJF** (ang. Shortest Job First) - najpierw najkrótsze zadanie jest algorytmem optymalnym, jeśli chcemy uzyskać minimalny czas oczekiwania procesów. Procesor jest przydzielany procesom według długości trwania ich faz procesora (tzn. zaczynając od tego o najkrótszej fazie, a kończąc na tym o najdłuższej). Problemem jest jednak przewidywanie czasu trwania kolejnej fazy procesu. Stosuje się oszacowania statystyczne, bazujące na historii wykonania procesu. Te oszacowania opierają się na średniej wykładowniczej. Jeśli zapamiętywane są tylko dwie ostatnie fazy procesora procesu, to ta średnia ma postać:  $\tau_{n+1} = \alpha \cdot \tau_n + (1 - \alpha) \cdot \tau_{n-1}$ , gdzie  $\tau$  jest czasem wykonania fazy procesora, a  $\alpha$  współczynnikiem z przedziału  $[0,1]$ . Jeśli jest zapamiętywanych więcej faz, to wzór na średnią wykładowiczą staje się bardziej skomplikowany:  $\tau_{n+1} = \alpha \cdot \tau_n + (1 - \alpha) \cdot \alpha \cdot \tau_{n-1} + \dots + (1 - \alpha)^j \cdot \alpha \cdot \tau_{n-j} + \dots + (1 - \alpha)^{n+1} \cdot \tau_0$ . Za  $\tau_0$  przyjmuje się stałą lub wartość średnią dla wszystkich procesów w systemie. Opis odnosi się do algorytmu w wersji bez wywłaszczeń.

## Średni czas oczekiwania

<u>Proces</u>	<u>Czas trwania fazy</u>
P <sub>1</sub>	6
P <sub>2</sub>	8
P <sub>3</sub>	7
P <sub>4</sub>	3



Średni czas oczekiwania procesów dla algorytmu SJF bez wywłaszczeń liczony jest w ten sam sposób jak dla algorytmu FCFS, czyli dla danych z rysunku będzie to:  $(3 + 16 + 9 + 0)/4 = 7\text{ms}$ .

## Algorytm SRT

Algorytm SRT (ang. Shortest Remaining Time) - najpierw najkrótszy pozostał czas, jest odmianą algorytmu SJF z wywłaszczeniem. Procesor jest odbierany wykonywanemu procesowi wtedy, kiedy do kolejki procesów gotowych nadchodzi proces o czasie trwania fazy procesora krótszym, niż czas konieczny do zakończenia fazy procesora bieżącego zadania.

## Średni czas oczekiwania

Proces	Czas trwania fazy	Czas nadania
P <sub>1</sub>	8	0
P <sub>2</sub>	4	1
P <sub>3</sub>	9	2
P <sub>4</sub>	5	3



Licząc średni czas oczekiwania procesów dla algorytmu SRT należy uwzględnić czasy częściowego wykonania faz procesora wywieszanych procesów i odjąć je od czasów ich oczekiwania. Należy również uwzględnić i odjąć od czasu oczekiwania czas nadania do kolejki procesów gotowych, dla wszystkich procesów. Reasumując otrzymujemy:  $((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3)) / 4 = 26 / 4 = 6,5 \text{ ms}$ .

## Szeregowanie priorytetowe

**Algorytm priorytetowy** przydziela procesor procesom według przypisanego im *priorytetu*. Priorytet najczęściej jest liczbą naturalną, która określa „ważność” procesu. Zazwyczaj im niższa jest wartość tej liczby, tym proces ma wyższy priorytet. Priorytety mogą być przydzielane *zewnętrznie* jak i *wewnętrznie*. Szeregowanie priorytetowe może odbywać się z wywłaszczeniem, jak i bez. Algorytm SJF (w obu wersjach) jest formą algorytmu priorytetowego. Przy planowaniu priorytetowym może dojść do zjawiska, które nazywamy *głodzeniem procesu*. Zachodzi ono wtedy, gdy w systemie jest proces o bardzo niskim priorytecie. Jeśli inne procesy będą miały zawsze wyższe priorytety to rzeczony proces może nigdy nie otrzymać dostępu do procesora. Aby uniknąć tego zjawiska stosuje się okresowe *postarzanie procesów*, czyli zwiększenie ich priorytetów.

## Średni czas oczekiwania

Proces	Czas trwania fazy	Priorytet
P <sub>1</sub>	10	3
P <sub>2</sub>	1	1
P <sub>3</sub>	2	3
P <sub>4</sub>	1	4
P <sub>5</sub>	5	2



Średni czas oczekiwania będzie policzony dla algorytmu priorytetowego bez wyjątków. Dla danych z rysunku obok będzie to  $(1 + 6 + 16 + 18)/5 = 8,2\text{ms}$ .

## Szeregowanie rotacyjne

**Algorytm rotacyjny** (ang. round robin) zwany także karuzelowym występuje tylko w formie wywłaszczeniowej i jest właściwy dla systemów interaktywnych. Każdemu procesowi system operacyjny przyznaje pewien *kwant* czasu na wykonanie fazy procesora. Jeśli proces wykona swoją fazę wcześniej, to dobrowolnie oddaje procesor innym procesom. Jeśli nie to procesor jest mu odbierany, a on wędruje na koniec kolejki procesów gotowych i musi czekać aż wszystkie pozostałe procesy wykorzystają swój kwant czasu. Kolejka procesów gotowych jest więc listą cykliczną. Dobierając kwant czasu należy uważać, aby nie był zbyt długi (wówczas otrzymamy algorytm FCFS) lub zbyt krótki (wówczas procesor więcej czasu będzie poświęcał na przełączanie kontekstu, niż na wykonywanie procesów).

## Średni czas oczekiwania

Proces      Czas trwania fazy

P <sub>1</sub>	24
P <sub>2</sub>	3
P <sub>3</sub>	3

Q = 4 ms



Licząc średni czas oczekiwania procesów dla algorytmu rotacyjnego należy uwzględnić czasy częściowych wykonania faz procesora poszczególnych procesów (może ich być kilka) i odjąć je od czasów ich oczekiwania. Obliczenia przeprowadzamy w następujący sposób:  $((26 - 5 \cdot 4) + 4 + 7)/3 = 17/3 \approx 5,67\text{ms}$ .

## Kolejki wielopoziomowe

Działanie opisanych wcześniej algorytmów można połączyć dzieląc kolejkę procesów gotowych na kilka. Procesy umieszczane są w tych kolejkach w zależności do jakiej kategorii należą lub jaki mają priorytet. Szeregowanie w obrębie poszczególny kolejek odbywa się różnymi algorytmami lub też tym samym algorytmem, ale z różnymi parametrami dla tego algorytmu (np. algorytm rotacyjny z różnym kwantem czasu).

Plan wykładu	Algorytm FCFS
Procesy	Algorytm SJF
Wątki	Algorytm SRT
Planowanie przydziału procesora	Algorytm priorytetowy
<b>Algorytmy szeregowania</b>	Algorytm rotacyjny
Planowanie w systemach wieloprocesorowych	Kolejki wielopoziomowe
Ocena algorytmów	Kolejki wielopoziomowe ze sprzężeniami zwrotnymi
	Przeglądy

## Kolejki wielopoziomowe ze sprzężeniami zwrotnymi

Opisany wcześniej schemat można uzupełnić o migrację procesów między kolejkami. Otrzymujemy w ten sposób wielopoziomową kolejkę ze sprzężeniami zwrotnymi (ang. Multilevel Feedback Queue). Autorem tego rozwiązania jest Leonard Kleinrock. Przechodzenie procesów między kolejkami jest zależne od czasu trwania ich poprzedniej fazy procesora. Na tym schemacie opiera się szeregowanie procesów w oryginalnym systemie Unix.

Plan wykładu	Algorytm FCFS
Procesy	Algorytm SJF
Wątki	Algorytm SRT
Planowanie przydziału procesora	Algorytm priorytetowy
Algorytmy szeregowania	Algorytm rotacyjny
Planowanie w systemach wieloprocesorowych	Kolejki wielopoziomowe
Ocena algorytmów	Kolejki wielopoziomowe ze sprzężeniami zwrotnymi
	Przykłady

## Szeregowanie w Windows 2000/XP

W systemach rodziny Windows<sup>1</sup> szeregowaniu podlegają nie procesy lecz wątki. W Windows 2000 i XP zastosowano algorytm bazujący na wielopoziomowych kolejkach ze sprzężeniami zwrotnymi. Istnieją trzy kategorie priorytetów, najniższy oznaczany przez 0, jest priorytetem systemowym i jest przeznaczony dla wątku bezczynności. Kolejne priorytety, z zakresu 1-15 są priorytetami, które ulegają dynamicznym zmianom. Ostatnia kategoria to priorytety wątków (tolerancyjnego) czasu rzeczywistego, które są statycznie i odpowiadają im wartości z zakresu 16-31. W przypadku wątków o dynamicznym priorytecie, każdy z nich dziedziczy po procesie, który w systemach Windows jest traktowany wyłącznie jako kontener na wątki, *priorytet bazowy*. W zależności od historii wykonania wątku planista dodaje do tego priorytetu odpowiedni modyfikator. Modyfikatory mają zakres wartości (-7)-(-3) i (+3)-(+7). Na podstawie tak obliczonego priorytetu wątki są przypisywane do odpowiednich kolejek. Jeśli w wyniku dodawania wyjdzie wartość mniejsza od 1 lub większa niż 15, to wynik jest odpowiednio zaokrąglany. Szeregowanie w systemach Windows NT opiera się na podobnym schemacie.

<sup>1</sup>Materiał bazuje na informacjach ze strony <http://wazniak.mimuw.edu.pl/>

## Szeregowanie w systemach wieloprocesorowych

Jeśli w systemie rozproszonym każdy procesor jest innego typu, to planowanie jest stosunkowo proste - każdy procesor otrzymuje sobie właściwe zadania. Jeśli jednak procesory są jednakowe, to planowanie może przebiegać na kilka sposobów. Każdy z procesorów może mieć osobną kolejkę zadań, a system operacyjny powinien dbać o to, aby liczby elementów tych kolejek były takie same lub porównywalne. Procesory mogą też mieć wspólną kolejkę zadań. W takim rozwiążaniu wyznacza się najczęściej jeden z procesorów do wykonywania kodu planisty. Jest to forma wieloprzetwarzania asymetrycznego.

## Ocena algorytmów

Zanim algorytm szeregowania zostanie zaimplementowany w pracującym systemie operacyjny, to należy (a przynajmniej warto) sprawdzić jego skuteczność. Wstępnie można dokonać *oceny analitycznej* algorytmu zakładając pewne statyczne, uśrednione obciążenie procesora. To oszacowanie wykonywane jest za pomocą diagramów Gantta. Mamy wtedy do czynienia z *modelowaniem deterministycznym*. To oszacowanie można poprawić stosując metodę *modeli kolejkowych*. Głównym elementem tej metody jest wzór Little'a  $n = \lambda \cdot W$ , gdzie  $\lambda$  to tempo przybywania nowych procesów do systemu,  $W$  to średni czas oczekiwania w kolejce, a  $n$  to średnia długość kolejki. Prostszą, ale równie wiarygodną metodą oceny algorytmów szeregowania jest symulowanie ich działania w komputerze. Dane dla symulatora na temat procesów może dostarczać generator liczb pseudolosowych o rozkładzie wykładniczym, lub można je pobrać z rzeczywistego systemu. Ostateczną oceną jest jednak zawsze implementacja algorytmu w prawdziwym systemie operacyjnym. W takim przypadku programista systemowy może zmieniać parametry algorytmu lub sam algorytm w oparciu o opinie użytkowników. Może się również okazać, że to użytkownicy (programiści) przystosują swoje aplikacje do nowego szeregowania.

## Pytania

?

Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne - synchronizacja i komunikacja procesów

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 1 listopada 2020

# Plan wykładu

## ① Synchronizacja

- ② Sytuacje hazardowe
- ③ Problem sekcji krytycznej
- ④ Warunki poprawności rozwiązania
- ⑤ Rozwiązanie programowe dla dwóch procesów
- ⑥ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ⑦ Niepodzielne rozkazy
- ⑧ Semafora
- ⑨ Regiony krytyczne
- ⑩ Monitory

## ③ Klasyczne problemy synchronizacji

- ⑪ Problem ograniczonego buforowania
- ⑫ Problem czytelników i pisarzy
- ⑬ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ⑯ Komunikacja pośrednia i bezpośrednia
- ⑰ Buforowanie
- ⑱ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ⑥ Niepodzielne rozkazy
- ⑦ Semafora
- ⑧ Regiony krytyczne
- ⑨ Monitory

## ③ Klasyczne problemy synchronizacji

- ⑩ Problem ograniczonego buforowania
- ⑪ Problem czytelników i pisarzy
- ⑫ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ⑬ Komunikacja pośrednia i bezpośrednia
- ⑭ Buforowanie
- ⑮ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
  - ④ Rozwiązanie programowe dla dwóch procesów
  - ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafony
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem ucząjących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semaforы
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem ucząjących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem ucząjących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem ucząjących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem ucząjących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

# Plan wykładu

## ① Synchronizacja

- ① Sytuacje hazardowe
- ② Problem sekcji krytycznej
- ③ Warunki poprawności rozwiązania
- ④ Rozwiązanie programowe dla dwóch procesów
- ⑤ Rozwiązania programowe dla wielu procesów

## ② Środki synchronizacji

- ① Niepodzielne rozkazy
- ② Semafora
- ③ Regiony krytyczne
- ④ Monitory

## ③ Klasyczne problemy synchronizacji

- ① Problem ograniczonego buforowania
- ② Problem czytelników i pisarzy
- ③ Problem uczących filozofów

## ④ Sieci Petriego

## ⑤ Komunikacja międzyprocesowa

- ① Komunikacja pośrednia i bezpośrednia
- ② Buforowanie
- ③ Obsługa wyjątków

Plan wykładu	Sytuacje hazardowe
Synchronizacja	Problem sekcji krytycznej
Środki synchronizacji	Warunki poprawności rozwiązania
Klasyczne problemy synchronizacji	Rozwiązanie programowe dla dwóch procesów
Sieci Petriego	Rozwiązania programowe dla wielu procesów
Komunikacja międzyprocesowa	

## Wprowadzenie

W systemach wielozadaniowych, podczas realizacji przez ustaloną liczbę procesów dostępu współbieżnego do dzielonych zasobów może dojść do *sytuacji hazardowych*, nazywanych również *wyścigiem* (ang. *race condition*), które prowadzą do błędów przetwarzania. Zjawiska te pojawiają się, kiedy następuje *przeplot operacji* modyfikacji lub operacji modyfikacji i odczytu stanu współdzielonego zasobu, przy czym operacje te pochodzą od różnych procesów. Jeśli dostęp do zasobu ogranicza się wyłącznie do odczytu, to jest dostępem zawsze bezpiecznym, nawet jeśli dochodzi od przeplotu operacji. Sytuacje hazardowe występują zarówno w przypadku procesów, jak i wątków (choć w dalszej części wykładu będziemy posługiwać się głównie pojęciem procesu). Obojętnym jest również, czy system komputerowy, w którym są wykonywane procesy jest wyposażony w jeden, czy większą liczbę procesorów.

## Przykład

Rozważmy dwa procesy, które chcą zmodyfikować wartość współdzielonej zmiennej, przy czym pierwszy chce tę wartość zwiększyć o jeden, a drugi zmniejszyć o jeden. Przymijmy, że wartość początkowa zmiennej wynosi 5. Zakładamy, że pojedynczy proces, aby zmodyfikować wartość zmiennej musi ją najpierw pobrać z pamięci operacyjnej, zapisać w rejestrze<sup>1</sup>, wykonać właściwą operację, a następnie zapisać wynikową wartość do pamięci. Procesy wykonujące wspomniane operacje mogą utracić procesor w każdej chwili, w wyniku zadziałania mechanizmu wywłaszczającego. Następne plansze prezentują dwa z możliwych scenariuszy wykonania wcześniej opisanych modyfikacji wartości wspólnej zmiennej (kropki oznaczają, że proces jest w danej chwili nieczynny).

---

<sup>1</sup>Proszę zauważyć, że ze względu na to iż w wyniku przełączenia kontekstu stany rejestrów są zapamiętywane, to procesy mogą posługiwać się tym samym rejestrem

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz pierwszy

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② Zwiększ wartość rejestru R1 o jeden. ( $R1=6$ )
- ③ ...
- ④ ...
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )
- ⑥ ...

### Proces drugi

- ① ...
- ② ...
- ③ Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )

### Wynik

Wartość wynikowa wynosi 4 (jako ostatni swój wynik do pamięci zapisał proces drugi). Tymczasem prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Scenariusz drugi

### Proces pierwszy

- ① Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ② ...
- ③ Zwiększ zawartość rejestru R1 o jeden. ( $R1=6$ )
- ④ ...
- ⑤ ...
- ⑥ Zapisz zawartość rejestru R1 do pamięci. ( $M=6$ )

### Proces drugi

- ① ...
- ② Odczytaj wartość z pamięci ( $M=5$ ) i umieść ją w rejestrze R1. ( $R1=5$ )
- ③ ...
- ④ Zmniejsz zawartość rejestru R1 o jeden. ( $R1=4$ )
- ⑤ Zapisz zawartość rejestru R1 do pamięci. ( $M=4$ )
- ⑥ ...

### Wynik

Wartość wynikowa wynosi tym razem 6 (jako ostatni swój wynik do pamięci zapisał proces pierwszy). Tak jak poprzednio prawidłowym wynikiem jest 5.

## Sekcja krytyczna

Fragment kodu, podczas którego realizacji proces wykonuje dostęp do zasobów wspólnie dzielonych nazywamy *sekcją krytyczną*. Zasoby te mogą być zarówno zasobami fizycznymi, jak i logicznymi, mogą mieć prostą budowę (jak zmienne prostych typów) lub złożoną (jak struktury danych). Jak wynika ze wcześniejszych rozważań, aby dostęp do zasobu współdzielonego był bezpieczny musimy zagwarantować niepodzielność wykonania przez procesy sekcji krytycznych. Inaczej: jeśli jeden z procesów korzystających z zasobu dzielnego rozpoczął wykonywanie sekcji krytycznej, to żaden z pozostałych procesów nie może rozpoczęć wykonywania sekcji krytycznej dotyczącej tego samego zasobu, dopóki ten pierwszy jej nie skończy. Rozpoczynanie sekcji krytycznej określamy mianem *wchodzenia do sekcji krytycznej* zaś jej kończenie *wychodzeniem* lub *opuszczaniem sekcji krytycznej*. Część kodu bezpośrednio poprzedzającą sekcję krytyczną nazywamy *sekcją wejściową*, natomiast część umieszczoną bezpośrednio za sekcją krytyczną określamy mianem *sekcji wyjściowej*. Pozostałą część kodu procesu będziemy nazywać *resztą*.

## Warunki poprawności rozwiązania problemu sekcji krytycznej

Każde rozwiązanie problemu sekcji krytycznej musi spełniać trzy warunki, aby być w pełni poprawnym:

- **Wzajemne wykluczanie (ang. *mutual exclusion*)** W danym czasie, w sekcji krytycznej może znajdować się tylko jeden proces.
- **Postęp** Jeśli nie wymaga tego warunek wzajemnego wykluczania, to proces nie powinien być wstrzymywany przed wejściem do sekcji krytycznej.
- **Ograniczone czekanie** Oczekiwanie każdego procesu na wejście do sekcji krytycznej powinno kiedyś się zakończyć. Inne procesy nie mogą wstrzymywać go w nieskończoność przed wejściem do sekcji krytycznej.

## Poprawne rozwiązanie programowe dla dwóch procesów

Pierwszym, który podał prawidłowe rozwiązanie problemu sekcji krytycznej dla **dwóch procesów** był holenderski matematyk T.J.Dekker. Zaprezentowane na następnej planszy rozwiązanie, w postaci fragmentu kodu w języku (pseudo)C jest modyfikacją algorytmu Dekkera i nosi nazwę algorytmu Petersona od nazwiska jego autora Gary'ego Petersona. Listing zawiera (oprócz zewnętrznej, nieskończonej pętli do... while) jedynie kod sekcji wejściowej i wyjściowej (w tym wypadku składa się ona z tylko jednego wiersza). Kod sekcji krytycznej i reszty procesu został zastąpiony ciągami znaków *Sekcja Krytyczna* i *Reszta*.

## Kod rozwiązania

### Współdzielone zmienne wymagane przez algorytm

```
bool flaga[2]; //tablica flag gotowości procesów do wejścia do s.k  
unsigned int numer; //numer procesu (0 lub 1), któremu zezwolono wejść do s.k.
```

#### Proces $P_0$

```
do {  
    flaga[0]=true;  
    numer=1;  
    while(flaga[1] && numer==1)  
        ;  
    Sekcja Krytyczna  
    flaga[0]=false;  
    Reszta  
} while(1);
```

#### Proces $P_1$

```
do {  
    flaga[1]=true;  
    numer=0;  
    while(flaga[0] && numer==0)  
        ;  
    Sekcja Krytyczna  
    flaga[1]=false;  
    Reszta  
} while(1);
```

## Dowód poprawności

### Wzajemne wykluczanie

Jeden z dwóch procesów wchodzi do sekcji krytycznej wtedy i tylko wtedy, kiedy flaga procesu przeciwnego ma wartość *false* lub gdy zmienna *numer* zawiera jego identyfikator. Może zaistnieć sytuacja, w której oba procesy będą miały ustawione flagi, ale zmienna *numer* może przyjąć tylko jedną wartość, a więc jeden z nich będzie wykonywał pętlę **while**, a drugi wejdzie do sekcji krytycznej.

## Dowód poprawności

### Wzajemne wykluczanie

Jeden z dwóch procesów wchodzi do sekcji krytycznej wtedy i tylko wtedy, kiedy flaga procesu przeciwnego ma wartość *false* lub gdy zmienna *numer* zawiera jego identyfikator. Może zaistnieć sytuacja, w której oba procesy będą miały ustawione flagi, ale zmienna *numer* może przyjąć tylko jedną wartość, a więc jeden z nich będzie wykonywał pętlę **while**, a drugi wejdzie do sekcji krytycznej.

### Postęp

Założymy, że proces o numerze 0 chce wejść do sekcji krytycznej, a proces o numerze 1 nie jest nią zainteresowany (bo wykonuje swoją *resztę*). Zmienna *numer* będzie miała wartość 1, ale flaga procesu o numerze 1 nie będzie ustawiona, a więc proces 0 nie zostanie powstrzymany przed wejściem do sekcji krytycznej.

## Dowód poprawności

### Wzajemne wykluczanie

Jeden z dwóch procesów wchodzi do sekcji krytycznej wtedy i tylko wtedy, kiedy flaga procesu przeciwnego ma wartość *false* lub gdy zmienna *numer* zawiera jego identyfikator. Może zaistnieć sytuacja, w której oba procesy będą miały ustawione flagi, ale zmienna *numer* może przyjąć tylko jedną wartość, a więc jeden z nich będzie wykonywał pętlę **while**, a drugi wejdzie do sekcji krytycznej.

### Postęp

Założymy, że proces o numerze 0 chce wejść do sekcji krytycznej, a proces o numerze 1 nie jest nią zainteresowany (bo wykonuje swoją *resztę*). Zmienna *numer* będzie miała wartość 1, ale flaga procesu o numerze 1 nie będzie ustawiona, a więc proces 0 nie zostanie powstrzymany przed wejściem do sekcji krytycznej.

### Ograniczone oczekiwanie

Instrukcja przypisania z sekcji wyjściowej gwarantuje, że proces będzie czekał na wejście do sekcji krytycznej tylko do momentu, gdy drugi z procesów zakończy swoją sekcję krytyczną i wykona sekcję wyjściową.

## Poprawne rozwiązanie programowe dla wielu procesów

Uogólnienie przedstawionego wcześniej rozwiązania na  $n$  procesów, nie jest proste, a otrzymany kod jest mniej czytelny niż w przypadku dwóch procesów. Zmianie ulega typ zmiennej współdzielonej, która jest tablicą flag. Każda flaga może przyjmować teraz trzy wartości: *puste* - proces jest poza sekcją krytyczną, *gotowy* - proces zgłasza swoją gotowość do wejścia do sekcji krytycznej, *w\_sekcji* proces jest w sekcji krytycznej. Zmienna *numer* również przyjmuje większy zakres wartości (od 0 do  $n-1$ ) niż poprzednio.

**Uwaga:** Zmienna *j* jest zmienną lokalną procesu, tzn. nie jest współdzielona z innymi procesami (poza właścicielem tej zmiennej, żaden inny proces nie ma do niej dostępu).

## Kod rozwiązania dla i-tego procesu

### Współdzielone zmienne wymagane przez algorytm

```
enum stan { puste, gotowy, w_sekcji};  
enum stan flaga[n];  
unsigned int numer; // przyjmuje wartości od 0 do n-1
```

### Proces $P_i$

```
unsigned int j; // przyjmuje wartości od 0 do n  
do {  
    do {  
        flaga[i]=gotowy;  
        j=numer;  
        while(i!=j)  
            if(flagaj!=puste) j=numero;  
            else j=(j+1) % n;  
        flaga[i]=w_sekcji;  
        j= 0;  
        while(j<n&&(j==i || flaga[j]!=w_sekcji)) j++;  
    } while(j<n&&(numero!=i || flaga[numero]!=puste));  
    numero=i;  
    Sekcja Krytyczna  
    j=(numero+1) % n;  
    while(flagaj==puste) j=(j+1) % n;  
    numero=j;  
    flaga[i]=puste;  
    Reszta  
} while(1);
```

## Dowód poprawności

### Wzajemne wykluczanie

Proces z grupy procesów ubiegających się o dostęp do wspólnego zasobu wchodzi do sekcji krytycznej wtedy i tylko wtedy, gdy jego flaga gotowości ma wartość `w_sekcji`, a flagi pozostałych procesów mają inną wartość. Ponieważ tylko on może ustawić swoją flagę na wspomnianą wartość oraz dokonuje sprawdzenia flag pozostałych procesów po jej ustawieniu, to warunek wzajemnego wykluczania jest zachowany.

## Dowód poprawności

### Wzajemne wykluczanie

Proces z grupy procesów ubiegających się o dostęp do współdzielonego zasobu wchodzi do sekcji krytycznej wtedy i tylko wtedy, gdy jego flaga gotowości ma wartość `w_sekcji`, a flagi pozostałych procesów mają inną wartość. Ponieważ tylko on może ustawić swoją flagę na wspomnianą wartość oraz dokonuje sprawdzenia flag pozostałych procesów po jej ustawieniu, to warunek wzajemnego wykluczania jest zachowany.

### Postęp

Wartość zmiennej `numer` ulega zmianie tylko wtedy gdy proces wchodzi lub wychodzi z sekcji krytycznej. Jeśli tylko jeden proces jest zainteresowany wejściem do sekcji krytycznej, a żaden inny nie wykonuje jej, ani nie ubiega się o wejście do niej, to może on wykonać sekcję krytyczną poza kolejnością wyznaczaną przez zmienną `numer`.

## Dowód poprawności

### Wzajemne wykluczanie

Proces z grupy procesów ubiegających się o dostęp do wspólnego zasobu wchodzi do sekcji krytycznej wtedy i tylko wtedy, gdy jego flaga gotowości ma wartość `w_sekcji`, a flagi pozostałych procesów mają inną wartość. Ponieważ tylko on może ustawić swoją flagę na wspomnianą wartość oraz dokonuje sprawdzenia flag pozostałych procesów po jej ustawieniu, to warunek wzajemnego wykluczania jest zachowany.

### Postęp

Wartość zmiennej `numer` ulega zmianie tylko wtedy gdy proces wchodzi lub wychodzi z sekcji krytycznej. Jeśli tylko jeden proces jest zainteresowany wejściem do sekcji krytycznej, a żaden inny nie wykonuje jej, ani nie ubiega się o wejście do niej, to może on wykonać sekcję krytyczną poza kolejnością wyznaczaną przez zmienną `numer`.

### Ograniczone oczekiwanie

Każdy proces opuszczający sekcję krytyczną w sekcji wyjściowej wyznacza swojego następcę do wejścia do sekcji krytycznej. W ten sposób każdy proces, który ubiega się o wykonanie sekcji krytycznej dostanie pozwolenie po co najwyżej  $n-1$  próbach.

## Algorytm piekarni

Innym rozwiązaniem problemu sekcji krytycznej dla  $n$  procesów jest algorytm piekarni, który został opracowany przez Lesliego Lamporta. Nazwa tego algorytmu wzięła się od sposobu w jaki piekarnie w Stanach Zjednoczonych sprzedają chleb. W polskich realiach odpowiada to sposobowi przyjmowania pacjentów przez lekarzy w przychodniach. Każdy pacjent musi się zarejestrować i otrzymać swój numer. Im niższa jest wartość tego numeru, tym szybciej jest jego właściciel obsługiwany. Podobne rozwiązanie można zastosować dla procesów ubiegających się o wejście do sekcji krytycznej. Jednakże w tym przypadku może zdarzyć się, że dwa procesy otrzymają ten sam numer. Taki konflikt rozstrzyga się porównując ich identyfikatory (PID), które też są numerami. Zanim zostanie przedstawiony kod rozwiązania musimy zdefiniować operację porównywania par liczb, oraz operację wybierania liczby maksymalnej ze zbioru liczb:

$$(a, b) < (c, d) \iff a < c \cup (a = c \cap b < d)$$

$\max(a_0, a_1, \dots, a_{n-1})$  jest taką liczbą  $k$ , że  $k \geq a_i$  dla  $i = 0, \dots, n - 1$

## Pseudokod rozwiązania dla procesu $P_i$

Współdzielone zmienne wymagane przez algorytm

```
bool wybrane[n];  
int numer[n];
```

Proces  $P_i$ :

```
do {  
    wybrane[i]=true;  
    numer[i]=max(numer[0],numer[1],...,numer[n-1])+1;  
    wybrane[i]=false;  
    for(j=0;j<n;j++)  
    {  
        while(wybrane[j])  
        ;  
        while(numer[j]!=0 && (numer[j],j)<(numer[i],i))  
        ;  
    }  
    Sekcja Krytyczna  
    numer[i]=0;  
    Reszta  
} while(1);
```

## Dowód poprawności

### Wzajemne wykluczanie

Zauważmy, że każdy proces, który wchodzi do sekcji krytycznej otrzymuje numer, który jest następcą największego z dotychczas wybranych numerów. Ponieważ operacja wybierania nie jest niepodzielna, to dodatkowo sprawdzane są unikatowe numery identyfikacyjne procesów, gdyby pojawiły się dwa lub większa liczba procesów o takich samych wybranych numerach. Operacja porównania numerów wstrzymywana jest do czasu zakończenia wybierania numeru przez nadchodzące procesy. To wszystko gwarantuje spełnienie warunku wzajemnego wykluczania.

## Dowód poprawności

### Wzajemne wykluczanie

Zauważmy, że każdy proces, który wchodzi do sekcji krytycznej otrzymuje numer, który jest następcą największego z dotychczas wybranych numerów. Ponieważ operacja wybierania nie jest niepodzielna, to dodatkowo sprawdzane są unikatowe numery identyfikacyjne procesów, gdyby pojawiły się dwa lub większa liczba procesów o takich samych wybranych numerach. Operacja porównania numerów wstrzymywana jest do czasu zakończenia wybierania numeru przez nadchodzące procesy. To wszystko gwarantuje spełnienie warunku wzajemnego wykluczania.

### Postęp

Ponieważ tylko procesy gotowe do wejścia do sekcji krytycznej wybierają numery, to zapewniony jest warunek postępu.

## Dowód poprawności

### Wzajemne wykluczanie

Zauważmy, że każdy proces, który wchodzi do sekcji krytycznej otrzymuje numer, który jest następcą najwiekszego z dotychczas wybranych numerów. Ponieważ operacja wybierania nie jest niepodzielna, to dodatkowo sprawdzane są unikatowe numery identyfikacyjne procesów, gdyby pojawiły się dwa lub większa liczba procesów o takich samych wybranych numerach. Operacja porównania numerów wstrzymywana jest do czasu zakończenia wybierania numeru przez nadchodzące procesy. To wszystko gwarantuje spełnienie warunku wzajemnego wykluczania.

### Postęp

Ponieważ tylko procesy gotowe do wejścia do sekcji krytycznej wybierają numery, to zapewniony jest warunek postępu.

### Ograniczone oczekiwanie

Procesy wchodzą do sekcji krytycznej w takim porządku w jakim nadeszły, a więc spełnienie warunku ograniczonego czekania jest zapewnione.

## Podsumowanie

Wszystkie przedstawione tu rozwiązania programowe są z teoretycznego punktu widzenia poprawne. Niestety, w praktyce te rozwiązania mogą zawieść, jeśli program będzie wykonywany na procesorze stosującym wykonywanie instrukcji poza kolejnością (ang. *out of order execution*). Poprawność tych rozwiązań może również być naruszona podczas wykonywania przez kompilator optymalizacji kodu wynikowego. Stosując te rozwiązania należy pamiętać o dodatkowych środkach, które pozwalają uniknąć opisanych problemów. Innymi wadami przedstawionych algorytmów są problemy z zastosowaniem ich do bardziej skomplikowanych zadań oraz to, że wymagają one aktywnego oczekiwania. Ta ostatnia wada wiąże się z problemem nazywanym *inwersją priorytetów*. Występuje on w systemach stosujących szeregowanie priorytetowe. Założymy, że w takim systemie pozwolenie na wejście do sekcji krytycznej uzyskuje proces niskopriorytetowy. Podczas jej wykonywania zostaje on wywieszczony przez proces wysokopriorytetowy, któryaczyna ubiegać się od dostępu do tej samej sekcji. Ponieważ jest ona zajęta, to ten proces przechodzi w stan aktywnego oczekiwania, a ze względu na jego priorytet procesor nie zostanie przydzielony procesowi niskopriorytetowemu, który mógłby zakończyć to oczekiwanie wychodząc z sekcji.

## Wprowadzenie

Najprostszym sposobem zapewnienia wyłączności i niepodzielności wykonania sekcji krytycznej jest wyłącznie na czas, kiedy przebywa w niej proces, systemu przerwań. Niestety, to rozwiązanie może prowadzić do większych problemów, niż sytuacje hazardowe (Co jeśli ktoś z instrukcji z sekcji krytycznej wygeneruje wyjątek?). Nie daje się ono również zastosować w systemach wieloprocesorowych. Zamiast tak radykalnego rozwiązania twórcy sprzętu, systemów operacyjnych i translatorów oferują szereg środków, które wspierają programistów w rozwiązywaniu problemu sekcji krytycznej. Dalej zostaną omówione najpopularniejsze z nich.

## Niepodzielne rozkazy sprzętowe

Większość współczesnych architektur sprzętowych oferuje rozkazy, które pozwalają wykonać w sposób niepodzielny, na prostych zmiennych takie operacje, jak dodawanie, odejmowanie, operacje binarne. Istnieją również rozkazy pomagające rozwiązać problem sekcji krytycznej dla operacji na strukturach danych. Są nimi rozkazy typu „testuj i ustaw” oraz „wymień”. Pierwszy z nich pozwala w sposób niepodzielny odczytać wartość zmiennej, a następnie ją zmodyfikować. Drugi z nich dokonuje zamiany wartości dwóch zmiennych, które pełnią rolę zamka i klucza. Oba rozkazy pozwalają zapewnić spełnienie warunku wzajemnego wykluczania i mogą posłużyć do implementowania opisanych wcześniej algorytmów rozwiązywania problemu sekcji krytycznej. Ponieważ blokują one magistralę pamięci, a w związku z tym także dostęp do określonej lokacji RAM innym rozkazom, to są one szczególnie przydatne w systemach wieloprocesorowych.

## Semafor

Semafor jest zmienną całkowitą, do której dostęp można uzyskać (poza inicjalizacją) jedynie za pomocą dwóch specjalnych operacji: *czekaj* i *sygnalizuj*. Pierwsza sprawdza, czy semafor ma wartość większą od zera. Jeśli tak, to zmniejsza ją o jeden, jeśli nie, to czeka aż tę wartość osiągnie i dopiero wtedy ją zmniejsza. Druga polega na zwiększeniu wartości semafora o jeden. Obie operacje są w całości wykonywane niepodzielnie. Takie rozwiązanie pierwszy zaproponował Edsger Dijkstra, dlatego te operacje są oznaczane odpowiednio symbolami P (od proben) i V (od verhogen). Stosuje się również angielskie określenia *up* i *down*. Istnieją różne wersje semaforów, niektóre z nich mogą przyjmować różne wartości, inne tylko dwie. Te ostatnie nazywane są semaforami binarnymi lub *muteksami*. Istnieją również dwa sposoby implementacji operacji czekaj. Pierwszy z nich wymaga aktywnego oczekiwania na podniesienie semafora i został już opisany wyżej. Semafony posiadające tak zaimplementowaną operację oczekiwania nazywa się „wirującymi blokadami” (ang. *spin-lock*) i są one stosowane w systemach wieloprocesorowych, wtedy, kiedy istnieje pewność, że proces będzie krótko czekał na podniesienie semafora. Drugi sposób polega na umieszczeniu wszystkich procesów czekających na podniesienie semafora w kolejce oczekiwania na to zdarzenie. Kiedy semafor zostanie podniesiony uaktywniany jest jeden z tych procesów i on może kontynuować swoje działanie. O procesie, który oczekuje na podniesienie semafora w kolejce mówimy, że został uśpiony, a proces, który został wybrany z tej kolejki oczekiwania mówimy, że został obudzony.

# Realizacja P i V

## Aktywne oczekiwanie

```
czekaj(S): while(S≤0);
           S--;
```

```
sygnalizuj(S): S++;
```

## Usypianie

```
czekaj(S): S--;
           if(S<0)
               Uśpij(proces);
```

```
sygnalizuj(S): S++;
           if(S≤0)
               Obudź(proces);
```

## Mutex

```
czekaj(S): if(S==1)
           S=0;
           else
               Uśpij(proces);
sygnalizuj(S): if(!Czeka(proces))
               S=1;
               else
                   Obudź(proces);
```

## Regiony

Choć semafory są skuteczne w rozwiązywaniu sekcji krytycznej i proste w zastosowaniu, to mogą być dosyć niewygodne w użyciu, tym samym prowadząc do powstania błędów logicznych w programach. Programista może np.: zapomnieć o umieszczeniu w programie instrukcji podnoszącej semafor. W takim wypadku może dojść do zablokowania działania wszystkich procesów czekających na podniesienie semafora. Aby zapobiec takim sytuacjom w językach programowania (głównie proceduralnych) wprowadzono instrukcje pozwalające tworzyć tzw. regiony krytyczne. Słowo kluczowe **shared** pozwala na określenie zmiennej, jako współdzielonej przez kilka procesów, natomiast instrukcja **region zmienna\_współdzielona do operacja;** gwarantuje, że operacja wykonana na zmiennej współdzielonej będzie niepodzielna. Kompilator do realizacji tej niepodzielności może użyć niejawnie semaforów. Do rozwiązywania zaawansowanych problemów synchronizacji przydaje się konstrukcja regionu warunkowego: **region zmienna\_współdzielona when warunek do operacja;**. Operacja zostanie wykonana, tylko wtedy, kiedy warunek będzie spełniony. Praktyczne problemy synchronizacji wymagają dosyć często, aby warunek był sprawdzany nie przed wejściem do sekcji krytycznej, ale w trakcie jej trwania, np.: przed wykonaniem jednej z kilku operacji. Do realizacji oczekiwania na spełnienie warunku służy instrukcja **await(warunek)**. Regiony krytyczne można zagnieżdżać, ale należy robić to ostrożnie, gdyż może to prowadzić do problemów zwanych **zakleszczeniami**.

## Słowo kluczowe volatile

W językach C, C++ i Java, w deklaracjach zmiennych można użyć słowa kluczowego *volatile*, które jest informacją dla kompilatora, że nie powinien stosować optymalizacji w dostępie do tej zmiennej, polegającej na odczycie kopii jej wartości z rejestru procesora. Tym samym wszelkie operacje dokonywane na tej zmiennej, są dokonywane bezpośrednio na jej oryginalnej, aktualnej wartości umieszczonej w pamięci operacyjnej. Mimo, że w pewnych warunkach i językach programowania słowo kluczowe *volatile* może dać efekt niepodzielnego dostępu do zmiennej, to nie należy go traktować jako środka synchronizacji. Nieporozumienia co do sposobu działania tego słowa często prowadzą do powstania programów, które są nieodporne na sytuacje hazardowe.

# Monitory

## Przykład w języku Java

```
class Monitor {  
    private int field;  
    public void synchronized setField(int f) {  
        field = f;  
    }  
    public int synchronized getField() {  
        return field;  
    }  
}
```

Monitory są środkiem synchronizacji właściwym dla języków obiektowych, choć zostały opracowane niezależnie od techniki obiektowej. Konstrukcja ta jest autorstwa Per Brinch Hanse na i C.A.R Hoare'a. Monitor można określić jako obiekt, którego wszystkie pola są prywatne, a ich wartości osiągalne tylko za pomocą metod obiektu, wykonywanych w sposób niepodzielny. Z obiektem monitora może być, podobnie jak z regionem, związany warunek. Monitory znalazły zastosowanie w języku Java, niestety ich realizacja nie jest do końca zgodna z intencjami twórców tego środka synchronizacji. Warunek w Javie jest związany z monitorem niejawnie i osiągalny za pomocą metod `wait()`, `notify()`, `notifyAll()`.

## Klasyczne problemy synchronizacji

Opisane w dalszej części wykładu problemy stanowią ważne zagadnienia dotyczące współbieżności, bowiem większość rzeczywistych problemów synchronizacji daje się sprowadzić do któregoś z nich. Dzięki temu można zastosować do nich znane i sprawdzone rozwiązania. Problemy te stanowią również dobry zestaw testowy (ang. *test suit*) dla każdego nowego rozwiązania problemu synchronizacji.

## Problem ograniczonego buforowania

Dane są dwa procesy. Jeden z nich jest *producentem*, który produkuje kolejne jednostki informacji, drugi z nich jest *konsumentem*, który zużywa te informacje. Żeby żadna z informacji wyprodukowanych przez producenta nie została stracona, są one buforowane. Jeśli konsument działa z taką samą lub większą szybkością niż producent, to zawsze będą wolne bufory i co najwyżej konsument będzie czekał na nowe jednostki informacji. Mamy wtedy do czynienia z problemem *nieograniczonego buforowania*. Jeśli jednak ten warunek nie jest spełniony, to może zabraknąć pustych buforów i producent będzie musiał zaczekać aż konsument opróżni któryś z zajętych buforów. Ten problem jest nazywany problemem *ograniczonego buforowania*. Na następnych planszach znajdują się dwa rozwiązania tego problemu. Pierwsze z nich pozwala zapełnić co najwyżej  $n-1$  buforów, drugie wymaga z kolei niepodzielności operacji modyfikacji wartości zmiennej *licznik*.

## Pierwsze rozwiązanie

### Współdzielone elementy

```
const int n = ...;  
typedef ... jednostka;  
jednostka bufor[n];  
unsigned int we,wy; // przyjmują wartości od 0 do n-1
```

### Producent

```
do {  
    ...  
    Produkuj jednostkę w nastp  
    ...  
    while((we + 1) % n == wy)  
        ;  
        bufor[we]=nastp;  
        we=(we + 1) % n;  
} while(1);
```

### Konsument

```
do {  
    while(we == wy)  
        ;  
        nastk= bufor[wy];  
        wy=(wy + 1) % n;  
        ...  
        Konsumuj jednostkę z nastk  
        ...  
} while(1);
```

## Drugie rozwiązanie

### Współdzielone elementy

Te same co poprzednio oraz dodana jest zmienna całkowita *licznik*, a usunięte zostały zmienne *we* i *wy* (są teraz lokalne).

#### Producent

```
do {  
    ...  
    Produkuj jednostkę w nastp  
    ...  
    while(licznik == n)  
        ;  
        bufor[we]=nastp;  
        we=(we + 1) % n;  
        licznik++;  
    } while(1);
```

#### Konsument

```
do {  
    while(licznik == 0)  
        ;  
        nastk= bufor[wy];  
        wy=(wy + 1) % n;  
        licznik--;  
    ...  
    Konsumuj jednostkę z nastk  
    ...  
}while(1);
```

## Problem czytelników i pisarzy

W problemie czytelników i pisarzy istnieją dwie grupy procesów, które mają dostęp do współdzielonego zasobu. Do pierwszej należą procesy, które wyłącznie odczytują stan współdzielonego zasobu i dlatego nazywamy je *czytelnikami*, do drugiej należą procesy, które mogą stan zasobu odczytywać lub zapisywać. Nazywamy je *pisarzami*. Jednoczesny dostęp kilku czytelników do zasobu jest możliwy, gdyż nie powoduje żadnych komplikacji, natomiast dostęp pisarzy musi odbywać się na zasadzie wyłączności i niepodzielności. Istnieje kilka wersji problemu czytelników i pisarzy, tu zostaną opisane tylko dwie. W *pierwszym* problemie czytelników i pisarzy, żaden czytelnik nie powinien czekać na dostęp do zasobu. Innymi słowy praca pisarzy może być wstrzymywana przez czytelników. W *drugim* problemie czytelników i pisarzy, jeśli któryś z pisarzy jest gotów do modyfikacji zasobu, to żaden czytelnik nie powinien uzyskać do niego dostępu. Innymi słowy, praca czytelników jest wstrzymywana przez pisarzy. Rozwiązania obu przypadków można uzyskać stosując dwa semafory, jeden zapewniający niepodzielność zliczania procesów pisarzy/czytelników czekających na dostęp do zasobu, drugi zapewniający wyłączność dostępu do zasobu.

## Problem pięciu ucząjących filozofów

W problemie ucząjących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztuciec (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztuciec. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem **zakleszczenia**. Innym razem może dojść do **zagłodzenia** jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

- Pozwolić jednocześnie zasiadać do stołu co najwyższej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

## Problem pięciu ucząjących filozofów

W problemie ucząjących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztuciec (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztuciec. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem **zakleszczenia**. Innym razem może dojść do **zagłodzenia** jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

- Pozwolić jednocześnie zasiadać do stołu co najwyższej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

## Problem pięciu uczących filozofów

W problemie uczących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztucie (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztucie. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem *zakleszczenia*. Innym razem może dojść do *zagłodzenia* jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

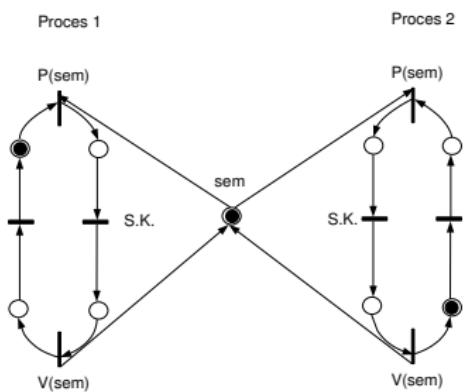
- Pozwolić jednocześnie zasiadać do stołu co najwyższej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

## Problem pięciu uczących filozofów

W problemie uczących filozofów pięciu filozofów (alegoria procesów) zasiada do okrągłego stołu. Każdy z nich ma swój talerz (alegoria zasobów lokalnych), po każdej stronie talerza leży sztucie (alegoria semafora). Na środku stołu stoi duży talerz z jedzeniem (alegoria zasobu współdzielonego). Każdy z filozofów może wykonywać w danej chwili jedną z dwóch czynności: myśleć lub jeść. Żeby filozof mógł jeść musi wziąć dwa sztućce, nabrać jedzenia z dużego talerza, na swój talerz i dopiero wtedy może przystąpić do konsumpcji. Problem ten jest źródłem wielu zagadnień związanych z synchronizacją, otóż może się zdarzyć, że wszyscy filozofowie zechcą jeść i jednocześnie sięgną po leżący po lewej stronie sztucie. Okaże się, że żaden z nich nie będzie mógł podnieść sztućca leżącego po prawej stronie (jest ich tylko pięć), koniecznego do spożycia posiłku. Jest to problem **zakleszczenia**. Innym razem może dojść do **zagłodzenia** jednego z filozofów przez pozostałych biesiadników. Trzy najpopularniejsze strategie rozwiązania tego problemu to:

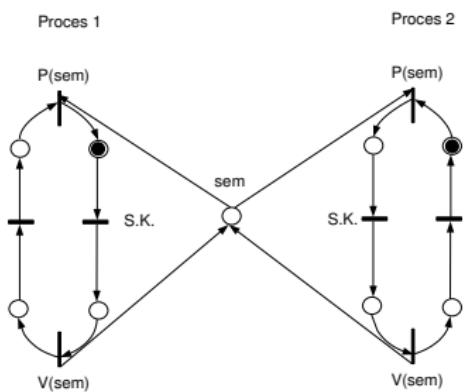
- Pozwolić jednocześnie zasiadać do stołu co najwyższej czterem filozofom.
- Pozwolić podnosić filozofom sztućce tylko wtedy, gdy oba są dostępne.
- Zastosować rozwiązanie asymetryczne: filozofowie o nieparzystych numerach podnoszą sztućce w kolejności lewy - prawy, a ci o numerach parzystych, w kolejności odwrotnej.

## Sieci Petriego



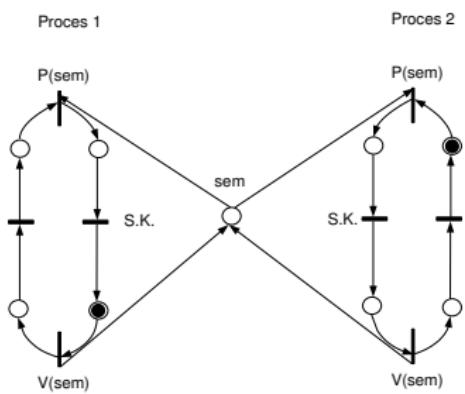
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

## Sieci Petriego



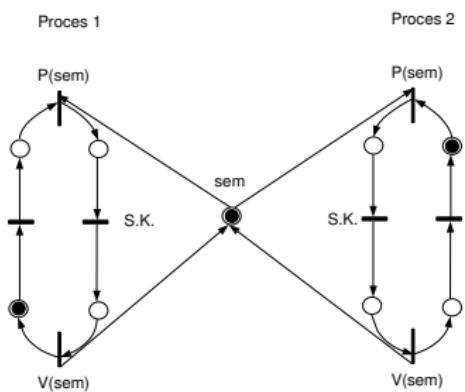
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

## Sieci Petriego



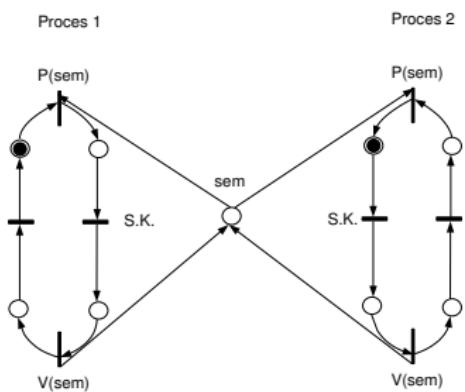
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

## Sieci Petriego



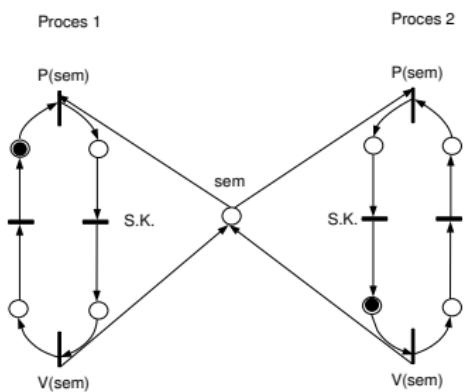
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

## Sieci Petriego



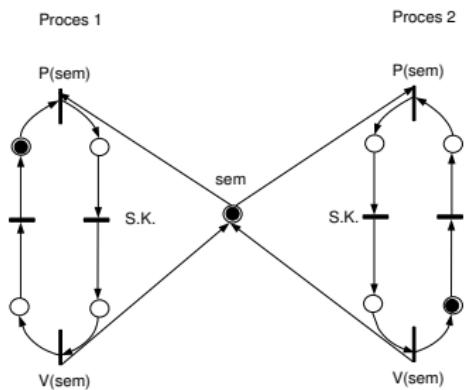
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

## Sieci Petriego



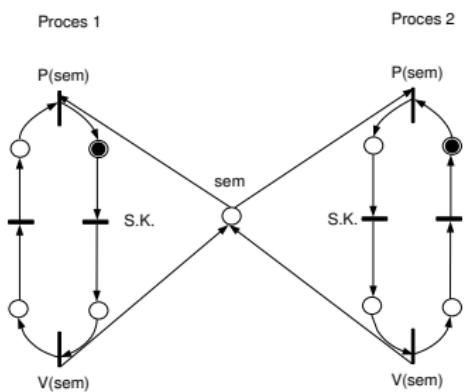
Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

## Sieci Petriego



Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywanie operacji. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

## Sieci Petriego



Sieci Petriego są matematycznym narzędziem analizy systemów współbieżnych i nadają się do modelowania problemów związanych z synchronizacją. Sieć Petriego jest grafem skierowanym, który składa się z dwóch rodzajów wierzchołków, nazywanych miejscami i przejściami. Przejście modelują wykonywane operacje. Przejście może zostać *odpalone* tylko wtedy, gdy we wszystkich miejscach znajdujących się na końcu krawędzi wchodzących do przejścia znajdują się tokeny. Sieci Petriego można opisywać równaniami i udowadniać poprawność systemów modelowanych przez nie. Obok znajduje się przykład sieci modelującej rozwiązanie problemu sekcji krytycznej dla dwóch procesów przy użyciu semafora.

Plan wykładu	Komunikacja bezpośrednią lub pośrednią
Synchronizacja	Buforowanie
Środki synchronizacji	Sytuacje wyjątkowe
Klasyczne problemy synchronizacji	
Sieci Petriego	
Komunikacja międzyprocesowa	

## Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia łączna. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Plan wykładu	Komunikacja bezpośrednią lub pośrednią
Synchronizacja	Buforowanie
Środki synchronizacji	Sytuacje wyjątkowe
Klasyczne problemy synchronizacji	
Sieci Petriego	
Komunikacja międzyprocesowa	

## Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia łączna. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

## Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia łączna. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

## Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia łączna. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

## Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyć komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia łączna. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

Plan wykładu	Komunikacja bezpośrednią lub pośrednią
Synchronizacja	Buforowanie
Środki synchronizacji	Sytuacje wyjątkowe
Klasyczne problemy synchronizacji	
Sieci Petriego	
Komunikacja międzyprocesowa	

## Wprowadzenie

Problemy synchronizacji procesów pojawiają się zawsze w kontekście szerszego zagadnienia jakim jest komunikacja międzyprocesorowa. Najprostszą i najszybszą formą komunikacji między grupą procesów jest komunikacja za pomocą pamięci dzielonej, jednakże dalsza część wykładu będzie dotyczyła komunikacji za pomocą *systemu komunikatów*. Większość współczesnych systemów operacyjnych umożliwia stosowanie obu form komunikacji. System operacyjny, aby zapewnić łączność za pomocą systemu komunikatów musi dostarczyć procesom dwóch operacji *nadaj* i *odbierz* oraz środków tworzenia łączna. W zależności od implementacji wymienionych składników możemy wyróżnić następujące rodzaje komunikacji:

- komunikacja bezpośrednia lub pośrednia,
- komunikacja symetryczna lub asymetryczna,
- komunikacja z buforowaniem automatycznym lub jawnym,
- komunikacja z wysyłaniem na zasadzie tworzenia kopii lub odwołania,
- komunikacja z komunikatami o stałej lub zmiennej długości.

## Komunikacja bezpośrednią lub pośrednią

W komunikacji bezpośrednią łącze jest tworzone tylko między parą procesów, przy czym może ono być jednokierunkowe lub dwukierunkowe. Procesy określają odbiorcę za pomocą jego identyfikatora, którym może być nazwa lub PID, podczas tworzenia łącza. W przypadku komunikacji pośredniej procesy przesyłają sobie komunikaty za pośrednictwem *portu*, który jest nazywany również *skrzynką pocztową*. Skrzynka ta może być tworzona za pomocą wywołania systemowego przez procesy. Właścicielem skrzynki zostaje proces, który wywołał to wywołanie, ale może on również przekazać prawa własności innemu procesowi. Port jest niszczony również za pomocą odpowiedniego wywołania systemowego. Prawa własności skrzynki mogą być również niezbywalne, wówczas taka skrzynka niszczona jest wraz z zakończeniem procesu-właściciela. W komunikacji pośredniej należy zapewnić system identyfikacji odbiorcy i nadawcy komunikatów znajdujących się w skrzynce.

## Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

- ➊ **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
- ➋ **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
- ➌ **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

- ➊ nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
- ➋ praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

## Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

- ① **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
- ② **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
- ③ **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

- ① nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
- ② praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

## Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

- ① **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
- ② **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
- ③ **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

- ① nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
- ② praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

## Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

- ① **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
- ② **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
- ③ **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

- ① nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
- ② praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

## Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

- ① **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
- ② **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
- ③ **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

- ④ nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
- ⑤ praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

## Buforowanie

Istnieją trzy najpopularniejsze scenariusze buforowania komunikatów wysyłanych przez łącze:

- ① **Pojemność zerowa** - czyli brak buforowania, nadawca musi czekać, aż odbiorca odbierze komunikat
- ② **Pojemność ograniczona** - bufor łącza może pomieścić ograniczoną liczbę komunikatów, nadawca czeka tylko wtedy, gdy bufor jest pełny,
- ③ **Pojemność nieograniczona** - bufor komunikatów ma nieograniczoną pojemność, w praktyce ten rodzaj buforowania otrzymuje się stosując dużo buforów i zapewniając że odbiorca będzie co najmniej tak szybki jak nadawca,

Istnieją również dwa przypadki, które nie pasują do żadnej z powyższych kategorii:

- ① nadawca nigdy nie jest opóźniany, a jeśli odbiorca nie zdąży odebrać komunikatu, to jest on po prostu tracony,
- ② praca nadawcy jest wstrzymywana do czasu otrzymania przez niego potwierdzenia odbioru poprzedniego komunikatu.

## Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwane będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Znieształcenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się冗余 (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

## Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwane będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Znieszczycenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się冗余 (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

## Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwane będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Znieształcenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się冗余 (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

## Sytuacje wyjątkowe

Podobnie jak w innych etapach przetwarzania informacji, tak i w komunikacji może dojść do powstania sytuacji wyjątkowych. Najczęściej spotykane to:

- **Zakończenie procesu** Jeśli odbiorca czeka na nadawcę, a on zakończył się, to to oczekiwane będzie nieskończone. Zakończenie odbiorcy jest mniej groźne w przypadku komunikacji buforowanej, chyba że nadawca musi czekać na potwierdzenie odbioru. W każdym z tych przypadków system operacyjny powinien interweniować.
- **Utrata komunikatu** Komunikat, jeśli jest nadawany przez sieć może ulec zagubieniu lub zniszczeniu. Istnieją trzy metody radzenia sobie z takim zdarzeniem: za wykrycie zagubienia komunikatu i jego retransmisję odpowiedzialny jest proces-nadawca (postępowanie właściwe dla protokołów bezpołączeniowych, np. UDP), system operacyjny wykrywa zaginięcie i powiadamia o nim nadawcę, który retransmituje komunikat, system operacyjny jest odpowiedzialny za wykrycie zagubienia pakietu i jego retransmisję (postępowanie właściwe dla protokołów połączeniowych, np. TCP).
- **Znieształcenie komunikatu** Komunikat przesyłany przez sieć może ulec przekłamaniu. Celem wykrycia takiej sytuacji lub wykrycia i naprawy stosuje się冗余 (informację nadmiarową), w postaci kodów detekcyjnych (np. CRC) lub detekcyjno-korekcyjnych np. kody Reeda-Solomona.

## Pytania

?

## Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne — Zakleszczenia

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 29 listopada 2020

# Plan wykładu

## ① Zakleszczenia

- ② Definicja
- ③ Dostęp do zasobów współdzielonych
- ④ Warunki konieczne do wystąpienia zakleszczenia
- ⑤ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ⑥ Wzajemne wykluczanie
- ⑦ Przetrzymywanie i oczekiwanie
- ⑧ Brak wywłaszczeń
- ⑨ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ⑩ Algorytm bankiera
- ⑪ Algorytm bezpieczeństwa
- ⑫ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ⑬ Algorytm wykrywania zakleszczenia
- ⑭ Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

### ① Definicja

- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
  - ③ Warunki konieczne do wystąpienia zakleszczenia
  - ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

- ⑥ Algorytm strusia

# Plan wykładu

## ① Zakleszczenia

- ① Definicja
- ② Dostęp do zasobów współdzielonych
- ③ Warunki konieczne do wystąpienia zakleszczenia
- ④ Metody opisu zakleszczeń

## ② Zapobieganie zakleszczeniom

- ① Wzajemne wykluczanie
- ② Przetrzymywanie i oczekiwanie
- ③ Brak wywłaszczeń
- ④ Czekanie cykliczne

## ③ Unikanie zakleszczeń

- ① Algorytm bankiera
- ② Algorytm bezpieczeństwa
- ③ Metoda dla zasobów reprezentowanych pojedynczo

## ④ Wykrywanie i wychodzenie z zakleszczeń

- ① Algorytm wykrywania zakleszczenia
- ② Metody wychodzenia z zakleszczeń

## ⑤ Łączenie metod postępowania z zakleszczeniami

## ⑥ Algorytm strusia

## Definicja

### Motto

„Jeśli dwa pociągi zbliżają się do siebie krzyżując swe tory, to oba powinny się całkowicie zatrzymać i nie ruszać ponownie, do czasu, aż drugi z nich odjedzie.”  
*prawo uchwalone przez legislaturę stanu Kansas (USA)*

Jeżeli grupa procesów oczekuje na zdarzenie, które może być spowodowane jedynie przez któryś z tych procesów, to taką sytuację nazywamy **zakleszczeniem**<sup>1</sup> lub *impasem* (ang. *deadlock*). Przez grupę rozumiemy dwa lub większą liczbę procesów. Do zjawiska zakleszczenia może zazwyczaj dojść przy próbie dostępu do zasobów współdzielonych. Istnieją również inne zjawiska, które mają skutki podobne do zakleszczenia, ale nie spełniają definicji zakleszczeń. Jednym z nich jest *autozakleszczenie*, do którego może dojść, jeśli proces na skutek błędu logicznego będzie próbował zająć zasób, który wcześniej już zajął. Innym przykładem jest *uwięzienie* (ang. *livelock*), kiedy dwie grupy procesów wykonują przeciwnostawne operacje na wspólnych zasobach, co prowadzi do sytuacji, w której mimo tego, że procesy pracują, to ich praca nie wykazuje postępu.

---

<sup>1</sup>W literaturze polskiej na określenie tego zjawiska był również używany termin „blokada”, obecnie już zarzucony.

## Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

- ➊ **Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
- ➋ **Użycie** Proces korzysta z zasobu.
- ➌ **Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

## Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

- ➊ **Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
- ➋ **Użycie** Proces korzysta z zasobu.
- ➌ **Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

## Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

- ① **Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
- ② **Użycie** Proces korzysta z zasobu.
- ③ **Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

## Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

- ① Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
- ② Użycie** Proces korzysta z zasobu.
- ③ Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

## Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

- ① **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwania na jego zwolnienie.
- ② **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
- ③ **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
- ④ **Czekanie cykliczne** Musi istnieć zbiór procesów  $\{P_0, P_1, \dots, P_n\}$  w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces  $P_n$  czeka na zasób(-oby) przetrzymywany przez  $P_0$ .

## Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

- ① **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwąć na jego zwolnienie.
- ② **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
- ③ **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
- ④ **Czekanie cykliczne** Musi istnieć zbiór procesów  $\{P_0, P_1, \dots, P_n\}$  w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces  $P_n$  czeka na zasób(-oby) przetrzymywany przez  $P_0$ .

## Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

- ① **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwąć na jego zwolnienie.
- ② **Przetrzymywanie i oczekивание** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
- ③ **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
- ④ **Czekanie cykliczne** Musi istnieć zbiór procesów  $\{P_0, P_1, \dots, P_n\}$  w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces  $P_n$  czeka na zasób(-oby) przetrzymywany przez  $P_0$ .

## Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

- ① **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwąć na jego zwolnienie.
- ② **Przetrzymywanie i oczekивание** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
- ③ **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
- ④ **Czekanie cykliczne** Musi istnieć zbiór procesów  $\{P_0, P_1, \dots, P_n\}$  w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces  $P_n$  czeka na zasób(-oby) przetrzymywany przez  $P_0$ .

## Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

- ① **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwąć na jego zwolnienie.
- ② **Przetrzymywanie i oczekивание** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
- ③ **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
- ④ **Czekanie cykliczne** Musi istnieć zbiór procesów  $\{P_0, P_1, \dots, P_n\}$  w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces  $P_n$  czeka na zasób(-oby) przetrzymywany przez  $P_0$ .

## Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

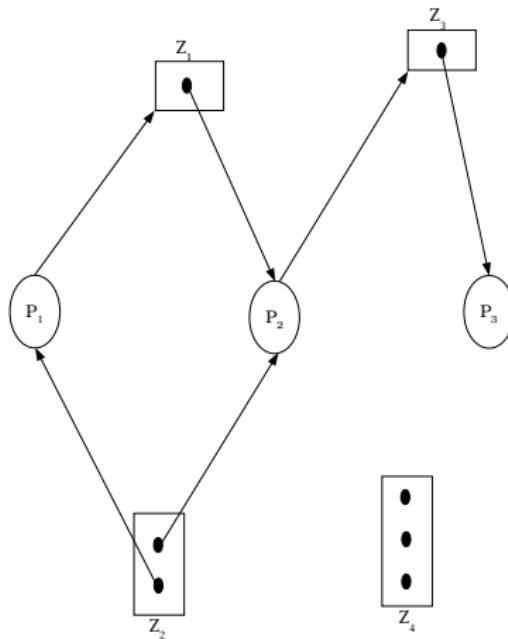
- ① **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekwać na jego zwolnienie.
- ② **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
- ③ **Brak wylącznień** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
- ④ **Czekanie cykliczne** Musi istnieć zbiór procesów  $\{P_0, P_1, \dots, P_n\}$  w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces  $P_n$  czeka na zasób(-oby) przetrzymywany przez  $P_0$ .

**Aby wystąpiło zakleszczenie, wszystkie powyższe warunki muszą zostać spełnione.**

## Metody opisu zakleszczeń

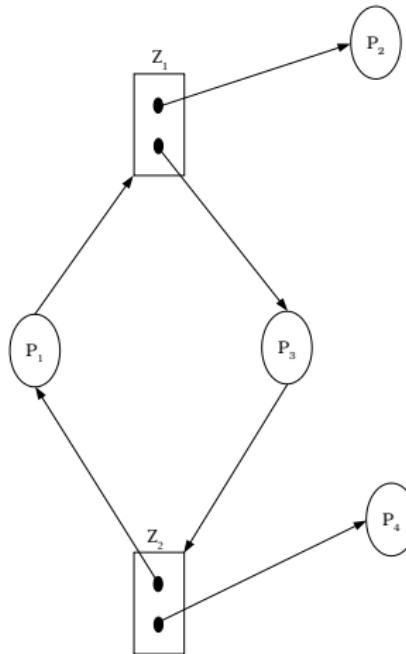
Za pomocą *grafa przydziału zasobów systemu* możemy opisać zjawisko występowania zakleszczeń. Taki graf jest grafem skierowanym, składającym się ze zbioru krawędzi  $\mathbb{K}$  i zbioru wierzchołków  $\mathbb{W}$ . Zbiór wierzchołków dzieli się na dwa podzbiory: zbiór  $\mathbb{P}$  wszystkich procesów systemu, oraz zbiór  $\mathbb{Z}$  wszystkich typów zasobów w systemie. Krawędź skierowaną od procesu  $P_i$  do zasobu  $Z_j$  ( $P_i \rightarrow Z_j$ ) nazywamy *krawędzią zamówienia*. Krawędź skierowaną odwrotnie ( $P_i \leftarrow Z_j$ ) nazywamy *krawędzią przydziału*. Na następnych trzech planszach znajdują się grafy przydziału zasobów, które modelują różne scenariusze związane z przydziałem zasobów.

## Graf acykliczny — brak zakleszczeń



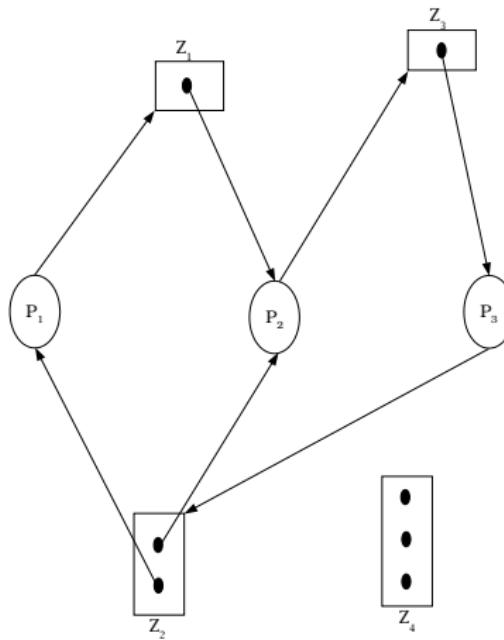
Przedstawiony po lewej stronie graf przydziału zasobów składa się z trzech wierzchołków  $P$  i czterech wierzchołków  $Z$ . Punkty wewnętrz wierzchołków  $Z$  oznaczają pojedyncze egzemplarze zasobów danego typu. Należy zwrócić uwagę, że krawędzie zamówienia biegną od wierzchołka procesu do wierzchołka typu zasobu, natomiast krawędzie przydziału biegną od konkretnego egzemplarza zasobu, do wierzchołka procesu, który go otrzymał. Ponieważ w grafie nie ma cyklu, to żadne procesy nie uległy zakleszczeniu.

## Graf cykliczny — brak zakleszczeń



W przedstawionym obok grafie przydzię-  
łu zasobów występuje cykl ( $P_1 \rightarrow Z_1 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_1$ ), mimo to nie dochodzi  
do zakleszczenia.

## Graf cykliczny — zakleszczenie



W grafie po lewej stronie występują dwa cykle minimalne:

$P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_3 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_1$

oraz

$P_2 \rightarrow Z_3 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_2$ .

W tym grafie dochodzi do zakleszczenia.

## Grafy przydziału zasobów — podsumowanie

*Warunkiem koniecznym* do wystąpienia zakleszczenia w systemie jest to, aby w grafie przydziału zasobów powstał cykl. Nie jest to jednak *warunek wystarczający*. Jak pokazuje to drugi z zaprezentowanych rysункów, w grafie może powstać cykl, mimo to nie dojdzie do zakleszczenia.

Plan wykładu	Wzajemne wykluczanie
Zakleszczenia	Przetrzymywanie i oczekiwanie
<b>Zapobieganie zakleszczeniom</b>	Brak wywłaszczeń
Unikanie zakleszczeń	Cykliczne czekanie
Wykrywanie i wychodzenie z zakleszczeń	
Łączenie metod	

## Zapobieganie zakleszczeniom

Istnieją trzy rodzaje metod radzenia sobie z zakleszczeniami. Nim zapoznamy się z nimi warto zaznaczyć, że większość dzisiejszych systemów operacyjnych nie stosuje żadnych metod zawiązanych z obsługą zakleszczeń, lub stosuje je nie wprost, czyli nie w postaci wbudowanych mechanizmów. Opisy tych metod zaczniemy od metod zapobiegania zakleszczeniom. Aby w systemie komputerowym nigdy nie wystąpiło zakleszczenie, trzeba zadbać, aby *co najmniej jeden* warunek konieczny do wystąpienia zakleszczenia nie był nigdy spełniony. Na kolejnych planszach zostaną przeanalizowane możliwości zapobiegania spełnieniu poszczególnych warunków.

	Plan wykładu	
	Zakleszczenia	
<b>Zapobieganie zakleszczeniom</b>		
	Unikanie zakleszczeń	
	Wykrywanie i wychodzenie z zakleszczeń	
	Łączenie metod	

## Wzajemne wykluczanie

Zarówno zasoby lokalne procesów, jak i zasoby przeznaczone tylko do odczytu (np.: wybrane przez twórców pliki, port zegara czasu rzeczywistego) nie muszą być objęte warunkiem wzajemnego wykluczania. Inaczej jest w przypadku tzw. *zasobów niepodzielnych*, które mogą być modyfikowane. One muszą być użytkowane na zasadzie wyłączności, czego wymaga problem sekcji krytycznej. Ponieważ takie zasoby zawsze istnieją w systemie komputerowym, to warunek wzajemnego wykluczania jest wręcz niezbędny do poprawnej pracy procesów użytkownika.

Plan wykładu	Wzajemne wykluczanie
Zakleszczenia	Przetrzymywanie i oczekiwanie
<b>Zapobieganie zakleszczeniom</b>	Brak wywłaszczeń
Unikanie zakleszczeń	Cykliczne czekanie
Wykrywanie i wychodzenie z zakleszczeń	
Łączenie metod	

## Przetrzymywanie i oczekiwanie

Aby zapewnić, że ten warunek nigdy nie będzie spełniony należy wymusić na procesach, aby zamawiały zasoby, tylko wtedy, gdy nie są w posiadaniu innych zasobów. Istnieją dwa sposoby na osiągnięcie tego celu. Pierwszy polega na tym, że każdy proces musi zamawiać wszystkie niezbędne do pracy zasoby zaraz po uruchomieniu. Wywołania wszystkich niezbędnych funkcji systemowych (wywołań) związanych z zamawianiem zasobów muszą w programach poprzedzać wszelkie inne operacje. Drugi sposób nakazuje procesowi przed zamówieniem nowego zasobu oddanie wszystkich, które posiada. Oba te sposoby mogą prowadzić do zmniejszenia stopnia wykorzystania zasobów współdzielonych i/lub do głodzenia procesów.

Plan wykładu	Wzajemne wykluczanie
Zakleszczenia	Przetrzymywanie i oczekiwanie
<b>Zapobieganie zakleszczeniom</b>	<b>Brak wywłaszczeń</b>
Unikanie zakleszczeń	Cykliczne czekanie
Wykrywanie i wychodzenie z zakleszczeń	
Łączenie metod	

## Brak wywłaszczeń

Brak spełnienia tego warunku można również zapewnić na dwa sposoby, poprzez określenie odpowiedniego protokołu zamawiania zasobów. W pierwszym rozwiążaniu, jeśli proces zamawiający zasób musi czekać na jego przydział, to system operacyjny niejawnie odbiera mu wszystkie zasoby jakie są w jego posiadaniu i niejawnie dopisuje je do listy jego zamówień. Proces jest budzony z oczekiwania dopiero wtedy, gdy można przydzielić mu wszystkie te zasoby. Drugi sposób zakłada, że jeśli proces zamawia zasób, który jest w posiadaniu innego procesu, który czeka na przydział innych zasobów, to temu ostatniemu procesowi zasób jest odbierany i przydzielany pierwszemu. Ostatni protokół nie nadaje się dla zasobów, których stanu nie można łatwo skopiować i w przyszłości odtworzyć.

## Czekanie cykliczne

Można zagwarantować niespełnienie warunku czekania cyklicznego nadając każdemu zasobowi określony, unikatowy numer porządkowy (liczbę naturalną) i wymuszając na procesach zamawianie zasobów, według rosnącej numeracji. Oznacza to, że proces, który ma już w swoim posiadaniu zasób nr 4, może zająć zasób nr 5, ale już nie może zająć zasobu nr 3, a nawet kolejnego egzemplarza zasobu nr 4. Alternatywnie można wymagać, aby proces posiadający zasób nr 4 zwolnił go przed zamówieniem zasobu nr 3. Na zasadzie konwencji ta metoda jest stosowana w jądrze Linuksa, tzn. programiści przestrzegają, aby np.: semafory były zajmowane i zwalniane w określonej kolejności.

## Unikanie zakleszczeń

Ponieważ zapobieganie zakleszczeniom polega na ograniczaniu przydziału zasobów procesom, to jego niekorzystnym skutkiem ubocznym może być zmniejszenie wykorzystania tych zasobów, co z kolei prowadzi do obniżenia przepustowości systemu. Istnieją jednak inne metody, które można stosować zamiast zapobiegania zakleszczeniom. Jedną z tych metod jest *unikanie zakleszczeń*. Analizując grafy przydziału zasobów, można zauważać, że system może znaleźć się w trzech stanach: *stanie bezpiecznym*, *stanie zagrożonym* i *stanie zakleszczenia*. Należy podkreślić, że stan zagrożenia nie zawsze jest stanem zakleszczenia, ale stan zakleszczenia jest zawsze stanem zagrożenia. Unikanie zakleszczeń pozwala utrzymywać system w stanie bezpiecznym. Aby to osiągnąć system operacyjny musi wiedzieć jak będzie następowało zamawianie zasobów przez poszczególne procesy. Mając te informacje może on dynamicznie określać, czy dany ciąg zamówień dokonywanych przez procesy jest *ciągiem bezpiecznym*, czy nie prowadzi do cyklicznego oczekiwania. W wyniku stosowania tej metody proces może nie otrzymać zasobu, mimo że jest on wolny, jeśli realizacja tego zamówienia prowadziłaby do stanu zagrożenia. Jednym z algorytmów pozwalających na unikanie zakleszczeń jest *algorytm bankiera* przedstawiony na następnych planszach.

## Struktury danych wymagane przez algorytm bankiera

Założymy, że w systemie mamy  $n$  procesów i  $m$  typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli  $\text{Dostępne}[j] == x$ , to oznacza, że dostępnych jest  $x$  egzemplarzy zasobu typu  $j$ .
- **Maksymalne** - macierz, o wymiarach  $n \times m$ , która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli  $\text{Maksymalne}[i][j] == x$ , to oznacza, że proces  $P_i$  może zamówić maksymalnie  $x$  egzemplarzy zasobu typu  $j$ .
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru  $n \times m$  określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że  $\text{Potrzebne}[i][j] == \text{Maksymalne}[i][j] - \text{Przydzielone}[i][j]$ .
- **Zamówienia** - tablica o  $m$  elementach opisująca pojedyncze zamówienie procesu  $P_i$  na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów:  $X \leq Y \Leftrightarrow \forall i = 1, 2, \dots, n, X[i] \leq Y[i]$

## Struktury danych wymagane przez algorytm bankiera

Założymy, że w systemie mamy  $n$  procesów i  $m$  typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli  $\text{Dostępne}[j] == x$ , to oznacza, że dostępnych jest  $x$  egzemplarzy zasobu typu  $j$ .
- **Maksymalne** - macierz, o wymiarach  $n \times m$ , która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli  $\text{Maksymalne}[i][j] == x$ , to oznacza, że proces  $P_i$  może zamówić maksymalnie  $x$  egzemplarzy zasobu typu  $j$ .
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru  $n \times m$  określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że  $\text{Potrzebne}[i][j] == \text{Maksymalne}[i][j] - \text{Przydzielone}[i][j]$ .
- **Zamówienia** - tablica o  $m$  elementach opisująca pojedyncze zamówienie procesu  $P_i$  na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów:  $X \leq Y \Leftrightarrow \forall i = 1, 2, \dots, n, X[i] \leq Y[i]$

## Struktury danych wymagane przez algorytm bankiera

Założymy, że w systemie mamy  $n$  procesów i  $m$  typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli  $\text{Dostępne}[j] == x$ , to oznacza, że dostępnych jest  $x$  egzemplarzy zasobu typu  $j$ .
- **Maksymalne** - macierz, o wymiarach  $n \times m$ , która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli  $\text{Maksymalne}[i][j] == x$ , to oznacza, że proces  $P_i$  może zamówić maksymalnie  $x$  egzemplarzy zasobu typu  $j$ .
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru  $n \times m$  określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że  $\text{Potrzebne}[i][j] == \text{Maksymalne}[i][j] - \text{Przydzielone}[i][j]$ .
- **Zamówienia** - tablica o  $m$  elementach opisująca pojedyncze zamówienie procesu  $P_i$  na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów:  $X \leq Y \Leftrightarrow \forall i = 1, 2, \dots, n, X[i] \leq Y[i]$

## Struktury danych wymagane przez algorytm bankiera

Założymy, że w systemie mamy  $n$  procesów i  $m$  typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli  $\text{Dostępne}[j] == x$ , to oznacza, że dostępnych jest  $x$  egzemplarzy zasobu typu  $j$ .
- **Maksymalne** - macierz, o wymiarach  $n \times m$ , która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli  $\text{Maksymalne}[i][j] == x$ , to oznacza, że proces  $P_i$  może zamówić maksymalnie  $x$  egzemplarzy zasobu typu  $j$ .
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru  $n \times m$  określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że  $\text{Potrzebne}[i][j] == \text{Maksymalne}[i][j] - \text{Przydzielone}[i][j]$ .
- **Zamówienia** - tablica o  $m$  elementach opisująca pojedyncze zamówienie procesu  $P_i$  na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów:  $X \leq Y \Leftrightarrow \forall i = 1, 2, \dots, n, X[i] \leq Y[i]$

## Struktury danych wymagane przez algorytm bankiera

Założymy, że w systemie mamy  $n$  procesów i  $m$  typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli  $\text{Dostępne}[j] == x$ , to oznacza, że dostępnych jest  $x$  egzemplarzy zasobu typu  $j$ .
- **Maksymalne** - macierz, o wymiarach  $n \times m$ , która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli  $\text{Maksymalne}[i][j] == x$ , to oznacza, że proces  $P_i$  może zamówić maksymalnie  $x$  egzemplarzy zasobu typu  $j$ .
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru  $n \times m$  określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że  $\text{Potrzebne}[i][j] == \text{Maksymalne}[i][j] - \text{Przydzielone}[i][j]$ .
- **Zamówienia** - tablica o  $m$  elementach opisująca pojedyncze zamówienie procesu  $P_i$  na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów:  $X \leq Y \Leftrightarrow \forall i = 1, 2, \dots, n, X[i] \leq Y[i]$

## Struktury danych wymagane przez algorytm bankiera

Założymy, że w systemie mamy  $n$  procesów i  $m$  typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli  $\text{Dostępne}[j] == x$ , to oznacza, że dostępnych jest  $x$  egzemplarzy zasobu typu  $j$ .
- **Maksymalne** - macierz, o wymiarach  $n \times m$ , która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli  $\text{Maksymalne}[i][j] == x$ , to oznacza, że proces  $P_i$  może zamówić maksymalnie  $x$  egzemplarzy zasobu typu  $j$ .
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru  $n \times m$  określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że  $\text{Potrzebne}[i][j] == \text{Maksymalne}[i][j] - \text{Przydzielone}[i][j]$ .
- **Zamówienia** - tablica o  $m$  elementach opisująca pojedyncze zamówienie procesu  $P_i$  na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów:  $X \leq Y \Leftrightarrow \forall i = 1, 2, \dots, n, X[i] \leq Y[i]$

## Algorytm bankiera

Założymy, że proces  $P_i$  dokonuje zamówienia. Aby sprawdzić, czy może zrealizować to zamówienie system operacyjny wykonuje następujący algorytm (zapis **Przydzielone**, oznacza i-ty wiersz macierzy **Przydzielone**, który traktujemy jak wektor):

- ① Jeśli  $\text{Zamówienia}_i \leq \text{Potrzebne}_i$ ; to wykonaj krok 2. W przeciwnym przypadku wyrzuć wyjątek - proces przekroczył zgłoszone maksymalne zapotrzebowanie na zasoby.
- ② Jeśli  $\text{Zamówienia}_i \leq \text{Dostępne}_i$ , to wykonaj krok 3. W przeciwnym razie proces  $P_i$  musi przejść w stan oczekiwania, bo żądane przez niego zasoby są niedostępne.
- ③ System podejmuje próbę przydzielenia zasobów procesowi  $P_i$ , modyfikując zawartość odpowiednich macierzy:

$\text{Dostępne} \leftarrow \text{Dostępne} - \text{Zamówienia}_i;$   
 $\text{Przydzielone}_i \leftarrow \text{Przydzielone}_i + \text{Zamówienia}_i;$   
 $\text{Potrzebne}_i \leftarrow \text{Potrzebne}_i - \text{Zamówienia}_i;$

Po wykonaniu tej modyfikacji system sprawdza, czy stan wynikowy jest bezpieczny, wykonując *algorytm bezpieczeństwa* opisany na następnej planszy. Jeśli po wykonaniu przedziału system byłby w stanie zagrożenia, to ten przedział jest wstrzymywany.

## Algorytm bezpieczeństwa

- ① Niech **Praca** i **Koniec** oznaczają wektory o długości odpowiednio  $m$  i  $n$ . W pierwszym kroku dokonywane są dwa przypisania: **Praca**  $\leftarrow$  **Dostępne** oraz **Koniec[i]**  $\leftarrow$  *false* dla  $i = 1, 2, \dots, n$ .
- ② Znajdujemy takie  $i$ , że zarówno:  
**Koniec[i] == false;**  
**Potrzebne<sub>i</sub> ≤ Praca<sub>i</sub>**;   
Jeśli takie  $i$  nie istnieje, to wykonujemy krok 4.
- ③ **Praca**  $\leftarrow$  **Praca** + **Przydzielone<sub>i</sub>**;  
**Koniec[i]**  $\leftarrow$  *true*;  
Skok do kroku 2.
- ④ Jeśli **Koniec[i] == true** dla wszystkich możliwych  $i$ , to system jest w stanie bezpiecznym.

## Unikanie zakleszczeń dla zasobów reprezentowanych pojedynczo

Złożoność algorytmu bankiera wynosi  $O(m \times n^2)$ . W systemach, w których występuje tylko jeden egzemplarz wszystkich typów zasobów można zastosować prostszy algorytm o mniejszej złożoności. Jest to algorytm grafowy, a graf na którym przeprowadzane są operacje jest zmodyfikowanym grafem przydziału zasobów. Modyfikacja polega na wprowadzeniu nowego rodzaju krawędzi, zwanych *krawędziami deklaracji*. Krawędź deklaracji  $P_i \rightarrow Z_j$  określa, że proces  $P_i$  będzie chciał w przyszłości zamówić zasób  $Z_j$ . Jeśli dojdzie do takiego zamówienia, to krawędź deklaracji zamieniana jest automatycznie w krawędź zamówienia. Dla odróżnienia na rysunku tych dwóch krawędzi, krawędź deklaracji rysujemy przerywaną linią. Po oddaniu przez proces zasobu krawędź przydziału jest zmieniana w krawędź deklaracji. Można wymagać, aby krawędzie deklaracji pojawiały się w grafie wraz z wierzchołkiem procesu, albo żeby nowe krawędzie tego typu były dodawane tylko wtedy, gdy wszystkie krawędzie związane z procesem są krawędziami deklaracji. Do wykrywania zakleszczeń w takim grafie wystarczy zastosować algorytm wykrywania cykli, o złożoności  $O(n^2)$ .

## Wykrywanie i wychodzenie z zakleszczeń

Trzecia metoda radzenia sobie z zakleszczeniami polega na ich wykrywaniu i wychodzeniu z nich. Potrzebne są więc dwa odrębne algorytmy

- Algorytm wykrywania zakleszczenia.
- Algorytm wychodzenia z zakleszczenia.

Analizę tego rozwiązania rozpoczęniemy opisem pierwszego z wymienionych algorytmów. Proszę zwrócić uwagę na podobieństwo tego algorytmu do algorytmu bezpieczeństwa.

## Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru  $n \times m$  określająca bieżące zamówienia każdego z procesów.

## Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru  $n \times m$  określająca bieżące zamówienia każdego z procesów.

## Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru  $n \times m$  określająca bieżące zamówienia każdego z procesów.

## Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru  $n \times m$  określająca bieżące zamówienia każdego z procesów.

## Struktury danych dla algorytmu wykrywania zakleszczeń

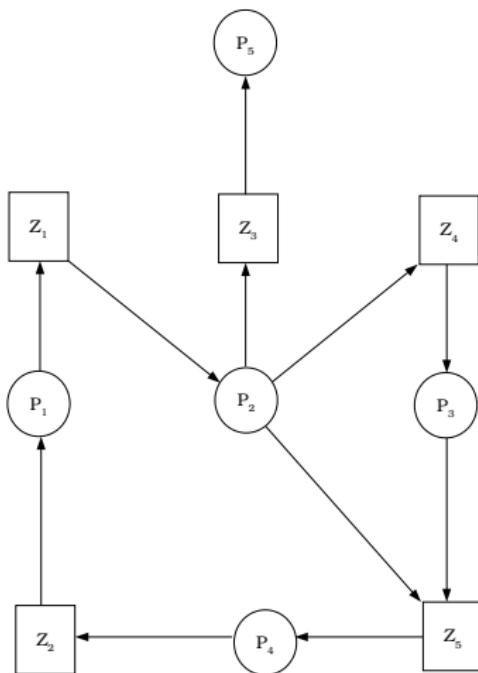
- **Dostępne** - tablica (wektor) o  $m$  elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach  $n \times m$ , określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru  $n \times m$  określająca bieżące zamówienia każdego z procesów.

Dodatkowo przyjmujemy że działanie operatora „mniejszy lub równy” jest takie samo, jak w algorytmie bankiera.

## Algorytm wykrywania zakleszczeń

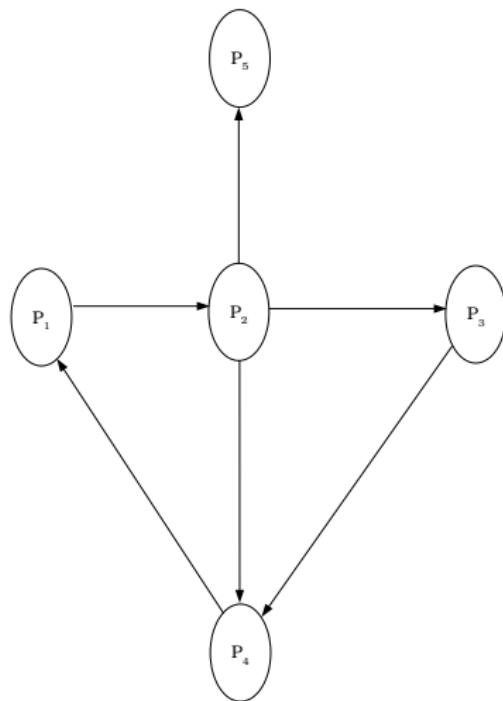
- ① Niech **Praca** i **Koniec** oznaczają wektory o długości odpowiednio  $m$  i  $n$ . W pierwszym kroku dokonywane są dwa przypisania: **Praca**  $\leftarrow$  **Dostępne** i jeśli **Przydzielone**  $\neq 0$  to **Koniec**[ $i$ ]  $\leftarrow$  *false*, w przeciwnym przypadku **Koniec**[ $i$ ]  $\leftarrow$  *true*, dla  $i = 1, 2, \dots, n$ .
- ② Znajdujemy takie  $i$ , że zarówno:  
**Koniec**[ $i$ ]  $=$  *false*;  
**Zamówienia** $_i \leqslant$  **Praca**;  
Jeśli takie  $i$  nie istnieje, to wykonujemy krok 4.
- ③ **Praca**  $\leftarrow$  **Praca** + **Przydzielone** $_i$ ;  
**Koniec**[ $i$ ]  $\leftarrow$  *true*;  
Skok do kroku 2.
- ④ Jeśli dla pewnych wartości  $i$  z przedziału  $[1, n]$ , **Koniec**[ $i$ ]  $=$  *false*, to system jest w stanie zakleszczenia, dokładniej wszystkie procesy dla których **Koniec**[ $i$ ]  $=$  *false* są w stanie zakleszczenia.

## Algorytm wykrywania zakleszczeń dla zasobów reprezentowanych pojedynczo



Algorytm wykrywania zakleszczeń można uprościć dla systemów, w których występuje tylko jeden egzemplarz zasobu danego typu. Można go sprowadzić do algorytmu wykrywania cykli w grafie oczekiwania. Taki graf powstaje z przekształcenia grafu przydziału zasobów, poprzez usunięcie wierzchołków reprezentujących typy zasobów i połączenie powstałych w ten sposób wolnych krawędzi.

## Algorytm wykrywania zakleszczeń dla zasobów reprezentowanych pojedynczo



Algorytm wykrywania zakleszczeń można uprościć dla systemów, w których występuje tylko jeden egzemplarz zasobu danego typu. Można go sprowadzić do algorytmu wykrywania cykli w grafie oczekiwania. Taki graf powstaje z przekształcenia grafu przydziału zasobów, poprzez usunięcie wierzchołków reprezentujących typy zasobów i połączenie powstałych w ten sposób wolnych krawędzi.

## Korzystanie z algorytmu wykrywania zakleszczeń

Jeśli algorytm wykrywania zakleszczeń ma zostać zastosowany w rzeczywistym systemie, to należy odpowiedzieć na pytanie: „Jak często należy go wykonywać?” Przewrotna odpowiedź brzmi: „Częściej niż występują zakleszczenia.” W praktyce może to oznaczać, że algorytm ten będzie wykonywany po każdym żądaniu przydziału zasobu. To z kolei może prowadzić do powstania dużych narutów czasowych, związanych z jego wykonaniem. Inne podejście może bazować na mierze przepustowości systemu. Jeśli spadnie ona poniżej określonego progu, to należy algorytm wykrywania zakleszczeń wykonać. Przy tym rozwiążaniu może okazać się niemożliwe zidentyfikowanie sprawców zakleszczenia.

## Wychodzenie z zakleszczenia

Po wykryciu zakleszczenia system operacyjny może powiadomić użytkownika i jemu pozostawić usunięcie tego problemu. Może również sam podjąć kroki zmierzające do wydostania się z zakleszczenia. Wszystkie strategie wychodzenia z zakleszczenia można podzielić na dwie kategorie:

- ① zakańczanie procesów,
- ② wywłaszczanie zasobów.

Kolejne plansze zawierają omówienie tych metod.

## Wychodzenie z zakleszczenia

Po wykryciu zakleszczenia system operacyjny może powiadomić użytkownika i jemu pozostawić usunięcie tego problemu. Może również sam podjąć kroki zmierzające do wydostania się z zakleszczenia. Wszystkie strategie wychodzenia z zakleszczenia można podzielić na dwie kategorie:

- ① zakańczanie procesów,
- ② wywłaszczanie zasobów.

Kolejne plansze zawierają omówienie tych metod.

## Wychodzenie z zakleszczenia

Po wykryciu zakleszczenia system operacyjny może powiadomić użytkownika i jemu pozostawić usunięcie tego problemu. Może również sam podjąć kroki zmierzające do wydostania się z zakleszczenia. Wszystkie strategie wychodzenia z zakleszczenia można podzielić na dwie kategorie:

- ① zakańczanie procesów,
- ② wywłaszczanie zasobów.

Kolejne plansze zawierają omówienie tych metod.

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
    - Zasoby wymagane przez proces do zakończenia pracy.
    - Ile innych procesów trzeba będzie zakończyć wraz z nim.
    - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

- ① **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
- ② **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
  - Priorytet procesu.
  - Bieżący i pozostały czas pracy procesu.
  - Zasoby posiadane przez proces.
  - Zasoby wymagane przez proces do zakończenia pracy.
  - Ile innych procesów trzeba będzie zakończyć wraz z nim.
  - Typ procesu (interaktywny lub wsadowy).

## Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznowianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to aby wybór ofiary nie prowadził do głodzenia jednego z procesów, które mogą uczestniczyć w zakleszczeniu.

## Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznowianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to aby wybór ofiary nie prowadził do głodzenia jednego z procesów, które mogą uczestniczyć w zakleszczeniu.

## Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznowianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to aby wybór ofiary nie prowadził do głodzenia jednego z procesów, które mogą uczestniczyć w zakleszczeniu.

## Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznowianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to aby wybór ofiary nie prowadził do głodzenia jednego z procesów, które mogą uczestniczyć w zakleszczeniu.

## Łączenie metod radzenia sobie z zakleszczeniami

Metody radzenia sobie z zakleszczeniami nie są metodami wzajemnie się wykluczającymi. Może się więc okazać korzystne zastosowanie w systemie wszystkich tych metod równocześnie. Łączenia działania takich algorytmów dokonuje się określając dla poszczególnych grup zasobów inne scenariusze rozwiązywania problemu zakleszczeń.

## Algorytm strusia

Wielu programistów systemowych stosuje podejście do problemu zakleszczeń, które nazywa się *algorytmem strusia*. Polega on po prostu na ignorowaniu problemu. Wbrew pozorom taka strategia nie jest bezpodstawna. Zakleszczenia mogą pojawiać się w systemie tak rzadko, że wprowadzenie mechanizmów zapobiegania, unikania bądź wykrywania i wychodzenia z zakleszczeń staje się nieekonomiczne, z uwagi na ograniczenia nakładane na system przez te mechanizmy.

## Pytania

?

Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne — Zarządzanie pamięcią operacyjną

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 29 listopada 2020

# Plan wykładu

- ① Zagadnienia podstawowe
  - ② Wiązanie adresów
  - ③ Ładowanie dynamiczne
  - ④ Łączenie dynamiczne
  - ⑤ Nakładki
- ② Wymiana
- ③ Przydział ciągłych obszarów
  - ④ Przydział pojedynczego obszaru
  - ⑤ Przydział wielu obszarów
- ④ Stronicowanie
  - ⑥ Zasada działania
  - ⑦ Wspomaganie sprzętowe
  - ⑧ Strony współdzielone i ochrona
- ⑤ Segmentacja
  - ⑨ Zasada działania
  - ⑩ Wspomaganie sprzętowe
  - ⑪ Współdzielone segmenty i ochrona
- ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
  - ③ Łączenie dynamiczne
  - ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

### ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

### ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

### ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

### ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

## ① Zagadnienia podstawowe

- ① Wiązanie adresów
- ② Ładowanie dynamiczne
- ③ Łączenie dynamiczne
- ④ Nakładki

## ② Wymiana

## ③ Przydział ciągłych obszarów

- ① Przydział pojedynczego obszaru
- ② Przydział wielu obszarów

## ④ Stronicowanie

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Strony współdzielone i ochrona

## ⑤ Segmentacja

- ① Zasada działania
- ② Wspomaganie sprzętowe
- ③ Współdzielone segmenty i ochrona

## ⑥ Segmentacja stronicowana

# Plan wykładu

- ① Zagadnienia podstawowe
  - ① Wiązanie adresów
  - ② Ładowanie dynamiczne
  - ③ Łączenie dynamiczne
  - ④ Nakładki
- ② Wymiana
- ③ Przydział ciągłych obszarów
  - ① Przydział pojedynczego obszaru
  - ② Przydział wielu obszarów
- ④ Stronicowanie
  - ① Zasada działania
  - ② Wspomaganie sprzętowe
  - ③ Strony współdzielone i ochrona
- ⑤ Segmentacja
  - ① Zasada działania
  - ② Wspomaganie sprzętowe
  - ③ Współdzielone segmenty i ochrona
- ⑥ Segmentacja stronicowana

# Wstęp

Najprostszy model pamięci operacyjnej opisuje ją jako ciąg komórek (lokacji) jednakowego rozmiaru, z których każda posiada unikalny identyfikator. Rozmiar komórki najczęściej wynosi jeden bajt, w przypadku ogólnym jest natomiast wyrażony słowem binarnym. Identyfikator komórki nazywany *adresem* jest liczbą całkowitą, najczęściej liczbą naturalną, która w sposób jednoznaczny identyfikuje tę komórkę. Procesor jest połączony z pamięcią trzema magistralami: *magistralą sterowania*, *magistralą danych* i *magistralą adresową*. W komunikacji między tymi układami często również pośredniczy jednostka zarządzania pamięcią (ang. Memory Management Unit - MMU). Pamięć operacyjna stanowi podstawowy magazyn danych procesora. Każdy program, który podlega wykonaniu musi być w niej umieszczony. W dalszej części wykładu zajmiemy się sposobami umiejscowienia go w pamięci operacyjnej oraz tym co dzieje się z wytworzonymi przez niego adresami na drodze procesor - pamięć. Nie będzie nas natomiast interesował sposób tworzenia tych adresów.

## Wiązanie adresów

Program komputerowy przed wykonaniem znajduje się najczęściej na dysku twardym lub innym nośniku w postaci wykonywalnego pliku binarnego. Na zlecenie użytkownika lub określonego procesu może on zostać załadowany do pamięci operacyjnej i wykonyany. W systemach wsadowych najpierw trafia do *kolejki wejściowej* na dysku. Jeśli program będzie wykonywany wielokrotnie, to może się okazać, że za każdym razem będzie ładowany do innego miejsca w pamięci. Pamięć komputera zaczyna się zazwyczaj od adresu zero, natomiast pamięć programu, czyli ten obszar pamięci fizycznej, który został przydzielony mu przez system operacyjny, najczęściej nie. Konieczne jest więc przeprowadzenie *wiązania adresów* wytwarzanych przez program z adresami fizycznymi. Ta operacja może zostać przeprowadzona na różnych etapach przygotowania programu do wykonania:

	Plan wykładu
Podstawowe zagadnienia	
	Wymiana
Przydzielanie obszarów ciągłych	
	Stronicowanie
	Segmentacja
	Segmentacja stronicowana
	Wiązanie adresów
	Ładowanie dynamiczne
	Łączenie dynamiczne
	Nakładki

## Wiązanie adresów

- **Etap komplikacji** W kodzie źródłowym adresy są wyrażane w postaci symbolicznej, czyli są to *nazwy zmiennych, podprogramów, itp..* Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie komplikacji, to kompilator przekształca adresy symboliczne na *adresy względne*, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładowający.
- **Etap wykonania.** Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

## Wiązanie adresów

- **Etap komplikacji** W kodzie źródłowym adresy są wyrażane w postaci symbolicznej, czyli są to nazwy zmiennych, podprogramów, itp.. Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie komplikacji, to kompilator przekształca adresy symboliczne na adresy względne, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładowający.
- **Etap wykonania.** Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

## Wiązanie adresów

- **Etap komplikacji** W kodzie źródłowym adresy są wyrażane w postaci symbolicznej, czyli są to nazwy zmiennych, podprogramów, itp.. Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie komplikacji, to kompilator przekształca adresy symboliczne na *adresy względne*, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładowający.
- **Etap wykonania.** Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

## Wiązanie adresów

- **Etap komplikacji** W kodzie źródłowym adresy są wyrażane w postaci symbolicznej, czyli są to nazwy zmiennych, podprogramów, itp.. Jeśli kompilator dysponuje informacjami, gdzie wygenerowany przez niego kod wynikowy będzie umieszczony w pamięci komputera (tak się dzieje jedynie w przypadku systemów, w których może być uruchamiany jedynie jeden proces użytkownika), to zamienia adresy symboliczne bezpośrednio na adresy fizyczne, nazywane *adresami bezwzględnymi*.
- **Etap ładowania** Jeśli informacje na temat umiejscowienia programu w pamięci nie są znane na etapie komplikacji, to kompilator przekształca adresy symboliczne na adresy względne, inaczej *relokowalne* liczone względem początku generowanego przez niego bloku kodu. W ten sposób powstaje kod *przemieszczalny* lub *relokowalny*. Wiązania adresów względnych z adresami fizycznymi dokonuje program ładowający.
- **Etap wykonania.** Jeśli procesor jest wyposażony w odpowiedni sprzęt, to możliwe jest przemieszczanie programu w pamięci podczas jego wykonania, np.: w celu zrobienia miejsca w pamięci innemu programowi, który będzie do niej załadowany. Tę metodę określa się również mianem *dynamicznego wiązania adresów*.

## Ładowanie dynamiczne

Jeśli program jest wygenerowany w postaci przemieszczalnej, to można zastosować technikę pozwalającą na oszczędzanie miejsca w pamięci operacyjnej. Można ów program podzielić na podprogramy, z których każdy będzie rezydował w osobnym pliku binarnym na dysku. Do pamięci jest ładowany jedynie program główny. Jeżeli zajdzie konieczność wywołania któregoś z podprogramów, to najpierw sprawdzane jest, czy znajduje się on w pamięci operacyjnej, jeśli nie to wywoływany jest program ładowany, który go tam umieści. Dzięki tej technice podprogramy są ładowane do pamięci tylko wtedy, gdy są potrzebne.

## Łączenie dynamiczne

Łączenie jest jednym z etapów translacji programu z kodu źródłowego na kod wynikowy. Polega ono na włączeniu wszystkich niezbędnych bibliotek do kodu wynikowego komplilowanego programu. W systemie są zazwyczaj biblioteki, z których korzysta wiele procesów. W wyniku zwykłego łączenia każdy z tych procesów otrzymuje swoją kopię kodu tych bibliotek, którą należy oczywiście umieścić w pamięci operacyjnej. *Łączenie dynamiczne* pozwala uniknąć naruszeń pamięci, które powstają w wyniku zwykłego łączenia. W kodach wynikowych programów stosujących tę technikę, w miejscu odwołania do podprogramu znajdującego się w bibliotece zewnętrznej znajduje się tzw. *zakładka*, czyli fragment kodu, który pozwala znaleźć w pamięci operacyjnej podprogram z odpowiedniej *biblioteki współdzielonej*. Jeśli ta biblioteka nie znajduje się w pamięci to jest ładowana. Usuwanie biblioteki przeprowadzane jest tylko wtedy, gdy jest to konieczne i żaden proces nie korzysta z niej. Biblioteki współdzielone są wyposażane w numer wersji, dzięki temu każdy proces korzystający z łączenia dynamicznego może określić, czy w systemie jest potrzebna mu wersja biblioteki. Pliki w których znajduje się kod binarny bibliotek są plikami wykonywalnymi, mogą być one zgromadzone w jednym miejscu na dysku (w jednym katalogu) bądź mogą być umieszczone w katalogu z kodem binarnym programu. W większości współczesnych systemów zakładka odwołuje się do procesu systemowego, który jest odpowiedzialny za zarządzanie bibliotekami współdzielonymi i ich udostępnianie procesom. W systemie Windows biblioteki współdzielone określane są skrótem DLL (ang. Dynamic Linked Libraries), natomiast w systemach uniksowych przez skrót SO (ang. Shared Object).

## Nakładki

Technika *nakładania* pozwala wykonać program, który ze względu na swoje rozmiary nie może być w całości umieszczony w pamięci operacyjnej. Przypomina ona w działaniu technikę dynamicznego ładowania. *Nakładka* jest fragmentem obrazu programu komputerowego, czyli takiej postaci programu jaka jest umieszczana w pamięci komputera. W czasie działania programu jest ładowana do pamięci ta jego część która jest zawsze niezbędna i która zawiera kod zarządzający nakładkami, oraz potrzebna w danej chwili nakładka. Jeśli ta nakładka przestanie być potrzebna to kod wykonujący nakładanie wysyła ją na dysk i ładuje do pamięci tę, która jest bieżąco potrzebna. Nakładanie nie wymaga wsparcia ze strony systemu operacyjnego i może w całości być realizowane na poziomie programu użytkownika. Oznacza to, że programista tworzący taki program musi samodzielnie zaimplementować tę technikę, co wymaga stosunkowo dobrej wiedzy na temat rozmieszczenia programu w pamięci operacyjnej.

## Prosta wymiana

Jeśli istnieje konieczność zrealizowania przetwarzania wielozadaniowego w systemie komputerowym o ograniczonej pojemności pamięci, to można posłużyć się techniką prostej wymiany. Dzięki niej jedynie proces, który jest w stanie aktywnym musi znajdować się w pamięci operacyjnej. Pozostałe procesy mogą być umieszczone w szybkiej *pamięci pomocniczej* umiejscowionej na dysku twardym. Przełączanie kontekstu obejmuje przeniesienie procesu, który utracił procesor do pamięci pomocniczej i załadowanie do pamięci operacyjnej procesu, który został wybrany przez planistę do wykonania. W systemach z priorytetowym szeregowaniem proces o wyższym priorytecie może wywalać z pamięci operacyjnej proces o niższym priorytecie, przy czym ten ostatni wraca do niej po wykonaniu ważniejszego zadania. Tę odmianę wymiany nazywamy *zwijaniem i rozwijaniem*. Wadą tego rozwiązania jest szybkość działania. Pamięć pomocnicza jest wielokrotnie wolniejsza od pamięci operacyjnej, dlatego przy wymianie potrzebne są informacje nie o tym ile pamięci zostało procesowi przydzielone, ale ile z tej pamięci faktycznie wykorzystuje. Pozwala to uniknąć transferu niepotrzebnych danych. Jeśli system nie stosuje dynamicznego wiązania adresów to przywracany proces musi trafić dokładnie w to samo miejsce w pamięci, które zajmował przed wymianą. Wymianie nie mogą podlegać procesy, które czekają na realizację operacji wejścia-wyjścia dla których bufory przydzielono w obszarze pamięci tych procesów. Prosta wymiana była stosowania w pierwszych systemach uniksowych w wersji BSD. Zastosowana w nich realizacja tej techniki była jednak doskonalsza od oryginalnej idei bo procesy były wymieniane tylko wtedy, gdy brakowało miejsca w pamięci operacyjnej.

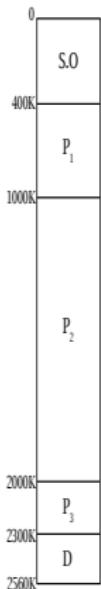
## Przydział pojedynczego obszaru

W najprostszym scenariuszu przydziału pamięci cała dostępna pamięć fizyczna jest przydzielana pojedynczemu procesowi, który oprócz określonych zdań wykonuje wszystkie prace charakterystyczne dla systemu operacyjnego, jak np. obsługa przerwań. Takie rozwiązanie jest spotykane najczęściej w systemach bazujących na mikrokontrolerach. Trochę bardziej skomplikowany jest przypadek, w którym w pamięci rezyduje system operacyjny i dokładnie jeden proces użytkownika. Ponieważ system przerwań stanowi część systemu operacyjnego i musi być wraz z nim chroniony, to pamięć dla systemu operacyjnego jest przydzielana tam, gdzie projektanci procesora określili położenie tablicy wektorów przerwań. Często jest to dolna część pamięci, rozpoczynająca się od adresu 0, rzadziej pamięć górna, kończąca się maksymalnym adresem. Decyzja o położeniu TWP może być też pozostawiana przez twórców sprzętu programiście systemowemu. Problem przydziału pamięci procesowi użytkownika jest trudniejszy. Kod tego procesu może być umieszczony bezpośrednio za kodem systemu operacyjnego. Takie rozwiązanie powoduje problemy, kiedy system operacyjny chce przydzieleć sobie więcej pamięci, np. na bufory wejścia-wyjścia lub gdy korzysta z kodu przejściowego, tj. takiego kodu, który nie rezyduje na stałe w pamięci, a jest ładowany w razie potrzeby. Doraźne rozwiązanie polega na umieszczeniu systemu operacyjnego na początku pamięci (w pamięci dolnej), a procesu użytkownika w pamięci górnej, tak aby ostatni adres programu użytkownika był jednocześnie ostatnim adresem w pamięci operacyjnej. Dzięki temu pośrodku pamięci operacyjnej powstaje obszar pamięci wolnej, z którego mogą dowolnie korzystać oba procesy. Bardziej ogólne rozwiązanie zostanie opisane na następnej planszy.

## Pamięć logiczna i fizyczna

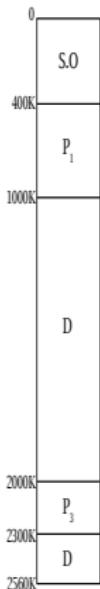
Zauważmy, że w opisany wyżej schemacie, do zapewnienia ochrony pamięci wystarczy zastosować tylko rejestr bazowy, opisany na wcześniejszych wykładach. Rejestr graniczny jest zbędny, bo jednemu procesowi można oddać do dyspozycji całą pamięć znajdująca się za adresem bazowym. Jeśli system operacyjny podczas pracy nie przydziela sobie więcej pamięci operacyjnej, to wartość rejestrów bazowych jest statyczna. Ponadto można w programie użytkownika umieścić adresy bezwzględne już na etapie komplikacji. Jeżeli jednak takie przydziały następują, to trzeba zastosować adresy względne i dynamicznie zmieniać wartość rejestrów bazowych. Przy takim podejściu zastosowanie wiązania adresów tylko podczas ładowania jest niewystarczające. Należyby przerywać pracę programu i ponownie go załadować, uwzględniając nową wartość rejestrów bazowych za każdym razem kiedy system operacyjny przydzieli sobie pamięć. Aby uniknąć takiego problemu stosuje się wiązanie dynamiczne. Rejestr bazowy jest wykorzystywany jako *rejestr relokacji* lub *rejestr przemieszczenia*, którego wartość jest dodawana do każdego adresu wygenerowanego przez proces użytkownika. Należy zauważyć, że mamy teraz do czynienia z dwoma obrazami pamięci. Pierwszy to pamięć rzeczywista, nazywana *pamięcią fizyczną* i związana z nią *fizyczna przestrzeń adresowa*. Jest to cała pamięć, którą dysponuje system komputerowy i którą zarządza system operacyjny. Drugim obrazem pamięci jest pamięć taka, jaką widzi ją proces użytkownika. Tę pamięć nazywamy *pamięcią logiczną* i jest z nią związana *logiczna przestrzeń adresowa*. Proces używa adresów *względnych* począwszy od adresu o wartości zero. Zanim te adresy dotrą do pamięci operacyjnej, to dodawana jest do nich wartość rejestrów relokacji, dzięki czemu adresowane są prawidłowe komórki. Innymi słowy proces posługuje się adresami względnymi z zakresu  $[0, \text{max}]$ , gdzie  $\text{max}$  oznacza adres względny o największej dla tego procesu wartości, a te adresy są tłumaczone dynamicznie na adresy fizyczne z przedziału  $[R, R+\text{max}]$ , gdzie  $R$  to bieżąca wartość rejestrów relokacji. Jeśli chcemy przemieścić proces w pamięci, to wystarczy go skopiować do obszaru docelowego i zmienić zawartość rejestrów bazowych.

## Przydział wielu obszarów



Wyobraźmy sobie, że w kolejce wejściowej systemu, który stosuje szeregowanie długoterminowe algorytmem FCFS, jest pięć procesów, których zapotrzebowania na pamięć wynoszą odpowiednio: 600KB, 1000KB, 300KB, 700KB i 500KB. Dostępnych jest tylko 2560KB pamięci operacyjnej. Można ją przydzielić od razu pierwszemu, drugiemu i trzeciemu procesowi. W pamięci zostanie miejsce wolne, nazywane krótko *dziurą*, które będzie niewystarczające, dla żadnego z pozostałych procesów. Będą musiały one poczekać, aż zakończy się jeden z trzech procesów, które są już w pamięci.

## Przydział wielu obszarów



Załóżmy, że swoją pracę jako pierwszy skończył proces drugi. W pamięci powstają zatem dwa ciągłe obszary wolne. Okazuje się, że dziurę po procesie  $P_2$  można przydzieleć tylko jednemu z procesów, które czekają na przydział pamięci. Tym procesem będzie proces  $P_4$ .

## Przydział wielu obszarów



Zauważmy, że po tym przydiale w pamięci operacyjnej nadal istnieją dwie dziury, o sumarycznej wielkości 560KB. Ta wielkość spełnia wymagania programu piątego (500KB), ale ponieważ te dziury nie tworzą obszaru spójnego, to nie można ich przydzielić temu procesowi.

## Przydział wielu obszarów



Przyjmijmy, że nastepnym procesem, który zakończy swoją pracę będzie proces  $P_1$ . Po jego zakończeniu w pamięci operacyjnej pojawia się ciągły obszar wolny o wielkości 600KB, co pozwala na spełnienie zapotrzebowania na pamięć ze strony procesu piątego. Z tej dziury procesowi  $P_5$  zostanie przydzielone dokładnie 500KB.

## Przydział wielu obszarów



Zauważmy, że po wykonaniu ostatniego z przydziałów w pamięci operacyjnej powstają trzy niespójne wolne obszary o łącznej pojemności 660KB.

## Strategie przydziału

W opisywanym schemacie mamy do czynienia z *dynamicznym przydziałem* pamięci. System operacyjny utrzymuje ewidencję wolnych oraz zajętych obszarów pamięci i na bieżąco podejmuje decyzję, który z wolnych obszarów przydzielić czekającym procesom. To planowanie wymaga określenia strategii wyboru dziury, jeśli jest ich wiele. Oto trzy najpopularniejsze strategie:

- **Pierwsza pasująca**-przydzielana jest pierwsza dziura, która spełnia wymagania procesu co do rozmiaru. Jej poszukiwanie może zawsze się rozpoczynać od początku wykazu miejsc wolnych lub od miejsca ostatniego przydziału.
- **Najlepiej pasującą**-przydziela się dziurę, która dokładnie spełnia wymagania co do rozmiaru, lub dla której różnica między jej rozmiarem, a rozmiarem wymaganym jest najmniejsza spośród pozostałych dziur.
- **Najgorzej pasującą**-przydziela się największą dziurę w systemie. Zakłada się, że proces zażąda w czasie wykonania dodatkowej pamięci. Przy zastosowaniu tej strategii istnieje duże prawdopodobieństwo, że żądanie to zostanie od razu zrealizowane.

Symulacje wykazały, że najlepsze rezultaty uzyskuje się przy zastosowaniu strategii **Pierwsza pasująca** i **Najlepiej pasującą**. Z praktycznego punktu widzenia ta pierwsza jest lepsza ponieważ nie tworzy dużych narzutów czasowych. Więcej informacji na temat strategii przydziału można znaleźć w książce D.E.Knuth'a „Sztuka programowania” tom I.

## Fragmentacja

Zjawisko fragmentacji pamięci występuje powszechnie w systemach, które stosują dynamiczny przydział pamięci. Rozróżniamy dwa rodzaje tego zjawiska:

- **Fragmentacja zewnętrzna**-występuje wtedy, gdy w pamięci operacyjnej istnieje wiele obszarów wolnych, których sumaryczna wielkość pozwalałaby na spełnienie żądania przydziału pamięci przez proces, ale każdy z osobna z tych obszarów ma zbyt małą pojemność, żeby to było możliwe.
- **Fragmentacja wewnętrzna**-występuje wtedy, gdy procesowi przydzielane jest więcej pamięci, niż on potrzebuje. Ta dodatkowa pamięć nie będzie nigdy przez niego wykorzystana. Przydział taki może być podyktowany względami ekonomicznymi: nie jest opłacalne utrzymywanie 2 bajtowej dziury w pamięci, jeśli należy zapamiętać o wiele więcej informacji o niej.

Fragmentacja podlega regule 50%, tzn. po dużej liczbie cykli przydziałów i zwolnień będziemy w stanie przydzielić tylko połowę wolnej pamięci. Możemy zapobiegać temu zjawisku stosując *upakowanie* pamięci, tzn. okresowe przemieszczanie procesów, tak aby w pamięci powstał jeden ciągły obszar. System operacyjny powinien tak wykonywać tę operację, aby czas potrzebny na jej wykonanie był jak najmniejszy. Ta technika może być stosowana w systemach, w których możliwa jest relokacja. Jeśli oprócz procesów przemieszczalnych w pamięci są procesy z adresami bezwzględnymi to można tę technikę połączyć z prostą wymianą.

## Fragmentacja

Zjawisko fragmentacji pamięci występuje powszechnie w systemach, które stosują dynamiczny przydział pamięci. Rozróżniamy dwa rodzaje tego zjawiska:

- **Fragmentacja zewnętrzna**-występuje wtedy, gdy w pamięci operacyjnej istnieje wiele obszarów wolnych, których sumaryczna wielkość pozwalałaby na spełnienie żądania przydziału pamięci przez proces, ale każdy z osobna z tych obszarów ma zbyt małą pojemność, żeby to było możliwe.
- **Fragmentacja wewnętrzna**-występuje wtedy, gdy procesowi przydzielane jest więcej pamięci, niż on potrzebuje. Ta dodatkowa pamięć nie będzie nigdy przez niego wykorzystana. Przydział taki może być podyktowany względami ekonomicznymi: nie jest opłacalne utrzymywanie 2 bajtowej dziury w pamięci, jeśli należy zapamiętać o wiele więcej informacji o niej.

Fragmentacja podlega regule 50%, tzn. po dużej liczbie cykli przydziałów i zwolnień będziemy w stanie przydzielić tylko połowę wolnej pamięci. Możemy zapobiegać temu zjawisku stosując *upakowanie* pamięci, tzn. okresowe przemieszczanie procesów, tak aby w pamięci powstał jeden ciągły obszar. System operacyjny powinien tak wykonywać tę operację, aby czas potrzebny na jej wykonanie był jak najmniejszy. Ta technika może być stosowana w systemach, w których możliwa jest relokacja. Jeśli oprócz procesów przemieszczalnych w pamięci są procesy z adresami bezwzględnymi to można tę technikę połączyć z prostą wymianą.

## Fragmentacja

Zjawisko fragmentacji pamięci występuje powszechnie w systemach, które stosują dynamiczny przydział pamięci. Rozróżniamy dwa rodzaje tego zjawiska:

- **Fragmentacja zewnętrzna**-występuje wtedy, gdy w pamięci operacyjnej istnieje wiele obszarów wolnych, których sumaryczna wielkość pozwalałaby na spełnienie żądania przydziału pamięci przez proces, ale każdy z osobna z tych obszarów ma zbyt małą pojemność, żeby to było możliwe.
- **Fragmentacja wewnętrzna**-występuje wtedy, gdy procesowi przydzielane jest więcej pamięci, niż on potrzebuje. Ta dodatkowa pamięć nie będzie nigdy przez niego wykorzystana. Przydział taki może być podyktowany względami ekonomicznymi: nie jest opłacalne utrzymywanie 2 bajtowej dziury w pamięci, jeśli należy zapamiętać o wiele więcej informacji o niej.

Fragmentacja podlega regule 50%, tzn. po dużej liczbie cykli przydziałów i zwolnień będziemy w stanie przydzielić tylko połowę wolnej pamięci. Możemy zapobiegać temu zjawisku stosując *upakowanie* pamięci, tzn. okresowe przemieszczanie procesów, tak aby w pamięci powstał jeden ciągły obszar. System operacyjny powinien tak wykonywać tę operację, aby czas potrzebny na jej wykonanie był jak najmniejszy. Ta technika może być stosowana w systemach, w których możliwa jest relokacja. Jeśli oprócz procesów przemieszczalnych w pamięci są procesy z adresami bezwzględnymi to można tę technikę połączyć z prostą wymianą.

## Ochrona

Jeśli dopusczamy relokację procesów podczas wykonania, to poznany na drugim wykładzie schemat prostej ochrony pamięci należy trochę zmodyfikować. Otóż najpierw należy sprawdzić, czy adres logiczny wygenerowany przez program nie jest większy niż zawartość rejestru granicznego, a następnie, jeśli ten test się powiodł dodać do niego zawartość rejestru przemieszczenia. Celem zmniejszenia fragmentacji programy dzieli się na części. Najczęściej są to dwie części: zawierająca kod i zawierająca dane. Do ochrony tych części służą osobne pary rejestrów bazowych i granicznych. Jeśli chcemy wykonać podział programów na więcej części, to musimy mieć odpowiednio więcej par wymienionych rejestrów. Zastosowanie zwielokrotnionych rejestrów bazowych i granicznych pozwala również na określenie sposobu dostępu do danego fragmentu pamięci (np. tylko odczyt dla stałych) lub współdzielenie tych fragmentów przez kilka procesów (np. obszar kodu).

## Stronicowanie

**Stronicowanie** (ang. paging) jest systemem zarządzania pamięcią, który rozwiązuje problem zewnętrznej fragmentacji pamięci pozwalając, aby pamięć przydzielana była *nieciągła*. Jest ono również w wielu wypadkach punktem wyjścia do implementacji bardziej zaawansowanych i nowoczesnych schematów zarządzania pamięcią. W każdym przypadku stronicowanie wymaga wspomagania sprzętowego. Stronicowanie jest jedną z możliwych realizacji przydziału dynamicznego pamięci. Dodatkowo można je łączyć z prostą wymianą<sup>1</sup>.

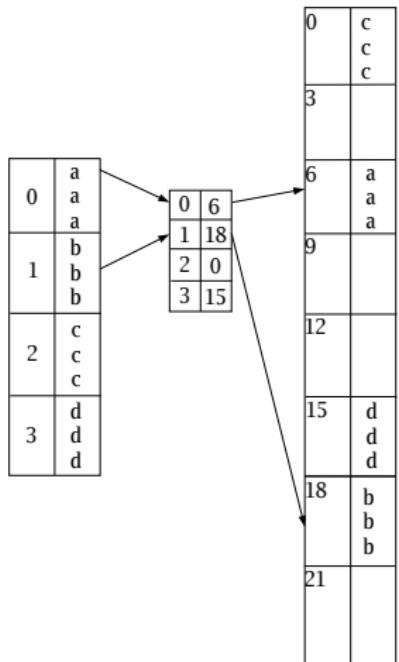
---

<sup>1</sup>Bardziej efektywnym rozwiązaniem jest stronicowanie na żądanie, które będzie omawiane na następnych wykładach

## Zasada działania

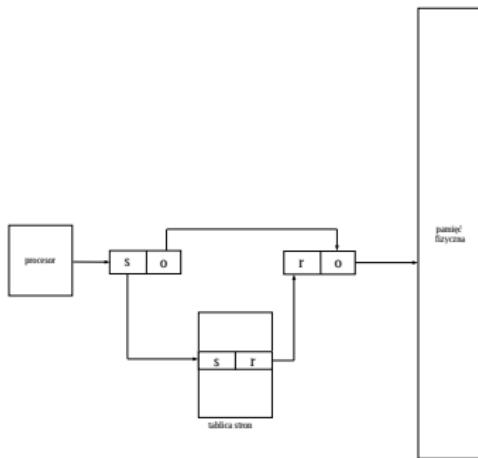
W stronicowaniu pamięć logiczna procesu jest podzielona na fragmenty o takiej samej wielkości, wyrażonej potęgą dwójki (zwykle od 512 do 2048 bajtów), nazywane *stronami*. Pamięć fizyczna jest podzielona na *ramki* nazywane również *stronami fizycznymi*, które mają ten sam rozmiar co strony. Również pamięć pomocnicza podzielona jest na obszary wielkości ramek nazywane *blokami*. Adres logiczny składa się z dwóch części: *numeru strony* oraz *przemieszczenia* względem początku strony. Podczas ładowania z pamięci pomocniczej do operacyjnej każda strona zostaje umieszczona w osobnej ramce. Rozmieszczenie to jest odwzorowywane w *tablicy stron*. Numery stron są traktowane jako indeksy tej tablicy, natomiast elementy zawierają adresy bazowe ramek, w których te strony zostały umieszczone. Translacja adresu logicznego na fizyczny wymaga odnalezienia w tablicy stron elementu o określonym przez numer strony indeksie, odczytaniu z niego adresu bazowego ramki i zastąpieniu nim numeru strony w adresie.

## Zasada działania



Aby proces mógł być załadowany do pamięci musi jedynie istnieć odpowiednia liczba wolnych ramek dla niego. Strony w ramkach nie muszą być umieszczane po kolej, fizycznie pamięć procesu nie musi być ciągła. Taka sytuacja jest przedstawiona na ilustracji obok. Po lewej stronie znajduje się pamięć logiczna, w środku tablica stron, a po prawej pamięć fizyczna. Stronicowanie eliminuje całkowicie fragmentację zewnętrzną, ale nie jest odporne na fragmentację wewnętrzną. Ta fragmentacja dotyczy ostatniej ramki przydzielonej procesowi i w skrajnych przypadkach może wynosić rozmiar strony minus jeden bajt.

## Translacja adresów



Aby stosować stronicowanie system musi być wyposażony w jednostkę MMU, która umożliwia translację adresów według przedstawionego obok schematu. Litera *s* oznacza numer strony, litera *r* adres bazowy ramki. Konieczność translacji adresu logicznego powoduje, że dostęp do pamięci odbywa się wolniej niż w systemach gdzie nie ma dynamicznego wiązania adresów.

## Tablica stron

Każdy proces ma swoją tablicę stron, która jest umieszczona w obszarze pamięci systemu operacyjnego. Procesor jest wyposażony w *rejestr bazowy tablicy stron*<sup>2</sup>, zawierający adres początku tablicy stron aktywnego procesu. Zawartość tego rejestru ulega oczywiście wymianie podczas przełączania procesów. Ponieważ każde odwołanie się przez proces użytkownika do pamięci operacyjnej pociąga za sobą konieczność translacji adresów, to programistom systemowym zależy, aby czas dostępu do tablicy stron był jak najkrótszy. Z pomocą przychodzą im projektanci sprzętu, umieszczając w procesorach *rejestry TLB* (ang. Translation Lookaside Buffers) nazywane w literaturze polskiej *rejestrami asocjacyjnymi*, a nawet *asocjacyjnymi buforami antycypacji translacji*. Jeśli tablica stron jest niewielkich rozmiarów, to można ją całkowicie w tych rejestrach umieścić, a ponieważ są one zbudowane z szybkich układów, to znalezienie adresu bazowego ramki na podstawie numeru strony jest wykonywane wielokrotnie szybciej niż przy bezpośrednim dostępie do pamięci. Jeśli tablica stron jest większa niż pojemność buforów, to przechowuje się w nich tylko te elementy tablicy stron, które są potrzebne. Jeśli jakiś element przestaje być potrzebny, to zostaje wymieniony na inny. Ta wymiana przebiega według następującego schematu: najpierw sprawdzane jest, czy rejestr asocjacyjny zawiera żądany numer strony, jeśli tak, to pobierany jest na jego podstawie adres bazowy ramki i następuje siegniecie do określonego fragmentu pamięci. Jeśli nie, to należy znaleźć w pamięci tablicę stron, odczytać z niej adres ramki (uaktualniając przy okazji rejesty TLB) i dopiero wtedy można sięgnąć do pamięci. Efektywny czas dostępu do pamięci zależy więc od *współczynnika trafień* (ang. hit ratio) i można go policzyć według wzoru:

$$t_{ema} = h \cdot (t_{TLBa} + t_{ma}) + (1 - h) \cdot (t_{TLBa} + 2 \cdot t_{ma}),$$

gdzie  $h$  jest współczynnikiem trafień, który jest wartością wyskalowaną od 0 do 1,  $t_{ema}$  jest efektywnym czasem dostępu do pamięci,  $t_{TLBa}$  czasem dostępu do rejestrów TLB, a  $t_{ma}$  czasem dostępu do pamięci. Współczesne platformy sprzętowe umożliwiają zarządzanie TLB z poziomu systemu operacyjnego.

<sup>2</sup>Uwaga: W opisie zakładamy, że MMU jest częścią procesora.

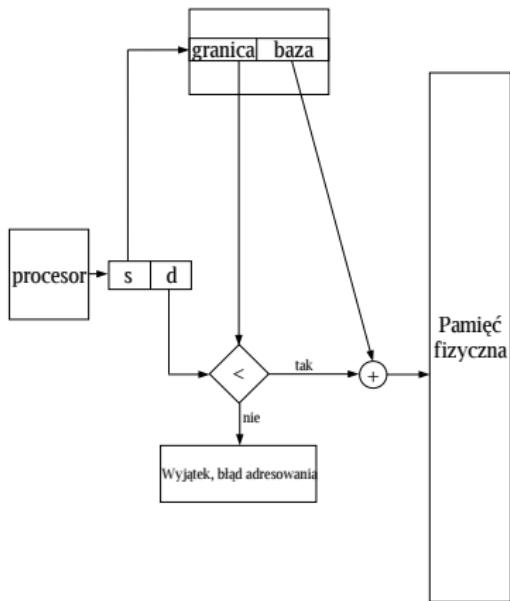
## Strony współdzielone i ochrona

W systemach stosujących stronicowanie można dopuścić możliwość współdzielenia stron przez wiele procesów. Jeśli mamy uruchomionych wiele instancji tego samego programu, to korzystnie jest im pozwolić współdzielić strony z kodem, o ile ten kod jest *kodem wznawialnym*, nazywanym również *współużywalnym* (ang. reentrant). Aby kod był wznawialny wymaga się aby nie podlegał modyfikacją podczas wykonania. Warunek ten jest spełniany w większości współczesnych systemów. Łatwo jest również zapewnić ochronę stron. Jeśli dany numer strony nie znajduje się w tablicy stron procesu, to oznacza to, że proces próbuje sięgnąć do nieistniejącej strony lub do strony, która do niego nie należy. Również łatwo jest sprawdzić, czy przemieszczenie w adresie logicznym jest prawidłowe: nie powinno ono przekraczać rozmiaru pojedynczej strony. Można również wprowadzić dodatkową ochronę. W tablicy stron przy każdej pozycji można umieścić trzy bity, które będą określały rodzaj dostępu do określonej strony: odczyt, zapis, wykonanie (w notacji uniksowej: rwx). Dzięki odpowiednim kombinacjom wartości tych bitów można uzyskać różne formy kontroli dostępu do stron, np. tylko odczyt i zapis. Naruszenie ochrony gwarantowanej przez te bity jest wykrywane przez sprzęt i obsługiwane na zasadzie wyjątku przez system operacyjny. Do ochrony samej tablicy stosowany jest niekiedy *rejestr długości tablicy stron*, który również określa, które elementy w tablicy są używane.

## Segmentacja

Technika stronicowania jest przezroczysta dla programistów piszących programy w językach wysokiego poziomu oraz w językach asemblerowych. Oznacza to, że programy te tworzone są w ten sam sposób jak dla komputerów, które są pozbawione możliwości stronicowania. Pamięć logiczna procesów jest dzielona na równe obszary, niezależnie od tego czy zawierają one dane, czy kod. W zarządzaniu pamięcią opartym na *segmentacji* (ang. segmentation) pamięć logiczna jest dzielona na obszary o różnej wielkości, które zawierają logicznie odrębne fragmenty kodu. W osobnym segmencie mogą być zamknięte rozkazy programu, w innym mogą być zamknięte dane, a w jeszcze innym stos. Tę segmentację można posunąć jeszcze dalej: w osobnych segmentach można zamykać np.: stałe programu lub poszczególne struktury danych. Segmentacja jest więc bliższa wyobrażeniu programisty na temat pamięci procesu, niż stronicowanie. Programista tworzący oprogramowanie w asemblerze może zazwyczaj określić do jakich segmentów zakwalifikować poszczególne fragmenty programu i ile tych segmentów będzie. W przypadku programisty piszącego w języku wysokiego poziomu ta czynność jest wykonywana automatycznie i niejawnie przez kompilator. Każdy segment ma swój unikalny numer. Podobnie jak w przypadku stronicowania do translacji adresów potrzebna jest tablica nazywana tutaj *tablicą segmentów*. Prosty schemat segmentacji stosuje część procesorów Intela, które wymuszają na programiście podział procesu na trzy segmenty: kodu, danych i stosu. Tablica segmentów jest w ich przypadku realizowana za pomocą rejestrów CS, DS i SS. Segmentacja jest pozbawiona fragmentacji wewnętrznej, ale obarczona jest fragmentacją zewnętrzną. Ponieważ rozmiary segmentów nie są na ogół duże, to nie jest ona tak dotkliwa jak w przypadku przydzielania obszarów ciągłych pamięci. Ponadto, ponieważ segmentacja podobnie jak stronicowanie jest formą dynamicznego wiążania adresów, to można do eliminacji fragmentacji wewnętrznej zastosować technikę upakowania.

## Translacja adresów



Każdy adres logiczny składa się z numeru segmentu  $s$  i z przesunięcia względem początku tego segmentu  $d$ . Translacja adresów odbywa się według schematu przedstawionego obok. Numer segmentu z adresu logicznego traktowany jest jako indeks w tablicy segmentów. Każdy element tej tablicy zawiera dwie wartości: wielkość segmentu (granica) oraz adres bazowy segmentu (baza). Jeśli przesunięcie w adresie logicznym jest większe niż wielkość segmentu, to generowany jest wyjątek adresowania. Jeżeli nie, to przesunięcie dodawane jest do adresu bazowego segmentu i wykonywane jest odwołanie do pamięci fizycznej. Translacja adresów w segmentacji wraz ze sprawdzeniem ich poprawności wykonywana jest sprzętowo.

## Tablica segmentacji

Podobnie jak w przypadku stronicowania dostęp do tablicy segmentów może zostać przyspieszony za pomocą rejestrów asocjacyjnych. Ponieważ każdy proces posiada własną tablicę segmentów, to konieczny jest również *rejestr bazowy tablicy segmentów* oraz *rejestr długości tablicy segmentów*. Pełnią one tę samą rolę co analogiczne rejesty w stronicowaniu.

## Ochrona i współdzielenie segmentów

Kontrola poprawności przesunięcia w adresie logicznym została już opisana. Numer segmentu również podlega sprawdzeniu. Jeśli nie istnieje element tablicy wskazywany przez ten numer, to oznacza, że proces odwołuje się do nieistniejącego segmentu lub do segmentu, który nie należy do niego. Taki proces należy zakończyć. Podobnie jak w przypadku stronicowania w tablicy segmentów można umieścić dodatkowe informacje o ochronie poszczególnych segmentów. Mogą to być informacje o trybie dostępu do zawartości tych segmentów (odczyt, zapis, wykonanie). Segmente mogą być współdzielone, ale należy zachować ostrożność pozwalając na użytkowanie jednego segmentu przez wiele procesów. Jeśli w tym segmencie są umieszczone odwołania do innych segmentów, to powinny mieć te segmenty takie same numery dla wszystkich procesów. Schemat ten się komplikuje, jeśli procesor posiada pośredni tryb adresowania pozwalający na wielokrotne odwołania do różnych segmentów.

## Segmentacja stronicowana

Oba opisane wcześniej mechanizmy zarządzania pamięcią można połączyć (oba są również realizacjami koncepcji dynamicznego przydziału pamięci). Z tego połączenia powstaje *segmentacja stronicowana*. Z jednej strony posiada ona tę zaletę stronicowania, że nie jest obarczona fragmentacją zewnętrzną, z drugiej strony pozwala lepiej dopasować się do logicznej struktury kodu procesu co jest zaletą segmentacji. To rozwiązanie stosował system Multics. Adres logiczny w segmentacji stronicowanej ma tę samą strukturę, co w zwykłej segmentacji. Z tablicy segmentów nie jest jednak odczytywany adres bazowy segmentu, ale adres początku tablicy stron tego segmentu. Przesunięcie w segmentie podzielone jest na dwie części: numer strony oraz przesunięcie na tej stronie. Translacja adresu w segmentacji stronicowanej jest więc skomplikowaną czynnością. Dodatkowo utrudnia ją fakt, że tablica segmentów, ze względu na duże rozmiary jest również stronicowana. Ten system zarządzania pamięcią ze względu na swój stopień skomplikowania jest rzadko stosowany.

## Pytania

?

Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne — Pamięć wirtualna cz. 1

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 6 grudnia 2020

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

# Plan wykładu

## ① Wprowadzenie

- ① Zasady lokalności czasowej i przestrzennej
- ② Pamięć wirtualna
- ③ Sposoby implementacji pamięci wirtualnej

## ② Stronicowanie na żądanie

- ① Podstawy
- ② Wymiana stron
- ③ Efektywność

## ③ Algorytmy wymiany stron

- ① Algorytm FIFO
- ② Algorytm OPT
- ③ Algorytm LRU
- ④ Algorytmy zbliżone do LRU
- ⑤ Algorytmy ad hoc

## Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

## Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

## Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

## Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

## Zasady lokalności

Uważna analiza działania procesu ujawnia, że podlega on dwóm *zasadom lokalności*.

### Zasada lokalności przestrzennej

Jeśli nastąpiło odwołanie procesu do pewnej lokacji w pamięci komputera, to następne odwołania z dużym prawdopodobieństwem będą dotyczyć lokacji położonej w pobliżu tej do której nastąpiło to odwołanie.

### Zasada lokalności czasowej (temporalnej)

Jeśli nastąpiło odwołanie procesu do określonej lokacji w pamięci komputera, to w niedalekiej przyszłości to odwołanie zostanie wielokrotnie powtórzone.

Jeśli system zarządzania pamięcią uwzględnia te dwie zasady, to w pamięci operacyjnej komputera pozostają tylko informacje, które w danej chwili są niezbędne do działania procesu.

## Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

## Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

## Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

## Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

## Implementacja pamięci wirtualnej

Najprostszym, najpowszechniejszym stosowanym, a jednocześnie najskuteczniejszym sposobem implementacji pamięci wirtualnej jest *stronicowanie na żądanie* (ang. demand paging). Z poznanych na poprzednim wykładzie technik zarządzania pamięcią do implementacji pamięci wirtualnej nadają się jeszcze segmentacja i segmentacja stronicowana, ale ze względu na małą popularność takich implementacji ich użycie będzie omówione tylko pobicieżnie w następnej części wykładu.

# Podstawy

Aby stronicowanie mogło służyć do implementacji pamięci wirtualnej, należy je przekształcić w stronicowanie na żądanie. Zasadnicza różnica między tymi technikami polega na sposobie ładowania procesu do pamięci. W stronicowaniu, zanim proces mógł się rozpoczęć, wszystkie jego strony musiały być załadowane do pamięci operacyjnej. Stronicowanie na żądanie stosuje technikę *leniwej wymiany* (ang. lazy swapper), tzn. wprowadza do pamięci operacyjnej stronę, dopiero wtedy, gdy nastąpi do niej odwołanie. Dzięki temu proces zajmuje niewielką ilość pamięci, jest tylko częściowo załadowany, ale jak wynika to z reguł lokalności czasowej i przestrzennej może się wykonywać. Ta technika wymaga wzbogacenia każdej pozycji w tablicy stron o dodatkowy bit, nazywany *bitem poprawności odniesienia*, który sygnalizuje, czy strona do której odwołuje się proces jest załadowana do pamięci operacyjnej. Jeśli nastąpi odwołanie do strony, której nie ma w pamięci fizycznej, to generowany jest wyjątek nazywany *błędem strony* (ang. page fault). Przyczyny tego błędu mogą być dwie: proces odwołuje się do nieistniejącej strony i powinien zostać zakończony lub proces odwołuje się do strony, która istnieje, ale zamiast w pamięci operacyjnej znajduje się na dysku i powinna być do niej sprowadzona. Należy zauważyć, że proces może zacząć się wykonywać, mając wyłącznie puste ramki, bez żadnej strony. To wykonanie rozpoczęcie się oczywiście błędem strony, którego obsługa spowoduje wprowadzenie do pamięci potrzebnej strony. Ta technika nazywa się *czystym stronicowaniem na żądanie*. Stronicowanie na żądanie wymaga, aby w pamięci masowej (najczęściej na dysku) został wydzielony obszar nazywany *obszarem wymiany* lub *przestrzenią wymiany*. Jest to plik lub partycja, które są zoptymalizowane pod względem przechowywania stron procesu i nazywane odpowiednio *plikiem wymiany* lub *partycją wymiany*. Urządzenie pamięci masowej, na którym osadzone są takie struktury nazywa się *urządzeniem stronicującym* lub *pamięcią pomocniczą*.

## Obsługa błędu strony

Błąd strony jest obsługiwany przez odpowiednią procedurę obsługi przerwania (wyjątku). Jednakże inne elementy systemu operacyjnego także wykonują dodatkowe czynności związane z obsługą tego błędu:

- ① zachowanie stanu bieżącego procesu,
- ② sprawdzenie poprawności adresu, który wygenerował wyjątek (jeśli adres był nieprawidłowy to kończone jest wykonanie procesu),
- ③ rozpoczęcie wykonania operacji wejścia-wyjścia, której celem jest załadowanie odpowiedniej strony do wolnej ramki w pamięci operacyjnej,
- ④ (nieobowiązkowo) przydzielenie procesora innemu procesowi na czas oczekiwania na zrealizowanie transmisji,
- ⑤ obsługa przerwania od dysku twardego, sygnalizującego zakończenie operacji sprowadzania strony do pamięci,
- ⑥ uaktualnienie tablicy stron,
- ⑦ oczekивание на przydzielenie procesowi dla którego została sprowadzona strona procesora,
- ⑧ wykonanie przerwanego przez błąd strony rozkazu.

W zależności od sposobu implementacji stronicowania na żądanie lista tych czynności może być uzupełniona o inne działania.

## Wymiana stron

Każdy proces otrzymuje określoną liczbę ramek w pamięci fizycznej. Dopóki są wolne ramki, dopóty można sprowadzać nowe strony do pamięci operacyjnej. Co jednak stanie się, kiedy wolnych ramek zabraknie? Jednym z rozwiązań jest na pewno zawieszenie wykonania procesu, który nie ma wolnej ramki do której można by załadować żądaną przez niego stronę. Istnieje jednak inne, korzystniejsze i częściej stosowane rozwiązanie. Jest nim *wymiana stron*. Polega ona na odnalezieniu strony, co do której istnieje podejrzenie, że nie będzie już używana, albo nie będzie używana w najbliższym czasie, wysłaniu jej do przestrzeni wymiany i sprowadzeniu w jej miejsce żądanej strony. Należy więc uzupełnić scenariusz obsługi błędu strony o następujące czynności:

- ① Jeśli nie istnieje wolna ramka, należy wytypować ramkę-ofiarę.
- ② Strona-ofiara jest zapisywana na dysku i aktualizowana jest tablica stron.
- ③ Do zwolnionej ramki wczytywana jest żądana strona.

Do wytypowania ramki-ofiary należy zastosować możliwie najbardziej skuteczny algorytm, który będzie działał szybko i w miarę precyzyjnie typował strony, które będą nieużywane. Algorytmy wymiany stron zostaną zaprezentowane w dalszej części wykłdu.

## Efektywność stronicowania na żądanie

Ponieważ obsługa wystąpienia błędy strony trwa stosunkowo długo, można się zastanowić jak wpływa ona na czas działania procesu, a w szczególności jak wpływa na czas dostępu do pamięci. Efektywny czas dostępu do pamięci w stronicowaniu na żądanie możemy wyrazić następującym wzorem:

$$t_{ema} = (1 - p) \cdot t_{ma} + p \cdot t_{pfh},$$

gdzie  $p$ -prawdopodobieństwo wystąpienia błędu,  $t_{ma}$  czas dostępu do pamięci,  $t_{pfh}$  czas obsługi błędu strony. Na czas obsługi błędu strony składają się czasy wykonania wszystkich czynności wymienionych na planszy pt. „Obsługa błędu strony”. Ogólnie czas ten jest długi, dlatego też dąży się nie tylko do jego skrócenia, ale również do zminimalizowania liczby błędów strony. Na tę ostatnią wielkość mają wpływ takie czynniki, jak liczba ramek, które zostały procesowi przydzielone i sprawność algorytmu wymiany. Problemem przydziału ramek zajmiemy się w drugiej części wykładu, teraz warto jednak wspomnieć, że aby proces w ogóle mógł się wykonać, musi w pamięci rezydować zawsze tyle ramek ile wymaga najdłuższy rozkaz procesora. Istnieją dodatkowe również wymagania nakładane na listę rozkazów i działanie procesora. Ponieważ błąd strony jest przerwaniem nieprecyzyjnym, to należy ponowić wykonanie rozkazu, który on przewał. Może to być trudne, jeśli rozkaz stosuje tryb adresowania z preautoinkrementacją lub preautodekrementacją. Również wielokrotny tryb pośredni nie jest wskazany, gdyż wymaga obecności w pamięci dużej liczby stron. Generalnie rozkazy, które bezpośrednio modyfikują zawartość w pamięci operacyjnej są problematyczne w stronicowaniu na żądanie.

## Efektywność stronicowania na żądanie

Opisując efektywność stronicowania na żądanie należy wspomnieć też o fragmentacji zewnętrznej, która występuje na poziomie nie poszczególnych lokacji pamięci, ale całych stron. Nie ma ona znaczenia dla aplikacji użytkownika, które posługują się adresami wirtualnymi (tak w przypadku pamięci wirtualnej nazywa się adresy logiczne). Stanowi ona za to problem dla urządzeń korzystających z transmisji DMA, ponieważ one posługują się wyłącznie adresami fizycznymi i pamięć na bufory dla nich musi być przydzielana w sposób ciągły. Nowsze systemy komputerowe wyposażone są w jednostkę IOMMU, która przeznaczona jest dla urządzeń wejścia-wyjścia i eliminuje konieczność przydzielania pamięci operacyjnej na bufory w sposób ciągły.

## Algorytmy wymiany-wprowadzenie

Wybierając algorytm wymiany stron zależy nam na tym, aby generował on jak najmniejszą liczbę błędów stron. Implementację takiego algortymu można przetestować tworząc za pomocą generatora liczb pseudolosowych *ciąg odwołań* (ang. reference string), czyli ciąg numerów stron, do których hipotetyczny proces mógłby się odwoływać. Należy również założyć pewną liczbę wolnych ramek, którymi będzie dysponował ten proces. Najczęściej działanie algorytmu bada się dla kilku różnych wartości tego czynnika, co pozwala sprawdzić, czy algorytm zachowuje się poprawnie, tzn. czy wraz ze wzrostem liczby ramek maleje liczba błędów stron.

## Algorytm FIFO

Algorytm FIFO jest najprostszym algorytmem wymiany stron, zawsze zastępuje tę stronę, która przebywa najdłużej w pamięci operacyjnej. Niestety takie działanie nie gwarantuje, że strona wymieniana nie będzie w najbliższym czasie potrzebna, dlatego algorytm FIFO generuje dużą liczbę błędów stron, a dodatkowo obciążony jest anomalią *Belady'ego*, tzn. wraz ze wzrostem liczby ramek może wzrastać liczba błędów stron. Działanie tego algorytmu jest przedstawione na następnej planszy. Górný wiesz tabeli, to ciąg odwołań, a trzy kolejne symbolizują ramki. Wystąpienie błędu strony sygnalizowane jest żółtym tłem kolumny tabeli.

## Algorytm FIFO

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
	1	1	1	1	1	0	0	0	3	3	3	3	2	2	2	2	2	2	1

Razem: 15 błędów stron

## Algorytm optymalny

Dla zagadnienia wymiany stron istnieje algorytm optymalny, czyli taki, który powoduje najmniejszą liczbę wymian stron, a zarazem najmniejszą liczbę błędów stron. Ten algorytm oznaczany jest skrótem OPT lub MIN. Jego działanie można scharakteryzować jednym zdaniem: „Wymień tę stronę, która najdłużej nie będzie używana”. W praktyce jednak nigdy tego algorytmu się nie stosuje, ponieważ nie można go zaimplementować. Nie istnieje żaden stuprocentowo pewny sposób na określenie, która ze stron będzie najdłużej niepotrzebna. Możemy jednak porównywać rzeczywiste algorytmy z algorytmem OPT i badać w jakiej skali go przybliżają. Działanie tego algorytmu zostanie zaprezentowane na następnej planszy.

## Algorytm optymalny

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1

Razem: 9 błędów stron

## Algorytm LRU

Algorytm LRU (ang. Least Recently Used) wymienia tą stronę, która najdawniej była używana. Po analizie działania tego algorytmu można stwierdzić, że stanowi on w pewnym sensie odwrotność działania algorytmu OPT. Używając języka potocznego, można napisać, że algorytm OPT „patrzy w przyszłość”, żeby znaleźć stronę do wymiany, a algorytm LRU „patrzy w przeszłość”. Algorytm LRU jest najpopularniejszym algorytmem stosowanym do wymiany stron. Efektywność jego działania jest zbliżona do efektywności algorytmu OPT. Następna plansza zawiera ilustrację działania tego algorytmu.

## Algorytm LRU

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
	1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	2	7	7	7

Razem: 12 błędów stron

## Implementacja LRU

Implementacja algorytmu LRU jest dosyć trudna i może wymagać wsparcia ze strony sprzętu. Aby określić, która ze stron była najdawniej używana stosuje się liczniki, albo stos. Zastosowanie liczników polega na wyposażeniu każdej pozycji tablicy stron w miejsce na wartość zegara logicznego. Wartość tego zegara jest zwiększana przy każdym odwołaniu do strony i jednocześnie zapisywana w miejscu tablicy stron, które jest związane ze stroną do której nastąpiło to odwołanie. Porównując te wartości dla każdej strony możemy sprawdzić, do której odwoływano się ostatnio. Problemem może być powstanie nadmiaru w zegarze logicznym. Stos zawiera numery stron do których się odwoływał proces ułożone na nim w ten sposób, że numer strony, która została ostatnio użyta jest na szczytce tego stosu, a numer strony, która najdawniej była nieużywana jest na jego dnie. Liczba elementów na stosie jest równa liczbie ramek. Ten stos jest najczęściej implementowany w postaci listy dwukierunkowej, której obsługa jest wspomagana sprzętowo. Zastosowanie stosu w implementacji algorytmu LRU pozwoliło określić klasę algorytmów nazywanych *algorytmami stosowymi*, które nie prowadzą do anomalii Belady'ego. Algorytm stosowy to taki algorytm dla którego zbiór stron obecnych w pamięci przy  $n$  dostępnych ramkach jest podzbiorem zbioru stron obecnych w pamięci, gdyby było dostępnych  $n+1$  ramek.

## Algorytmy zbliżone do LRU

Jeśli w systemie nie ma odpowiednich środków sprzętowych do realizacji algorytmu LRU, to można zastosować metodę, która będzie dawała rezultaty zbliżone do rezultatów tego algorytmu. Na następnych planszach zostanie opisanych kilka takich algorytmów.

## Algorytm dodatkowych bitów odniesienia

Wiele systemów stosuje *bity odniesienia*, które są ustawiane dla stron do których następowało odwołanie. Zamiast pojedynczego bitu można zastosować w tablicy stron cały bajt. Co określony czas system operacyjny modyfikuje wartości tych bajtów, przesuwając ich zawartość o jedno miejsce w prawo i ustawiając na najstarszym bicie jedynkę, jeśli ostatnio nastąpiło odniesienie do strony lub zero jeśli tego odniesienia nie było. Interpretując powstały w ten sposób wzorzec binarny jako liczbę naturalną można określić kolejność odwołań do stron.

## Algorytm drugiej szansy

Mając tylko jeden bit odniesienia można zastosować algorytm drugiej szansy, nazywany również algorytmem zegarowym. W tym algorytmie przeszukiwana jest tablica stron w poszukiwaniu strony, która ma wyzerowany bit odniesienia. Jeśli strona ma bit odniesienia ustawiony na jeden, to algorytm go ustawia na zero, ale nie wymienia tej strony dając jej drugą szansę na pozostanie w pamięci operacyjnej. Jeśli jednak przy kolejnym wykonaniu tego algorytmu bit odniesienia tej strony będzie równy zero, to strona ta zostanie wymieniona.

## Algorytmy LFU i MFU

Algorytm LFU (ang. Least Frequently Used) wymienia te strony, do których najrzadziej się odwoływano. Aby określić częstotliwość odwoływania się do konkretnej strony, z każdym elementem tablicy stron związany jest licznik odwołań. Odwrotnie działa algorytm MFU (ang. Most Frequently Used) - wymienia on tę stronę, do której najczęściej się odwoływano, wychodząc z założenia, że nie będzie już potrzebna. Oba algorytmy są rzadko stosowane, bo nie przybliżają dobrze algorytmu OPT.

## Bit odniesienia i bit modyfikacji

Jeśli tablica stron wyposażona jest w *bit modyfikacji* sygnalizujący, czy do danej strony został wykonany zapis, to można w połączeniu z bitem odniesienia wykorzystać go do określenia przydatności strony do wymiany. Możemy wyróżnić cztery klasy stron, w zależności od ustawienia tych bitów (pierwszy jest bitem odniesienia, drugi modyfikacji):

- **(0,0)** - strona nie była używana ostatnio i nie jest zmieniona, idealna kandydatka do wymiany, nie trzeba jej nawet zapisywać do przestrzeni wymiany,
- **(0,1)** - strona nie była używana ostatnio, a więc może być wymieniona, ale trzeba ją zapisać do pamięci pomocniczej, bo jej stan uległ zmianie,
- **(1,0)** - strona używana, ale nie zmieniona, może być potrzebna, ale ewentualna jej wymiana nie wymagałaby zapisu jej zawartości na dysk,
- **(1,1)** - strona używana i zmieniona, najgorsza kandydatka do wymiany

Wymianę stron zaczyna się od tych, które należą do pierwszej klasy, jeśli nie ma takowych, to brane są pod uwagę strony z następnych klas.

## Algorytmy ad hoc

Dosyć często opisane wcześniej algorytmy są „wzbogacane” o dodatkowe elementy usprawniające ich działanie. Takim elementem może być stała pula wolnych ramek. Kiedy trzeba wymienić stronę, to typowana jest ramka-ofiara, ale stronę umieszcza się w pierwszej wolnej ramce i wznawia się wykonanie procesu. Kiedy urządzenie stronicujące nie ma innych zleceń, to zapisuje stronę z ramki-ofiary, a ramka trafia do puli ramek wolnych. Inne rozwiązanie polega na utrzymywaniu listy stron zmodyfikowanych i sukcesywnym ich zapisywaniu do przestrzeni wymiany, jeśli urządzenie stronicujące nie ma innych zleceń. Jeszcze inne rozwiązanie polega na utrzymywaniu puli wolnych ramek, wraz z informacją, które strony zawierały te ramki. Ponieważ do czasu modyfikacji, zawartość żadnej z tych ramek nie ulegnie zmianie, to jeśli nastąpi odwołanie do strony, która w takiej ramce się znajdowała, będzie łatwo tę stronę „odzyskać”.

## Pytania

?

**Koniec**

Dziękuję Państwu za uwagę!

## Systemy Operacyjne — Pamięć wirtualna cz. 2

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 12 grudnia 2020

# Plan wykładu

## ① Przydział ramek

- Wstęp
- Minimalna liczba ramek
- Algorytmy przydziału ramek

## ② Szamotanie

- Definicja zjawiska i jego przyczyna
- Model zbioru roboczego
- Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- Przydział globalny i lokalny
- Stronicowanie wstępne
- Wielkość strony
- Wpływ stronicowania na żądanie na wykonanie programu
- Blokowanie stron
- Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- Segmentacja na żądanie
- Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ④ Definicja zjawiska i jego przyczyna
- ⑤ Model zbioru roboczego
- ⑥ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ⑦ Przydział globalny i lokalny
- ⑧ Stronicowanie wstępne
- ⑨ Wielkość strony
- ⑩ Wpływ stronicowania na żądanie na wykonanie programu
- ⑪ Blokowanie stron
- ⑫ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ⑬ Segmentacja na żądanie
- ⑭ Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ④ Przydział globalny i lokalny
- ⑤ Stronicowanie wstępne
- ⑥ Wielkość strony
- ⑦ Wpływ stronicowania na żądanie na wykonanie programu
- ⑧ Blokowanie stron
- ⑨ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ⑩ Segmentacja na żądanie
- ⑪ Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ④ Przydział globalny i lokalny
- ⑤ Stronicowanie wstępne
- ⑥ Wielkość strony
- ⑦ Wpływ stronicowania na żądanie na wykonanie programu
- ⑧ Blokowanie stron
- ⑨ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ⑩ Segmentacja na żądanie
- ⑪ Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ④ Przydział globalny i lokalny
- ⑤ Stronicowanie wstępne
- ⑥ Wielkość strony
- ⑦ Wpływ stronicowania na żądanie na wykonanie programu
- ⑧ Blokowanie stron
- ⑨ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ⑩ Segmentacja na żądanie
- ⑪ Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ④ Przydział globalny i lokalny
- ⑤ Stronicowanie wstępne
- ⑥ Wielkość strony
- ⑦ Wpływ stronicowania na żądanie na wykonanie programu
- ⑧ Blokowanie stron
- ⑨ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ⑩ Segmentacja na żądanie
- ⑪ Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Plan wykładu

## ① Przydział ramek

- ① Wstęp
- ② Minimalna liczba ramek
- ③ Algorytmy przydziału ramek

## ② Szamotanie

- ① Definicja zjawiska i jego przyczyna
- ② Model zbioru roboczego
- ③ Częstotliwość błędów strony

## ③ Inne zagadnienia dotyczące stronicowania na żądanie

- ① Przydział globalny i lokalny
- ② Stronicowanie wstępne
- ③ Wielkość strony
- ④ Wpływ stronicowania na żądanie na wykonanie programu
- ⑤ Blokowanie stron
- ⑥ Implementacja tablic stron

## ④ Inne metody realizacji pamięci wirtualnej

- ① Segmentacja na żądanie
- ② Segmentacja stronicowana na żądanie

# Wstęp

W poprzedniej części wykładu ustaliliśmy, że wpływ na efektywność stronicowania na żądanie, oprócz algorytmu wymiany stron, może mieć również metoda przydziału wolnych ramek procesom. Kiedy w systemie pracują tylko dwa procesy, system operacyjny i proces użytkownika, to przydział ramek jest prosty. Podział zbioru wolnych ramek jest dokonywany tak, aby proporcja była korzystna dla procesu użytkownika (w końcu systemy komputerowe są tworzone po to by wykonywać procesy użytkownika, a system operacyjny ma tylko nimi zarządzać i ułatwiać ich pracę). Jeśli się skończy pula wolnych ramek, to strony procesu użytkownika lub systemu operacyjnego podlegają wymianie. Możliwe są również scenariusze, w których system przekazuje część swoich ramek na rzecz procesu użytkownika (np. ramek przeznaczonych na bufory, które aktualnie nie są używane) lub w których system utrzymuje zawsze pewną liczbę wolnych ramek, aby usprawnić proces wymiany stron. Problem przydziału ramek komplikuje się, gdy mamy do czynienia z systemem wielozadaniowym.

## Minimalna liczba ramek

Aby proces mógł wykonać choć jeden rozkaz, to w pamięci komputera muszą się znajdować jednocześnie wszystkie strony, których ten rozkaz może dotyczyć. Ponieważ nie można określić jaki to będzie rozkaz przed załadowaniem programu do pamięci, to system operacyjny zakłada najgorszy scenariusz i przydziela tyle ramek, aby mogły się w nich zmieścić wszystkie strony konieczne do wykonania najbardziej złożonego rozkazu na liście rozkazów procesora. Przez złożoność rozumiemy tu złożoność jego trybu adresowania. Najbardziej korzystnie pod tym względem wypadają procesory RISC, w których istnieją zazwyczaj tylko proste tryby adresowania, najgorzej procesory, które pozwalają na wielopoziomowe adresowanie pośrednie (należy ograniczyć liczbę poziomów do pewnej liczby, po której przekroczeniu generowany jest wyjątek).

## Algorytmy przydziału ramek

W systemach wielozadaniowych możliwych jest wiele strategii przydziału programom użytkownika wolnych ramek. Najprostszy jest *przydział równy* w którym każdemu procesowi przydziela się taką samą liczbę ramek. Liczba ta jest określona wzorem  $m/n$ , gdzie  $m$  jest całkowitą liczbą wolnych ramek, a  $n$  jest liczbą procesów. Ponieważ każdy z nich ma inne zapotrzebowanie na pamięć, taki przydział nie jest optymalny. Bardziej sprawiedliwy jest *przydział proporcjonalny*, który uwzględnia rozmiar każdego z procesów. Założymy, że  $s_i$  jest wielkością pamięci wirtualnej procesu  $p_i$ . Sumaryczny rozmiar pamięci wirtualnej wszystkich procesów jest zatem równy  $S = \sum_i s_i$ . Liczba ramek przydzielona programowi  $p_i$  jest określona wzorem  $a_i \approx \frac{s_i}{S} \cdot m$ . Ta wartość musi zostać zaokrąglona do najbliższej wartości całkowitej, większej od minimalnej liczby ramek, jaką należy procesowi przydzielić. Metody wywodzące się z przydziału proporcjonalnego oprócz rozmiaru procesu mogą uwzględniać też inne jego właściwości, np. priorytet.

## Szamotanie - definicja i przyczyna

W systemie, w którym procesom są przyporządkowane priorytety można pozwolić, aby proces o wysokim priorytecie odbierał w razie potrzeby ramki przydzielone procesom o niższym priorytecie. Przy takim rozwiążaniu, jeśli liczba ramek, którą dysponuje proces niskopriorytetowy spadnie poniżej określonej wartości, to proces ten zaczyna intensywnie wymieniać swoje strony. Jeśli proces zużywa więcej czasu na wymianę stron niż na wykonanie to mówimy o zjawisku *szamotania* (ang. trashing). Szczególnie intensywnie to zjawisko przebiegało w pierwszych systemach, które stosowały równocześnie planowanie długoterminowe i stronicowanie na żądanie. Szamotanie choć jednego z procesów powoduje zwiększenie obciążenia urządzenia wymiany, a zmniejszenie obciążenia procesora. Ponieważ zadaniem planisty długoterminowego jest dbanie o to, aby procesor był maksymalnie wykorzystany, to żeby zwiększyć jego obciążanie wprowadzał on do pamięci nowy proces. Ten nowy proces otrzymywał do dyspozycji ramki, które zabierane były procesom już rezydującym w pamięci. To z kolei powodowało zwiększenie intensywności szamotania procesów, które obejmowało do tej pory to zjawisko i doprowadzenie do niego innych procesów. W efekcie planista długoterminowy uzyskiwał efekt odwrotny do zamierzonego: małe obciążenie procesora, a duże jednostki stronicującej.

## Zbiór roboczy

Aby wyeliminować szamotanie należy zadbać o to, by proces zawsze dysponował wystarczającą liczbą ramek, aby pomieścić jednocześnie w pamięci wszystkie strony niezbędne w danej chwili do jego działania. Wykonanie procesu podlega *modelowi strefowemu*. *Strefą* nazywamy zbiór stron, które pozostają we wspólnym użyciu (muszą razem być obecne w pamięci operacyjnej). Taką strefę mogą tworzyć np. strony na których umieszczony jest jakiś podprogram lub jakaś struktura danych. Program składa się wielu stref, niekoniecznie rozłącznych. Podczas wykonania procesu sterowanie przechodzi od bieżącej strefy do następnej. Model zbioru roboczego jest ściśle powiązany z modelem stref. Przyjmuje się pewien parametr  $\Delta$ , który określa szerokość *okna zbioru roboczego*, czyli liczbę ostatnich odniesień procesu do stron, które powinien system operacyjny zapamiętać. *Zbiorem roboczym* (ang. working set) nazywamy wszystkie strony do których nastąpiło ostatnich  $\Delta$  odniesień. Należy zadbać o to by każdy proces miał tyle ramek, aby mógł utrzymać w pamięci swój zbiór roboczy. Problemem jest dobranie takiego parametru  $\Delta$  aby model zbioru roboczego działał. Do szamotania nie dojdzie, jeśli rozmiar sumy zbiorów roboczych  $WSS_i$  wszystkich procesów będzie zawsze mniejszy od liczby dostępnych ramek  $D$ , co można zapisać nierównością  $D > \sum_i WSS_i$ .

## Częstotliwość błędów strony

Prostszym sposobem niedopuszczania do zaistnienia szamotania jest monitorowanie częstotliwości występowania błędów stron we wszystkich procesach. Zakładamy przy tym pewną maksymalną i minimalną wartość graniczną. Jeśli któryś z procesów wykazuje za mało błędów stron, to część ramek jest mu odbierana i przekazywana procesowi, który wykazuje zbyt dużą liczbę błędów stron.

## Przydział lokalny i globalny

Istnieją dwie strategie przeprowadzania wymiany strony dla pojedynczego procesu wykonywanego w systemie wielozadaniowym. Pierwsza, nazywana *wymianą globalną*, polega na objęciu algorytmem wymiany wszystkich ramek w pamięci fizycznej. Oznacza to, że proces (a właściwie system operacyjny w imieniu procesu) może odebrać ramkę innemu procesowi i umieścić w niej swoją stronę. Druga strategia jest nazywana *wymianą lokalną*. W tym przypadku system operacyjny wymienia strony należące jedynie do procesu, który wykazał błąd strony. Dodatkowo, liczba ramek przydzielonych procesowi nie ulega zmianie, jeśli w systemie nie ma wolnych ramek. Wymiana lokalna ogranicza nasilanie zjawiska szamotania, ale z kolei wymiana globalna daje lepszą przepustowość systemu i pozwala dostosować liczbę ramek procesu do jego zbioru roboczego.

## Stronicowanie wstępne

Jeśli proces ulega szamotaniu, to może być całkowicie wycofany z pamięci operacyjnej do przestrzeni wymiany, do momentu aż w pamięci głównej pojawi się odpowiednia liczba ramek pozwalająca na jego prawidłowe wykonanie. Aby program nie był sprowadzany do pamięci strona po stronie, tak jak ma to miejsce na początku jego wykonywania, to można zastosować *stronicowanie wstępne*, czyli wprowadzić do pamięci wszystkie jego strony, które zostały wycofane. Skuteczność tej techniki zależy od tego ile z tych stron będzie przydatnych podczas dalszej pracy procesu. Jeśli zbyt mało, to nie opłaca się tej techniki stosować.

## Rozmiar strony

Chociaż w większości platform sprzętowych wpływ programistów systemowych na wielkość strony jest ograniczony do wyboru między kilkoma opcjami zaoferowanymi przez twórcę procesora (najczęściej są one dwie), to warto rozważyć zalety stosowania małych i dużych stron. Za stosowaniem małych stron przemawiają następujące czynniki:

- mniejsza fragmentacja wewnętrzna,
- mniejszy czas przesyłania strony z pamięci pomocniczej do operacyjnej lub odwrotnie,
- lepsza *rozdzielcość* tzn. stosunek ilości informacji potrzebnej w danej chwili do ilości informacji niepotrzebnej, zawartych w obrębie strony.

Za stosowaniem stron dużych przemawiają z kolei:

- mniejsza liczba operacji wejścia-wyjścia wykonywanych podczas działania procesu,
- mniejszy rozmiar tablicy stron,
- mniejsza liczba błędów stron.

Obecnie stosuje się najczęściej duże strony. Procesory firmy Intel i pokrewne (32-bitowe i 64-bitowe) pozwalają na wybór między 4KiB, a 4MiB. Procesory DEC Alpha (64-bitowe) stosują strony o wielkości 8KiB. Procesor Motorola 68030 (32-bitowy) pozwala programiście systemowemu na wybór rozmiaru strony w zakresie od 256 B do 32 KiB.

## Rozmiar strony

Chociaż w większości platform sprzętowych wpływ programistów systemowych na wielkość strony jest ograniczony do wyboru między kilkoma opcjami zaoferowanymi przez twórcę procesora (najczęściej są one dwie), to warto rozważyć zalety stosowania małych i dużych stron. Za stosowaniem małych stron przemawiają następujące czynniki:

- mniejsza fragmentacja wewnętrzna,
- mniejszy czas przesyłania strony z pamięci pomocniczej do operacyjnej lub odwrotnie,
- lepsza *rozdzielcość* tzn. stosunek ilości informacji potrzebnej w danej chwili do ilości informacji niepotrzebnej, zawartych w obrębie strony.

Za stosowaniem stron dużych przemawiają z kolei:

- mniejsza liczba operacji wejścia-wyjścia wykonywanych podczas działania procesu,
- mniejszy rozmiar tablicy stron,
- mniejsza liczba błędów stron.

Obecnie stosuje się najczęściej duże strony. Procesory firmy Intel i pokrewne (32-bitowe i 64-bitowe) pozwalają na wybór między 4KiB, a 4MiB. Procesory DEC Alpha (64-bitowe) stosują strony o wielkości 8KiB. Procesor Motorola 68030 (32-bitowy) pozwala programiście systemowemu na wybór rozmiaru strony w zakresie od 256 B do 32 KiB.

## Rozmiar strony

Chociaż w większości platform sprzętowych wpływ programistów systemowych na wielkość strony jest ograniczony do wyboru między kilkoma opcjami zaoferowanymi przez twórcę procesora (najczęściej są one dwie), to warto rozważyć zalety stosowania małych i dużych stron. Za stosowaniem małych stron przemawiają następujące czynniki:

- mniejsza fragmentacja wewnętrzna,
- mniejszy czas przesyłania strony z pamięci pomocniczej do operacyjnej lub odwrotnie,
- lepsza *rozdzielcość* tzn. stosunek ilości informacji potrzebnej w danej chwili do ilości informacji niepotrzebnej, zawartych w obrębie strony.

Za stosowaniem stron dużych przemawiają z kolei:

- mniejsza liczba operacji wejścia-wyjścia wykonywanych podczas działania procesu,
- mniejszy rozmiar tablicy stron,
- mniejsza liczba błędów stron.

Obecnie stosuje się najczęściej duże strony. Procesory firmy Intel i pokrewne (32-bitowe i 64-bitowe) pozwalają na wybór między 4KiB, a 4MiB. Procesory DEC Alpha (64-bitowe) stosują strony o wielkości 8KiB. Procesor Motorola 68030 (32-bitowy) pozwala programiście systemowemu na wybór rozmiaru strony w zakresie od 256 B do 32 KiB.

## Rozmiar strony

Chociaż w większości platform sprzętowych wpływ programistów systemowych na wielkość strony jest ograniczony do wyboru między kilkoma opcjami zaoferowanymi przez twórcę procesora (najczęściej są one dwie), to warto rozważyć zalety stosowania małych i dużych stron. Za stosowaniem małych stron przemawiają następujące czynniki:

- mniejsza fragmentacja wewnętrzna,
- mniejszy czas przesyłania strony z pamięci pomocniczej do operacyjnej lub odwrotnie,
- lepsza *rozdzielcość* tzn. stosunek ilości informacji potrzebnej w danej chwili do ilości informacji niepotrzebnej, zawartych w obrębie strony.

Za stosowaniem stron dużych przemawiają z kolei:

- mniejsza liczba operacji wejścia-wyjścia wykonywanych podczas działania procesu,
- mniejszy rozmiar tablicy stron,
- mniejsza liczba błędów stron.

Obecnie stosuje się najczęściej duże strony. Procesory firmy Intel i pokrewne (32-bitowe i 64-bitowe) pozwalają na wybór między 4KiB, a 4MiB. Procesory DEC Alpha (64-bitowe) stosują strony o wielkości 8KiB. Procesor Motorola 68030 (32-bitowy) pozwala programiście systemowemu na wybór rozmiaru strony w zakresie od 256 B do 32 KiB.

## Rozmiar strony

Chociaż w większości platform sprzętowych wpływ programistów systemowych na wielkość strony jest ograniczony do wyboru między kilkoma opcjami zaoferowanymi przez twórcę procesora (najczęściej są one dwie), to warto rozważyć zalety stosowania małych i dużych stron. Za stosowaniem małych stron przemawiają następujące czynniki:

- mniejsza fragmentacja wewnętrzna,
- mniejszy czas przesyłania strony z pamięci pomocniczej do operacyjnej lub odwrotnie,
- lepsza *rozdzielcość* tzn. stosunek ilości informacji potrzebnej w danej chwili do ilości informacji niepotrzebnej, zawartych w obrębie strony.

Za stosowaniem stron dużych przemawiają z kolei:

- mniejsza liczba operacji wejścia-wyjścia wykonywanych podczas działania procesu,
- mniejszy rozmiar tablicy stron,
- mniejsza liczba błędów stron.

Obecnie stosuje się najczęściej duże strony. Procesory firmy Intel i pokrewne (32-bitowe i 64-bitowe) pozwalają na wybór między 4KiB, a 4MiB. Procesory DEC Alpha (64-bitowe) stosują strony o wielkości 8KiB. Procesor Motorola 68030 (32-bitowy) pozwala programiście systemowemu na wybór rozmiaru strony w zakresie od 256 B do 32 KiB.

## Rozmiar strony

Chociaż w większości platform sprzętowych wpływ programistów systemowych na wielkość strony jest ograniczony do wyboru między kilkoma opcjami zaoferowanymi przez twórcę procesora (najczęściej są one dwie), to warto rozważyć zalety stosowania małych i dużych stron. Za stosowaniem małych stron przemawiają następujące czynniki:

- mniejsza fragmentacja wewnętrzna,
- mniejszy czas przesyłania strony z pamięci pomocniczej do operacyjnej lub odwrotnie,
- lepsza *rozdzielcość* tzn. stosunek ilości informacji potrzebnej w danej chwili do ilości informacji niepotrzebnej, zawartych w obrębie strony.

Za stosowaniem stron dużych przemawiają z kolei:

- mniejsza liczba operacji wejścia-wyjścia wykonywanych podczas działania procesu,
- mniejszy rozmiar tablicy stron,
- mniejsza liczba błędów stron.

Obecnie stosuje się najczęściej duże strony. Procesory firmy Intel i pokrewne (32-bitowe i 64-bitowe) pozwalają na wybór między 4KiB, a 4MiB. Procesory DEC Alpha (64-bitowe) stosują strony o wielkości 8KiB. Procesor Motorola 68030 (32-bitowy) pozwala programiście systemowemu na wybór rozmiaru strony w zakresie od 256 B do 32 KiB.

## Rozmiar strony

Chociaż w większości platform sprzętowych wpływ programistów systemowych na wielkość strony jest ograniczony do wyboru między kilkoma opcjami zaoferowanymi przez twórcę procesora (najczęściej są one dwie), to warto rozważyć zalety stosowania małych i dużych stron. Za stosowaniem małych stron przemawiają następujące czynniki:

- mniejsza fragmentacja wewnętrzna,
- mniejszy czas przesyłania strony z pamięci pomocniczej do operacyjnej lub odwrotnie,
- lepsza *rozdzielcość* tzn. stosunek ilości informacji potrzebnej w danej chwili do ilości informacji niepotrzebnej, zawartych w obrębie strony.

Za stosowaniem stron dużych przemawiają z kolei:

- mniejsza liczba operacji wejścia-wyjścia wykonywanych podczas działania procesu,
- mniejszy rozmiar tablicy stron,
- mniejsza liczba błędów stron.

Obecnie stosuje się najczęściej duże strony. Procesory firmy Intel i pokrewne (32-bitowe i 64-bitowe) pozwalają na wybór między 4KiB, a 4MiB. Procesory DEC Alpha (64-bitowe) stosują strony o wielkości 8KiB. Procesor Motorola 68030 (32-bitowy) pozwala programiście systemowemu na wybór rozmiaru strony w zakresie od 256 B do 32 KiB.

## Wpływ stronicowania na żądanie na wykonanie procesu

Choć stronicowanie na żądanie jest przezroczyste dla programisty piszącego aplikacje dla użytkownika, to dokonany przez niego dobór struktur danych oraz sposobu odwoływania do nich może mieć wpływ na częstotliwość błędów stron generowanych przez jego program. Zalecane jest stosowanie dużej liczby struktur odznaczających się dobrą lokalnością, a małej struktur odznaczających się złą lokalnością. Do pierwszej kategorii zalicza się między innymi stos, do drugiej tablica z adresowaniem mieszanym (ang. hash table). Również etapy komplikacji i ładowania mogą mieć znaczenie dla częstości błędów stron generowanych przez proces. Można ją zmniejszyć, jeśli kompilator będzie oddzielał kod od danych, a program ładowający wyrównywał rozmieszczenie spójnych elementów programu (jak podprogramy i duże struktury danych) do granicy stron. Stronicowanie na żądanie ma wpływ na efektywność aplikacji wieloprocesorowych (wielordzeniowych).

## Blokowanie stron

Jeśli system operacyjnych pozwala, żeby proces użytkownika umieszczał bufory dla operacji wejścia-wyjścia w obrębie swojej przestrzeni adresowej, to musi także zapewnić, że strona je zawierająca pozostanie w ramce do czasu zrealizowania tej operacji i że ramka ta nie zostanie przydzielona innemu procesowi. Działanie to wymaga wsparcia sprzętowego, w postaci odpowiedniego mechanizmu kontrolującego *bit blokady* (ang. lock) w tablicy stron. Jeśli jest on ustawiony dla którejś ze stron, to oznacza, że tej strony nie można wymienić, nawet jeśli jest idealną kandydatką do wymiany i że musi ona pozostać w ramce. Ramka ta nie może także zmienić właściciela. Blokowanie stron może również być użyte do ograniczenia szamotania procesów o niskim priorytecie. Zablokowanie stron takiego procesu uniemożliwia odebranie ich przez proces o wysokim priorytecie.

## Implementacje tablicy stron

Ponieważ tablice stron mogą być bardzo dużych rozmiarów, to stosowane są różne ich implementacje, celem zmniejszenia ich wielkości. Jedną z nich jest *odwrócona tablica stron* (ang. inverted page table). W takiej tablicy indeksami są fizyczne adresy stron (adresy bazowe ramek), a wartościami elementów numery stron i identyfikatory procesów do których one należą. Ponieważ pamięć fizyczna jest zazwyczaj mniejsza od pamięci wirtualnej, to tym samym rozmiar odwróconej tablicy stron jest mniejszy od „zwykłej” tablicy stron. Aby przyspieszyć wyszukiwanie informacji w tej tablicy stosowane jest haszowanie, zazwyczaj realizowane sprzętowo. Wadą tego rozwiązania jest niemożność szybkiego określenia, czy błąd strony spowodowany jest jej brakiem w pamięci operacyjnej, czy też tym, że ta strona nie istnieje. Inna wada, to brak lokalności odwołań, co może stanowić problem dla rejestrów TLB. Odwrócone tablice stron stosowane są w 64-bitowych platformach sprzętowych. Innym popularnym rozwiązaniem jest *wielopoziomowa tablica stron* (ang. multilevel page table). Polega ono na podziale tablicy na np.: trzy części nazywane *katalogiem głównym*, *katalogiem pośrednim* i tablicą stron. Pozycje w katalogu głównym wskazują na pozycje w katalogu pośrednim, a te na pozycje w tablicy stron. Rozwiązanie to zmniejsza ilość pamięci operacyjnej wymaganej do przechowania tablicy stron (niepotrzebne elementy mogą być trzymane w przestrzeni wymiany) oraz pozwala na lepszą współpracę tablicy stron z buforami TLB. System operacyjny utrzymuje osobne tablice zawierające numery bloków pamięci pomocniczej, w których umieszczone są wymienione strony procesu. Korzysta z nich procedura obsługi błędu strony.

## Segmentacja na żądanie

*Segmentacja na żądanie* jest alternatywnym w stosunku do stronicowania na żądanie sposobem realizacji koncepcji pamięci wirtualnej. Nie jest ona tak wydajna jak stronicowanie, ale również nie wymaga takiego jak ono wsparcia sprzętowego. Stosował ją system OS/2. Tablica segmentów nazywana tutaj *tablicą deskryptorów segmentów* zawiera *deskryptory segmentów*, w których są umieszczone wszelkie informacje związane z poszczególnymi segmentami, takie jak ich wielkość, tryb ochrony i umiejscowienie. W każdym z nich umieszczony jest również *bit poprawności* określający, czy dany segment znajduje się w pamięci operacyjnej, czy też w przestrzeni wymiany. Jeśli proces odwołuje się do segmentu, którego nie ma w pamięci, to generowany jest *błąd segmentu* i w wyniku jego obsługi żądany segment jest sprowadzany do pamięci. Do określenia, który segment ma być wymieniony, w przypadku takiej konieczności służy zawarty w deskryptorze *bit udostępnienia*. Istnieją również wywołania systemowe, które pozwalają procesowi użytkownika wskazać systemowi, które z jego segmentów można usunąć z pamięci, a które powinny w niej pozostać.

## Segmentacja stronicowana na żądanie

Obie metody realizacji pamięci wirtualnej można połączyć w jeden schemat nazywany *segmentacją stronicowaną na żądanie*. Ten schemat zarządzania pamięcią stosuje system OS/2 Warp. Również system Linux korzysta na platformach sprzętowych udostępniających segmentację z segmentów, choć podstawę jego systemu zarządzania pamięcią dla wszystkich platform stanowi stronicowanie na żądanie. Segmentacja jest wykorzystywana przez niego w platformach x86, jako uzupełnienie mechanizmu ochrony stron. Korzysta on z segmentów kodu i danych zarówno dla jądra, jak i procesów użytkownika, z tym, że obejmują one zasięgiem całą pamięć wirtualną, a wymianie podlegają strony wewnętrz segmentów procesu użytkownika. W bardzo ograniczonym zakresie wykorzystywany jest również segment TSS (ang. Task State Segment). W segmentacji stronicowanej na żądanie mogą być jednocześnie używanie strony o różnych rozmiarach.

Plan wykładu  
Przydział ramek  
Szamotanie  
Inne zagadnienia dotyczące stronicowania na żądanie  
Inne metody realizacji pamięci wirtualnej

Segmentacja na żądanie  
**Segmentacja stronicowana na żądanie**

## Pytania

?

**Koniec**

Dziękuję Państwu za uwagę!

## Systemy Operacyjne — Pamięć masowa

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 19 grudnia 2020

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ③ Pierwsze rozwiążanie
  - ④ Dyski magnetyczne
  - ⑤ Dyski optyczne
  - ⑥ Układy scalone
- ⑦ Katalog urządzenia
- ⑧ Zarządzanie obszarami wolnymi
  - ⑨ Wektorowy
  - ⑩ Lista powiązana
  - ⑪ Grupowanie
  - ⑫ Zliczanie
- ⑬ Przydział miejsca na dysku
  - ⑭ Przydział ciągły
  - ⑮ Przydział liniowy
  - ⑯ Przydział indeksowy
- ⑰ Planowanie dostępu do dysku
  - ⑱ Metoda FCFS
  - ⑲ Metoda SSTF
  - ⑳ Metoda SCAN
  - ㉑ Metoda C-SCAN
  - ㉒ Metody LOOK i C-LOOK
  - ㉓ Kolejki do sektorów
- ㉔ Inne zagadnienia
  - ㉕ Sposoby zarządzania obszarami wolnymi

# Plan wykładu

## 1 Wprowadzenie

### 2 Pamięć masowa

- 3 Pierwotne rozwiązań
- 3 Dyski magnetyczne
- 3 Dyski optyczne
- 3 Układy scalone

### 3 Katalog urządzenia

### 4 Zarządzanie obszarami wolnymi

- 3 Wektorowy
- 3 Lista powiązana
- 3 Grupowanie
- 3 Zliczanie

### 5 Przydział miejsca na dysku

- 3 Przydział ciągły
- 3 Przydział liniowy
- 3 Przydział indeksowy

### 6 Planowanie dostępu do dysku

- 3 Metoda FCFS
- 3 Metoda SSTF
- 3 Metoda SCAN
- 3 Metoda C-SCAN
- 3 Metody LOOK i C-LOOK
- 3 Kolejki do sektorów

### 7 Inne zagadnienia

- 3 Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
    - ② Dyski magnetyczne
    - ③ Dyski optyczne
    - ④ Układy scalone
  - ② Katalog urządzenia
  - ③ Zarządzanie obszarami wolnymi
    - ① Wektorowy
    - ② Lista powiązana
    - ③ Grupowanie
    - ④ Zliczanie
  - ④ Przydział miejsca na dysku
    - ① Przydział ciągły
    - ② Przydział liniowy
    - ③ Przydział indeksowy
  - ⑤ Planowanie dostępu do dysku
    - ① Metoda FCFS
    - ② Metoda SSTF
    - ③ Metoda SCAN
    - ④ Metoda C-SCAN
    - ⑤ Metody LOOK i C-LOOK
    - ⑥ Kolejki do sektorów
  - ⑦ Inne zagadnienia
    - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
    - ③ Dyski optyczne
    - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektorowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektorowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektorowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wykaz blokowy
  - ② Lista powiększana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział skoły
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przedziałowy
  - ② Przydział liniowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda CSCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda CSCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Sposoby zarządzania obszarami wolnymi

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów

⑦ Inne zagadnienia

# Plan wykładu

- ① Wprowadzenie
- ② Pamięć masowa
  - ① Pierwsze rozwiązania
  - ② Dyski magnetyczne
  - ③ Dyski optyczne
  - ④ Układy scalone
- ③ Katalog urządzenia
- ④ Zarządzanie obszarami wolnymi
  - ① Wektor bitowy
  - ② Lista powiązana
  - ③ Grupowanie
  - ④ Zliczanie
- ⑤ Przydział miejsca na dysku
  - ① Przydział ciągły
  - ② Przydział listowy
  - ③ Przydział indeksowy
- ⑥ Planowanie dostępu do dysku
  - ① Metoda FCFS
  - ② Metoda SSTF
  - ③ Metoda SCAN
  - ④ Metoda C-SCAN
  - ⑤ Metody LOOK i C-LOOK
  - ⑥ Kolejki do sektorów
- ⑦ Inne zagadnienia
  - ① Efektywność i niezawodność oraz hierarchia pamięci

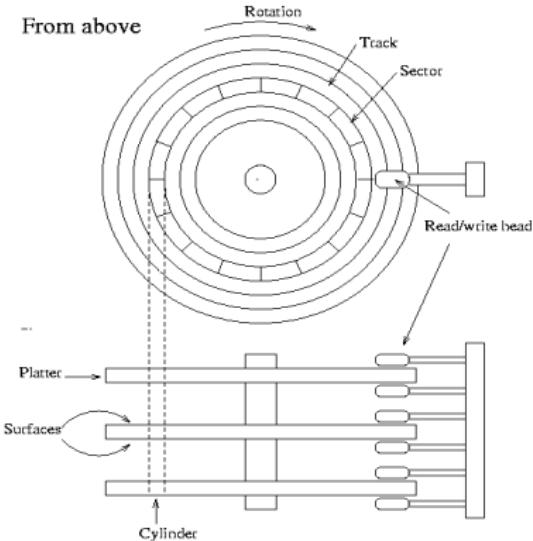
## Wprowadzenie

Pamięć operacyjna komputera (RAM), mimo że jest głównym magazynem danych z którego korzysta procesor i urządzenia wejścia-wyjścia poprzez kanały DMA, to nie jest wystarczająco pojemna, aby pomieścić wszystkie dane, jakimi dysponują użytkownicy systemu komputerowego. Dodatkowo jest ona ulotna, co oznacza, że po wyłączeniu zasilania jej zawartość jest tracona. Konieczne więc było wprowadzenie dodatkowej pamięci, która byłaby szybka, pojemna i przechowywała dane w sposób trwałym. Ten rodzaj pamięci nazywamy *pamięcią masową* lub *pamięcią zewnętrzną* (ang. external memory, secondary storage). Oprócz przechowywania danych ten rodzaj pamięci służy również do realizacji przestrzeni wymiany (nazywanej także *pamięcią pomocniczą*) w implementacjach pamięci wirtualnej. Ze względu na popularność i efektywność rozwiązania dalsza część wykładu będzie dotyczyła głównie dysków twardych, ale niektóre zagadnienia przynoszą się również na inne urządzenia pamięci masowej.

## Pierwsze rozwiązania

Pierwszą realizacją pamięci masowej były karty i taśmy perforowane (dziurkowane). Ich zaletą była możliwość odczytywania i bezpośredniej manipulacji danymi przez człowieka. Niestety czytniki taśm i kart perforowanych były urządzeniami powolnymi i zawodnymi. Dodatkowo karty i taśmy jako nośniki danych miały niewielką pojemność. Zastąpiły je taśmy magnetyczne. Miały one dużą pojemność i były stosunkowo szybkie. Informacje na taśmach magnetycznych przechowywane były w plikach. Pliki rozdzielane były odpowiednimi znacznikami identyfikującymi koniec pliku i początek następnego. Niestety taśmy magnetyczne oferowały jedynie *dostęp sekwencyjny*, co oznaczało, że aby odczytać plik zapisany na końcu taśmy należało odczytać wszystkie go poprzedzające. Obecnie taśmy magnetyczne stosuje się jedynie w urządzeniach do tworzenia kopii zapasowych danych, tzw. streamerach.

# Dyski twarde



Obecnie podstawowym rodzajem pamięci masowej jest **dysk twarde**. Fizycznie dysk twarde jest zbudowany z jednego lub kilku talerzy wykonanych z sztywnego materiału (stąd inna nazwa dysku twardego: *dysk sztywny*). Każdy z tych dysków jest pokryty obustronnie warstwą nośnika magnetycznego. Wszystkie dyski są osadzone na jednej osi i wirują z taką samą prędkością kątową. Z każdą stroną dysku jest stowarzyszona głowica potrafiąca zapisywać i odczytywać informacje. Logicznie każda strona dysku jest podzielona na koncentryczne okręgi zwane **ścieżkami**. Ścieżki na talerzach, które są osiągalne przy tym samym ustawieniu głowic tworzą **cylinder**. Każda ścieżka jest podzielona na **sektory** o takiej samej wielkości. Sektory są oddzielane za pomocą odpowiednich znaczników.

(Rysunek pochodzi z: [http://www.faqs.org/docs/linux\\_admin/x1001.html](http://www.faqs.org/docs/linux_admin/x1001.html))

## Dyski twarde

Typowy rozmiar sektora wynosi 512 bajtów. Posługiwanie się tak małą jednostką pamięci jest stosunkowo niewygodne, więc system operacyjny łączy je w większe jednostki nazywane *jednostkami alokacji* lub *blokami alokacji* (krótko: *blokami*). Aby zaadresować pojedynczy sektor należy podać trzy współrzędne: numer głowicy, numer cylindra (ścieżki) i numer sektora. Aby ułatwić ten proces system operacyjny stosuje konwersję adresu dla bloków alokacji. Adres jest przeliczany z trójwymiarowego na jednowymiarowy za pomocą równania:  $nb = ns + ls \cdot (nh + lh \cdot nc)$ , gdzie  $nb$  jest numerem bloku,  $ns$  jest numerem sektora,  $ls$  liczbą sektorów na ścieżce,  $nh$  numerem głowicy,  $lh$  liczbą głowic,  $nc$  liczbą ścieżek w cylindrze. Taki proces przeliczania adresów nazywamy *linearyzacją*. Jak wynika z opisu, konstrukcja dysku twardego umożliwia *bezpośredni dostęp* do każdego miejsca na dysku. Ten rodzaj dostępu nazywamy również *dostęmem swobodnym*. Typowy dysk twarty, jest dyskiem z ruchomymi głowicami, co oznacza, że każda głowica umieszczona jest na końcu ramienia, które ustawia ją nad określoną ścieżką. Istnieją również konstrukcje w których głowice są nieruchome, ale każda ścieżka ma swoją głowicę. Umieszczone są one na ramieniu przewieszonym wzdłuż talerza. Pierwowzorem tego rozwiązania były *bębny magnetyczne*, które miały kształt cylindra powleconego nośnikiem magnetycznym. Powierzchnia tego nośnika była podzielona logicznie na ścieżki, a każda ze ścieżek dysponowała własną głowicą.

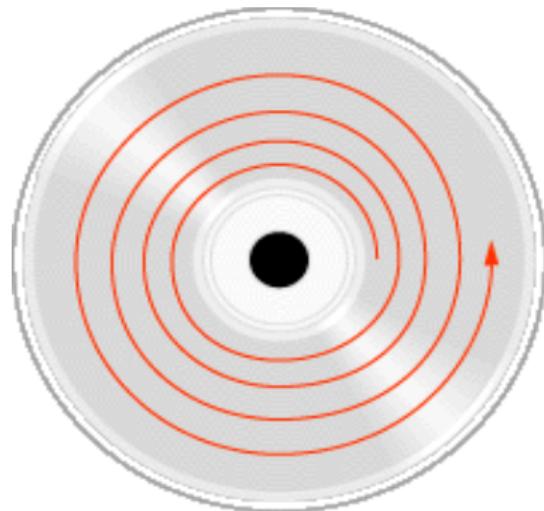
## Dyskietki

Dyski twardy są najczęściej instalowane na stałe w komputerach. Choć istnieje możliwość uczynienia ich mobilnymi za pomocą tzw. kieszeni na dyski twardy, to to rozwiązanie nie jest zbyt poręczne. Odmianą dysku magnetycznego, który jest przeznaczony do przenoszenia danych między systemami komputerowymi jest dysk elastyczny zwany popularnie dyskietką. Składa się ona z obudowy i zawartego w niej dysku wykonanego z elastycznego materiału (stąd nazwa), pokrytego nośnikiem magnetycznym i dodatkowo warstwą ochronną. Dyskietki mają wielokrotnie mniejszą pojemność niż dyski twardy. Zasadnicza różnica między tymi dwoma nośnikami informacji polega na zachowaniu głowic podczas pracy. Talerze dysku twardego zamknięte są w hermetycznej obudowie i kiedy wirują tworzy się na ich powierzchni poduszka powietrzna, po której ślizgają się głowice. Jeśli głowica zetknie się z powierzchnią talerza, to nastąpi jej uszkodzenie, dlatego przed wyłączeniem dysku jego sterownik wykonuje operację *parkowania głowic*, czyli umieszcza je w bezpiecznym miejscu tak, aby nie mogły uszkodzić nośnika<sup>1</sup>. W przypadku dyskietki głowica porusza się po warstwie ochronnej. Dopóki nie zostanie ona uszkodzona, dyskietka nadaje się do użytku.

---

<sup>1</sup>Kiedyś tę operację należało wykonywać samodzielnie, za pomocą odpowiedniego programu.

## Dyski optyczne



©2000 How Stuff Works

Najpopularniejszymi rodzajami dysków optycznych są obecnie dyski CD, DVD i Blue-ray (w odmianach tylko do odczytu, do jednokrotnego zapisu, do wielokrotnego zapisu). Dane w tego typu dyskach zapisywane są na pojedynczej ścieżce, która tworzy „muszlę ślimaka” i rozpoczyna się od środka dysku. Jest ona podzielona na sektory o wielkości 2048 bajtów (2 KiB). Dyski zapisywane jednokrotnie są pokryte substancją, która przy podgrzaniu silniejszym światłem lasera staje się matowa. Dyski zapisywane wielokrotnie są pokryte specjalnym stopem metali, który w zależności od temperatury podgrzania staje się przezroczysty lub matowy. Dyski optyczne, ze względu na sposób zapisu danych nie nadają się do realizacji urządzenia wymiany. Odczyt danych z dysków optycznych wymaga stażej prędkości liniowej, a więc te nośniki wirują ze zmienną prędkością kątową.

(Ilustracja pochodzi  
z <http://computer.howstuffworks.com/cd.htm>)

## Układy scalone

Do realizacji pamięci masowej nadają się pamięci trwałe wykonane w postaci układów scalonych. Pierwszymi historycznie pamięciami tego typu były pamięci ROM (ang. Read Only Memory). Dane zapisywane w nich były na stałe przez producenta za pomocą procesu fotolitografii. Później pojawiły się pamięci PROM (ang. Programmable Read Only Memory). Te pamięci użytkownik mógł zapisać, ale tylko raz. Możliwość kasowania danych po raz pierwszy pojawiła się w pamięciach typu EPROM (ang. Erasable Programmable Read Only Memory). Dane kasowane były przy pomocy promieniowania ultrafioletowego, dlatego te układy miały charakterystyczne, kwarcowe „okienko” w obudowach. Możliwość kasowania zawartości przy pomocy impulsu elektrycznego posiadają pamięci EEPROM (ang. Electrically-Erasable Programmable Read-Only Memory). Udoskonaleniem tych pamięci są pamięci Flash-EEPROM nazywane krótko Flash. Obecnie istnieje wiele odmian pamięci Flash-EEPROM wykonanych w postaci urządzeń USB (PenDrive), kart SD/MMC itd. Mają one w porównaniu z dyskami twardymi małą pojemność, ale istnieją wersje systemów operacyjnych, które całkowicie można umieścić na tych urządzeniach, wraz z oprogramowaniem użytkowym (np.: SLAX Linux). Duże pojemności oferują urządzenia SSD (ang. Solid State Drive), które w przyszłości mogą zastąpić dyski twarde.

## Katalog urządzenia

*Katalog urządzenia*, zwany również *katalogiem głównym* lub *katalogiem fizycznym*, jest miejscem, gdzie zgromadzone są dane na temat plików zawartych w pamięci masowej. Jest on umiejscowiony w określonym miejscu urządzenia. Może to być drugi sektor urządzenia (pierwszy jest zarezerwowany na program ładowający system operacyjny) lub któryś z początkowych sektorów. Do informacji zgromadzonych w katalogu głównym należą, lub mogą należeć: nazwy plików, rozmieszczenie plików na dysku, rozmiar i typy plików, identyfikatory ich właścicieli, informacje o ochronie, czas i data utworzenia, modyfikacji lub odczytu. Katalog ten ma ograniczoną pojemność.

## Zarządzanie obszarami wolnymi

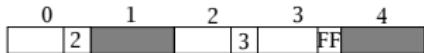
System operacyjny utrzymuje wykaz wolnych bloków alokacji, aby móc prawidłowo i sprawnie nimi zarządzać. Taki wykaz jest przechowywany w urządzeniu, nawet jeśli ono nie jest podłączone do systemu komputerowego lub gdy system ten jest wyłączony. Wykaz wolnych bloków może być zrealizowany na kilka sposobów.

## Wektor bitowy

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	1	1	1	0	0	0	1	1	1	0	1	0	1	0	0	0	0

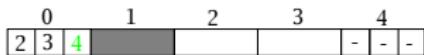
Wektor bitowy lub *mapa bitowa* jest ciągiem bitów. Wartość każdego bitu opisuje zajętość bloku, którego numer odpowiada pozycji bitu w wektorze. Jeśli bit ten ma wartość 1 to został już przydzielony, jeśli 0 to jest wolny. Znajdywanie bloków wolnych na dysku przy pomocy wektora bitowego umożliwia rozkazy bitowe procesorów. Metoda ta nie jest wygodna dla dużych dysków. Ponadto nie pozwala oznaczyć, np.: bloków uszkodzonych.

## Lista powiązana



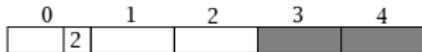
Bloki wolne mogą być powiązane w listę. W tym przypadku w każdym takim bloku umieszczany jest wskaźnik zawierający numer kolejnego wolnego bloku. Ostatni wolny blok zawiera we wskaźniku pewną ustaloną wartość sygnalizującą koniec listy (wartość ta oznaczana jest jako nil lub null). Wskaźnik do pierwszego wolnego bloku jest umieszczany w określonym miejscu na dysku. Wadą tego rozwiązania jest przeglądanie listy wolnych bloków - wymaga ono dużej ilości czasu.

## Grupowanie



W tej metodzie poświęca się jeden z bloków wolnych po to aby, utrzymywać w nim numery innych bloków wolnych. Ten blok nazywa się *blokiem indeksowym*. Jeśli jeden blok indeksowy nie wystarcza do zapamiętania numerów wszystkich bloków wolnych, to stosuje się kilka bloków indeksowych połączonych w listę. Ostatni wskaźnik w takim bloku jest wskaźnikiem do kolejnego bloku indeksowego.

## Zliczanie



W metodzie zliczania zapamiętywany jest na liście adres pierwszego wolnego bloku i liczba bloków wolnych występujących za nim. Takie rozwiązanie jest efektywne, jeśli na dysku występuje mało zgrupowań dużej liczby wolnych bloków, przylegających do siebie.

## Przydział miejsca na dysku

Równie ważne co zarządzanie obszarami wolnymi na dysku jest przydzielanie miejsca na pliki. Istnieje kilka metod rozwiązania tego problemu, ale należy pamiętać, że każda z nich jest obarczona fragmentacją wewnętrzną.

## Przydział ciągły

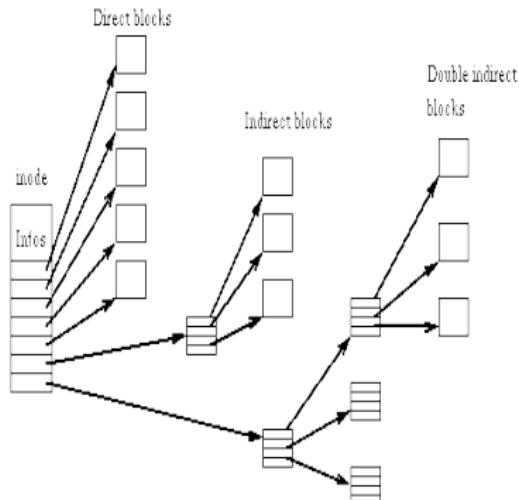
W metodzie przydziału ciągłego, plikom przydzielane są przylegające do siebie bloki alokacji. W katalogu zapamiętywany jest numer pierwszego bloku i liczba bloków przydzielonych plikowi. Przydział ten pozwala zarówno na dostęp sekwencyjny, jak i swobodny do pliku. Do wyboru miejsca na dysku dla pliku stosuje się takie same strategie, jak w przypadku przydziału obszarów ciągłych w pamięci operacyjnej. Przydział ten prowadzi do fragmentacji zewnętrznej, która może być likwidowana techniką upakowania, ale jest to działanie czasochłonne. Kłopotliwe jest również dopisywanie informacji do pliku. Wiąże się ono z koniecznością skopiowania istniejącej już zawartości pliku do innego wolnego miejsca, które pomieściłoby powiększony plik.

## Przydział listowy

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
5	0	0	0	7	FE	10	0	0	16	0	0	0	0	0	FF	0	0

W przydziale listowym wszystkie bloki należące do jednego pliku są powiązane w listę. Poświęca się część bloku na zapamiętanie wskaźnika do następnego bloku należącego do pliku. Ostatni blok zawiera we wskaźniku ustaloną wartość. Zaletą tego rozwiązania jest brak fragmentacji zewnętrznej, bo pliki nie muszą zajmować obszaru ciągłego na dysku. Wadami natomiast mała odporność na błędy (by ją zwiększyć podwaja się wskaźniki), dostęp wyłącznie sekwencyjny do pliku i poświęcenie części przydzielonego bloku na wskaźnik. Dwie ostatnie wady można wyeliminować stosując tablicę alokacji plików, która pozwala zapamiętać informacje nie tylko o przydzielonych blokach, ale również o uszkodzonych i wolnych. Numer pierwszego bloku przydzielonego plikowi zapamiętywany jest w katalogu.

## Przydział indeksowy



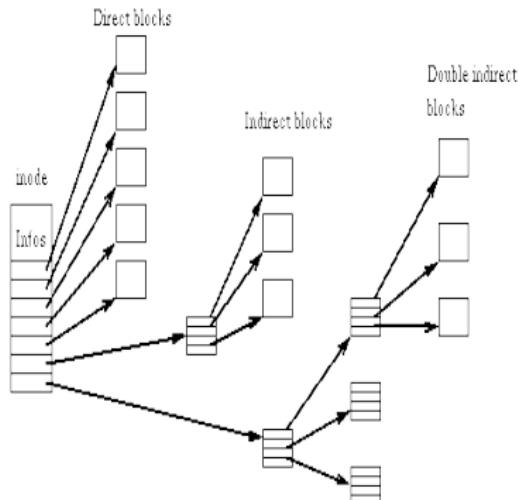
W metodzie przydziału indeksowego wykorzystuje się bloki indeksowe do zapamiętania numerów kolejnych bloków przydzielonych plikowi. Jeśli plik jest duży, to jeden blok indeksowy może nie wystarczyć. Stosuje się więc trzy rozwiązania:

- Schemat listowy-ostatni wskaźnik w bloku indeksowym wskazuje kolejny blok indeksowy.
- Indeks wielopoziomowy-buduje się hierarchię bloków indeksowych, w postaci drzewa. Bloki indeksowe, które są liśćmi wskazują na bloki danych
- Schemat kombinowany-po raz pierwszy zastosowany w Uniksie BSD. Kilka pierwszych wskaźników bloku indeksowego wskazuje bloki z danymi, drugi od końca na pojedynczo pośredni blok indeksowy, a ostatni na blok powojewnie pośredni, tak jak przedstawia umieszczony obok rysunek

W przydziale indeksowym możliwy jest zarówno dostęp sekwenncyjny, jak i swobodny do pliku. Nie występuje fragmentacja zewnętrzna. W katalogu zapamiętywany jest numer pierwszego bloku indeksowego związanego z plikiem.

(Źródło ilustracji: <http://wikipedia.org>)

## Przydział indeksowy



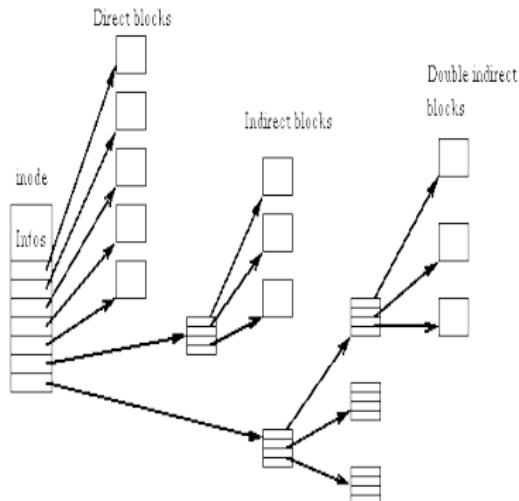
W metodzie przydziału indeksowego wykorzystuje się bloki indeksowe do zapamiętania numerów kolejnych bloków przydzielonych plikowi. Jeśli plik jest duży, to jeden blok indeksowy może nie wystarczyć. Stosuje się więc trzy rozwiązania:

- **Schemat listowy**-ostatni wskaźnik w bloku indeksowym wskazuje kolejny blok indeksowy.
- Indeks wielopoziomowy-buduje się hierarchię bloków indeksowych, w postaci drzewa. Bloki indeksowe, które są liśćmi wskazują na bloki danych
- Schemat kombinowany-po raz pierwszy zastosowany w Uniksie BSD. Kilka pierwszych wskaźników bloku indeksowego wskazuje bloki z danymi, drugi od końca na pojedynczo pośredni blok indeksowy, a ostatni na blok powojewnie pośredni, tak jak przedstawia umieszczony obok rysunek

W przydziale indeksowym możliwy jest zarówno dostęp sekwenncyjny, jak i swobodny do pliku. Nie występuje fragmentacja zewnętrzna. W katalogu zapamiętywany jest numer pierwszego bloku indeksowego związanego z plikiem.

(Źródło ilustracji: <http://wikipedia.org>)

## Przydział indeksowy



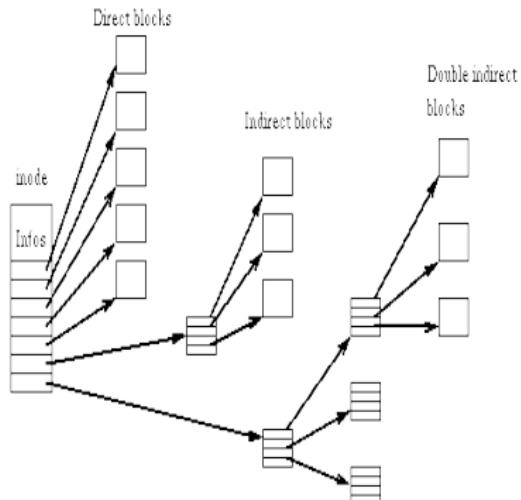
W metodzie przydziału indeksowego wykorzystuje się bloki indeksowe do zapamiętania numerów kolejnych bloków przydzielonych plikowi. Jeśli plik jest duży, to jeden blok indeksowy może nie wystarczyć. Stosuje się więc trzy rozwiązania:

- **Schemat listowy**-ostatni wskaźnik w bloku indeksowym wskazuje kolejny blok indeksowy.
- **Indeks wielopoziomowy**-buduje się hierarchie bloków indeksowych, w postaci drzewa. Bloki indeksowe, które są liścimi wskazują na bloki danych
- Schemat kombinowany-po raz pierwszy zastosowany w Uniksie BSD. Kilka pierwszych wskaźników bloku indeksowego wskazuje bloki z danymi, drugi od końca na pojedynczo pośredni blok indeksowy, a ostatni na blok powojewnie pośredni, tak jak przedstawia umieszczony obok rysunek

W przydziale indeksowym możliwy jest zarówno dostęp sekwenncyjny, jak i swobodny do pliku. Nie występuje fragmentacja zewnętrzna. W katalogu zapamiętywany jest numer pierwszego bloku indeksowego związanego z plikiem.

(Źródło ilustracji: <http://wikipedia.org>)

## Przydział indeksowy



W metodzie przydziału indeksowego wykorzystuje się bloki indeksowe do zapamiętania numerów kolejnych bloków przydzielonych plikowi. Jeśli plik jest duży, to jeden blok indeksowy może nie wystarczyć. Stosuje się więc trzy rozwiązania:

- **Schemat listowy**-ostatni wskaźnik w bloku indeksowym wskazuje kolejny blok indeksowy.
- **Indeks wielopoziomowy**-buduje się hierarchię bloków indeksowych, w postaci drzewa. Bloki indeksowe, które są liśćmi wskazują na bloki danych
- **Schemat kombinowany**-po raz pierwszy zastosowany w Uniksie BSD. Kilka pierwszych wskaźników bloku indeksowego wskazuje bloki z danymi, drugi od końca na pojedynczo pośredni blok indeksowy, a ostatni na blok po-dwójnie pośredni, tak jak przedstawia umieszczony obok rysunek

W przydziale indeksowym możliwy jest zarówno dostęp sekwenncyjny, jak i swobodny do pliku. Nie występuje fragmentacja zewnętrzna. W katalogu zapamiętywany jest numer pierwszego bloku indeksowego związanego z plikiem.

(Źródło ilustracji: <http://wikipedia.org>)

Plan wykładu	Metoda FCFS
Wprowadzenie	Metoda SSTF
Katalog urządzeń	Metoda SCAN
Zarządzanie obszarami wolnymi	Metoda C-SCAN
Przydział miejsca na dysku	Metoda LOOK i C-LOOK
Planowanie dostępu do dysku	Kolejki do sektorów
Inne zagadnienia	

## Planowanie dostępu do dysku

Czas dostępu do danych na dysku zależy od trzech składowych: czasu potrzebnego na ustawienie głowicy nad odpowiednią ścieżką, nazywanego *czasem przeszukiwania* (ang. seek time), czasu oczekiwania na pojawienie się pod głowicą odpowiedniego sektora nazywanego *czasem oczekiwania* (ang. latency time) oraz czasu poświęconego na transfer informacji, nazywanego *czasem przesłania*. Najbardziej znaczącą (i jedyną na którą możemy mieć wpływ) jest składowa związana z czasem przeszukiwania. Metody dostępu pozwalają zminimalizować ten czas.

## Metoda FCFS

Metoda FCFS może być użyta nie tylko do szeregowania procesów, ale również do szeregowania żądań dostępu do dysku. Jest ona łatwa w implementacji, ale niestety daje niedobre rezultaty, prowadzi do znacznych wychyłów głowicy dysku, nie tylko powodując duże czasy przeszukiwania, ale również negatywnie wpływając na żywotność mechanizmu pozycjonowania głowicy.

## Metoda SSTF

Metoda SSTF (ang. Shortest Seek-Time-First) polega na dynamiczny wybieraniu spośród zleceń tego, które dotyczy ścieżki położonej najbliżej bieżącego położenia głowicy. Przeliczanie odległości następuje po każdym przesunięciu głowicy. Metoda ta jest optymalna, ale może prowadzić do zagłodzenia żądań.

## Metoda SCAN

W tej metodzie głowica dysku porusza się jednostajnym ruchem od brzegu talerza, ku centrum i z powrotem. Podczas tego ruchu realizuje napływające zamówienia. Jeśli zamówienie jest zgodne z obecnym jej kierunkiem ruchu, to jest realizowane, jeśli nie, to jest realizowane dopiero po zmianie kierunku. Przypomina to odśnieżanie drogi: pług porusza się w jednym kierunku usuwając śnieg, a z tyłu płyty pada śnieg, który zostanie przez niego usunięty, kiedy będzie jechał w przeciwnym kierunku.

Plan wykładu	Metoda FCFS
Wprowadzenie	Metoda SSTF
Katalog urządzenia	Metoda SCAN
Zarządzanie obszarami wolnymi	<b>Metoda C-SCAN</b>
Przydział miejsca na dysku	Metoda LOOK i C-LOOK
Planowanie dostępu do dysku	Kolejki do sektorów
Inne zagadnienia	

## Metoda C-SCAN

Metoda ta jest drobną modyfikacją metody SCAN. Główica po osiągnięciu środka dysku wraca natychmiast do jego krawędzi, nie realizując przy tym żadnych zleceń. Zlecenia są realizowane wyłącznie podczas ruchu głowicy od krawędzi do środka talerza.

## Metoda LOOK i C-LOOK

Metody te są modyfikacjami metod SCAN i C-SCAN. Główica podąża w jednym kierunku dotąd, dopóki są zamówienia do ścieżek położonych przed nią. Jeśli tych zamówień zabraknie, to głowica zmienia kierunek ruchu i zależnie od metody wykonuje jałowy ruch w kierunku krawędzi talerza lub podejmuje realizację nowych zleceń.

Plan wykładu	Metoda FCFS
Wprowadzenie	Metoda SSTF
Katalog urządzeń	Metoda SCAN
Zarządzanie obszarami wolnymi	Metoda C-SCAN
Przydział miejsca na dysku	Metoda LOOK i C-LOOK
Planowanie dostępu do dysku	
Inne zagadnienia	Kolejki do sektorów

## Kolejki do sektorów

W przypadku niektórych dysków opłacalne jest utrzymywanie kolejek żądań dostępu do sektorów, tak aby można było jednorazowo odczytać lub zapisać wszystkie żądane sektory, kiedy głowica znajduje się nad ścieżką je zawierającą. Ta technika wywodzi się z bębnów magnetycznych. Obecnie, kiedy stosowane są dyski z dużą pamięcią podręczną (ang. cache memory), pozwalającą zapamiętać nawet do kilku takich ścieżek, przydatność tej metody jest znikoma.

## Efektywność i niezawodność

Dyski twarde składają się z części elektronicznej stanowiącej ich sterownik, oraz części mechanicznej. Aby zwiększyć niezawodność dysków dubluje się sterowniki, a nawet dubluje się całe dyski, które pracują w trybie *odzwierciedlenia* (ang. mirroring). Do zwiększenia szybkości działania dysków używano techniki nazywanej *podziałem na paski* (ang. stripping). Polegała ona na podziale pojedynczego bloku na podbloki, które były umieszczane na osobnych dyskach, tak aby można je było odczytać równolegle (równocześnie) w jednym żądaniu. Technika ta stała się podstawą do budowy macierzy RAID (ang. Redundant Array of Inexpensive Disks) - nadmiarowej macierzy niedrogich dysków. Z czasem akronim RAID zaczęto tłumaczyć jako nadmiarowa macierz niezależnych dysków (ang. Redundant Array of Independent Disks).

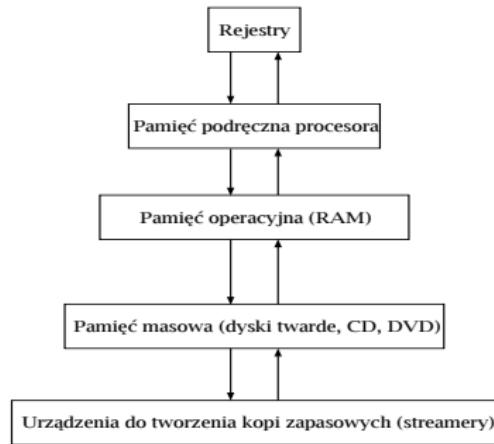
# Macierze RAID

Typy macierzy RAID:

- RAID 0 podział bloków na paski rozmieszczone na kilku dyskach,
- RAID 1 podział na paski i zdublowanie każdego dysku,
- RAID 2 podział odbywa się na poziomie bajtów; na każdym dysku zapisywany jest jeden bit słowa opatrzonego kodem detekcyjno-korekcyjnym Hamminga; wymaga dużej liczby dysków i synchronizacji ich głowic,
- RAID 3 uproszczona wersja RAID 2, w której do każdego słowa dołączany jest bit parzystości, zapisywany na wyznaczonym do tego dysku,
- RAID 4 jak wyżej, ale parzystość jest obliczana na poziomie pasków, nie wymaga synchronizacji,
- RAID 5 paski parzystości nie są przechowywane na pojedynczym dysku, a zapisywane z przeplotem (ang. interleaving), czyli rozproszone po wszystkich dyskach macierzy.

Niektóre poziomy macierzy RAID mogą być ze sobą kombinowane.

## Hierarchia pamięci



Ilustracja obok przedstawia hierarchię pamięci systemu komputerowego. Na szczytce znajduje się pamięć, o najkrótszym czasie dostępu, a zarazem o najmniejszej pojemności. Są nią rejestrzy procesora. Na dole hierarchii znajduje się pamięć w postaci taśm magnetycznych, o bardzo dużej pojemności, ale za to o bardzo długim czasie dostępu. Poziomy pośrednie tworzy pamięć podręczna procesora, która odwzorowuje część pamięci operacyjnej, pamięć operacyjną i pamięć pomocniczą. Po analizie rysunku można dojść do wniosku, że stan pamięci danego poziomu jest podzbiorem stanu pamięci znajdującej się od niej niżej w hierarchii.

Plan wykładu

Wprowadzenie

Katalog urządzeń

Zarządzanie obszarami wolnymi

Przydział miejsca na dysku

Planowanie dostępu do dysku

Inne zagadnienia

Efektywność i niezawodność

Hierarchia pamięci

## Pytania

?

**Koniec**

Dziękuję Państwu za uwagę!

# Systemy Operacyjne — Systemy plików

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 2 stycznia 2021

# Plan wykładu

## ① Wstęp

## ② Pliki

- ③ Definicja pliku
- ④ Typ pliku
- ⑤ Operacje na plikach
- ⑥ Metody dostępu
- ⑦ Semantyka integralności

## ③ Katalogi

- ⑧ Katalogi fizyczne i logiczne
- ⑨ Wykaz pozycji
- ⑩ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ③ Definicja pliku
- ④ Typ pliku
- ⑤ Operacje na plikach
- ⑥ Metody dostępu
- ⑦ Semantyka integralności

## ③ Katalogi

- ⑧ Katalogi fizyczne i logiczne
- ⑨ Wykaz pozycji
- ⑩ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

### ① Definicja pliku

- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

### ① Definicja pliku

### ② Typ pliku

### ③ Operacje na plikach

### ④ Metody dostępu

### ⑤ Semantyka integralności

## ③ Katalogi

### ① Katalogi fizyczne i logiczne

### ② Wykaz pozycji

### ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
  - ④ Metody dostępu
  - ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Plan wykładu

## ① Wstęp

## ② Pliki

- ① Definicja pliku
- ② Typ pliku
- ③ Operacje na plikach
- ④ Metody dostępu
- ⑤ Semantyka integralności

## ③ Katalogi

- ① Katalogi fizyczne i logiczne
- ② Wykaz pozycji
- ③ Struktury katalogów

## ④ Ochrona plików

# Wstęp

Sposób przechowywania informacji jest specyficzny dla każdego rodzaju pamięci<sup>1</sup>, dla tego też systemy operacyjne oferują specjalne struktury danych nazywane plikami. Pliki pozwalają uzyskać dostęp do informacji w ten sam sposób, niezależnie od tego gdzie ta informacja jest zapisana. Aby ułatwić zarządzanie plikami systemy operacyjne utrzymują informację o nich w katalogach, które z kolei są organizowane w struktury katalogów. Struktury katalogów wraz z plikami tworzą *system plików*. System plików jest najbardziej dostrzegalną dla użytkownika częścią systemu operacyjnego. Często systemy operacyjne obsługują więcej niż jeden system plików.

---

<sup>1</sup>Proszę porównać np. dysk CD i dysk twardy.

## Definicja pliku

*Plik* jest abstrakcyjnym typem danych dostarczany przez system operacyjny w celu umożliwienia spójnej metody dostępu do informacji umiejscowionych w pamięciach różnego typu. Pliki posiadają szereg cech, z których najważniejszą, dla użytkownika jest jednoznacznie je identyfikująca nazwa. Format pliku może być swobodny lub ściśle określony. Do plików o swobodnym formacie należą pliki tekstowe, w których dane są przechowywane w postaci ciągów znaków zakończonych znakiem(-kami) końca wiersza. Pliki o określonym formacie zawierają najczęściej numeryczne dane binarne lub rekordy danych. Ogólnie, pliki mogą przechowywać programy, zarówno w postaci źródłowej, jak i wynikowej oraz dane. Informacje w plikach umieszczane są porcjami, które nazywamy *rekordami logicznymi*. Rozmiar takiego rekordu jest zmienny, w szczególności może on wynosić jeden bajt. Ponieważ rekordy logiczne są rozmieszczane w blokach alokacji, a rzadko się zdarza, aby rozmiar rekordu logicznego był wielokrotnością rozmiaru bloku alokacji, to jest to przyczyną powstawania fragmentacji.

## Typy plików

*Typ pliku* jest cechą pliku, która określa jego strukturę. Podczas projektowania systemu operacyjnego należy określić w jaki sposób i czy w ogóle będzie on interpretował typy plików. Jednym z najciekawszych przykładów systemów, które rozpoznają typ pliku jest TOPS-20. W tym systemie wraz z plikiem zawierającym kod wynikowy programu znajdowały się również pliki z kodem źródłowym. Jeśli data modyfikacji pliku wynikowego lub jego wersja była późniejsza niż plików z kodem źródłowym, to system operacyjny przed uruchomieniem programu wywoływał kompilator, który generował nowszą wersję wynikową programu. Po aktualizacji program był ostatecznie wykonywany. Pozwolenie systemowi operacyjnemu na rozpoznawanie typów plików wiąże się z koniecznością umożliwienia mu definiowania nowych typów plików. Przykładem systemów dysponujących taką cechą są systemy rodziny Windows, w których użytkownik może skojarzyć określoną akcję z konkretnym typem pliku. Inne podejście może zakładać, że system operacyjny w ogóle nie będzie interpretował typów plików. Takie podejście zastosowano w systemie Unix. Zawartość plików w tym systemie jest traktowana jako ciąg bajtów, a ich interpretacja jest pozostawiana aplikacjom użytkownika.

## Operacje na plikach

Ponieważ plik jest abstrakcyjną strukturą danych, to system operacyjny musi dostarczyć definicji operacji, które mogą być na niej wykonywane. Istnieje pięć takich operacji, które są uznawane za podstawowe: *tworzenie pliku*, *zapis pliku*, *odczyt pliku*, *ustawienie wskaźnika pliku*, *usunięcie pliku*. Wszystkie te operacje wiążą się z przeszukiwaniem katalogu i znalezieniem pozycji odpowiadającej plikowi, na którym operacja ma zostać przeprowadzona (w przypadku tworzenia pliku musi to być pozycja pusta). Aby wyeliminować przeszukiwanie katalogu z tych operacji wprowadza się dodatkową operację nazywaną *otwieraniem pliku*. Jeśli plik jest otwierany, to system operacyjny tworzy w pamięci operacyjnej kopię jego wpisu w katalogu. Operacją uzupełniającą do otwierania jest *zamykanie pliku*, które powoduje zapisanie wpisu z pamięci operacyjnej na dysk, jeśli te wersje się różnią i usunięcie wpisu z pamięci. Niektóre systemy dokonują niejawnego otwierania i zamykania plików, ale większość wymaga jawnego użycia przez aplikację użytkownika odpowiednich wywołań systemowych. Ze wskaźnika pliku, oprócz operacji jego ustawiania korzystają również operacje odczytu i zapisu. Wartość początkowa wskaźnika danego pliku może być zapisywana w jego pozycji katalogu, ale najczęściej jest przy otwieraniu ustawiana na zero (początek pliku). Wymienione na początku operacje są operacjami podstawowymi. Część innych operacji możemy stworzyć używając tych podstawowych, jak np. kopiowanie plików. Inne jak np.: zmiana nazwy pliku muszą być wykonywane przez system operacyjny. Systemy wielozadaniowe dostarczają także operacji blokowania (ang. lock) części lub całości plików, celem ochrony ich przed współbieżną modyfikacją.

## Metody dostępu

Dostęp do informacji zawartych w pliku może być wykonywany na kilka sposobów, ale najpopularniejsze są dwa. Od metody dostępu do informacji zawartych w pliku zależy efektywność pracy z pamięcią masową. Niektóre systemy operacyjne pozwalają korzystać z tylko jednej metody dostępu do pliku, inne z kilku.

## Dostęp sekwencyjny

*Dostęp sekwencyjny* jest metodą dostępu opartą na taśmowym modelu pliku. Wskaźnik pliku jest zawsze zwiększany, czyli można „poruszać się” jedynie „do przodu” pliku. Większość systemów pozwala zwiększać wskaźnik o jeden, czyli możliwy jest odczyt lub zapis kolejnych bloków alokacji pliku. Istnieją jednak systemy, w których można zwiększać wartość wskaźnika o pewną ustaloną wartość  $n$ , która może być większa od jeden.

## Dostęp swobodny

W *dostępie swobodnym* zwanym również *dostępnem bezpośrednim* wskaźnik pliku może być zwiększany lub zmniejszany o dowolną wielkość. Pozwala to na zapis lub odczyt dowolnego bloku alokacji pliku bez konieczności odczytywania jego poprzedników. Ten sposób jest szybszy od dostępu sekwencyjnego. Dodatkową jego zaletą jest, że umożliwia również sekwencyjny odczyt/zapis pliku. Wystarczy, że te operacje będą wykonywane na kolejnych blokach alokacji pliku.

## Dostęp indeksowy

Ten rodzaj dostępu jest modyfikacją dostępu bezpośredniego. Nadaje się do zastosowania w systemach plików tworzonych na potrzeby baz danych. W *dostępie indeksowym* z każdym dużym plikiem kojarzony jest mały plik, którego zawartość pełni podobną rolę, jak rola indeksu w książkach. Ułatwia ona wyszukiwanie informacji w dużym pliku. W małym pliku zawarte są hasła związane z wyszukiwanymi informacjami oraz numery bloków alokacji dużego pliku, od których rozpoczyna się ciąg bloków w których te informacje są zapisane.

## Semantyka spójności

*Semantyka spójności* nazywana również *semantyką spoistości* lub *semantyką integralności* jest ważnym pojęciem określającym reguły dostępu do plików współdzielonych w systemach wielozadaniowych, dzięki którym zmiany w takich plikach mogą być dokonywane w sposób bezpieczny. Semantyka ta określa również zakres wzajemnej wiadomości zmian jakich dokonują współbieżnie użytkownicy w pliku. Następne slajdy zawierają opis trzech semantyk spójności, pochodzących z różnych systemów operacyjnych. Wszystkie operacje dokonywane na pliku, wraz z jego otwarciem i zamknięciem będą określane w tych opisach mianem *sesji plikowej*.

# Semantyka spójności systemu Unix

Semantyka ta definiuje dwie reguły:

- ① Jeśli plik jest współdzielony przez kilku użytkowników (np. przez kilka procesów) i jeden z nich dokona modyfikacji zawartości tego pliku, to jej skutki są od razu widoczne dla pozostałych użytkowników.
- ② Możliwe jest użycie trybu współdzielenia, w którym użytkownicy dzielą wskaźnik pliku. Modyfikacja jego zawartości przez któregoś z użytkowników ma wpływ na operacje wykonywane na pliku przez pozostałych użytkowników.

W semantyce spójności systemu Unix, zasób jakim jest plik ma tylko jeden obraz, widoczny dla każdego z użytkowników. Semantyka ta powoduje opóźnienia w realizacji operacji wykonywanych na pliku, ze względu na konieczność modyfikacji (zapisu) współdzielonego pliku na zasadach wyłączności.

# Semantyka spójności systemu Unix

Semantyka ta definiuje dwie reguły:

- ① Jeśli plik jest współdzielony przez kilku użytkowników (np. przez kilka procesów) i jeden z nich dokona modyfikacji zawartości tego pliku, to jej skutki są od razu widoczne dla pozostałych użytkowników.
- ② Możliwe jest użycie trybu współdzielenia, w którym użytkownicy dzielą wskaźnik pliku. Modyfikacja jego zawartości przez któregoś z użytkowników ma wpływ na operacje wykonywane na pliku przez pozostałych użytkowników.

W semantyce spójności systemu Unix, zasób jakim jest plik ma tylko jeden obraz, widoczny dla każdego z użytkowników. Semantyka ta powoduje opóźnienia w realizacji operacji wykonywanych na pliku, ze względu na konieczność modyfikacji (zapisu) współdzielonego pliku na zasadach wyłączności.

# Semantyka spójności systemu Unix

Semantyka ta definiuje dwie reguły:

- ❶ Jeśli plik jest współdzielony przez kilku użytkowników (np. przez kilka procesów) i jeden z nich dokona modyfikacji zawartości tego pliku, to jej skutki są od razu widoczne dla pozostałych użytkowników.
- ❷ Możliwe jest użycie trybu współdzielenia, w którym użytkownicy dzielą wskaźnik pliku. Modyfikacja jego zawartości przez któregoś z użytkowników ma wpływ na operacje wykonywane na pliku przez pozostałych użytkowników.

W semantyce spójności systemu Unix, zasób jakim jest plik ma tylko jeden obraz, widoczny dla każdego z użytkowników. Semantyka ta powoduje opóźnienia w realizacji operacji wykonywanych na pliku, ze względu na konieczność modyfikacji (zapisu) współdzielonego pliku na zasadach wyłączności.

## Semantyka sesji plikowej

Ta semantyka została zastosowana po raz pierwszy w systemie Andrew. Definiowała ona następujące reguły:

- 1 Jeśli kilku użytkowników ma otwarty plik współdzielony i jeden z nich dokona modyfikacji jego zawartości, to jej skutki nie są widoczne natychmiast dla pozostałych.
- 2 Zmiany w pliku są widoczne tylko dla tych użytkowników, którzy zamknęli plik i ponownie go otwarli.

Te reguły wymagają, aby każdy z użytkowników pliku otrzymywał jego kopię, unikalny obraz tego pliku dostępny dla każdego z nich osobno.

## Semantyka sesji plikowej

Ta semantyka została zastosowana po raz pierwszy w systemie Andrew. Definiowała ona następujące reguły:

- ➊ Jeśli kilku użytkowników ma otwarty plik współdzielony i jeden z nich dokona modyfikacji jego zawartości, to jej skutki nie są widoczne natychmiast dla pozostałych.
- ➋ Zmiany w pliku są widoczne tylko dla tych użytkowników, którzy zamknęli plik i ponownie go otwarli.

Te reguły wymagają, aby każdy z użytkowników pliku otrzymywał jego kopię, unikalny obraz tego pliku dostępny dla każdego z nich osobno.

## Semantyka sesji plikowej

Ta semantyka została zastosowana po raz pierwszy w systemie Andrew. Definiowała ona następujące reguły:

- ① Jeśli kilku użytkowników ma otwarty plik współdzielony i jeden z nich dokona modyfikacji jego zawartości, to jej skutki nie są widoczne natychmiast dla pozostałych.
- ② Zmiany w pliku są widoczne tylko dla tych użytkowników, którzy zamknęli plik i ponownie go otwarli.

Te reguły wymagają, aby każdy z użytkowników pliku otrzymywał jego kopię, unikalny obraz tego pliku dostępny dla każdego z nich osobno.

## Semantyka niezmiennych plików współdzielonych

Tę semantykę stosuje się głównie w systemach rozproszonych. Jeśli plik jest współdzielony, to możliwy jest tylko jego odczyt, nie można zaś zmodyfikować jego zawartości. Tak, jak zazwyczaj plik jest udostępniony użytkownikom za pomocą jego nazwy, ale w tym wypadku nazwa odnosi się nie do pliku w tradycyjnym znaczeniu, ale do jego określonej zawartości, która nie może podlegać zmianom.

## Katalogi fizyczne i logiczne

Na poprzednim wykładzie wprowadzone zostało pojęcie katalogu fizycznego, czyli katalogu głównego urządzenia. Ponieważ jego pojemność (liczba wpisów) jest ograniczona, to w systemach operacyjnych są definiowane *katalogi logiczne*. Katalog logiczny jest strukturą, której zawartość stanowią wykazy informacji o innych plikach<sup>2</sup>, czyli tak zwane *metadane plików*. Tradycyjnie katalogi logiczne najczęściej są implementowane jako pliki, choć część systemów operacyjnych stosuje inne rozwiązania. Do metadanych plików umieszczonych w katalogach mogą zaliczać się: unikatowa *nazwa pliku*, informacja o *typie pliku* (np.: trzyliterowe rozszerzenie nazwy), *wskaźnik pliku*, *rozmiar pliku* (w bajtach lub jednostkach alokacji), *lokalizacja pliku*, czyli np. numer pierwszego bloku alokacji przedzielonego plikowi, *informacje o ochronie*, czyli o trybie dostępu, *licznik użycia*, określający ilu użytkowników korzysta w danej chwili z pliku, lub ile istnieje tzw. dowiązań do pliku, *identyfikator właściciela i/lub twórcy pliku* oraz *czasy utworzenia, ostatniej modyfikacji i/lub ostatniego dostępu* do pliku. Katalogi tworzą *strukturę katalogów*. Taka struktura może obejmować nie tylko jedno urządzenie fizyczne, ale również kilka takich urządzeń (np. struktura katalogów systemu Unix). Dzięki temu system operacyjny ukrywa przed użytkownikiem faktyczną lokalizację plików.

---

<sup>2</sup>Wpisy tego samego typu co w katalogu fizycznym.

## Wykaz pozycji

**Katalog jest zbiorem pozycji, które zawierają metadane plików. Od tego w jaki sposób zostanie ten wykaz pozycji zaimplementowany zależy szybkość odnajdywania informacji o plikach, a także usuwania i tworzenia plików. Oto niektóre z możliwych realizacji:**

- **Tablica**-wymaga liniowego czasu przeszukiwania. Pozycja pliku, który został usunięty może być oznaczana w specjalny sposób jako pusta i później użyta do utworzenia nowego pliku. Jeśli nie ma pustych pozycji, to informacje o nowym pliku są umieszczane w pierwszej wolnej pozycji wykazu. Aby przyspieszyć wyszukiwanie można zastosować algorytm wyszukiwania binarnego, ale to wymaga utrzymywania wykazu plików w stanie posortowanym.
- **Lista powiązana**-również wymaga liniowego czasu przeszukiwania, ale operacje wstawiania i usuwania pozycji we wpisie są łatwiejsze do przeprowadzenia, w szczególności łatwiej jest utrzymywać wykaz w stanie posortowania.
- **Drzewo binarne powiązane**-wymaga logarytmicznego czasu przeszukiwania
- **Tablica z haszowaniem**(ang. hash table)-pozwala na szybsze wyszukiwanie, jak również tworzenie i usuwanie plików, niż zwykła tablica. Problemem jest jednak zmiana funkcji haszującej, której postać jest uzależniona od wielkości tej struktury.

## Wykaz pozycji

Katalog jest zbiorem pozycji, które zawierają metadane plików. Od tego w jaki sposób zostanie ten wykaz pozycji zaimplementowany zależy szybkość odnajdywania informacji o plikach, a także usuwania i tworzenia plików. Oto niektóre z możliwych realizacji:

- **Tablica**-wymaga liniowego czasu przeszukiwania. Pozycja pliku, który został usunięty może być oznaczana w specjalny sposób jako pusta i później użyta do utworzenia nowego pliku. Jeśli nie ma pustych pozycji, to informacje o nowym pliku są umieszczane w pierwszej wolnej pozycji wykazu. Aby przyspieszyć wyszukiwanie można zastosować algorytm wyszukiwania binarnego, ale to wymaga utrzymywania wykazu plików w stanie posortowanym.
- **Lista powiązana**-również wymaga liniowego czasu przeszukiwania, ale operacje wstawiania i usuwania pozycji we wpisie są łatwiejsze do przeprowadzenia, w szczególności łatwiej jest utrzymywać wykaz w stanie posortowania.
- **Drzewo binarne** powiązane-wymaga logarytmicznego czasu przeszukiwania
- **Tablica z haszowaniem**(ang. hash table)-pozwala na szybsze wyszukiwanie, jak również tworzenie i usuwanie plików, niż zwykła tablica. Problemem jest jednak zmiana funkcji haszującej, której postać jest uzależniona od wielkości tej struktury.

## Wykaz pozycji

Katalog jest zbiorem pozycji, które zawierają metadane plików. Od tego w jaki sposób zostanie ten wykaz pozycji zaimplementowany zależy szybkość odnajdywania informacji o plikach, a także usuwania i tworzenia plików. Oto niektóre z możliwych realizacji:

- **Tablica**-wymaga liniowego czasu przeszukiwania. Pozycja pliku, który został usunięty może być oznaczana w specjalny sposób jako pusta i później użyta do utworzenia nowego pliku. Jeśli nie ma pustych pozycji, to informacje o nowym pliku są umieszczane w pierwszej wolnej pozycji wykazu. Aby przyspieszyć wyszukiwanie można zastosować algorytm wyszukiwania binarnego, ale to wymaga utrzymywania wykazu plików w stanie posortowanym.
- **Lista powiązana**-również wymaga liniowego czasu przeszukiwania, ale operacje wstawiania i usuwania pozycji we wpisie są łatwiejsze do przeprowadzenia, w szczególności łatwiej jest utrzymywać wykaz w stanie posortowania.
- **Drzewo binarne powiązane**-wymaga logarytmicznego czasu przeszukiwania
- **Tablica z haszowaniem**(ang. hash table)-pozwala na szybsze wyszukiwanie, jak również tworzenie i usuwanie plików, niż zwykła tablica. Problemem jest jednak zmiana funkcji haszującej, której postać jest uzależniona od wielkości tej struktury.

## Wykaz pozycji

Katalog jest zbiorem pozycji, które zawierają metadane plików. Od tego w jaki sposób zostanie ten wykaz pozycji zaimplementowany zależy szybkość odnajdywania informacji o plikach, a także usuwania i tworzenia plików. Oto niektóre z możliwych realizacji:

- **Tablica**-wymaga liniowego czasu przeszukiwania. Pozycja pliku, który został usunięty może być oznaczana w specjalny sposób jako pusta i później użyta do utworzenia nowego pliku. Jeśli nie ma pustych pozycji, to informacje o nowym pliku są umieszczane w pierwszej wolnej pozycji wykazu. Aby przyspieszyć wyszukiwanie można zastosować algorytm wyszukiwania binarnego, ale to wymaga utrzymywania wykazu plików w stanie posortowanym.
- **Lista powiązana**-również wymaga liniowego czasu przeszukiwania, ale operacje wstawiania i usuwania pozycji we wpisie są łatwiejsze do przeprowadzenia, w szczególności łatwiej jest utrzymywać wykaz w stanie posortowania.
- **Drzewo binarne powiązane**-wymaga logarytmicznego czasu przeszukiwania
- **Tablica z haszowaniem**(ang. hash table)-pozwala na szybsze wyszukiwanie, jak również tworzenie i usuwanie plików, niż zwykła tablica. Problemem jest jednak zmiana funkcji haszującej, której postać jest uzależniona od wielkości tej struktury.

## Wykaz pozycji

Katalog jest zbiorem pozycji, które zawierają metadane plików. Od tego w jaki sposób zostanie ten wykaz pozycji zaimplementowany zależy szybkość odnajdywania informacji o plikach, a także usuwania i tworzenia plików. Oto niektóre z możliwych realizacji:

- **Tablica**-wymaga liniowego czasu przeszukiwania. Pozycja pliku, który został usunięty może być oznaczana w specjalny sposób jako pusta i później użyta do utworzenia nowego pliku. Jeśli nie ma pustych pozycji, to informacje o nowym pliku są umieszczane w pierwszej wolnej pozycji wykazu. Aby przyspieszyć wyszukiwanie można zastosować algorytm wyszukiwania binarnego, ale to wymaga utrzymywania wykazu plików w stanie posortowanym.
- **Lista powiązana**-również wymaga liniowego czasu przeszukiwania, ale operacje wstawiania i usuwania pozycji we wpisie są łatwiejsze do przeprowadzenia, w szczególności łatwiej jest utrzymywać wykaz w stanie posortowania.
- **Drzewo binarne powiązane**-wymaga logarytmicznego czasu przeszukiwania
- **Tablica z haszowaniem**(ang. hash table)-pozwala na szybsze wyszukiwanie, jak również tworzenie i usuwanie plików, niż zwykła tablica. Problemem jest jednak zmiana funkcji haszującej, której postać jest uzależniona od wielkości tej struktury.

## Katalog jednopoziomowy

*Katalog jednopoziomowy* jest najprostszym przykładem struktury katalogowej. Do jej realizacji wystarczy jedynie katalog urządzenia. Ponieważ wszystkie pliki w katalogu muszą mieć unikatową nazwę, a dodatkowo część systemów operacyjnych ogranicza długość nazwy pliku, to katalogi jednopoziomowe mają ograniczone zastosowanie. W szczególności nie mogą one być stosowane w systemach wielodostępnych. Zastosowanie takiej struktury w obecnych systemach jest bardzo rzadkie.

## Katalog dwupoziomowy

*Struktura katalogu dwupoziomowego* pozwala na tworzenie katalogów logicznych, które pełnią rolę *katalogów domowych* użytkowników (ang. home directory). Tak więc każdy użytkownik posiada własny katalog, w którym może przechowywać własne pliki. Wykonanie unikalności nazwy pliku obowiązuje tylko w obrębie katalogu, a więc jeśli jeden z użytkowników nazwie swój plik, tak jak inny użytkownik, to nie dochodzi do konfliktu nazw. Katalog dwupoziomowy pozwala również korzystać użytkownikom z plików innych użytkowników. Wystarczy jeśli poprzedzą oni nazwę pliku nazwą katalogu użytkownika, w którym ten plik się znajduje. Taka konstrukcja (nazwa katalogu razem z nazwą pliku) nazywa się *ścieżką* (ang. path). Oprócz katalogów użytkowników w katalogu głównym tworzony jest katalog zawierający oprogramowanie systemowe. Aby uniknąć konieczności podawania przez użytkowników ścieżki do plików znajdujących się w katalogu systemowym w systemach operacyjnych stosuje się *ścieżkę wyszukiwania* (ang. search path), która określa kolejność w jakiej system przeszukuje katalogi w poszukiwaniu żądanego przez użytkownika pliku. Jeśli w tej ścieżce na pierwszej pozycji wymieniony jest katalog systemowy, to jest on najpierw przeszukiwany, a następnie jest przeszukiwany katalog użytkownika.

## Drzewo katalogów

Aby użytkownik mógł posiadać pewną liczbę plików o takich samych nazwach oraz aby mógł lepiej zarządzać swoimi plikami należy pozwolić mu na zakładanie własnych katalogów, które mogą tworzyć wielopoziomową strukturę. Ta struktura ma postać drzewa. W drzewie katalogów występuje pojęcie *katalogu bieżącego* (ang. working directory lub current directory). Jest to katalog, który jest w danej chwili używany przez użytkownika (np. proces). Użytkownik ma możliwość poruszania się po strukturze katalogów, gdyż każdy katalog ma dwie specjalne pozycje, które stanowią dowiązanie do samego siebie (nazywane najczęściej za pomocą jednej kropki) i dowiązanie do katalogu znajdującego się bezpośrednio wyżej w hierarchii (nazywane najczęściej za pomocą dwóch kropek). W drzewie katalogów występuje również pojęcie *ścieżki względnej*, która określa położenie pliku względem katalogu bieżącego i *ścieżki bezwzględnej* określającej położenie pliku względem katalogu głównego. Stosowane są dwa podejścia odnośnie usuwania katalogów. W pierwszym zakłada się, że jeśli użytkownik chce skasować katalog, to jego zawartość, wraz z podkatalogami również należy skasować i wykonuje się ich rekurencyjne usuwanie. W drugim podejściu system odmawia skasowania katalogu jeśli nie jest on pusty.

## Graf katalogów

Aby ułatwić użytkownikom współdzielenie pliku, czyli współbieżny dostęp do jego zawartości, wprowadzane są *dowiązania* (ang. link). Wprowadzenie dowiązań przekształca drzewo katalogów w *graf katalogów*. Istnieją dwa rodzaje dowiązań: *dowiązania sztywne* lub *twarde* (ang. hard links) i *dowiązania symboliczne* (ang. symbolic links). Tworzenie dowiązania twardego sprowadza się do skopiowania pozycji z katalogu źródłowego do docelowego. Zarówno oryginał jak i kopia są nie do odróżnienia, co może sprawiać problemy przy przeszukiwaniu katalogu (niekończące się pętle) i przy usuwaniu pliku. Jeśli usuniemy plik za pomocą dowiązania, to zostaje oryginalny wpis, odwrotna sytuacja również jest możliwa. Częściowym rozwiązaniem tego problemu może być związanie z zawartością pliku *wykazu dowiązań* lub *licznika dowiązań* i usuwanie oryginalnego pliku tylko wtedy gdy taki wykaz jest pusty lub gdy licznik dowiązań jest równy zero. Wymienionych wad są w większości pozbawione dowiązania symboliczne, które są specjalnymi pozycjami w katalogu będącymi wskaźnikami na nazwę oryginalnego pliku. Dzięki temu można wypisać nazwę dowiązania przy przeszukiwaniu katalogu, ale go nie rozwiązywać (ang. resolve), co zapobiega powstawaniu nieskończonych pętli. Operacja usunięcia dowiązania zniszczy tylko dowiązanie, oryginalny plik pozostanie w systemie plików. Problem usunięcia pliku jest również częściowo rozwiązany - jeśli zniknie pozycja pliku, to również jego nazwa, na którą wskazuje dowiązanie. Wielu problemów unika się jeśli graf katalogów jest acykliczny. Można to zapewnić wykonując operacje wykrywania odwołań cyklicznych połączoną z odśmiecaniem (ang. garbage collection). Jest to jednak czasochłonna operacja.

## Ochrona plików

Są dwa główne cele ochrony plików - ochrona przed zniszczeniem, którą zapewnia się wprowadzając mechanizmy automatycznego tworzenia kopii zapasowych oraz zapewnienie poufności ich treści. To ostatnie zadanie wymaga zapewnienia kompromisu pomiędzy możliwością współdzielenia plików przez użytkowników, a koniecznością ograniczenia dostępu do nich. Ograniczenia te dotyczą operacji wykonywanych na plikach (odczyt, zapis, wykonanie, modyfikacja, dopisywanie) oraz operacji wykonywanych na katalogach (przeglądanie, poruszanie się po strukturze katalogów, usuwanie i tworzenie plików). Ochrona może dotyczyć zawartości pliku lub ścieżki dostępu do pliku. Częściej stosuje się to drugie rozwiązanie. Istnieje również wiele sposobów implementacji tej ochrony.

## Nazywanie

Najprostszy sposób ochrony plików polega na zakazaniu wyświetlania nazw plików znajdujących się w katalogach. Wówczas ma do nich dostęp jedynie użytkownik, który te nazwy zna. Ten rodzaj ochrony jest niewygodny i zawodny. Ponieważ długość nazw plików jest ograniczana przez systemy, a użytkownicy tworzą nazwy mnemotechniczne, to łatwo jest intruzowi taką nazwę ustalić.

## Hasła

Pliki mogą być chronione przez system *haseł*. Jest on bardziej bezpieczny, niż ukrywanie nazw, ale posiada swoje wady. Jeśli z każdym plikiem zwiążemy hasło, to liczba haseł, którą musi zapamiętać użytkownik staje się bardzo duża. Jeśli z drugiej strony zastosujemy jedno hasło do wszystkich plików, lub do wydzielonej grupy plików, to złamanie tego hasła umożliwi intruzowi dostęp do dużej ilości informacji zawartej w systemie. Ochrona za pomocą haseł nie pozwala również na ograniczenie dostępu do pliku. Albo dajemy użytkownikowi całkowity dostęp do zawartości pliku, albo w ogóle go zabraniamy.

## Wykaz dostępów

Z każdym plikiem lub katalogiem można związać *wykaz dostępów* (ang. access control list), czyli opis praw jakie przysługują poszczególnym użytkownikom do tego pliku (czy użytkownik może dokonywać zapisu, odczytu itd.) po ich uwierzytelnieniu przez system operacyjny. Jeśli użytkownik próbuje wykonać jakąś operację, to system operacyjny odnajduje jego identyfikator w wykazie dostępów i sprawdza jakie prawa mu przysługują. Jeśli w wykazie znajduje się operacja, którą użytkownik chce wykonać, to jest ona wykonywana, w przeciwnym razie system odmawia dostępu użytkownikowi. Wadą tego rozwiązania może okazać się wielkość takiego wykazu.

## Dostęp grupowy

W *dostępie grupowym* z każdym plikiem lub katalogiem związanych jest dziewięć bitów. Każda trójka bitów opisuje prawa dostępu do pliku pozwalające na odczyt, zapis i wykonanie (rwx w notacji uniksowej). Również znaczenie każdej z tych trójkę jest inne. Pierwsza opisuje prawa właściciela pliku (user w notacji uniksowej), druga prawa grupy użytkowników do której należy właściciel pliku (group w notacji uniksowej), a trzecia określa prawa dostępu dla pozostałych użytkowników w systemie (others w notacji uniksowej). Grupy użytkowników są zdefiniowane w specjalnie chronionym pliku systemowym. Dzięki *dostępowi grupowemu* możliwe jest współdzielenie plików. Również ilość informacji związanych z ochroną plików, które trzeba zapamiętać jest mała w stosunku do wykazu dostępów.

# Pytania

?

Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne — Ochrona

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 18 stycznia 2021

## Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ③ Ochrona
  - ④ Bezpieczeństwo
  - ⑤ Polityka bezpieczeństwa i mechanizmy
  - ⑥ Domeny i wiedza konieczna
- ⑦ Macierze dostępów
  - ⑧ Tablica globalna
  - ⑨ Wykaz dostępów
  - ⑩ Wykaz możliwości
  - ⑪ Mechanizm zamka z kluczem
- ⑫ Dynamiczne struktury ochrony
- ⑬ Unieważnianie praw dostępu

# Plan wykładu

## ① Wstęp

### ② Podstawowe pojęcia

- ③ Ochrona
- ④ Bezpieczeństwo
- ⑤ Polityka bezpieczeństwa i mechanizmy
- ⑥ Domeny i wiedza konieczna

### ③ Macierze dostępów

- ⑦ Tablica globalna
- ⑧ Wykaz dostępów
- ⑨ Wykaz możliwości
- ⑩ Mechanizm zamka z kluczem

### ④ Dynamiczne struktury ochrony

### ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

## ① Wstęp

## ② Podstawowe pojęcia

### ① Ochrona

- ② Bezpieczeństwo
- ③ Polityka bezpieczeństwa i mechanizmy
- ④ Domeny i wiedza konieczna

## ③ Macierze dostępów

- ① Tablica globalna
- ② Wykaz dostępów
- ③ Wykaz możliwości
- ④ Mechanizm zamka z kluczem

## ④ Dynamiczne struktury ochrony

## ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
    - ③ Polityka bezpieczeństwa i mechanizmy
    - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

## ① Wstęp

## ② Podstawowe pojęcia

- ① Ochrona

- ② Bezpieczeństwo

- ③ Polityka bezpieczeństwa i mechanizmy

- ④ Domeny i wiedza konieczna

## ③ Macierze dostępów

- ① Tablica globalna

- ② Wykaz dostępów

- ③ Wykaz możliwości

- ④ Mechanizm zamka z kluczem

## ④ Dynamiczne struktury ochrony

## ⑤ Unieważnianie praw dostępu

# Plan wykładu

## ① Wstęp

## ② Podstawowe pojęcia

- ① Ochrona
- ② Bezpieczeństwo
- ③ Polityka bezpieczeństwa i mechanizmy
- ④ Domeny i wiedza konieczna

## ③ Macierze dostępów

- ① Tablica globalna
- ② Wykaz dostępów
- ③ Wykaz możliwości
- ④ Mechanizm zamka z kluczem

## ④ Dynamiczne struktury ochrony

## ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

# Plan wykładu

- ① Wstęp
- ② Podstawowe pojęcia
  - ① Ochrona
  - ② Bezpieczeństwo
  - ③ Polityka bezpieczeństwa i mechanizmy
  - ④ Domeny i wiedza konieczna
- ③ Macierze dostępów
  - ① Tablica globalna
  - ② Wykaz dostępów
  - ③ Wykaz możliwości
  - ④ Mechanizm zamka z kluczem
- ④ Dynamiczne struktury ochrony
- ⑤ Unieważnianie praw dostępu

## Wstęp

Wraz z rozwojem systemów komputerowych narastała konieczność zapewnienia ochrony zasobów przed nieuprawnionym użyciem. Po raz pierwszy taki wymóg pojawił się na większą skalę w systemach wieloprogramowych i dotyczył ochrony pamięci. Na szeroką skalę sprawa bezpieczeństwa obecna jest w systemach wielodostępnych, których użytkownicy zainteresowani są kwestią poufności swoich danych.

# Ochrona

*Ochrona* jest mechanizmem służącym do kontrolowania dostępu procesów, wątków i użytkowników (ludzi) do zasobów systemu komputerowego. Dostarcza ona środków pozwalających określić rodzaj i zakres stosowanej kontroli, jak również pozwalających ją egzekwować. Dobrze zaimplementowana ochrona przyczynia się również do wykrywania i usuwania usterek obecnych w systemie.

## Bezpieczeństwo

Pojęciem szerszym znaczeniowo w stosunku do ochrony jest *bezpieczeństwo*, które możemy rozważyć w dwóch aspektach: niezawodnościowym (ang. safety) i związanym z kontrolą dostępu do danych (ang. security). Bezpieczeństwo niezawodnościowe ma na celu zapewnienie ciągłości usług oferowanych przez system komputerowy, nawet wtedy, kiedy doszło w nim do wystąpienia określonych usterek. Bezpieczeństwo związane z dostępem do danych ma na celu zapewnienie ich poufności. Oba te aspekty bezpieczeństwa są z sobą powiązane w sposób nierozerwalny. Pojęcie bezpieczeństwa jest również związane z prawie każdą dziedziną informatyki, a nawet wykracza poza nią (np.: kwestie prawne).

## Polityka bezpieczeństwa

*Polityka bezpieczeństwa* jest zbiorem postanowień odnośnie zakresu stosowanych środków bezpieczeństwa i postępowania w przypadku ich naruszenia. Dobrze zdefiniowana polityka bezpieczeństwa swoim zakresem obejmuje nie tylko system komputerowy, ale całość środków instytucji w której ten system jest zainstalowany. Sformułowanie prawnej polityki bezpieczeństwa nie może być obowiązkiem wyłącznie jednej osoby. Do jej określenia potrzebny jest zespół składający się z osób zarówno z wiedzą techniczną, jak i prawniczą.

## Mechanizmy

Zadaniem programisty systemowego jest dostarczenie środków, zwanych *mechanizmami* pozwalającymi na realizację polityki bezpieczeństwa w zakresie systemów komputerowych. Zbiór mechanizmów zaimplementowanych w systemie operacyjnym powinien być na tyle elastyczny, aby umożliwić realizację każdego z możliwych wariantów polityki bezpieczeństwa, dlatego ważne jest rozdzielenie tych dwóch pojęć. Dobrym przykładem rozdzielenia mechanizmu od polityki jest system uwierzytelniania użytkowników wykorzystujący moduły PAM, który został zaimplementowany między innymi w systemach Solaris, Linux, FreeBSD. Dalsza część wykładu będzie dotyczyć mechanizmów ochrony.

## Wiedza konieczna

System komputerowy składa się z zasobów i procesów. Jeśli zasoby powiążemy z operacjami, które mogą być na nich wykonywane, to możemy je traktować jako obiekty. Proces, który odwołuje się do takiego obiektu może wykonywać wyłącznie operacje, które są dla niego zdefiniowane. Co więcej, nie każdy proces powinien móc w każdej chwili wykonać wszystkie operacje przewidziane dla danego obiektu. W tej kwestii stosowana jest zasada *wiedzy koniecznej*, czyli *proces powinien móc wykonać na zasobie tylko te operacje, które są konieczne w danej chwili do kontynuacji jego pracy*.

## Domeny ochrony

Aby móc zastosować w praktyce schemat wiedzy koniecznej należy wprowadzić pojęcie *domeny ochrony*. Domena definiuje jakie zasoby są dostępne i jakie operacje może na nich wykonać proces w tej domenie pracujący. Innymi słowy domena określa prawa dostępu do obiektów. Domenę opisujemy jako zbiór par uporządkowanych *<nazwa-objektu, zbiór-praw>*. Domeny nie muszą być rozłączne, mogą mieć części wspólne.

## Macierze dostępu

Do modelowania systemu ochrony służą *macierze dostępu*. Wiersze tych macierzy zawierają nazwy domen, kolumny nazwy obiektów, natomiast każdy element zbiór praw dostępu od obiektu. Element macierzy o współrzędnych  $[i,j]$  określa jakie czynności mogą być wykonane w domenie o numerze  $i$  na obiekcie o numerze  $j$ . Na następnej planszy znajduje się przykładowa macierz dostępu. Wynika z niej np. że jeśli proces działa w domenie  $D_3$ , to może na obiekcie  $F_2$  wykonać jedynie operację odczytu.

## Macierze dostępu

<i>obiekt domena</i>	<i>F<sub>1</sub></i>	<i>F<sub>2</sub></i>	<i>F<sub>3</sub></i>	<i>cdrom</i>	<i>drukarka</i>
<i>D<sub>1</sub></i>	czytaj		czytaj		
<i>D<sub>2</sub></i>				czytaj	drukuj
<i>D<sub>3</sub></i>		czytaj	wykonaj		
<i>D<sub>4</sub></i>	czytaj pisz		czytaj pisz		

## Macierze dostępu

Z ilustracji umieszczonej na poprzedniej planszy wynika, że macierz dostępu jest macierzą rzadką. Choć istnieje efektywna reprezentacja takich struktur, bazująca na dynamicznych listach cyklicznych, to w opisywanych zastosowaniach jest mało praktyczna. Są za to stosowane inne formy implementacji macierzy dostępu.

## Tablica globalna

Macierz dostępu można zaimplementować w postaci tablicy składającej się z trójkę uporządkowanych <domena,obiekt,prawa>. Taka tablica ma dwie główne wady. Pierwszą wadą jest jej rozmiar, który może powodować, że nie będzie się mieściła w całości w pamięci operacyjnej, a drugi to powielanie informacji dotyczących domen i obiektów.

## Wykaz dostępów

Z każdym obiektem można związać wykaz domen i praw jakie w nich przysługują użytkownikowi obiektu. Jeśli w pewnej domenie nie przysługują żadne prawa do obiektu, to można ją pominąć. Dodatkowo schemat ten można rozszerzyć o pewne standardowe prawa, które obowiązują zawsze, niezależnie w jakiej domenie znajduje się użytkownik obiektu. Aby przyspieszyć wykonywanie operacji na obiekcie, najpierw powinien być przeszukiwany zbiór praw standardowych, a dopiero potem wykaz związany z danym obiektem.

## Wykaz możliwości

W systemie można stworzyć chronione obiekty, które reprezentują domeny. Jeśli użytkownik (proces) znajduje się w domenie, to jest związany z nim taki obiekt i dzięki niemu może on wykonać operacje na innych obiektach, do których dana domena zapewnia dostęp. Proces nigdy nie ma bezpośredniego dostępu do wykazu możliwości, ale zlecając systemowi operacyjnemu wykonanie operacji na obiekcie podaje jako parametr tej operacji obiekt wykazu możliwości. Jeśli zawiera on wskaźnik na obiekt, na którym ma być wykonana operacja, to jest ona realizowana, w przeciwnym wypadku następuje odmowa dostępu. Aby zapewnić ochronę wykazów możliwości należy zastosować środki pozwalające odróżnić je od innych obiektów. Rozwiązania tego problemu są dwa:

- ➊ Każdy obiekt jest etykietowany. Etykieta jest znacznikiem bitowym, który pozwala odróżnić możliwości od innych obiektów. Do wprowadzenia takiego rozwiązania wystarcza jeden bit, ale w systemach je stosujących używa się kilku bitów na etykię, dzięki temu możliwe jest odróżnianie nie tylko możliwości od innych zmiennych, ale również rozróżnianie zmiennych całkowitych, zmiennopozycyjnych, wskaźników. Etykieta stanowi więc reprezentację typu zmiennej na poziomie kodu maszynowego, a nie źródłowego. Etykiety mogą być interpretowane za pomocą środków programowych i sprzętowych.
- ➋ Rozdzielenie przestrzeni adresowej procesu na część zawierającą jego kod i dane, oraz część zawierającą wykazy możliwości. To rozwiązanie szczególnie dobrze nadaje się do zastosowania w systemach stosujących segmentację pamięci.

## Wykaz możliwości

W systemie można stworzyć chronione obiekty, które reprezentują domeny. Jeśli użytkownik (proces) znajduje się w domenie, to jest związany z nim taki obiekt i dzięki niemu może on wykonać operacje na innych obiektach, do których dana domena zapewnia dostęp. Proces nigdy nie ma bezpośredniego dostępu do wykazu możliwości, ale zlecając systemowi operacyjnemu wykonanie operacji na obiekcie podaje jako parametr tej operacji obiekt wykazu możliwości. Jeśli zawiera on wskaźnik na obiekt, na którym ma być wykonana operacja, to jest ona realizowana, w przeciwnym wypadku następuje odmowa dostępu. Aby zapewnić ochronę wykazów możliwości należy zastosować środki pozwalające odróżnić je od innych obiektów. Rozwiązania tego problemu są dwa:

- ① Każdy obiekt jest etykietowany. Etykieta jest znacznikiem bitowym, który pozwala odróżnić możliwości od innych obiektów. Do wprowadzenia takiego rozwiązania wystarcza jeden bit, ale w systemach je stosujących używa się kilku bitów na etykię, dzięki temu możliwe jest odróżnianie nie tylko możliwości od innych zmiennych, ale również rozróżnianie zmiennych całkowitych, zmiennopozycyjnych, wskaźników. Etykieta stanowi więc reprezentację typu zmiennej na poziomie kodu maszynowego, a nie źródłowego. Etykiety mogą być interpretowane za pomocą środków programowych i sprzętowych.
- ② Rozdzielenie przestrzeni adresowej procesu na część zawierającą jego kod i dane, oraz część zawierającą wykazy możliwości. To rozwiązanie szczególnie dobrze nadaje się do zastosowania w systemach stosujących segmentację pamięci.

## Wykaz możliwości

W systemie można stworzyć chronione obiekty, które reprezentują domeny. Jeśli użytkownik (proces) znajduje się w domenie, to jest związany z nim taki obiekt i dzięki niemu może on wykonać operacje na innych obiektach, do których dana domena zapewnia dostęp. Proces nigdy nie ma bezpośredniego dostępu do wykazu możliwości, ale zlecając systemowi operacyjnemu wykonanie operacji na obiekcie podaje jako parametr tej operacji obiekt wykazu możliwości. Jeśli zawiera on wskaźnik na obiekt, na którym ma być wykonana operacja, to jest ona realizowana, w przeciwnym wypadku następuje odmowa dostępu. Aby zapewnić ochronę wykazów możliwości należy zastosować środki pozwalające odróżnić je od innych obiektów. Rozwiązania tego problemu są dwa:

- ① Każdy obiekt jest etykietowany. Etykieta jest znacznikiem bitowym, który pozwala odróżnić możliwości od innych obiektów. Do wprowadzenia takiego rozwiązania wystarcza jeden bit, ale w systemach je stosujących używa się kilku bitów na etykię, dzięki temu możliwe jest odróżnianie nie tylko możliwości od innych zmiennych, ale również rozróżnianie zmiennych całkowitych, zmiennopozycyjnych, wskaźników. Etykieta stanowi więc reprezentację typu zmiennej na poziomie kodu maszynowego, a nie źródłowego. Etykiety mogą być interpretowane za pomocą środków programowych i sprzętowych.
- ② Rozdzielenie przestrzeni adresowej procesu na część zawierającą jego kod i dane, oraz część zawierającą wykazy możliwości. To rozwiązanie szczególnie dobrze nadaje się do zastosowania w systemach stosujących segmentację pamięci.

## Mechanizm zamka i klucza

Ten mechanizm jest rozwiązaniem pośrednimi między wykazem możliwości, a wykazem dostępów. Każda domena dysponuje zbiorem unikatowych wzorców binarnych nazywanych *kluczami*. Również z każdym obiektem związany jest zbiór unikatowych wzorców binarnych nazywanych *zamkami*. Jeśli proces pracujący w określonej domenie chce wykonać na obiekcie jakąś operację, to może dokonać tego tylko wtedy, gdy w tej domenie znajduje się klucz pasujący do określonego zamka w wykazie należącym do obiektu.

## Dynamiczne struktury ochrony

Większość złożonych procesów podczas pracy korzysta z dużej liczby obiektów i dodatkowo zmienia w trakcie działania sposób korzystania z nich. W systemach ochrony, w których zawartość domen jest statyczna oznaczałoby to konieczność umieszczenia w domenie wszystkich praw dostępu, jakie mogą być przydatne podczas całego cyklu wykonania danemu procesowi. Takie rozwiązywanie stanowi jawne naruszenie reguły wiedzy koniecznej. Aby do tego nie dopuścić należy umożliwić dynamiczne tworzenie nowych domen lub zmianę zawartości istniejących domen i umożliwić procesom przełączanie między domenami podczas pracy. To ostatnie rozwiązanie wymaga potraktowania domen jako obiektów iłączenia ich do macierzy dostępów. Dla domen musi być też sformułowane nowe prawo przełącz, które pozwala procesowi na przełączanie się między domenami. Na następnej planszy znajduje się ilustracja macierzy dostępów z domenami.

# Dynamiczne struktury ochrony

<i>obiekt</i>	<i>F<sub>1</sub></i>	<i>F<sub>2</sub></i>	<i>F<sub>3</sub></i>	<i>cdrom</i>	<i>drukarka</i>	<i>D<sub>1</sub></i>	<i>D<sub>2</sub></i>	<i>D<sub>3</sub></i>	<i>D<sub>4</sub></i>
<i>domena</i>									
<i>D<sub>1</sub></i>	czytaj		czytaj				przelacz		
<i>D<sub>2</sub></i>				czytaj	drukuj			przelacz	przelacz
<i>D<sub>3</sub></i>		czytaj	wykonaj						
<i>D<sub>4</sub></i>	czytaj pisz		czytaj pisz			przelacz			

## Dynamiczne struktury ochrony

Dopełnieniem włączania domen jako obiektów do macierzy dostępów jest zezwolenie na modyfikację zawartości domen. Można to zrealizować wprowadzając trzy dodatkowe prawa: *kopiowania, właściciela i kontroli*. Prawo kopiowania występuje w połączeniu z innymi prawami. Jeśli użytkownik (proces) pracuje w domenie, w której występuje prawo kopiowania, to może on przenieść prawo, które występuje wraz z prawem kopiowania do innej domeny. To przeniesienie najczęściej odbywa się na jeden z dwóch sposobów:

- prawo jest kopiowane do domeny docelowej, a z domeny źródłowej usuwane,
- prawo jest kopiowane, ale bez prawa kopiowania.

Ostatni wariant nazywamy *ograniczonym kopiowaniem*. W macierzach dostępów prawo kopiowania oznaczane jest gwiazdką i obowiązuje w zakresie kolumny, w której występuje. Dwie następne plansze pokazują sposób funkcjonowania prawa ograniczonego kopiowania.

## Dynamiczne struktury ochrony

Dopełnieniem włączania domen jako obiektów do macierzy dostępów jest zezwolenie na modyfikację zawartości domen. Można to zrealizować wprowadzając trzy dodatkowe prawa: *kopiowania, właściciela i kontroli*. Prawo kopiowania występuje w połączeniu z innymi prawami. Jeśli użytkownik (proces) pracuje w domenie, w której występuje prawo kopiowania, to może on przenieść prawo, które występuje wraz z prawem kopiowania do innej domeny. To przeniesienie najczęściej odbywa się na jeden z dwóch sposobów:

- prawo jest kopiowane do domeny docelowej, a z domeny źródłowej usuwane,
- prawo jest kopiowane, ale bez prawa kopiowania.

Ostatni wariant nazywamy *ograniczonym kopiowaniem*. W macierzach dostępów prawo kopiowania oznaczane jest gwiazdką i obowiązuje w zakresie kolumny, w której występuje. Dwie następne plansze pokazują sposób funkcjonowania prawa ograniczonego kopiowania.

## Dynamiczne struktury ochrony

Dopełnieniem włączania domen jako obiektów do macierzy dostępów jest zezwolenie na modyfikację zawartości domen. Można to zrealizować wprowadzając trzy dodatkowe prawa: *kopiowania, właściciela i kontroli*. Prawo kopiowania występuje w połączeniu z innymi prawami. Jeśli użytkownik (proces) pracuje w domenie, w której występuje prawo kopiowania, to może on przenieść prawo, które występuje wraz z prawem kopiowania do innej domeny. To przeniesienie najczęściej odbywa się na jeden z dwóch sposobów:

- prawo jest kopiowane do domeny docelowej, a z domeny źródłowej usuwane,
- prawo jest kopiowane, ale bez prawa kopiowania.

Ostatni wariant nazywamy *ograniczonym kopiowaniem*. W macierzach dostępów prawo kopiowania oznaczane jest gwiazdką i obowiązuje w zakresie kolumny, w której występuje. Dwie następne plansze pokazują sposób funkcjonowania prawa ograniczonego kopiowania.

# Dynamiczne struktury ochrony

<i>obiekt</i>	$F_1$	$F_2$	$F_3$
<i>domena</i>			
$D_1$	wykonaj		pisz"
$D_2$	wykonaj	czytaj"	wykonaj
$D_3$	wykonaj		

# Dynamiczne struktury ochrony

<i>obiekt</i>	$F_1$	$F_2$	$F_3$
<i>domena</i>			
$D_1$	wykonaj		pisz*
$D_2$	wykonaj	czytaj*	wykonaj
$D_3$	wykonaj	czytaj	

## Dynamiczne struktury ochrony

Jeśli proces działa w domenie w której obowiązuje prawo właściciela dla określonego obiektu, to może on usuwać i dodawać nowe prawa do innych domen. To prawo, podobnie jak prawo kopiowania działa w obrębie kolumn. Następne dwie plansze ilustrują sposób działania takiego prawa.

# Dynamiczne struktury ochrony

<i>obiekt</i>	<i>F<sub>1</sub></i>	<i>F<sub>2</sub></i>	<i>F<sub>3</sub></i>
<i>domena</i>			
<i>D<sub>1</sub></i>	właściciel wykonaj		pisz
<i>D<sub>2</sub></i>		właściciel czytaj*	właściciel czytaj* pisz*
<i>D<sub>3</sub></i>	wykonaj		

## Dynamiczne struktury ochrony

<i>obiekt</i>	<i>F<sub>1</sub></i>	<i>F<sub>2</sub></i>	<i>F<sub>3</sub></i>
<i>domena</i>			
<i>D<sub>1</sub></i>	właściciel wykonaj		
<i>D<sub>2</sub></i>		właściciel czytaj* pisz*	właściciel czytaj* pisz*
<i>D<sub>3</sub></i>		pisz	pisz

## Dynamiczne struktury ochrony

Uzupełnieniem dwóch przedstawionych praw jest prawo *kontroli* pozwalające modyfikować domeny. Jest ono stosowane tylko w odniesieniu do obiektów domen. W odniesieniu do macierzy dostępów oznacza to, że proces działający w domenie w której zostało umieszczone prawo kontroli innej domeny może w niej dodawać i usuwać prawa dla wszystkich należących do niej obiektów. Innymi słowy, prawo kontroli działa w obrębie wierszy. Następna plansza ilustruje działanie tego prawa.

# Dynamiczne struktury ochrony

<b>obiekt</b>	<b>F<sub>1</sub></b>	<b>F<sub>2</sub></b>	<b>F<sub>3</sub></b>	<b>cdrom</b>	<b>drukarka</b>	<b>D<sub>1</sub></b>	<b>D<sub>2</sub></b>	<b>D<sub>3</sub></b>	<b>D<sub>4</sub></b>
<b>domena</b>									
D <sub>1</sub>	czytaj		czytaj				przelacz		
D <sub>2</sub>				czytaj	drukuj		przelacz	kontroluj przelacz	
D <sub>3</sub>		czytaj	wykonaj						
D <sub>4</sub>	pisz		pisz			przelacz			

## Unieważnianie praw dostępu

**Zezwolenie na modyfikacje zawartości domen wymusza zastosowanie unieważniania praw dostępu. Implementując taką możliwość należy rozstrzygnąć następujące kwestie:**

- Czy prawa mają być unieważniane natychmiast, czy z opóźnieniem?
- Czy unieważnienie dotyczy wszystkich użytkowników (unieważnienie ogólne), czy tylko określonej grupy (unieważnienie wyborcze)?
- Czy unieważniane są wszystkie prawa (unieważnienie całkowite), czy tylko część z nich (unieważnienie częściowe)?
- Czy odbieramy prawa użytkownikom permanentnie (unieważnienie na stałe), czy też istnieje możliwość ich przywrócenia (unieważnienie czasowe)?

## Unieważnianie praw dostępu

Zezwolenie na modyfikacje zawartości domen wymusza zastosowanie unieważniania praw dostępu. Implementując taką możliwość należy rozstrzygnąć następujące kwestie:

- Czy prawa mają być unieważniane natychmiast, czy z opóźnieniem?
- Czy unieważnienie dotyczy wszystkich użytkowników (unieważnienie ogólne), czy tylko określonej grupy (unieważnienie wybiórcze)?
- Czy unieważniane są wszystkie prawa (unieważnienie całkowite), czy tylko część z nich (unieważnienie częściowe)?
- Czy odbieramy prawa użytkownikom permanentnie (unieważnienie na stałe), czy też istnieje możliwość ich przywrócenia (unieważnienie czasowe)?

## Unieważnianie praw dostępu

Zezwolenie na modyfikacje zawartości domen wymusza zastosowanie unieważniania praw dostępu. Implementując taką możliwość należy rozstrzygnąć następujące kwestie:

- Czy prawa mają być unieważniane natychmiast, czy z opóźnieniem?
- Czy unieważnienie dotyczy wszystkich użytkowników (unieważnienie ogólne), czy tylko określonej grupy (unieważnienie wybiórcze)?
- Czy unieważniane są wszystkie prawa (unieważnienie całkowite), czy tylko część z nich (unieważnienie częściowe)?
- Czy odbieramy prawa użytkownikom permanentnie (unieważnienie na stałe), czy też istnieje możliwość ich przywrócenia (unieważnienie czasowe)?

## Unieważnianie praw dostępu

Zezwolenie na modyfikacje zawartości domen wymusza zastosowanie unieważniania praw dostępu. Implementując taką możliwość należy rozstrzygnąć następujące kwestie:

- Czy prawa mają być unieważniane natychmiast, czy z opóźnieniem?
- Czy unieważnienie dotyczy wszystkich użytkowników (unieważnienie ogólne), czy tylko określonej grupy (unieważnienie wybiórcze)?
- Czy unieważniane są wszystkie prawa (unieważnienie całkowite), czy tylko część z nich (unieważnienie częściowe)?
- Czy odbieramy prawa użytkownikom permanentnie (unieważnienie na stałe), czy też istnieje możliwość ich przywrócenia (unieważnienie czasowe)?

## Unieważnianie praw dostępu

Zezwolenie na modyfikacje zawartości domen wymusza zastosowanie unieważniania praw dostępu. Implementując taką możliwość należy rozstrzygnąć następujące kwestie:

- Czy prawa mają być unieważniane natychmiast, czy z opóźnieniem?
- Czy unieważnienie dotyczy wszystkich użytkowników (unieważnienie ogólne), czy tylko określonej grupy (unieważnienie wybiórcze)?
- Czy unieważniane są wszystkie prawa (unieważnienie całkowite), czy tylko część z nich (unieważnienie częściowe)?
- Czy odbieramy prawa użytkownikom permanentnie (unieważnienie na stałe), czy też istnieje możliwość ich przywrócenia (unieważnienie czasowe)?

## Unieważnienie praw dostępu

Jeśli macierz dostępów jest zaimplementowana jako wykaz dostępów, to unieważnianie stosunkowo łatwo jest zaimplementować i jest ono natychmiastowe, może być ogólne lub wybiórcze, całkowite lub częściowe, stałe lub czasowe. Wykazy możliwości natomiast są rozproszone po całym systemie, co utrudnia przeprowadzanie unieważniania. Dlatego stosuje się jeden z następujących schematów:

## Wtórne pozyskiwanie

Możliwości są okresowo usuwane ze wszystkich domen. Proces, który potrzebuje danej możliwości powinien wykryć jej usunięcie i spróbować ją ponownie pozyskać. Jeśli została ona usunięta to nie będzie mógł jej uzyskać.

## Wskaźniki zwrotne

Każdy obiekt posiada listę wskaźników na związane z nim możliwości. Jeśli należy przeprowadzić unieważnienie, to dokonuje się modyfikacji odpowiedniego wskaźnika, tak aby wskazywał on na inną możliwość. Schemat ten jest elastyczny i ogólny, ale kosztowny w realizacji.

## Sposób pośredni

Możliwości powiązane są z obiektami za pomocą elementów tablicy globalnej. Jeśli należy przeprowadzić unieważnienie, to wyszukuje się i usuwa zawartość odpowiedniego elementu tej tablicy. Tak opróżniony element można wykorzystać później dla innych możliwości. Ten schemat nie pozwala na częściowe unieważnianie.

## Klucze

Z każdym obiektem wiązany jest *klucz główny*, który jest unikatowym wzorcem binarnym (ang. binary pattern). Ten klucz może być ustalany lub zmieniany za pomocą *operacji ustawienia klucza* (ang. set-key). Tę operację może wykonywać tylko ustalona grupa procesów (np. właściciel obiektu do którego zmieniane są prawa). Każda możliwość podczas tworzenia „stemplowana” jest bieżącą wartością klucza. Zanim zostanie wykonana na obiekcie operacja, na którą zezwala możliwość porównywana jest wartość klucza tej możliwości z kluczem głównym obiektu. Jeśli te wartości się nie zgadzają to następuje odmowa dostępu do obiektu. Aby więc unieważnić prawa dostępu wystarczy zmienić wartość klucza głównego. Ten schemat nie pozwala na częściowe odbieranie praw. Można usunąć tę niedogodność stosując nie jeden klucz główny, a listę takich kluczy. Jeszcze elastyczniejsze rozwiązanie zakłada zgromadzenie wszystkich kluczy głównych w tablicy globalnej. Dzięki temu jeden klucz mógłby być powiązany z kilkoma obiektami jednocześnie. Ustalenie, czy możliwość jest prawomocna wymagałoby odnalezienia w tablicy globalnej klucza o takiej samej wartości, jak jej klucz.

## Pytania

?

Koniec

Dziękuję Państwu za uwagę!

# Systemy Operacyjne — Wirtualizacja

Arkadiusz Chrobot, Grzegorz Łukawski

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 16 stycznia 2021

# Plan

- 1 Wstęp
- 2 Emulacja i symulacja
  - Symulacja
  - Emulacja
- 3 Virtualizacja
  - Hipernadzorca typu 1
  - Hipernadzorca typu 2
  - Parawirtualizacja
- 4 Zarządzanie zasobami w wirtualizacji
- 5 Virtualizacja - podsumowanie
- 6 Zakończenie

# Wstęp

Virtualizacja jest techniką zarządzania zasobami, która umożliwia ich współdzielenie przez procesy lub nawet systemy komputerowe za pomocą:

- podziału,
- agregacji.

W węższym znaczeniu oznacza zastosowanie **maszyn wirtualnych**, celem umożliwienia wykonania oprogramowania, które przeznaczone jest dla tej samej platformy sprzętowej, ale wymaga specjalnego środowiska wykonawczego. Ten aspekt wirtualizacji oraz pojęcia pokrewne, takie jak **emulacja i symulacja** będą przedmiotem tego wykładu.

# Symulacja

Symulacja jest pojęciem o szerszym znaczeniu niż emulacja. Oznacza ono imitowanie przez program komputerowy zachowania określonego modelu abstrakcyjnego. W przypadku informatyki model ten dotyczy najczęściej urządzenia przetwarzającego informację. Jako przykłady można podać symulatory Maszyny Turinga i Maszyny RAM.

## Emulacja

Emulacja polega na symulowaniu przez program komputerowy uruchomiony w systemie komputerowym o danej strukturze pełnego zachowania innego systemu komputerowego, najczęściej o odmiennej konstrukcji. Program symulujący określany jest mianem **emulatora**.

Istnieją dwa rodzaje emulatorów:

- ① **emulatory pełne** - działają zgodnie ze specyfikacją określonego fizycznego komputera (przykłady: QEMU, Bochs, UAE, Frodo),
- ② **emulatory API** - emulują jedynie działanie interfejsu programistycznego określonego systemu operacyjnego i nazywane są często **warstwami kompatybilności** (np. WINE, DosBox, dosemu).

Warto zauważyć, że emulatory API nie są całkowicie zgodne z podaną wyżej definicją emulatora - symulują działanie tylko jednej z warstw systemu operacyjnego. W związku z tym niektórzy informatycy są zdania, że nie należy nazywać ich emulatorami. Szczególnie podkreślają to twórcy WINE, którego nazwa jest rozwijana rekurencyjnie do „WINE Is Not an Emulator”.

## Emulacja - możliwości realizacji

Emulatory pełne symulują działanie platformy sprzętowej do poziomu kodu maszynowego. Istnieją dwie możliwości realizacji tak dokładnego odwzorowania. Pierwsza polega na **interpretacji** programu przygotowanego dla danej platformy przez emulator. Druga oparta jest na dynamicznej rekompilacji, tzn. tłumaczeniu określonego bloku kodu zamiast pojedynczej instrukcji.

## Emulatory - motywacja

Emulatory tworzone są z trzech powodów:

- ① potrzeba użycia określonej platformy sprzętowej, która nie jest dostępna,
- ② potrzeba uruchomienia oprogramowania przeznaczonego na starsze platformy sprzętowe,
- ③ stworzenie platformy testowej dla oprogramowania.

W latach 80 ubiegłego wieku termin „emulacja” oznaczał sprzętową imitację danej platformy z użyciem mikrokodu, natomiast „symulacja” oznaczała imitację na poziomie oprogramowania.

# Wprowadzenie

Podczas gdy „emulacja” oznacza głównie symulację określonej platformy sprzętowej na innej platformie, to **wirtualizacja** najczęściej oznacza w tym kontekście stworzenie kopii takiej samej platformy na jakiej przeprowadzana jest wirtualizacja. Kopia ta, nazywana **maszyną wirtualną**, nie jest dokładna z racji tego, że nie może ona używać w całości zasobów należących do maszyny fizycznej. Niemniej jednak jest ona na tyle wierna, że pozwala uruchomić system operacyjny i oprogramowanie użytkowe.

W artykule z 1974 roku Gerald J. Popek i Robert P. Goldberg określili kryteria jakie powinna spełniać poprawnie działająca maszyna wirtualna.

- ① **Odpowiedniość** – program działający na maszynie wirtualnej musi zachowywać się dokładnie tak samo jakby był wykonywany na maszynie fizycznej.
- ② **Kontrola zasobów** – maszyna wirtualna musi w pełni kontrolować wszystkie zasoby podlegające wirtualizacji.
- ③ **Wydajność** – większość instrukcji musi być wykonywana na maszynie fizycznej, bez udziału maszyny wirtualnej.

W praktyce warunki te nie muszą (i zazwyczaj nie są) dokładnie wypełnione. Powoduje to ograniczenia w działaniu maszyn wirtualnych oraz niższą wydajność.

## Virtualizacja - motywacja

### Zastosowania virtualizacji:

- zwiększenie niezawodności serwerów,
- prosta migracja maszyn wirtualnych,
- możliwość jednoczesnej pracy wielu wersji systemu operacyjnego,
- wirtualny hosting,
- testowanie systemów operacyjnych, aplikacji i oprogramowania sieciowego.

## Virtualizacja - motywacja

### Zastosowania virtualizacji:

- zwiększenie niezawodności serwerów,
- prosta migracja maszyn wirtualnych,
- możliwość jednoczesnej pracy wielu wersji systemu operacyjnego,
- wirtualny hosting,
- testowanie systemów operacyjnych, aplikacji i oprogramowania sieciowego.

## Virtualizacja - motywacja

### Zastosowania virtualizacji:

- zwiększenie niezawodności serwerów,
- prosta migracja maszyn wirtualnych,
- możliwość jednoczesnej pracy wielu wersji systemu operacyjnego,
- wirtualny hosting,
- testowanie systemów operacyjnych, aplikacji i oprogramowania sieciowego.

## Virtualizacja - motywacja

### Zastosowania virtualizacji:

- zwiększenie niezawodności serwerów,
- prosta migracja maszyn wirtualnych,
- możliwość jednoczesnej pracy wielu wersji systemu operacyjnego,
- wirtualny hosting,
- testowanie systemów operacyjnych, aplikacji i oprogramowania sieciowego.

## Virtualizacja - motywacja

Zastosowania virtualizacji:

- zwiększenie niezawodności serwerów,
- prosta migracja maszyn wirtualnych,
- możliwość jednoczesnej pracy wielu wersji systemu operacyjnego,
- wirtualny hosting,
- testowanie systemów operacyjnych, aplikacji i oprogramowania sieciowego.

## Hipernadzorca typu 1 - wymagania

G.J.Popek i R.P.Goldberg w przytoczonym wcześniej artykule podali również wymagania, które musi spełniać architektura, aby umożliwić realizację wirtualizacji z użyciem hipernadzorcy typu 1. Wyróżnili oni trzy grupy rozkazów procesora: **rozkazy zwykłe**, które mogą być wykonywane w obu trybach pracy procesora, **rozkazy uprzywilejowane**, których próba wykonania w trybie użytkownika powoduje wygenerowanie wyjątku oraz **rozkazy wrażliwe**, które powinny być wykonywane tylko w trybie jądra. Dana architektura umożliwia realizację wirtualizacji z użyciem hipernadzorcy typu 1, jeśli zbiór jej instrukcji wrażliwych jest podzbiorem zbioru instrukcji uprzywilejowanych. W przypadku architektur PC taką wirtualizację umożliwiają procesory firmy AMD z technologią SVM (ang. Secure Virtual Machine)<sup>1</sup> oraz firmy Intel z technologią VT (ang. Virtualization Technology), które umożliwiają hipernadzorcy określenie, które rozkazy powinny generować wyjątki przy próbie wykonania ich w trybie użytkownika, a które nie.

<sup>1</sup> AMD zdecydowało się później przemianować tę technologię na AMD - V.

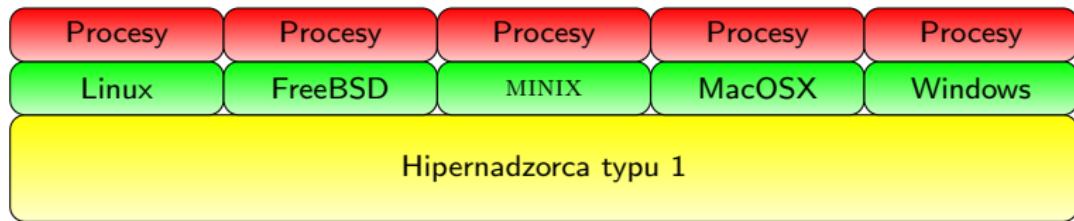
## Hipernadzorca typu 1 - charakterystyka działania

Działanie wirtualizacji opartej o hipernadzorcę typu 1 można opisać następująco:

- w trybie jądra procesora działa tylko hipernadzorca nazywany również **monitorem maszyn wirtualnych**,
- hipernadzorca kontroluje pracę wszystkich pracujących maszyn wirtualnych,
- system operacyjny maszyny wirtualnej, nazywany **systemem operacyjnym - gościem** działa w trybie użytkownika maszyny rzeczywistej oraz w trybie jądra maszyny wirtualnej.
- procesy użytkownika działają w trybie użytkownika zarówno maszyny wirtualnej, jaki i rzeczywistej,
- rozkazy nieuprzywilejowane są wykonywane przez system operacyjny - gościa i procesy użytkownika bezpośrednio,
- jeśli pojawi się wyjątek spowodowany przez próbę wykonania rozkazu uprzywilejowanego, to hipernadzorca sprawdza, co próbowało go wykonać; jeśli był to proces użytkownika, to uruchamiana jest obsługa wyjątku w systemie - gościu, jeśli był to system - gość, to hipernadzorca wykonuje ten rozkaz.

Virtualizację tego typu zastosowano po raz pierwszy w komputerach typu mainframe firmy IBM. System operacyjny obsługujący to rozwiązanie początkowo nazwano CP/CMS, a później przemianowano na VM/370.

## Hipernadzorca typu 1



Rysunek: Wirtualizacja z użyciem hipernadzorcy typu 1

## Hipernadzorcy typu 2 - wymagania

Virtualizacja z użyciem hipernadzorcy typu 2 nie wymaga wsparcia ze strony procesora w postaci takich technologii jak Intel VT lub AMD - V. Przykładami takich hipernadzorców są VirtualBox i VirtualPC.

## Hipernadzorca typu 2 - charakterystyka działania

Działanie wirtualizacji z użyciem hipernadzorcy typu 2 można opisać następująco:

- hipernadzorca jest oprogramowaniem działającym w trybie użytkownika pod kontrolą systemu operacyjnego nazywanego **systemem operacyjnym - gospodarzem**,
- z punktu widzenia użytkownika hipernadzorca zachowuje się jak emulator maszyny rzeczywistej na której jest uruchomiony; pozwala zainstalować system operacyjny (nazywany systemem operacyjnym - gościem) i uruchamiać procesy użytkownika,
- problem rozkazów wrażliwych hipernadzorca rozwiązuje stosując **translację binarną**, która polega na znalezieniu **bloków podstawowych**, czyli ciągów rozkazów zakończonych dowolnym rozkazem zmieniającym przepływ sterowania i zastąpieniu wszystkich instrukcji wrażliwych znajdujących się w takich blokach wywołaniami procedur hipernadzorcy,
- celem zwiększenia wydajności stosowana jest translacja z wyprzedzeniem oraz pamięci podręczne.

Wbrew intuicji, wirtualizacja z użyciem hipernadzorcy typu 2 może być w pewnych warunkach wydajniejsza, niż wirtualizacja z użyciem hipernadzorcy typu 1.

## Hipernadzorca 2



Rysunek: Wirtualizacja z użyciem hipernadzorcy typu 2

## Parawirtualizacja - wymagania

Parawirtualizacja jest wydajniejsza niż opisane wcześniej sposoby wirtualizacji, ale wymaga specjalnie przygotowanej wersji systemu operacyjnego - gościa. Przykładem hipernadzorcy umożliwiającego parawirtualizację jest Xen.

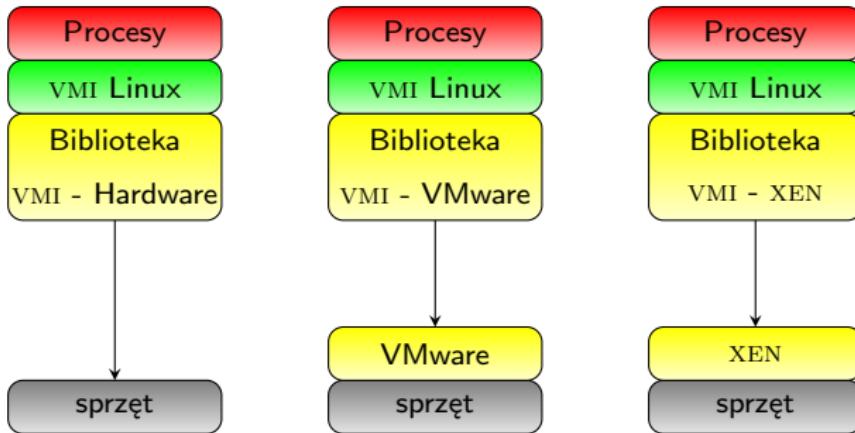
## Parawirtualizacja - charakterystyka działania

Działanie parawirtualizacji można scharakteryzować następująco:

- hipernadzorca działa w trybie systemowym maszyny rzeczywistej, tak jak hipernadzorca typu 1,
- hipernadzorca posiada własne API,
- kod źródłowy systemu operacyjnego - gościa jest specjalnie spreparowany; każde odwołanie do instrukcji wrażliwej zastąpiono w nim wywołaniem podprogramu wchodzącego w skład API hipernadzorcy.
- wykonywalna wersja systemu - gościa jest uruchamiana w trybie użytkownika maszyny rzeczywistej; nie wykonuje ona żadnych (lub prawie żadnych) rozkazów wrażliwych, jedynie korzysta z API hipernadzorcy.

Parawirtualizacja może być stosowana na jednym komputerze, obok wcześniej wymienionych sposobów virtualizacji. Hipernadzorców, którzy umożliwiają parawirtualizację jest wielu i każdy z nich posiada swoje API. Aby umożliwić systemowi - gościowi współpracę z nimi wszystkimi, bez konieczności wykonywania dodatkowych modyfikacji wprowadzono, warstwę pośredniczącą nazywaną **biblioteką VMI** (ang. Virtual Machine Interface).

## Parawirtualizacja



Rysunek: Parawirtualizacja z użyciem techniki VMIL

## Virtualizacja pamięci

Oprócz procesora również pozostałe zasoby maszyny rzeczywistej muszą podlegać virtualizacji. Najważniejszym z nich jest pamięć operacyjna. We współczesnych systemach najczęściej stosowanym sposobem zarządzania RAM jest stronicowanie na żądanie, co utrudnia gospodarowanie pamięcią na rzecz maszyn wirtualnych. Kłopotliwy scenariusz powstaje wtedy, gdy co najmniej dwie maszyny wirtualne próbują odwzorować swoje strony na te same ramki. Rozwiążaniem problemu jest utrzymywanie przez hipernadzorcę dla każdej maszyny wirtualnej specjalnej tablicy stron, która stanowi kolejną warstwę w procesie translacji numeru strony (zamienia adres ramki wirtualnej maszyny na adres ramki maszyny rzeczywistej). Inny problem polega na tym, że system operacyjny - gość może zmienić zawartość tablicy bez używania rozkazów wrażliwych, stosując jedynie zwykły zapis do pamięci. Ta modyfikacja powinna być wykryta przez hipernadzorce, żeby mógł uaktualnić tablicę dla maszyny wirtualnej. W przypadku wirtualizacji hipernadzorca wykrywa, które strony zawierają tablicę strony systemu - gościa i oznacza je jako *tylko do odczytu*. Każda próba modyfikacji zawartości tych stron generuje wyjątek, który obsługuje hipernadzorca. W przypadku parawirtualizacji system - gość powiadamia hipernadzorce o zmianie zawartości swojej tablicy stron.

## Virtualizacja urządzeń wejścia-wyjścia

Hipernadzorca musi także przechwytywać polecenia sterowników urządzeń systemu operacyjnego - gościa. Konieczność ta wynika z potrzeby ochrony sprzętu oraz niemożliwości przydzielenia niektórych zasobów (np. dysku) w całości pojedynczej maszynie wirtualnej. Zaletą takiego rozwiązania jest również udogodnienie polegające na tym, że oprogramowanie obsługujące sprzęt starego lub innego typu można wykonać z użyciem urządzeń nowego typu. Konwersją poleceń między oboma typami urządzeń zajmie się ponownie hipernadzorca. Hipernadzorca musi zadbać również o poprawność przebiegu transmisji DMA. W wykonaniu tego zadania pomaga sprzęt w postaci IOMMU. Problem wirtualizacji obsługi wejścia-wyjścia można również rozwiązać przeznaczając jedną z maszyn wirtualnych do tego zadania i przekierowując do niej żądania wykonania operacji pochodzące z innych maszyn. To rozwiązanie jest szczególnie wygodne w przypadku paravirtualizacji i stosuje je Xen. Maszyna wirtualna, która zajmuje się wykonywaniem operacji I/O nazywa się w tej platformie **domeną 0**. Takie podejście do realizacji operacji wejścia-wyjścia korzystne jest również w wirtualizacji z użyciem hipernadzorcy typu 1, bo nie musi on zawierać sterownika do danego urządzenia. Hipernadzorca typu 2 może polegać na tym, że sterownik określonego urządzenia posiada system operacyjny - gospodarz.

## Virtualizacja - podsumowanie

Virtualizacja w przypadku sprzętu klasy PC i stacji roboczych jest nowością. Wiele problemów, które obecnie powoduje gorszą wydajność maszyn wirtualnych w przyszłości powinno być rozwiązywanych za pomocą odpowiedniego wsparcia sprzętowego. Należy również nadmienić, że pojęcie maszyny wirtualnej nie jest ściśle związane z opisanym rodzajem virtualizacji. Przykładem są tu maszyny wirtualne stosowane w językach programowania Java (JVM, Dalvik, ART) i C# (CLR). Pewną formę virtualizacji wykorzystują systemy rodziny Windows do wykonywania programów napisanych dla systemu DOS. Rolę hipernadzorcy pełni tu procesor, który tworzy maszyny wirtualne dla których zachowuje się jak swój 16-bitowy poprzednik (procesor 8086). Tryb pracy procesora, w którym imituje działanie starych procesorów nazywany jest **trybem wirtualnym**.

## Źródła

Do przygotowania niniejszego tekstu wykorzystano artykuł w Wikipedii na temat emulatorów: <http://en.wikipedia.org/wiki/Emulate> oraz informacje zawarte w literaturze podanej na stronie przedmiotu.

# Pytania

?

Koniec

Dziękuję Państwu za uwagę!