Spis treści

Jak coś da się przyporządkować do jakiejś wyraźnej kategorii, to piszmy coś w stylu [nazwaKategorii] Treść pytania
Przykładowo:
[stronicowanie] Wyliczenie adresu fizycznego w stronicowaniu
[planowanie] Round robin
Tutaj jest kopia opracowania z odpowiedziami na resztę pytań (praca w toku)
https://docs.google.com/document/d/10olodg9exsKieiR6ryC6ZZXlF70HJDJynMBzEuapeil/edit
Nie chciałam wrzucać tutaj od razu bo nie wiem czy są poprawne, lub nie wiem czy dobrze zrozumiałam treść pytania (jak więcej osób zatwierdzi je wrzuce ładne odp tutaj)

Wykład 1
-System operacyjny jako rozszerzona maszyna.
-Cechy podejścia warstwowego
[mikrojądro] Cechy mikrojądra
[mikrojądro] Zalety mikrojądra
[mikrojądro] Wady mikrojądra
[Linux] Wymień przynajmniej pięć innowacji w System V Release 4
[Linux] Cechy charakterystyczne jądra linuxa
[Windows] Funkcje jądra (kernela) w Windowsie
[Windows] Jakie funkcje pełni moduł jądra w systemie Windows
[Windows] Wymienić 5 podmodułów modułu wykonawczego Windowsa.
[Windows] Usługi Windowsa

Wykład 2. Zarządzanie procesami: procesy, planowanie przydziału procesora
-Narysuj i skomentuj diagram stanów procesu (5-stanowy)
[process, process lekki, wątek] Czym są wątki jądra i do czego służą?
[process, process lekki, wątek] Co to jest lekki proces? I jakie jest jego powiązanie z wątkami użytkownika?
[process, process lekki, wątek] Ograniczenie wątków użytkownika bez wsparcia jądra
[process, process lekki, wątek] Jakie problemy napotkano pisząc bibliotekę do M:N wielowątkowości.
[planowanie] 5 zasad przy planowaniu procesora
[planowanie] Decyzje, które mają miejsce przy planowaniu wywłaszczeniowym, a których nie ma w planowaniu niewywłaszczeniowym
[planowanie] Opisać planowanie gwarantowane
[planowanie] Omów algorytm drugiej szansy
[planowanie] Planowanie priorytetowe, jak sobie radzono z zagłodzeniem procesów.
[planowanie] Planowanie loteryjne
[planowanie] Round robin

[planowanie] Zalety Gang Scheduling
[planowanie] Na jakiej podstawie można przydzielić proces do innej kolejki w planowaniu wielopoziomowym
[planowanie] planowanie wielopoziomowych kolejek ze sprzężeniem zwrotnym
[planowanie] omówić FIFO przydzielanie i jak eliminuje się problem zagłodzenia
[planowanie] wady i zalety jednej kolejki do wielu procesorów i osobnej dla każdego

Wykład 3. Zarządzanie procesami: Synchronizacja
[sekcja krytyczna] Warunki sekcji krytycznej
[sekcja krytyczna] Algorytm sekcji krytycznej dla 2 procesów
[sekcja krytyczna] rozwiązanie problemu sekcji krytycznej
[semafory] napisać kody semaforów aktywnie czekających (czekaj, sygnalizuj)
[semafory] Napisać algorytm bez aktywnego czekania - semafory.

Wykład 4. Zarządzanie procesami: Zakleszczenia
-graf przydziałów
-Co zrobić, by zapobiec czekaniu cyklicznemu na zasoby przez procesy?
-Zapobieganie przetrzymywaniu i oczekiwaniu.
-Omówić algorytm bankiera.

Wykład 5. Zarządzanie pamięcią: informacje wstępne, wymiana
[stronicowanie] Zalety stronicowania.
[stronicowanie] Stronicowanie wielopoziomowe - dlaczego stosujemy?
[stronicowanie] Wyliczenie adresu fizycznego (na podstawie logicznego) w stronicowaniu.
-Zalety segmentowania.
-Co to są bufory translacji adresów stron (ang. translation look-aside buffers) (TLB)

Wykład 6. Pamięć wirtualna
-Optymalny algorytm wymiany stron.
-LRU na czym polega i (2 rodzaje zastosowania?).
-Napisać algorytm zegarowy.

Wykład 7. System plików - interfejs i realizacja
[przydział miejsca na dysku] Wady i zalety przydziału ciągłego.
[przydział miejsca na dysku] Wady i zalety indeksowego systemu plików.
[przydział miejsca na dysku] Wady i zalety listowego przydzielania pamięci.
-Informacje w i-węźle o systemie plików.

Wykład 8. Systemy wejścia-wyjścia
[planowanie dostępu do dysku] Planowanie metodą SCAN i C-SCAN.
[planowanie dostępu do dysku] SSTF.
[planowanie dostępu do dysku] Wyjaśnić: LOOK, C-LOOK.

Pytania nieprzyporządkowane jeszcze albo nieokreślone

-wady zalety dla wszystkich procesorów/wady zalety dla 1 kolejka = 1 procesor

---

Opracowanie

Wykład 1

System operacyjny jako rozszerzona maszyna.

– architektura komputerów na poziomie języka maszynowego jest trudna do użycia w programach (szczególnie: operacje wejścia/wyjścia)
– zadaniem systemu operacyjnego jest ukrycie tej złożoności i dostarczenie programiście bardziej przyjaznego interfejsu
– system operacyjny udostępnia maszynę rozszerzoną /wirtualną, łatwiejszą do programowania

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
- podejście warstwowe ułatwia wyszukiwanie błędów i weryfikację systemu
- Możliwe wady realizacji warstwowych: mniejsza wydajność, obecnie ogranicza się liczbę warstw
- Przykład systemu warstwowego: system THE
    - Warstwa 5: Programy użytkowe
    - Warstwa 4: Buforowanie urządzeń wejścia i wyjścia
    - Warstwa 3: Program obsługi kontroli operatora
    - Warstwa 2: Zarządzanie pamięcią
    - Warstwa 1: Planowanie przydziału procesora
    - Warstwa 0: Sprzęt

[mikrojądro] Cechy mikrojądra

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. Serwery

komunikują się między sobą i z aplikacjami za pośrednictwem operacji komunikacji międzyprocesowej dostarczanych przez mikrojądro.

[mikrojądro] Zalety mikrojądra

● **Jednolite interfejsy-** procesy nie muszą odróżniać usług jądra od usług użytkownika, gdyż wszystkie usługi są udostępniane za pomocą przekazywania komunikatów.
● **Rozszerzalność** – architektura mikrojądra sprzyja rozszerzalności, gdyż pozwala dodawać nowe usługi oraz obsługiwać zróżnicowane istniejące usługi; dodanie nowej funkcji wymaga dodania nowego serwera lub modyfikacji istniejącego, nie trzeba generować nowego jądra
● **Elastyczność** – można zarówno dodawać nowe usługi i rozbudowywać system jak też usuwać pewne usługi, aby uzyskać mniejszą i wydajniejszą realizację
● **Przenośność między platformami-** całość lub duża część kodu zależnego od architektury sprzętowej znajduje się w mikrojądrze, łatwość przenoszenia na inną architekturę, gdyż trzeba zmieniać tylko dobrze określone, nieliczne moduły
● **Stabilność-** niewielkie mikrojądro może być dokładnie przetestowane
● **Obsługa systemów rozproszonych-** komunikaty skierowane do serwera usługi mogą dotyczyć zarówno serwerów zlokalizowanych na tym samym komputerze jak i na innych
● **Obsługa systemów obiektowych** – rozwiązania obiektowe wymuszają większą dyscyplinę podczas projektowania mikrojądra oraz modularnych rozszerzeń.

[mikrojądro] Wady mikrojądra
Wydajność:

- Komunikacja - intensywnie wykorzystywane jest jądro. Komunikacja klient-serwer np. manager okienek > system plików (system plików jest tutaj usługą serwerem) wygląda tak: klient (wysyła zapytanie) > jądro > serwer (wysyła odp) > jądro > klient. Komunikacja między jądrem a serwerami użytkownika wymaga przełączania kontekstów, co jest bardzo kosztowne.
- Synchronizacja - w przypadku synchronizacji działań dokonywanych przez serwery poziomu użytkownika

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

- Wsparcie dla TCP/IP
- Sockety
- UFS (Unix File System)
- Wsparcie dla wielu grup
- C shell
- Korn shell
- Wsparcie standardów POSIX oraz X/Open
- x86 sterowniki
- możliwość wykonywania różnego rodzaju plików wykonywalnych
- osobne sposoby obsługi procesów z podziałem czasu i procesów realtime

- wprowadzenie vnodów, co pozwala na obsługę wielu systemów plików
- Nowa implementacja pamięci wirtualnej ze stronicowaniem w trybie kopiowania przy zapisie
- Pamięć dzielona

[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 wywlaszczalne. Z komercyjnych realizacji Uniksowych, np. Solaris i Irix mają wywlaszczalne jądra, ale tradycyjne implementacje nie były wywlaszczalne
- Linux ma specyficzne podejście do wielowątkowości. Nie ma rozróżnienia wątków i zwykłych procesów. Dla jądra wszystkie procesy są takie same, niektóre jedynie współdzielą zasoby

(Tego nie było)
[Windows] Funkcje jądra (kernela) w Windowsie.

(z książki:)

- Planowanie procesów
- obsługa przerwań i sytuacji wyjątkowych
- synchronizacja procesora na niskim poziomie
- podejmowanie działania naprawczuch po awarii zasilania.

(jak zrozumiałem, raczej chodzi o samo jądro a nie moduł)
Windows. Moduł jądra:

- Odpowiada za szeregowanie zadań, harmonogram realizacji wątków, obsługę sytuacji wyjątkowych i synchronizację 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] Jakie funkcje pełni moduł jądra w systemie Windows

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

[Windows] Wymienić 5 podmodułów modułu wykonawczego Windowsa.

- **Menadżer konfiguracji** — komponent odpowiedzialny za implementację i zarządzanie rejestrem systemowym
- **Menadżer 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.
- **Menadżer I/O** — komponent implementuje niezależne od sprzętu wejście/wyjście oraz jest odpowiedzialny za dyspozycję sterownikami urządzeń.
- **Menadżer 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.
- **Funkcje Windows Management Instrumentation** — umożliwiają sterownikom urządzeń publikowanie informacji wydajnościowych oraz konfiguracyjnych oraz otrzymywanie poleceń z usługi WMI — trybu użytkownika.
- **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.
- **Menedżer Obiektów (Object Manager)** - nadzoruje użytkowanie wszystkich obiektów, w tym obiektów katalogowych, obiektów dowiązań symbolicznych, obiektów semaforów, zdarzeń, procesów, wątków, portów i plików.
- **Wywołanie Procedury Lokalnej (Local Procedure Call) -** przekazywanie zamówień i wyników między procesami klienta i serwera na tej samej maszynie.

Z opracowania do zerówki:

1. **Object Manager** - tak jak w Unixie wszystkie zasoby reprezentuje plik, tak w Windowsie reprezentują **obiekty**. Jest to bardzo zbliżony model do tego co znamy z programowania obiektowego.

2. **Cache Controller** - wspólne miejsce do obsługi pamięci podręcznej pamięci, I/O i sterowników
3. **Configuration Manager** - implementuje Rejestr (regedit i te sprawy)
4. **I/O Manager** - implementuje niezależnie od sprzętu całą obsługę wejścia/wyjścia i dysponuje sterownikami urządzeń
5. **Local Procedure Call (LPC)** - taki mechanizm IPC
6. **Memory Manager** - zarządza pamięcią wirtualną, ochroną pamięci, stronicowaniem
7. **Process Structure** - obsługuje tworzenie i usuwanie procesów i wątków (wykonywanie to zadanie jądra). Implementuje koncept *Job*'a, grupy procesów które mogą być usunięte jako całość, coś jak linkowanie w Erlangu.
8. **PnP Manager (Plug and Play)** - obsługuje wykrywanie urządzeń i automatyczne wczytywanie sterowników. Sporo funkcjonalności działa też w obszarze użytkownika, gdzie np. odpalany jest Windows Update w celu pobrania brakujących sterowników
9. **Power Manager** - obsługuje zdarzenia związane z zasilaniem, usypia CPU, kontroluje sterowniki za pomocą specjalnych przerwań
10. **Security Reference Monitor** - obsługuje wszystkie sprawy związane z egzekwowaniem uprawnień itp.
11. **GDI** - mój ulubieniec. Prastary framework do rysowania, GUI i renderowania czcionek. Siedzi w kernelu bo dawno temu dawało to sporą wydajność.

[Windows] Usługi Windowsa

Wykład 2

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

[process, process lekki, wątek] Czym są wątki jądra i do czego służą?

Wątki jądra - lekkie procesy, działające asynchronicznie w przestrzeni jądra (podobnie jak zwykłe wątki w przestrzeni użytkownika), niezwiązane z żadnym procesem użytkownika.
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

Dodatkowe informacje:

- 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 następujących zasobów:
    - ❖ własnego stosu,
    - ❖ przestrzeni do przechowywania kontekstu rejestrów w czasie, gdy wątek nie jest wykonywany
- tworzenie i stosowanie wątków jądra nie jest kosztowne
- przełączanie kontekstu między wątkami jądra jest szybkie, ponieważ nie trzeba zmieniać odwzorowań pamięci
- wątki poziomu jądra nie są nowym pomysłem: procesy systemowe jak demon stronicujący (pagedaemon) pełnią w tradycyjnych jądrach uniksowych te funkcje co wątki jądra

[process, process lekki, wątek] Co to jest lekki proces? I jakie jest jego powiązanie z wątkami użytkownika?

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

[process, process lekki, wątek] Ograniczenie wątków użytkownika bez wsparcia jądra.

Wątki użytkownika: Ograniczenia

- ograniczenia wynikają głównie z braku przepływu informacji między jądrem i biblioteką wątków
    - ❖ jądro nie ma informacji o wątkach użytkownika, nie może używać swoich mechanizmów ochrony do ich ochrony przed niedozwolonym dostępem ze strony innych wątków
    - ❖ szeregowanie przez jądro i bibliotekę, przy czym żadne z nich nie posiada wiedzy o czynnościach drugiego

- każdy proces działa we własnej przestrzeni adresowej, wątki użytkownika nie mają takiej ochrony – biblioteka wątków musi zapewnić mechanizmy synchronizacji
- niemożność jednoczesnego wykonywania wątków, w przypadku maszyny wieloprocesorowej

[process, process lekki, wątek] Jakie problemy napotkano pisząc bibliotekę do M:N wielowątkowości.

(poprawcie to proszę, tłumaczyłem z ang na polski :D)

- Problem komunikacji pomiędzy UTS a KS - np.: UTS przełącza na bardziej znaczący wątek użytkownika, żeby on wykonywał się na potocznym wątku jądra, i w tym samym czasie KS przełącza wątek jądra, bo jego czas już wygasł tym samym dając wykonywać się mniej znaczącemu wątku użytkownika
- 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.

The many-to-many model employs two schedulers: one in the kernel and one in the user threads library. It is not immediately obvious how the kernel scheduler can cooperate with the user scheduler. For example, say the user scheduler has a high-priority thread to schedule, so it preempts the execution of a lower-priority thread, reassigning its kernel thread to the high-priority user thread. But, at the same time, the kernel scheduler decides that the kernel thread's time slice has expired and reassigns the processor to another kernel thread, perhaps one that has been assigned by our user-level scheduler to a lower-priority thread. Thus the thread deemed the most important by the user-level scheduler is denied immediate use of the processor by the kernel scheduler in favor of a less important thread.
Another problem is the number of kernel threads. How many kernel threads should be created to support a particular process? If there are too few, then the available concurrency will not be realized--user threads that are ready to run will stand idle, even though there may also be idle processors. If there are too many, then the kernel may needlessly be multiplexing a number of kernel threads on a smaller number of processors, wasting time doing the context switching, even though the application has no need for such time slicing.
One might be tempted to give a process just as many kernel threads as there are processors. But if a user thread executes a blocking system call (such as reading from an I/O device) or suffers a page fault, then its underlying kernel thread also blocks--another user thread may be ready to execute, but no kernel thread is available to be assigned to it.

[planowanie] 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

[planowanie] Decyzje, które mają miejsce przy planowaniu wywłaszczeniowym, a których nie ma w planowaniu niewywłaszczeniowym.

Decyzje o przydzielaniu procesora zapadające w następujących sytuacjach:
- Proces przeszedł od stanu aktywności do stanu gotowości (np. wskutek wystąpienia przerwania)
- Proces przeszedł od stanu czekania, do stanu gotowości (np. po zakończeniu operacji we/wy)

Dotyczą planowania wywłaszczeniowego, nie ma ich w planowaniu niewywłaszczeniowym.

[planowanie] Opisać planowanie gwarantowane.

Planowanie gwarantowane (Guaranteed Scheduling)
- realne obietnice szybkości przetwarzania dla użytkowników:
    - jeśli jest zalogowanych n użytkowników, to każdy użytkownik otrzymuje 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 oznacza, że użytkownik wykorzystał tylko połowę przysługującego mu czasu
        - współczynnik 2.0 oznacza, że 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

[planowanie] Omów algorytm drugiej szansy

Algorytm drugiej szansy przypomina algorytm FIFO. Wszystkie strony znajdujące się w pamięci fizycznej tworzą kolejkę FIFO. Strony są sprowadzane do pamięci na żądanie, czyli na skutek odwołania do nich. Tak więc, gdy wstawiamy stronę na koniec kolejki FIFO, to jej bit odwołania jest ustawiany na 1. Gdy chcemy wybrać stronę-ofiarę, to kandydatem jest strona na początku kolejki FIFO. Możliwe są dwa przypadki:

- Jeśli bit odwołania tej strony jest równy 0, to faktycznie staje się ona ofiarą.
- Jeśli jej bit odwołania jest równy 1, to dostaje ona "drugą szansę":
    1. Jej bit odwołania jest ustawiany na 0.
    2. Strona pozostaje w pamięci.
    3. Jest ona przenoszona na koniec kolejki FIFO.
    4. Kandydatką na ofiarę jest następna strona w kolejce (ta strona również może dostać drugą szansę itd.).

[planowanie] Planowanie priorytetowe, jak sobie radzono z zagłodzeniem procesów.

Planowanie priorytetowe:

- każdemu procesowi przypisuje się priorytet
- procesor przydziela się temu procesowi, którego priorytet jest najwyższy
- w razie równych priorytetów - alg. FCFS

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.)

Planowanie priorytetowe może być wywłaszczające lub niewywłaszczające.

Problem nieskończonego blokowania (ang. indefinite blocking), głodzenia (ang. starvation) - procesy niskopriorytetowe czekają w nieskończoność na procesor

Rozwiązanie: postarzanie procesów niskopriorytetowych (ang. aging) - podwyższanie priorytetów długo oczekujących procesów.

[planowanie] Planowanie loteryjne

Planowanie loteryjne (ang. Lottery Scheduling)

- 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:

- planowanie loteryjne jest algorytmem bardzo czułym - jeżeli procesowi zwiększy się ilość losów, to jego szansa zwycięstwa już w następnym losowaniu zasobów wzrośnie
- 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.

[planowanie] Round robin

Planowanie rotacyjne (ang. round robin, RR)

- podobny do FCFS, dodano wywłaszczanie
- ustala się małą jednostkę czasu - kwant czasu (ang. time quantum) ; 10-100 ms
- kolejka procesów gotowych do wykonania - kolejka cykliczna
- każdy proces dostaje procesor na odcinek czasu, nie dłuższy od kwantu czasu, kiedy jest generowane przerwanie od zegara i proces jest usuwany na koniec kolejki procesów gotowych
- planista pobiera procesy w kolejności przyjścia FCFS
- średni czas oczekiwania: dość wysoki

[planowanie] Zalety Gang Scheduling.

- 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)

[planowanie] 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 niskopriorytetowej kolejce może zostać przeniesiony do kolejki o wyższym priorytecie.

Taki sposób postarzania procesów zapobiega ich głodzeniu.

[planowanie] 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

[planowanie] Omówić FIFO przydzielanie i jak eliminuje się problem zagłodzenia.

[planowanie] Wady i zalety jednej kolejki do wielu procesorów i osobnej dla każdego

Osobna kolejka dla każdego:

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

Wspólna kolejka dla wszystkich procesorów:

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

Wykład 3

[sekcja krytyczna] Warunki sekcji krytycznej.

Sekcja krytyczna:

- segment kodu w którym można aktualizować wspólne dane.
- tylko jeden proces jednocześnie może przebywać w swojej sekcji krytycznej
- n procesów w systemie { P0 P1 ...Pn-1 }
- każdy proces ma segment kodu zwany sekcją krytyczną (ang. *critical section*}
- jednocześnie tylko jeden proces może wykonywać swoją sekcję krytyczną -> wykonanie sekcji krytycznej podlega wzajemnego wykluczaniu (ang. mutual exclusion) w czasie
- konieczność zdefiniowania protokołu określającego współpracę między procesami

1. **Wzajemne wykluczanie**: Jeśli proces P, działa w swojej sekcji krytycznej, to żaden inny proces nie działa w sekcji krytycznej.
2. **Postęp**: Jeśli żaden proces nie działa w sekcji krytycznej oraz istnieją procesy, które chcą wejść do sekcji krytycznych, to tylko procesy niewykonujące swoich reszt mogą kandydować jako następne do wejścia do sekcji krytycznych i wybór ten nie może być odwlekany w nieskończoność.
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 sekcji krytycznej i zanim uzyskał na to pozwolenie.

[sekcja krytyczna] Algorytm sekcji krytycznej dla 2 procesów.

```
Procesy: P0  i P1
j=1-i

Proces i:
var znacznik: array[0..1] of boolean; {określą, że dany
      proces jest gotowy  do wejścia sekcji krytycznej}
numer: 0..1; {określa, który proces ma prawo do  wejścia
      do sekcji krytycznej}
znacznik[0]:= false;
znacznik[1]:= false;
numer:= dowolna;
repeat
          znacznik[i]:=true;
          numer:=j;
          while(znacznik[j] and numer=j) do nic;
          sekcja krytyczna
          znacznik[i]:=false;
       reszta
until false;
```

[sekcja krytyczna] rozwiązanie problemu sekcji krytycznej.

Warunki, jakie musi spełniać rozwiązanie problemu sekcji krytycznej:

1. Wzajemne wykluczanie:
   Jeżeli proces Pi działa w swojej sekcji krytycznej, to żaden inny proces nie działa w sekcji krytycznej.
2. Postęp:
   Jeżeli żaden proces nie działa w sekcji krytycznej i istnieją procesy, które chcą wejść do sekcji krytycznych, to tylko procesy niewykonujące swoich reszt mogą kandydować jako następne do wejścia do sekcji krytycznych i wybór ten nie może być odwlekany w nieskończoność.
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 sekcji krytycznej i zanim uzyskał na to pozwolenie.

[semafory] napisać kody semaforów aktywnie czekających (czekaj, sygnalizuj)

S - semafor

```
czekaj(S):
        while S ≤ 0 do nic;
        S:= S-1;

sygnalizuj(S):
        S:=S+1;
```

[semafory] Napisać algorytm bez aktywnego czekania - semafory.

Semafory: Rozwiązanie bez stosowania aktywnego czekania

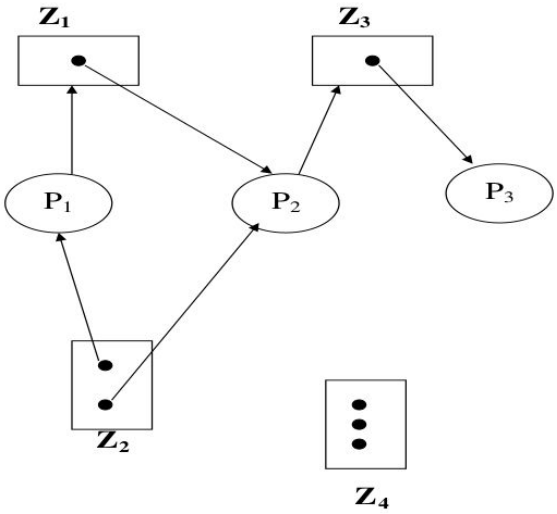
```
type semaphore = record
   wartość:integer;
   L: list of proces; {gdy proces musi czekać pod
                       semaforem, to jest dołączany do listy}
end

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

sygnalizuj(S): S.wartość:=S.wartość+1;
               if S.wartość ≤ 0
                      then begin
                        usuń jakiś proces P z S.L;
                        obudź(P);
                      end
```

Wykład 4

Graf przydziałów

Zakleszczenie można opisać za pomocą grafu skierowanego zwanego grafem przydziału zasobów systemu (ang. system resource-allocation graph):

- Zbiór wierzchołków jest podzielony na dwa rodzaje węzłów:
  - $P = \{P_1, P_2, ..., P_n\}$ - zbiór wszystkich **procesów systemu**, oznaczane ◯
  - $Z = \{Z_1, Z_2, ..., Z_m\}$ - zbiór wszystkich **typów zasobów systemowych**, oznaczane □, a każdy egzemplarz tego zasobu oznaczany kropką jest w prostokącie
- Są dwa rodzaje krawędzi:
  - **krawędź zamówienia** $P_i \rightarrow Z_j$ - proces P_i zamówił egzemplarz zasobu typu Z_j i obecnie czeka na ten zasób
  - **krawędź przydziału** $Z_j \rightarrow P_i$ - egzemplarz zasobu typu Z_j został przydzielony do procesu P_i



W oparciu o definicję grafu przydziału zasobów można wykazać, że:

- 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

<u>Co zrobić, by zapobiec czekaniu cyklicznemu na zasoby przez procesy?</u>

- zagwarantowanie, że czekanie cykliczne nigdy nie wystąpi przez wymuszenie całkowitego uporządkowania wszystkich typów zasobów i wymaganie, by każdy proces zamawiał je we wzrastającym porządku numeracji
- Z = {Z1, Z2 ,..., Zn} - zbiór typów zasobów
- F -> N
- każdemu typowi zasobów przyporządkowuje się w sposób jednoznaczny liczbę naturalną, która umożliwia określenie, który zasób poprzedza inny w uporządkowaniu

<u>Zapobieganie przetrzymywaniu i oczekiwaniu.</u>

- trzeba zagwarantować, że jeżeli proces zamawia zasób, to nie powinien on mieć żadnych innych zasobów
- proces zamawia i dostaje wszystkie potrzebne zasoby, zanim rozpocznie działanie

Wady powyższych 2 protokołów:

- wykorzystanie zasobów może być niskie, gdyż wielu przydzielonych zasobów żaden proces nie będzie potrzebował przez długi czas
- może dojść do głodzenia procesów - proces potrzebujący kilku popularnych zasobów może być odwlekany w nieskończoność, z powodu przydziału zasobów innym procesom

Omówić algorytm bankiera.

Może być stosowany w systemie przydziału zasobów, gdzie każdy typ zasobu ma wiele egzemplarzy (graf przydziału zasobów w tym przypadku nie nadaje się).

Pochodzenie nazwy algorytmu bankiera – mógłby służyć w systemie bankowym do zagwarantowania, że bank nie zainwestuje gotówki tak, że uniemożliwiłoby mu to zaspokojenie wymagań wszystkich jego klientów.

Idea algorytmu:

- 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

Wykład 5

[stronicowanie] Zalety stronicowania.

Zalety:

- brak problemu fragmentacji zewnętrznej,
- wspomaganie dla współdzielenia i ochrony pamięci.

Wady:

- narzut czasowy przy transformacji adresu,
- narzut pamięciowy (na potrzeby tablicy stron),
- fragmentacja wewnętrzna (niewielka).
- zwiększony czas na przełączenie kontekstu

[stronicowanie] Stronicowanie wielopoziomowe - dlaczego stosujemy?

Dlaczego:

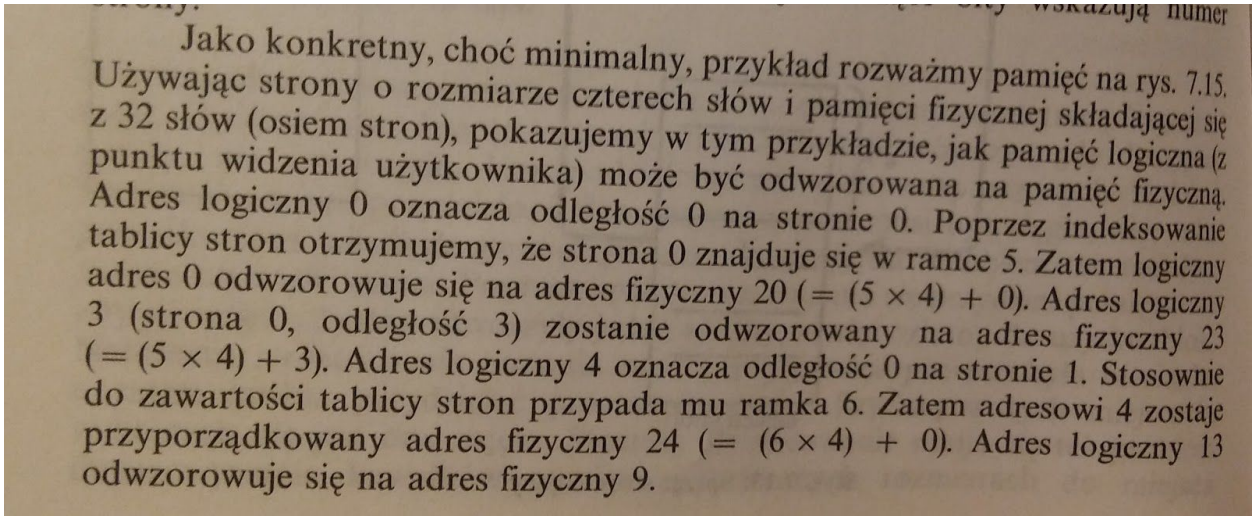
- tablica stron może być b. duża (~ milion wpisów)
- podział tablicy stron na mniejsze części, by uniknąć pełnego ciągłego obszaru w pamięci operacyjnej

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

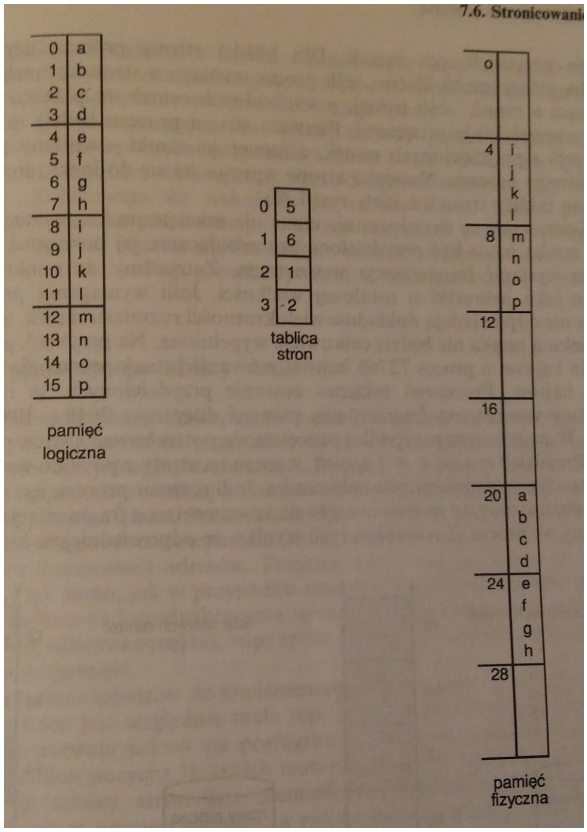
[stronicowanie] Wyliczenie adresu fizycznego (na podstawie logicznego) w stronicowaniu.

Adres fizyczny = wartość bazy + adres logiczny

Chyba
Tablica_stron[nr_strony]+offset = adr_fizyczny
// co to za książka?
//Silberschatz podstawy systemów operacyjnych 90% wykładów z niej jest



Jako konkretny, choć minimalny, przykład rozważmy pamięć na rys. 7.15. Używając strony o rozmiarze czterech słów i pamięci fizycznej składającej się z 32 słów (osiem stron), pokazujemy w tym przykładzie, jak pamięć logiczna (z punktu widzenia użytkownika) może być odwzorowana na pamięć fizyczną. Adres logiczny 0 oznacza odległość 0 na stronie 0. Poprzez indeksowanie tablicy stron otrzymujemy, że strona 0 znajduje się w ramce 5. Zatem logiczny adres 0 odwzorowuje się na adres fizyczny 20 (= (5 × 4) + 0). Adres logiczny 3 (strona 0, odległość 3) zostanie odwzorowany na adres fizyczny 23 (= (5 × 4) + 3). Adres logiczny 4 oznacza odległość 0 na stronie 1. Stosownie do zawartości tablicy stron przypada mu ramka 6. Zatem adresowi 4 zostaje przyporządkowany adres fizyczny 24 (= (6 × 4) + 0). Adres logiczny 13 odwzorowuje się na adres fizyczny 9.



-Zalety segmentowania.

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>

Zaleta segmentacji:

- 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

-Co to są bufory translacji adresów stron (ang. translation look-aside buffers) (TLB)
//propozycja odpowiedzi: krótka zwięzła i na temat ??

Problem: Obsługa wirtualnej pamięci stronicowanej wymaga wykonywania dużej liczby operacji. Szczególnie niekorzystne jest tutaj obciążanie czasu dostępu do danych i rozkazów czasem na dodatkowy dostęp do pamięci operacyjnej, związany z odczytem deskryptorów stron.

Rozwiązanie: Aby zredukować te dodatkowe nakłady czasu, w nowoczesnych mikroprocesorach wprowadzono odpowiednik pamięci podręcznej na deskryptory ostatnio wykorzystywanych stron. Jest to bardzo szybka pamięć nazywana **buforem translacji adresów** (ang. **translation look-aside buffer - TLB**). W czasie translacji adresów sprawdzana jest najpierw obecność deskryptora stron w buforze translacji adresów a dopiero przy negatywnym wyniku testu, podejmowane jest odczytanie i sprowadzenie do TLB deskryptora z pamięci operacyjnej.
Jednym zdaniem: Bardzo szybka pamięć znajdująca się w nowoczesnych mikroprocesorach w której przechowuje się deskryptory niedawno wykorzystywanych stron.

Wykład 6

-Optymalny algorytm wymiany stron.

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.

Wykład 7

[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 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)
- 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

[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

-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 ustawione 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 napotkane zamówienia.
- gdy osiągnie przeciwległy koniec: natychmiast wraca do początku dysku.

[planowanie dostępu do dysku] SSTF.

Strategia SSTF (ang. *shortest seek time first*) polega na tym, że w pierwszej kolejności obsługiwane jest to odwołanie do dysku, które jest najbliżej aktualnej pozycji głowicy. Strategia ta jest dobra, jeżeli dysk nie jest intensywnie używany przez wiele procesów. W przeciwnym przypadku jest ona podatna na zagłodzenie. Może się zdarzyć, że pewne odwołanie będzie oczekiwać na wykonanie, ale cały czas będą spływać inne odwołania, które będą bliżej aktualnej pozycji głowicy.

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

Strategia look stanowi modyfikację strategii scan. Głowice nie są przesuwane do skrajnych cylindrów jeżeli nie ma takiej potrzeby. Zawracają po obsłużeniu skrajnego odwołania. Strategia ta jest czasami nazywana "windową", gdyż ruchy głowic przypominają poruszanie się windy, która jeździ w górę i w dół, zabierając oczekujących pasażerów. Podobnie jak w przypadku strategii scan, średni czas oczekiwania na realizację odwołania do dysku zależy od miejsca na dysku. Wady tej jest pozbawiona strategia c-look, opisana poniżej.

Strategia c-look stanowi modyfikację strategii look. Podobnie, jak w przypadku c-scan, odwołania do dysku są realizowane tylko gdy głowice przesuwają się w jedną stronę po dysku. Po zrealizowaniu skrajnego odwołania głowice zawracają i przesuwają się do skrajnego odwołania po drugiej stronie dysku. Strategia ta jest trochę wolniejsza niż strategia look, ale za to średni czas oczekiwania na realizację odwołania jest taki sam dla wszystkich cylindrów