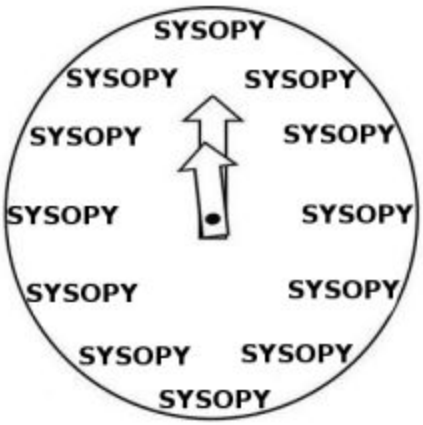
Opracowanie - Księga Koźlaka

Spisał Aleksander Mikucki, z zapożyczeniami z opracowania 2017



Wykład 1

Struktura systemu operacyjnego

- **Urządzenia Fizyczne** – kable danych, zasilające, zintegrowane układy elektroniczne
- **Mikroprogram** – siedzi w ROM, bezpośrednio **kontroluje urządzenia fizyczne**
- **Język Maszynowy** – **zbiór instrukcji mikroprogramu**, 50-300 instrukcji do przesyłania danych, skoków, porównywania, obliczeń itd. Urządzenia WE/WY są obsługiwane przez wpisywanie wartości do **rejestrów urządzeń**.
- **System Operacyjny** – **abstrakcja złożoności języka maszynowego**, tryb jądra lub użytkownika
- **Powyżej SO** – oprogramowanie i aplikacje w trybie użytkownika

System operacyjny

Jak rząd – nie ma sam żadnej użytecznej funkcji, ale przygotowuje środowisko dla innych, pożytecznych programów

System operacyjny jako rozszerzona/wirtualna maszyna

– architektura komputerów na poziomie **języka maszynowego jest trudna do użycia w programach**, np. wejście wyjście - trudno użyć
– system operacyjny **udostępnia maszynę rozszerzoną/wirtualną,** łatwiejszą do programowania**, ukrywającą złożoność** - przyjazny interfejs

System operacyjny jako zarządca zasobów

Udostępnia użytkownikowi różne zasoby (pamięć, drukarki, dyski), kontroluje dostęp, **zapobiega konfliktom i chaosowi**.

System operacyjny jako program sterujący

Nadzoruje działanie programów użytkownika, przeciwdziała błędom i zapobiega niewłaściwemu użyciu.

Cele systemu operacyjnego

Efektywność, Wygoda
Kiedyś stawiano na efektywność, teraz na wygodę

Typy systemów:

1. **Wsadowy**

SO na stałe w pamięci, przekazuje sterowanie od zadania do zadania
Grupowanie podobnych zadań we wsadzie (batchu)
Bezczynne czekanie na WE/WY

Odkryliśmy wieloprogramowanie – system wieloprogramowalny wsadowy
Job pool – zadania czekające w buforze na dysku
Planowanie zadań (scheduling)
SO utrzymuje w pamięci kilka zadań naraz – wieloprogramowanie

	nie da się ingerować w wykonujące się zadanie **Trzeba przewidzieć każdy scenariusz** **Trudny w testowaniu**

2. **Z podziałem czasu**

Wielozadaniowość – bardzo szybkie przełączanie się między zadaniami, **podział czasu** procesora między użytkowników
Dodatkowo:
Bezpośredni (online) dostęp do systemu plików (plik to zestaw powiązanych informacji)**.**
Interakcyjność – dialog użytkownika z systemem
Szybki czas odpowiedzi – kilka sekund

Wsadowe: dobre dla WIELKICH zadań, bez dozoru
Interakcyjne: wiele krótkich działań o nieprzewidywalnych rezultatach.

Z czasem cechy dużych systemów interakcyjnych trafiają do PC (ochrona plików, wielozadaniowość)

3. **Równoległe**

3.1 Wieloprocesorowe (ściśle związane)

Wspólna szyna, zegar, peripherals i czasami pamięć.

Szybsze (ale nie liniowo do n, niezawodne i tolerujące awarie, **łagodna degradacja**

Podwojenie funkcji 2 kopie tych samych obliczeń, sprawdzamy co jakiś czas czy się zgadzają **-kosztowne**	**Wieloprzetwarzanie Symetryczne** Identyczne kopie, komunikują się **+wiele procesów równocześnie** **-nierówne obciążenie**	**Wieloprzetwarzanie Asymetryczne** **Wyróżniony procesor główny (zarządca)**

3.2 Rozproszone

Wiele procesorów, zupełnie odrębnych

Komunikacja przy użyciu linii komunikacyjnych

Stanowiska / węzły

Podział zasobów (bardziej udostępnianie, jak drukarki w sieci domowej) **Przyspieszenie obliczeń, load sharing** **Niezawodność** **komunikacja**	

4. **Czasu rzeczywistego**

Gdy mamy surowe wymagania na czas wykonania operacji, **przetwarzanie MUSI/POWINNO się zakończyć przed upływem czasu**

Hard RTOS **Gwarantuje** Specjalna konstrukcja (brak pamięci wirtualnej) Nie jest to żaden z systemów uniwersalnych	**Soft RTOS** **Bardzo się stara** Krytyczne zadania mają pierwszeństwo Nie wsadzisz takiego do robota przemysłowego Większość współczesnych SO spełnia wymagania

Hierarchia pamięci

Drogie bity, szybkie — Tanie bity, wolne

Rejestry > Cache > Pamięć > HD > CD, DVD, BR > Taśma magnetyczna

Składowe systemu: zarządzanie ...

Procesami (tworzenie, usuwanie, wstrzymywanie, wznawianie, zakleszczenia, synch)

Pamięcią operacyjną (obszary wolne, zajęte, przydzielanie i zwalnianie)

Plikami (tworzenie, usuwanie, manipulowanie, odwzorowywanie do pamięci pomocniczej, składowanie na nośnikach)

WE/WY (buforowanie, pamięć podręczna, spooling)

Pamięcią pomocniczą (zarządzanie obszarami, przydzielanie)

Praca sieciowa (dostęp do sieci)

System ochrony (nadzorowanie dostępu do programów, procesów plików)
System interpretacji poleceń (tworzenie, zarządzanie procesami, dostęp do plików, ochrony, sieci)

Usługi systemu:

Dla użytkownika:
Wykonanie programu, operacje WE/WY, manipulowanie systemem plików, komunikacja, wykrywanie błędów
Dla optymalizacji działania:
Rozliczanie (jak kto korzysta z zasobów)
Ochrona (nadzór nad dostępami do zasobów)

Struktura

Prosta, jak MSDOS, bez określonej struktury.
MSDOS - bezpośrednie procedury WE/WY, brak odporności na błędy.

UNIX - oryginalny UNIX miał jądro i programy systemowe.

Użytkownicy

_____
Powłoki i polecenia, kompilatory i interpretery, biblioteki

_____
Interfejs funkcji systemowych jądra

_____
Sygnały, filesystem, pamięć, wymiana, stronicowanie, całe to sysopowe gówno
---
Interfejs między jądrem a sprzętem

_____
Sprzęt

Cechy podejścia warstwowego

- dzielenie systemu operacyjnego na warstwy (poziomy)
- każda nowa warstwa jest zbudowana powyżej warstw niższych
- najniższa warstwa (0) – sprzęt
- najwyższa warstwa (N) – interfejs użytkownika
- dowolna warstwa pośrednia M:
    - zawiera dane i procedury, które mogą być wywołane z warstw wyższych (M+1)
    - może wywoływać operacje dotyczące niższych

wyszukiwanie błędów i weryfikacja systemu	**mniejszości gua wydajność, obecnie ogranicza się liczbę warstw**

Koncepcja maszyny wirtualnej

Tworzy wrażenie, że proces pracuje na wirtualnym procesorze, z wirtualną pamięcią
TYLKO DLA NIEGO <3
Zajebiste do emulacji innych systemów operacyjnych i architektur, trudna w realizacji, operacje mogą trwać dłużej (bo interpretowane) lub krócej (bo symulowane)

Odizolowanie od innych maszyn **Ochrona zasobów** **Łatwiejsze badania nad SO** **Cross platform dla programów użytkowych**	**Trudna w realizacji**

[mikrojądro] Cechy mikrojądra

Trend: wyjebać kod systemu do warstw wyższych, realizujmy funkcje systemowe jako procesy użytkownika. Jądro ma być jak najmniejsze.
Mikrojądra z reguły zawierają tylko proste operacje zarządzania procesami i pamięcią oraz mechanizmy komunikacji międzyprocesowej, reszta funkcjonalności przeniesiona jest do procesów systemowych pracujących w trybie użytkownika, zwanych **serwerami**. Windows NT zbliżony do mikrojądra

Jednolite interfejsy – wszystko po komunikatach **Rozszerzalność** **Elastyczność** **Przenośność między platformami – zmienia się tylko mikrojądro** **Stabilność – łatwo testować** **Obsługa systemów rozproszonych – bo komunikaty** **Obsługa systemów obiektowych**	**Wydajność** **Komunikacja – wszystko idzie przez mikrojądro (context switch)** **Synchronizacja**

W systemach monolitycznych brak struktury
W warstwowej zmiany mogą psuć inne warstwy

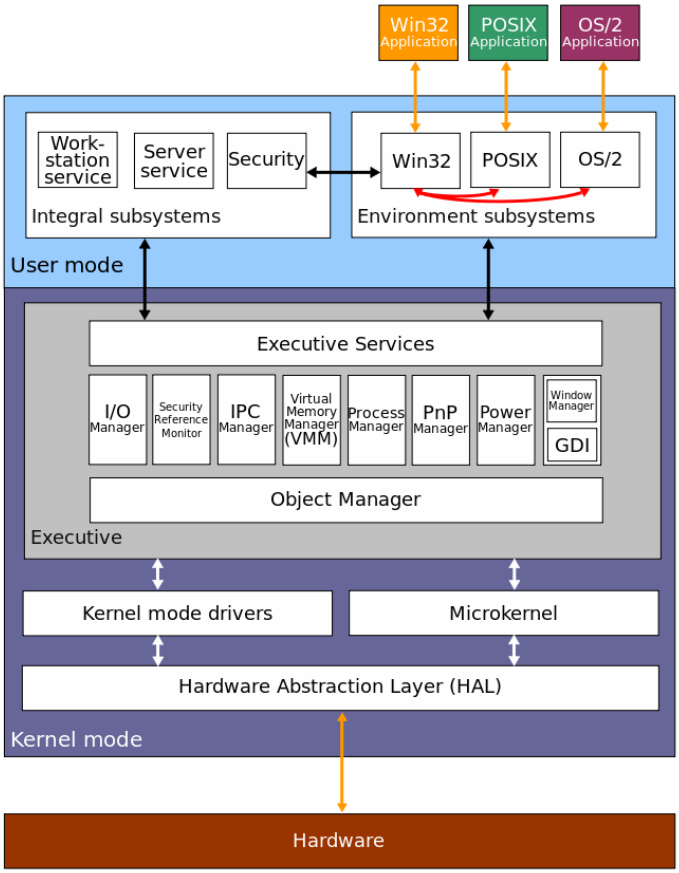
[Linux] Wymień przynajmniej pięć innowacji w System V Release 4

- Wsparcie dla TCP/IP □
- Sockety ⚽
- UFS (Unix File System) 🗂️
- Wsparcie dla wielu grup ✖
- C shell 🐚
- Pamięć dzielona
- Korn shell
- itp...

[Linux] Cechy charakterystyczne jądra linuxa

- Linux wspiera **dynamiczne ładowanie modułów jądra** — Choć jądro Linuksa jest monolityczne, możliwe jest ładowanie kodu jądra na żądanie
- Linux **wspiera symetryczną wieloprocesorowość** (SMP). Obecnie komercyjne wersje Uniksa wspierają SMP, ale tradycyjne implementacje nie wspierały
- **Jądro Linuksa jest wywłaszczalne**. Z komercyjnych realizacji Uniksowych, np. Solaris i Irix mają wywłaszczalne jądra, ale tradycyjne implementacje nie były wywłaszczalne
- Linux ma specyficzne podejście do wielowątkowości. **Nie ma rozróżnienia wątków i zwykłych procesów.** Dla jądra wszystkie procesy są takie same, niektóre jedynie współdzielą zasoby

Windows. Moduł jądra (kernel):



Odpowiada za:

- Szeregowanie zadań
- Harmonogram realizacji wątków
- Obsługa sytuacji wyjątkowych
- Synchronizacja pracy wieloprocesorowej

Jądro kieruje wątki do wykonania na dostępnym procesorze

Każdy wątek ma przydzielany priorytet, wątki o wyższych priorytetach mogą wywłaszczać wątki o niższych priorytetach

moduł jądra nie jest stronicowany (nonpageable), strony nie są usuwane z pamięci do obszaru wymiany (pliku tymczasowego PAGEFILE.SYS)

kod modułu nie jest wywłaszczalny, natomiast pozostałe oprogramowanie np. stosowane w modułach wykonawczych Windows jest wywłaszczalne (preemptive)

Moduł jądra zarządza dwoma klasami obiektów:
obiektami dyspozytora

obiektami sterującymi

Moduł jądra może być wykonywany równocześnie na wszystkich procesorach komputerów wieloprocesorowych i sam synchronizuje dostęp do swoich krytycznych miejsc w pamięci.

[Windows] Wymienić 5 podmodułów modułu wykonawczego (EXECUTIVE) Windowsa.

- **Menadżer pamięci podręcznej (cache)** — poprawia wydajności wejścia/wyjścia plikowego, poprzez przechowywanie danych dopiero co odczytanych w pamięci głównej dla szybkiego dostępu do nich oraz poprzez opóźnianie zapisu — gromadzenie danych w pamięci przeznaczonych do zapisu na dysku.
- **Menadżer pamięci wirtualnej** — komponent implementujący pamięć wirtualną, czyli schematu zarządzania pamięcią dostarczającego dużej, prywatnej przestrzeni adresowej dla każdego procesu.
- **Menadżer I/O** — komponent implementuje niezależne od sprzętu wejście/wyjście oraz jest odpowiedzialny za dyspozycję sterownikami urządzeń.
- **Menedżer Plug and Play (PnP)** — określa wymagany sterownik dla konkretnego urządzenia po czym ładuje ten sterownik.
- **Menadżer zasilania** — koordynuje zdarzenia związane z zasilaniem oraz generuje notyfikacje zarządzania zasilaniem I/O przeznaczone dla sterowników urządzeń.
- **Menedżer konfiguracji** — komponent odpowiedzialny za implementację i zarządzanie rejestrem systemowym
- **Menedżer procesu i wątku** — komponent odpowiedzialny za tworzenie i usuwanie (przerywanie działania) procesów i wątków.
- **Monitor bezpieczeństwa** — egzekwuje politykę bezpieczeństwa na lokalnym komputerze. Chroni on zasoby systemu operacyjnego poprzez ochronę i audytowanie działających obiektów.
- **Funkcje Windows Management Instrumentation** — umożliwiają sterownikom urządzeń publikowanie informacji wydajnościowych oraz konfiguracyjnych oraz otrzymywanie poleceń z usługi WMI — trybu użytkownika.
- **Menedżer Obiektów** (Object Manager)
- **Wywołanie Procedury Lokalnej** (Local Procedura Call)

Usługi Windowsa

Usługi to programy użytkowe działające w tle Windows. Nie mamy do nich bezpośredniego dostępu, nie uruchamiamy ich jak zwykłych aplikacji. **Większość usług jest zarządzanych przez system,** a jako że Windows ma budowę modułową, **poszczególne usługi możemy włączać i wyłączać** według potrzeb. Czasem pozwala to zwolnić cenne zasoby systemowe - tak potraktowany system wyraźnie przyspiesza.

HAL

Warstwa abstrakcji sprzętowej (ang. hardware abstraction layer – HAL) – sterownik urządzenia dla płyty głównej. **Stanowi ogniwo pośredniczące między sprzętem a jądrem systemu operacyjnego.** Odseparowuje konkretną architekturę systemu komputerowego od oprogramowania użytkowego.

Wykład 2

Program - zawartość pliku na dysku

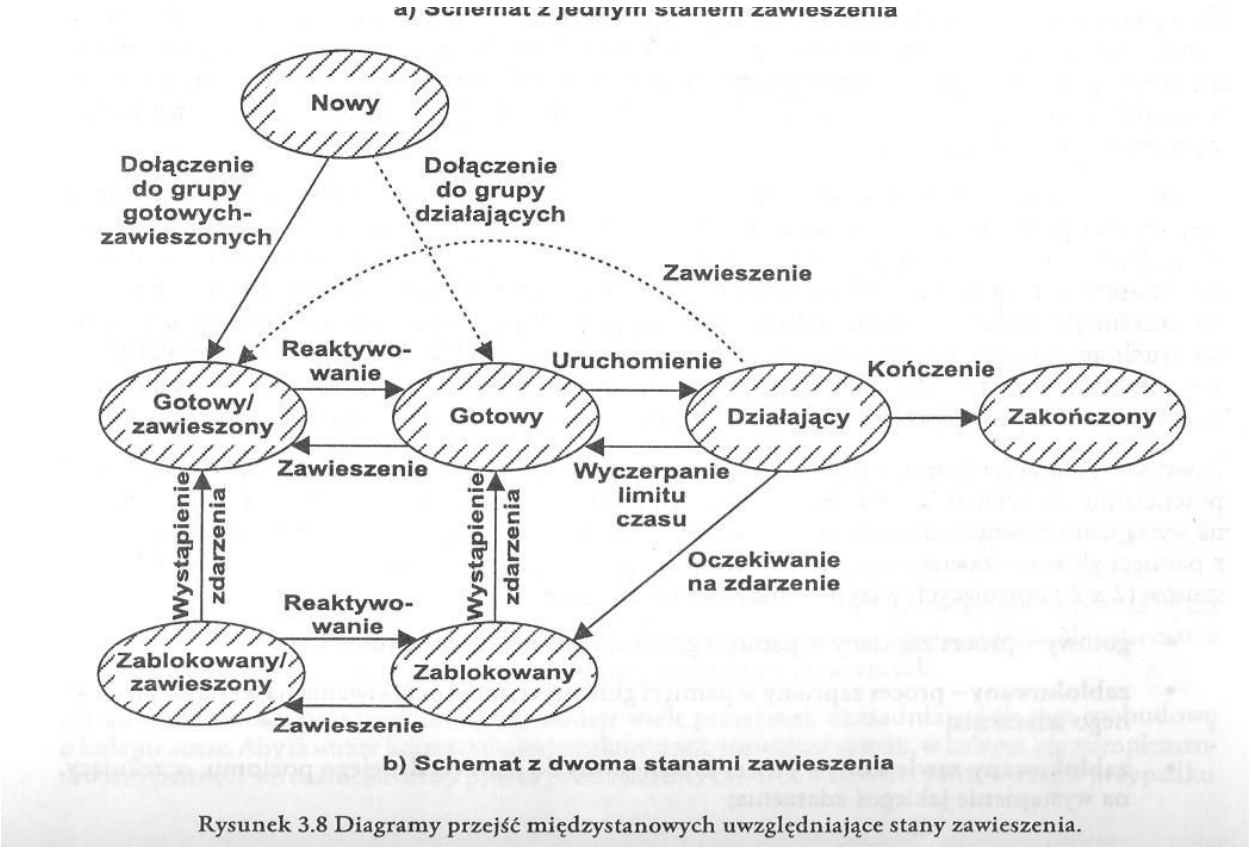
Proces - wykonujący się program

Proces składa się z:

- kod programu
- licznik rozkazów
- zawartość rejestrów procesora
- własny stos z danymi tymczasowymi
- sekcja danych (zmienne globalne)

Narysuj i skomentuj diagram stanów procesu (5-stanowy)

Nowy - proces został utworzony
Aktywny - są wykonywane instrukcje
Czekający - proces czeka na zakończenie jakiegoś zdarzenia (np. operacji we/wy)
Gotowy - proces czeka na przydział procesora
Zakończony - proces zakończył działanie



Rysunek 3.8 Diagramy przejść międzystanowych uwzględniające stany zawieszenia.

Tutaj idea jest taka że dla stanów gotowy i zablokowany=czekający dorabiamy analogiczne stany gotowy/zawieszony i zablokowany/zawieszony.

Chodzi o to żeby zakończenie operacji WE/WY mogło zmienić stan procesu zawieszonego..

Blok kontrolny procesu

- **PCB - Blok kontrolny procesu zawiera:**
  - Stan procesu
  - licznik rozkazów
  - rejestry
  - info o planowaniu przydziału procesora: priorytet, wskaźniki do kolejek itp.
  - info o zarządzaniu pamięcią: rejestry graniczne, tablice segmentów
  - info o stanie we/wy: otwarte pliki, urządzenia

Różne rodzaje kolejek:

Job queue – procesy w systemie
Ready queue – procesy gotowe do działania w pamięci głównej
Device queue – w kolejce na przydział do danego urządzenia (każde urządzenia ma **swoją** kolejkę)

Planiści

Proces systemowy który wybiera zadania z kolejki
Długoterminowy - z dysku ładuje do pamięci (kilka minut)
Krótkoterminowy - wybiera jeden proces z kolejki gotowych i przydziela mu procesor (kilka *ms*)

Procesy dzielimy na takie co zajmują się głównie WE/WY, i takie co liczą dużo w CPU.
Dobry planista długoterminowy robi ich idealny cocktail (mieszankę).
Jest też planista **średnioterminowy**, usuwający procesy z pamięci (swapping)

Przełączenie kontekstu

Zapisanie stanu starego procesu, załadowanie stanu nowego.

Tworzenie procesu

Proces macierzysty i potomek
Zasoby albo od OS, albo część zasobów procesu macierzystego
Proces macierzysty albo działa współbieżnie do potomka, albo czeka na zakończenie.
Może albo stać się kopią, albo otrzymać nowy program.
FORK,
CLONE - dziecko i rodzic dzielą się częściami kontekstu wykonywania czyli wirtualną przestrzeń adresową, tablicę deskryptorów plików i tablicę obsługi sygnałów.
VFORK - dziecko “pożycza” przestrzeń adresową rodzica, ten jest suspendowany, EXECVE, WAIT

Proces = zbiór wątków i zestaw zasobów

Ciężki proces = zadanie z jednym wątkiem

Wątki

Podstawowa jednostka wykorzystania procesora, licznik rozkazów + rejestry + obszar stosu, może współdzielić kod, dane, zasoby (**współpracujące wątki, bez mechanizmów ochrony)**. **Tanie przełączanie**

Wątki poziomu użytkownika

Jądro nie ma pojęcia o ich istnieniu **(taki trochę ukryty romans procesu z wielowątkowością)**
Realizacja z użyciem biblioteki.
Synchronizacja, szeregowanie i zarządzanie bez udziału jądra
Zapamiętywanie kontekstu bez udziału jądra
Własny stos, przestrzeń na kontekst rejestrów i stan wątku.

Jak działa szeregowanie? you might ask
Jądro szereguje procesy. Wewnątrz tych procesów proces sam szereguje sobie wątki, stosując np. Pthreads.
Jeżeli jest wywłaszczany proces, to wątki razem z nim.
Funkcja blokująca w wątku powoduje zablokowanie wszystkich wątków procesu.

naturalny sposób	**Brak przepływu informacji między jądrem i biblioteką wątków**

zapisu programów	**Trochę jakby dwóch ślepych grało w berka – brak ochrony dostępu między wątkami, nieefektywne szeregowanie** **Konieczność synchronizacji** **brak jednoczesnego wykonywania** (na maszynie wieloprocesorowej)

Wątki jądra:romans

przełączanie kontekstu między wątkami jądra jest szybkie, ponieważ nie trzeba zmieniać odwzorowań pamięci

- zastosowania wątków jądra
  - wątki jądra są przydatne do wykonywania pewnych operacji takich, jak asynchroniczne wejście-wyjście
  - zamiast udostępniać osobne operacje asynchronicznego we/wy, jądro może realizować je, tworząc odrębny wątek do wykonania każdego zlecenia
  - wątki poziomu jądra można wykorzystywać też do obsługi przerwań

wątek jądra nie musi być związany z procesem użytkownika
jądro tworzy go i usuwa wewnętrznie w miarę potrzeb
wątek odpowiada za wykonanie określonej czynności, współdzieli tekst jądra i jego dane globalne, jest niezależnie szeregowany i wykorzystuje standardowe mechanizmy jądra, takie jak sleep() czy wakeup()
wątki potrzebują jedynie własnego stosu, przestrzeni do przechowywania kontekstu rejestrów
tworzenie i stosowanie wątków jądra nie jest kosztowne
ć (nie zużywają zasobów jądra)

Proces lekki - wspierany przez jądro wątek poziomu użytkownika

- **wysokopoziomowe pojęcie abstrakcyjne oparte na wątkach jądra** - system udostępniający procesy lekkie, musi także udostępniać wątki jądra
- **każdy proces lekki jest związany z wątkiem jądra**, ale niektóre wątki jądra są przeznaczone do realizacji pewnych zadań systemowych, wtedy nie przypisuje się im żadnych lekkich procesów
- **w każdym procesie może być kilka procesów lekkich**, każdy wspierany przez oddzielny wątek jądra, współdzielą one przestrzeń adresową i inne zasoby procesu
- **procesy lekkie są niezależnie szeregowane przez systemowego planistę**,
- procesy lekkie mogą wywoływać funkcje systemowe, które powodują wstrzymanie w oczekiwaniu na we/wy lub zasób
- proces działający w systemie wieloprocesorowym może uzyskać rzeczywistą równoległość wykonania, **każdy proces lekki może być uruchamiany na oddzielnym procesorze**
- wielowątkowe procesy są przydatne wtedy, gdy każdy wątek jest w miarę niezależny i rzadko porozumiewa się z innymi wątkami

	Wymagają użycia funkcji systemowych - kosztowne (bo zmiana trybu, przejście przez granice ochrony) **Możliwość zmonopolizowania procesora** **Zużycie znaczących zasobów jądra**

Szeregowanie wątków

Mamy dwa schedulery:

- **Dla procesów i wątków systemowych KS (kernel scheduler)**
- **Dla wątków użytkownika UTS (user threads scheduler)**

Modele wielowątkowości

1:N

KS widzi jeden wątek na proces - nic nie wie o romansie
UTS widzi N wątków - poligamicznym romansie
To logiczne, jądro nie ma udziału w wielowątkowości w tym modelu

1:1

KS widzi wszystkie wątki otwarty związek? Nie wiem, ta analogia się chyba poszła jebać 5 stron temu :(
UTS nie jest potrzebny
Logiczne, jądro zajmuje się całą wielowątkowością
Tworzenie i synchronizacja nadal biblioteka, ale scheduling to już rola jądra

M:N

Mapowanie N wątków użytkownika (N>M) na M wątków jądra
Biada temu kto odważy się to implementować :(

Jakie problemy napotkano pisząc bibliotekę do M:N wielowątkowości.

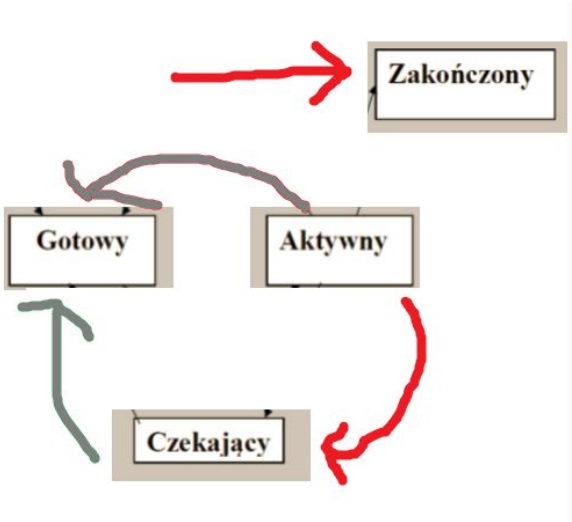
- Problem komunikacji pomiędzy UTS a KS - np.: UTS przydziela wątek użytkownika o wyższym priorytecie, aby wykonywał się na powiązanym wątku jądra, ale w tym samym czasie KS przełącza wątek jądra, bo jego czas już wygasł, potencjalnie przydzielając procesor wątkowi użytkownika o mniejszym priorytecie.
- Problem ilości wątków jądra - jeśli będzie za mało, to nie skorzystamy w pełni z wielowątkowości: wątki użytkownika zostaną bez roboty; jeśli za dużo, to tracimy sporo czasu na przełączanie kontekstu.

PLANOWANIE PRZYDZIAŁU PROCESORA

5 zasad przy planowaniu procesora

- **Wykorzystanie procesora** – procesor powinien być zajęty pracą
- **Przepustowość** (ang. throughput) – liczba procesów kończonych w jednostce czasu
- **Czas cyklu przetwarzania** (ang. turnaround time) – czas upływający między nadejściem procesu do systemu i zakończeniem przetwarzania procesu;suma czasów czekania na wejście do pamięci, czekania w kolejce procesów gotowych do wykonania, wykonania procesu przez procesor i wykonywania operacji we/wy
- **Czas oczekiwania** – suma okresów oczekiwania przez proces w kolejce procesów gotowych do wykonania (algorytm planowania ma wpływ na tę wielkość)
- **Czas odpowiedzi** (ang. response time) – czas upływający między wysłaniem żądania i pierwszą odpowiedzią

<u>Kiedy zapadają decyzje o przydziale procesora:</u>

Wywłaszczające	**Niewywłaszczające**
 Proces może przejść ze stanu aktywności do stanu gotowości w wyniku przerwania Inny proces może przejść do do stanu gotowości ze stanu czekania i np. mieć większy priorytet	 Proces sam musi oddać procesor - albo się zakończyć, albo przejść do stanu czekania

Ekspedytor przekazuje procesor do procesu, przełącza kontekst, przełącza do trybu użytkownika, wznawia program na podstawie zapisanych rejestrów

Algorytmy planowania

1. **FCFS = *First Come First Served***
   #NIEWYWŁASZCZAJĄCY #INSTABOY

   Proces który pierwszy zamawia procesor pierwszy go otrzymuje.

	Długi czas oczekiwania **Krótkie procesy muszą czekać na długie** **Efekt konwoju (wszystkie procesy czekają na zwolnienie procesora przez jeden duży proces)** **Jak ktoś jechał za tirem na autostradzie to wie**

2. **SJF = *Shortest Job First***

Ten proces który ma najkrótszy następny przydział procesora jest pierwszy
Jak stoisz w kolejce do lekarza, i starsza pani wbija się przed tobą "Ja tylko jedno pytanie" //xDDD
#OPTYMALNY
Może być wywłaszczający (bo pojawił się proces o krótszym zapotrzebowaniu na kolejny cykl) lub niewywłaszczający

3. **Priorytetowe**
   - Wywłaszczający lub niewywłaszczający
   - Ten z większym priorytetem jest pierwszy.
   - W razie równych priorytetów -> FCFS.
   - Problem nieskończonego blokowania -> procesy z l*ow prio* czekają w nieskończoność
     - ***SOLUTION***! : postarzanie -> podwyższanie priorytetów procesów długo oczekujących

   Sposób definicji priorytetu:

   - **Wewnętrzny** – używa się mierzalnej właściwości procesu (np. limity czasu, wielkość obszaru wymaganej pamięci, liczba otwartych plików, stosunek średniej fazy we/wy do średniej fazy procesora)
   - **Zewnętrzny** – kryteria zewnętrzne wobec SO (np. ważność procesu, rodzaj i kwota opłat użytkownika, instytucja sponsorująca pracę itd.)

4. **Gwarantowane (Guaranteed Scheduling)**
   - dla n użytkowników: każdy dostaje 1/n czasu procesora
   - System musi dysponować następującymi informacjami:
     - jak wiele czasu CPU użytkownik otrzymał dla wszystkich swoich procesów od czasu zalogowania się
     - jak długo każdy użytkownik jest zalogowany
     - czas procesora przysługujący użytkownikowi jest ilorazem:

czas zalogowania użytkownika/ilość zalogowanych użytkowników

- Określenie czasu przypadającego na proces/użytkownika
    - wyliczenie stosunku dotąd przyznanego czasu CPU do czasu przysługującego
        - współczynnik 0.5 = użytkownik wykorzystał tylko połowę przysługującego mu czasu
        - współczynnik 2.0 = użytkownik otrzymał dwa razy więcej czasu, niż mu przysługuje
        - proces jest uruchamiany z najniższym priorytetem, gdy jego współczynnik jest wyższy od współczynników innych procesów

Systemy czasu rzeczywistego:

- może być zastosowana podobna idea
- procesy których ‘deadline’ się zbliża otrzymują priorytety gwarantujące im pierwszeństwo wykonania

5. **<u>Loteryjne</u>**
    - daje niezłe wyniki, przy znacznie prostszej od algorytmu **Guaranteed Scheduling** realizacji
    - każdy proces dostaje los do różnych rodzajów zasobów (np. właśnie CPU)
    - podczas procesu planowania wybierany jest (metoda losową) los (bilet loteryjny ang. lottery tickets) i proces, który jest jego właścicielem dostaje zasób
    - bardziej istotne procesy mogą mieć przyznaną większą ilość losów, co zwiększa ich szanse zwycięstwa w losowaniu

Właściwości planowania loteryjnego:

- jeżeli procesowi zwiększy się ilość losów -> szansa zwycięstwa rośnie
- współpracujące procesy \mogą wymieniać losy między sobą np.
    - klient wysyła komunikat do procesu serwera i przechodzi w stan oczekiwania na wynik
    - może następnie przekazać swoje losy serwerowi co zwiększa szansę, że serwer zostanie szybko wykonany
    - po obsłużeniu zlecenia przez serwer, zwraca on losy klientowi,
    - klient zostaje uruchomiony
    - w przypadku braku klientów serwer nie potrzebuje losów

Planowanie loteryjne może być używane do rozwiązywania problemów, które trudno rozwiązać innymi metodami.

6. **<u>Round Robin = Rotacyjne</u>**
    - podobny do FCFS dodano wywłaszczenie
    - kolejka cykliczna
    - proces dostaje CPU na odcinek czasu i po jego upływie przenoszony na koniec kolejki
    - czas oczekiwania - dość wysoki
    - wydajność zależy od kwantu czasu

<u>Wielopoziomowe planowanie kolejek</u>

- procesy są zaliczane do kilku grup np. procesy pierwszoplanowe (ang. foreground)- interakcyjne, oraz drugoplanowe (ang. background) - wsadowe.

- kolejka procesów gotowych jest rozdzielona na osobne kolejki
- Każda kolejka ma swój algorytm planujący np FCFS, Round Robin itp.

Planowanie wielopoziomowych kolejek ze sprzężeniem zwrotnym

Planowanie wielopoziomowych kolejek ze sprzężeniem zwrotnym umożliwia przemieszczanie się procesów między kolejkami.
Idea: rozdzielenie procesów o różnych rodzajach faz procesora.

- proces, który zużywa za dużo czasu procesora, zostaje przeniesiony do kolejki o niższym priorytecie
- np., 3 kolejki, im zadanie ~w kolejce o niższym priorytecie, tym na dłużej może uzyskać procesor:
    - 1 - algorytm z kwantem 8
    - 2 - algorytm z kwantem 16
    - 3 - algorytm FCFS

Parametry określające planistę wielopoziomowych kolejek ze sprzężeniem zwrotnym:

- liczba kolejek
- algorytm planowania dla każdej kolejki
- metoda użyta do decydowania o awansie procesu do kolejki o wyższym priorytecie
- metoda użyta do zdymisjonowania procesu do kolejki o niższym priorytecie
- metoda wyznaczania kolejki, do której trafia proces

Na jakiej podstawie można przydzielić proces do innej kolejki w planowaniu wielopoziomowym?

Przemieszczanie procesów z jednej kolejki do drugiej jest możliwe w planowaniu ze sprzężeniem zwrotnym.

- Jeśli proces zużywa za dużo czasu procesora, to zostanie przeniesiony do kolejki o niższym priorytecie.
- proces oczekujący zbyt długo w nisko priorytetowej kolejce może zostać przeniesiony do kolejki o wyższym priorytecie.

Taki sposób postarzania procesów zapobiega ich głodzeniu.

Planowanie wieloprocesorowe:

- Dwie metody:
    - Każdy procesor planuje swoje działanie
    - asymetryczne wieloprzetwarzanie=wyróżniony jeden procesor - serwer główny
        - serwer główny podejmuje wszystkie decyzje planistyczne, wykonuje operacje we/wy i czynności systemowe
        - pozostałe procesory wykonują tylko kod użytkowy

Planowanie wątków

- **współdzielenie obciążenia (Load Sharing)**
    - stosowana jest globalna kolejka gotowych do wykonania wątków, procesor gdy jest wolny wybiera wątek z kolejki
    - Zalety:

● Obciążenie rozproszone równomiernie między procesory ● Nie jest wymagany centralny planista

- Globalna kolejka może być obsługiwana przy użyciu dowolnego algorytmu

- **Szeregowanie grupowe (zespołowe) (Gang scheduling)**
  - zbiór wątków tego samego procesu jest jednocześnie wykonywany na wielu procesorach
  - Zalety:
    - Jednoczesne szeregowanie wątków tworzących jeden proces
    - Bardzo korzystne w przypadku aplikacji równoległych, których wydajność znacznie spada, gdy część aplikacji nie działa razem z innymi
    - Przydatne także w przypadku aplikacji mniej wrażliwych na kwestie wydajnościowe
    - Szeregowanie grupowe minimalizuje przełączenia procesów (wątki nie muszą oddawać procesora w oczekiwaniu na działania np. zwolnienia muteksów, wysłanie danych ze strony innych wątków)

- **Rezerwacja procesorów (Dedicated processor assignment)**
  - przydział wątków do wykonania na konkretnych procesorach

- **Dynamiczne szeregowanie (Dynamic scheduling)**
  - ilość wątków w procesach może być zmieniana w trakcie wykonania

Wady i zalety jednej kolejki do wielu procesorów i osobnej dla każdego

Osobna kolejka dla każdego:

Zapobieganie powstawania wąskiego gardła.	Procesor z pustą kolejką byłby bezczynny, podczas gdy inny procesor miałby nadmiar pracy.

Wspólna kolejka dla wszystkich procesorów:

Każdy procesor ma zawsze zadanie do wykonywania.	Dwa procesory mogą wybrać ten sam proces z kolejki.

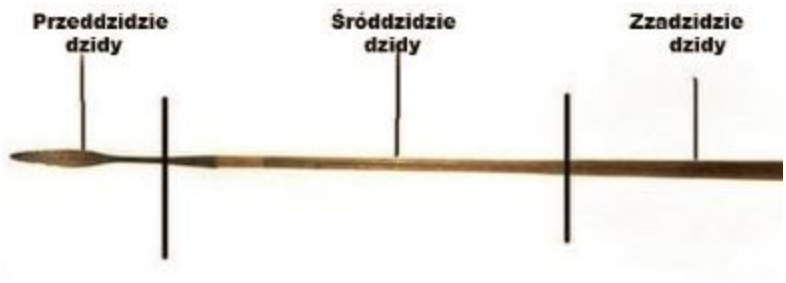
Wykład 3

Sekcja krytyczna

Segment kodu w którym można aktualizować wspólne dane.
Tylko jeden proces jednocześnie może przebywać w swojej sekcji krytycznej.
Wykonanie sekcji krytycznej podlega **wzajemnemu wykluczeniu (mutual exclusion)**
Potrzebny jest protokół określający współpracę między procesami

Rodzaje kodu

Sekcja wejściowa
Sekcja krytyczna
Sekcja wyjściowa
Reszta



Warunki sekcji krytycznej

1. **Wzajemne wykluczenie** – jeżeli proces Pi działa w sekcji krytycznej (SK) to żaden inny proces nie może działać w SK.
2. **Postęp** – **jeżeli żaden proces nie działa w SK** i istnieją procesy, które chcą wejść do SK to tylko **procesy niewykonujące swoich reszt mogą kandydować o wejście do SK**, a **wybór procesu nie może być odwlekany w nieskończoność**.

Chodzi o to żeby wchodziły do niej procesy nie mające nic do roboty (nie wykonujące reszty) żeby nie zrobiło się takie sekcje-krytyczne zagłodzenie.

3. **Ograniczone czekanie** – musi istnieć wartość graniczna liczby wejść innych procesów do ich sekcji krytycznych po tym, gdy dany proces zgłosił chęć wejścia do swojej SK i zanim uzyskał pozwolenie.

Algorytm sekcji krytycznej dla 2 procesów

```
zainteresowany[0] = false;
zainteresowany[1] = false;
czyja_kolej;
```

```
P0: zainteresowany[0] = true;
    czyja_kolej = 1;
    while (zainteresowany[1] && czyja_kolej == 1)
    {
        // czekaj
    }
    // sekcja krytyczna
    ...
    // koniec sekcji krytycznej
    zainteresowany[0] = false;
```

```
P1: zainteresowany[1] = true;
    czyja_kolej = 0;
    while (zainteresowany[0] && czyja_kolej == 0)
    {
        // czekaj
    }
    // sekcja krytyczna
    ...
    // koniec sekcji krytycznej
    zainteresowany[1] = false;
```

Algorytm piekarni

1. Przy wejściu do sklepu klient dostaje numerek.
2. Obsługa rozpoczyna od klienta z najmniejszym numerem.

3. Jest szansa, że dwa klienci dostaną ten sam numerek, wtedy obsługiwany jest ten z wcześniejszą nazwą.

Algorytm piekarni - realizacja

```
{wspólne struktury danych}
var wybrane: array[0..n-1] of boolean;
        numer: array[0..n-1] of integer;
        for i:=0 to n-1 do wybrane[i]:= false;
        for i:=0 to n-1 do numer[i]=0;
repeat
        wybrane[i]:= true;
        numer[i]:=max(numer[0],numer[1],...,numer[n-1])+1;
        wybrane[i]:=false;
        for j:=0 to n-1
                do begin
                while wybrane[j] do nic;
                while numer[j] ≠ 0
                        and (numer[j],j) < (numer[i],i) do nic
                end
        sekcja krytyczna
        numer[i]:=0
        reszta
until false
```

Semafory - narzędzie synchronizacji

Semafor zliczający

Zmienna całkowita, możemy **inicjalizować, czekać, sygnalizować**
Tutaj implementacja z aktywnym czekaniem – procesy usiłują wejść do sekcji krytycznej muszą wykonywać instrukcje pętli marnując cykle procesora.

```
Czekaj:
   while S <= 0: czekaj;
   S = S - 1;

Sygnalizuj:
   S = S + 1;
```

Te operacje są tak niepodzielne

Możemy je zastosować do rozwiązania problemu sekcji krytycznej - taki wspólny semafor (mutex) jest dzielony przez n procesów. Na początku mutex=1, operacja czekaj go dekrementuje.

```
repeat
      czekaj(mutex)
      sekcja krytyczna
      sygnalizuj(mutex)
until false
```

Jeden problem, a imię jego aktywne czekanie
Marnuje cykle procesora, tzw. Spinlock
Może to być spoko w systemach wieloprocesorowych ((bo nie jest marnowany czas na przełączanie kontekstu)), ale nadal trochę głupie.

Rozwiązanie:
Lista czekających semaforów

```
type semaphore = record
    wartosc:integer;
    L: list of proces; {gdy proces musi czekac pod semaforem, to jest dolaczany do
listy}
end

czekaj(S):
    S.wartosc:=S.wartosc-1;
    if S.wartosc<0 then begin
        dolacz dany proces do S.L;
        blokuj;
    end;

sygnalizuj(S):
    S.wartosc:=S.wartosc+1;
    if S.wartosc <= 0 then begin
        usun jakis proces P z S.L;
        obudz(P);
    end;
```

W tej realizacji wartość semafora może być ujemna, liczba na minusie oznacza wtedy ilość procesów czekających na semafor.

Semafor binarny

Może przyjmować dwie wartości: 0 i 1

Da się zaimplementować semafor zliczający z użyciem dwóch binarnych i dodatkowego integera:

```
czekaj(S):

czekaj(S1);
C:=C-1;
if C<0  then begin
        sygnalizuj(S1);
                czekaj(S2);
        end
sygnalizuj(S1);
```

```
sygnalizuj(S):

czekaj(S1);
C:=C+1;
if C≤0
else  then sygnalizuj(S2)
        sygnalizuj(S1);
```

Monitor

Składnik programu złożony z deklaracji zmiennych wspólnych dzielonych przez kooperujące procesy oraz zbioru wszystkich procedur operujących na tych zmiennych. Może istnieć możliwość definiowania typów monitorowych, i potem utworzenia wielu monitorów danego typu

```
var identyfikator-monitora: monitor
  deklaracje-zmiennych;
  procedury;
  lista-udostępnionych-na-zewnątrz-nazw-
  procedur;
end.
```

monitor – kolekcja wszystkich regionów krytycznych związanych z tą samą zmienną dzieloną, zebranych w jednym module

Monitor zapewnia mutual exclusion jego procedure
Ma też operacje kolejkowe:
var kol:queue
delay(kol) zawieszenie procesu wykonującego procedurę w kolejce kol, z opuszczeniem monitora i umożliwieniem wejścia innego procesu
continue(kol) jeżeli w kolejce jest proces, to jest on przenoszony do kolejki wejściowej z pierwszeństwem, wznawiany od miejsca wywołania delay
clear(kol) czyści kolejkę
empty(kol) = isEmpty

Dwa rodzaje:
Z sygnalizacją: continue natychmiast wznawia **(- dodatkowe przełączenia stanu)**
Z powiadamianiem i rozgłaszaniem: continue zastąpione przez notify, wznowienie gdy będzie wolny monitor, można też uruchomić wszystkie procesy oczekujące w kolejce danego warunku (broadcast)

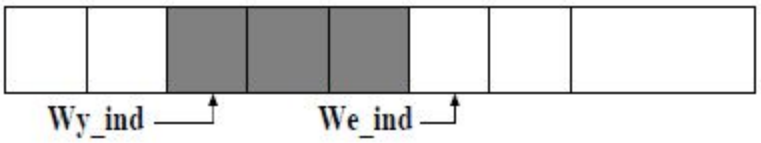
Monitory z sygnalizacją (C. Hoar) – Funkcja signal/continue wywołana przez proces w monitorze wznawia wykonanie procesu zawieszonego funkcją delay/wait, proces wywołujący sygnał opuszcza monitor (kończąc wykonanie funkcji monitora) lub zostaje zablokowany – Wady: • gdy proces wykonujący signal nie kończy swego działania z monitorem, wówczas potrzebne są dwa dodatkowe przełączenia stanu procesu – jego zawieszanie i wznowienie • Szeregowanie procesów związanych z sygnałem musi być niezawodne •

Monitory z powiadamianiem i rozgłaszaniem – powiadamianie: procedura signal zastąpiona przez notify, informuje odpowiednią kolejkę o zajściu warunku, proces z kolejki zostaje wznowiony nie natychmiast, ale wtedy, gdy będzie taka możliwość (wolny monitor) – rozgłaszanie (broadcast): umożliwia uruchomienie wszystkich procesów oczekujących w kolejce związanej z danym warunkiem, gdy będą ku temu możliwości

Problemy synchronizacyjne

Problem producentów i konsumentów

- rozwiązanie wykorzystujące semafor reprezentujący liczbę elementów w buforze
- B: array(0..nieskończoność) of Integer;
- We_ind, Wy_ind: Integer:=0;
- Elementy: Semefor:=0;




```
task body Producent is
        I: Integer;
begin
        loop
            Produkuj(I);
            B(We_ind):=I;
            We_ind:=We_ind+1;
            Sygnalizuj(Elementy)
        end loop
end Producent
```

```
task body Konsument is
        I: Integer;
begin
    loop
        Czekaj(Elementy);
        I:=B(Wy_ind);
        Wy_ind:=Wy_ind+1;
        Konsumuj(I);
    end loop
end Konsument
```

Oczywiście w prawdziwej implementacji wartałoby dodać mod n.

Problem czytelników i pisarzy

Problemy synchronizacji: Problem czytelników i pisarzy

- **Czytelnicy -** procesy nie wykluczają się nawzajem
- **Pisarze -** procesy wykluczają każdy inny proces (zarówno czytelnika jak i pisarza)

```
task body Czytelnik is
begin
   loop
        Zacznij_czytanie;
        Czytaj_dane;
        Zakończ_czytanie;
   end loop;
end Czytelnik

task body Pisarz is
begin
   loop
        Zacznij_pisanie;
        Zapisz_dane;
        Zakończ_pisanie;
   end loop;
end Pisarz
```

Jeżeli są oczekujący pisarze, to nowy czytelnik musi zaczekać (co najmniej) do zakończenia pisania przez pierwszego pisarza.no
Oczekujący czytelnicy będą wznawiane przed następnym pisaniem.
Tutaj rozwiązanie na monitorach:

```
monitor Monitor_czytelnicy_pisarze is
  Czytelnicy: Integer:=0;{Licznik czytelników, którzy czytają}
  Pisanie: Boolean:=False; {Czy jakiś proces pisze}
  Można_czytać, można_pisać: Warunek; {wstrzym. czyt. i pisarzy}

procedure Zacznij_czytanie is
begin
   if Pisanie or Niepusta(Można_pisać) then
        Czekaj(Można_czytać);
   end if
   Czytelnicy:=Czytelnicy+1;
   Sygnalizuj(Można_czytać);{Kaskada wznowień}
end Zacznij_czytanie

procedure Zakończ_czytanie is
begin
   Czytelnicy:=Czytelnicy-1;
   if Czytelnicy=0 then Sygnalizuj(Można_pisać);end if;
end Zakończ_czytanie
...

 procedure Zacznij_pisanie is
 begin
    if Czytelnicy != 0 or Pisanie then
              Czekaj(Można_pisać);
       end_if
    Pisanie:=true;
 end Zacznij_pisanie

 procedure Koniec_pisania is
 begin
    Pisanie:=False;
    if Niepusta(Można_czytać) then
        Sygnalizuj(Można_czytać);
    else
        Sygnalizuj(Można_pisać)
    end if;
 end Koniec_pisania;

 end Monitor_czytelnicy_pisarze
```

Problem 5 filozofów

Przykład rozwiązania: Rozwiązanie z ograniczeniem liczby filozofów przebywających w jadalni.

```
Jadalnia: Semafor:=4;
Widelec: array(0..4) of Semafor;
task body Filozof is
begin
   loop
       Myśl;
       Czekaj(Jadalnia);
       Czekaj(Widelec(I));
       Czekaj(Widelec((I+1) mod 5));
       Jedz;
       Sygnalizuj(Widelec(I));
       Sygnalizuj(Widelec((I+1) mod 5));
       Sygnalizuj(Jadalnia);
      end loop;
end Filozof;
```

Wykład 4 - Zakleszczenia

Zakleszczenie

Wiele procesów rywalizuje o skończone zasoby
Gdy proces nie może wyjść ze stanu oczekiwania, bo zamawiane przez niego zasoby są przetrzymywane przez inne procesy.

Porządek używania zasobu

Zamówienie (w tym czekanie aż się uwolni) **> Użycie > Zwolnienie**
Robimy to funkcjami systemowymi lub semaforami

Warunki konieczne zajścia zakleszczenia

Wzajemne wykluczanie **Istnieje niepodzielny zasób, proces musi**	**Przetrzymywanie i oczekiwanie** **Istnieje proces, który ma jeden zasób, i**

poczekać aż inny proces przestanie go używać	**oczekuje na zwolnienie kolejnego**
Brak wywłaszczeń **Tylko proces przetrzymujący zasób może go zwolnić**	**Czekanie cykliczne** **Istnieje cykl procesów, każdy proces czeka na zasób przetrzymywany przez poprzedni proces.**

Graf przydziałów

Graf skierowany, którego wierzchołkami są zasoby i procesy, ma 2 rodzaje krawędzi

Wierzchołki:

P={P1..Pn} – zbiór procesów systemu

Z={Z1..Zm} – zbiór zasobów

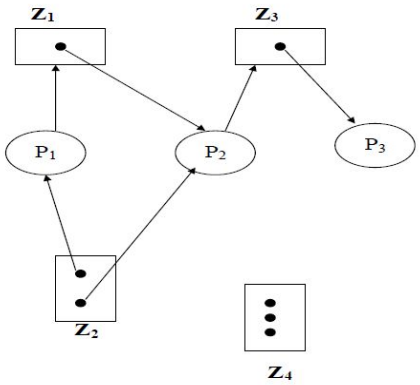
Krawędzie:

- **krawędź zamówienia** (ang. request edge) Pi-> Zj - krawędź skierowana od procesu do typu zasobu, oznacza, że **proces Pi zamówił egzemplarz zasobu typu Zj** i **obecnie czeka na ten zasób**
- **krawędź przydziału** (ang. assignment edge) Zj-> Pi - krawędź skierowana od typu zasobu do procesu, oznacza, że **egzemplarz zasobu typu Zj został przydzielony do procesu Pi**

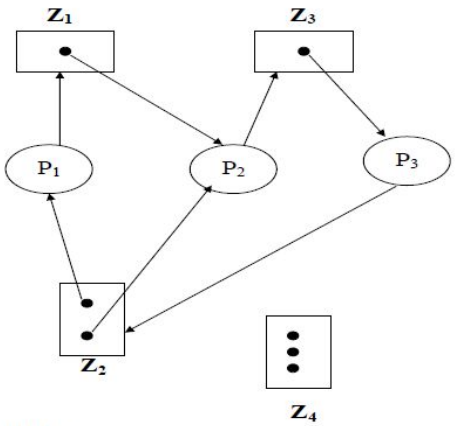
Reprezentacja graficzna:

+ każdy proces Pi jest przedstawiany w postaci kółka
+ każdy typ zasobu Zj jest przedstawiany w postaci prostokąta
+ każdy egzemplarz zasobu jest oznaczony kropką umieszczoną w prostokącie
+ **krawędź zamówienia sięga do brzegu prostokąta Zj,**
+ **krawędź przydziału musi wskazywać na jedną z kropek w prostokącie.**

Graf przydziału zasobów – przykład 1



Graf przydziału zasobów – przykład 2

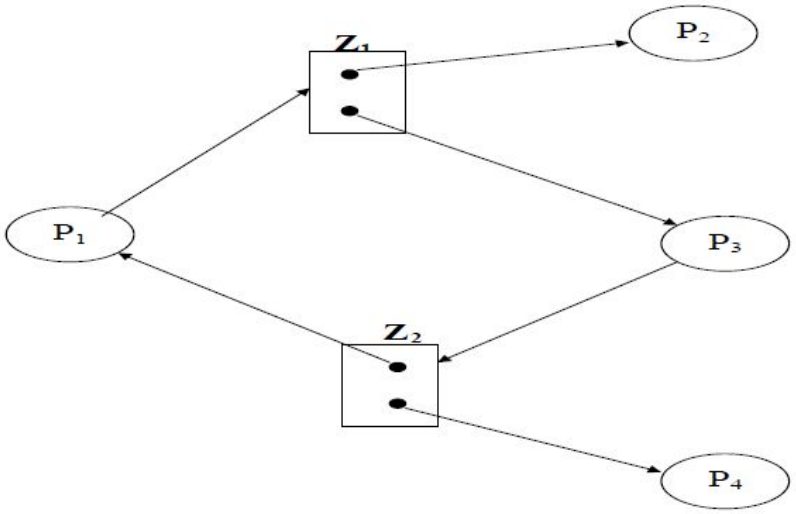


niejąc cykle:

P1→Z1→P2→ Z3 → P3→ Z2→ P1

P2→ Z3 → P3→ Z2→P2

Graf przydziału zasobów z cyklem, ale bez zakleszczenia



Istnieje cykl: P1→Z1→P3→Z2→P1

Nie ma zakleszczenia -- proces P4 może zwolnić egzemplarz zasobu Z2, który może zostać przydzielony procesowi P3, co rozerwie cykl.

Prawdziwe są 3 fakty:

- **+ jeśli graf nie zawiera cykli, to w systemie nie ma zakleszczonych procesów**
- **+ jeśli graf zawiera cykl, to w systemie może dojść do zakleszczenia**
- **+ jeżeli zasób każdego typu ma tylko jeden egzemplarz, to zakleszczenie jest faktem**

Co robić z zakleszczeniem:

Zapobieganie zakleszczeniom:

Przynajmniej jeden warunek konieczny do wystąpienia zakleszczenia nie ma być spełniony. Nakładamy ograniczenia na sposób zamawiania zasobów. (- słabe wykorzystanie urządzeń, ograniczona przepustowość systemu)

Unikanie zakleszczeń

SO trzyma dodatkowe informacje, dla każdego zamówienia decydujemy czy można je zrealizować.

Można też w razie czego usuwać zakleszczenia lub zakładać że się nie pojawią.

Metody zapobiegania zakleszczeniom

Uniemożliwienie WK Wzajemne wykluczanie:
np. użycie zasobów dzielonych które nie powodują zakleszczeń
Uniemożliwienie WK Przetrzymywanie i oczekiwanie:
Proces zamawia i dostaje wszelkie potrzebne zasoby, zanim rozpoczyna działanie. **(- niskie wykorzystanie zasobów, głodzenie)**
Uniemożliwienie WK Brak wywłaszczeń:

Gdy proces ma zapotrzebowanie na zasób którego nie można mu natychmiast dać, traci wszystkie zasoby. Wznawia się go, gdy można mu dać nowe zasoby + te które miał na początku.
Uniemożliwienie WK Czekanie cykliczne:
Każdemu zasobowi przyporządkowuje się liczbę naturalną, nie da się zająć zasobu, jeżeli mamy już zasoby o większym numerze

Metody unikania zakleszczeń

NP: Każdy proces deklaruje maksymalną liczbę zasobów danego typu, algorytm jest w stanie zagwarantować, że nigdy nie dojdzie do zakleszczenia

Stan bezpieczny

Stan systemu jest bezpieczny, jeżeli istnieje porządek przydzielenia zasobów każdemu procesowi nawet w stopniu maksymalnym, tak aby uniknąć zakleszczenia.
Wtedy i tylko wtedy gdy istnieje **ciąg bezpieczny. To uszeregowanie procesów, gdzie zapotrzebowanie każdego procesu może być zaspokojone przez zasoby wolne i zajęte przez procesy przed nim.**

Stan zagrożenia - nie istnieje ciąg procesów o podanych właściwościach

Może prowadzić do zakleszczeń

Algorytm grafu przydziału zasobów

Wierzchołki:
P={P1..Pn} – zbiór procesów systemu
Z={Z1..Zm} – zbiór zasobów
Krawędzie:

- **krawędź zamówienia** (ang. request edge) Pi-> Zj - krawędź skierowana od procesu do typu zasobu, oznacza, że **proces Pi zamówił egzemplarz zasobu typu Zj** i **obecnie czeka na ten zasób**
- **Krawędź deklaracji - jak krawędź zamówienia, ale linią przerywaną - proces może kiedyś zamówić ten zasób**
- **krawędź przydziału** (ang. assignment edge) Zj-> Pi - krawędź skierowana od typu zasobu do procesu, oznacza, że **egzemplarz zasobu typu Zj został przydzielony do procesu Pi**

Gdy proces zamawia zasób, to krawędź deklaracji -> krawędź zamówienia
Gdy proces zwalnia zasób, to krawędź przydziału-> krawędź deklaracji (w przeciwną stronę)

Algorytm jest prosty, jeżeli powstałby cykl to nie możemy zamienić krawędzi zamówienia na krawędź przydziału

algorytm bankiera

Idea
• gdy proces wchodzi do systemu – musi zadeklarować maksymalną liczbę egzemplarzy każdego typu zasobu, które będą mu potrzebne
• zadeklarowane liczby nie mogą przekraczać ogólnych ilości zasobów w systemie

• przy każdym zamówieniu zasobów przez proces, system określa, czy ich przydzielenie pozostawi system w stanie bezpiecznym

– jeśli tak: zasoby zostaną przydzielone
– jeśli nie: proces musi zaczekać, aż inne procesy zwolnią wystarczającą liczbę zasobów

Likwidowanie zakleszczeń

Zakończenie wszystkich zakleszczonych procesów
Usuwanie pojedynczo, może od razu nie zadziałać (powtarzamy), jakie kryteria doboru?

§ - wybieramy tak aby minimalizować koszty, wycofujemy proces (albo do stanu bezpiecznego, albo całkowicie)

Możemy też mieszać te metody, określając klasy zasobów dla których stosujemy różne algorytmy.

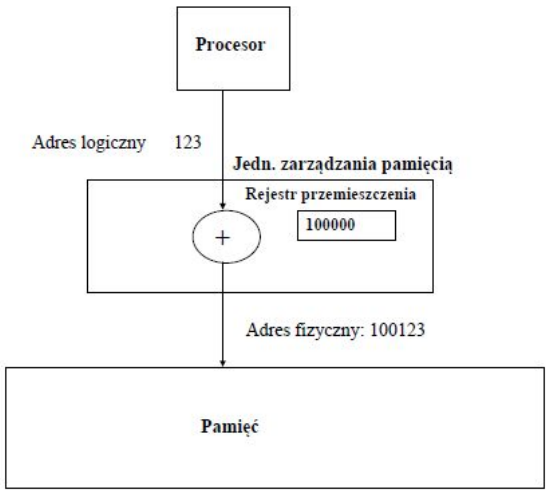
Wykład 5

Cykl wykonania rozkazu

Pobranie z pamięci > dekodowanie > ew. pobranie argumentów > wykonanie operacji > zapis wyników

Pamięć to wielka struktura oznaczona adresami
Adres logiczny - wytworzony przez procesor = adres wirtualny
Adres fizyczny - rzeczywiste umiejscowienie w pamięci
Dwie przestrzenie adresowe: logiczna i fizyczna to zbiory wszystkich adresów

MMU - memory management unit - moduł tłumaczący adresy z logicznych na fizyczne
Rejestr przemieszczenia - wartość dodana do adresu logicznego daje adres fizyczny (inaczej wartość bazy)



Wiązanie adresów

Skojarzenie zmiennych w programie z adresami względnymi (liczonymi od początku modułu) lub bezwzględnymi
Kiedy to robimy:
Przy kompilacji - możemy się odwoływać do adresów bezwzględnych
Przy ładowaniu - tzw. Kod przemieszczalny
Przy wykonaniu - podczas wykonania proces może przemieszczać się w pamięci

Ładowanie dynamiczne

Ładujemy podprogram do pamięci w momencie wykonania.

Nie zostanie załadowany nieużywany program **Gdy okazjonalnie wykonujemy duży fragment kodu (np. Obsługa błędu)** **Można przechowywać w pamięci tylko fragment** **Nie wymaga specjalnego wsparcia ze strony OS**	

Konsolidacja bibliotek

Statyczna - dołączamy je do binarnego obrazu programu **(-duplikacja kodu)**
Dynamiczna - zostaje opóźniona, w miejscu wywołania znajduje się stub (namiastka) procedury, która wskazuje jak znaleźć lub załadować odpowiedni kawałek kodu. **(+jedna kopia, aktualizowanie jest łatwe, wymaga wsparcia OS)**

Nakładki

Kawałek kodu zastępujący program już w pamięci operacyjnej **(trudne do zaprojektowania)**

Wymiana

Proces wymienialny - przesyłany z pamięci do pamięci pomocniczej i odwrotnie, zazwyczaj powraca w to samo miejsce, chyba że wiązanie nie jest przy kompilacji.

	Jej realizacja zajmuje dużo czasu

Rzadko stosowana, w unix standardowo zabroniona

Przydział ciągły

Pojedynczego obszaru - ochrona danych i kodu procesów przed zmianami
Do rejestru przemieszczenia dochodzi rejestr graniczny
Wielu obszarów

Problem dynamicznego przydziału pamięci

First fit- pierwsza dziura o odpowiedniej wielkości, szukanie od początku lub od punktu zatrzymania
Best fit - najmniejsza z dostatecznie dużych dziur (trzeba przejrzeć całą listę, najmniejsze pozostałości)
Worse fit - największa dziura, pozostałości mogą być bardziej użyteczne

Fragmentacja

Zewnętrzna - zostają za małe kawałki pamięci
Wewnętrzna - nie opłaca się trzymać informacji o małych kawałkach, zostają przydzielone do zajętych większych
Upakowanie - takie przemieszczenie pamięci by uzyskać duży blok (zmiana adresów)
Stronicowanie - inna metoda radzenia sobie z zewn. Fragmentacją

Stronicowanie

Pamięć fizyczną dzielimy na ramki
Pamięć logiczną na strony
Tej samej długości
Przy wykonywaniu procesu strony w pamięci pomocniczej wprowadzamy do ramek pamięci operacyjnej

Adres to para (s,o) - numer strony i offset

Tablica stron - adresy bazowe wszystkich stron w pamięci operacyjnej

Zawiera też bit ochrony (can write, bit poprawności - należy do procesu, nie należy do procesu)
Nie eliminujemy wewnętrznej fragmentacji, **tablica ramek** zawiera informacje czy ramki pamięci operacyjnej są puste, i do jakiego procesu są przydzielone

• Ominięcie problemu dopasowania kawałków pamięci o zmiennych rozmiarach do miejsca w pamięci pomocniczej • eliminuje się zewnętrzną fragmentację • wspomaganie dla współdzielenia i ochrony pamięci.	stronicowanie nie eliminuje wewnętrznej fragmentacji, straty z nią związane mogą osiągać maksymalnie: rozmiar ramki – 1 słowo/bajt na proces system operacyjny potrzebuje informacji na temat stanu pamięci fizycznej (wolne i zajęte ramki) informacje te są przechowywane w tablicy ramek (ang. *frame table*) stronicowanie oddziela pamięć oglądaną przez użytkownika od pamięci fizycznej

	Wydłuża czas przełączania kontekstu narzut czasowy przy transformacji adresu, narzut pamięciowy (na potrzeby tablicy stron)

Stronicowanie wielopoziomowe

Dlaczego:

- tablica stron może być b. duża (~ milion wpisów)
- podział tablicy stron na mniejsze części, by uniknąć pełnego ciągłego obszaru w pamięci operacyjnej

- Realizacja:
  - stronicowanie dwupoziomowe -- tablica stron jest podzielona na strony

Wyliczenie adresu fizycznego (na podstawie logicznego) w stronicowaniu

Adres fizyczny = wartość bazy (z rejestru bazowego tablicy stron) + adres logiczny

Adres wirtualny stanowi trójka:
<identyfikator-procesu, numer-strony, odległość>
Wpis w odwróconej tablicy stron jest parą:
<identyfikator-procesu, numer-strony>

Odwrócona tablica stron

Jedna pozycja dla każdej rzeczywistej strony pamięci
Zawiera adres wirtualny i informacje o procesie, jedna na system, jest to tablica haszująca

Segmentacja

- **Segmentacja** – schemat zarządzania pamięcią, w którym:
  - **przestrzeń adresów logicznych jest zbiorem segmentów**
  - każdy segment ma swoją nazwę i długość
  - adresy logiczne określają nazwę segmentu i odległość wewnątrz segmentu

- różnica między stronicowaniem i segmentacją:
  - **segmentacja: użytkownik określa każdy adres za pomocą nazwy segmentu i jego długości**
  - **stronicowanie: użytkownik określa tylko pojedynczy adres, który jest dzielony przez sprzęt na numer strony i odległość w sposób niewidoczny dla użytkownika**
  - segmenty są numerowane

• adres logiczny : <numer-segmentu, odległość>

powiązanie ochrony pamięci z segmentami Dzielenie kodu od danych - segmenty są określonymi semantycznie fragmentami programów, można zakładać, że wszystkie dane w segmencie będą wykorzystywane w taki sam sposób: **segmenty kodu i segmenty danych** **Możliwe jest dzielenie tylko fragmentów programów**- wspólne pakiety podprogramów mogą być dzielone między wielu użytkowników jeśli zdefiniuje się je jako segmenty dzielone, przeznaczone tylko do czytania. **z każdą pozycją w tablicy segmentów są związane bity ochrony,** ○ **segment do odczytu/modyfikowania** ○ **segment danych/do wykonywania**	segmentacja może spowodować **zewnętrzną fragmentację**, jeśli każdy z bloków wolnej pamięci jest za mały, by pomieścić cały segment → trzeba czekać na zwolnienie pamięci lub zastosować upakowanie

<u>Co to są bufory translacji adresów stron (ang. Translation lookaside buffers (TLB))?</u>

Problem: czas potrzebny na dostęp do komórki pamięci użytkownika przy przechowywaniu tablicy stron w pamięci operacyjnej, gdzie do wskazywania położenia służy rejestr bazowy tablicy stron.

Standardowym rozwiązaniem tego problemu jest zastosowanie specjalnej, małej i szybko przeszukiwanej, sprzętowej pamięci podręcznej, zwanej rozmaicie – rejestrami asocjacyjnymi (ang. associative registers) lub buforami translacji adresów stron (ang. translation look-aside buffers - TLBs)

Zbiór rejestrów asocjacyjnych jest zbudowany z wyjątkowo szybkich układów pamięciowych. Każdy rejestr składa się z dwu części: klucza i wartości. Porównywanie danego obiektu z kluczami w rejestrach asocjacyjnych odbywa się równocześnie dla wszystkich kluczy. Jeśli obiekt zostanie znaleziony, to wynikiem jest odpowiadające danemu kluczowi pole wartości. Szukanie to jest szybkie, jednak stosowny sprzęt - drogi. Liczba pozycji w buforze TLB wynosi zazwyczaj od 8 do 2048. Rejestry asocjacyjne są używane wraz z tablicą stron w następujący sposób. Rejestry asocjacyjne zawierają tylko kilka wpisów z tablicy stron. Gdy procesor utworzy adres logiczny, wówczas wynikający z niego numer strony jest porównywany ze zbiorem rejestrów asocjacyjnych, które zawierają numery stron i odpowiadających im ramek. Jeśli numer strony zostanie odnaleziony w rejestrach asocjacyjnych, to odpowiedni numer ramki uzyskuje się natychmiast i używa go przy dostępie do pamięci. W porównaniu z bezpośrednim dostępem do pamięci całe zadanie może wydłużyć się o mniej niż 10%.

Wykład 6

Pamięć wirtualna

- **Pamięć wirtualna** - technika umożliwiająca wykonywanie procesów, które nie są w całości w pamięci operacyjnej.

Możliwość utworzenia dużej abstrakcyjnej pamięci głównej → Możliwość uruchomienia programów które są większe niż pamięć fizyczna	Obniżenie wydajności systemu

-Optymalny algorytm

Algorytm optymalny:
- cechuje go najniższy współczynnik braków stron
- nie jest dotknięty anomalią Belady'ego
- używany głównie do studiów porównawczych

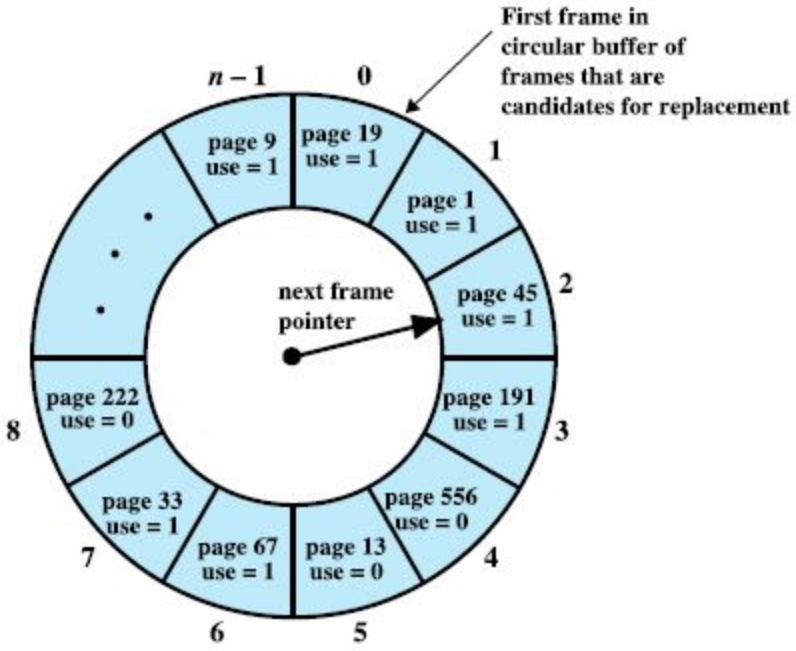
Idea algorytmu: zastąpiona zostaje strona, która najdłużej nie będzie używana.
Wady algorytmu idealnego:
- trudność realizacyjna
- wymaga wiedzy o przyszłej postaci ciągu odniesień.

-LRU na czym polega i (2 rodzaje zastosowania?)

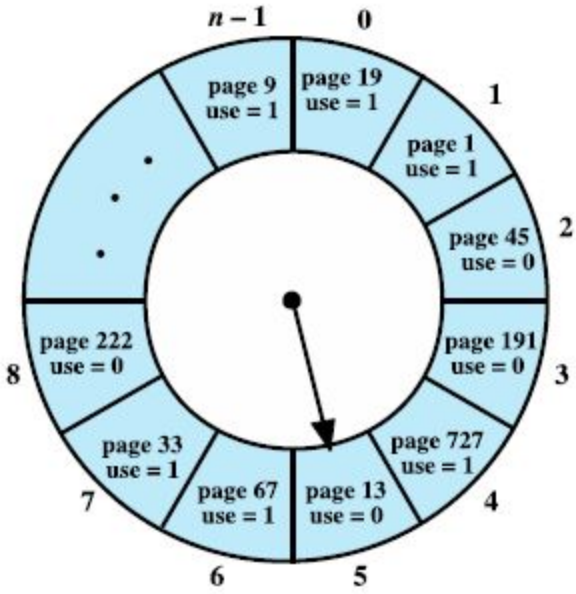
Algorytm LRU (zastępowanie najdawniej używanych stron, ang. least recently used)
- zastępowana jest strona, która najdawniej nie była używana
- do oszacowania sytuacji w przyszłości jest używane badanie stanów w niedawnej przeszłości
- z każdą stroną jest kojarzony czas jej ostatniego użycia
- algorytm stosowany dość często, uznawany za dość dobry
- trudność: implementacja zastępowania stron wg. kolejności wynikającej z algorytmu LRU
    1. (może być konieczne znaczne wsparcie sprzętowe;
    2. problem, jak określić kolejność ramek na podstawie ich ostatniego użycia)
- zastępowanie stron metodą LRU nie jest dotknięte anomalią Belady'ego

Sposoby realizacji LRU:
- zastosowanie licznika
- zastosowanie stosu

Napisać algorytm zegarowy.



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Algorytm zastępowania stron na zasadzie drugiej szansy

- algorytm oparty na algoritma FIFO
  - po wybraniu strony sprawdza się jej wartość jej bitu odniesienia
- jeżeli jest 0, to strona zostaje zastąpiona

- jeżeli jest 1, to strona pozostaje w pamięci, ale jej bit odniesienia jest zerowany, a jej czas przybycia ustawiany na czas bieżący

Sposoby realizacji algorytmu drugiej szansy

- kolejka cykliczna
- wskaźnik pokazuje na stronę, która może być zastąpiona w następnej kolejności
- po zapotrzebowaniu na ramkę wskaźnik przemieszcza się naprzód, aż zostanie odnaleziona strona z wyzerowanym bitem odniesienia
- podczas przemieszczania wskaźnika, analizowane bity odniesienia są zerowane

Ulepszony algorytm drugiej szansy

- brane są pod uwagę: bit odniesienia i bit modyfikacji, są one traktowane jak uporządkowana para

Istnieją 4 klasy stron

- (0,0) – ostatnio nie używana i nie zmieniana – najlepsza strona do zastąpienia
- (0,1) – ostatnio nie używana, ale zmieniona – nieco gorsza kandydatka, gdyż trzeba ją będzie zapisać na dysk przed zastąpieniem
- (1,0) – używana ostatnio, ale nie zmieniona – niedługo może być używana
- (1,1) – używana ostatnio i zmieniona – niedługo może być używana, w przypadku wymiany trzeba ją będzie zapisać na dysk

Działanie algorytmu

- postępowanie analogiczne jak w algorytmie zegarowym, ale sprawdza się, do której klasy strona należy
- zastępowana jest pierwsza napotkana strona z najniższej niepustej klasy

Wykład 7

Warstwowy system plików

- Programy użytkowe
- Logiczny system plików
- Moduł organizacji pliku
- Podstawowy system plików
- Sterowanie wejściem-wyjściem
- Urządzenia

[przydział miejsca na dysku] Wady i zalety przydziału ciągłego

Zalety:

- liczba operacji przeszukiwania dysku przy dostępie do danych – minimalna
- czas przeszukiwania – minimalny
- dostęp do pliku: można się odnosić bezpośrednio do i-tego bloku pliku

Wady:

- zewnętrzna/wewnętrzna fragmentacja
- problem znalezienia miejsca na nowy plik

[przydział miejsca na dysku] Wady i zalety indeksowego systemu plików

Zalety:

- przydział indeksowy umożliwia dostęp bezpośredni bez powodowania zewnętrznej fragmentacji
- aby przeczytać blok i jest używany wskaźnik z pozycji o numerze i w bloku indeksowym i odnajduje się odpowiedni blok

Wady:

- marnowanie przestrzeni, wskaźniki indeksowe zawierają na ogół więcej miejsca od wskaźników w przydziale listowym

[przydział miejsca na dysku] Wady i zalety listowego przydzielania pamięci

Zalety:

- każdy plik jest listą powiązanych ze sobą bloków dyskowych, które mogą być dowolnie rozmieszczone na dysku
- katalog zawiera wskaźniki do pierwszego i ostatniego bloku w pliku
- przydział listowy nie ma zewnętrznej fragmentacji, plikowi mogą być przydzielone dowolne bloki z listy wolnych bloków; nie trzeba deklarować rozmiaru pliku w momencie jego tworzenia

Wady:

- ograniczenie jego efektywnego zastosowania do plików o dostępie sekwencyjnym (aby znaleźć i-ty blok pliku trzeba zacząć od początku i dojść do i-tego bloku – $0(n)$)
- w każdym bloku trzeba poświęcić pewien obszar na reprezentację wskaźnika co sprawia, że pliki zajmują więcej miejsca
- powyższa niedogodność jest usuwana przez grupowanie bloków w tzw. grona/klastry i przydzielanie plikom klastrów zamiast bloków, co również polepsza przepustowość, ale powoduje wzrost wewnętrznej fragmentacji
- błąd wskaźnika mógłby spowodować dowiązanie pliku do innego pliku lub do listy wolnych obszarów

-Informacje w i-węźle o systemie plików

FREE BSD: i-węzeł zawiera:

- typ pliku i tryb dostępu
- identyfikatory dostępu właściciela i grupy
- czas stworzenia, czasy ostatniego odczytu i zapisu
- rozmiar pliku

- sekwencja wskaźników na bloki
- liczba bloków i liczba wpisów w katalogu
- rozmiar bloków danych
- flagi ustawiane przez jądro i użytkownika
- numer generacyjny przypisany plikowi
- rozmiar rozszerzonej informacji dotyczącej atrybutów
- zero lub więcej wpisów z rozszerzonymi atrybutami

Wykład 8

[planowanie dostępu do dysku] Planowanie metodą SCAN i C-SCAN

Planowanie metodą SCAN:

- głowica startuje od jednego końca dysku i przemieszcza się do przeciwległego końca, potem zmienia kierunek ruchu itd.
- rozważamy skrajne położenie głowicy: największe zagęszczenie niezrealizowanych zadań dotyczy przeciwległego końca dysku.

Planowanie metodą C-SCAN:

- głowica przemieszcza się od jednego końca dysku do drugiego, obsługująca napotkanie zamówienia.
- gdy osiągnie przeciwległy koniec: natychmiast wraca do początku dysku.

[planowanie dostępu do dysku] SSTF

Planowanie metodą SSTF (najpierw najkrótszy czas przeszukiwania – *shortest seek-time first*)

- łączenie obsługi wszystkich zamówień sąsiadujących z bieżącym położeniem głowicy

W algorytmie SSTF wybiera się zamówienie z minimalnym czasem przeszukiwania względem bieżącej pozycji głowicy. Ponieważ czas przeszukiwania wzrasta proporcjonalnie do liczby cylindrów odwiedzanych przez głowicę, w algorytmie SSTF wybiera się zamówienie najbliższe bieżącemu położeniu głowicy.

[planowanie dostępu do dysku] Wyjaśnić: LOOK, C-LOOK.

Planowanie metodą LOOK, C-LOOK:

- głowica przemieszcza się nie pomiędzy krańcami dysku, ale skrajnymi zamówieniami.

Zauważmy, że zgodnie z podanym przez nas opisem zarówno w planowaniu SCAN, jak i C-SCAN głowica przemieszcza się zawsze od jednego skrajnego położenia na dysku do drugiego. W praktyce żaden z tych algorytmów nie jest implementowany w ten sposób. Najczęściej głowica przesuwa się do skrajnego zamówienia w każdym kierunku. Natychmiast potem głowica wykonuje zwrot, nie dochodząc do skrajnego położenia na dysku. Takie wersje algorytmów SCAN i C-SCAN nazywają się odpowiednio LOOK i C-LOOK, ponieważ przed kontynuowaniem ruchu, „patrzy się” w nich, czy w danym kierunku znajduje się jakieś zamówienie