

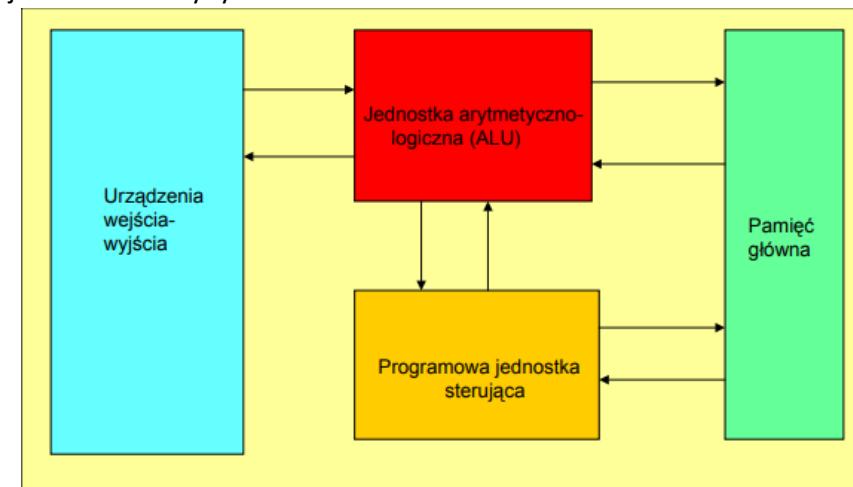
WYKŁAD 1

1. Podaj przykłady trzech różnych definicji systemu operacyjnego.
 - System operacyjny - program który kontroluje i nadzoruje dostęp programów do zasobów sprzętowych.
 - System operacyjny jest programem, który działa jako pośrednik między użytkownikiem komputera a sprzętem komputerowym
 - System operacyjny jest programem który ma za zadanie stworzenie środowiska do wykonywania innych programów.
2. Wymień rodzaje systemów operacyjnych.
 - Wsadowe
 - Wieloprogramowe
 - Z podziałem czasu
 - Rozproszone
 - Zgrupowane
 - Czasu rzeczywistego
 - Kieszonkowe
3. Omów idee wieloprogramowania i podziału czasu.
 - a. Systemy wieloprogramowania są systemami, które są w stanie przechowywać w pamięci operacyjnej wiele zadań a CPU w czasie bycia niewykorzystanym przez zadanie A jest w stanie prowadzić obliczenia na temat zadania B
4. Co to jest spooling i swapping?
 - a. Spooling – umożliwianie wykonywania zadań w czasie operacji IO innych zadań
 - b. Swapping – zadane zamiast być bezczynne w pamięci jest zapamiętywane na dysku w zbiorze wymiany a następnie inne zadanie zostaje z tego zbioru wczytane i wykonane.
 - c. Zapobiega to bezczynności procesora

WYKŁAD 2

1. Wymień składowe systemu komputerowego.
 - a. hardware
 - b. operating systems
 - c. software
 - d. end-users
2. Wymień funkcje ogólne komputera.
 - a. Przetwarzanie danych
 - b. Przechowywanie danych
 - c. Przenoszenie danych
 - d. Sterowanie funkcjami
3. Podaj strukturę komputera.
 - a. Jednostka centralna – sterowanie działaniem komputera
 - b. Pamięć główna – przechowywanie danych
 - c. Układ IO – przenoszenie danych
 - d. Połączenia systemu – łączność z procesorem sterującym funkcjami
4. Podaj strukturę procesora.
 - a. ALU jako centrum procesora łączy się z programową jednostką sterującą, pamięcią główną oraz urządzeniami IO

- b. Pamięć główna połączona jest z programową jednostką sterującą
 - c. Wszystkie obiekty połączone są dwustronnie
5. Podaj strukturę jednostki sterującej.
- Układy logiczne sterowania połączone z rejestrami i dekoderami, które przekazują dane do pamięci sterującej
6. Narysuj schemat maszyny von Neumanna.



- a.
7. Opisz architekturę maszyny IAS : pamięć, rejesty, format danych i rozkazów.
- Pamięć 1K słów 40b
 - Rejestry podstawowe (buforowe, adresowe, rozkazowe i bufory rozkazów)
 - Dane były podawane w ramkach 40b po 20b na rozkaz, z czego każda operacja składała się z kodu i adresu
8. Scharakteryzuj kolejne generacje systemów komputerowych, podaj przedziały czasowe oraz implementacje.
- Komputery lampowe (operacje zapisywane na taśmiech)
 - Komputery tranzystorowe (mniejsze i bardziej wydajne)
 - Komputery oparte na układach scalonych (redukcja masy, rozmiaru i poboru mocy)
 - Komputery oparte na układach IV generacji (dalejsza redukcja rozmiaru i wzrost mocy obliczeniowej)
9. Opisz skale integracji mikroukładów.
(ilość urządzeń w chipie)
- Mała – do 100
 - Średnia – do 3000
 - Wielka – do 100,000
 - Bardzo wielka – do 100,000,000
 - Ultrawielka – ponad 100,000,000
10. Podaj wzory na MIPS i MFLOPS.
- $MIPS = (T * 10^5)^{-1} * I_c = (CPI * 10^6)^{-1} * f$
 - $MFLOPS = (\text{czas wykonania} * 10^6)^{-1} * (\text{liczba wykonanych operacji})$
11. Podaj wzór Amdahla.

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + \frac{f}{N}}$$

a.

12. Oblicz przyspieszenie programu posiadającego x% kodu zrównoleglonego na y procesorach.

- $\text{Speedup} = \frac{x*0,01}{y}$

WYKŁAD 3

1. Wymień linie magistrali systemowej.



2. Wymień funkcje jednostki centralnej.

Funkcje jednostki centralnej

- ❖ CPU = procesor = **jednostka centralna**
- ❖ Pobieranie rozkazów z pamięci
- ❖ Interpretowanie rozkazów
- ❖ Pobieranie danych (z pamięci lub we/wy)
- ❖ Przechowywanie danych w pamięci
- ❖ Przetwarzanie danych - wykonywanie rozkazów
- ❖ Zapisywanie wyników (do pamięci lub na we/wy)

3. Opisz rejesty CPU oraz PSW.

Rejestry procesora

- ◆ licznik programu (PC) - adres rozkazu do pobrania
 - ◆ rejestr rozkazu (IR) - kod rozkazu
 - ◆ rejestr adresowy pamięci (MAR) - adres lokacji
 - ◆ rejestr buforowy pamięci (MBR) - dane do/z pamięci
-
- ◆ rejesty PSW (ang. program status word) - słowo stanu programu, informacje o stanie

4. Wymień kategorie urządzeń we/wy.

Kategorie wejścia/wyjścia

- ◆ Rodzaje urządzeń we/wy
 - ◆ przeznaczone do odczytu przez człowieka (np. monitor ekranowy, wydruk, dźwięk)
 - ◆ przeznaczone do odczytu przez maszynę (np. dyski magnetyczne, taśmy, czujniki w robotach)
 - ◆ komunikacyjne (np. modem, karta sieciowa)

5. Wymień i opisz sposoby realizacji we/wy.

Sposoby realizacji we/wy

- ◆ programowane - dane są wymieniane między procesorem a modułem we/wy, procesor czeka na zakończenie operacji we/wy
- ◆ sterowane przerwaniami - procesor wydaje operację we/wy i wykonuje dalsze rozkazy do momentu zakończenia operacji we/wy (przerwanie we/wy)
- ◆ bezpośredni dostęp do pamięci (ang. direct memory access - DMA) - moduł we/wy wymienia dane bezpośrednio z pamięcią bez udziału procesora

6. Wymień i opisz metody dostępu do pamięci.

Pamięć - metody dostępu (1)

- ◆ dostęp sekwencyjny (ang. sequential)
 - ◆ dostęp liniowy blok po bloku wprzód lub wtył
 - ◆ czas dostępu zależy od pozycji bloku względem pozycji bieżącej
 - ◆ np. taśmy
- ◆ dostęp bezpośredni (ang. direct)
 - ◆ każdy blok ma unikalny adres
 - ◆ czas dostępu realizowany przez skok do najbliższego otoczenia i sekwencyjne przeszukiwanie
 - ◆ np. dysk magnetyczny

Pamięć - metody dostępu (2)

- ◆ dostęp swobodny (ang. random)
 - ◆ każda adresowalna lokacja w pamięci ma unikatowy, fizycznie wbudowany mechanizm adresowania
 - ◆ czas dostępu nie zależy od poprzednich operacji i jest stały
 - ◆ np. RAM
- ◆ dostęp skojarzeniowy (ang. associative)
 - ◆ dane są lokalizowane raczej na podstawie porównania z ich zawartością niż na podstawie adresu
 - ◆ czas dostępu nie zależy od poprzednich operacji i jest stały
 - ◆ np. pamięć podręczna (ang. cache)

7. Wymień rodzaje pamięci ze względu na własności fizyczne.

Pamięć - własności fizyczne

- ❖ zanikanie, rozpad (ang. decay)
- ❖ ulotność (ang. volatility)
- ❖ wymazywalność (ang. erasable)
- ❖ zasilanie do utrzymania zawartości

8. Opisz zasadę lokalności odniesień.

Zasada lokalności odniesień

- ❖ zasada lokalności odniesień (ang. locality of reference) oznacza, że w czasie wykonania programu odwołania do danych i rozkazów mają tendencję do gromadzenia się
- ❖ przyczyna: programy zwykle zawierają tablice deklaracji zmiennych oraz stałych i wiele pętli iteracyjnych i podprogramów
 - ◆ lokalność przestrzenna – grupowanie odniesień do tych samych miejsc w pamięci
 - ◆ skłonność do sekwencyjnego sięgania po rozkazy lub dane (np. tablica)
 - ◆ lokalność czasowa - skłonność do odwołań do ostatnio wykorzystywanych miejsc w pamięci (np. pętla iteracyjna)
 - ◆ operacje na tablicach - dostęp do zgrupowanych słów
 - ◆ wykorzystanie zasady lokalności odniesień pozwala na zmniejszenie częstotliwości dostępu
 - ◆ przykład: pamięć podręczna (ang. cache, fr. cacher) procesora

9. Opisz działanie pamięci podręcznej.

Pamięć podręczna - działanie

- ❖ Cache zawiera fragment pamięci głównej
- ❖ Procesor sprawdza czy aktualnie potrzebne do wykonania rozkazu słowo z pamięci jest w cache'u
 - ◆ jeśli nie, to blok pamięci o ustalonej liczbie K słów zawierający potrzebne słowo jest ściągnięty do pamięci podręcznej
- ❖ Cache zawiera znaczniki identyfikujące bloki pamięci głównej
- ❖ Zrealizowana po raz pierwszy w 1969 na komputerze IBM S/360 model 85

10. Opisz sposoby mapowania dla pamięci podręcznej.

11. Założymy, że mamy komputer z pamięcią główną o x M adresowalnych bajtach oraz pamięć podręczną o pojemności y KB i blokach z B mapującą bezpośrednio, skojarzeniowo, k-drożnie sekcyjnie-skojarzeniowo pamięć główną:

- (a) jaka jest długość adresu tego komputera (w bitach)?
- (b) jaki jest bitowy podział adresu pamięci na znacznik(tag), sekcje, oraz słowo?
- (c) w którym wierszu pamięci podręcznej znajdą się następujące bajty (zapis adresu bajtu szesnastkowy), jaki jest ich tag i numer słowa: < ax1 >, < ax2 >, < ax3 >, < ax4 >, < ax5 >, < ax6 >, < ax7 >?
- (d) w którym wierszu pamięci podręcznej znajdą się następujące bajty (zapis adresu bajtu dziesiętnego), jaki jest ich tag i numer słowa: < ad1 >, < ad2 >, < ad3 >, < ad4 >, < ad5 >, < ad6 >, < ad7 >?

1. Wymień tryby adresowania w instrukcjach maszynowych.

Tryby adresowania

- ◆ natychmiastowy (ang. immediate)
- ◆ bezpośredni (ang. direct)
- ◆ pośredni (ang. indirect)
- ◆ rejestrowy (ang. register)
- ◆ rejestrowy pośredni (ang. register indirect)
- ◆ z przesunięciem (indeksowanie) (ang. displacement (indexed))
- ◆ stosowy (ang. stack)

2. Opisz adresowanie natychmiastowe.

Adresowanie natychmiastowe

- ◆ Argument (ang. operand) jest częścią rozkazu
- ◆ A – zawartość pola adresowego w rozkazie
- ◆ Operand = A
 - ◆ np. ADD 5
 - ◆ dodaj 5 do zawartości akumulatora
 - ◆ 5 jest argumentem
- ◆ Nie ma odniesienia do pamięci w celu pobrania argumentu
- ◆ Zaoszczędzony jeden cykl pamięci
- ◆ Wielkość operandu ograniczona przez rozmiar pola adresowego

Rozkaz

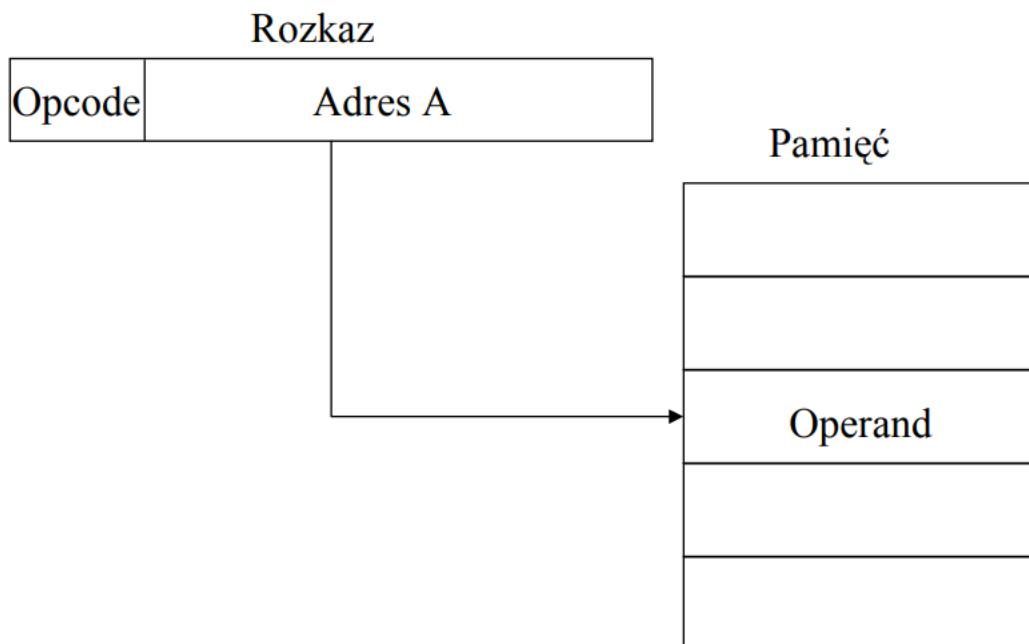
Opcode	Operand
--------	---------

3. Opisz adresowanie bezpośrednie.

Adresowanie bezpośredni

- ◆ Pole adresowe zawiera adres operandu
- ◆ EA - efektywny adres (ang. effective address) lokacji zawierający odniesiony argument
- ◆ $EA = A$
 - ◆ np. ADD A
 - ◆ dodaj zawartość komórki A do akumulatora
 - ◆ pod adresem A znajduje się operand
- ◆ Jedno odniesienie do pamięci
- ◆ Nie są potrzebne dodatkowe obliczenia
- ◆ Zakres adresacji ograniczony przez wielkość pola adresowego (słowo - opcode)

Adresowanie bezpośredni (c.d.)

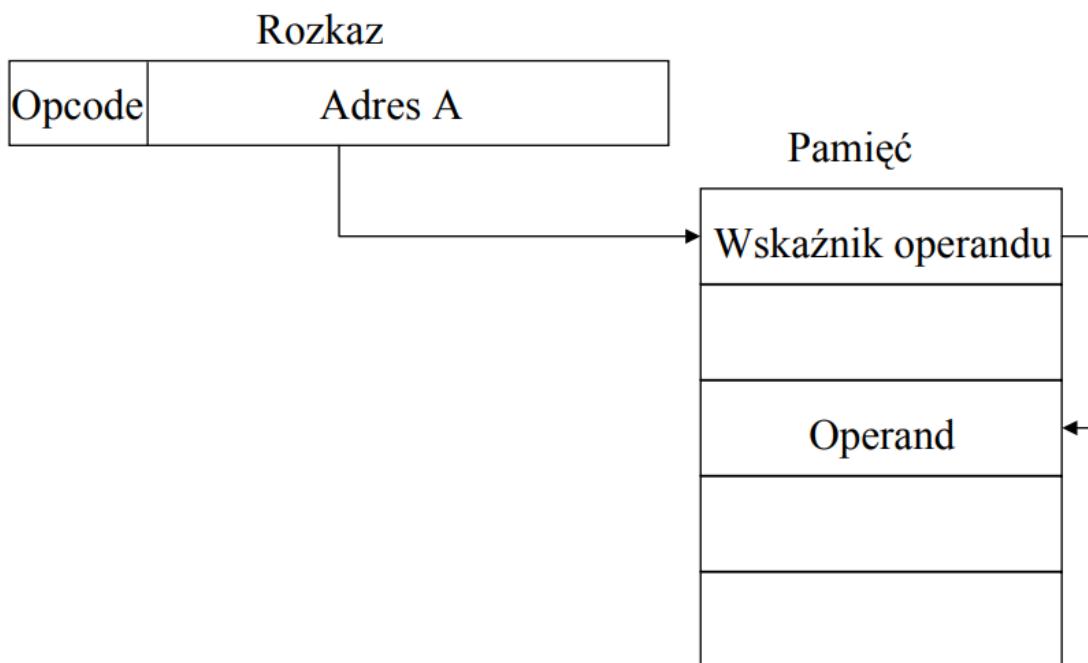


4. Opisz adresowanie pośrednie.

Adresowanie pośrednie

- ◆ Pole adresowe odnosi się do słowa w pamięci, które zawiera pełnej długości adres argumentu
- ◆ (X) – zawartość lokacji X (rejestr lub adres pamięci)
- ◆ EA = (A)
 - ◆ znajdź A, znajdź adres (A) pod którym jest operand
- ◆ np. ADD (A)
 - ◆ dodaj zawartość komórki pamięci wyznaczonej przez zawartość pod adresem A do akumulatora
- ◆ Większa przestrzeń adresowa: 2^n gdzie n = długość słowa
- ◆ Może być zagnieżdżone (ang. nested, multilevel, cascaded)
 - ◆ np. EA = (((A)))
 - ◆ [Zad.](#) Narysuj diagram
- ◆ Wiele odniesień do pamięci głównej w celu pobrania argumentu (dlatego wolne)

Adresowanie pośrednie (c.d.)

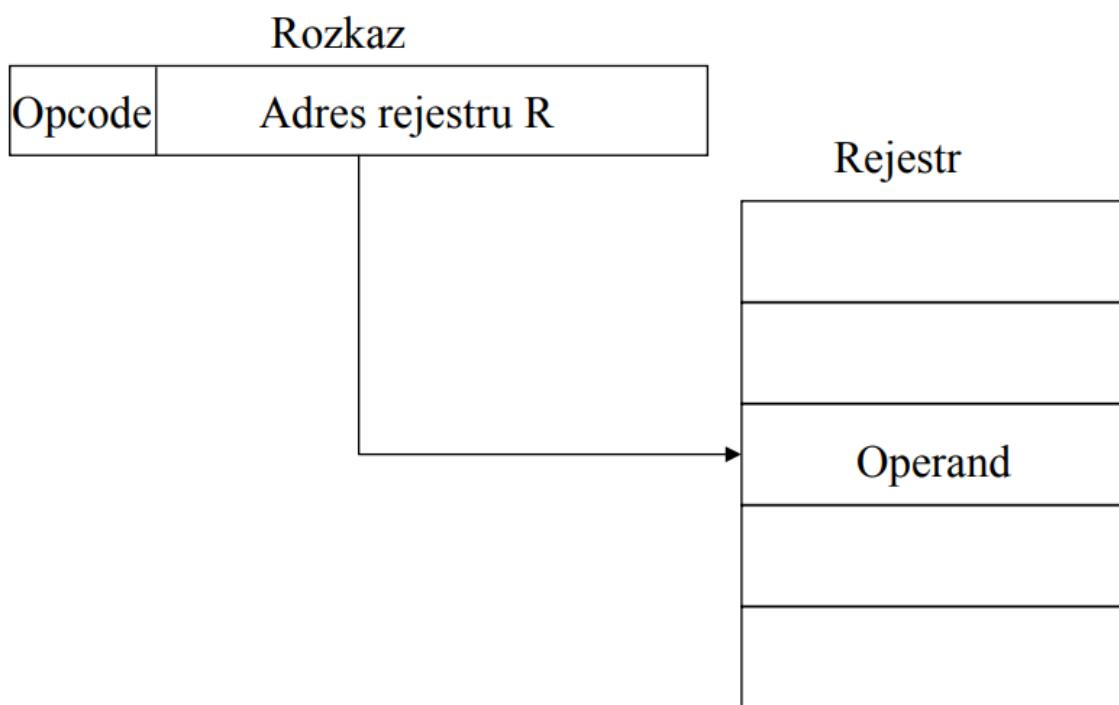


5. Opisz adresowanie rejestrowe.

Adresowanie rejestrowe

- ◆ Operand znajduje się w rejestrze określonym w polu adresowym
- ◆ EA = R
- ◆ Ograniczona liczba rejestrów (32)
- ◆ Małe pole adresowe (zwykle 3 do 5 bitów)
 - ◆ krótsze rozkazy i czas pobrania z rejestru
- ◆ Ograniczona przestrzeń adresowa

Adresowanie rejestrowe (c.d.)

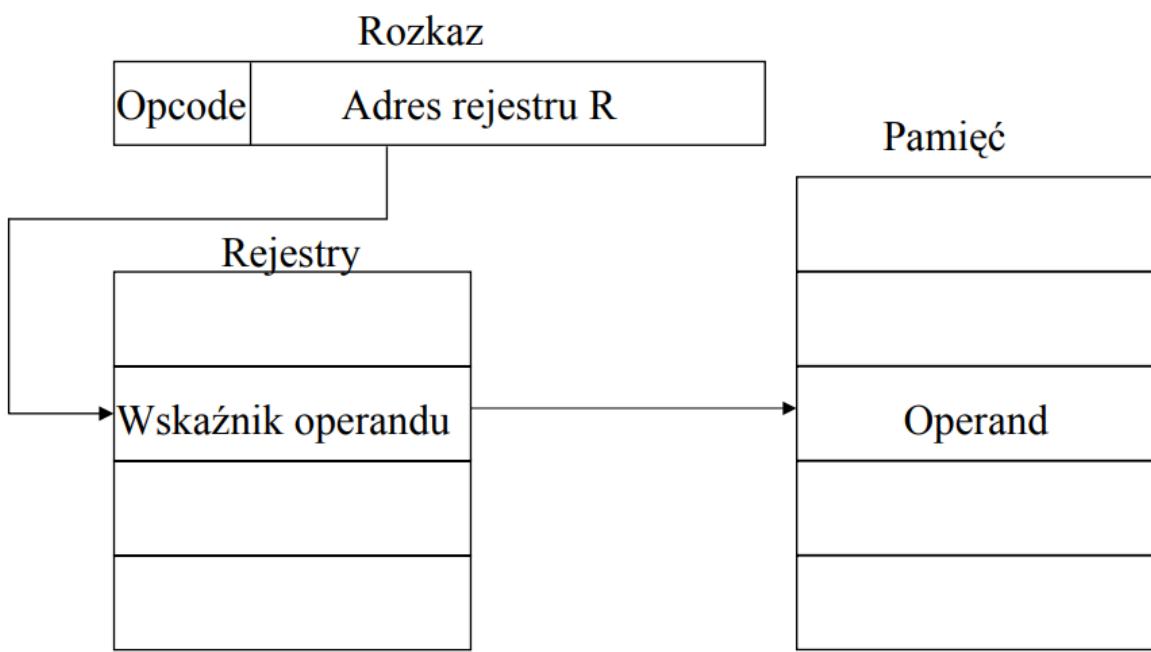


6. Opisz adresowanie pośrednie rejestrowe.

Pośrednie adresowanie rejestrowe

- ❖ Adres w rejestrze odnosi się do słowa w pamięci, które zawiera adres operandu
- ❖ EA = (R)
- ❖ Operand jest w komórce pamięci wskazanej przez adres zawarty w rejestrze R
- ❖ Duża przestrzeń adresowa (2^n)
- ❖ O jedno odniesienie do pamięci mniej niż przy adresowaniu pośrednim

Pośrednie adresowanie rejestrowe (c.d.)



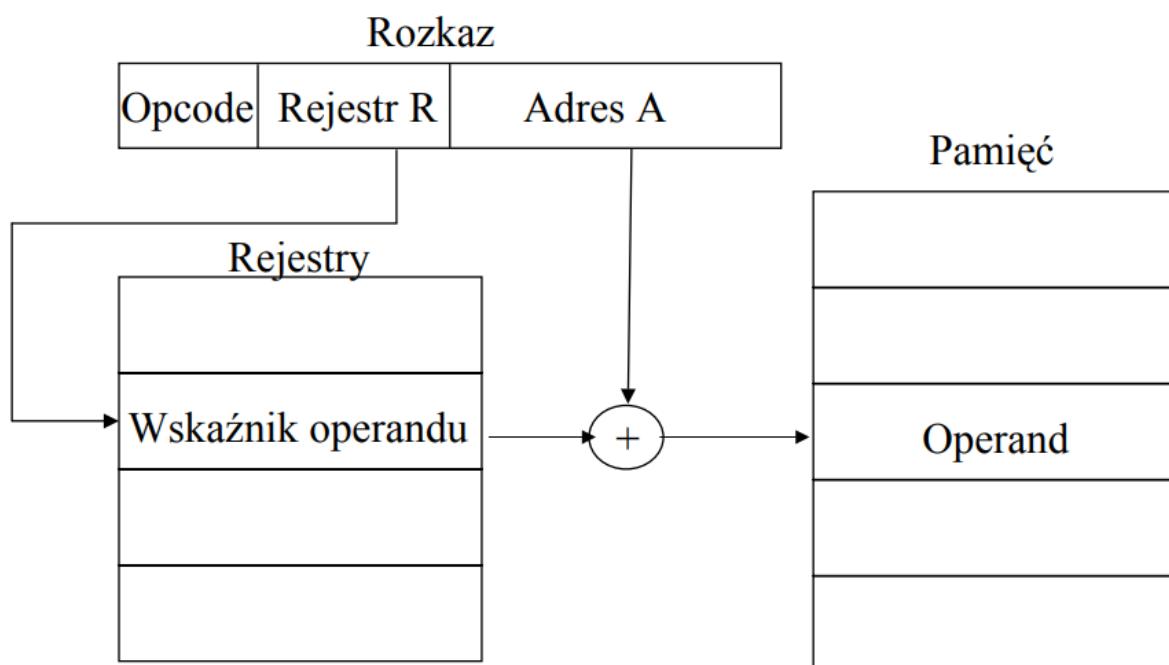
7. Wymień rodzaje adresowania z przesunięciem. 8. Opisz adresowanie względne.

9. Opisz adresowanie z rejestrów podstawowym. 10. Opisz adresowanie indeksowane.

Adresowanie z przesunięciem

- ◆ $EA = A + (R)$
- ◆ Pole adresowe zawiera dwie wartości
 - ◆ A = wartość bazowa
 - ◆ R = rejestr zawierający przesunięcie
 - ◆ lub odwrotnie
- ◆ Najczęstsze zastosowania
 - ◆ adresowanie względne (ang. relative addressing)
 - ◆ adresowanie z rejestrów podstawowym (ang. base-register addressing)
 - ◆ indeksowanie (ang. indexed addressing)

Adresowanie z przesunięciem (c.d.)



Adresowanie z przesunięciem (c.d.)

◆ **adresowanie względne**

- ◆ R = licznik programu, PC
- ◆ EA = A + (PC)
- ◆ zgodność z zasadą lokalności odniesień (np. w pamięci podręcznej)

◆ **adresowanie z rejestrów podstawowym**

- ◆ A zawiera przesunięcie a R adres bazowy
- ◆ R może zawierać adres bezpośredni lub pośredni
- ◆ np. rejesty segmentowe w 80x86
- ◆ zgodność z zasadą lokalności odniesień

Adresowanie z przesunięciem (c.d.)

◆ indeksowanie

- ◆ A = adres bazowy, R = przesuniecie
- ◆ EA = A + R
- ◆ adresowanie dobre dla tablic: EA = A + R; R++
- ◆ autoindeksowanie: EA = A + (R); (R) <- (R) + 1

◆ kombinacje

- ◆ indeksowanie wtórne (ang. postindex): EA = (A) + (R)
- ◆ indeksowanie wstępne (ang. preindex): EA = (A + (R))

11. Opisz adresowanie stosowe.

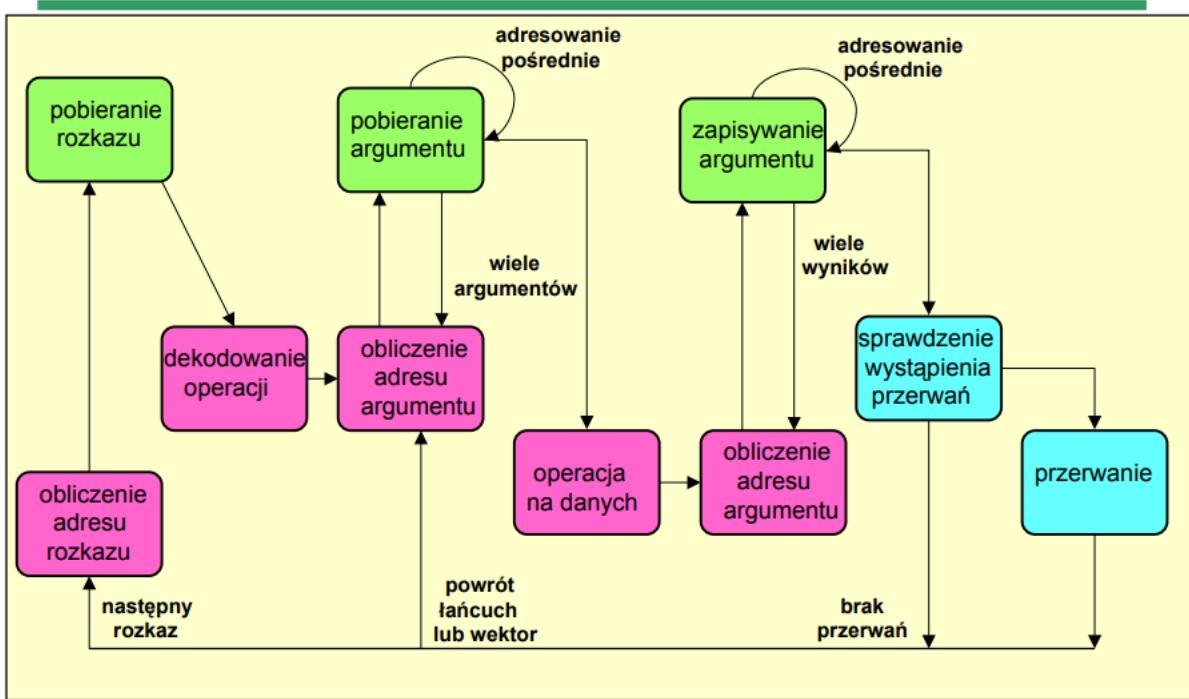
Adresowanie stosowe

◆ Operand znajduje się na wierzchołku stosu wskazanym przez rejestr

- ◆ np. ADD
- ◆ Usuń (POP) dwa wierzchołkowe elementy stosu i dodaj

12. Narysuj graf stanów cyklu rozkazu z przerwaniami.

Graf stanów cyklu rozkazu z przerwaniami



13. Opisz potokowe przetwarzanie rozkazów.

Potokowe przetwarzanie rozkazów

- ◆ Czas wykonywania rozkazu jest zwykle większy od czasu pobierania a więc musi być zwłoka wynikająca z zajętości bufora
- ◆ Rozkaz skoku warunkowego powoduje, że adres następnego rozkazu jest nieprzywidzialny
- ◆ Gdy rozkaz skoku warunkowego przechodzi z etapu pobierania do wykonywania możemy pobrać również rozkaz następny
 - ◆ jeśli jednak skok nastąpi to musi on zostać usunięty
- ◆ Aby uzyskać większe przyspieszenie potok musi być przetwarzany w większej liczbie etapów

14. Opisz sprzętową ochronę adresów.

Sprzętowa ochrona adresów

- ❖ sprzęt jednostki centralnej porównuje każdy adres wygenerowany w trybie użytkownika z zawartością rejestrów bazowego i granicznego
- ❖ wartości rejestru bazowego i granicznego mogą być załadowane jedynie w trybie monitora (load jest instrukcją uprzywilejowaną)
- ❖ przerwanie protekcja pamięci

15. Opisz zegarową ochronę CPU.

Ochrona CPU

- ❖ Zegar jest ustawiany przez system operacyjny przed przekazaniem sterowania do programu użytkownika
 - ❖ w trakcie wykonywania programu zegar jest zmniejszany
 - ❖ jeśli zegar osiągnie 0 generowane jest przerwanie
- ❖ ładuj zegar jest rozkazem uprzywilejowanym
- ❖ zegar może być wykorzystany do realizacji podziału czasu (ang. time sharing) - przerwanie zegarowe następuje po wykorzystaniu kwantu czasu przez proces

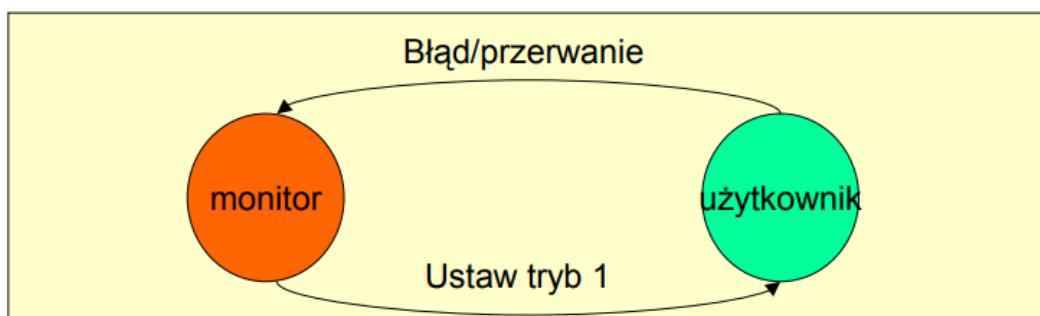
16. Opisz dualny tryb operacji.

Dualny tryb operacji

- ◆ System wielozadaniowy musi chronić system operacyjny oraz wykonywane programy przed każdym niewłaściwie działającym programem
- ◆ Należy wyposażyć sprzęt w środki pozwalające na rozróżnienie przynajmniej dwóch trybów pracy:
 - ◆ tryb użytkownika (ang. user mode) - wykonanie na odpowiedzialność użytkownika
 - ◆ tryb monitora (ang. monitor mode) = tryb nadzorcy (ang. supervisor mode) = tryb systemu (ang. system mode) = tryb uprzywilejowany (ang. privileged mode) - wykonanie na odpowiedzialność systemu operacyjnego

Dualny tryb operacji (c.d.)

- ◆ Bit trybu (ang. mode bit) w sprzęcie komputerowym wskazujący na bieżący tryb pracy : system (0), użytkownik (1)
- ◆ Wystąpienie błędu: przejście w tryb 0



- ◆ Rozkazy uprzywilejowane (ang. privileged) - tryb systemu

WYKŁAD 5

1. Wymień składowe systemu operacyjnego.

Składowe systemu

- ◆ Zarządzanie procesami (ang. process management)
- ◆ Zarządzanie pamięcią operacyjną (ang. main memory management)
- ◆ Zarządzanie plikami (ang. file management)
- ◆ Zarządzanie systemem we/wy (ang. I/O system management)
- ◆ Zarządzanie pamięcią pomocniczą (ang. secondary-storage management)
- ◆ Praca sieciowa (ang. networking)
- ◆ System ochrony (ang. protection system)
- ◆ System interpretacji poleceń (ang. command-interpreter system)

2. Wymień usługi systemu operacyjnego.

Usługi systemu operacyjnego

- ◆ Wykonanie programu - system powinien móc załadować program do pamięci i wykonać go
- ◆ Operacje we/wy - program użytkowy nie wykonuje bezpośrednio operacji we/wy więc taką usługę musi oferować system
- ◆ Manipulowanie systemem plików - program musi mieć możliwość (pod kontrolą) do czytania, pisania, tworzenia i usuwania plików
- ◆ Komunikacja - wymiana informacji pomiędzy procesami wykonywanymi na tym samym lub zdalnym komputerze
 - ◆ np. za pomocą pamięci dzielonej (ang. shared memory) lub przekazywania komunikatów (ang. message passing)
- ◆ Wykrywanie błędów - zapewnienie prawidłowości działania komputera poprzez wykrywanie i obsługę wszystkich błędów w jednostce centralnej, pamięci operacyjnej, urządzeniach we/wy (np. błąd sumy kontrolnej) i w programie użytkownika (np. przekroczenie czasu)

3. Co to są wywołania (funkcje) systemowe?

Wywołania systemowe

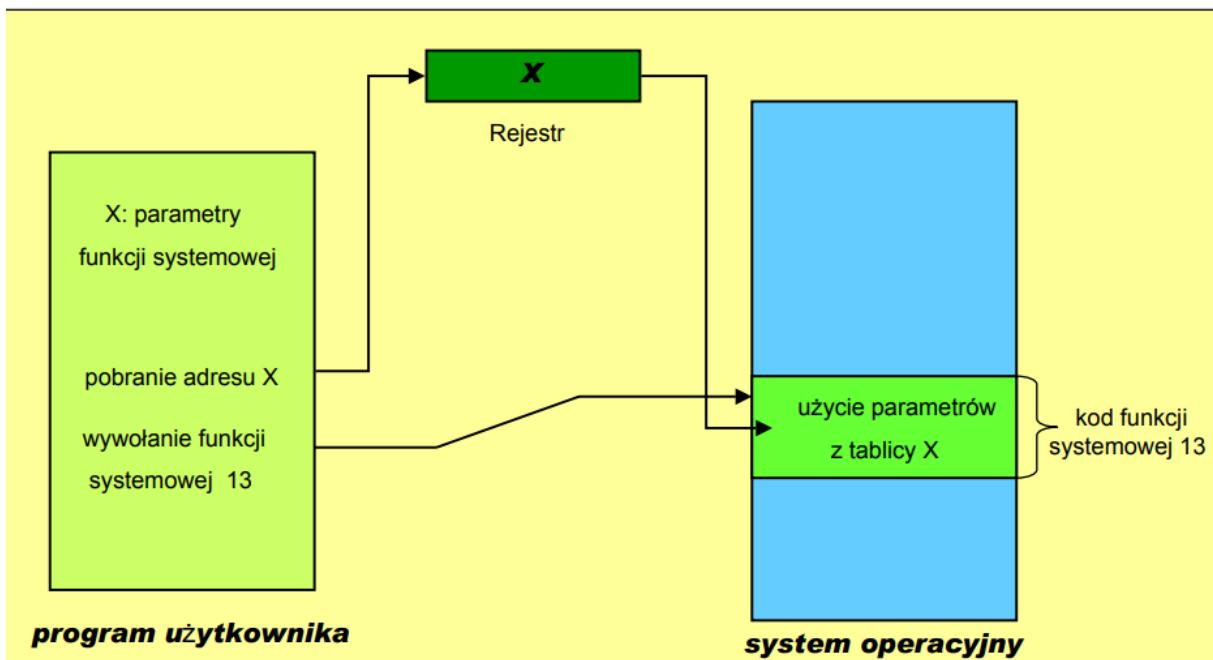
- ❖ Wywołania (funkcje) systemowe (ang. system calls) tworzą interfejs między wykonywanym programem a systemem operacyjnym
 - ◆ dostępne na poziomie języka maszynowego (asemblera) - IBM/370: rozkaz SIO (start input/output)
 - ◆ pewne języki zastępują asembler w programowaniu systemowym i umożliwiają bezpośrednie wykonywanie funkcji systemowych (np. C, C++, Bliss, PL/360, PERL)
 - ◆ Win32 API (ang. Application Programmer Interface) - wielki zbiór procedur dostarczanych przez Microsoft, które umożliwiają realizację funkcji systemowych
 - ◆ np. *CreateProcess()* wywołuje funkcję jądra *NTCreateProcess()*
 - ◆ API interfejsu wywołań systemowych w Linuxie bazuje na standardzie POSIX – biblioteka C (glibc)
- ❖ Proste skopiowanie pliku jest w rzeczywistości skomplikowanym przedsięwzięciem

Wywołania systemowe (c.d.)

- ❖ Są trzy metody przekazywania parametrów między wykonywanym programem a systemem operacyjnym
 - ◆ umieszczenie parametrów w rejestrach jednostki centralnej
 - ◆ procesory x86: rejesty EBX, ECX, EDX, ESI, EDI
 - ◆ wywołanie systemowe zwraca wartość w EAX
 - ◆ zapamiętanie parametrów w tablicy w pamięci operacyjnej i przekazanie adresu tej tablicy jako parametru w rejestrze
 - ◆ składowanie (ang. push) przez program parametrów na stosie (ang. stack) i zdejmowanie (ang. pop) ze stosu przez system operacyjny

4. Wymień podstawowe metody przekazywania parametrów między procesem a systemem.

Przekazywanie parametrów za pomocą tablicy



5. Wymień rodzaje wywołań (funkcji) systemowych.

Wywołania systemowe - podział

- ◆ nadzorowanie procesów
- ◆ zarządzanie plikami
- ◆ zarządzanie urządzeniami
- ◆ utrzymywanie informacji
- ◆ komunikacja

6. Wymień struktury systemów operacyjnych oraz przykłady ich realizacji.

Struktura warstwowa

- ❖ System operacyjny jest podzielony na warstwy (poziomy) (ang. layers (levels)) zbudowane powyżej warstw niższych
 - ❖ warstwę najniższą (warstwę 0) stanowi sprzęt; warstwę najwyższą (warstwę N) stanowi interfejs z użytkownikiem
- ❖ podejście warstwowe realizuje modularność (ang. modularity): warstwy są wybrane w ten sposób, że każda korzysta z funkcji (operacji) i usług tylko niżej położonych warstw

Modularność

- ❖ Większość współczesnych systemów operacyjnych ma zaimplementowane w jądrze systemu moduły (np. Linux)
 - ❖ podejście zorientowane obiektowo
 - ❖ moduły komunikują się za pomocą interfejsów
 - ❖ moduł jest ładowany dynamicznie do jądra tylko wtedy gdy jest potrzebny
 - ❖ np. obsługa karty sieciowej
- ❖ Pewne podobieństwo do struktury warstwowej jednak większa elastyczność

7. Wyjaśnij zasadę maszyny wirtualnej.

Maszyny wirtualne

- ◆ Maszyna wirtualna (ang. virtual machine) jest logiczną konsekwencją podejścia warstwowego - jądro systemu jest traktowane jako sprzęt
 - ❖ IBM VM/370, JavaOS (JVM), Tryb MS-DOS, Cygwin, VMware (<http://www.vmware.com/download/>), Virtual PC 2004, Bochs (<http://bochs.sf.net/>), UML (<http://user-mode-linux.sf.net/>), XEN (<http://www.cl.cam.ac.uk/Research/SRG/netos/xen>) , PC2007 (<http://www.microsoft.com/downloads>) All Downloads->Virtual PC, VirtualBox (<http://www.virtualbox.org/>), .NET, KVM
- ◆ Maszyna wirtualna dostarcza identycznego interfejsu dla sprzętu
- ◆ System operacyjny gospodarza (ang. host OS) równolegle obsługuje systemy operacyjne gości (ang.guest OS)
- ◆ System operacyjny tworzy wirtualne systemy komputerowe, każdy proces ma do dyspozycji własne (wirtualne) jądro, właski pamięć drukarki
- ◆ Zasoby fizycznego komputera są dzielone w celu utworzenia maszyn wirtualnych
 - ❖ planowanie przydziału procesora jest tak wykorzystane, że użytkownik ma wrażenie jakoby miał do dyspozycji własny procesor
 - ❖ spooling i system zarządzania plikami jest wykorzystany tak, że powstaje wrażenie użytkowania drukarki, czytnika na wyłączność
 - ❖ zwykłe terminale użytkownika funkcjonują jak konsole operatorskie maszyny wirtualnej (np. system interakcyjny CMS)

8. Co to jest mikrojądro i jakie są jego wady i zalety?

Mikrojądro

- ◆ jądro zredukowane do małego zbioru funkcji rdzeniowych tzw. mikrojądro
- ◆ większość operacji w przestrzeni użytkownika
 - ◆ obsługa: urządzeń, plików, pamięci wirtualnej, grafiki i ochrona przy pomocy komunikatów (np. IPC)
- ◆ Przykłady
 - ◆ Mach - opracowany w Carnegie-Mellon University (połowa lat 80-tych)
 - ◆ <http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html>
 - ◆ Tru64 UNIX - Digital UNIX
 - ◆ <http://www.tru64.org/>
 - ◆ Apple MacOS X (Darwin) Server
 - ◆ <http://www.apple.com/macosx/>
 - ◆ QNX - przykład systemu czasu rzeczywistego
 - ◆ <http://www.qnx.com/>
 - ◆ Windows NT 4.0 - rozwiązań hybrydowe: warstwowe podejście do modelu klient-serwer

Zalety i wady mikrojáder

- ◆ Jednolity interfejs dla procesów
 - ◆ Łatwość w kontrolowaniu kodu systemu operacyjnego
 - ◆ niezawodność - małe jądro łatwiej przetestować (ok. 300 kB kodu, 140 funkcji systemowych)
 - ◆ rozszerzalność – dodanie nowego mechanizmu wymaga jedynie modyfikacji serwera
 - ◆ Przenośność systemu (na inną architekturę)
 - ◆ Obsługa systemów rozproszonych
 - ◆ Obsługa systemów zorientowanych obiektywo
-
- ◆ Wydajność - zbudowanie, zakolejkowanie, wysłanie, odebranie, potwierdzenie komunikatu
 - ◆ Poprawianie wydajności prowadzi do rozbudowy mikrojádra

9. Omów trzy przykłady realizacji maszyny wirtualnej.

KVM

- ◆ Rozwiązania wirtualizacyjne
 - ◆ sprzętowe – procesory z obsługą wirtualizacji (HVM) – x_86, x86_64, AMD-V
 - ◆ egrep '(vmx|svm)' /proc/cpuinfo
 - ◆ programowe
 - ◆ hypervisory (VMM, virtual machine monitor) - implementują składowe systemu operacyjnego (np. VMWare, Xen, KVM)
- ◆ Architektura wirtualizacji
 - ◆ pełna wirtualizacja – nie wymaga modyfikowania jądra systemu (VMWare)
 - ◆ parawirtualizacja – systemy wykorzystują zmodyfikowane jądro
 - ◆ Xen, UML, Linux-Vserver
 - ◆ emulacja – instrukcja w systemie gościa wykonuje się w trybie emulacji
 - ◆ Qemu, Hercules (np. IBM MVT/TSO)
- ◆ KVM – wykorzystuje wirtualizację sprzętową
 - ◆ środowisko gościa składa się z Qemu i systemu gościa
 - ◆ pamięć gościa jest mapowania na przestrzeń pamięci procesu
 - ◆ procesy w maszynie wirtualnej są wątkami w systemie gospodarza
 - ◆ moduł kvm obsługuje zwirtualizowany dostęp do rejestrów CPU

10. Jaka jest różnica pomiędzy hyperwizorem typu 0, 1 i 2?

Typ 0 – wykorzystanie minimalnych zasobów software-owych w celu wirtualizacji gościa na hardware gospodarza

Typ 1 – bare-metal – uruchamiany bezpośrednio na fizycznej warstwie systemu wirtualizacji nie wymaga ładowania systemu operacyjnego który nim zarządza, dzięki czemu są najbardziej efektywne oraz w wyniku braku komponentów „pod” nimi są bezpieczniejsze

Typ 2- instalowane na systemie operacyjnym gospodarza. Są najmniej wydajne ponieważ zasoby które pobierają są im nadawane przez system operacyjny gospodarza i przechodzą przez specjalne oprogramowanie do wirtualizacji

11. Omów rodzaje nieautoryzowanego wyjawienia.

Nieautoryzowane wyjawienie

- ◆ ujawnienie (ang. exposure) – np. przypadkowe opublikowanie w sieci informacji poufnych
- ◆ przechwycenie (ang. interception) – np. pakietów w bezprzewodowej sieci LAN
- ◆ wnioskowanie (ang. inference) – np. na podstawie szczegółowych danych
- ◆ przejęcie (ang. intrusion)- np. dostępu do wrażliwych danych poprzez złamanie zabezpieczeń systemu

12. Omów rodzaje podstępów komputerowych.

Podstęp (ang. deception)

- ◆ maskarada (ang. masquerade)
 - ◆ zalogowanie się na czyjeś konto
 - ◆ uzyskanie nieautoryzowanego dostępu - koń trojański (ang. Trojan horse)
- ◆ sfałszowanie (ang. falsification)
 - ◆ wprowadzenie fałszywych danych do pliku lub bazy danych
- ◆ odzegnanie (ang. repudiation)
 - ◆ zaprzeczanie wysyłania lub posiadaniu danych

13. Omów rodzaje przerwania działania systemu komputerowego.

Przerwanie działania

- ◆ niezdolność do pracy (ang. incapacitation) – atak na dostępność
 - ◆ zniszczenie sprzętu
 - ◆ złośliwe oprogramowanie (malware - ang. malicious software)
 - ◆ trojany, wirusy (ang. viruses), robaki (ang. worms)
- ◆ zmiana działania (ang. corruption) – atak na integralność systemu
 - ◆ złośliwe oprogramowanie powoduje niepożądane działanie systemu
 - ◆ modyfikacja oprogramowania poprzez wstawienie tylnej furtki (ang. backdoor)
- ◆ utrudnianie pracy (ang. obstruction)
 - ◆ obciążanie systemu (odmowa usługi, DOS – ang. denial of service)
 - ◆ zmiana działania łączy komunikacyjnych

14. Omów rodzaje usurpacji komputerowej.

Uzurpacja

- ◆ niewłaściwe przywłaszczenie (ang. missappropriation)
 - ◆ kradzież serwisu (np. CPU)
 - ◆ DDoS (ang. distributed DOS) – kradzież CPU i zasobów systemu operacyjnego porzez zmasowany atak z sieci
- ◆ niewłaściwe użycie (ang. misuse)
 - ◆ uzyskanie dostępu do systemu (np. przez hakera (ang. hacker))

15. Omów kategorie intruzów komputerowych.

Intruzi (ang. intruders)

◆ **podsywający**

- ❖ masquerader (pol. przebieraniec) – ten kto penetruje system kontroli dostępu użytkowników w celu ich wykorzystania

◆ **nadużywający**

- ❖ misfeasor (pol. nadużywający władzy) – użytkownik wykorzystujący dostęp do danych do których nie ma uprawnień

◆ **skrywający**

- ❖ clandestine (pol. tajny) user – ten kto przejmuje kontrolowanie (ang. audit) systemu aby uniknąć weryfikacji

16. Podaj klasyfikację wirusów komputerowych.

Klasyfikacja wirusów

◆ **cel infekcji (ang. target)**

- ❖ wirus w sektorze startowym (ang. boot sector infector)
- ❖ wirus w pliku systemowym (ang. file infector)
- ❖ makrowirus (ang. macro virus) – infekuje dokumenty, np. w MS Word

◆ **strategia ukrycia (ang. concealment strategy)**

- ❖ wirus zakodowany (ang. encrypted) – każdorazowe generowanie klucza i dekodowanie wirusa, gdy wirus się replikuje koduje się za pomocą nowego klucza
- ❖ wirus utajniony (ang. stealth) – przed antywirusem np. skompresowany wirus
- ❖ wirus polimorficzny (ang. polymorphic) – mutujący się (zmieniający swój wzorzec bitowy)
- ❖ wirus metamorficzny (ang. metamorphic) – mutujący, zmieniający swoje działanie i wygląd

Boty

- ◆ ang. bot (robot), zombie, drone
- ◆ potajemne przejęcie kontroli nad innymi hostami (setki, tysiące) podłączonymi do Internetu w celu utworzenia sieci botów (ang. botnet)
- ◆ Przypuszczenie ataku poprzez mechanizm RCF (ang. remote control facility)
 - ◆ trudno jest go przypisać konkretnemu twórcy bota
- ◆ Wykorzystanie botnetu
 - ◆ DDoS (distributed-denial-of-service) – atak na serwisy sieciowe
 - ◆ spamming – wysłanie e-maili
 - ◆ sniffing – podsłuchiwanie ruchu sieciowego
 - ◆ keylogging – podsłuchiwanie klawiatury
 - ◆ rozpowszechnianie złośliwego oprogramowania
 - ◆ instalowanie BHO (browser helper object) – dodatkowe reklamy
 - ◆ ataki na sieci IRC, gry sieciowe, itp.

Rootkity

- ◆ ang. rootkits
- ◆ Zbiór programów zainstalowanych w systemie w celu uzyskania uprawnień administratora (root) oraz ukrycie swojego istnienia
 - ◆ trwałe – aktywujące się w czasie ładowania systemu
 - ◆ nietrwałe - w pamięci RAM
 - ◆ w trybie jądra (np. funkcja systemowa) lub użytkownika (API programu użytkownika)
- ◆ Instalacja: wykorzystanie luki w systemie (np. otwarcie portu), uzyskanie uprawnień roota (cracking, malware, luka w systemie), upload rootkita, ukrycie rootkita podczas instalacji, otwarcie portu dla dalszych działań
- ◆ Tablica wywołań systemowych
 - ◆ modyfikacja tablicy wywołań systemowych
 - ◆ modyfikacja wywołań systemowych
 - ◆ przekierowanie tablicy wywołań systemowych

WYKŁAD 6

1. Podaj przykłady uruchamiania procesów w systemie Unix.

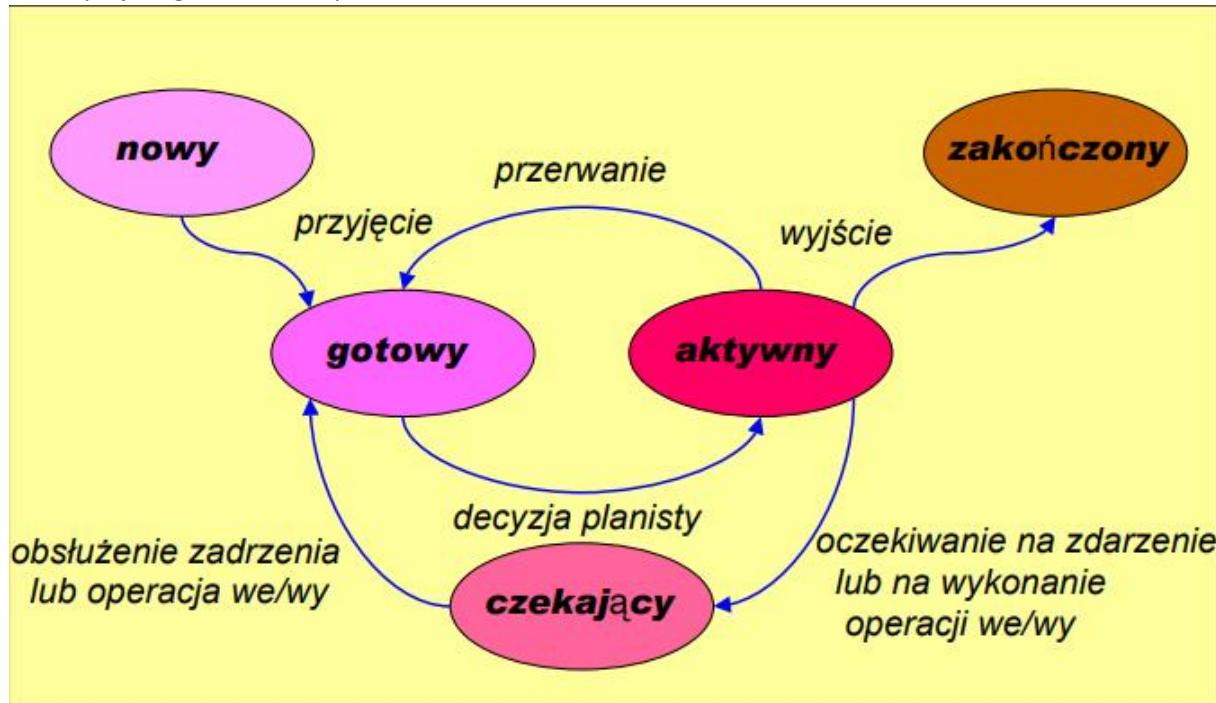
Przykłady procesów (Unix)

- ◆ Proces to program działający we własnej przestrzeni adresowej
 - ◆ proces wsadowy – polecenie *batch, at, cron*
 - ◆ procesy interakcyjne
 - ◆ procesy pierwszoplanowe – związane z terminaliem (np. polecenie *ls*)
 - ◆ procesy drugoplanowe – wykonywane w tle
 - ◆ demony (ang. daemons) – wystartowane serwisy
 - ◆ init, syslogd, sendmail, lpd, crond, getty, bdflush, pagedaemon, swapper, inetd, named, routed, dhcpcd, portmap, nfsd, smbd, httpd, ntpd
- ◆ Limity zasobów procesów: polecenia *limit, ulimit*

2. Podaj komendy do sterowania procesami w systemie Unix.

- ◆ Uruchamianie w tle: *&*
 - ◆ np. *ls -R / &*
- ◆ Zatrzymanie procesu pierwszoplanowego: *^Z*
- ◆ Ponowne uruchamianie procesu w tle: *bg*
- ◆ Listowanie procesów w tle: *jobs*
- ◆ Odwołanie do procesu n w tle: *%n*
- ◆ Odwołanie do procesu xyz w tle: *%?xyz*
- ◆ Przenoszenie z tła na pierwszy plan: *fg %*

3. Narysuj diagram stanów procesów.



4. Opisz zawartość PCB.

Blok kontrolny procesu (PCB)

- ◆ Każdy proces w systemie operacyjnym jest reprezentowany przez blok kontrolny procesu (ang. process control block - PCB) zawierający
 - ◆ stan procesu - gotowy, nowy, aktywny, czekający, zatrzymany
 - ◆ licznik rozkazów - adres następnego rozkazu do wykonania w procesie
 - ◆ rejestrzy procesora - zależą od architektury komputera: akumulatory, rejestrzy (ogólne, bazowe, indeksowe) wskaźniki stosu przechowywane aby proces mógł być kontynuowany po przerwaniu
 - ◆ stan procesora (PSW – IBM/370, EFLAGS – Pentium II)
 - ◆ informacje o planowaniu przydziału procesora - priorytet procesu, wskaźniki do kolejek porządkujących zamówienia
 - ◆ informacje o zarządzaniu pamięcią - zawartości rejestrów granicznych, tablice stron, tablice segmentów w zależności od systemu używanej pamięci
 - ◆ informacje do rozliczeń - ilość zużytego czasu procesora i czasu rzeczywistego, ograniczenia czasowe, numery kont, numery zadań
 - ◆ informacje o stanie we/wy - lista zaalokowanych urządzeń, wykaz otwartych plików

5. Omów przyczyny przełączania procesu.

- ◆ Przerwanie (ang. interrupt) – zdarzenie zewnętrzne w stosunku do procesu
- ◆ Pułapka (ang. trap) – zdarzenie wewnętrz aktualnie przetwarzanego procesu, błąd lub warunek wyjątku
- ◆ Polecenie administracyjne (ang. supervisor call) – wywołanie funkcji systemu operacyjnego

6. Omów rodzaje planistów i zadania przez nich realizowane.

Planisci

- ◆ Planista długoterminowy (ang. long-term scheduler) lub planista zadań (ang. job scheduler) - wybiera procesy, które powinny być sprowadzone do pamięci z kolejek procesów (niegotowych)
 - ◆ zazwyczaj na dyskach
- ◆ Planista krótkoterminowy (ang. short-term scheduler) lub planista przydziału procesora (ang. CPU scheduler) - wybiera proces następny do wykonania z kolejki procesów gotowych i przydziela mu procesor

- ◆ Planista krótkoterminowy jest wołany bardzo często (milisekundy) dlatego musi być bardzo szybki
 - ◆ raz na 100ms
 - ◆ Planista długoterminowy jest wołany rzadko (sekundy, minuty) dlatego może nie być szybki
 - ◆ Planista długoterminowy nadzoruje stopień wieloprogramowości (tzn. liczbę procesów w pamięci)
 - ◆ Proces może być opisany jako jeden z
 - ◆ ograniczony przez we/wy (ang. I/O bound) - więcej czasu zajmuje we/wy niż dostęp do procesora
 - ◆ ograniczony przez dostęp do procesora (ang. CPU bound) - więcej czasu zajmują obliczenia, we/wy sporadyczne
 - ◆ Planista długoterminowy powinien dobrać mieszankę procesów (ang. process mix) zawierającą zarówno procesy ograniczone przez we/wy jak i procesor
 - ◆ Planista średnioterminowy (ang. medium-term scheduler) – usuwa procesy z pamięci, zmniejszając stopień wieloprogramowości, a później sprowadza je ponownie
 - ◆ uzyskanie lepszej mieszanki procesów
 - ◆ żądania przydziału pamięci przekraczają ilość dostępnej pamięci
 - ◆ postępowanie takie nazywamy wymianą (ang. swapping)
-

7. Omów przełączanie kontekstu.

- ◆ Gdy procesor przełącza do innego procesu system musi zachować stan starego procesu i załadować zachowany stan nowego procesu
 - ◆ czynność tę nazywamy przełączaniem kontekstu (ang. context switch)
- ◆ Przełączanie kontekstu jest ceną za wieloprogramowość
 - ◆ system operacyjny nie wykonuje wtedy żadnej użytecznej pracy
- ◆ Czas przełączenia kontekstu zależy od sprzętu (zwykle od 1 do 1000 µs)
 - ◆ np. od ilości rejestrów (Sun UltraSPARC ma po kilka zbiorów rejestrów)

8. Omów tworzenie procesu zombie i orphan.

Zombie – potomek przestał istnieć, ale rodzic nadal w pamięci

Orphan – rodzic przestał istnieć ale potomek nadal jest w pamięci

9. Omów komunikację pośrednią i bezpośrednią między procesami.

Komunikacja bezpośrednią

- ◆ Proces musi jawnie nazwać odbiorcę
 - ◆ nadaj(P,komunikat) - nadaj komunikat do procesu P
 - ◆ odbierz(Q,komunikat) - odbierz komunikat od procesu Q
- ◆ Właściwości łącza
 - ◆ łącze jest ustanawiane automatycznie, do komunikowania wystarczy znajomość identyfikatorów
 - ◆ łącze dotyczy dokładnie dwóch procesów
 - ◆ między każdą parą procesów istnieje dokładnie jedno łącze
 - ◆ łącze jest zwykle dwukierunkowe, ale może być jednokierunkowe

Komunikacja pośrednia

◆ Komunikaty są nadawane i odbierane za pomocą skrzynek pocztowych (ang. mailboxes) nazywanych także portami (ang. ports)

- ◆ każda skrzynka ma swój unikalny identyfikator
- ◆ procesy komunikują się jeśli mają wspólną skrzynkę

◆ Własności łącza

- ◆ łącze jest ustanawiane jedynie wtedy gdy procesy dzielą skrzynkę
- ◆ łącze może być związane z więcej niż dwoma procesami
- ◆ każda para procesów może mieć kilka łączy z których każdy odpowiada jakiejś skrzynce
- ◆ łącze może być jedno- lub dwukierunkowe

◆ System operacyjny dostarcza mechanizmów do

- ◆ tworzenia nowej skrzynki
- ◆ nadawania i odbierania komunikatów za pomocą skrzynki
- ◆ likwidowania skrzynki

◆ Dzielenie skrzynki : P, Q, R dzielą skrzynkę A

- ◆ P nadaje; Q i R odbierają
- ◆ który proces otrzyma komunikat nadany przez P?

◆ Rozwiążanie

- ◆ zezwalać jedynie na łącza między dwoma procesami
- ◆ pozwalać najwyżej jednemu procesowi na odbiór
- ◆ pozwalać aby system wybrał i poinformował odbiorcę

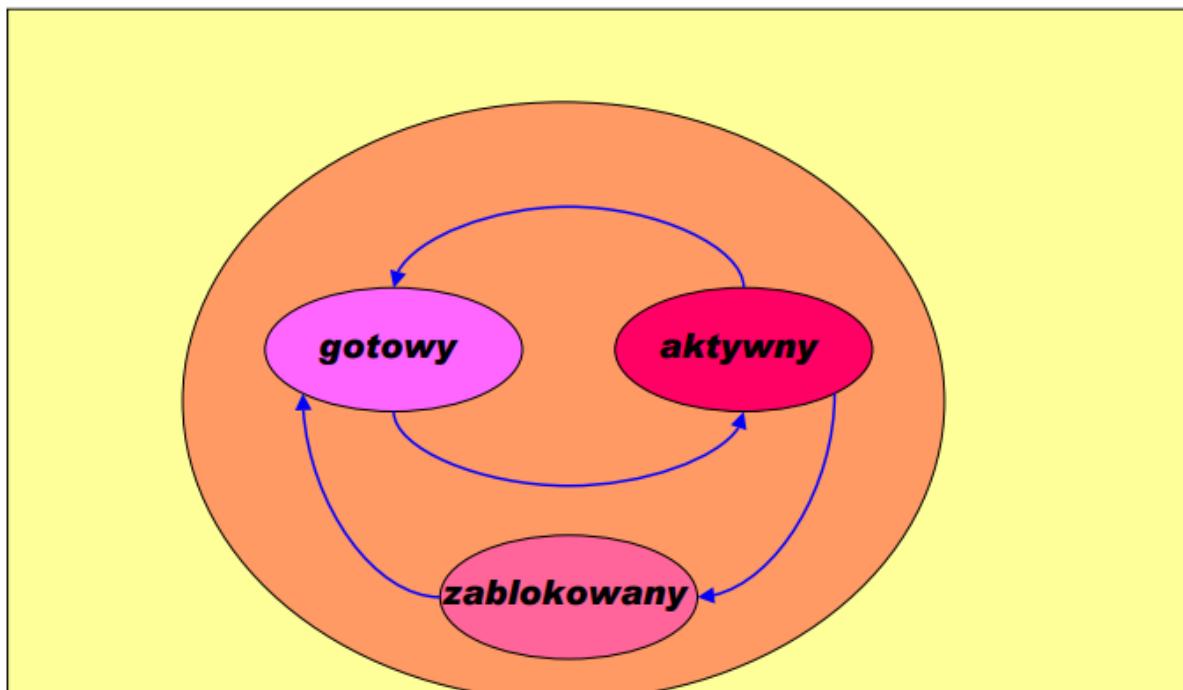
10. Wymień zasoby używane przez wątki.

Wątki

- ◆ Wątek (ang. thread) nazywany niekiedy procesem lekkim (ang. lightweight process - LWP) jest podstawową jednostką wykorzystania procesora. W skład tej jednostki wchodzą
 - ◆ licznik rozkazów
 - ◆ zbiór rejestrów
 - ◆ obszar stosu
- ◆ Wątek współużytkuje z innymi równorzędnymi wątkami
 - ◆ sekcję kodu
 - ◆ sekcję danych
 - ◆ zasoby systemu (takie jak otwarte pliki i sygnały) zwane wspólnie zadaniem (ang. task)
- ◆ Proces tradycyjny lub ciężki (ang. heavyweight) to zadanie o jednym wątku

11. Narysuj diagram stanów wątku.

Stany wątku



12. Wymień typy wątków.

Typy wątków

- ◆ Wątki (ang. kernel-level, KLT) obsługiwane przez jądro
 - ◆ Mach, OS/2, Windows NT/2K/XP, Linux, BeOS, Mac OS X
- ◆ Wątki tworzone na poziomie użytkownika (ang. user-level, ULT) za pomocą funkcji bibliotecznych
 - ◆ system Andrew
 - ◆ P-wątki (ang. Pthreads) - implementacje w postaci bibliotek
 - ◆ standard POSIX (IEEE 1003.1c) definiujący interfejs programów użytkowych (API) do tworzenia wątków oraz ich synchronizowania
 - ◆ Mach: C-threads,
 - ◆ Solaris 2: UI-threads
 - ◆ zaleta: szybsze przełączanie, wada: planowanie wątków
- ◆ Hybrydowe podejście - Solaris 2
- ◆ Wątki zarządzane przez JVM

13. Wymień sposoby odwzorowania wątków.

Sposoby odwzorowania wątków

- ◆ Wiele do jednego (ang. Many-to-One)
 - ◆ biblioteka “green threads” Solaris 2
 - ◆ wykonanie blokującego wywołania systemowego blokuje proces
- ◆ Jeden do jednego (ang. One-to-One)
 - ◆ OS/2, Windows NT/2K/XP, Linux
 - ◆ równoległe wykonywanie wątków (na procesorach)
 - ◆ ograniczona ilość wątków
- ◆ Wiele do wielu (ang. Many-to-Many)
 - ◆ Solaris 2, IRIX, HP-UX, True 64 (Digital UNIX)
 - ◆ funkcjonalność w Windows NT/2K/XP w postaci biblioteki włókien (ThreadFiber)
- ◆ Jeden do wielu (ang. One-to-Many)
 - ◆ wątek może migrować pomiędzy systemami (Clouds, Emerald)

WYKŁAD 7

1. Wymień sytuacje w jakich planista przydziału procesora podejmuje decyzję o przydiale procesora.

Planista przydziału procesora

- ◆ Planista (**krótkoterminowy**) przydziału procesora (ang. CPU scheduler) wybiera jeden proces spośród przebywających w pamięci procesów gotowych do wykonania i przydziela mu procesor
- ◆ Decyzje o przydiale procesora podejmowane są
 - ◆ 1. gdy proces przeszedł od stanu aktywności do stanu czekania
 - ◆ np. z powodu operacji we/wy
 - ◆ np. czekanie na zakończnie potomka
 - ◆ 2. gdy proces przeszedł od stanu aktywności do stanu gotowości
 - ◆ np. wskutek przerwania
 - ◆ 3. gdy proces przeszedł od stanu czekania do stanu gotowości
 - ◆ np. po zakończeniu operacji we/wy
 - ◆ 4. gdy proces kończy działanie
- ◆ W pkt. 2 i 3 można dokonać wyboru procesu któremu przydzielić CPU

- ◆ Planowanie w sytuacjach 1 i 4 nazywamy niewyłaszczającym (ang. nonpreemptive)
- ◆ Algorytm planowania (szeregowania) nazywamy wywłaszczającym (ang. preemptive) w pozostałych sytuacjach
 - ◆ systemy Windows 9x, 2K, XP, Mac OS X
- ◆ Planowanie bez wywłaszczeń: proces, który otrzyma procesor, zachowuje go tak długo aż nie odda go z powodu przejścia w stan oczekiwania lub zakończenia
 - ◆ planowanie to nie wymaga zegara
 - ◆ systemy Windows 3.x i Apple Macintosh OS
- ◆ Planowanie wywłaszczające: drogie i ryzykowne
- ◆ Co się stanie gdy wywłaszczony zostanie proces w trakcie wykonywania funkcji systemowej (np. zmiany danych w blokach opisu kolejki we/wy)?
- ◆ UNIX czeka z przełączeniem kontekstu do zakończenia wywołania systemowego lub do zablokowania procesu na we/wy
 - ◆ model ten słabo nadaje się do przetwarzania w czasie rzeczywistym
- ◆ Nie można wywłaszczać procesu gdy wewnętrzne struktury jądra są niespójne
- ◆ Blokowanie przerwań przy wejściu do ryzykownych fragmentów kodu jądra
 - ◆ włączanie i wyłączanie przerwań jest kosztowne (np. wiele procesorów)

2. Wymień warunki planowania bez wywłaszczeń.

- ◆ Planowanie bez wywłaszczeń: proces, który otrzyma procesor, zachowuje go tak długo aż nie odda go z powodu przejścia w stan oczekiwania lub zakończenia
 - ◆ planowanie to nie wymaga zegara
 - ◆ systemy Windows 3.x i Apple Macintosh OS

3. Co to jest dispatcher oraz dispatch latency?

Ekspedytor

- ◆ Ekspedytor (ang. dispatcher) jest modułem, który faktycznie przekazuje procesor do dyspozycji procesu wybranego przez planistę krótkoterminowego; obowiązki ekspedytora to:
 - ◆ przełączanie kontekstu
 - ◆ przełączanie do trybu użytkownika
 - ◆ wykonanie skoku do odpowiedniej komórki w programie użytkownika w celu wznowienia działania programu
- ◆ Opóźnienie ekspedycji (ang. dispatch latency) to czas, który ekspedytor zużywa na wstrzymanie jednego procesu i uaktywnienie innego

4. Co to jest przepustowość oraz wykorzystanie CPU?

◆ Definicje

- ◆ Wykorzystanie procesora (ang. CPU utilization) – procent czasu, przez który procesor pozostaje zajęty
 - ◆ najlepiej by było gdyby procesor był nieustannie zajęty pracą
 - ◆ powinno się mieścić od 40% (słabe obciążenie systemu) do 90% (intensywna eksploatacja)
- ◆ Przepustowość (ang. throughput) - liczba procesów kończących w jednostce czasu
 - ◆ długie procesy - 1 na godzinę, krótkie - 10 na sekundę

5. Podaj definicje czasu (cyklu) przetwarzania, czasu oczekiwania, czasu odpowiedzi.

◆ Definicje (c.d.)

- ◆ Czas cyklu przetwarzania (ang. turnaround time) – czas między nadaniem procesu do systemu a chwilą zakończenia procesu
 - ◆ suma czasów czekania na wejście do pamięci, czekania w kolejce procesów gotowych, wykonywania procesu przez CPU i wykonywania operacji we/wy
- ◆ Czas oczekiwania (ang. waiting time) - suma okresów, w których proces czeka w kolejce procesów gotowych do działania

◆ Definicje (c.d.)

- ◆ Czas odpowiedzi lub reakcji (ang. response time) - ilość czasu między wysłaniem żądania a pojawieniem się odpowiedzi bez uwzględnienia czasu potrzebnego na wyprowadzenie odpowiedzi (np. na ekran).
 - ◆ czas odpowiedzi jest na ogół uzależniony od szybkości działania urządzenia wyjściowego
 - ◆ miara zastępująca miarę czasu cyklu przetwarzania w systemach interakcyjnych
 - ◆ np. kliknięcie myszą obiektu – mniej niż 0,1s

6. Wymień kryteria optymalizacji algorytmu planowania.

Kryteria optymalizacji algorytmów planowania

- ◆ Maksymalne wykorzystanie procesora
- ◆ Maksymalna przepustowość
- ◆ Minimalny czas cyklu przetwarzania
- ◆ Minimalny czas oczekiwania
- ◆ Minimalny czas odpowiedzi
 - ◆ optymalizujemy miary średnie, maksymalne (ew. minimalne)
 - ◆ minimalizowanie wariancji czasu odpowiedzi zamiast średniego czasu odpowiedzi w systemach z podziałem czasu
 - ◆ mało algorytmów minimalizujących wariancję
 - ◆ pożądany system z sensownym i przewidywalnym czasem odpowiedzi zamiast systemu o lepszym średnio czasie odpowiedzi i bardzo zmiennym

7. Opisz algorytm FCFS.

Planowanie metodą FCFS

- ◆ Pierwszy zgłoszony-pierwszy obsłużony (ang. first-come, first-served - FCFS)
- ◆ Implementuje się łatwo za pomocą kolejek FIFO - blok kontrolny procesu dołączany na koniec kolejki, procesor dostaje PCB z czoła kolejki
- ◆ Przykład: Proces Czas trwania fazy

P_1	24
P_2	3
P_3	3

- ◆ Przypuśćmy, że procesy nadjejdą w porządku: P_1 , P_2 , P_3

8. Opisz algorytm SJF wywłaszczający

Planowanie metodą SJF

- ◆ Algorytm napierw najkrótsze zadanie (ang. shortest-job-first - SJF) wiąże z każdym procesem długość jego najbliższej z przyszłych faz procesora; gdy procesor staje się dostępny wówczas zostaje przydzielony procesowi o najkrótszej następnej fazie procesora (gdy fazy są równe to stosujemy planowanie FCFS)
- ◆ Algorytm SJF może być:
 - ◆ wywłaszczający - SJF usunie proces jeśli nowy proces w kolejce procesów gotowych ma krótszą następną fazę procesora od czasu do zakończenia procesu
 - ◆ gdy w kolejce procesów gotowych są procesy o jednakowych fazach to stosujemy FCFS
 - ◆ algorytm SRTF (ang. shortest-remaining-time-first) - najpierw najkrótszy pozostały czas
 - ◆ niewywłaszczający - pozwól procesowi zakończyć
- ◆ SJF charakteryzuje to, że
 - ◆ ma dobry czas odpowiedzi dla krótkich procesów
 - ◆ jest krzywdzący dla procesów długich
 - ◆ może powodować zagłodzenie procesów
- ◆ Planowanie metodą SRTF zwykle daje lepszy czas przetwarzania niż SJF

9. Opisz algorytm HRRN.

Planowanie metodą HRRN

- ◆ Def. Stosunkiem reaktywności (ang. response ratio) nazywamy liczbę $R = 1 + w/t$, gdzie w oznacza czas oczekiwania na procesor zaś t - fazę procesora
- ◆ Największy stosunek reaktywności jako następny (ang. highest response ratio next - HRRN)
- ◆ Faworyzuje krótkie zadania, lecz po pewnym czasie oczekiwania dłuższy proces uzyska CPU
- ◆ Podobnie jak SJF i SRTF również algorytm HRRN wymaga oszacowania dla następnej fazy procesora

10. Podaj wzór na oszacowanie następnej fazy procesora.

Następna faza procesora

- ◆ SJF jest optymalny: daje minimalny średni czas oczekiwania dla danego zbioru procesów
- ◆ Nie ma sposobu na poznanie długości następnej fazy, możemy ją jedynie oszacować
- ◆ Można tego dokonać wyliczając średnią wykładniczą poprzednich faz procesora
- ◆ $t(n)$ = długość n -tej fazy procesora
- ◆ a - liczba z przedziału $[0,1]$, zwykle 0.5
- ◆ Definiujemy średnią wykładniczą jako:

$$s(n+1) = a*t(n) + (1-a)*s(n)$$

gdzie $s(n+1)$ = przewidywana długość następnej fazy a $s(n)$ przechowuje dane z minionej historii; $s(0)$ – stała (np. średnia wzięta z całego systemu)

11. Opisz planowanie priorytetowe.

Planowanie priorytetowe

- ◆ SJF jest przykładem planowania priorytetowego (ang. priority scheduling) w którym każdemu procesowi przypisuje się priorytet (liczbę)
- ◆ Priorytety należą do pewnego skończonego podzbioru liczb naturalnych np. [0..7], [0,4095]
- ◆ Prz. nice {+|- n} *polecenie*
- ◆ Procesor przydziela się procesowi o najwyższym priorytecie (jeśli priorytety są równe to FCFS)
 - ◆ planowanie priorytetowe wywłaszczające
 - ◆ planowanie priorytetowe niewywłaszczające
- ◆ SJF - priorytet jest odwrotnością następnej fazy
- ◆ Problem: nieskończone zablokowanie (ang. indefinite blocking) lub głodzenie (ang. starvation) - procesy o małym priorytecie mogą nigdy nie dostać czasu procesora
- ◆ Kraży taka pogłoska, że gdy w 1973 r. wycofywano z eksploatacji w MIT komputer IBM 7094 wykryto zagłodzony niskopriorytetowy proces przedłożony do wykonania jeszcze w 1967 r.
- ◆ Rozwiązanie: postarzanie (ang. aging) polegające na podwyższeniu priorytetu procesów oczekujących już zbyt długo
- ◆ Prz. Proces ma priorytet 127, co 15 min zwiększamy priorytet o 1 więc w najgorszym przypadku (tzn. jeśli nie dostanie się do CPU) po 32 godzinach proces będzie miał najwyższy priorytet równy 0 (co wcale nie oznacza, że dostanie się wtedy do CPU)

12. Opisz algorytm RR.

Planowanie rotacyjne

- ◆ Planowanie rotacyjne, RR (ang. round-robin, time-slicing) zaprojektowano dla systemów z podziałem czasu
- ◆ Każdy proces otrzymuje małą jednostkę czasu, nazywaną kwantem czasu (ang. time quantum, time slice) zwykle od 10 do 100 milisekund. Gdy ten czas minie proces jest wywłączany i umieszczany na końcu (ang. tail) kolejki zadań gotowych (FCFS z wywłączczeniami)
- ◆ średni czas oczekiwania jest stosunkowo długi
- ◆ Jeśli jest n procesów w kolejce procesów gotowych a kwant czasu wynosi q to każdy proces otrzymuje $1/n$ czasu procesora porcjami wielkości co najwyżej q jednostek czasu.
- ◆ Każdy proces czeka nie dłużej niż $(n-1)*q$ jednostek czasu
- ◆ Wydajność algorytmu
 - ◆ gdy q duże to RR(q) przechodzi w FCFS
 - ◆ gdy q małe to mamy dzielenie procesora (ang. processor sharing) ale wtedy q musi być duże w stosunku do przełączania kontekstu (inaczej mamy spowolnienie)

13. Rozważmy procesy: P1, P2, P3, P4, P5 o następujących czasach trwania fazy

(ms): f1, f2, f3, f4, f5, które przybyły o czasie t1, t2, t3, t4, t5 i priorytetach:

p1, p2, p3, p4, p5 przy czym najwyższy priorytet ma p1

. Narysuj diagramy

Gantta dla algorytmów FCFS, HRRN, SJF, SRTF, priorytetowy z i bez

wywłaszczeń, RR (kwant=q) oraz oblicz ich średnie czasy cyklu przetwarzania i oczekiwania. Uwaga: jeśli nie są podane t1, t2, t3, t4, t5 to przyjmujemy, że t1 = t2 = t3 = t4 = t5 = 0 oraz że procesy przybyły w porządku P1, P2, P3, P4, P5 jeśli zaś nie są podane p1, p2, p3, p4, p5 to przyjmujemy p1 = p2 = p3 = p4 = p5.

14. Opisz algorytm wielopoziomowych kolejek ze sprzężeniem zwrotnym.

Kolejki wielopoziomowe ze sprzężeniem zwrotnym

- ◆ Kolejki wielopoziomowe z sprzężeniem zwrotnym (ang. multilevel feedback queue scheduling) umożliwiają przesuwanie procesów między kolejkami
- ◆ Proces, który zużywa za dużo procesora można zdymisjonować poprzez przesunięcie go do kolejki o niższym priorytecie i dzięki temu zapobiec zagłodzeniu innych procesów
- ◆ Postępowanie takie prowadzi do pozostawienia procesów ograniczonych przez we/wy oraz interakcyjnych w kolejkach o wyższych priorytetach

15. Opisz planowanie wątków.

Planowanie wątków

- ◆ PCS (ang. process-contention-scope) – planowanie ULT na bazie LWP (struktura pośrednia jądra) zwykle priorytetowe
 - ◆ modele M:1 (many-to-one) i M:M (many-to-many)
 - ◆ nie ma podziału czasu między wątkami o równym priorytecie
- ◆ SCS (ang. system-contention-scope) – planowanie wszystkich wątków KLT w systemie
 - ◆ model 1:1 (one-to-one) (XP, Solaris, Linux)
- ◆ POSIX Pthread API – planowanie wątków
 - ◆ Klasy: SCHED_FIFO, SCHED_RR , SCHED_OTHER
 - ◆ Polityki: PTHREAD_SCOPE_PROCESS – planowanie PCS, PTHREAD_SCOPE_SYSTEM – planowanie SCS (na systemach M:M tworzy LWP dla każdego ULT t.j. efektywnie mamy 1:1)
 - ◆ Pthread API pozwala programistom na zmianę priorytetu wątku (librt)

16. Wymień metody oceny algorytmów planowania.

Ocena algorytmów

- ◆ Modelowanie deterministyczne - przyjmuje się konkretne, z góry określone obciążenie robocze systemu i definiuje zachowanie algorytmu w warunkach tego obciążenia; jest to odmiana oceny analitycznej (ang. analytic evaluation)
 - ◆ dla danego zbioru procesów mających zadane uporządkowanie i wyrażone w milisekundach fazy procesora rozważamy algorytmy planowania (FCFS, SJF, RR (o zadanym kwancie czasu) , itp.)
 - ◆ Pytanie: który algorytm minimalizuje czas oczekiwania?
- ◆ Modelowanie deterministyczne jest proste i szybkie, daje konkretne liczby pozwalające dokonać wyboru algorytmu planowania
- ◆ Modelowanie deterministyczne wymaga jednak specyficznych sytuacji i dokładnej wiedzy dlatego nie zasługuje na miano ogólnie użytecznego

WYKŁAD 8

1. Wymień i scharakteryzuj kategorie ziarnistości.

Ziarnistość

◆ Ziarnistość (ang. Granularity) to częstotliwość synchronizacji między procesami w systemie

- ◆ paralelizm niezależny (ang. independent)
- ◆ paralelizm bardzo gruboziarnisty (bardzo niedokładny) (ang. very coarse)
- ◆ paralelizm gruboziarnisty (niedokładny) (ang. coarse)
- ◆ paralelizm średnioziarnisty (średni) (ang. medium)
- ◆ paralelizm drobnoziarnisty (dokładny) (ang. fine)

◆ http://www.bit-tech.net/gaming/pc/2006/11/02/Multi_core_in_the_Source_Engin/1

Rozmiar ziarna	Opis	Częstotliwość synchronizacji (instrukcje)
Drobne	Paralelizm w jednym strumieniu instrukcji	<20
Średnie	Przetwarzanie równolegle albo wielozadaniowość w ramach jednej aplikacji	20-200
Grube	Wieloparallelizm współbieżnych procesów w środowisku wieloprogramowym	200-2000
Bardzo grube	Przetwarzanie rozproszone w wielu węzłach sieci tworzących jedno środowisko obliczeniowe	2000-1M
Niezależne	Wiele niezwiązańanych ze sobą procesów	Nie dotyczy

2. Wymień i scharakteryzuj techniki szeregowania wątków na wielu procesorach.

Szeregowanie wieloprocesorowe

- ◆ Ang. Scheduling
 - ◆ Przydzielanie procesów procesorom
 - ◆ Użycie wieloprogramowości na poszczególnych procesorach
 - ◆ Rzeczywiste skierowanie (ang. dispatching) procesu do wykonania
- ◆ Metoda szeregowania jest zależna od ziarnistości i ilości procesorów

Szeregowanie wątków

- ◆ ang. Thread Scheduling
- ◆ Wykonanie wątku jest oddzielone od reszty procesu
- ◆ Aplikację można realizować jako zbiór wątków, które współpracują i wykonują się współbieżnie w tej samej przestrzeni adresowej
- ◆ Jeśli różne wątki aplikacji wykonują się jednocześnie na różnych procesorach, wzrost wydajności bywa ogromny

Szeregowanie wątków na wielu procesorach

- ◆ ang. Multiprocessor Thread Scheduling
- ◆ Współdzielenie obciążenia (ang. load sharing)
 - ◆ procesy nie są przydzielane konkretnemu procesorowi
- ◆ Szeregowanie grupowe lub zespołowe (szeregowanie zbiorów) (ang. gang scheduling)
 - ◆ zbiór powiązanych wątków jest jednocześnie kierowany do wykonania przez zbiór wielu procesorów, na zasadzie jeden do jednego
- ◆ Rezerwacja procesorów (dedykowane przydzielanie procesora)(ang. dedicated processor assignment) – przeciwieństwo współdzielenia
 - ◆ wątkom są przydzielane procesory, na zasadzie jeden do jednego
- ◆ Szeregowanie dynamiczne (ang. dynamic scheduling)
 - ◆ liczba wątków w procesie może się zmieniać w czasie wykonania

3. Wymień i scharakteryzuj klasy zadań czasu rzeczywistego.

Szeregowanie RT

- ◆ System RT to taki system operacyjny w którym prawidłowe działanie zależy nie tylko od logicznych wyników obliczeń, ale także od czasu, w jakim zostaną one uzyskane
- ◆ Zadania lub procesy próbują kontrolować lub reagować na zdarzenia zachodzące w świecie zewnętrznym
- ◆ Zdarzenia te zachodzą w „czasie rzeczywistym” dlatego zadania muszą nadążać za zdarzeniami, które ich dotyczą

Szeregowanie RT (c.d.)

- ◆ Klasyfikacja zadań
 - ◆ twarde zadanie czasu rzeczywistego (ang. hard real-time task) musi dotrzymać wyznaczonego terminu
 - ◆ miękkie zadanie czasu rzeczywistego (ang. soft real-time task) może nie dotrzymać wyznaczonego terminu
 - ◆ zadanie nieokresowe (ang. aperiodic task) ma określony termin (ang. deadline) rozpoczęcia {lub | i} zakończenia
 - ◆ zadanie okresowe (ang. periodic task) ma określony termin rozpoczęcia {lub | i } zakończenia „dokładnie co T jednostek” lub „raz na okres T”

4. Scharakteryzuj sytuację odwrócenia priorytetów.

Odwrocenie priorytetów

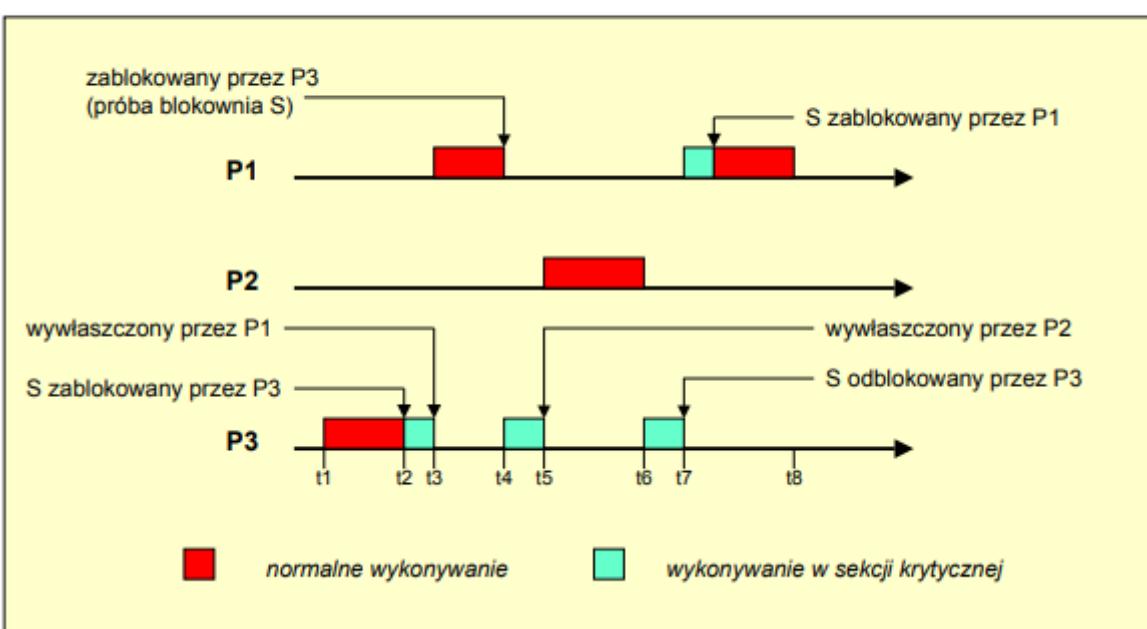
◆ Unikanie nieograniczonego odwrócenia priorytetów

- ◆ protokół dziedziczenia priorytetów (ang. priority-inheritance protocol) - w czasie użycia zasobów proces dziedziczy wysoki priorytet (po użyciu stary priorytet jest przywracany)
- ◆ pułap priorytetów (ang. priority ceiling) – przydzielenie zasobu skutkuje zmianą na priorytet o jeden wyższy od najwyższego priorytetu w systemie; zwolnienie zasobu przywraca stary priorytet

◆ Prz. Pathfinder na Marsie (lipiec, 1997)

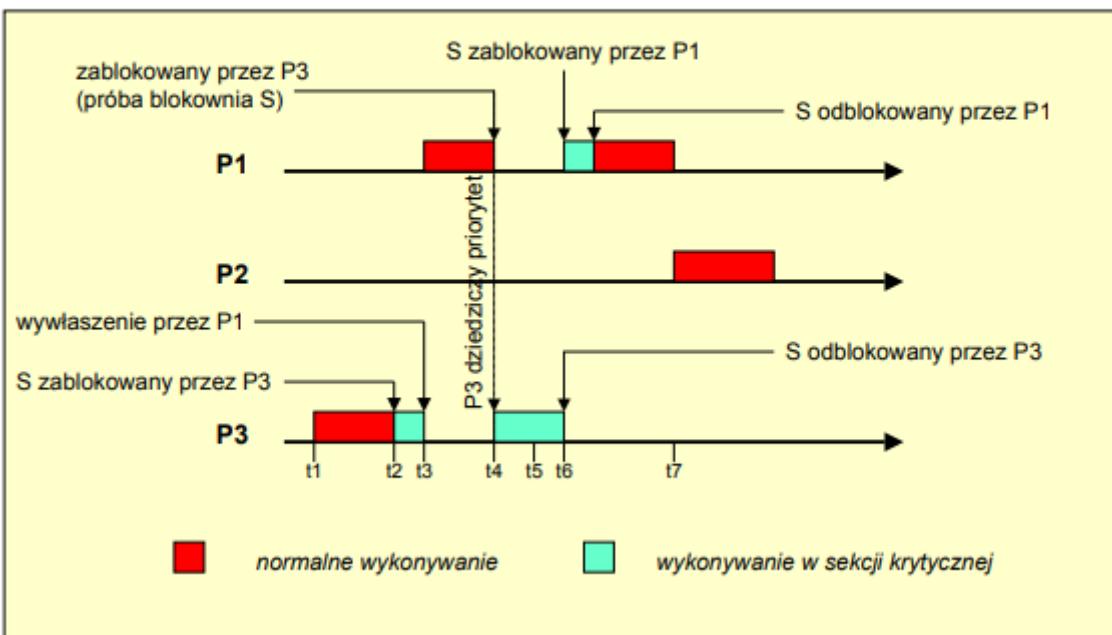
- ◆ P1: sprawdzenie działania pojazdu i oprogramowania
 - ◆ jeśli zadanie przekroczy termin to integralność Pathfindera jest nadwyrężona i oprogramowanie się resetuje (utrata danych)
- ◆ P2: przetwarzanie zdjęć
- ◆ P3: testowanie sprzętu

Odwrocenie priorytetów



5. Scharakteryzuj pojęcia odwrócenia i dziedziczenia priorytetów .

Dziedziczenie priorytetów



6. Wymień i scharakteryzuj algorytmy szeregowania czasu rzeczywistego.

Algorytmy szeregowania RT

- ◆ ang. Real-Time Scheduling
- ◆ Statyczne algorytmy tabelaryczne (ang. static table-driven)
 - ◆ na podstawie statycznej analizy wykonalnych planów szeregowania buduje się harmonogram określający kiedy zadanie ma się wykonywać
 - ◆ stosuje się do zadań okresowych
 - ◆ najpierw najwcześniejszy termin (ang. earliest deadline first, EDF)
- ◆ Statyczne algorytmy priorytetowe z wywłączaniem (ang. static priority-driven preemptive)
 - ◆ analiza statyczna wykonalnych planów na podstawie której przydziela się zadaniom priorytety i stosuje się tradycyjnego planistę priorytetowego z wywłączaniem
 - ◆ algorytm częstotliwościowo monotoniczny (ang. rate monotonic scheduling, RMS)

7. Podaj warunek konieczny na dotrzymanie twardych terminów dla zadań okresowych.

Szeregowanie terminowe (c.d.)

◆ Wykorzystywane informacje

- ◆ czas gotowości (ang. ready time) – czas w którym proces staje się gotowy do wykonania (znany w przypadku zadania okresowego)
- ◆ termin rozpoczęcia (ang. starting deadline) – czas w którym zadanie musi się rozpocząć
- ◆ termin zakończenia (ang. completion deadline) – czas w którym zadanie musi się zakończyć
- ◆ czas przetwarzania (ang. processing time) – czas wymagany na wykonanie zadania
 - ◆ podany przez użytkownika lub wyznaczony przy pomocy średniej wykładniczej

8. Opisz algorytm RMS i podaj górne oszacowanie na wykorzystanie CPU.

Szeregowanie metodą RMS

- ◆ okres T_i zadania P_i to ilość czasu od przybycia jednej instancji P_i do następnej
 - ◆ zadanie o okresie 50ms ma częstotliwość 20Hz
- ◆ czas wykonywania C_i - czas CPU każdej instancji P_i
- ◆ aby możliwe było dotrzymanie twardych terminów musi być spełniona nierówność
$$C_1/T_1 + C_2/T_2 + \dots + C_n/T_n \leq 1 \quad (S)$$
- ◆ Można wykazać, że algorytm EDF spełnia nierówność (S)
- ◆ Szeregowanie częstotliwościowe monotoniczne (ang. rate monotonic scheduling, RMS) - zadanie o krótszym okresie ma wyższy priorytet. Wykorzystanie procesora dla RMS:

$$C_1/T_1 + C_2/T_2 + \dots + C_n/T_n \leq n(2^{1/n} - 1) \rightarrow_n \ln 2 \approx 0,6931$$

WYKŁAD 9

1. Opisz szkodliwą rywalizację (ang. race condition).

- ◆ Sytuacja w której kilka procesów współbieżnie wykonuje działania na tych samych danych i w konsekwencji wynik tych działań zależy od porządku w jakim one następowały nazywa się szkodliwą rywalizacją (lub sytuacją wyścigów) (ang. race condition)

2. Opisz problem sekcji krytycznej.

Problem sekcji krytycznej

- ◆ W systemie działają jednocześnie procesy: $P_0, P_1, P_2, \dots, P_{n-1}$
- ◆ Każdy proces ma segment kodu, zwany sekcją krytyczną (ang. critical section), w którym może zmieniać wspólne zmienne wykorzystywane przez $P_0, P_1, P_2, \dots, P_{n-1}$
- ◆ Gdy proces P_i , $i=0..n-1$, wykonuje się w swojej sekcji krytycznej to proces P_k , gdzie $k \neq i$, nie może wykonywać się w swojej sekcji krytycznej
- ◆ Wykonywanie sekcji musi podlegać wzajemnemu wykluczaniu w czasie (ang. mutual exclusion)
- ◆ Jakie warunki musi spełniać protokół dostępu do wspólnych zmiennych?
 - ◆ procesy działają z prędkościami niezerowymi i nie ma ograniczeń na relatywne ich prędkości
 - ◆ proces w sekcji wejściowej (ang. entry section) prosi o zezwolenie na wejście do sekcji krytycznej, wykonuje sekcję krytyczną, opcjonalnie wykonuje sekcję wyjściową (ang. exit section) a następnie pozostały kod zwany resztą (ang. reminder section)

3. Wymień i scharakteryzuj warunki sekcji krytycznej.

Warunki sekcji krytycznej

- ◆ (E): Wzajemne wykluczanie (ang. mutual exclusion): jeśli proces P_i działa w swojej sekcji krytycznej, to żaden inny proces nie działa w swojej sekcji krytycznej
- ◆ (P): Postęp (ang. progress): jeśli żaden proces nie działa w swojej sekcji krytycznej i jakieś procesy chcą wejść do swoich sekcji krytycznych wówczas kandydatem do wejścia do swojej sekcji krytycznej może być jedynie proces, który nie wykonuje swojej reszty i wybór ten nie może być odwlekany w nieskończoność
- ◆ (W): Ograniczone czekanie (ang. bounded waiting): jeśli dany proces zgłosił chęć wejścia do swojej sekcji krytycznej to musi istnieć graniczna liczba wejść innych procesów do swoich sekcji krytycznych po której dany proces uzyska pozwolenie na wejście do swojej sekcji krytycznej

4. Scharakteryzuj sprzętowe środki synchronizacji.

Sprzętowe środki synchronizacji

- ◆ Wiele komputerów posiada sprzętowe wsparcie do rozwiązywania problemów sekcji krytycznej
- ◆ W środowisku jednoprocesorowym można zakazać (zamaskować) przerwania w trakcie modyfikowania zmiennej dzielonej
 - ◆ gwarantuje to brak wywłaszczeń tzn. gwarantuje sekwencyjność wykonywania się kodu
 - ◆ rozwiązanie to jest nieprzydatne w środowisku wieloprocesorowym ze względu na konieczność przekazywania komunikatów do pozostałych procesorów
- ◆ Współczesne komputery posiadają rozkazy maszynowe wykonywane w sposób niepodzielny (nieprzerywalny, ang. atomically)
 - ◆ testuj pamięć i ustaw (TestAndSet)
 - ◆ zamień zawartość dwóch słów w pamięci (Swap)

5. Scharakteryzuj semafory.

Semafor zliczający (definicja)

- ◆ Semafor zliczający (ang. counting) można inicjalizować tylko za pomocą liczby dodatniej
- ◆ Operacja **wait** powoduje zmniejszenie wartości semafora
 - ◆ jeśli wartość semafora stanie się wartością ujemną to dojdzie do zablokowania operacji **wait**
 - ◆ w przeciwnym razie wykonywanie procesu jest kontynuowane
- ◆ Operacja **signal** powoduje zwiększenie wartości semafora
 - ◆ jeśli wartość semafora jest nie większa od zera to proces zablokowany przez operację **wait** zostaje odblokowany

Semafor binarny (definicja)

- ◆ Semafor binarny można inicjalizować za pomocą 0 lub 1
- ◆ Operacja **wait** sprawdza wartość semafora
 - ◆ jeśli wynosi ona 0 to proces wykonujący **wait** zostaje zablokowany
 - ◆ jeśli wynosi ona 1 to wartość semafora zmieniana jest na 0 i proces kontynuuje wykonanie
- ◆ Operacja **signal** sprawdza czy jakieś procesy nie czekają pod semaforem
 - ◆ jeśli tak to proces zablokowany przez **wait** zostaje odblokowany
 - ◆ jeśli żaden proces nie jest zablokowany to wartość semafora zostaje ustwiona na 1

6. Scharakteryzuj monitory.

Monitory

- ◆ Typ monitor składa się z deklaracji zmiennych określających stan obiektu oraz treści procedur realizujących działania na tym obiekcie
- ◆ W danej chwili we wnętrzu monitora może być aktywny tylko jeden proces
- ◆ Procedura wewnętrz monitora może korzystać tylko ze zmiennych zadeklarowanych w nim lokalnie
- ◆ Programista nie musi jawnie kodować barier synchronizacyjnych

```
monitor monitor-name
{
    // shared variable declarations
    procedure P1 (...) { .... }

    ...
    procedure Pn (...) {.....}

    initialization code ( ....) { ... }

}
```

7. Scharakeryzuj zmienne warunkowe

Zmienne typu condition

- ◆ Aby monitor mógł synchronizować potrzebna jest konstrukcja pod nazwą warunek (ang.condition)
- ◆ programista deklaruje jedną lub więcej zmiennych typu *condition*
 - ◆ `condition x, y;`
- ◆ Operacje na zmiennej typu condition:
 - ◆ `x.wait ()` – proces wywołujący tę operację zostaje zawieszony do czasu, aż inny proces wywoła operację `x.signal()`;
 - ◆ `x.signal ()` – wznowia dokładnie jeden z zawieszonych procesów (który wywołał `x.wait ()`) a jeśli nie ma takiego procesu ta operacja ta nie ma żadnych skutków

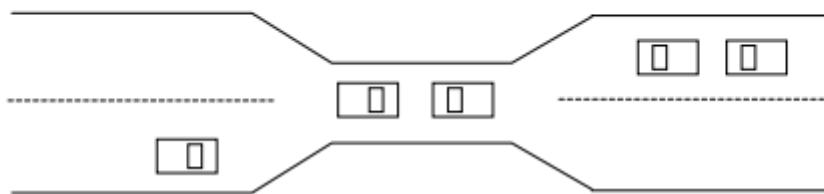
Zmienne typu condition (c.d.)

- ◆ Założmy, że proces P wywołał `x.signal ()` oraz, że proces Q jest zawieszony na warunku `x`
- ◆ dwie możliwości
 - ◆ signal and wait – P albo czeka aż Q opuści monitor albo czeka na inny warunek
 - ◆ signal and continue – Q albo czeka aż P opuści monitor albo czeka na inny warunek
- ◆ kompromis – gdy P wykona operację signal natychmiast opuszcza monitor
- ◆ Concurrent Pascal, Mesa, C# (C-sharp), Java

WYKŁAD 10

1. Podaj przykład kodu programu z impasem i bez impasu.

Przejazd przez most



- ◆ Ruch tylko w jedną stronę
- ◆ Każda z części mostu można uważać za zasób
- ◆ Jeśli nastąpi impas to może zostać rozwiązyany jedynie przez wycofanie się (wywłaszczenie z zasobu i wycofanie procesu)
- ◆ Kilka samochodów może zostać wycofanych
- ◆ Możliwe zagłodzenie (kierowców)

Przykład kodu

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;
void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources();
    up(&resource_2);
    up(&resource_1);
}
void process_B(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources();
    up(&resource_2);
    up(&resource_1);
}
# kod bez impasu
```

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;
void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources();
    up(&resource_2);
    up(&resource_1);
}
void process_B(void) {
    down(&resource_2);
    down(&resource_1);
    use_both_resources();
    up(&resource_1);
    up(&resource_2);
}
# kod z impasem
```

2. Podaj definicję impasu.

Problem impasu

- ◆ Kilka procesów blokuje jakiś zasób i czeka na otrzymanie innego zasobu blokowanego przez jakiś proces z tego zbioru
- ◆ Efekt - może się zdarzyć, że oczekujące procesy nigdy już nie odmienią swego stanu, ponieważ zamawiane przez nie zasoby są przetrzymywane przez inne procesy
- ◆ Nastąpił impas, zakleszczenie, blokada (ang. deadlock)
- ◆ We współczesnych systemach operacyjnych
 - ◆ z jednej strony wzrasta liczba procesów, aplikacji wielowątkowych i zasobów
 - ◆ z drugiej strony mamy coraz więcej długowiecznych serwerów (np. plików i baz danych)
 - ◆ coraz większe prawdopodobieństwo wystąpienia impasu
 - ◆ brak środków do zapobiegania impasom

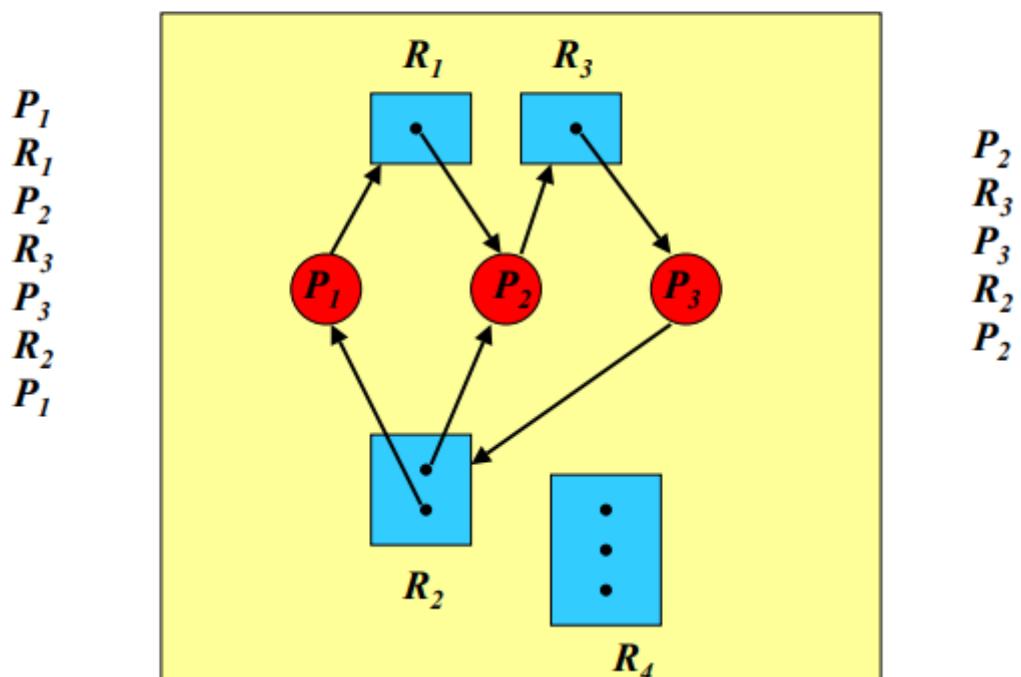
3. Kiedy może wystąpić impas w systemie?

Charakterystka impasów

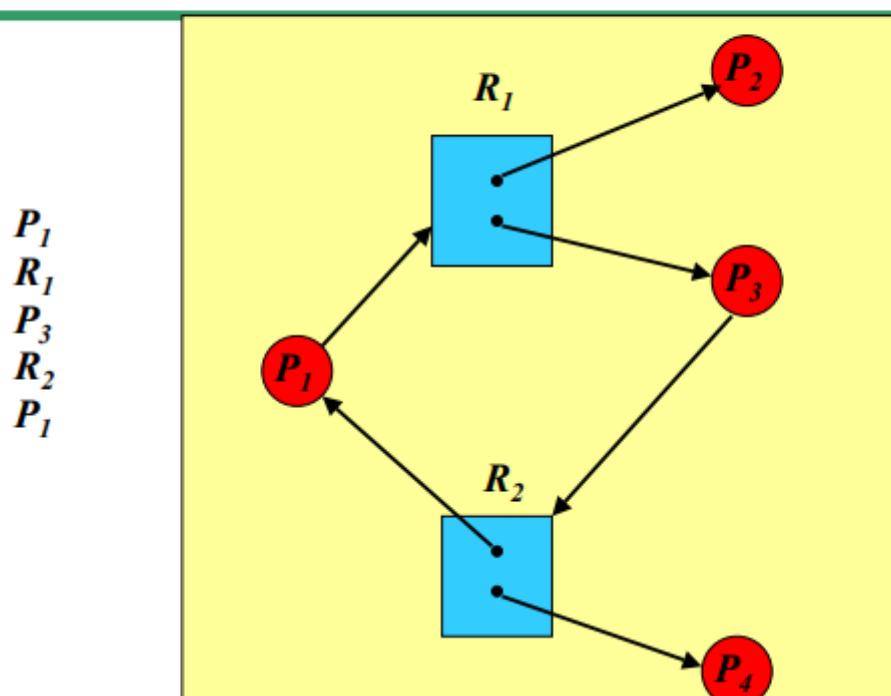
- ◆ Do impasów może dochodzić wtedy, kiedy w systemie zachodzą jednocześnie cztery warunki
 - ◆ wzajemne wykluczanie (ang. mutual exclusion)
 - ◆ przetrzymywanie i oczekiwanie (ang. hold and wait)
 - ◆ brak wywłaszczeń (ang. no preemption)
 - ◆ czekanie cykliczne (ang. circular wait)
- ◆ Warunek konieczny wystąpienia impasu

4. Podaj przykład grafu przydziału zasobów z cyklem i bez impasu.

Graf przydziału zasobów z impasem (2 cykle minimalne)



Graf przydziału zasobów z cyklem i bez impasu



5. Opisz sposób zapobiegania impasom.

Zapobieganie impasom

- ◆ Zapewnić, aby nie mógł wystąpić przynajmniej jeden z czterech warunków koniecznych:
- ◆ Wzajemne wykluczanie - dotyczy jedynie zasobów niepodzielnych (np. drukarek); nie można zaprzeczyć temu warunkowi bowiem niektóre zasoby są z natury niepodzielne

6. Opisz algorytm unikania impasu dla zasobów reprezentowanych jednokrotnie.

Unikanie impasu

- ◆ Wymaga dodatkowej informacji o tym jak będzie następowało zamawianie zasobów
- ◆ W najprostszym i najbardziej użytecznym modelu zakłada się, że system zadeklaruje maksymalną liczbę zasobów każdego typu, którą mógłby potrzebować
- ◆ Stan przydziału zasobów (ang. resource-allocation state) jest określony przez liczbę dostępnych (ang. available) i przydzielonych (ang. allocated) zasobów oraz przez maksymalne zapotrzebowanie procesów
- ◆ Algorytm unikania impasu (ang. deadlock-avoidance) sprawdza dynamicznie stan przydziału zasobów aby zapewnić, że nigdy nie dojdzie do spełnienia warunku czekania cyklicznego

7. Podaj definicję stanu bezpiecznego.

Stan bezpieczny

- ◆ Kiedy proces żąda wolnego zasobu system musi zadecydować czy przydzielanie zasobu pozostawi system w stanie bezpiecznym
- ◆ System jest w stanie bezpiecznym (ang. safe) jeśli istnieje taki porządek przydzielania zasobów każdemu procesowi, który pozwala na uniknięcie impasu
- ◆ System jest w stanie bezpiecznym jeśli istnieje bezpieczny ciąg procesów
- ◆ System jest w stanie zagrożenia (ang. unsafe) jeśli nie jest w stanie bezpiecznym

8. Opisz algorytm bankiera.

Algorytm przydziału zasobów - wiele egzemplarzy zasobu

- ◆ Algorytm grafu przydziału zasobów (dla unikania impasu) jest bezużyteczny w przypadku gdy każdy typ zasobu ma wiele egzemplarzy
- ◆ Algorytm zamawiania (alokacji) zasobów oraz algorytm bankiera (ang. banker's algorithm)
 - ◆ mniej wydajny od schematu grafu przydziału zasobów
- ◆ Każdy proces gdy wchodzi do systemu musi zadeklarować maksymalną liczbę używanych egzemplarzy każdego typu zasobu
 - ◆ liczba ta nie może przekroczyć ogólnej liczby zasobów
- ◆ Jeśli proces zażąda zasobu może się okazać, że musi poczekać
- ◆ Jeśli proces uzyska potrzebne do działania zasoby to musi je zwrócić w skończonym czasie

9. Opisz algorytm zamawiania zasobów.

Algorytm zamawiania zasobów przez proces P_i

Request_i - wektor żądań P_i . Jeśli $Request_i[j] == k$ to P_i chce k egzemplarzy zasobu typu R_j

1. Jeśli $Request_i \leq Need_i$ goto 2. W przeciwnym razie błąd
2. Jeśli $Request_i \leq Available$, goto 3. W przeciwnym razie P_i czeka
3. System próbuje zaalokować zasoby P_i modyfikując stan

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- Jeśli stan safe \Rightarrow alokuj zasoby do P_i .
- Jeśli stan unsafe $\Rightarrow P_i$ musi czekać na realizację i poprzedni stan przydziału zasobów jest odtwarzany

11. Opisz algorytm wykrywania impasu dla zasobów reprezentowanych jednokrotnie.

Użycowanie algorytmu wykrywania impasu

- ◆ Kiedy i jak często należy wywoływać algorytm wykrywania impasu? Odpowiedź zależy od tego
 - ◆ jak często może wystąpić impas
 - ◆ ile procesów popadnie w impas w przypadku jego wystąpienia
- ◆ Jeśli algorytm impasu jest wywoływany losowo to może występować wiele cykli w grafie zasobów i w efekcie identyfikacja procesu, który spowodował impas może być utrudniona

Wykrywanie impasu

- ◆ W systemie, w którym nie stosuje się algorytmu zapobiegania impasom ani ich unikania może wystąpić impas
- ◆ Musz istnieć w systemie algorytmy
 - ◆ wykrywania impasu
 - ◆ likwidowania impasu
- ◆ Zasoby reprezentowane jedno- i wielokrotnie

Typy zasobów reprezentowane jednokrotnie

- ◆ Tworzymy graf oczekiwania (ang. wait-for graph)
 - ◆ węzły grafu są procesami
 - ◆ $P_i \rightarrow P_j$ gdy P_i czeka na P_j
- ◆ Okresowo wykonujemy algorytm szukający cyklu w grafie oczekiwania
- ◆ Rząd algorytmu wykrywania cykli w grafie oczekiwania wynosi n^2 , przy czym n jest liczbą wierzchołków grafu

12. Opisz algorytm wykrywania impasu dla zasobów reprezentowanych wielokrotnie.

Typy zasobów reprezentowane wielokrotnie

- ◆ Metoda grafu oczekiwania jest bezużyteczna do wykrywania impasu gdy każdy typ zasobu ma wiele egzemplarzy
 - ◆ Algorytm wykrywania impasu bada czy istnieje ciąg bezpieczny dla procesów, które trzeba dokończyć
 - ◆ korzysta ze struktur danych algorytmu bankiera
 - ◆ *Available*: wektor długości m oznacza liczbę dostępnych zasobów każdego typu
 - ◆ *Allocation*: macierz $n \times m$ definiuje liczbę zasobów każdego typu aktualnie zaalokowanych do każdego z procesów
 - ◆ *Request*: macierz $n \times m$ określa bieżące zamówienie każdego procesu. Jeśli $\text{Request}[i,j] = k$, to proces P_i zamawia dodatkowo k egzemplarzy zasobu typu R_j .
-

13. Opisz sposoby likwidowania impasu.

Likwidowanie impasu

- ◆ Zaniechanie (abort) wszystkich zakleszczonych procesów
- ◆ Usuwanie procesów (wywłaszczanie) pojedynczo, aż do wyeliminowania cyklu impasu
- ◆ Na kolejność zaniechania (abort) ma wpływ
 - ◆ priorytet procesu
 - ◆ długość czasu wykonania i czas pozostały do zakończenia
 - ◆ ilość zasobów danego typu użytkowanych przez proces
 - ◆ zasoby i typy potrzebne do zakończenia
 - ◆ ilość procesów do przerwania
 - ◆ interakcyjność lub wsadowość

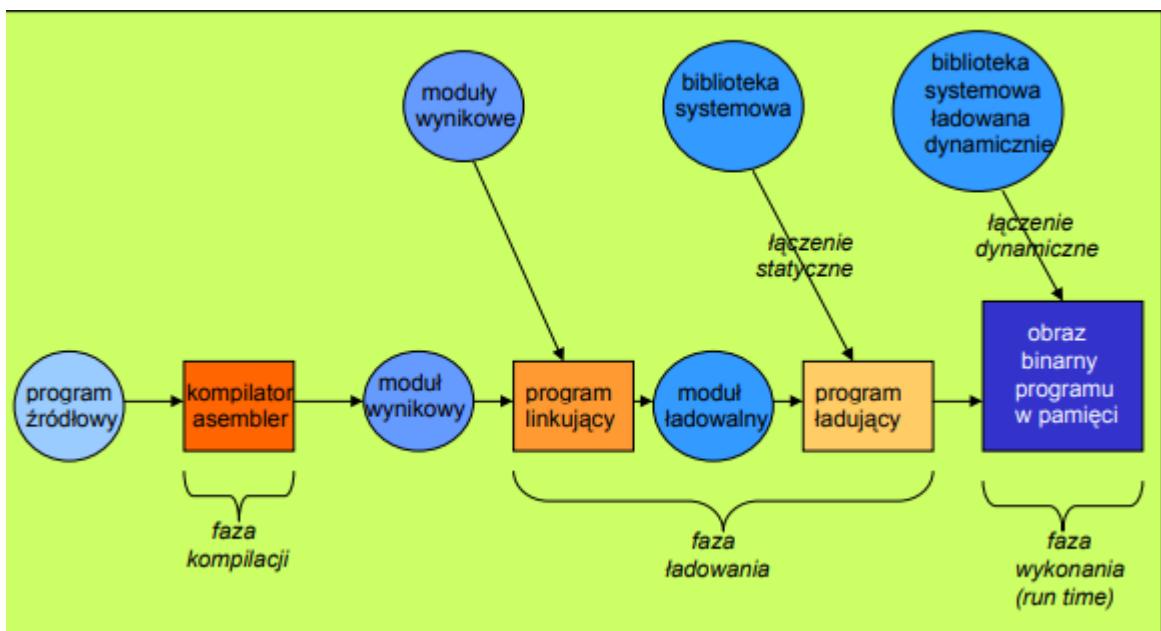
- ◆ Usuwanie procesów (wywłaszczanie)
- ◆ Wybór ofiary - minimalizowanie kosztów
- ◆ Wycofanie (ang. rollback) - wycofanie procesu do bezpiecznego stanu, od którego można go będzie wznowić
- ◆ Głodzenie - ten sam proces może być zawsze ofiarą, podobnie jak i ten sam proces może być ciągle wycofywany. Trzeba zadbać aby proces mógł być delegowany do roli ofiary tylko skońzoną liczbę razy
- ◆ Podejście mieszane
- ◆ Podstawowe strategie
 - ◆ zapobieganie (ang. prevention)
 - ◆ unikanie (ang. avoidance)
 - ◆ wykrywanie (ang. detection)
- ◆ Strategie stosowane osobno nie nadają się do rozwiązywania problemów z przydzielaniem zasobów
- ◆ Podział wszystkich zasobów na hierarchicznie uporządkowane klasy
- ◆ W obrębie klas dobieramy najodpowiedniejsze techniki pokonywania impasu

WYKŁAD 11

1. Wymień etapy przetwarzania programu użytkownika.

Podstawy

- ◆ Program musi być wprowadzony do pamięci operacyjnej i przydzielony odpowiedniemu procesowi
- ◆ *Kolejka wejściowa* – (ang. input queue) zbiór procesów czekających na dysku na wprowadzenie do pamięci w celu wykonania
- ◆ Program użytkownika, zanim zostanie wykonany, przechodzi przez kilka faz



2. Opisz działanie MMU.

Jednostka zarządzania pamięcią (MMU)

- ◆ Urządzenie sprzętowe dokonujące odwzorowania adresów wirtualnych na fizyczne nazywa się MMU (ang. memory-management unit)
- ◆ Istnieją wiele sposobów odwzorowywania adresów
- ◆ Proste MMU
 - ◆ do każdego adresu wytwarzanego przez proces użytkownika dodawana jest wartość rejestru przemieszczenia (ang. relocation register) w chwili odwoływania się do pamięci
- ◆ MSDOS używa czterech rejestrów przemieszczenia (architektura Intel 80x86)
- ◆ program użytkownika nigdy nie ma do czynienia z *rzeczywistym* adresem; program ten działa na adresach *logicznych*

3. Opisz różnicę między adresacją logiczną i fizyczną.

Logiczna i fizyczna przestrzeń adresowa

- ◆ Adres wytworzony przez CPU nazywamy adresem logicznym (ang. logical address)
- ◆ Adres oglądany przez jednostkę pamięci nazywamy adresem fizycznym (ang. physical address)
 - ◆ adres logiczny i fizyczny jest taki sam podczas komplikacji i ładowania;
 - ◆ adresy logiczny (wirtualny) i fizyczny adres różnią się podczas wykonania
- ◆ Zbiór wszystkich adresów logicznych generowanych przez program nazywamy logiczną przestrzenią adresową (ang. logical address space)
- ◆ Zbiór wszystkich adresów fizycznych odpowiadających wygenerowanym przez program adresom logicznym nazywamy fizyczną przestrzenią adresową (ang. physical address space)

4. Opisz zasadę konsolidacji dynamicznej.

Konsolidacja dynamiczna

- ◆ Konsolidację (łączenie) opóźnia się do czasu wykonania
- ◆ W obrazie binarnym, w miejscu odwołania bibliotecznego znajduje się tylko *namiastka* (ang. stub) procedury będąca małym fragmentem kodu, wskazującym jak odnaleźć odpowiedni, rezydujący w pamięci podprogram biblioteczny lub jak załadować bibliotekę jeśli podprogramu nie ma w pamięci
- ◆ Namiastka wprowadza na swoje miejsce adres podprogramu i go wykonuje
- ◆ System operacyjny sprawdza czy podprogram jest w pamięci a jeśli go tam nie ma to go sprowadza
- ◆ Konsolidacja dynamiczna (ang. dynamic linking) wymaga wspomagania ze strony systemu operacyjnego, niektóre systemy realizują jedynie konsolidację statyczną (ang. static linking)

5. Wymień sposoby jak na podstawie listy wolnych dziur spełnić zamówienie procesu na pamięć.

Problem dynamicznego przydziału pamięci

Jak na podstawie listy wolnych dziur spełnić zamówienie na obszar o rozmiarze n?

- ◆ **Pierwsze dopasowanie:** (ang. first-fit) - przydziel *pierwszą* dziurę o wystarczającej wielkości
- ◆ **Najlepsze dopasowanie:** (ang. best-fit) - przydziel *najmniejszą* z dostatecznie dużych dziur; przejrzyj całą listę, chyba że jest uporządkowana według rozmiarów. Strategia ta zapewnia najmniejsze pozostałości po przydziale
- ◆ **Najgorsze dopasowanie:** (ang. worst-fit) - przydziel *największą* dziurę; należy również przeszukać całą listę. Strategia ta pozostawia po przydziale największą dziurę, która może okazać się bardziej użyteczna niż pozostałość wynikającą z podejścia polegającego na przydziale najlepiej pasującej dziury

6. Wyjaśnij różnicę między fragmentacją zewnętrzną i wewnętrzną.

- ◆ Fragmentacja zewnętrzna (ang. external fragmentation) – suma wolnych obszarów w pamięci wystarcza na spełnienie zamówienia ale nie tworzą one ciągłego obszaru
 - ◆ IBM OS/360 MVT
- ◆ Fragmentacja wewnętrzna (ang. internal fragmentation) – zaalokowana pamięć jest nieznacznie większa od żądania alokacji pamięci; różnica ta stanowi bezużyteczny kawałek pamięci wewnętrz przydzielonego obszaru
 - ◆ IBM OS/360 MFT

7. Opisz paging (pol. stronicowanie).

Stronicowanie (ang. Paging)

- ◆ Stronicowanie to schemat zarządzania pamięcią zezwalający na nieciągłość fizycznej przestrzeni adresowej
- ◆ Logiczna przestrzeń adresowa procesu może być odwzorowana w sposób *nieciągły*
 - ◆ tj. procesowi można przydеляć dowolne dostępne miejsca w pamięci fizycznej
- ◆ Pamięć fizyczną dzieli się na bloki stałej długości zwane ramkami (ang. frames) (rozmiar jest potągą 2, między 512B a 16MB)
- ◆ Pamięć logiczną dzieli się na bloki tego samego rozmiaru zwane stronami (ang. pages)
- ◆ Pamiętana jest lista wolnych ramek

8. Podaj schemat translacji adresu przy stronicowaniu.

Schemat Translacji Adresu

- ◆ Stronicowanie wymaga wsparcia sprzętowego
- ◆ Adres wygenerowany przez CPU jest dzielony na dwie części:
 - ◆ *Numer strony (ang. Page number) (p)* – używany jako indeks w tablicy stron zawierającej adresy bazowe wszystkich stron w pamięci fizycznej
 - ◆ *Odległość na stronie (ang. Page offset) (d)* – w połączeniu z adresem bazowym definiuje fizyczny adres pamięci posyłany do jednostki pamięci

9. Opisz wykorzystanie TLB w stronicowaniu.

- ◆ Zasady zastępowania – zastąp najdawniej używany element (LRU, niezbędne wsparcie sprzętowe)
- ◆ Niektóre z pozycji TLB (np. dotyczące kodu jądra systemu) mogą być nieusuwalne i wtedy nazywamy je przypiętymi (ang. wired down)
- ◆ ASID (ang. address space identifiers) – identyfikatory w TLB jednoznacznie określające związany z daną pozycją proces
 - ◆ jeśli ASID w TLB i ASID strony nie są zgodne to chybienie TLB
 - ◆ ochrona procesów
- ◆ Jeśli bufor TLB nie udostępnia odrębnych identyfikatorów ASID to przy wyborze nowej tablicy stron (np. przełączenie kontekstu) bufor TLB musi zostać wykasowany (ang. erased, flushed)

10. Podaj wzór na efektywny czas dostępu do pamięci.

Efektywny czas dostępu do pamięci (EAT)

- ◆ Przeglągnięcie bufora TLB wynosi ε jednostek czasu
- ◆ Niech cykl pamięci wynosi λ jednostek czasu
- ◆ Współczynnik trafień (ang. hit ratio) – procent numerów stron odnajdowywanych w buforze TLB
- ◆ Niech współczynnik trafień wynosi α
- ◆ Efektywny czas dostępu do pamięci (ang. effective memory-access time (EAT))

$$\text{EAT} = (\lambda + \varepsilon)\alpha + (2\lambda + \varepsilon)(1 - \alpha) = 2\lambda + \varepsilon - \lambda\alpha$$

Np. $\varepsilon=20\text{ns}$, $\lambda=100\text{ns}$, $\alpha=0.80$; $\text{EAT}= 120*0.8 + 220*0.2 = 140\text{ns}$

11. Opisz sposób ochrony pamięci przy stronicowaniu.

Ochrona pamięci

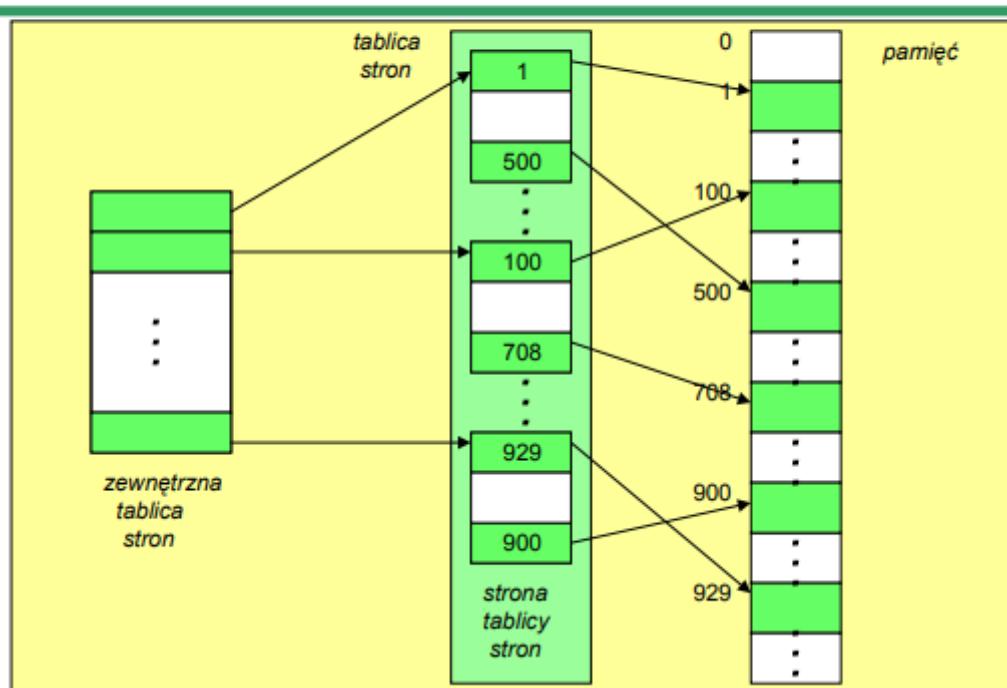
- ◆ Ochrona pamięci jest zaimplementowana za pomocą bitów ochrony przypisanych każdej ramce
- ◆ *Bit poprawności (ang. Valid-invalid bit)* - każdy wpis w tablicy stron zostaje uzupełniony o dodatkowy bit:
 - ◆ “poprawny” (“valid”) oznacza, że strona, z którą jest on związany, znajduje się w logicznej przestrzeni adresowej procesu, a więc jest ona dozwolona (ang. legal page)
 - ◆ “niepoprawny” (“invalid”) oznacza, że strona nie należy do logicznej przestrzeni adresowej procesu tzn. niepoprawne odwołanie do strony

12. Podaj schemat translacji adresu przy stronicowaniu dwupoziomowym.

Przykład dwupoziomowego stronicowania (1)

- ◆ Logiczna przestrzeń adresowa na 32-bitowej maszynie z rozmiarem strony 4K jest podzielona na:
 - ◆ 20-bitowy numer strony
 - ◆ 12-bitowa odległość na stronie
- ◆ Ponieważ dzielimy tablicę stron na strony, numer strony podlega dalszemu podziałowi na:
 - ◆ 10-bitowy numer strony
 - ◆ 10-bitowa odległość na stronie
- ◆ Schemat wstępnie odwzorowywanej tablicy stron (ang. forward-mapped page table)
 - ◆ zastosowany w architekturze Pentium II

Schemat dwupoziomowej tablicy stron



13. Podaj schemat translacji adresu przy haszowanej tablicy stron.

Haszowane tablice stron

- ◆ Co robić jeśli przestrzeń adresowa > 32 bitów?
- ◆ Numer strony pamięci wirtualnej jest odwzorowany (ang. hashed) przy pomocy funkcji haszującej (mieszającej) na pozycje w tablicy (ang. hashed page table)
- ◆ Wszystkie strony wirtualne którym odpowiada ta sama pozycja w tablicy (kolizja) zostają umieszczone na jednej liście (metoda łańcuchowa)
- ◆ Elementy listy powiązanej: numer strony wirtualnej (p), numer ramki pamięci (r), wskaźnik do następnego elementu listy

14. Opisz haszowanie liniowe i łańcuchowe.

Nie wiem ale haszowanie zaczyna się od 53 strony
Ja też nie wiem. To jest lekko pojęte.

15. Opisz algorytm wyszukiwania w haszowanej tablicy stron.

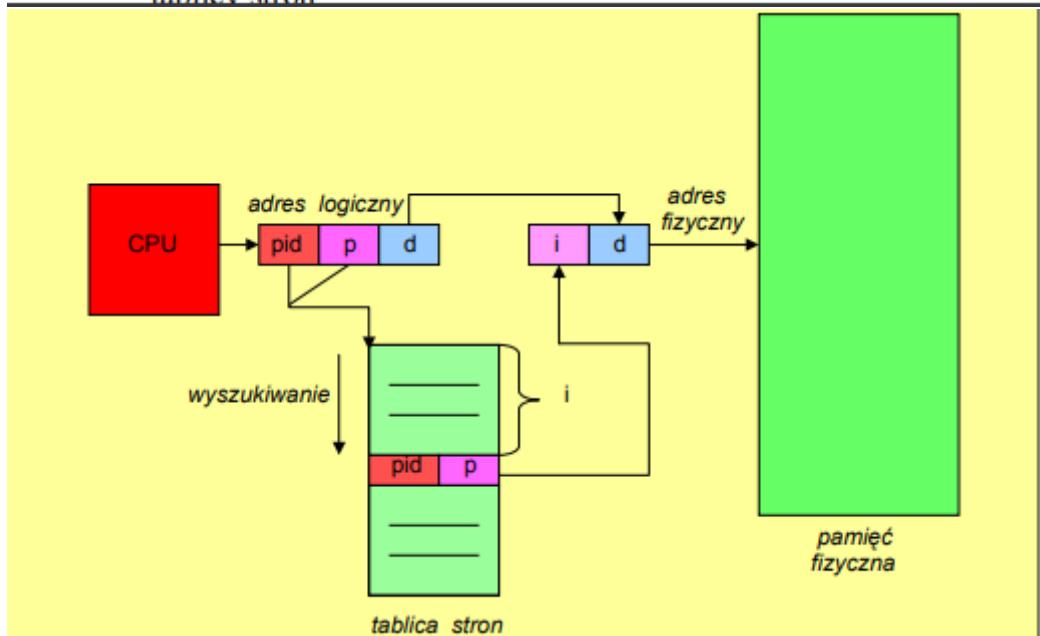
Algorytm wyszukiwania w haszowanej tablicy stron

- ◆ Numer strony wirtualnej jest odwzorowany przy pomocy funkcji mieszającej na element listy powiązanej w tablicy stron: (q, s, *), (p, r, *),....
- ◆ Numer strony jest porównywany z pierwszym elementem (q) w tablicy
 - ◆ jeśli numer strony jest zgodny to używa się odpowiadającej mu ramki (s)
 - ◆ w przeciwnym razie wybieramy następny (*) adres, t.j. (p, r, *)
 - ◆ itd.
- ◆ Tablice stron grupowanych (ang. clustered page tables) dla 64b przestrzni adresowych - każdy wpis w tablicy odnosi się do grupy stron (np. 16)

16. Podaj schemat translacji adresu przy odwróconej tablicy stron.

Odwrócona tablica stron

- ◆ Odwrócona tablica stron (ang. inverted page table) ma po jednej pozycji dla każdej rzeczywistej strony pamięci (ramki)
- ◆ Każda pozycja zawiera adres wirtualny strony przechowywanej w ramce rzeczywistej pamięci oraz informacje o procesie posiadającym stronę (np. PID)
 - ◆ w systemie jest jedna tablica ale wiele przestrzeni adresowych (potrzebny jest ASID)
 - ◆ IBM RT, 64b architektury: UltraSPARC, PowerPC
- ◆ Zmniejsza się rozmiar pamięci potrzebnej do pamiętania wszystkich tablic stron, jednak zwiększa się czas potrzebny do przeszukania tablicy przy odwołaniu do strony
 - ◆ aby to złagodzić wprowadza się tablicę z haszowaniem (ang. hash table) co ogranicza szukanie do jednego lub najwyżej kilku wpisów w tablicy stron



17. Opisz schemat stron dzielonych.

Strony dzielone

◆ Dzielenie kodu

- ◆ jedna kopia kodu nie modyfikującego samego siebie tj. kodu czystego (ang. pure code) lub wznowialnego (ang. reentrant) jest dzielona pomiędzy procesy (np. editory tekstu, kompilatory, system okien)
- ◆ kod dzielony musi być widziany pod tą samą lokacją w logicznej przestrzeni adresowej dla wszystkich procesów

◆ Kod prywatny i dane

- ◆ każdy proces ma własną kopię kodu i danych
- ◆ strony dla prywatnego kodu i danych mogą się pojawić w dowolnym miejscu logicznej przestrzeni adresowej

18. Podaj schemat translacji adresu przy segmentacji.

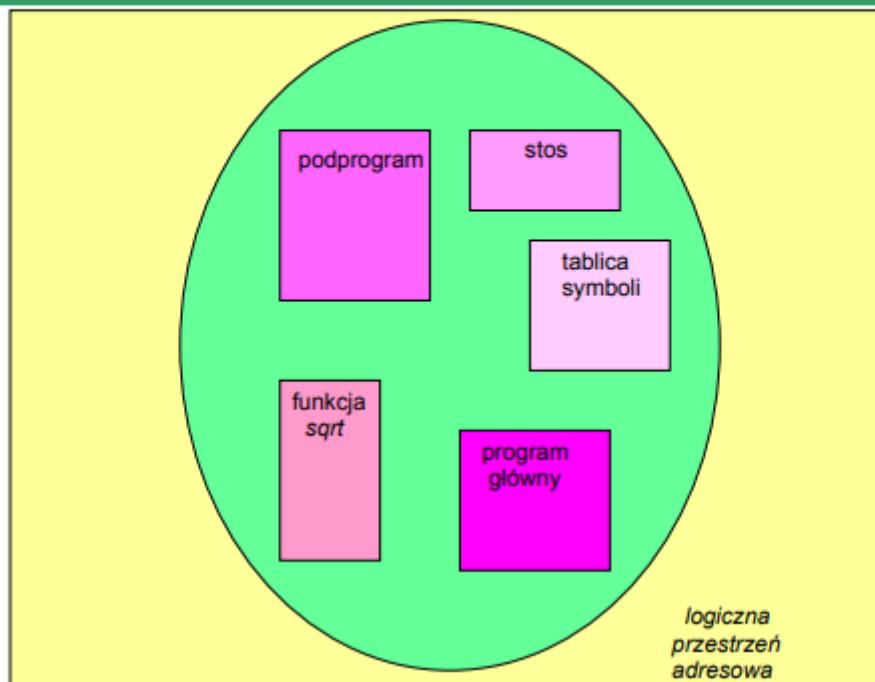
Segmentacja

◆ Segmentacja (ang. segmentation) to schemat zarządzania pamięcią który urzeczywistnia sposób widzenia pamięci przez użytkownika

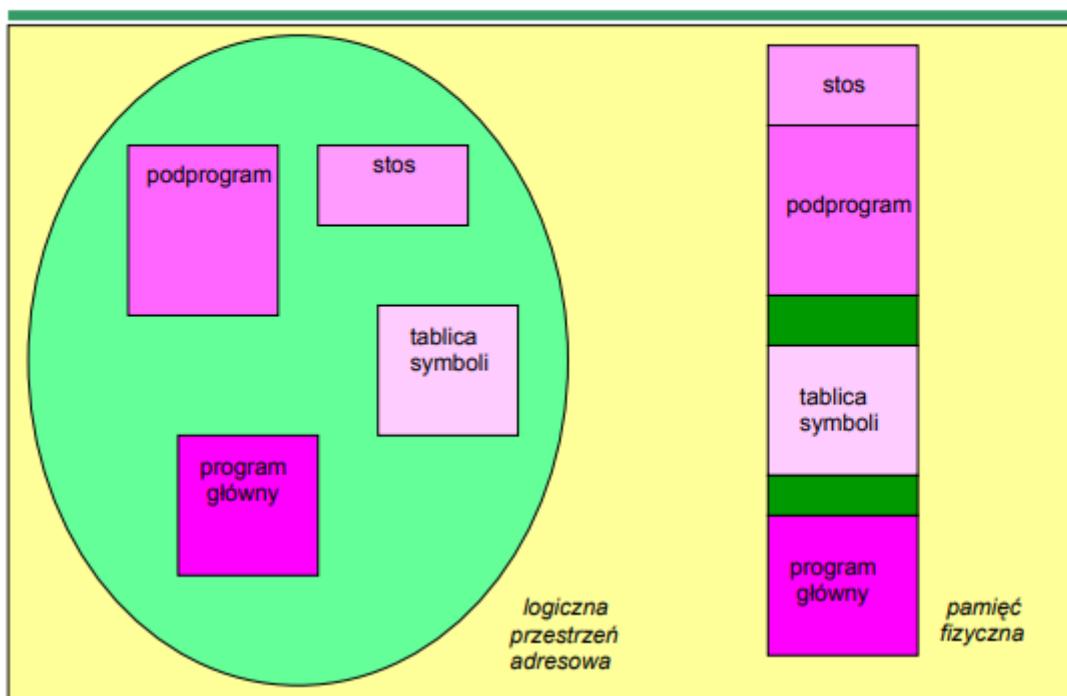
◆ Program jest zbiorem segmentów czyli jednostek logicznych takich jak:

program główny,
procedura,
funkcja,
zmienne lokalne, zmienne globalne,
common block,
stos, tablica symboli, arrays

Program z punktu widzenia użytkownika



Filozofia segmentacji



19. Opisz polecenia do ustalania, monitorowania wielkości pamięci i strony w systemie Solaris i Linux

Ustalanie wielkości pamięci

- ◆ pagesize
 - ◆ Linux: getconf PAGESIZE
- ◆ AIX
 - ◆ lsattr -HE -l sys0 -a realmem
- ◆ FreeBSD
 - ◆ grep memory /var/run/dmesg.boot
- ◆ HP-UX
 - ◆ dmesg | grep Phys
- ◆ Linux
 - ◆ free
- ◆ Solaris
 - ◆ dmesg | grep mem
- ◆ True64
 - ◆ vmstat | grep '^Total'

Monitorowanie wykorzystania pamięci

- ◆ ps -ayl (Solaris)
 - ◆ używana pamięć fizyczna (wirtualna) w kolumnie RSS (SZ)
 - ◆ udział w zużyciu pamięci w kolumnie %MEM
- ◆ ps vx (Linux)
 - ◆ MAJFL – liczba braków stron
- ◆ vmstat
 - ◆ memory - swap lub free
 - ◆ page
 - ◆ re - liczba przywołanych stron pamięci
 - ◆ pi - liczba wczytanych stron
 - ◆ fr - liczba stron zwolnionych
 - ◆ po - liczba wypisanych stron
- ◆ top
- ◆ KDE System Guard (ksysguard)

WYKŁAD 12

1. Wymień zalety pamięci wirtualnej.

Podstawy (c.d.)

- ◆ Pamięć wirtualna (ang. virtual memory) – pozwala na odseparowanie pamięci logicznej użytkownika od pamięci fizycznej
 - ◆ można jedynie część programu załadować do pamięci w celu wykonania
 - ◆ logiczna przestrzeń adresowa procesu może dlatego być znacznie większa niż fizyczna przestrzeń adresowa
 - ◆ potrzeba wymiany stron między dyskiem a pamięcią
- ◆ Pamięć wirtualną można zaimplementować jako:
 - ◆ stronicowanie na żądanie (ang. demand paging)
 - ◆ segmentacja na żądanie (ang. demand segmentation)
 - ◆ np. Burroughs, IBM OS/2

2. Wymień sposoby implementacji pamięci wirtualnej.

Podstawy (c.d.)

- ◆ Pamięć wirtualna (ang. virtual memory) – pozwala na odseparowanie pamięci logicznej użytkownika od pamięci fizycznej
 - ◆ można jedynie część programu załadować do pamięci w celu wykonania
 - ◆ logiczna przestrzeń adresowa procesu może dlatego być znacznie większa niż fizyczna przestrzeń adresowa
 - ◆ potrzeba wymiany stron między dyskiem a pamięcią
- ◆ Pamięć wirtualną można zaimplementować jako:
 - ◆ stronicowanie na żądanie (ang. demand paging)
 - ◆ segmentacja na żądanie (ang. demand segmentation)
 - ◆ np. Burroughs, IBM OS/2

3. Opisz stronicowanie na żądanie.

Stronicowanie na żądanie

- ◆ System stronicowania na żądanie jest podobny do stronicowania z wymianą
- ◆ Procedura leniwej wymiany (ang. lazy swapper)
 - ◆ nigdy nie dokonuje się wymiany strony w pamięci jeśli nie jest to konieczne
 - ◆ mniej operacji we/wy
 - ◆ mniej pamięci
 - ◆ szybsza reakcja
 - ◆ więcej użytkowników
- ◆ Jeśli strona jest potrzebna ⇒ odwołaj się
 - ◆ niepoprawne odwołanie ⇒ abort
 - ◆ brak strony w pamięci ⇒ sprowadź stronę do pamięci
- ◆ Zgodność z Zasadą Lokalności Odniesień
- ◆ Proces jest traktowany jako ciąg stron
- ◆ Procedura wymiany dotyczy całego procesu
- ◆ Procedura stronicująca (ang. pager) dotyczy poszczególnych stron procesu
- ◆ Gdy proces ma zostać wprowadzony do pamięci, wówczas procedura stronicująca zgaduje jakie strony będą w użyciu przed ponownym załadowaniem na dysk (ang. swap space)
- ◆ Nigdy nie dokonuje się wymiany całego procesu dlatego używamy określenia stronicowanie (ang. page) zamiast wymiana (ang. swap)

4. Opisz procedurę obsługi braku strony.

Procedura obsługi braku strony

- ◆ 1. System operacyjny sprawdza wewnętrzną tablicę w PCB oraz decyduje, że:
 - ◆ 2. jeśli odwołanie niedozwolone - kończy proces (abort)
 - ◆ 2. jeśli odwołanie dozwolone tylko zabrało stronę w pamięci to sprowadza tę stronę
- ◆ 3. System znajduje wolną ramkę na liście wolnych ramek
- ◆ 4. System wczytuje (swap in) stronę z dysku do wolnej ramki
- ◆ 5. System wstawia bit 1 (lub v) w tablicy stron
- ◆ 6. System wznowia wykonanie przerwanego rozkazu

5. Podaj wzór na obliczenie sprawności stronicowania.

Sprawność stronicowania na żądanie

- ◆ Prawdopodobieństwo braku strony p
 - ◆ jeśli $p=0$ to brak braku stron
 - ◆ jeśli $p=1$ to każde odwołanie generuje brak strony
- ◆ Efektywny czas dostępu (EAT)
 - ◆ $EAT = (1-p) * (\text{czas dostępu}) + p * (\text{czas obsługi strony})$
 - ◆ czas dostępu - do pamięci (10 do 200 ns)
 - ◆ czas obsługi strony
 - ◆ obsługa przerwania wywołanego brakiem strony (1 do 100 μs)
 - ◆ przełączenie strony $\cong 8\text{ms}$
 - ◆ wznowienie procesu (1 do 100 μs)

6. Opisz procedurę zastępowania stron.

Zastępowanie stron

- ◆ Prz. Mamy 40 ramek, 4 procesy 10 stronicowe, każdy zużywa w danym przebiegu 5 stron
 - ◆ możemy zatem wykonać osiem takich procesów
 - ◆ jeden z procesów przy innym przebiegu potrzebuje dodatkowej strony
- ◆ Ochrona przed nadprzydziałem (ang. over-allocation) pamięci przez dodanie do obsługi braku strony możliwości zastąpienia strony (ang. page replacement).
- ◆ Procedura obsługi braków stron
 - ◆ 1. Zlokalizowanie potrzebnej strony na dysku
 - ◆ 2. Odnalezienie wolnej ramki
 - ◆ jeśli ramka istnieje - zostaje użyta
 - ◆ w przeciwnym razie typowanie ramki ofiary (ang. victim frame)
 - ◆ ramka ofiara zapisana na dysk; zmień tablicę stron i tablicę ramek
 - ◆ 3. Wczytanie potrzebnej strony; zmień tablice stron i ramek
 - ◆ 4. Wznowienie procesu

7. Opisz algorytm zastępowania stron FIFO.

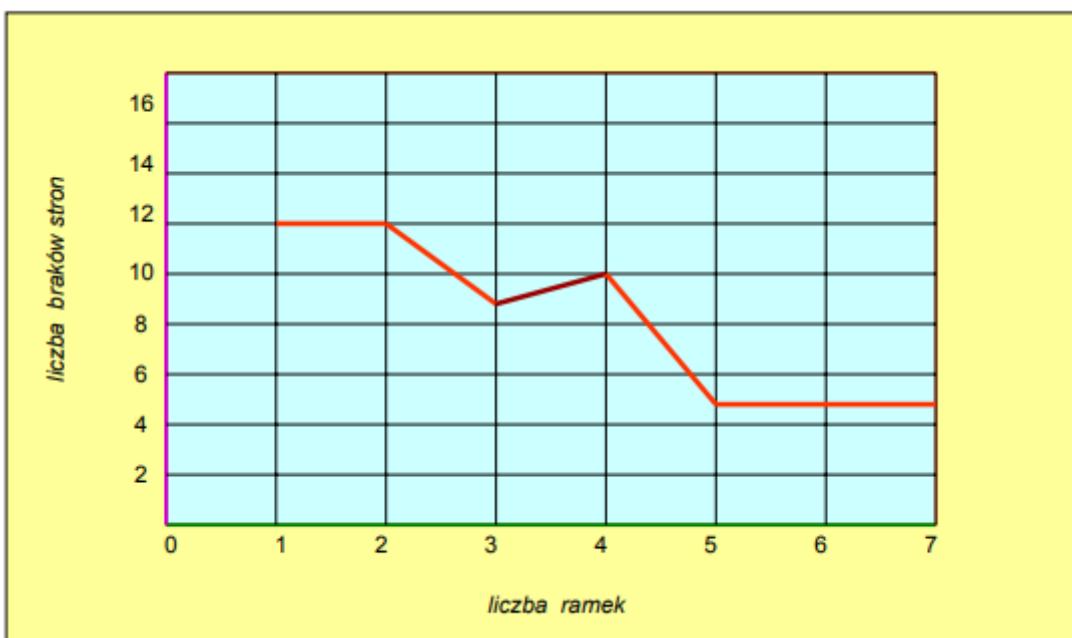
Algorytm zastępowania FIFO

- ◆ Algorytm FIFO (ang. First-In First-Out) stowarzysza z każdą ze stron czas kiedy została ona sprowadzona do pamięci
- ◆ Jeśli trzeba zastąpić stronę to zastępowana jest najstarsza ze stron
- ◆ Implementuje się za pomocą kolejek FIFO (IPC System V, POSIX)
- ◆ Dla ciągu odniesień z trzema ramkami mamy 15 braków stron (w tym trzy początkowe braki stron)

8. Skonstruuj przykład ilustrujący anomalię Belady'ego.

Anomalia Belady'ego

- ◆ Anomalia Belady'ego (ang. Belady's anomaly) odzwierciedla fakt, że w niektórych algorytmach zastępowania stron współczynnik braków stron może wzrastać ze wzrostem wolnych ramek (mimo, że intuicja zdaje się sugerować, że zwiększenie pamięci procesu powinno polepszyć jego działanie)
- ◆ Poszukiwanie optymalnego algorytmu zastępowania stron, który cechuje najniższy współczynnik braków stron
 - ◆ FIFO jest prosty w implementacji ale daje słabe wyniki



9. Opisz algorytm zastępowania stron LRU.

Algorytm LRU

- ◆ Zastąp tę stronę, która najdawniej była użyta (ang. Least Recently Used)
- ◆ Przykład 3 ramek
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

1	7	2	2	4	4	4	0	1	1	1		1
2	0	0	0	0	3	3	3	0	0		2	
3	1	1	3	3	2	2	2	2	2	7	3	

12 braków stron

- ◆ Nie jest dotknięty anomalią Belady'ego
- ◆ Odwracalność (C. G. J. Jacobi – „Zawsze odwracaj”)

10. Opisz sposoby implementacji algorytmu LRU.

Algorytm LRU (c.d.)

- ◆ Dwie implementacje

- ◆ Liczniki - do każdej pozycji w tablicy stronłączamy rejestr czasu użycia, do procesora zaś dodajemy zegar logiczny lub licznik. Wskazania ‘zegara’ są zwiększane wraz z każdym odniesieniem do pamięci. Ilekroć występuje odniesienie do pamięci, tylekroć zawartość rejestru zegara jest kopowana do pola czasu użycia należącego do danej strony w tablicy stron
- ◆ Stos - przy każdym odwołaniu do strony jej numer wyjmuje się ze stosu i umieszcza na szczycie - najlepsza implementacja to dwukierunkowa lista ze wskaźnikami do czoła i do końca
 - ◆ najwyżej 6 zmian wskaźników
 - ◆ nie jest potrzebne przeszukiwanie listy

11. Opisz algorytm zastępowania stron OPT.

Algorytm optymalny

◆ Zastąp tę stronę, która najdłużej nie będzie używana;
nazywany OPT lub MIN

◆ Przykład 3 ramek

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

1	7	2	2	2	2	2	7	
2	0	0	0	4	0	0	0	9 braków stron
3	1	1	3	3	3	1	1	

◆ Nie ma anomalii Belady'ego

◆ Jeśli dwie ramki będą jednakowo długo nieużywane to stosujemy FIFO

◆ Bardzo trudny do realizacji bo wymaga wiedzy o przyszłej postaci ciągu odniesień (podobnie jak przy planowaniu procesora metodą SJF)

◆ Używany głównie w studiach porównawczych

◆ wiedza o tym, że jakiś algorytm odbiega od optymalnego o 12,3% a średnio jest od niego gorszy o 4,7% może okazać się cenną

◆ inna sprawa to wartość wartości średniej

12. Scharakteryzuj algorytmy stosowe.

Algorytmy stosowe

- ◆ Def. Algorytm stosowy (ang. stack algorithm) to taki algorytm dla którego zbiór stron w pamięci w przypadku n ramek jest podzbiorem zbioru stron w pamięci w przypadku n+1 ramek
- ◆ Przykład: LRU
- ◆ Własność: klasa algorytmów stosowych nie jest dotknięta anomalią Belady'ego
- ◆ Implementacja LRU wymaga wsparcia sprzętowego: uaktualnianie pól zegara lub stosu musi być dokonywane przy każdym odniesieniu do pamięci
- ◆ Możemy nie mieć takiego sprzętu

13. Opisz algorytm dodatkowych bitów odniesienia.

Algorytmy przybliżające LRU

- ◆ Bit odniesienia (ang. reference bit, use bit)
 - ◆ z każdą stroną stwarzamy na początku bit 0
 - ◆ czytanie lub pisanie na stronie ustawia bit na 1
 - ◆ zastąp stronę stronę jeśli ma bit 0
 - ◆ nie można poznać porządku użycia stron
- ◆ Algorytm dodatkowych bitów odniesienia
 - ◆ każda strona ma 8 bitowy rejestr w pamięci głównej
 - ◆ cyklicznie (co 100ms) przerwanie zegarowe powoduje wywołanie procedury która powoduje wprowadzenie bitu odniesienia na najbardziej znaczącą pozycję rejestru oraz przesunięcie pozostałych w prawo o 1 bit
 - ◆ 00000000 – strona nieużywana przez osiem cykli
 - ◆ 11111111 - strona używana co najmniej jeden raz w każdym cyklu
 - ◆ interpretujemy rejesty jako liczby bez znaku, tzn. strona najdawniej użyta ma najmniejszą zawartość rejestru

14. Opisz zegarowy algorytm drugiej szansy (CLOCK).

Algorytm zegarowy

- ◆ Algorytm drugiej szansy (ang. second chance) - wykorzystuje strategię zegarową (ang. clock policy)
 - ◆ jeśli strona jest po raz pierwszy ładowana do pamięci odpowiadający jej bit odniesienia w ramce jest ustawiany na 1 a wskaźnik przesuwany cyklicznie na następną ramkę
 - ◆ jeśli nastąpiło odwołanie do strony to bit odniesienia = 1
 - ◆ jeśli brak jest strony to
 - ◆ następuje cykliczne przeszukanie ramek kandydatek do zastąpienia podczas którego każdy bit odniesienia = 1 jest zerowany
 - ◆ zostaje zastąpiona pierwsza strona z bitem odniesienia = 0
 - ◆ wskaźnik zostaje ustawiony na następną ramkę
 - ◆ Algorytm zegarowy lub CLOCK
 - ◆ Implementacja przy pomocy kolejki cyklicznej
-

16. Opisz ulepszony algorytm drugiej szansy.

Ulepszony algorytm drugiej szansy

- ◆ (x,y) - x - bit odniesienia, y - bit modyfikacji przypisany jest (w CPU) do każdej ramki
- ◆ 4 klasy (od najniższej)
 - ◆ $(0,0)$ - nie używana ostatnio (ang. not accessed recently) i nie zmieniana (ang. not modified): najlepsza ofiara
 - ◆ $(0,1)$ - nie używana ostatnio ale zmieniona: gorsza ofiara bo wymaga zapisu na dysk choć zgodnie z zasadą lokalności pewnie nie zostanie użyta
 - ◆ $(1,0)$ - używana ostatnio i czysta: jeszcze gorsza ofiara bo zgodnie z zasadą lokalności może być wkrótce użyta
 - ◆ $(1,1)$ - używana ostatnio i zmieniana - najgorsza ofiara, prawdopodobnie (zgodnie z zasadą lokalności) będzie zaraz użyta a ponadto wymaga zapisu na dysku

-
- ◆ 1. Skanowanie bufora ramek od pozycji wskaźnika
 - ◆ bity nie są zmieniane; do wymiany pierwsza ramka $(0,0)$
 - ◆ 2. Jeśli w kroku 1 nie znaleziono ofiary to następuje skanowanie bufora w celu znalezienia ramki $(0,1)$
 - ◆ do wymiany pierwsza znaleziona; bity odniesienia są w trakcie zerowane
 - ◆ 3. Jeśli w kroku 2 nie znaleziono ofiary to wskaźnik znajduje się w położeniu początkowym i wszystkie bity odniesienia są wyzerowane
 - ◆ powtarzamy krok 1 a w przypadku nie znalezienia ofiary krok 2
 - ◆ znajdujemy ramkę ofiarę; wymieniamy stronę; ustawiamy wskaźnik na następną ramkę w buforze
 - ◆ Ulepszony algorytm drugiej szansy w pierwszej kolejności zastępuje stronę która nie była zmodyfikowana
 - ◆ Algorytm ten stosowany jest przez zarządcę pamięci wirtualnej komputera Macintosh

17. Opisz schematy przydziału ramek.

Przydział ramek

- ◆ Każdy proces wymaga minimalnej liczby ramek
- ◆ Przykład: IBM 370 – 6 ramek potrzebnych do wykonania rozkazu typu SS MVC:
 - ◆ rozkaz długości 6 bajtów może zajmować 2 strony
 - ◆ dane wejściowe (256 znaków) mogą zajmować 2 strony
 - ◆ obszar na który przesyłamy może zajmować 2 strony
 - ◆ gdy MVC jest argumentem rozkazu EXECUTE jeszcze 2!
- ◆ Dwa główne schematy przydziału
 - ◆ przydział równy (ang. equal allocation)
 - ◆ przydział proporcjonalny (ang. proportional allocation)