Politechnika Wrocławska
Wydział Elektroniki
Informatyka

Opracowanie zagadnień z obrony inżynierskiej ze specjalności: INT, INS

Autorzy:
Piotr Chmiel
Kamil Machnicki
Łukasz Matysiak
Konrad Posłuszny
Jakub Zgraja

Wrocław
28 stycznia 2016

Spis treści

Część I

Pytania kierunkowe

1 K1 – Paradygmat programowania obiektowego

Paradygmat - zbiór mechanizmów działania, wzorów, które definiują sposób realizacji programu.

Programowanie obiektowe stanowi podejście do implementacji algorytmów, które opiera się na wykorzystaniu tak zwanych **obiektów**. Są to twory, które łączą w sobie *dane* (pola obiektu) i *zachowania* (metody obiektu) oraz komunikują się ze sobą w celu wykonania pewnych zadań.

Paradygmat ten ma stanowić ułatwienie w pisaniu i utrzymywaniu kodu, który może być używany wielokrotnie w różnych projektach jednak nie jest odpowiedzią na wszystkie problemy i oprócz swoich zalet ma również wady – np. przez to, że obiekty posiadają wewnętrzny stan, programowanie współbieżne staje się o wiele trudniejsze, gdyż musimy zapobiegać wyścigom czy zagłodzeniu.

Pojęciami, na których się skupię będą:

- Klasa
- Abstrakcja
- Enkapsulacja
- Dziedziczenie
- Polimorfizm
- Wzorce projektowe

1.1 Klasa

Stanowi pewien zbiór cech i zachowań danego obiektu, który jest jej instancją.

Przykładem może być klasa *Pies,* która zawiera cechy takie jak oczy, pysk oraz umiejętność szczekania.

Języki programowania można podzielić ze względu na to, w jaki sposób pojęcie klasy jest zrealizowane. Na przykład w językach *Java*, *C#* czy *Python* każda klasa zawsze ma swoją klasę bazową (zazwyczaj dziedziczona niejawnie i jest to przeważnie klasa bazowa *Object*). Język *C++* na przykład, można nazwać językiem hybrydowym, gdyż umożliwia tworzenie obiektów, lecz nie zmusza do tego programisty (można programować zarówno strukturalnie jak i obiektowo). Istnieją także języki, których obiektowość opiera się na **prototypach** - nowe obiekty tworzone są w oparciu na istniejące już obiekty, nie zaś na podstawie zdefiniowanej klasy. Przykładem takiego języka jest większość języków interpretowanych, np. *JavaScript*.

Głównym zyskiem obiektowości jest modularność - program można podzielić na mniejsze, gotowe do użycia moduły. Zwiększa to także czytelność kodu - łatwiej jest się odnaleźć w metodach

przypisanych do obiektów niż w gąszczu funkcji. Łatwiejsze jest także rozbudowywanie projektów bez integracji w istniejące funkcjonalności.

1.2 Abstrakcja

Czyli poziom ogólności, pozwala nam na upraszczanie problemu poprzez zredukowanie właściwości do jedynie tych kluczowych dla algorytmu (ignorowanie zbędnych detali poprzez wyizolowanie kluczowych aspektów).

Dla przykładu - możemy mówić o *Bazie danych* jako o fragmencie kodu, który będzie stanowił interfejs do komunikacji z pewną bazą danych, jednak nie ma to dla nas większego znaczenia w jaki sposób to będzie realizowane.

Przekładając to na banalny przykład z życia codziennego – kierowca nie musi przechodzić szczegółowych szkoleń za każdym razem, gdy zmienia samochód. Wystarczy mu jedynie podstawowa wiedza o jego działaniu, a takie rzeczy jak znajomość budowy silnika nie są mu potrzebne.

1.3 Enkapsulacja

Inaczej hermetyzacja, polega na celowym ukrywaniu wnętrza obiektów tak, aby zmiany w jego stanie mogły dokonać tylko metody wewnętrzne tego obiektu. Zwiększa to bezpieczeństwo kodu oraz odporność na błędy, a także pozwala podzielić kod na mniejsze fragmenty.

Problem pojawia się, gdy udostępnimy wersję klasy, która ma pewne pola publiczne. Jeśli użytkownicy zaczną z tych pól korzystać, to aktualizacja może spowodować poważne problemy.

Enkapsulacja pomaga ustrzec nas przed niepożądanym korzystaniem z mechanizmów wewnątrz tworzonej przez nas klasy, a na użytek świata zewnętrznego pozwala nam wystawić tylko zdefiniowany przez nas interfejs.

Powracając do przykładu z samochodem – kierowca powinien otwierać okno za pomocą guzika lub korbki, a nie poprzez wybijanie szyby.

Warto wspomnieć, że języki w różny sposób podchodzą do omawianych przeze mnie pojęć. Python jest językiem, w którym pojęcie enkapsulacji praktycznie nie istnieje – wszystko jest dostępne dla wszystkich i jeśli ktoś bardzo chce użyć zmiennych, które wyraźnie oznaczyliśmy jako do użytku wewnętrznego to może to zrobić na własną odpowiedzialność.

Pełna enkapsulacja występuje wtedy, gdy wszystkie pola klasy znajdują się w sekcji prywatnej, a dostęp do nich możliwy jest tylko i wyłącznie poprzez metody.

1.4 Dziedziczenie

Jest narzędziem, które dzięki odpowiednio zaprojektowanej relacji **klasa bazowa - klasa pochodna** pozwala na rozszerzanie funkcjonalności bez duplikowania kodu. Dzięki temu możemy tworzyć hierarchiczne struktury przechodząc od typów najbardziej ogólnych aż do tych szczegółowo opisujących dany obiekt bądź zjawisko.

W odniesieniu do samochodów – możemy dostać wersję podstawową jakiegoś pojazdu, a następnie ją rozszerzyć o stylowe neony, naklejki z ogniem i spoilery.

1.4.1 Dziedziczenie wielokrotne

Tutaj znowu w zależności od języka możemy mieć możliwość skorzystania z mechanizmu dziedziczenia wielokrotnego lub nie. Językami to umożliwiającymi są na przykład *C++* czy *Python*.

Polega ono na tym, że klasa pochodna może dziedziczyć po kilku klasach bazowych. Jest to potężny mechanizm, jednak należy go używać z głową, gdyż może powodować problemy takie jak np. *the diamond problem* (gdy pewna klasa X dziedziczy po dwóch klasach, które to mają wspólną klasę bazową i obie przeciążają tę samą metodę klasy bazowej, z której zatem klasy wywołana zostanie ta metoda, gdy X jej nie przeciąży?).

1.5 Polimorfizm

Pozwala nam na wyabstrahowanie pewnych zachowań od konkretnych typów danych. Dzięki niemu możemy wybrać zachowanie w zależności od kontekstu. Mówiąc ściślej, do jednej referencji można przypisać obiekty różnych typów, które dziedziczą po typie referencji, i wywołując przeciążoną metodę użyta zostanie metoda przypisanego typu referencji.

Dla odmiany rozpatrzmy przykład, w którym klasą **bazową** będzie **Zwierze,** a klasami **pochodnymi** będą **Pies** oraz **Ryba.** Klasa bazowa ma zadeklarowaną metodę (funkcję, którą możemy wywołać na rzecz obiektu) o nazwie *dajGlos.* Klasy pochodne mogą tę funkcję zdefiniować po swojemu i w ten sposób po wywołaniu metody *dajGlos* na obiekcie klasy *Pies* usłyszymy szczekanie, a po wywołaniu metody na obiekcie klasy *Ryba* usłyszymy tylko ciche bulgotanie, któremu towarzyszyć będzie bezgłośne osądzanie naszych wyborów życiowych.

Polimorfizm można podzielić na statyczny i dynamiczny. Polimorfizm **statyczny** (inaczej zwany *wczesnym wiązaniem*) – decyzja o użytym typie zostaje podjęta już na etapie kompilacji – w języku *C++* jest to realizowane za pomocą szablonów bądź przeciążonych operatorów. Polimorfizm **dynamiczny** (*późne wiązanie*) – wybór zostaje podjęty w czasie wykonywania programu – w języku *C++* zrealizowane przy użyciu wskaźników przeciążając metody wirtualne w klasach pochodnych.

1.6 Wzorce projektowe

Powstały, aby spisać często napotykane podczas programowania problemy oraz zdefiniować sprawdzone rozwiązania.

Jest to temat bardzo atrakcyjny dla początkujących programistów i jest jednocześnie bardzo pomocny i bardzo niebezpieczny, ponieważ początkujący mogą chcieć korzystać ze wzorców gdzie tylko mogą, nie zważając na to czy faktycznie są one potrzebne.

Należy pamiętać o antywzorcach złotym młotku oraz srebrnym pocisku – nie wszystko co się do tej pory sprawdziło gdzie indziej sprawdzi się w naszym przypadku, a to że doskonale znamy jakąś technologię nie znaczy, że jest ona zawsze odpowiednia.

Wzorce projektowe dzielimy wg trzech kategorii:

- konstrukcyjne – opisujące proces tworzenia nowych obiektów, przykładowo **fabryka, singleton,**

- strukturalne – opisujące struktury powiązanych ze sobą obiektów, przykładowo **adapter, dekorator,**
- behawioralne – opisujące zachowanie i odpowiedzialność współpracujących ze sobą obiektów, przykładowo **strategia, null object.**

2 K2 – Arytmetyka stało- i zmiennoprzecinkowa

Ludzie spodziewają się, że komputery będą dokładne i nieomylne. Spodziewają się, że po wpisaniu 0.1 + 0.2 kalkulator odpowie: 0.3.

Prędzej czy później każdy programista próbuje czegoś takiego i natrafia na problem: komputer twierdzi, że wynik tego dodawania to np. 0.30000001. Skąd takie rzeczy?

Każda informacja w komputerze jest przedstawiona za pomocą podstawowych jednostek – bitów. W przypadku liczb, można je reprezentować za pomocą typów {stało-,zmienno-}przecinkowych. Różnią się cechami takimi jak:

- sposób zapisu,
- zakres wartości,
- precyzja operacji.

Typy stałoprzecinkowe posiadają wiele reprezentacji.

2.1 Naturalny kod binarny (NBC)

Jest to system pozycyjny o podstawie 2. Liczby zapisane w tej reprezentacji nie posiadają znaku, nie można za jego pomocą zapisać ułamków. Wartość liczby można zaprezentować za pomocą wzoru (liczba *n-bitowa*):

$$x = \sum_{i=0}^{n-1} 2^i \times x_i \tag{1}$$

Na przykład:

DEC	BIN (NKB)
0	00
1	01
2	10
3	11

2.2 Kod uzupełnieniowy U2

W celu reprezentacji liczby całkowitej potrzebne jest zdefiniowanie znaku liczby. Wartość liczb zapisuje się podobnie jak w wypadku NKB, jednak znak liczby jest określony przez bit najbardziej znaczący (0 – liczba dodatnia, 1 – liczba ujemna). Aby otrzymać wartość n-bitowej liczby zapisanej w kodzie U2 można skorzystać z następującego wzoru:

$$x = (-1)^{n-1} + \sum_{i=0}^{n-2} 2^i \times x_i \tag{2}$$

Jak wynika ze wzoru, o ile liczba dodatnia wygląda tak samo jak w NKB, to postać liczby ujemnej zapisanej w U2 wcale nie jest taka sama jak w NKB z dodaną jedynką na przodzie.

DEC	NKB	U2
-3	—	101
-2	—	110
-1	—	111
0	000	000
1	001	001
2	010	010
3	011	011

Dwa łatwe sposoby na odczytanie wartości liczby ujemnej w U2:

- Neguj wszystkie bity i dodaj 1.
- Potraktuj najstarszy bit jak ujemną wartość na tej pozycji i dodaj do niej pozostałe bity.

W dwójkowym systemie uzupełnieniowym porządek kodów arytmetycznych jest zachowany w całym zakresie. $\{0, x_{n-2}, \ldots, x_1, x_0\}$ reprezentuje dodatnią liczbę x, a $\{1, x_{n-2}, \ldots, x_1, x_0\}$ to zapis liczby **większej o x od najmniejszej liczby** -2^{n-1}.

Podobnie jak dla NKB, nie jest możliwe zastosowanie tutaj zapisu ułamków.

2.3 Typ stałoprzecinkowy

Reprezentacja ta jest podobna do kodu uzupełnieniowego, jednakże można tutaj określić pozycję przecinka. Dla liczb o podstawie β oraz ustalonej liczbie pozycji części ułamkowej – r, wartość każdej liczby jest podana z dokładnością β^{-r}. Można ją przedstawić za pomocą iloczynu liczby całkowitej, wtedy zapis wygląda następująco:

$$\{x_m, x_{m-1}, \ldots, x_1, x_0, \ldots, x_{-r}\} \tag{3}$$

Podczas operacji z liczbami stałoprzecinkowymi można otrzymać przeniesienie w wypadku operacji dodawania lub mnożenia.

2.4 Inne systemy stałoprzecinkowe

- system obciążony,
  - \+ unikatowa reprezentacja zera,
  - \+ zgodność uporządkowania liczb i ich reprezentacji,
  - \- wymagana korekcja wyników działań arytmetycznych,
  - \- problematyczne przy dzieleniu i mnożeniu.
- system ze znakowaną cyfrą.

2.5 Typ zmiennoprzecinkowy

2.5.1 Budowa

Przedstawione do tej pory systemy były **stało**przecinkowe. Znaczy to tyle, że miały z góry określoną precyzję i rozmiar.

Jedną z wad typów stałoprzecinkowych jest ilość miejsca potrzebna do przechowywania liczb: aby przechować liczbę z zakresu <0; 4 294 967 295> potrzebujemy aż 32 bitów pamięci.

Wyobraźmy sobie, że potrzebne nam są liczby rzędu rozmiaru wszechświata albo wielkości atomu.

Gdybyśmy takie liczby chcieli zapisać w typie stałoprzecinkowym to musielibyśmy zmarnować ogromne ilości pamięci na przechowywanie zer.

Rozwiązanie? Notacja naukowa i zmienny przecinek.

Liczba zmiennoprzecinkowa jest reprezentowana przez 4 liczby (S, β, M, E):

- S – znak,
- β – podstawa,
- E – wykładnik,
- M – mantysa.

$$x = (-1)^S \times \beta^E \times M \tag{4}$$

Łatwo zauważyć, że jest to po prostu notacja naukowa, lecz jak tak właściwie taka liczba wygląda?

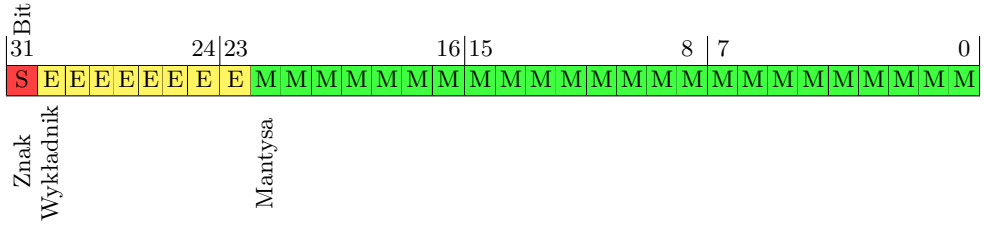
Pierwszy bit jest bitem znaku – 0 dla liczb dodatnich i 1 dla liczb ujemnych. Tak jest zawsze.

Kolejne bity to bity wykładnika i to ile tych bitów jest zależy od typu (precyzji). Dla liczby pojedynczej precyzji, czyli typu 32-bitowego, liczba bitów wykładnika wynosi 8.

Ważne: wykładnik jest zapisany w systemie obciążonym +N ($N = 2^{k-1} - 1$, gdzie k to liczba bitów wykładnika) – dla 32-bitowej liczby jest to system +127.

Pozostałe bity to liczby części ułamkowej. Ważne jest tutaj pojęcie liczby znormalizowanej, czyli takiej, dla której przyjmujemy, że część ułamkowa ma na początku ukrytą jedynkę – jest więc postaci 1.bb...bb, a nie 0.bb...bb. Więcej o tym za chwilę. **Ważne:** mantysa zawsze jest liczbą dodatnią.

Tablica 1: Liczba zmiennoprzecinkowa pojedynczej precyzji (32 bity)



Standard wyróżnia następujące precyzje:

- połowicza (16 bit),
- pojedyncza (32 bit),
- podwójna (64 bit),
- podwójna rozszerzona (80 bit).

Przykład: Liczba 17_{10} zapisana jako 32-bitowa liczba zmiennoprzecinkowa ma postać:

0|10000011|00010000000000000000000

czyli

- Znak – 0 -> liczba dodatnia
- Wykładnik – $10000011_2 == 2_{10}^{131-127} == 2_{10}^4$
- Mantysa – $00010000000000000000000_2 == 1.0625_{10}$

Po złożeniu wychodzi nam: $(-1)^0 * 2^4 * 1.0625 == 1 * 16 * 1.0625 == 17.0$

Jaka jest główna zaleta takiego zapisu? Ogromny wręcz zakres wartości możliwych do zapisania: korzystając z 32-bitowego typu zmiennoprzecinkowego możemy zapisać wartości z oszałamiającego zakresu <-3.40282e+38, 3.40282e+38>.

Powróćmy do pojęcia liczby znormalizowanej.

2.5.2 Liczby znormalizowane i zdenormalizowane

Standard IEEE754 wymaga, aby liczby były znormalizowane (oprócz wartości specjalnych) – mantysa powinna być w zakresie $1 \leqslant |M| < 2$. Liczbę taką można poznać po tym, że wykładnik jest **różny** od zera. Wtedy mantysa liczby jest postaci 1.bb...bb.

W wypadku, gdy wykładnik liczby jest równy zero, mówimy, że liczba jest zdenormalizowana, a jej mantysa ma postać 0.bb...bb.

Liczby zdenormalizowane są potrzebne do zapisania wartości tak bliskich 0, że ich ukryty bit w mantysie musi wynosić 0.

Będąc przy tym temacie warto wspomnieć o wartościach specjalnych.

2.5.3 Wartości specjalne

Liczby zmiennoprzecinkowe zawierają zarezerwowane wartości dla liczb specjalnych.

Wartości specjalne to:

- Zero – oprócz bitu znaku zawiera same zera (tak więc mamy tu zero dodatnie i ujemne).
- $\pm\infty$ – wszystkie bity wykładnika wynoszą 1, mantysa 0.
- Nie-liczba (NaN) – wykładnik również przyjmie same jedynki, natomiast mantysę $\neq 0$.
- Liczby zdenormalizowane posiadają w wykładniku same zera, natomiast mantysa jest postaci $0.bbb\ldots bb$.

Tablica 2: Liczby w standardzie IEEE754

Znak	**Wykładnik**	**Mantysa**	**Wartość**
0	$00\cdots00$	$00\cdots00$	$+0$
0	$00\cdots00$	$00\cdots01$ $\vdots$ $11\cdots11$	Dodatnia zdenormalizowana $(0.f \times 2^{-N+1})$
0	$00\cdots01$ $\vdots$ $11\cdots10$	$XX\cdots XX$	Dodatnia znormalizowana $(1.f \times 2^{E-N})$
0	$11\cdots11$	$00\cdots00$	$+\infty$
0	$11\cdots11$	$00\cdots01$ $\vdots$ $01\cdots11$	SNaN
0	$11\cdots11$	$1X\cdots XX$	QNaN
1	$00\cdots00$	$00\cdots00$	-0
1	$00\cdots00$	$00\cdots01$ $\vdots$ $11\cdots11$	Ujemna zdenormalizowana $(-0.f \times 2^{-N+1})$
1	$00\cdots01$ $\vdots$ $11\cdots10$	$XX\cdots XX$	Ujemna znormalizowana $(-1.f \times 2^{E-N})$
1	$11\cdots11$	$00\cdots00$	$-\infty$
1	$11\cdots11$	$00\cdots01$ $\vdots$ $01\cdots11$	SNaN
1	$11\cdots11$	$1X\cdots XX$	QNaN

N – obciążenie wykładnika, X – dowolna wartość, QNaN – quiet not-a-number (nie zgłaszają wyjątków), SNaN – signaling not-a-number (zgłaszają wyjątki)

Tablica 3: Specjalne operacje arytmetyczne

Operacja	**Wynik**
$\frac{n}{\pm\infty}$	0
$\pm\infty \times \pm\infty$	$\pm\infty$
$\frac{\pm X}{0}$	$\pm\infty$
$X \times \pm\infty$	$\pm\infty$
$\infty + \infty$ $\infty - -\infty$	$+\infty$
$-\infty - \infty$ $-\infty + -\infty$	$-\infty$
$\infty - \infty$ $-\infty + \infty$	NaN
$\frac{\pm 0}{\pm 0}$	NaN
$\frac{\pm\infty}{\pm\infty}$	NaN
$\infty \pm 0$	NaN

Standard wyróżnia również wyjątki:

- invalid operation — niewłaściwa operacja,
- division by zero — dzielenie przez zero,
- overflow — nadmiar – liczba jest za duża,
- underflow — niedomiar – utrata precyzji przy liczbach bliskich zero,
- inexact — niedokładność – podczas operacji zaokrąglania.

2.6 Działania

2.6.1 Arytmetyka stałoprzecinkowa

- **Dodawanie/odejmowanie**
  - dodawanie kolejnych cyfr, z zachowaniem ich pozycji
  - występują przeniesienia
  - odejmowanie – dodawanie liczby przeciwnej (odjętej od zera)
- **Mnożenie**
  - mnożenie mnożnej kolejnymi cyframi mnożnika, potem dodanie kolejnych sum częściowych
  - w systemie uzupełnieniowym jest podobnie jak w naturalnym, gdy mnożnik ujemny wymagana jest korekcja wyniku
- **Dzielenie**
  - wynik dzielenia jest przybliżony (nie ma gwarancji, że dla dzielnej X oraz dzielnika D istnieje liczba Q, że $X = Q * D$, można to skorygować dodając resztę z dzielenia)

– właśnie ogarnąłem, że opisałem fakty, których uczyliśmy się w podstawówce

2.6.2 Arytmetyka zmiennoprzecinkowa

W przypadku liczb zmiennoprzecinkowych ważna jest kolejność działań! Wynika to z błędów precyzji i zaokrągleń.

- **Dodawanie/odejmowanie**
  - wymaga wyrównania wykładników (mniejszy do większego)
  - denormalizacja jednej liczby, następnie dodanie/odjęcie mantys, potem normalizacja wyniku
  - nadmiar/niedomiar może wystąpić przy normalizacji
- **Mnożenie/dzielenie**
  - mnożenie – przemnożenie mantys, dodanie wykładników, normalizacja
  - dzielenie – podzielenie mantys, odjęcie wykładników, normalizacja
  - może wystąpić nadmiar/niedomiar

2.6.3 Zaokrąglanie

Na koniec powiedzmy sobie o zaokrągleniach liczb.

Są one potrzebne, ponieważ w systemie binarnym nie zawsze da się dokładnie zapisać wszystkie liczby (ten problem występuje również w innych systemach).

Przykładowo, liczba dziesiętna $\frac{1}{10}_{10} == 0.1_{10}$ daje się łatwo zapisać w systemie dziesiętnym, jednak w binarnym jest ona ułamkiem okresowym $0.1(0011)_2$. Ze względu na naturę komputerów, pamięć jest ograniczona, a co za tym idzie nie jesteśmy w stanie zapisać liczb z nieskończoną precyzją.

Kiedy komputer napotyka liczbę, której nie jest w stanie dokładnie zapisać, zaokrągla ją.

Zanim przejdziemy do omawiania konkretnych zaokrągleń, porozmawiajmy o trzech bitach, które są ukryte za mantysą: G,R,S.

Bity G(uard)R(ound)S(ticky) istnieją dla celów zaokrągleń i przechowywane są w nich bity niemieszczące się w mantysie. Już tłumaczę.

Jeśli chcemy naszą mantysę podzielić przez np. 8, czyli przesunąć bity w prawo o 3 pozycje, to wygląda to następująco:

mantysa |GRS

000111010|000

xxx000111|010

001000111|010

Skąd wzięło się 001 na miejscu xxx i czym w końcu są te bity GRS?

Bity 001 na początku wzięły się stąd, że mieliśmy do czynienia z liczbą znormalizowaną, więc na początku stała ukryta 1, a że mantysa zawsze jest dodatnia, to przed nią stoi nieskończenie wiele zer. Jasne? Jasne.

Bity G oraz R to po prostu dwa dodatkowe bity używane przez implementację do przechowywania dodatkowych wartości.

Bit S mówi nam czy po prawej od bitów G oraz R może znajdować się jeszcze jakaś '1' – jeśli jest możliwe, że wśród utraconych bitów była jakaś '1' to bit jest ustawiany na '1', a w przeciwnym wypadku na '0'.

Dobrze, rozpatrzmy różne zaokrąglenia na przykładzie liczby $0.1(0011)_2$

Zaokrąglanie do zera

Na początku wybieramy ile miejsca możemy poświęcić na zapisanie naszej liczby, a resztę kasujemy. Bitami GRS w ogóle się nie przejmujemy. Wybierzmy 4 miejsca po przecinku.

$0.10011001100110011...0011_2 \Rightarrow 0.1001_2$

Już. Proste, prawda?

Zaokrąglanie do $+\infty$

Niezależnie od wartości, zawsze zaokrąglamy w górę.

Zaokrąglanie do $-\infty$

Niezależnie od wartości, zawsze zaokrąglamy w dół – czyli dla ujemnych wartości, liczba będzie jeszcze mniejsza.

Zaokrąglanie do najbliższej, w kierunku parzystej

Jest to domyślny tryb zaokrąglania w IEEE-754. Przy tym zaokrąglaniu bity GRS zaczynają mieć znaczenie. Jeżeli $GRS \geqslant 101$ (czyli 101, 110 lub 111), to zaokrąglamy w górę – dodajemy 1 do *ulp* (ang. *unit in the last place*, najmłodszy bit). Jeżeli nastąpi przepełnienie mantysy (wyjdzie poza zakres $1 \leqslant M < 2$), to należy ją znormalizować oraz zwiększyć wykładnik. Jeśli $GRS = 100$ to mamy dwie sytuacje:

- $ulp = 1 \Rightarrow$ zaokrąglamy w górę
- $ulp = 0 \Rightarrow$ nic nie robimy

Zostaje jeszcze ostatnia możliwość: $GRS = 0xx$ (bity R i S nie mają znaczenia) – wtedy nie zmieniamy mantysy.

Zaokrąglanie do najbliższej, w kierunku nieskończoności

Zaokrąglanie to działa podobnie do powyższego dla liczb dodatnich, dla liczb ujemnych zaokrąglenie w dół. Zostało stworzone dla systemu o podstawie dziesiętnej.

2.7 Podsumowanie

Cecha	L. stałoprzecinkowe	L.zmiennoprzecinkowe
Reprezentacja	Tak jak w systemie pozycyjnym lub uzupełnieniowym	Reprezentacja przez: znak, wykładnik, mantysę
Dokładność	Mały zakres, dokładne wyniki	Duży zakres, możliwe zaokrąglenia, utrata precyzji
Obsługa błędów	Może wystąpić przepełnienie	Zaokrąglenia, błędne operacje, przepełnienie, niedomiar
Wartości specjalne	—	+0, -0, $+\infty$, $-\infty$, QNaN, SNaN, liczby zdenormalizowane
Arytmetyka	Wszystkie twierdzenia prawdziwe	Kolejność działań wpływa na wynik

3 K3 – Normalizacja schematu bazy danych

Normalizacja schematu bazy danych, czyli sprowadzanie schematu do jednej z postaci normalnych jest dokonywana, aby przeciwdziałać lub też zapobiegać problemom, które pojawiają się podczas cyklu życia bazy danych.

Przykładowe problemy związane z użytkowaniem i utrzymaniem bazy danych:

- brak spójności danych,
- zbyt skomplikowane zapytania,
- anomalie spowodowane aktualizacją danych.

Normalizacja najczęściej sprowadza się do rozbijania tabel na mniejsze, jednak co ważne, nie wpływa ona na dane, a jedynie na sposób w jaki je przechowujemy.

Sprowadzenie schematu do którejś z postaci normalnych polega na sprawieniu, by schemat spełniał warunki określone przez postać do której dążymy oraz przez wszystkie poprzednie.

Przykładowo: schemat jest w 3NF jeśli spełnia warunki 3NF, 2NF oraz 1NF.

Pan Codd, który był twórcą pojęcia normalizacji, wymyślił trzy postaci normalne, z których trzecia jest przez większość uważana za wystarczającą, jest ich jednak więcej.

Zanim omówimy poszczególne postaci, chciałbym wspomnieć o jednej ważnej rzeczy – **normalizacja bazy danych jest bardzo dobrą praktyką, która nie zawsze jest jednak konieczna, a czasami może być wręcz niepożądana.**

Wynika to z tego, że po rozbiciu na wiele tabel, system bazodanowy musi wykonywać złączenia przy pozyskiwaniu danych, co może niekorzystnie wpływać na wydajność systemu.

1NF

Określa podstawowe zasady, które musi spełnić każda dobrze zorganizowana baza danych:

1. Tabele nie posiadają powtarzających się kolumn.
2. Dane w każdej kolumnie są niepodzielne.
3. Każdy wiersz daje się jednoznacznie zidentyfikować (klucz główny).
4. Kolejność wierszy i kolumn nie ma znaczenia.
5. Dane w jednej kolumnie są tego samego rodzaju.

Czy poniższa tabela spełnia warunki 1NF?

Tablica 4: Transakcje - przed normalizacją

Klient	Towar
Adam Adamowicz	Lalka barbie
Anna Annowska	Lalka barbie, Wino grzane

Widzimy, że tabela nie jest w 1NF, ponieważ kolumna *Klient* zawiera dane, które da się rozbić na dwie mniejsze kolumny – imię oraz nazwisko klienta. Brakuje również unikalnego identyfikatora wierszy. Widzimy też, że w kolumnie *Towar* pojawia się wiele wartości, co jest błędem.

Tabela po sprowadzeniu do 1NF wygląda następująco:

Tablica 5: Transakcje - 1NF

Id	Imię	Nazwisko	Towar
1	Adam	Adamowicz	Lalka barbie
2	Anna	Annowska	Lalka barbie
3	Anna	Annowska	Wino grzane

2NF

Druga postać normalna jeszcze bardziej zagłębia się w usuwanie redundancji:

1. Spełnia warunki 1NF.
2. Atrybuty **niekluczowe** zależą funkcyjnie od **pełnego** klucza.

Co to tak właściwie oznacza?

Znaczy to tyle, że jeśli któraś z kolumn niekluczowych zależy tylko od części klucza, to jest to błąd.

Załóżmy tabelę, w której klucz jest złożony z kolumn **id_kursu** oraz **id_semestru**:

Tablica 6: Kursy - 1NF

id_kursu	id_semestru	sala	nazwa_kursu	id_prowadzacego	nazwisko_prowadzacego
I1	2015-16-z	10	Programowanie	1	Nauczycielowicz
I1	2015-16-l	10	Programowanie	1	Nauczycielowicz
E1	2015-16-z	15	Elektronika	2	Prowadzącywicz

Widzimy, że kolumna *nazwa_kursu* zależy tylko od części klucza - *id_kursu,* ponieważ nazwa kursu zależy od jego id, ale nie ma nic wspólnego z semestrem, w którym się odbywa. Kolumny *sala, id_prowadzacego* oraz *nazwisko_prowadzacego* nie naruszają warunków 2NF, ponieważ

- kolumny *sala* oraz *id_prowadzacego* zależą od całości klucza – prowadzący i sala zmienia się zarówno ze względu na przedmiot jak i na semestr,
- kolumna *nazwisko_prowadzacego* wcale nie zależy od klucza.

Aby sprowadzić tabelę do 2NF należy rozbić ją w następujący sposób:

Tablica 7: Kursy - 2NF

id_kursu	id_semestru	sala	id_prowadzacego	nazwisko_prowadzacego
I1	2015-16-z	10	1	Nauczycielowicz
I1	2015-16-l	10	1	Nauczycielowicz
E1	2015-16-z	15	2	Prowadzącywicz

Tablica 8: Nowa tabela wyekstrahowana z tabeli Kursy - 2NF

id_kursu	nazwa_kursu
I1	Programowanie
E1	Elektronika

3NF

Trzecia postać normalna idzie o krok dalej:

1. Spełnia warunki 2NF.
2. Atrybuty **niekluczowe** zależą **tylko** od **pełnego** klucza.

Różnica między 2NF i 3NF brzmi subtelnie, a polega na tym, że z tabeli wyciągamy wszystkie dane, które w żaden sposób nie zależą od klucza.

Tabela *Kursy* sprowadzona do 3NF wygląda następująco:

Tablica 9: Kursy - 3NF

id_kursu	id_semestru	sala	id_prowadzacego
I1	2015-16-z	10	1
I1	2015-16-l	10	1
E1	2015-16-z	15	2

Tablica 10: Nowa tabela wyekstrahowana z tabeli Kursy - 3NF

id_prowadzacego	nazwisko_prowadzacego
1	Nauczycielowicz
2	Prowadzącywicz

4 K4 – Model warstwowy TCP/IP

Model TCP/IP jest modelem warstwowej struktury protokołów komunikacyjnych. Dzięki niemu dokonywana jest współpraca między różnym rodzajem sprzętu, technologii sieciowych oraz oprogramowania. Funkcje sieci komputerowych są podzielone na grupy, które są realizowane na różnych warstwach.

Stopień skomplikowania sieci komputerowych spowodował, że różne funkcje sieci należy podzielić na grupy (warstwy). Każda warstwa ma swoją określoną rolę. Nawiązanie połączenia między np. serwerami pocztowymi (pracują na tej samej warstwie) na dwóch komputerach w rzeczywistości odbywa się przez wykorzystanie warstw niższych.

W latach osiemdziesiątych stworzono model ISO/OSI. Został zaprojektowany jako „otwarty" model – opublikowany za darmo, bez restrykcji patentowych bądź ograniczeń rozpowszechniania. Aktualnie model ten jest traktowany jako wzorzec dla protokołów komunikacyjnych. Posiada on siedem warstw:

7	aplikacji
6	prezentacji
5	sesji
4	transportowa
3	sieciowa
2	łącza danych
1	fizyczna

Warstwa fizyczna zapewnia przekaz bitów między stacjami połączonymi fizycznym medium (kable miedziane, światłowody, łącza radiowe).

Warstwa łącza danych umieszcza dane w ramkach, które zawierają ciągi kontrolne. Warstwa ta również sprawdza jakość przekazywanych informacji, próbuje naprawić ewentualne błędy.

Warstwa sieciowa jest odpowiedzialna za ustalenie drogi do docelowego urządzenia. Jednostką danych w tej warstwie są pakiety. Występuje tutaj segmentacja danych.

Warstwa transportowa ma nadzór nad połączeniem (inicjacja, zrywanie) między dwoma stacjami. Warstwa ta jest również odpowiedzialna za podział danych na bloki, kontrolę poprawności transmisji, rozpoznawanie duplikatów oraz sprawdzanie poprawności adresowania.

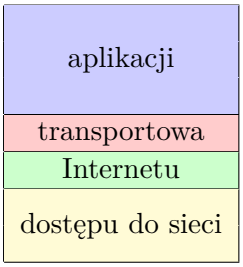
Warstwa sesji synchronizuje przesył danych, wznawia go po przerwaniu połączenia.

Warstwa prezentacji przekształca dane użytkowników do postaci standardowej, używanej w sieci (np. zamiana Little Endian na Big Endian). Innymi cechami tej warstwy jest kompresja oraz szyfrowanie danych.

Warstwa aplikacji zapewnia obsługę użytkowników w dostępie do usług (transmisja plików, zdalny terminal, poczta, etc.).

Koncepcja modelu TCP/IP jest identyczna jak w modelu OSI/ISO. Obecnie jest wykorzystywany jako struktura Internetu. Architektura tego modelu została opracowana w celu umożliwienia komunikacji między systemami pochodzącymi od różnych dostawców. W porównaniu do modelu

OSI posiada mniej warstw, analogiczne warstwy zostały zaznaczone identycznym kolorem, jak w poprzedniej tabeli.



Warstwa dostępu do sieci może zawierać protokoły dynamicznego przydzielania adresów IP. Protokoły działające w tej warstwie: Ethernet, WiFi.

Warstwa Internetu przetwarza dane posiadające adresy IP. Protokół działający w tej warstwie: IP.

Warstwa transportowa wykorzystuje protokoły TCP oraz UDP. Przesyłanie danych do odpowiednich aplikacji odbywa się za pomocą portów określonych dla każdego połączenia.

Warstwa aplikacji wykorzystuje różne protokoły (HTTP, XMPP, FTP, IRC, SSH, etc.).

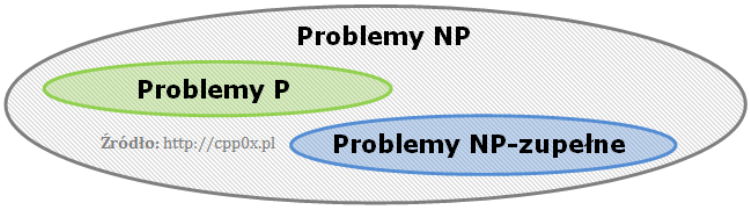
Uzupełnienie: `http://lucc.pl/inf/technologie_sieciowe_1/walkowiak_wyklady/w3_-_model_TCPIP.ppt`

5 K5 – Ocena złożoności algorytmów

Złożoność algorytmu jest miarą ilości zasobów potrzebnych do rozwiązania danego problemu o określonym rozmiarze. Dla przykładu w algorytmie rozkładu liczb na czynniki pierwsze zauważyć można, że im większa liczba, tym dłużej będziemy ją rozbijać i wykonamy więcej obliczeń. Faktem zatem jest, że im większy rozmiar danych wejściowych, tym więcej zasobów potrzebujemy do rozwiązania problemu. Złożoność algorytmu można więc określić mianem **funkcji rozmiaru danych wejściowych**.

Wyróżnić możemy kilka **klas złożoności**, czyli grup zagadnień o podobnej złożoności obliczeniowej. Są to:

1. **NP** (*nondeterministic polynomial*) - problemy decyzyjne rozwiązywalne niedeterministycznym algorytmem wielomianowym.
2. **P** (*deterministic polynomial*) - problemy decyzyjne, które można rozwiązać deterministycznym algorytmem o złożoności wielomianowej.
3. **NP-zupełny** - problemy decyzyjne, których znalezienie rozwiązania nie jest możliwe w czasie wielomianowym.
4. **NP-trudny** (silnie NP-zupełny) - samo sprawdzenie rozwiązania problemu jest co najmniej tak trudne jak każdego innego problemu NP.



Rysunek 1: Klasy złożoności

Aby określić złożoność stosowane są metody **oceny złożoności**. W celu ujednolicenia wyników i uniezależnienia się od środowiska, ocena złożoności rozpatrywana jest modelem abstrakcyjnym nie uwzględniającym platformy czy mocy obliczeniowej maszyny. Wyróżniamy następne modele złożoności:

1. **Czasowe** - czas wykonywania algorytmu wynikająca z liczby operacji elementarnych wykonywanych podczas przebiegu programu.
2. **Pamięciowe** - pamięcio-żerność oraz zasobo-żerność - ilość pamięci operacyjnej, bądź przestrzeni dyskowej potrzebnej do rozwiązania problemu. Zależy od liczby oraz złożoności użytych struktur.

Przy wyborze algorytmu zazwyczaj decydująca jest złożoność czasowa - o wiele bardziej miarodajny model.

Wyróżniamy także podział złożoności algorytmów ze względu na instancję problemu:

1. **Optymistyczna** - najlepszy możliwy przypadek (minimalna ilość wykonywanych operacji).
2. **Pesymistyczna** - najgorszy scenariusz (wykonywane są wszystkie operacje).

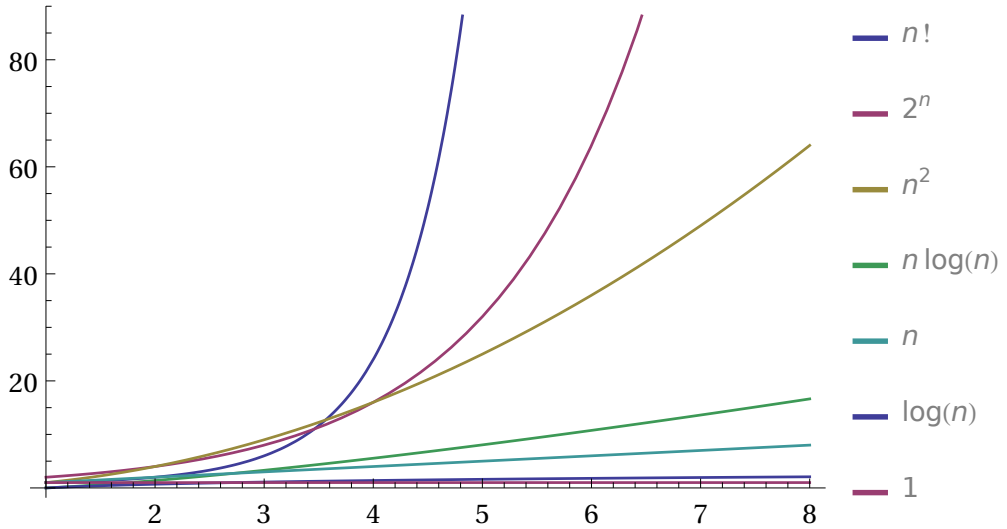
3. **Średnia** - wartość oczekiwana wykonywania algorytmu.

Do oceny złożoności algorytmu potrzebny jest **rząd wielkości** oraz zmienność zależnie od instancji problemu. Nie potrzebne są dokładne wartości, dlatego też stosuje się **asymptotyczne tempo wzrostu**, które to informuje nas o zmianie (wzroście bądź spadku) złożoności algorytmu w zależności od ilości danych wejściowych. Opisuje jak szybko funkcja rośnie, maleje, bądź pozostaje stała.

Asymptotyczne tempo wzrostu określane jest za pomocą *notacji asymptotycznych*. Istnieje kilka notacji *Bachmann-Landaua*:

- $\mathcal{O}$ (omikron, duże O) - Funkcja asymptotycznie **niewiększa**. Jest to ograniczenie górne, czyli taka funkcja, dla której wszystkie wartości funkcji badanej, będą od niej niewiększe (mniejsze bądź równe).
- Ω (omega) - Funkcja asymptotycznie **niemniejsza**. Jest to granica dolna, czyli funkcja, dla której wszystkie wartości funkcji badanej, będą od niej niemniejsze (większe bądź równe).
- Θ (theta) - Funkcja asymptotycznie **podobna**. Jest to oszacowanie dokładne, czyli funkcje ograniczające badaną funkcję od góry i od dołu.
- o - (małe O) - Funkcja asymptotycznie mniejsza. Rzadko stosowana.
- ω - (mała omega) - Funkcja asymptotycznie większa. Rzadko stosowana.

Złożoności kategoryzuje się do rzędów złożoności. Określają one jak szybko złożoność algorytmu rośnie wzraz ze wzrostem instancji problemu.



Rysunek 2: Porównanie rzędów złożoności

Wyróżnić można (n - *liczba danych wejściowych*):

- $\mathcal{O}(1)$ - **złożoność stała** - liczba operacji jest niezależna od rozmiaru problemu. Przykład: Ustalenie, czy liczba binarna jest dodatnia bądź ujemna, obliczenie $(-1)^n$, odwołanie się do n-tego elementu stałej w rozmiarze tablicy.
- $\mathcal{O}(\log n)$ - **logarytmiczna** - liczba operacji rośnie proporcjonalnie do logarytmu z rozmiaru

problemu. Przykład: Wyszukiwanie binarne w posortowanej tablicy, dodanie/usunięcie elementu z drzewa binarnego.

- $\mathcal{O}(n)$ - **liniowa** - liczba operacji jest wprost proporcjonalna do rozmiaru problemu. Przykład: Znalezienie min/max elementu w nieposortowanej tablicy, dodanie dwóch n-bitowych liczb całkowitych w sumatorze z przeniesieniem szeregowym.
- $\mathcal{O}(n \log n)$ - **quasi-liniowa** - liczba operacji proporcjonalna do iloczynu rozmiaru problemu i jego logarytmu. Sortowania przez porównania (MergeSort (przez scalenie), HeapSort (kopcowe), QuickSort (szybkie)), FFT (Szybka Transformata Fouriera).
- $\mathcal{O}(n^2)$ - **kwadratowa** - liczba operacji rośnie proporcjonalnie do kwadratu rozmiaru problemu. Przykład: Prosty algorytm mnożenia dwóch n-bitowych liczb, sortowania: BubbleSort (bąbelkowe), SelectionSort (przez wybieranie), InsertionSort (przez wstawianie), znalezienie najkrótszej ścieżki w grafie.
- $\mathcal{O}(n^k)$ - **wielomianowa** - liczba operacji rośnie proporcjonalnie do wielomianu rozmiaru problemu. Przykład: Programowanie liniowe, maksymalne skojarzenie dla grafu dwudzielnego.
- $\mathcal{O}(2^n)$ - **wykładnicza** - liczba operacji rośnie proporcjonalnie do wartości wykładniczej rozmiaru problemu. Przykład: Programowanie dynamiczne dla problemu komiwojażera.
- $\mathcal{O}(n!)$ - **rzędu silnia** - liczba operacji rośnie proporcjonalnie do silni rozmiaru problemu. Przykład: Algorytm brute-force dla problemu komiwojażera.

6 K6 – Język UML w projektowaniu oprogramowania

UML w kontekście projektowania oprogramowania jest językiem pozwalającym na modelowanie i opisywanie systemów i ich części.

Pozwala na standaryzowany, graficzny zapis systemu z uwzględnieniem zarówno jego części konceptualnych, takich jak funkcje i procesy biznesowe, jak i obiektów fizycznych jakimi mogą być bazy danych czy warstwa sprzętowa.

W jaki sposób używa się UMLa do modelowania? Korzysta się z diagramów, które są podzielone ze względu na modelowanie strukturalne – diagramy pakietów, klas, komponentów itd – oraz behawioralne – diagramy sekwencji, przypadków użycia, stanu itd.

UML jest przydatnym narzędziem nie tylko w procesie modelowania, ale również podczas szybkich rozmów między programistami kiedy chce się szybko nakreślić ideę.

Omówmy kilka najczęściej spotykanych diagramów.

6.1 Diagramy strukturalne

6.1.1 Diagram pakietów

Pełni rolę organizacyjną. Zawiera w sobie inne elementy języka, zazwyczaj diagramy klas. Jest to ogólna wizualizacja systemu, z przedstawieniem zależności pomiędzy jego częściami.

6.1.2 Diagram klas

Składa się z zamodelowanych klas oraz relacji między nimi.

Pojedynczą klasę umieszcza się w prostokącie podzielonym na trzy części

- nazwę,
- atrybuty,
- funkcje.

Atrybuty oraz funkcje oznacza się modyfikatorem dostępu (np. + dla public, - dla private), składniki statyczne zaznacza się poprzez podkreślenie, a stereotypy – elementy służące do doprecyzowania semantyki elementu, np. oznaczenie jako klucz główny – umieszcza się w podwójnych nawiasach ostrych «».

Relacje między klasami:

- **Zależność – strzałka przerywana** – podstawowa relacja mówiąca, że obiekt może w jakiś sposób korzystać lub wpływać na inne obiekty.
- **Asocjacja – linia ciągła** – obiekty wykorzystują inne obiekty (człowiek używa magazynu do przechowywania rzeczy).

- **Agregacja częściowa – pusta strzałka z rombem** – obiekt posiada inne obiekty, ale nie są one do niego przypisane na wyłączność (kiedy człowiek umrze, jego samochód dalej istnieje).
- **Agregacja całkowita – pełna strzałka z rombem** – obiekt posiada inne obiekty i jest za nie odpowiedzialny (kiedy człowiek umrze, jego serce umiera wraz z nim).
- **Dziedziczenie – pusta strzałka** – określa hierarchię dziedziczenia.

6.1.3 Diagram wdrożenia

Pokazuje on strukturę fizyczną systemu, z uwzględnieniem obiektów systemu oraz elementami zewnętrznymi (np. maszyna obliczeniowa, czy też baza danych). Diagram ten jest przydatny przy wdrażaniu dużych systemów.

6.2 Diagramy behawioralne

6.2.1 Diagram przypadków użycia

Diagram umożliwia analizę oraz dokumentację wymagań co do funkcjonalności systemu – w jasny sposób pokazuje co jest od niego wymagane i pomaga w opracowaniu projektu oraz przetestowaniu już zbudowanego systemu. Co ważne, diagram nie posiada szczegółowych informacji na temat przypadków użycia – takie informacje są umieszczane np. na diagramie stanu czy aktywności.

Diagram składa się z

- przypadków użycia – ciąg akcji, z uwzględnieniem ich różnych wariantów, które mogą zostać wykonane podczas interakcji systemu z użytkownikiem,
- aktorów – mogą być to ludzie-użytkownicy, ale również inne systemy czy urządzenia,
- związków – powiązanie pomiędzy elementami diagramu.

6.2.2 Diagram sekwencji

Pokazuje w sposób zgodny z intuicją kolejność wywołanych operacji i przepływ sterowania pomiędzy obiektami.

Diagram zbudowany jest z

- prostokątów, które oznaczają obiekty,
- pionowych linii życia tychże obiektów,
- komunikatów wymienianych między obiektami.

Czas jest reprezentowany w postaci pionowej osi diagramu, a zajętość obiektu jest symbolizowana przez prostokąt umieszczony na jego linii życia.

Do komunikatów wymienianych między obiektami zalicza się między innymi

- wywołanie funkcji,

- powrót z wywołania,
- wywołanie asynchroniczne.

Na tym diagramie można również umieścić bloki, które przedstawiają specjalne interakcje (np. pętle, alternatywne przejście, sekcję krytyczną).

6.2.3 Diagram komunikacji

Jest to diagram podobny do diagramu sekwencji, jednak skupia się on na obiektach, które są składnikiem interakcji oraz komunikatów, które przesyłają. W odróżnieniu od diagramu sekwencji, nie skupia się na aspekcie czasowym. Rozmieszczenie obiektów jest kluczowe, aby w jasny sposób przedstawić relacje między sobą. Komunikacja jest przedstawiona za pomocą linii łączącej obiekty, natomiast przesyłane dane znajdują się ponad nimi (np. student zalicza egzamin).

7 K7 – Generowanie realistycznych obrazów scen 3-D za pomocą metody śledzenia promieni

Fotorealizm jest to metodyka starająca się w jak najlepszym stopniu oddać świat rzeczywisty za pomocą wygenerowanego obrazu. Aby to osiągnąć wykorzystuje się różne metody takie jak *fotogrametria* (łączenie zdjęć tego samego obiektu z wielu różnych perspektyw w celu wygenerowania obiektu 3-D - technologia użyta np. przy tworzeniu gry *The Vanishing of Ethan Carter*), metody próbkowania przestrzeni, czy też metoda śledzenia promieni.

Metoda próbkowania przestrzeni polega na analizowaniu toru promieni od źródła światła, poprzez odbicia od obiektów sceny, aby finalne trafić do oka obserwatora albo wylecieć poza scenę. Z każdego źródła światła wyprowadzany jest pęk promieni (**dyskretyzacja światła**), i im więcej promieni, tym dokładniejszy stanie się obraz. Tutaj pojawia się problem, gdyż generowana jest bardzo duża liczba promieni, z czego ogromna większość nie przecina nawet rzutni (nie dociera do oka obserwatora), przez co operacje wykonane zostały bezsensownie. Stąd też wymyślona została metoda śledzenia promieni rozwiązująca ten problem.

Metoda śledzenia promieni (ang. *Ray Tracing*) jest techniką służącą do generowania fotorealistycznych scen trójwymiarowych 3-D. Opiera się na śledzeniu wyłącznie tych promieni, które docierają do obserwatora. Cechą charakterystyczną jest to, że promienie nie są analizowane normalnym torem, tj. od źródła światła, bądź światła odbitego, do obserwatora, lecz właśnie **od oka obserwatora do elementów na scenie**. Zatem dla każdego piksela obrazu wynikowego wyprowadzany jest jeden promień, od którego w następstwie zależy wartość koloru tego piksela.

Wyprowadzony promień może nie trafić w żaden obiekt na scenie - piksel przyjmuje wtedy określony kolor tła. Promień może także trafić na źródło światła - piksel zyskuje kolor źródłowy światła. Promień może również trafić w jakiś obiekt na scenie. Jeśli trafi na taki obiekt, to wyznaczane są **punkty przecięcia**, następnie brany jest punkt przecięcia najbliższy początkowi promienia (gdyż punktów przecięcia dla obiektu może być kilka) i dla niego obliczany jest kolor punktu za pomocą wybranego modelu oświetlenia, na przykład popularnego **modelu Phonga**. Obliczane są w tym momencie także cienie używając pomocniczych promieni biegnących do źródła światła - gdy wiązka przetnie inny obiekt to oryginalny punkt jest zaciemniony. Całą procedurę można następnie powtarzać rekurencyjnie śledząc kolejne promienie odbite (zwane wtórnymi) i załamane tak, aby uzyskać efekt przedmiotów odbijających się w sobie nawzajem.

Przebieg działania algorytmu zapisywany jest w postaci **drzewa** (graf nieskierowany). Skrócony przebieg algorytmu przedstawić można w kilku krokach:

1. Dla każdego piksela obrazu wyprowadź **promień pierwotny**.
2. Dla każedgo napotkanego obiektu oblicz odbicia i wyprowadź **promienie wtóre**. Każdy punkt odbicia zapisz jako nowy węzeł drzewa.
3. Dla każdego węzła wyznacz oświetlenie lokalne korzystając z wybranego modelu oświetlenia.
4. Przechodząc od liści drzewa dodawaj kolejne wartości oświetlenia lokalnego ustalając tym samym ostateczną wartość piksela, od którego wyszedł promień pierwotny.

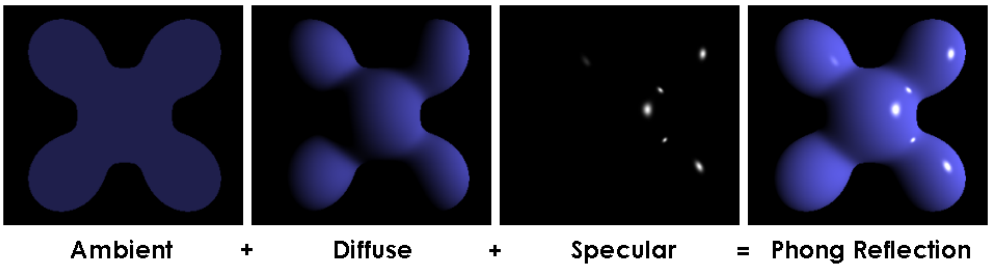
Algorytm kończy swe działanie w momencie, gdy:

- Promień nie trafia w żaden obiekt na scenie

- Promień trafia w obiekt całkowicie rozpraszający światło
- Promień trafia na obiekt, w którym następuje całkowite wewnętrzne odbicie
- Wyczerpana została ilość rekursywnych odbić

Model Phonga służy do wyznaczania oświetleń lokalnych. Na model ten składają się trzy rodzaje oświetlenia:

1. **Światło ambientowe** - światło otoczenia, jest to wartość stała przypisana do danej sceny.
2. **Światło rozproszone** - podstawowy kolor obiektu, wyliczany na podstawie **modelu Lamberta** (*Lambert, Lambert ty chuju* - Wiedźmin Geralt z Rivii) i widoczny przede wszystkim na powierzchniach matowych (drewno, papier).
3. **Światło odbite** - odpowiada za efekt połysku dla powierzchni śliskich, błyszczących (Passat metallic nówka nie śmigana trzy razy klepana). Wskazuje kierunek promienia odbitego.



Rysunek 3: Model Phonga

Owe składowe są mnożone przez procentowy współczynnik wpływu każdej składowej a następnie sumowane według poniższego wzoru:

$$I = k_a I_a + k_d I_d + k_s I_s \tag{5}$$

Gdzie:

- I – natężenie światła w punkcie
- I_a – (ang. *ambient*) natężenie światła otoczenia
- I_d – (ang. *diffuse*) natężenie światła rozproszonego
- I_s – (ang. *specular*) natężenie światła odbitego lustrzanie
- k – procentowy współczynnik wpływu składowych (przypisane do danego obiektu)

Prawo Lamberta mówi, że natężenie światła rozproszonego jest wprost proporcjonalne do kosinusa kąta pomiędzy wektorem promienia a wektorem normalnym do powierzchni. W skrócie - im mniejszy kąt, tym mniej światła jest rozproszone (powierzchnia skierowana prostopadle do promienia jest najjaśniejsza).

W modelu Phonga używany jest także dodatkowy efekt zwany **tłumieniem atmosferycznym**, polegające na spadku natężenia światła wraz z rosnącą odległością od obserwatora. Zjawisko to

można wykorzystać do generacji realistycznych warunków pogodowych, takich jak deszcz, zamieć, mgła.

Metoda śledzenia promieni mimo, że daje zdumiewające rezultaty to ma w sobie kilka poważnych wad:

- Jest **bardzo kosztowna obliczeniowo** przez dużą liczbę obliczeń potrzebnych do wyliczenia koloru każdego piksela. Najbardziej kosztowne jest liczenie kolizji, co optymalizować można np. algorytmem brył otaczających, czyli wpisaniem zaawansowanych brył w większe, prostsze kształty - jeśli nie wystąpi kolizja z prostym kształtem, to na pewno nie nastąpi także z tym zaawansowanym.
- Może występować **zjawisko aliasingu**, które sprawia, że niektóre małe obiekty są pomijane, a obiekty o ostrych krawędziach mogą być zniekształcone (poszarpane). W celu uniknięcia tego problemu stosowanych jest wiele metod **antyaliasingu** na przykład nadpróbkowanie (*supersampling - SSAA*), gdzie jeden promień pierwotny zastępowany jest **wiązką promieni** i kolor piksela jest zazwyczaj średnią z tej wiązki
- Przyrost ilości źródeł światła i obiektów znacznie pogorsza czas renderowania
- Nie wszystkie kierunki padania promieni są rozpatrywane
- Każdą klatkę trzeba generować od nowa - nie można wykorzystać informacji z poprzednich klatek

Wartym zaznaczenia jest fakt, że **śledzenie promieni dla każdego z pikseli odbywa się niezależnie** od innych pikseli, także metodę tę można **zrównoleglić** (zarówno dla grupy pikseli bądź promieni) na przykład wykorzystując zestawy instrukcji MMX oraz SSE z architektury SIMD równoległące te same operacje arytmetyczne na liczbach całkowitych w procesorze, bądź używając do tego celu dobrodziejstw najnowszych procesorów graficznych - na przykład technologii NVIDIA CUDA, która pozwala na otrzymanie obrazu generowanego tą metodą w czasie rzeczywistym.

8 K8 – Mechanizmy systemu operacyjnego wspomagające synchronizację procesów

System operacyjny jest oprogramowaniem, którego jedną z głównych ról jest synchronizacja procesów. Czym jest proces? Jest to wykonujący się program, czyli zbiór instrukcji dla procesora przechowywany w postaci pliku.

Każdy proces wymaga przydziału zasobów komputera. Czas procesora, pamięć, dostęp do urządzeń – w mniejszym lub większym stopniu, proces zawsze wymaga pewnych zasobów.

Procesy mogą wykonywać się zupełnie niezależnie od siebie – korzystając np. z innych urządzeń – lub współbieżnie obok siebie korzystając z np. tych samych plików.

Jak łatwo się domyślić, procesy współbieżne wymagają synchronizacji ze strony zarządcy zasobów – systemu operacyjnego.

Typowe problemy związane z procesami współbieżnymi to na przykład:

- problem sekcji krytycznej – kiedy procesy współdzielą strukturę danych, a operacje na nich muszą być atomowe, tj. nie mogą zostać przerwane w środku wykonywania się,
- problem czytelników i pisarzy – synchronizacja dostępu do zasobów dla procesów dokonujących i niedokonujących w nich zmian,
- problem producenta i konsumenta – przetwarzanie danych przez proces może się odbyć jedynie, po otrzymaniu ich od innego procesu,
- problem ucztujących filozofów – procesy korzystają ze wspólnych zasobów, które pobierają i zwalniają wg potrzeb.

Sekcja krytyczna to sekwencja operacji wykonywanych na zasobie (np. pamięci, pliku), które muszą zostać wykonane w trybie wyłącznym przez jeden proces, tj. podczas ich wykonywania, inny proces nie może działać na zasobach przez niego zajętych. Aby poprawnie rozwiązać problem sekcji krytycznej, należy zaimplementować algorytm, który będzie spełniał kilka warunków:

- instrukcje w sekcji krytycznej nie mogą być przeplatane – może w niej być wyłącznie jeden proces,
- nie można zakładać w jakiej kolejności i z jaką szybkością wykonają się dane procesy,
- proces nie może zatrzymać się w sekcji krytycznej,
- nie mogą występować zakleszczenia (jeden proces czeka na zwolnienie zasobów przez drugi, a drugi czeka na zwolnienie zasobów pierwszego, więc obydwa się zatrzymują),
- każdy proces musi wejść do sekcji krytycznej (nie może wystąpić zagłodzenie).

Istnieją przeróżne mechanizmy synchronizacji implementowane przez jądro.

Semafory są typem danych, które służą do kontroli dostępu zasobów przez wiele procesów. Semafory są zmienną całkowitą, która przyjmuje wartości nieujemne (tj. $\geqslant 0$) lub dla semaforów binarnych – wartości logiczne. Na semaforach można wykonywać dwie operacje – zmniejszenie (zajęcie, podniesienie,) oraz zwiększenie (zwolnienie, opuszczenie). Synchronizacja polega na blokowaniu procesu w operacji zajęcia semafora, gdy jego wartość po zajęciu jest ujemna, do czasu, aż wartość

ta nie zostanie zwiększona do nieujemnej (np. przez inny proces, który zwolni semafor). Wyróżnia się rodzaje semaforów takie jak:

- binarne – zmienna przyjmuje tylko wartości *true* (semafor otwarty) lub *false* (semafor zamknięty),
- zliczające – zmienna przyjmuje wartości całkowite nieujemne, a aktualna wartość jest zwiększana/zmniejszana o 1 w wyniku zwolnienia/zajęcia semafora,
- uogólnione – semafor zliczający, ale może zwiększać/zmniejszać wartość o dowolną liczbę (oczywiście, wartość wciąż musi być nieujemna).

Muteksy (`mutual exclusion` – wzajemne wykluczanie) są szczególnym przypadkiem semaforów. Muteks obsługują operacje blokowania i zwalniania, jednak mogą być zwolnione tylko przez proces, które je zajął. #FIXMEDzięki temu wyeliminowane jest zjawisko zakleszczenia oraz niespójności danych.

Spinlocki – podobne do semaforów, jednak oczekiwanie na zwolnienie blokady odbywa się na zasadzie aktywnego czekania, przez co zajmują czas procesora, może wystąpić zagłodzenie lub czekać w nieskończoność.

Monitory są strukturalnym narzędziem synchronizacji wątków. Składają się ze zmiennych oraz procedur operujących na nich, zebrane w jeden moduł. Dostęp do zmiennych jest możliwy wyłącznie za pomocą procedur monitora, a w danej chwili tylko jeden proces może wywoływać procedury monitora. Gdy inny proces wywoła procedurę monitora, to będzie on zablokowany do chwili opuszczenia monitora przez pierwszy proces. Istnieje możliwość wstrzymania i wznowienia procedur monitora za pomocą zmiennych warunkowych. Na zmiennych warunkowych można wykonywać operacje `wait` (wstrzymanie procesu i umieszczenie go na końcu kolejki) oraz `signal` (odblokowanie jednego z oczekujących procesów). Procesy oczekujące na wejście do monitora zorganizowane są w kolejkę FIFO.

9 K9 – Programowalne scalone układy cyfrowe PLD, CPLD oraz FPGA

Programowalne układy scalone to układy scalone, których funkcjonalność jest definiowana przez użytkownika końcowego, a nie producenta. Wszystkie wyprodukowane przez producenta układy są identyczne (co pozwala zmniejszyć koszty). Pozwala to na zaprojektowanie, uruchomienie i testowanie urządzenia. Programowanie odbywa się za pomocą tworzenia połączeń w istniejącej sieci ścieżek sygnałowych.

Układy programowalne można podzielić ze względu na ich strukturę:

- PLD – (programmable logic device) najprostsze,
- CPLD – complex PLD,
- FPGA – field-programmable gate array.

Układy PLD mogą realizować funkcje logiczne (tworząc układy kombinacyjne bądź sekwencyjne). Programowanie takich układów bazuje na ustawieniu odpowiednich bitów, by zostały zrealizowane dane funkcje. Każde urządzenie posiada dany zbiór wejść oraz wyjść do układu.

Najprostsze układy PLD (SPLD) pozwalają na tworzenie prostych układów. Do nich możemy zaliczyć układy PLE, PAL, PLA. Ich budowa opiera się na matrycach funkcji AND oraz OR – realizują funkcje postaci $(x_1 \wedge x_2 \wedge ...) \vee (x_m \wedge x_{m+1} \wedge ...) \vee ...$ lub $(x_1 \vee x_2 \vee ...) \wedge (x_m \vee x_{m+1} \vee ...) \wedge ...$ (dla sklerotyków: $\wedge$ – AND, $\vee$ – OR).

Układy PLA posiadają programowalne matryce AND oraz OR. Dowolna linia iloczynu logicznego z matrycy AND (tzw. *term*) może zostać podłączony do realizacji dowolnej operacji sumy logicznej OR. Inna architektura, PAL posiada programowalną matrycę AND oraz stałą (nieprogramowalną) matrycę OR. Natomiast układy PLE posiadają stałą matrycę bramek AND oraz programowalną matrycę bramek OR.

Ze względu na ograniczone możliwości logiczne – małą liczbę bramek oraz małą liczbę wejść/wyjść, układy SPLD mogły zastępować klasyczne obwody logiczne (czyli coś typu jak kleciliśmy na LUC-u).

Układy CPLD, jak nazwa wskazuje są bardziej złożone. Koncepcyjnie są podobne do SPLD, ale mają większe możliwości logiczne i funkcjonalne. Zbudowane są z zespołu struktur PAL połączonych ze sobą programowalną matrycą (switching matrix). Blok funkcjonalny CPLD posiada matrycę AND, makrokomórki oraz zadaną liczbę wyjść. Makrokomórki składają się zazwyczaj z bramek (np. AND, OR) oraz przerzutników. Sygnały sterujące makrokomórkami można podzielić na globalne (wspólne dla wszystkich makrokomórek), lokalne (wspólne dla zespołu połączonych ze sobą makrokomórek) oraz indywidualne (wpływają na działanie jednej makrokomórki). Pojedyncza makrokomórka realizuje prostą funkcję logiczną, większa ich liczba może być połączona ze sobą w bloki funkcyjne, tworząc funkcje z większą liczbą zmiennych.

Najbardziej skomplikowane są układy FPGA. Mają one jeszcze większe możliwości logiczne w stosunku do CPLD oraz są szybsze w działaniu. Zbudowane są one z konfigurowalnych elementów logicznych (CLB). W skład elementu logicznego wchodzą zazwyczaj generator funkcji logicznych, przerzutnik i programowalne multipleksery. Generator funkcji logicznych określany jest jako LUT (look up table). Elementy logiczne są łączone w bloki. Połączenia między CLB są programowalne, podobnie jak w układach CPLD za pomocą programowalnej matrycy. LUT-y mają inne podejście do

generowania funkcji – zamiast programowania połączeń między bramkami, LUT-y działają bardziej jak tabele prawdy – dla zadanego wejścia generują odpowiednie wyjście — **to jest znacząca różnica między FPGA a CPLD**.

Do zalet struktur FPGA można zaliczyć elastyczność architektury (równoległość przetwarzania, dowolną szerokość ścieżki danych), wielokrotne użycie tych samych zasobów sprzętowych, czy też rekonfigurowalność.

Do programowania programowalnych układów scalonych wykorzystywane są języki opisu sprzętu, takie jak VHDL lub Verilog. Opis układu jest tworzony w sposób behawioralny. Narzędzie syntezy przetwarza taki zapis na konkretną realizację sprzętową. W przeciwieństwie do mikrokontrolerów, układy potrafią przetwarzać równolegle, a μC przetwarzają sekwencyjnie. Nie są one ograniczone zbiorem funkcji – μC posiadają określony zestaw instrukcji. Wadą układów programowalnych w stosunku do nieprogramowalnych jest większy pobór mocy. Na układach programowalnych FPGA można zaprogramować procesor programowy – PicoBlaze, MicroBlaze.

Układy PLD mają szerokie spektrum zastosowań. Można do tego zaliczyć logikę scalającą – interfejs dla mikrokontrolerów umożliwiający współpracę z innymi modułami (pamięć, układy peryferyjne). Poprzez możliwość przetwarzania równoległego, układy PLD nadają się jako akceleratory sprzętowe. Akceleratory są wykorzystywane przy przetwarzaniu grafiki, dźwięku lub wideo.

10 K10 – Optyczne nośniki informacji

Dyski optyczne zaliczają się do nośników informacji. Zostały wprowadzone jako alternatywa dla dysków magnetycznych. Odczyt danych z dysku optycznego odbywa się za pomocą wiązki światła (a konkretniej promienia lasera) – nazwa opiera się na zasadzie działania takiego nośnika.

W dzisiejszych czasach nośniki optyczne mają różne pojemności, stosowane są w różnych dziedzinach (np. na nich dystrybuowana jest muzyka, czy też filmy).

Do dysków optycznych zaliczamy:

LaserDisc – analogowy dysk optyczny, który miał 30 cm średnicy, składał się z dwóch jednostronnych aluminiowych dysków pokrytych plastikiem. Na nośniku był zapisywany analogowy sygnał wideo oraz analogowy i/lub cyfrowy sygnał dźwiękowy. Posiadał formaty kodowania:

- CAV – stała prędkość kątowa, dyski pracowały ze stałą prędkością kątową 1800rpm, z odczytem jednej klatki na obrót, pojemność wynosiła 30 min na stronę, CAV posiadała funkcje stop klatki, zmiany prędkości odtwarzania oraz cofanie,
- CLV – stała prędkość liniowa, nie posiada funkcji z CAV, przez zmniejszenie prędkości obrotowej zwiększyła się pojemność do 60 min obrazu lub dźwięku,
- CAA (wariant CLV) – stałe przyspieszenie kątowe, wynalezione by wyeliminować zjawisko wchodzenia jednego kanału na drugi z CLV, różnica między CAA a CLV polega na dopasowaniu kąta obrotu dysku, zamiast stopniowego liniowego zwalniania, dzięki CAA znacznie poprawiła się jakość obrazu na nośniku,

CompactDisk – początkowo był stworzony tylko do przechowywania muzyki, później dodano możliwość przechowywania danych. Płyta CD ma średnicę 12 cm i może pomieścić do 80 min dźwięku lub 700 MB danych. Istnieje również format mini-CD, jednak jest on mniej popularny (do 24 min dźwięku, 210 MB pojemności; inny format – „wizytówka" CD – do 6 min dźwięku, do 65 MB pojemności). Formaty CD:

- Audio-CD – format to dwukanałowy 16-bitowy PCM, z próbkowaniem 44.1 kHz na kanał, dźwięk mono nie występuje w implementacji, zazwyczaj dźwięk mono jest przedstawiony jako dwa takie same kanały w stereo. CD-Text jest rozszerzeniem Audio-CD, zezwala na zachowanie informacji tekstowych (np. nazwa albumu, artysty, ścieżek).
- Video-CD – cyfrowy standard do przechowywania wideo na płytach CD. W porównaniu z kasetami VHS jakość wideo nie pogarsza się po każdym użyciu. Wykorzystywane rozdzielczości to 352x240 oraz 352x288.
- CD-R – płyta może zostać raz zapisana, posiada warstwę barwnika, który jest stapiany przy zapisywaniu,
- CD-RW – płyta do zapisu wielokrotnego, kasowanie danych odbywa się za pomocą lasera (następuje topnienie stopu, traci swoją strukturę polikrystaliczną).

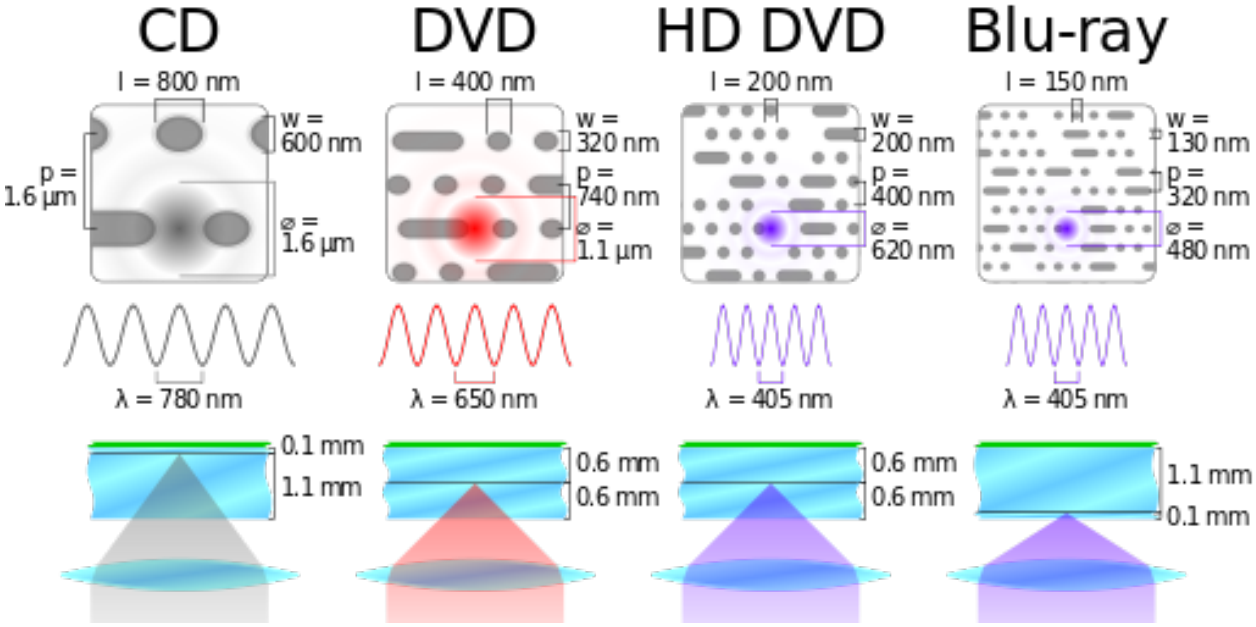
DVD – medium do przechowywania różnego typu danych, zazwyczaj do oprogramowania oraz filmów. Nośniki posiadają takie same wymiary jak płyty CD, jednak można na nich zapisać więcej danych. Zwiększenie danych jest możliwe dzięki zwiększeniu gęstości zapisu danych (użyto lasera o krótszej długości fali). Powstały również płyty dwustronne oraz dwuwarstwowe. Podobnie jak w przypadku płyt CD, istnieją tutaj płyty jednokrotnego oraz wielokrotnego zapisu. Płyty DVD

również posiadają formaty Audio oraz Video. Płyty audio posiadają dużą możliwość konfiguracji kanałów, z różnym próbkowaniem (nawet do 24-bit, 192 kHz). W porównaniu do CD-Audio pozwala zapisać więcej dźwięku w lepszej jakości. Zapis wideo na płytach DVD-Video wykorzystuje kodek MPEG-2. Najczęściej stosowane rozdzielczości to 720x576 oraz 720x480. Następcą DVD jest HD DVD, który zawierał gęstszy zapis danych, jednakże został porzucony na rzecz płyt Blu-Ray (wykorzystują tę samą długość fali lasera).

Dostęp do drugiej warstwy możliwy jest przez przepuszczenie lasera przez pierwszą, częściowo przepuszczalną warstwę. Zmiana ścieżki może potrwać kilka sekund, przez co może spowodować chwilowe zatrzymanie pracy napędu.

Blu-ray jest kolejnym formatem dysków optycznych. Dyski posiadają 25 GB na warstwę, przy zapisie dwu-warstwowym daje to 50 GB. W 2008 roku przedstawiono szesnasto-warstwową płytę BD, o pojemności 400 GB. Istnieją również formaty trzy- i cztero-warstwowe płyt BD. Nazwa technologii wywodzi się od koloru wykorzystywanego lasera.

Rysunek 4: Porównanie formatów płyt. Źródło: Wikipedia

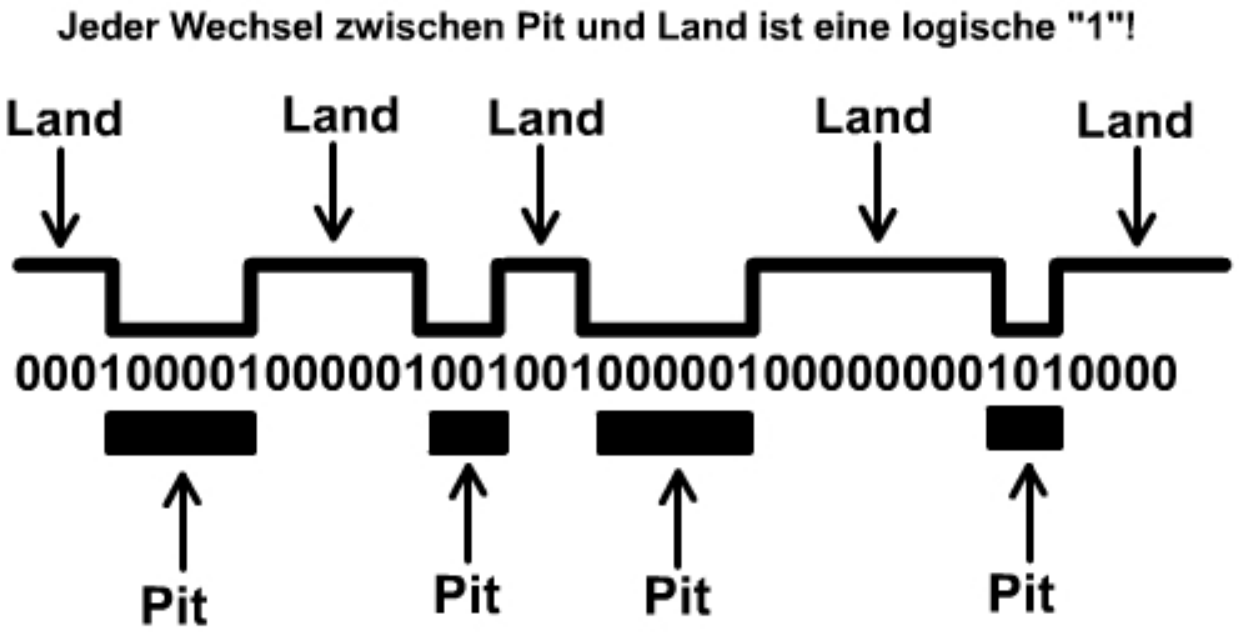


HVD (Holographic Versatile Disc) – technologia dysków optycznej nowej generacji, teoretycznie pozwala pomieścić do 6 TB na jednej warstwie. Zapis danych odbywa się w trójwymiarowej przestrzeni dysku. Wykorzystywane są dwa lasery – zielony oraz czerwony.

NRZI – metoda odwzorowania sygnału binarnego, bez powrotów do zera. Wykorzystuje dwa stany logiczne. Przejście między nimi następuje, gdy transmitowany bit wynosi 1, w przypadku bitu 0 zmiana nie następuje.

Stany logiczne na płycie CD są reprezentowane przez pity (wgłębienia) oraz landy (płaskie powierzchnie, przerwy między wgłębieniami). Gdy wiązka trafia na pit, to jest rozpraszana i nie wraca do czytnika. Przy landzie wiązka jest odbijana i trafia z powrotem do czujnika. Każda przejście z landu na pit o odwrotnie powoduje zmianę stanu (logiczne 0 lub 1). Dane zapisywane są od środka na zewnątrz płyty.

Rysunek 5: Stany logiczne odczytywane z powierzchni płyty



Kodowanie wykorzystywane w płytach to EFM (eight to fourteen modulation). Jest wykorzystywane ze względu na ograniczenia technologiczne (szybkości światła i odpowiedzi modulatora). Przez to długości pitów i landów muszą znajdować się w przedziale od 3 do 11 bitów kanałowych (ciąg kilku pitów/landów nie może być krótszy niż 3 bity kanałowe oraz dłuższy niż 11). Dlatego każde 8 bitów jest zapisywane za pomocą 14 bitów kanałowych. Do konwersji jest wykorzystywana tabela, która jest zapisana w firmware napędu.

Dla płyt DVD, ze względu na mniejsze odległości stosowany jest zapis bajtów na 16 bitowych sekwencjach. Długości pitów i landów muszą być w przedziale od 3 do 14 bitów kanałowych.

Płyty Blu-ray wykorzystują kodowanie podobne do EFM, 17PP (one seven parity preserving). Wykorzystuje kodowanie, w którym każde 2 bity są zapisywane na 3 bitach, a samo kodowanie odbywa się w grupach po 2 albo 4 bity. Dodatkowo występują bity kontrolne (DC), po których wystąpieniu (jeśli DC == 1), zmieniana jest polaryzacja strumienia NRZI.

A teraz z innej beczki... Innym optycznym nośnikiem informacji są **kody kreskowe**. Skanery odczytują zapisane dane. Mogą być wydrukowane na różnych powierzchniach. Składają się z jasnych oraz ciemnych elementów. Istnieje wiele rodzajów kodów kreskowych, można je pogrupować według zastosowania, np. identyfikacja towarów (EAN, Code 11), kody pocztowe (POSTNET, PLANET, etc.). Istnieją również dwuwymiarowe kody kreskowe – składają się one z macierzy elementów (jasnych/ciemnych). Do nich można zaliczyć kody takie jak: kody QR, czy Aztec Code (wykorzystywany w polskich dowodach rejestracyjnych).

Ten temat to temat rzeka, można opowiedzieć o wielu rzeczach (IrDA, światłowody i pewnie wiele innych). Ostatnim rodzajem, o którym chcę wspomnieć to dyski magnetooptyczne. Są stworzone z tworzywa sztucznego oraz warstwy materiału magnetycznego, zabezpieczone w kasecie, chroniąc przed uszkodzeniami mechanicznymi. Podobnie jak w wypadku płyt omówionych wcześniej, istnieją dyski magnetooptyczne zapisywane jedno- oraz wielokrotnie. Odczyt danych z takiego

dysku wykorzystywane jest zjawisko Kerra (zmiana współczynnika załamania światła przez pole magnetyczne). Przy odbiciu lasera od namagnesowanego miejsca następuje „skręcenie" w kierunku zależnym od pola magnetycznego i ta zmiana jest wykrywana przez głowicę napędu i traktowana jako stan logiczny.

Część II

Pytania specjalnościowe – INT

11 S1 – Tryby komunikacji między procesami w standardzie Message Passing Interface

MPI stanowi standard przesyłania komunikatów pomiędzy procesami programów równoległych działających na jednym lub więcej komputerach. Pracuje natywnie z C, C++, Fortranem. Istnieją rozwiązania dla innych języków np. Python, lecz stanowią tylko nakładki na wyżej wymienione. Dostępne wersje biblioteki:

- Wersja 1.2 (MPI-1) – wprowadza około 150 funkcji;
- Wersja 2.1 (MPI-2) – rozszerzona o nową funkcjonalność, razem ponad 500 funkcji

Standard/biblioteka występują w wielu implementacjach. Najpopularniejsze z nich to: MPICH / MPICH2, OpenMPI, Intel MPI, HP MPI.

Informacje podstawowe:

- program lub jego części mogą wykonywać się jednocześnie na więcej niż jednej jednostce obliczeniowej;
- jednostka obliczeniowa: procesor, rdzeń procesora, wątek sprzętowy, wątek systemowy, inne;
- **Komunikatorem** nazywam zbiór procesów mogących się wzajemnie komunikować;
  - `MPI_Comm_size(MPI_COMM_WORLD, &np)` – zwraca listę procesów wewnątrz komunikatora;
- Tak samo jak w przypadku procesów w systemie operacyjnym wewnątrz komunikatora każdy proces otrzymuje unikalny identyfikator (numer) tzw. **rank**;
  - `MPI_Comm_rank(MPI_COMM_WORLD, &rank)` – zwraca rank aktualnego procesu;
- W czasie inicjacji programu biblioteka tworzy domyślny komunikator o nazwie `MPI_COMM_WORLD` zawierający wszystkie dostępne procesy
- Programista może definiować własne komunikatory zawierające zbiory procesów.
- **Komunikat** – paczka danych przesyłana pomiędzy procesami przez łącze / sieć łącząca procesory w ramach komunikatora. Paczka składa się z ustalonej liczby elementów określonego typu.
- Wywołania wszelkich funkcji i procedur biblioteki MPI muszą znajdować się pomiędzy instrukcjami:
  - `MPI_Init(&argc, &argv)` - λ Inicjuje działanie biblioteki i domyślnego komunikatora (sesję);
  - `MPI_Finalize(void)` - λ Kończy działanie sesji MPI;
  - Powyższe funkcje muszą wywołać wszystkie procesy w sesji;

- MPI definiuje własne typy danych, które powinny być odpowiednikami typów w C, np. `MPI_INT` – int, Dodatkowo `MPI_BYTE` oraz `MPI_PACKED`

Podstawowe tryby komunikacji:

- Komunikacja **point to point**;
- Komunikacja **kolektywna** (jak ktoś zapomni można powiedzieć grupowa).

Aby przesłać paczkę należy zdefiniować:

- Identyfikator (rank) procesu wysyłającego
- Identyfikator (rank) procesu odbierającego
- Adres źródłowy i adres docelowy
- Typ i wielkość danych przesyłanych w paczce
- Komunikator

Komunikacja **point to point** to najprostsza możliwa forma. Proces wysyła komunikat do innego procesu. Nadawca wysyła komunikat funkcją **Send** natomiast odbiorca wywołuje funkcję **Receive**. Podstawowe funkcje służące do wymiany informacji to `MPI_Send` / `MPI_Recv`, które są funkcjami blokującymi tzn. blokują dalsze wykonanie programu do czasu zakończenia komunikacji.

Komunikacja **point-to-point** składa się z różnych trybów. Zależą one od funkcji wykorzystanych do przesyłania komunikatów. Tryby:

- Standardowy - `MPI_Send`;
- Synchroniczny - `MPI_Ssend`;
- Asynchroniczna / buforowana - `MPI_Bsend`;
- Ready - `MPI_Rsend`;
- różnią się tym jak restrykcyjna ma być sychronizacja;
- Wszystkie tryby są blokujące – wysyłanie kończy się w momencie gdy użytkownik może bezpiecznie odczytać i modyfikować bufor wejściowy (tablicę zawierającą komunikat)

Tryb wysyłania	**Działanie**
Standardowy `MPI_Send`	Wysyłanie blokujące, biblioteka decyduje o użyciu wewnętrznego bufora
Synchroniczny `MPI_SSend`	Pełna synchronizacja, nie zakłada wcześniejszego wywołania `MPI_Recv`, zakończy wykonanie, gdy odbiorca rozpocznie odbieranie
Buforowany `MPI_BSend`	Korzysta z bufora użytkownika, nie wymaga wcześniejszego wywołanie `MPI_Recv`, zakończy wykonanie po umieszczeniu wiadomości w buforze. Wymaga zapewnienia przez użytkownika bufora odpowiedniej wielkości. `MPI_Buffer_attach` tworzy bufor, `MPI_Buffer_detach` zwalnia bufor. W przypadku przepełnienia bufora nastąpi błąd.
Ready send `MPI_RSend`	Zakłada wcześniejsze wywołanie `MPI_Recv`. Zakończy się błędem, jeżeli nie wystąpi wczesniej odpowiednie wywołanie `MPI_Recv`. Potencjalnie niebezpieczne, niezalecane do użycia.

- Odbieranie `MPI_Recv` działa ze wszystkimi trybami wysyłania

Komunikacja **kolektywna** to taka, w której uczestniczy grupa procesów MPI. Dana funkcja wywoływana jest przez wszystkie procesy w komunikatorze.

W MPI wyróżniamy następujące typy komunikacji grupowej:

- Bariera synchronizacyjna
- operacja Broadcast – nadaj wiadomość wielu procesom
- Operacja typu rozrzuć (z ang. scatter)
- Operacja typu zbierz (z ang. gather)
- Operacja wszyscy do wszystkich
- Operacja redukcji (np. globalna suma, globalne maksimum, ...)

Podobnie jak w przypadku komunikacji point-to-point. W tym przypadku, także z odpowiednim typem komunikacji związana jest funkcja MPI.

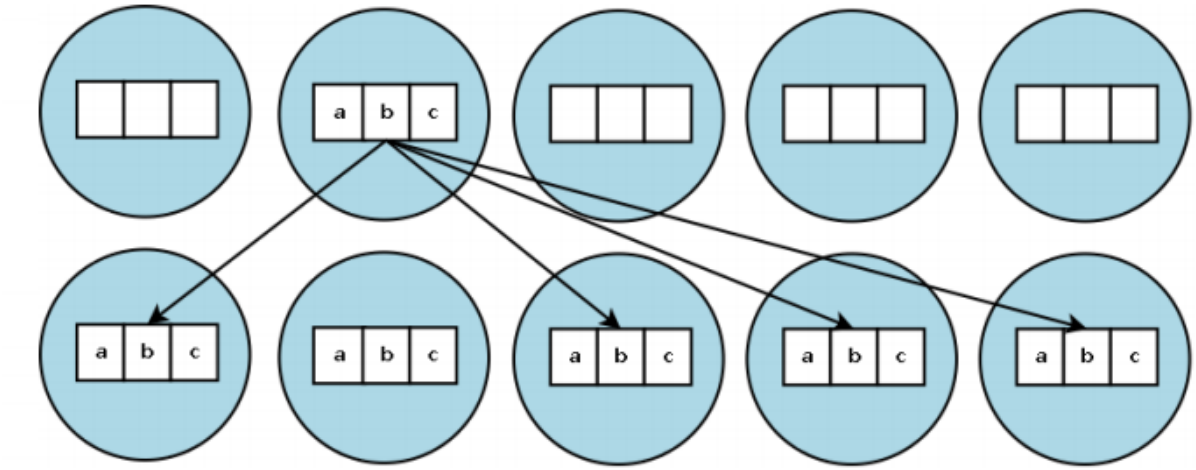
Bariera synchronizacyjna jak sama nazwa wskazuje służy do synchronizacji:

```
int MPI_Barrier(MPI_Comm comm);
```

Wywołanie jej w pewnym miejscu programu powoduje, będzie on czekał, aż wszystkie pozostałe jego instancje dojdą do tego miejsca i dopiero potem ruszy dalej.

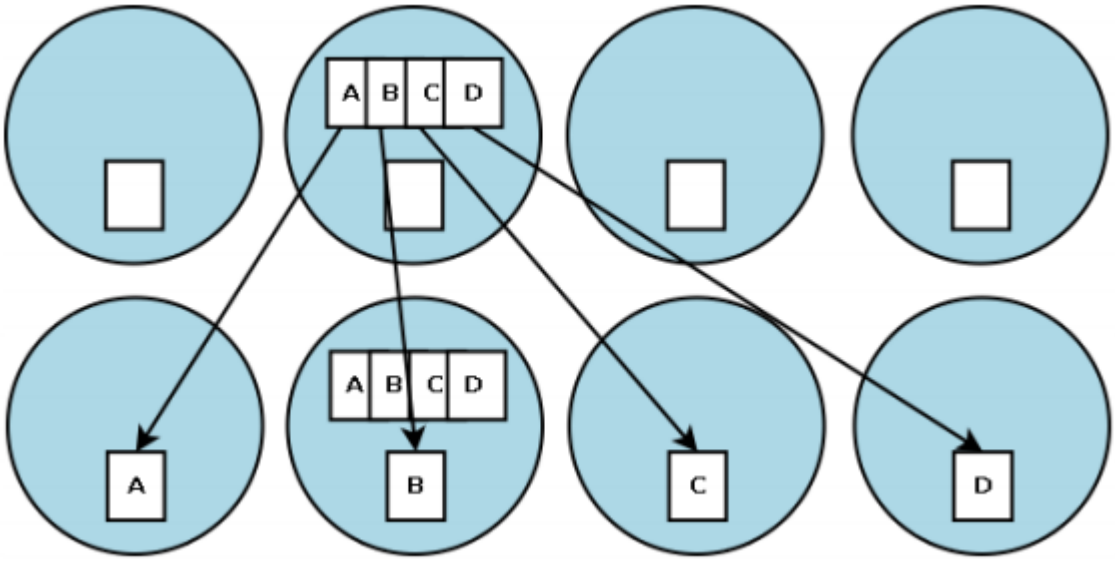
Operacja **Broadcast** polega na rozesłaniu tej samej wiadomości do wszystkich procesów grupy.

Rysunek 6: Brodcast



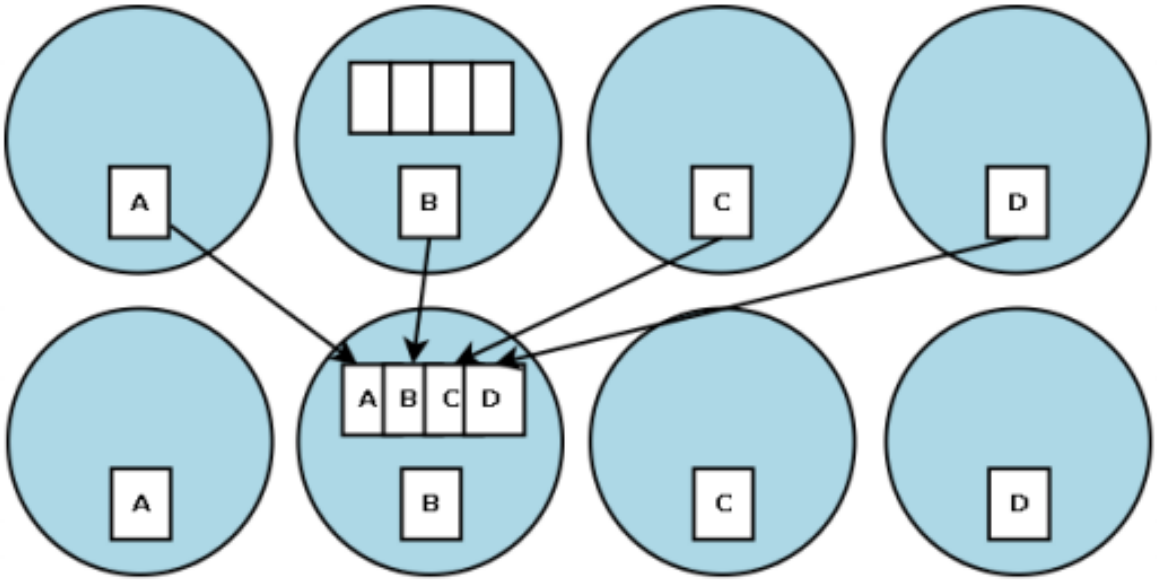
Operacja `MPI_Scatter` polega na rozesłaniu przez proces root wektora sendbuf podzielonego na kawałki sendcount do procesów w grupie (również do siebie):

Rysunek 7: Scatter



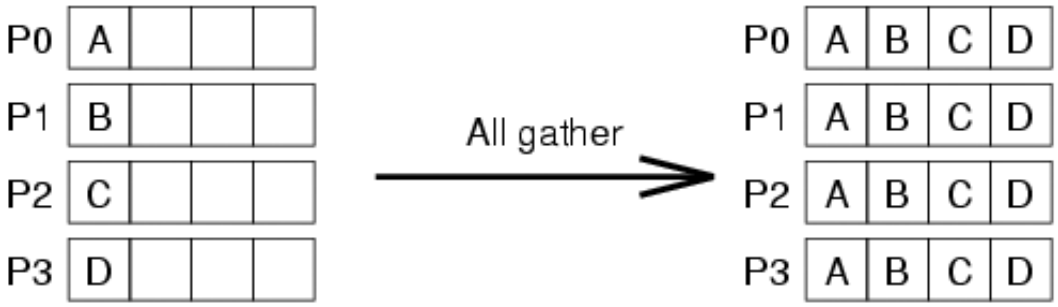
Operacja **MPI_Gather** odwrotna do scatter, czyli zebranie kawałków wektora z procesorów w grupie i zapisanie ich w buforze wybranego procesora.

Rysunek 8: Gather



Operacja **MPI_AllGather** jest analogiczna do **MPI_Gather** z tą różnicą, że wynik jest umieszczany w **recv_buff** każdego procesu. Można tą funkcję traktować jako ciąg kolejnych wywołań **MPI_Gather** każdorazowo z innym numerem procesu **root**

Rysunek 9: AllGather



Operacja wszyscy do wszystkich (`MPI_Alltoall`). Każdy z procesów wysyła do procesu i-tego sendcount kolejnych elementów typu sendtype, poczynając od elementu i*sendtype tablicy sendbuf.

Operacja redukcji (`MPI_Reduce`) służy do wykonywania globalnej operacji na elementach procesów należących do grupy. Pozwala wykonać na przykład sumowanie wszystkich częściowych wyników otrzymanych w procesach i umieszczenie wyniku w zmiennej. Przykładowe operatory to `MPI_MAX`, `MPI_MIN`, `MPI_SUM` - kompletna lista znajduje się w dokumentacji. Istnieje również możliwość definiowania własnych operatorów dla funkcji `MPI_Reduce()`.Warto wspomnieć o funkcji `MPI_AllReduce`. Funkcja identyczna z poprzednią, różniąca się jedynie tym, że po jej wykonaniu wynik agregacji z użyciem operatora `op` znajduje się w zmiennej `result` we wszystkich procesach.

12 S2 – HTML DOM i XHTML – cel i charakterystyka

12.1 HTML (HyperTextMarkup Language)

Cel:

- Zapewnienie uniwersalnego, przenaszalnego, niezależnego od platformy sprzętowej i systemu operacyjnego standardu opisującego struktury strony internetowej;
- Nadanie elementom strony odpowiedniego znaczenia semantycznego strony w zależności od ich zawartości np. tag `<em>`;
- Umożliwienie pozycjonowania strony w wyszukiwarkach internetowych w zależności od zawartości strony np. tag `<article>`

Charakterystyka:

Mianem HTML określamy standard języka znaczników używanego do tworzenia stron internetowych. Znaczniki są interpretowane przez przeglądarkę internetową i na ich podstawie w połączeniu ze stylami CSS oraz skryptami najczęściej napisanymi w języku JavaScript renderowana jest strona internetowa. Język jest niezależny od platformy sprzętowej oraz systemu operacyjnego, natomiast wymaga posiadania przeglądarki internetowej wspierającej standard w, którym napisano stronę. Znaczniki są słowami kluczowymi otoczonymi nawiasami ostrokątnymi. Zazwyczaj występują parami tzn. Istnieje tag otwierający i zamykający np.

```
<p>This is some text in a paragraph.</p>
```

Tag zamykający zawiera / po otwarciu < . Występują, także tagi nie wymagające zamknięcia np.

```
<img src=śmiley.gifąlt=Śmiley face"height="42"width="42»
```

Tagi mogą posiadać atrybuty np. src w przykładzie wyżej. Atrybuty dzielą się na obowiązkowe i nieobowiązkowe. Atrybut obowiązkowy jest wymagany do prawidłowego działania danego tagu. Przykładowo, jeżeli nie podamy atrybutu src tagowi img nasz obraz nie wyświetli się. Natomiast atrybuty nieobowiązkowe nie są wymagane do prawidłowego działania tagu np. kiedy nie podamy atrybutu height zostanie on zastąpiony wartością domyślną. Często atrybuty zastępowane są wartościami ustawianymi dla danego tagu w języku CSS. Przyjmuje się konwencje, że nazwy znaczników pisane są małymi literami. Co ciekawe jeżeli spojrzymy na pierwszą stronę napisaną w tym języku były one pisane wielkimi literami.

HTML nie jest językiem restrykcyjnym. Kiedy projektant strony zapomni o zamknięciu jakiegoś tagu wówczas strona wyświetli się w przeglądarce, nie zostanie wyświetlony żaden błąd, a strona prawdopodobnie zostanie wyświetlona poprawnie zgodnie z oczekiwaniami. Oczywiście w przypadku nieprawidłowego kodowania dokumentu html wyświetlanie strony zależy od silnika danej przeglądarki. Obecnym standardem języka jest wersja 5 opublikowana oficjalnie w 2014 roku przez firmę W3C zajmującą się standaryzacją tego języka. W historii standaryzacją zajmowały się IETF oraz ISO. O standardzie strony informuje nas nagłówek zawarty w pierwszej linii pliku z rozszerzeniem .html. Najnowszy standard oznaczony jest jako `<!DOCTYPE html>`

W przypadku standardów zachowano kompatybilność wsteczną.

HTML 5 wprowadził wiele nowości w porównaniu do wersji 4 opublikowanej w 1997 roku. Są nimi nowe tagi np. `<article>`, `<aside>` mające w lepszy niż sposób niż dotychczas opisywać semantykę strony. Atrybuty możemy podawać otoczne nawiasami oraz ' ', a także bez nawiasów. Dodano tagi związane z przesyłaniem mediów jak <video> oraz <audio> niewymagające korzystania z dodatkowych bibliotek i wtyczek jak Silverlight lub Flash. Dodaje także funkcjonalności niezwiązne z tagami jak:

- Local storage `http://www.w3schools.com/html/html5_webstorage.asp`
- Workery działające w tle `http://www.w3schools.com/html/html5_webworkers.asp`

Typowa struktura znaczników strony HTML:

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

Stronę otacza tag `html` w nim zawarte są dwa podstawowe tagi `head` i `body`. `Head` zawiera metadane dotyczące strony: język, kodowanie, tytuł. Często umieszcza się dam odwołania do plików ze stylami css i skryptami js. W tagu body zawarte są wszystkie tagi związane z zawartością strony.

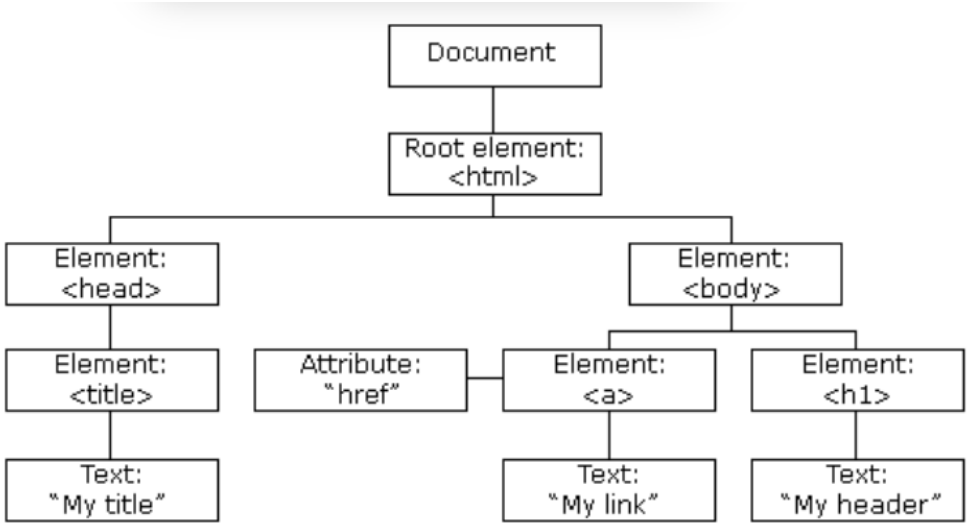
12.2 DRZEWO DOM (DOKUMENT OBJECT MODEL)

Cel:

- Dostarczenie prostej reprezentacji hierarchii znaczników strony HTML.
- Dostarczenie modelu umożliwiającego programiście modyfikować, dodawać, usuwać elementy dokumentu .html korzystając z własności drzewa (jego przesukiwania)

Charakterystyka:

Dokument HTML możemy przedstawić w postaci drzewa, które rozpoczyna się od tzw. Klasy dokument reprezentującej aktualną stronę Internetową. Zawiera m.in. dane o szerokości i wysokości okna. Do jej atrybutów programista ma dostęp z poziomu języka JavaScript. Obrazek mówi więcej niż tysiąc słów:



Korzeniem jest tag `<html>`. Kolejne węzły stanowią tagi zawarte wewnątrz znacznika `<html>`. Dzieckiem węzła jest tag bezpośrednio zawarty wewnątrz danego tagu.

12.3 XHTML extensible Hyper Text Markup Language

Cel:

Dostarczenie restrykcyjnego standardu rozszerzającego HTML, aby zwiększyć jakość stron internetowych wymagając od programistów poprawnej składni zgodnie z obowiązującym standardem.

Charakterystyka:

Jak już zostało wcześniej wspomniane w języku HTML, gdy programista popełni błąd strona zostanie wyświetlona przez przeglądarkę. Aby, zapewnić poprawność i spójność kodu został stworzony język XHTML. Posiada identyczne tagi i składnię jak w przypadku języka HTML. Jednak tutaj można wymienić kilka istotnych różnic:

- Jeśli strona XHTML zawiera błędy, nie może zostać wyświetlona;
- Strony XHTML: `http://www.w3.org/1999/xhtml`;
- Element główny (html) musi zawierać atrybut xmlns określający przestrzeń nazw XHTML: `http://www.w3.org/1999/xhtml`;
- Znacznikowi otwierającemu każdego niepustego elementu powinien odpowiadać znacznik zamykający (np. `<li> ... </li>`);
- Puste elementy muszą także być zamykane (np. zamiast `<br>` musi być `<br/>`, albo `<br></br>`);
- Elementy muszą być zagnieżdżane w odpowiedni sposób (np. zamiast `<p>Tekst z <em>wyróżnieniem</p></em>` – `<p>Tekst z <em>wyróżnieniem</em></p>`); wprawdzie w HTML także istniał taki wymóg, lecz nie był egzekwowany przez przeglądarki;
- Nazwy elementów i atrybutów XHTML muszą być pisane małymi literami;

- Wszystkie wartości atrybutów muszą być ujęte w cudzysłów (podwójny, np. `<td rowspan="3»`, albo apostrof, np. `<td rowspan='3'>`);
- Niedozwolona jest minimalizacja atrybutów (np. zamiast `<textarea readonly>` musi być `<textarea readonly=żeadonly»`);
- XHTML muszą mieć typ zawartości `application/xhtml+xml` (lub inny XML)
- Element główny (`html`) musi zawierać atrybut `xmlns` określający przestrzeń nazw

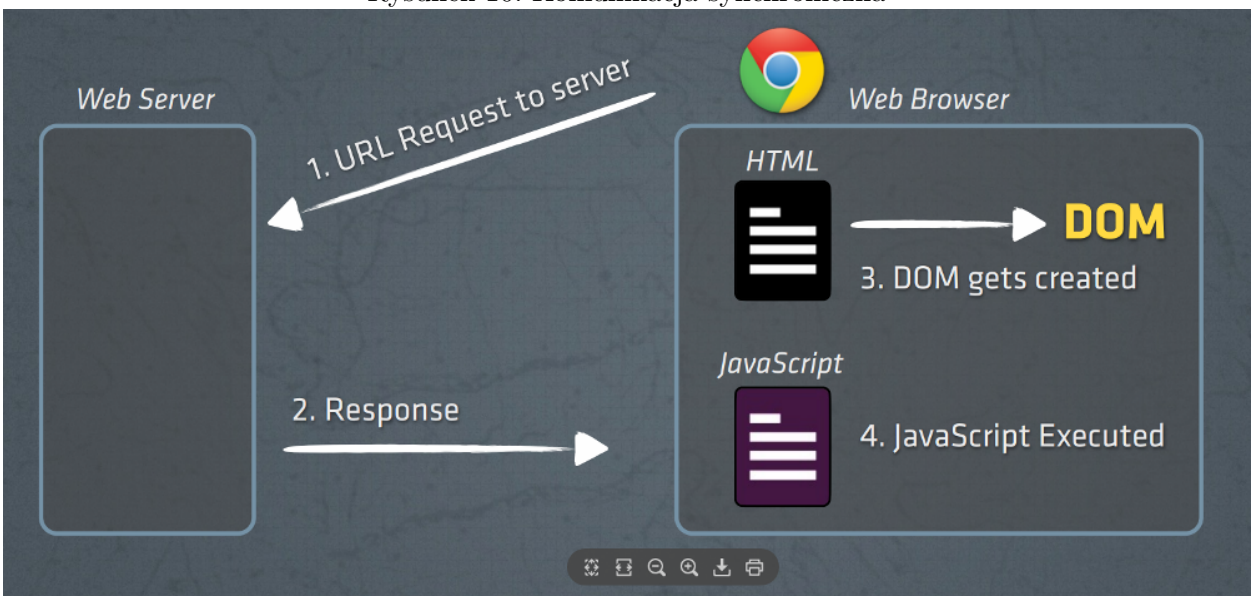
13 S3 – Asynchroniczna komunikacja serwerem HTTP w technologii AJAX

Nazwa AJAX (Asynchronous JavaScript and XML) została wymyślona w 2005 roku. Miała ona opisywać istotę nowych narzędzi udostępnionych przez firmę Google (Google Suggest i Google Maps). Nie jest to oficjalna technologia tzn. nie ma na to standardu opisanego w RFC lub innym tego typu dokumencie. Określenie AJAX opisuje współdziałanie kilku technologii – języka JavaScript, protokołu http, mechanizmów przeglądarek internetowych i serwerów sieciowych. Generalnie AJAX mówi o asynchronicznej komunikacji z serwerem http, zapewniając pobieranie i wyświetlanie materiałów bez konieczności przeładowywania całej strony. Sama nazwa może być myląca, nie trzeba rozumieć XML, aby zrozumieć AJAX.

Czym różni się komunikacja asynchroniczna od komunikacji synchronicznej?

Komunikacja synchroniczna, to standardowy sposób komunikacji, kiedy to wysyłając zadanie blokując czekamy ze zwróceniem interfejsu do klienta, aż zadanie zostanie przetworzone. W przypadku krótkich transakcji taka sytuacja jest do przyjęcia, jednak w przypadku bardzo długich transakcji nie ma ona racji bytu.

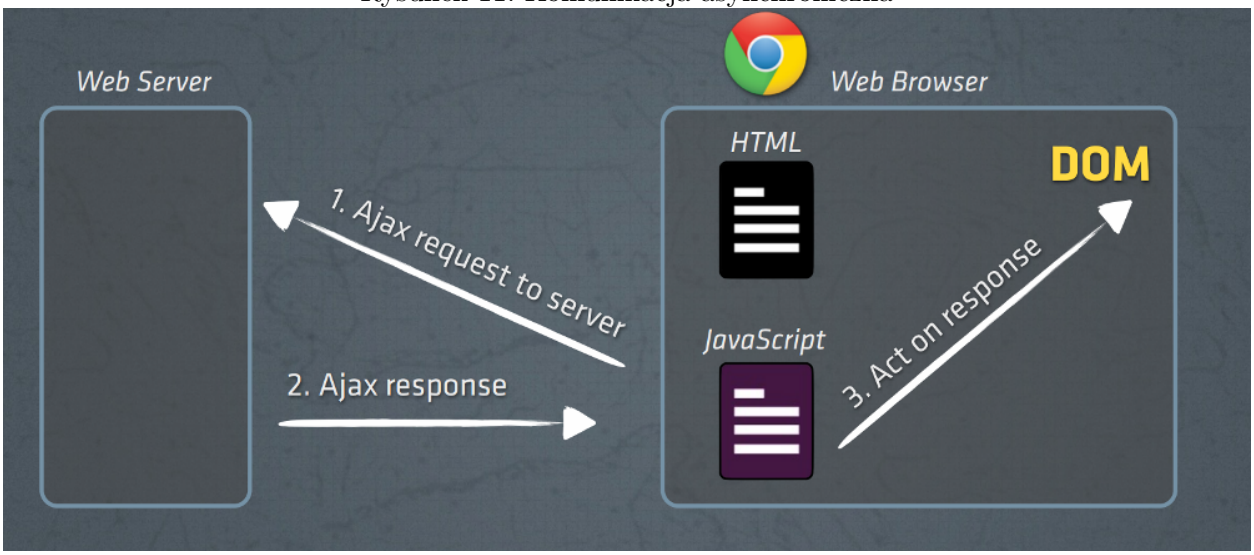
Rysunek 10: Komunikacja synchroniczna



Komunikacja asynchroniczna pozwala nam wysyłać zapytanie w tle bez konieczność czekania ze zwróceniem interfejsu klienta. Kiedy zapytanie się skończy zostanie wywołana wcześniej zdefiniowana funkcja. Dzięki temu możemy:

- aktualizować zawartość strony przez konieczności przeładowywania całego drzewa DOM.
- wysyłać dane do serwera po załadowaniu się całej strony;
- otrzymywać odpowiedź od serwera po załadowaniu się całej strony;
- wysyłać dane do serwera w tle

Rysunek 11: Komunikacja asynchroniczna



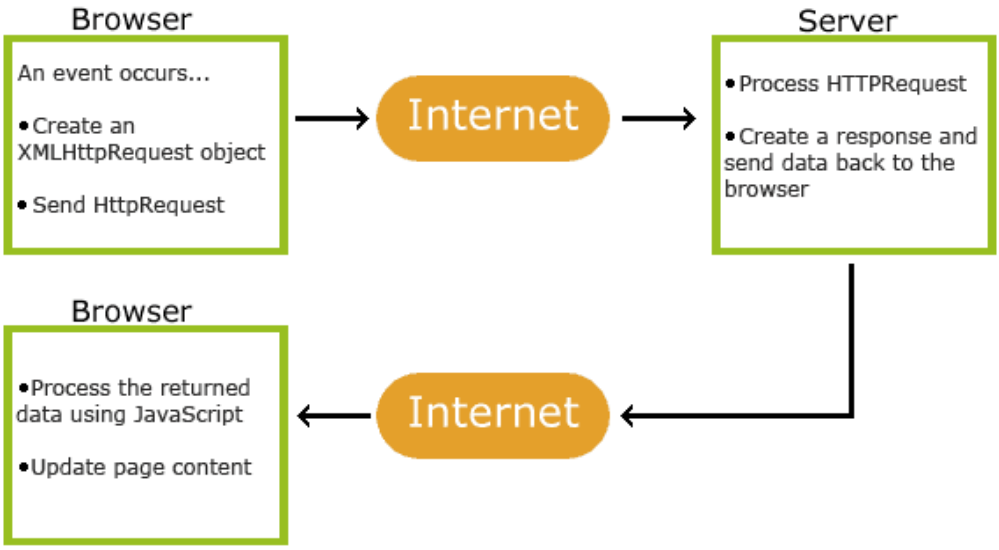
Przykłady zastosowań AJAX:

- Wyświetlanie nowych danych HTML bez konieczności odświeżania strony;
- Przesyłanie formularzy i natychmiastowe wyświetlanie wyników;
- Logowanie bez opuszczania strony;
- Kontrolka do oceny materiałów za pomocą liczby gwiazdek;
- Przeglądanie informacji z bazy danych.

Co jest nam potrzebne do rozpoczęcia działania?

- **Przeglądarka internetowa** – wbudowany obiekt XMLHttpRequest – wprowadzony w przeglądarce Internet Explorer 5;
- **Język JavaScript** – wykonywanie zadań w AJAX-ie np. wysyła żądania na serwer, oczekuje i przetwarza odpowiedź, aktualizuje stronę przez dodanie nowych materiałów lub zmianę jej wyglądu
- **Serwer Sieciowy** – odbiera zadanie od przeglądarki i przesyła odpowiedź z danymi.

Rysunek 12: Klasyczne AJAX-owe przetwarzanie



Implementacja zapytań ASYNCHRONICZNYCH w czystym JavaScripcie:

1. Utworzenie obiektu XMLHttpRequest (dostępny natywnie w przeglądarkach) – więcej informacji `https://developer.mozilla.org/pl/docs/XMLHttpRequest`
2. Specyfikacja typu zapytania – funkcja `open(method, url, async)`;
3. Specyfikacja zdarzenia `onreadystatechange`:
   (a) pole obiektu XMLHttpRequest przechowuje funkcję, która jest automatycznie wywoływana kiedy `readyState` zmienia się;
   (b) pole `readyState` przechowuje status żądania HTTP zmieniającego się od 0 — 4
      i. 0 – zapytanie niezainicjalizowane;
      ii. 1 – nawiązanie połączenia z serwerem;
      iii. 2 – zapytanie zostało otrzymane;
      iv. 3 – przetwarzanie zapytania;
      v. 4 – zapytanie przetworzone, odpowiedź otrzymana;
4. Wysłanie zapytania funkcja send:
   (a) `send()` – wysyła zapytanie wykorzystując GET;
   (b) `send(string)` – wysyła dane na serwer;

Obiekty promise – JavaScript Obiekt promise jest używany do zadań asynchronicznych i odroczonych w czasie. Reprezentuje operację, która nie została jeszcze zakończona, ale oczekuje na zakończenie w przyszłości. Obiekt może znajdować się w 3 stanach:

- `pending` – początkowy stan obiektu promise;
- `fulfilled` – stan reprezentujący operację zakończoną powodzeniem;

- `rejected` – stan reprezentujący operację zakończoną niepowodzeniem;

Obiekty promise wykorzystują do zapytań asnychronicznch biblioteki i frameworki webowe np. jQuery oraz AngularJS.

14 S4 – Technologie platformy Java EE

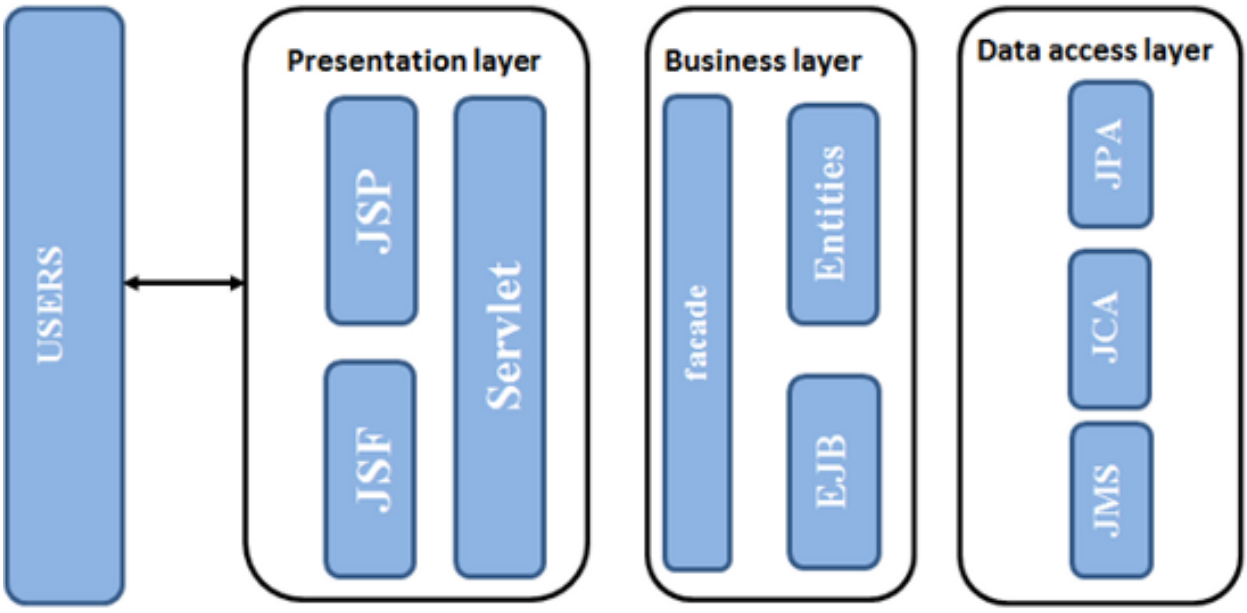
Java Platform, Enterprise Edition znana jako Java EE (nie używać JEE `https://java.net/projects/javaee-spec/pages/JEE`) to **specyfikacja** platformy programistycznej Java. Nie jest to implementacja, a tak naprawdę rozszerzenie wersji SE o komponenty (interfejsy) sieciowe, które umożliwiają pisanie aplikacji o przeznaczeniu komercyjnym. Twórcą tej technologi był *Sun Microsystems*, ale kupił go *Oracle*. DO uruchomienia aplikacji potrzebny jest serwer aplikacji Java EE. Do najczęściej używanych i zarazem najpopularniejszych serwerów należą:

- Apache Tomcat
- GlassFish
- IBM WebSphere
- JBoss Application Server

Cechy Java EE

- bezpieczeństwo,
- skalowalność,
- przenośność,
- wielowarstwowość.

Aplikację napisaną zgodnie ze standardem Java EE można podzielić na minimum 3 warstwy:



Rysunek 13: Warstwy aplikacji Java EE

- Warstwa prezentacji – w tej warstwie zdefiniowany jest interfejs użytkownika.
- Warstwa biznesowa – tworzenie komponentów implementujących logikę biznesową.

- Warstwa dostępu do danych – zapewnia izolację danych.

W warstwie prezentacji można wyróżnić 3 zasadnicze sposoby generowania interfejsu użytkownika.

JSF – framework, bazujący na języku Java, który upraszcza tworzenie interfejsu użytkownika do aplikacji Java EE. Obecnie domyślną technologią widoku dla stron JSF jest technologia Facelets. Składa się z kontrolek umożliwiających szybkie tworzenie interfejsu użytkownika.

JSP – to technologia umożliwiająca tworzenie dynamicznych dokumentów WWW w formatach HTML, XHTML, DHTML oraz XML z wykorzystaniem języka Java, wplecionego w kod HTML danej strony. W tym aspekcie, jest to rozwiązanie podobne do PHP. Strona JSP w procesie translacji jest zamieniana na serwlet. Każde wywołanie strony JSP z poziomu klienta (przeglądarki) wykonywane jest przez skompilowany serwlet.

W części dynamicznej JSP można wykorzystać 5 elementów:

- <%@ dyrektywa %>
- <%! deklaracja %>
- <%= wyrażenie %>
- <% kod %>
- <%– komentarz –%>

Dodatkowo, dzięki użyciu **JSTL** dodane zostają dodatkowe tagi do **JSP**, co umożlwia m.in. tworzenie pętli. Tagi te posiadają swoje prefiksy np. Core tags,–c; XML–x; HTML-h. Przykład: <c:forEach...>.

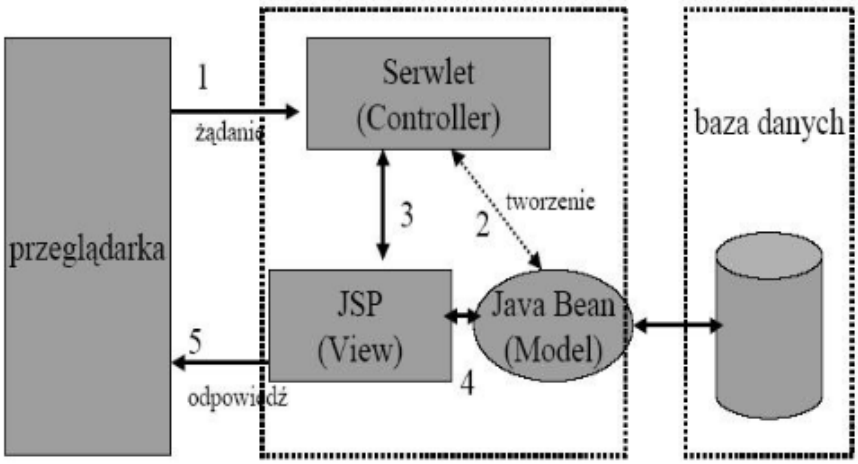
Serwlety – jest to samodzielna klasa Javy, która dziedziczy po HttpServlet. Zawiera metody niezbędne do obsłużenia zapytania http. Osadzana jest w kontenerze webowym (aplikacja), który wywołuje jej metodę doPost() lub doGet() w celu obsłużenia przychodzącego żądania. Servlet taki wówczas może odesłać odpowiedź, lub przekierować żądanie do innej części aplikacji (sendRedirect). Każde zapytanie realizowane jest w oddzielnym wątku.

Typowe zastosowania serwletu:

- przekierowanie do strony JSP,
- kontakt z bazą danych,
- uwierzytelnianie użytkowników,
- sterowanie aplikacją.

Cykl życia serwletu składa się z następujących kroków:

1. Załadowanie klasy serwletu do pamięci (tylko raz, podczas pierwszego ładowania strony).
2. Wywołanie konstruktora oraz init().
3. Instancja rezyduje w pamięci w oczekiwaniu na zapytania. Jeśli nadejdzie nowe zapytanie, serwer tworzy nowe obiekty (request i response) oraz wywołuje service() przekazując request i response do odpowiedniej metody zaimplementowanej przez programistę.
4. Po zakończeniu pracy serwletu wywołane jest destroy(), aby zwolnić zasoby.



Rysunek 14: Współpraca JSP i serwletów - jako MVC (Model View Controller)

Następnym ważnym elementem występującym tym razem w warstwie biznesowej są **Enterprise Java Beans**. Jest to specyfikacja tworzenia aplikacji korporacyjnych. Opiera się ona na tworzeniu komponentów – ziaren EJB, które są osadzane na serwerze aplikacji, a następnie ten serwer udostępnia je do wykonania lokalnie lub zdalnie (np. RMI). EJB tworzy rozproszoną, bezpieczną i transakcyjną klasę Java.

Ziarna EJB dzielą się na:

- Session Beans – stanowe (skojarzone z apl. klienta, pamiętają swój stan) oraz bezstanowe (może być za każdym razem używany przez innego klienta, pamięta tylko stan na czas wywołania)
- Message-Driven Beans
- Entity Beans (w EJB 3.0 zastąpione przez **JPA**)

Każde z ziaren ma różne zastosowanie. Ziarna sesyjne są używane do umieszczania w nich logiki aplikacji – czyli kodu, który przetwarza dane. Encyjne EJB reprezentują w sposób obiektowy dane (np. dostarczają obiektowego spojrzenia na relacyjną bazę danych). Ziarna sterowane komunikatami znajdują zastosowanie w przetwarzaniu asynchronicznym i w zaawansowanych modelach współpracy oprogramowania.

RMI – Remote Method Invocation. Jest to protokół umożliwiający zdalne wywoływanie metod obiektów w aplikacjach Java. Obiekty te mogą znajdować się w innych maszynach wirtualnych Javy, które mogą znajdować się na innych komputerach. Obiekty zdalne rejestrowane są pod wybranymi nazwami w serwisie RMI Registry. Aplikacje korzystające z RMI dzielą prace na:

- Odebranie żądania od klienta.
- Wykonanie zadania.
- Odesłanie odpowiedzi.

JDBC – Java DataBase Connectivity. Odpowiednik ODBC dla Javy. Jest to interfejs oprogramowania pozwalający na połączenie z bazą danych SQL przez aplikacje Java.

JPA – JavaPersistence API jest standardem ORM dla języka Java. Z punktu widzenia programisty jest to możliwość operowania na obiektach - zwanych encjami - oraz zapisywania wyników operacji do relacyjnej bazy danych za pomocą obiektu EntityManager. Implementacją która z tego korzysta jest np. **Hibernate**.

Popularne frameworki Java EE:

Spring – Spring powstał jako alternatywa dla programowania aplikacji z użyciem Enterprise JavaBeans. Framework ma pomagać przy tworzeniu aplikacji na każdym poziomie. Ułatwia korzystanie z istniejących technologii.

Play! – Inspirowany na Ruby on Rails oraz Django. Jest bezstanowy, RestFull. Bazuje na wzorcu MVC.

Vaadin – używa Google Web Toolkit do generowania stron dla użyszkodnika. Wykorzystuje widgety oraz programowanie zdarzeniowe, więc tworzenie stron przypomina soft do tworzenia GUI, niż typowy webdev.

15 S5 – Komunikacja procesów przez pamięć dzieloną

Komunikacja międzyprocesowa jest potrzebna do poprawnego funkcjonowania systemów wieloprocesorowych. Istnieje wiele metod komunikacji między nimi. Można do nich zaliczyć metody takie jak: pliki, łącza nienazwane oraz nazwane, kolejki, gniazda, **pamięć dzieloną** i inne, np. MPI. Każda z nich zawiera charakterystyczne cechy, ma swoje wady i zalety.

W odróżnieniu od wątków, komunikacja między procesami jest trudniejsza w zrealizowaniu, ponieważ wątki dzielą zmienne globalne oraz przestrzeń adresową. Każdy proces posiada własny kontekst i przestrzeń adresową – procesy nie mogą ingerować w pamięć innych procesów, ponieważ to mogłoby mieć katastrofalne skutki.

Jak już wspominałem, istnieje wiele sposobów komunikacji międzyprocesowej. Jedną z nich jest komunikacja przez pamięć dzieloną. Jeżeli ktoś nie wie dlaczego tak to podkreślam, to zapraszam do zajrzenia na tytuł pytania egzaminacyjnego. Jest to jedna z najszybszych form komunikacji, ponieważ przesył danych między procesami nie jest wymagany, jądro nie jest pośrednikiem w komunikacji. Operacje odbywają się poprzez odwołanie do innych segmentów pamięci (ale pamiętajmy, że w wirtualnej przestrzeni adresowej, procesy nie operują na fizycznych adresach). Aby uzyskać taki segment, należy go najpierw odwzorować, za pomocą funkcji dostępnej w danej bibliotece (POSIX, WinAPI, Boost, etc.).

Należy jednak pamiętać, że wykorzystanie pamięci dzielonej, jak to z współbieżnością bywa, niesie ze sobą wymóg synchronizacji dostępu tej pamięci między procesami, które na niej operują. Możliwe rozwiązania synchronizacyjne to semafory, muteksy, monitory, etc. (więcej w punkcie dot. synchronizacji danych).

Istnieje wiele architektur, których celem jest współbieżne przetwarzanie. Jednakże, komunikacja przez pamięć dzieloną może zostać wykorzystana na systemach jedno- lub wieloprocesorowych, ze wspólną pamięcią. Niezależnie od wykorzystanej biblioteki, schemat wykorzystania pamięci współdzielonej jest podobny:

- utworzenie segmentu,
- ustalenie rozmiaru segmentu,
- odwzorowanie segmentu w przestrzeni adresowej procesu,
- operacje na segmencie,
- wycofanie odwzorowania,
- odłączenie segmentu (gdy licznik odwołań użycia segmentu spadnie do zera, segment jest usuwany).

Wymogiem jest wzajemne wykluczanie, ponieważ nie powinno dojść do sytuacji, w których procesy modyfikują ten sam fragment danych – może to mieć niepożądane skutki.

Zarówno w systemach UNIXowych oraz Windows wykorzystanie pamięci dzielonej działa na podobnej zasadzie. Standard POSIX, wykorzystywany w systemach UNIXowych udostępnia dwa zestawy funkcji do operacji na pamięci współdzielonej – POSIX oraz System V. Nie można ich ze sobą mieszać.

- `shm_open()`/*`shmget()`** - utworzenie segmentu

- `ftruncate()`/*`shmget()`** - ustalenie rozmiaru segmentu
- `mmap()`/*`shmat()`** - pobranie adresu odwzorowania segmentu
- `munmap()`/*`shmdt()`** - wycofanie odwzorowania segmentu
- `shm_unlink()`/*`shmctl()`** - odłączenie segmentu

$^\star \rightarrow$ *wywodzą się z* ***System V***; w zestawie funkcji POSIX w odróżnieniu od System V, odwoływanie się do segmentów pamięci dzielonej (funkcje do tworzenia, odłączania segmentów) może odbywać się przez nazwy, w drugim standardzie poprzez indeksy.

Pamięć dzielona pozwala na komunikację pomiędzy różnymi procesami w jednym systemie. Jest to najszybszy sposób komunikacji, ponieważ nie wykorzystuje jądra. Wadą tego rozwiązania jest konieczność stosowania technik, które zapewniają poprawność przetwarzania danych. Oprócz tego, rozwiązanie to nie sprawdzi się w systemach sieciowych.

16 S6 – Metody komunikacji międzyprocesowej w systemach lokalnych i rozproszonych

Komunikacja międzyprocesowa - (*Inter-Process Communication* - IPC) - sposób komunikacji pomiędzy wielomoa procesami systemu operacyjnego uruchomionego na maszynie lokalnej, bądź rozproszonej grupie autonomicznych maszyn połączonych w sieć. Techniki komunikacji możemy podzielić na:

- Przekazywanie komunikatów
- Synchronizację
- Współdzielenie pamięci
- Wywoływanie procedur

Główne cechy systemu rozproszonego:

- **Dzielenie zasobów** - maszyny rozproszonego systemu mają możliwość współdzielenia urządzeń (np. drukarki, skanery), dzielenia plików oraz usług
- **Otwartość** - podatność na rozszerzenia, możliwość rozbudowy systemu zarówno sprzętowo jak i programowo
- **Współbieżność** - zdolność przetwarzania wielu zadań równocześnie
- **Skalowalność** - zachowanie podobnej wydajności systemu przy zmienianiu skali systemu (np. dodawanie nowych maszyn, procesorów)
- **Tolerowanie awarii** - zdolność do dalszego działania systemu mimo wystąpienia błędów w oprogramowaniu lub uszkozeń fizycznych
- **Przezroczystość** - cały system zachowuje się tak, jakbyśmy pracowali na jednej maszynie, nie zas na wielu osobnych jednoskach połączonych w jeden system

Tablica 11: Metody komunikacji międzyprocesowej

Metoda komunikacji	**Systemy operacyjne**	**Lokalne**	**Rozproszone**
Pliki	Większość systemów	tak	tak
Gniazdka	Większość systemów	tak	tak
Sygnały	POSIX	tak	nie
Kolejki komunikatów	POSIX, Windows	tak	nie
Łącza nienazwane	POSIX, Windows	tak	nie
Łacza nazwane	POSIX, Windows	tak	nie
Semafory	POSIX, Windows	tak	nie
Pamięć współdzielona	POSIX, Windows	tak	nie
Zmapowany plik	POSIX, Windows	tak	tak
Przekazywanie komunikatów	Systemy z MPI, RPC, COBRA, Java RMI	tak	tak

16.1 Pliki

Jest to najłatwiejszy sposób wymagający jedynie otwarcia pliku. Dla zachowania spójności danych tylko jeden proces może zapisywać do pliku w danym czasie, zaś czytać może nieograniczona ilość procesów. Występuje tutaj potrzeba zastosowania jakiegoś sposobu synchronizacji procesów zapisujących i informowania innych o zmianach. Komunikacja może odbywać się w obrębie jednogo systemu operacyjnego, bądź też między różnymi systemami współdzielącymi przestrzeń dyskową.

Przykłady funkcji POSIX:

- `fopen()` - otwarcie pliku
- `fwrite()` - zapis do pliku
- `fread()` - odczyt z pliku
- `fclose()` - zamknięcie pliku

16.2 Gniazdka

Umożliwiają komunikację dwukierunkową w systemach lokalnych i rozproszonych - pozwalają na odbieranie i wysyłanie danych po obu stronach. W systemach UNIX gniazdko traktowane jest jak deskryptor otwartego pliku, więc można na nim używać funkcji jak do plików (`read()`, `write()`, `close()`). Oba punkty końcowe połączenia identyfikowane są za pomocą adresu IP oraz portu. Komunikacja przebiegać może bezpołączeniowo po protokole UDP za pomocą ciągu niezależnych pakietów (datagramów), bądź połączeniowo po TCP czyli przy pomocy strumienia danych.

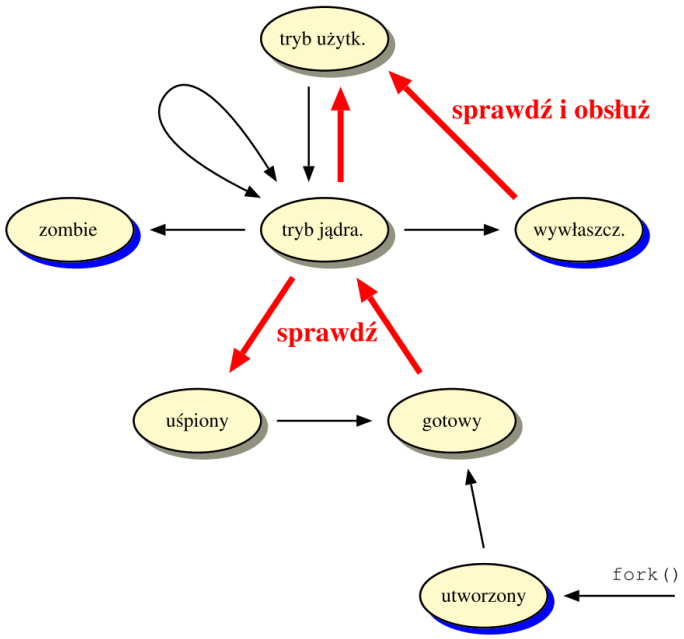
Przykłady funkcji POSIX:

- `socket()` - utworzenie gniazdka
- `bind()` - nadanie gniazdku adresu
- `connect()` - połączenie się z serwerem TCP
- `accept()` - połączenie się z klientem TCP
- `listen()` - rejestracja gniazdka TCP
- `send()`, `recv()` - wysłanie i odebranie wiadomości TCP
- `sendto()`, `recvfrom()` - wysłanie i odebranie wiadomości UDP
- `write()`, `read()` - wysłanie i odebranie wiadomości TCP a także UDP po wykonaniu asocjacji gniazdka funkcją `connect()`
- `close()` - zamknięcie gniazdka

16.3 Sygnały

Znane także jako przerwania programowe - są to asynchroniczne powiadomienia wysyłane do innych procesów wymagające natychmiastowej reakcji ze strony procesu. Źródłem przerwań może

być system operacyjny gdy nastąpi wydarzenie awaryjne, wykonanie odpowiednich funkcji w programie, bądź zabicie programu spod konsoli funkcją kill. Po otrzymaniu przerwania, system zaprzestaje wykonywania aktualnych instrukcji (przerwana może być każda nie-atomowa operacja) i rozpoczyna obsługę przerwania wykonując ustalone wcześniej funkcje przechwyconych przerwań, bądź domyślne funkcje obsługi przerwań. Przechwycony może być każdy sygnał z wyjątkiem SIGKILL oraz SIGSTOP. Sygnały obsługiwane sątylko przy przejściu między trybem jądra systemu a trybem użytkownika.



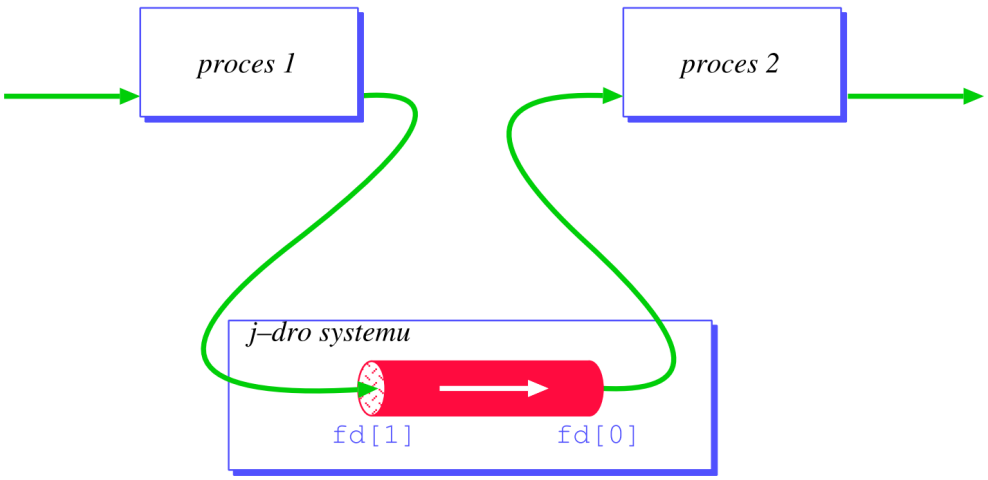
Rysunek 15: Sygnały

Przykłady funkcji POSIX:

- `signal()`, `sigset()` - rejestruje funkcję obsługi sygnału
- `kill()`, `alarm()`, `raise()` - wysyła sygnał

16.4 Łącza nienazwane

Jedna z prostszych metod komunikacji. Są to strumienie (pipe) pozwalajace łączyć spokrewnione ze sobą procesy w relacji jednokierunkowej macierzysty - potomny. Zazwyczaj są wykorzystywane do łączenia wyjścia stdout jednego procesu z wejściem stdin drugiego. Tworzenie łącza wypełnia dwuelementową tablicę deskryptorów, z czego pierwszy z nich jest końcem otwartym do czytania, a drugi służy do pisania. Oba końce traktowane są w systemie UNIX jako zwykłe deskryptory plików, więc można na nich używać funkcji jak do plików (`read()`, `write()`, `close()`). Nieużywany deskryptor musi zostać zamknięty funkcją `close()`. Jądro systemu gwarantuje, że operacje zapisu nie przekraczające rozmiaru bufora (w wielkości 4kB) są atomiczne, niepodzielne. W powłokach systemów UNIX można używać łącza w formie pionowej linii do przesyłania wyników jednego polecenia do wejścia kolejnego.



Rysunek 16: Łącze nienazwane

Przykłady funkcji POSIX:

- `pipe()` - tworzy łącze nienazwane
- `dup()`, `dup2()` - duplikowanie deskryptorów stdin, stdout
- `write()` - zapisuje do łącza
- `read()` - odczytuje z łącza
- `close()` - zamyka łącze

16.5 Łącza nazwane

Inaczej strumienie FIFO, mogą być używane przez procesy w żaden sposób ze sobą nie spokrewnione, a nawet przez procesy różnych użytkowników. Szczególnie stosowane, gdy nie można zastosować relacji macierzysty - potomny. Strumienie FIFO posiadają dowiazanie w systemie plików jako pliki specjalne - oprócz atrybutów zwykłych plików takich jak nazwa, właściciel, grupa czy prawa dostępu, posiadają cechę dodatkową - element odczytany z pliku FIFO jest z niego automatycznie usuwany. Funkcja użyta do otwarcia łącza blokuje proces do momentu otwarcia drugiego powiązanego łącza. Kolejki FIFO mają z reguły większy rozmiar 4-16kB.

Przykłady funkcji POSIX:

- `mknod()` - tworzy łącze nazwane
- `open()` - otwiera łącze do pisania lub czytania
- `write()` - zapisuje do łącza
- `read()` - odczytuje z łącza
- `unlink()` - usuwa łącze

Miedzy łączami nienazwanymi a nazwanymi istnieje wiele podobieństw:

- Zamknięcie końca do pisania generuje u czytelników EOF
- Zamknięcie końca do czytania generuje sygnał SIGPIPE u pisarzy
- Zapis i odczyt przez standardowe funkcje `read()` i `write()`
- Niepodzielność zapisó mniejszych niż rozmiar strumienia
- Blokowanie procesów w przypadku przepełnienia lub pustego bufora

16.6 Kolejki komunikatów

Jest to lista o określonej pojemności zawierająca komunikaty o zadanym maksymalnym rozmiarze. Komunikacja odbywa się tylko w jedną stronę - nowe komunikaty dodawane są na koniec listy. Dzięki temu zachowana jest kolejność komunikatów. Każdy komunikat ma swój typ, dzięki czemu można obsługiwać kilka strumieni komunikatów w ramach jednej kolejki poprzez selektywne odbieranie komunikatów wybranego typu. Komunikacja odbywa się asynchronicznie, co znaczy, że odbiorca i nadawca nie muszą się łączyć w tym samym czasie. Komunikaty w kolejce ponadto przechowywane są do czasu odebrania ich przez inny proces. Kolejki komunikatów posiadają również inne ważne cechy:

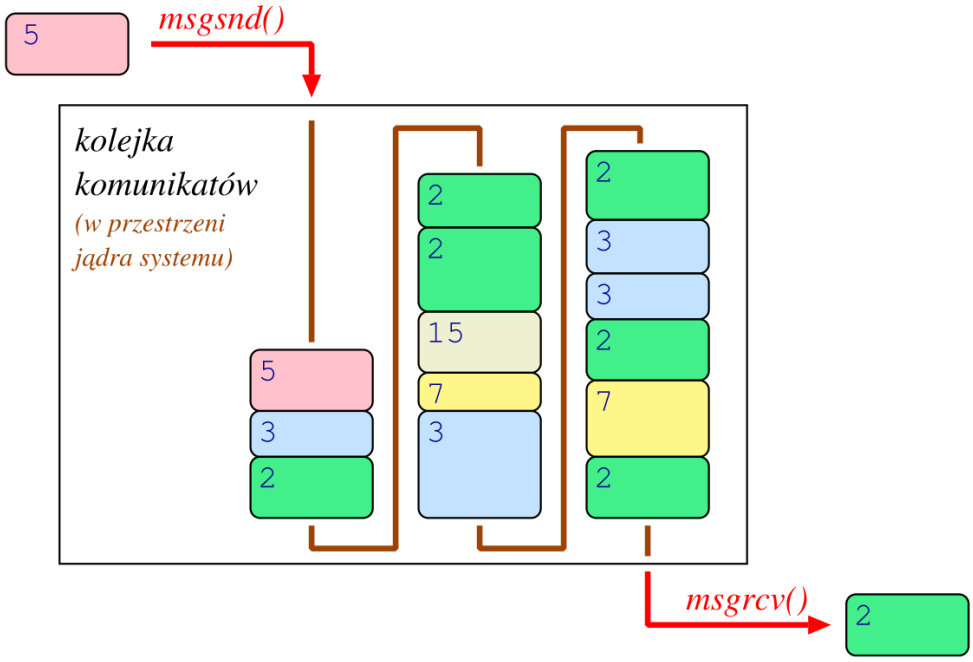
- Posiadają unikalne nazwy umożliwiające procesom identyfikację kolejki
- Można nadawać priorytety - komunikaty o wyższym priorytecie są umieszczane na początku
- Wiele procesów może pisać i czytać do/z kolejki
- W kolejce mogą się znajdować komunikaty różnej długości (w przeciwieństwie do FIFO)
- Kolejce można przypisać maksymalną liczbę komunikatów - po jej przekroczeniu, proces piszący zostaje zablokowany
- Można testować status kolejki, np. liczbę komunikatów (kolejki FIFO tego nie umożliwiają)

Przykłady funkcji POSIX:

- `msgget()` - tworzy kolejkę
- `msgsnd()` - wysyła komunikat
- `msgrcv()` - odbiera komunikat
- `msgctl()` - modyfikuje parametry kolejki (usunięcie poprzez ustawienie parametru IPC_RMID)

Przykłady funkcji System V:

- `mq_open()` - tworzy kolejkę
- `mq_send()` - wysyła komunikat
- `mq_receive()` - odbiera komunikat
- `mq_close()` - zamyka kolejkę
- `mq_unlink()` - usuwa kolejkę



Rysunek 17: Kolejka komunikatów

16.7 Funkcja select

16.8 Semafory

chroniona zmienna systemowa, operacje atomowe V (inc), P (dec), nie może być < 0, synchronizacja procesów

16.9 Pamięć współdzielona

jeden proces tworzy obszar pamięci RAM, pozostałe mają do niego dostęp - wspólny segment pamięci, odwzorowana jako własny obszar pamięci logicznej, wymaga dodatkowej synchronizacji

16.10 Zmapowany plik w pamięci

16.11 Przekazywanie komunikatów

RPC - zdalne wywoływanie procedur RMI - technologia Java, stub COBRA - komunikacja między różnymi językami MPI - obliczenia wielowątkowe

17 S7 – Protokoły Internetu, Ochrona danych i uwierzytelnianie w Internecie

17.1 Protokoły Internetu

Protokół Internetowy - wyspecyfikowany zbiór zasad i reguł umożliwiających komunikację urządzeń w danej warstwie modelu.

Protokoły dzielą się ze względu na warstwę, w której pracują. Najważniejsze protokoły, z jakimi spotkać się można w Internecie to:

- Warstwa dostępu do sieci
  - **CSMA/CD** - protokół dostępowy do sieci implementujący wykrywanie kolizji w łączu oraz ponawianie transmisji, stosowany jest w technologiach Ethernet i WiFi, gdzie każde urządzenie w sieci posiada unikalny 48-bitowy heksadecymalny adres MAC przypisany do urządzenia
- Warstwa Internetu
  - **IP** - przesyła dane w postaci pakietów, bezpołączeniowy (nie ma gwarancji dotarcia pakietów do celu, przez co niezawodność musi być zapewniana przez protokoły wyższych warstw), pierwotnie istniał podział na klasy A-E, obecnie wykorzystywane są dwie wersje adresów: 32-bitowy IPv4 oraz 128-bitowy IPv6
  - **ICMP** - funkcja diagnostyczna - pomoc przy znajdowaniu problemu, wykorzystywany przez ping i traceroute
- Warstwa transportowa
  - **TCP** - połączeniowy, niezawodny (sumy kontrolne, numery sekwencyjne), gwarancja dostarczenia pakietu w całości, w odpowiedniej kolejności, bez duplikatów, wykorzystuje *Three-way handshake* do nawiązania połączenia
  - **UDP** - bezpołączeniowy, zawodny (brak retransmisji i sum kontrolnych), możliwy multicast, szybszy od TCP dlatego stosowany przy wideokonferencjach czy grach sieciowych
- Warstwa aplikacji
  - **DNS** - zamienia adresy znane użytkownikowi na adres liczbowy, zrozumiały dla urządzeń tworzących sieć komputerową, posługuje się do komunikacji głównie protokołem UDP na porcie 53
  - **Telnet** - służy do terminalowego połączenia ze zdalną maszyną, nie jest szyfrowany, działa na porcie 23
  - **SSH** - następca protokołu Telnet rozszerzający go o możliwość przesyłania plików, zdalnej kontroli zasobów, czy tunelowania, szyfruje transfer danych najczęściej przy pomocy szyfru AES, działa na porcie 22, oparte na nim są m.in. protokoły SCP i SFTP
  - **SSL** - protokół używany do zabezpieczania połączeń Internetowych, zawiera w sobie algorytmy symetryczne (przesył danych) i asymetryczne (inicjalizacja połączenia)

- **TLS** - rozszerzenie SSL zwiększające bezpieczeństwo, zawiera certyfikaty potwierdzające tożsamość
- **HTTP** - bezstanowy (nie zapisuje żadnych informacji o poprzednich transakcjach, przez co do zapamiętywania danych należy skorzystać z plików kolajnych Cookie), pozwalający na przesyłanie różnych zasobów, w tym dokumentów hipertekstowych, poprzez realizację zapytań (GET, PUT, POST, DELETE i inne), działający domyślnie na porcie 80 protokołu TCP
- **HTTPS** - nakładka na protokół HTTP dodający szyfrowanie zrealizowany przy pomocy TLS (dawniej SSL), działa na porcie 443 TCP
- **FTP** - protokół komunikacyjny klient-serwer umożliwiający dwukierunkowy transfer plików, zrealizowany przy pomocy dwóch połączeń TCP (dane - port 20, polecenia - port 21), brak szyfrowania - uwierzytelnianie odbywa się plaintextem
- **SMTP** - tekstowy protokół poczty elektronicznej, pracuje na porcie 25
- **IMAP** - protokół poczty elektronicznej, stanowi następcę protokołu POP3, jednak w przeciwieństwie do niego nie musi pobierać całej zawartości skrzynki na dysk, lecz operuje "w locie", obsługuje wielu użytkowników w tym samym czasie, system flag statusów wiadomości, zarządzanie folderami zdalnymi, pracuje na porcie 143

Część powyższych protokołów zawiera się w zbiorze protokołów **IPSec** służący zesawianiu bezpiecznego połączenia oraz bezpiecznego przesyłania pakietów IP. Odbywa się to na zasadzie *kapsułkowania* - oryginalny pakiet IP jest szyfrowany, następnie otrzymuje nowy nagłówek IPSec i w takiej formie jest przekazywany dalej. IPSec wykorzystywany jest przy tworzeniu sieci VPN.

VPN (*Virtual Private Network*) jest tunelem, w którym utworzona jest sieć prywatna będąca częścią sieci publicznej. Dane w obrębie tej sieci prywatnej są kompresowane i szyfrowane. VPN stosowane jest do zdalnego łączenia się z wewnętrzną siecią firmową, zapewniania bezpieczeństwa przy korzystaniu z niezabeczpieczonej sieci publicznej czy do omijania cenzury.

Jak działa *SSL*?

1. Przeglądarka inicjuje połączenie
2. Przeglądarka pobiera klucz publiczny serwera
3. Przeglądarka generuje jednorazowy klucz sesji
4. Przeglądarka szyfruje tak wygenerowany klucz sesji i wysyła go do serwera, który za pomocą klucza prywatnego jest w stanie go odszyfrować
5. W tym momencie tylko klient i serwer są w posiadaniu klucza sesji, który od tej pory jest wykorzystywany jako klucz symetryczny w komunikacji.

17.2 Ochrona danych

W Internecie czyha wiele niebezpieczeństw. Istnieje kilka rodzajów możliwych ataków:

- **Złośliwe oprogramowanie** - konie trojańskie, wirusy, spyware

- **Ataki na infrastrukturę informatyczną** - DoS (*Denial of Service* - nieprzerwane przeładowywanie zasobów sieciowych (np. *flooding*), czy też zasobów lokalnych (np. *fork-bomba*)), DDoS (*Distributed DoS* - zorganizowany i skoordynowany DoS na daną maszynę, system maszyn czy usługę przeprowadzony najczęściej przez grupę nieświadomie zainfekowanych maszyn *zombie* będących częścią *botnetu*), SQL Injection, szukanie luk w oprogramowaniu czy konfiguracji
- **Man in the Middle** - podsłuchiwanie i modyfikacja wiadomości pomiędzy dwoma stronami w Internecie, przykład: postawienie publicznej sieci WiFi i przepuszczenie całego nieszyfrowanego ruchu przez kontrolowane przez siebie proxy
- **Phising** - podawanie się za instytucje, firmy i osoby, podmienianie stron Internetowych, preparowanie fałszywych maili, w celu zazwyczaj uzyskania dostępu do konta, wyłudzenia danych osobowych, czy kradzieży danych karty kredytowej, na przykład: Nigerian Scam (wyciągnięcie niewielkiej kwoty pieniędzy w celu umożliwienia rzekomego transferu o wiele większej kwoty), Pharming (przekierowywanie poprawnego adresu WWW na podmienioną stronę poprzez podmianę serwera DNS bądź złośliwe oprogramowanie)

Przed powyższymi atakami należy się bronić. Istnieje wiele sposobów ochrony:

- Szkolenie pracowników - poszerzanie wiedzy z zakresu ochrony danych, zwiększa się w ten sposób przede wszystkim odporność na ataki socjotechniczne
- Szyfrowanie połączeń przy pomocy sieci VPN, włączenie SSL w protokołach HTTP, IMAP, SMTP, używanie SSH zamaist FTP czy Telnetu
- Stosowanie odpowiedniej infrastruktury technicznej - np. stosowanie światłowodów zamiast kabli miedzianych, gdyż miedziaki narażone są na podsłuchiwanie poprzez anteny skupiające wypromieniowaną energię przez kabel, światłowód będąc dielektrykiem nie wypromieniowuje energii, a sprzęt do jego podsłuchu jest o wiele droższy
- Nie trzymanie haseł w postaci jawnej, zapisywanie ich jako skrót kryptograficzny uznawany jako bezpieczny (SHA512 jest o wiele bardziej bezpieczny niż MD5)
- Systemy przechowujące poufne informacje w ogóle nie powinny być podłączone do Internetu

Istnieją dwa rodzaje szyfrowania z wykorzystaniem klucza:

- **Algorytmy symetryczne** - do szyfrowania i odszyfrowywania wykorzystuje ten sam klucz, albo klucze, które w momencie gdy poznamy jednego z nich, możemy na jego podstawie wygenerować drugi, przykłady algorytmów: AES, DES, Blowfish
- **Algorytmy asymetryczne** - wykorzystuje klucz publiczny, znany wszystkim odbiorcom, oraz prywatny, różny dla każdego klienta; znając tylko klucz publiczny nie ma możliwości odszyfrowania wiadomości ani wygnerowania klucza prywatnego, ponadto wykorzystywane są operacje jednokierunkowe - takie, które łątwo jest wykoać w jedną stronę, zaś trudno w drugą; przykłady użyć: RSA, GPG, SSH

17.3 Uwierzytelnianie w Internecie

Uwierzytelnianie jest to sposób weryfikacji osoby, urządzenia bądź usługi. Może się odbywać za pomocą jednego systemu - uwierzytelnienie jednoetapowe (np. hasło statyczne) - badź być złożeniem

dwóch, bądź kilku uwierzytelnień na raz - uwierzytelnienie wieloetapowe (np. hasło statyczne + token).

- **Hasła statyczne** - ciąg znaków bądź samych cyfr (PIN) znany tylko przez osobę uwierzytelniającą, problemem jest trudność w zapamiętywaniu hasła (najlepiej różnego dla różnych usług) i tworzeniu bezpiecznego hasła, problematyczne jest także przechowywanie hasła po stronie serwera - nie może być przetrzymywany jako *plain-text*, lecz zaszyfrowany bądź zahashowany
- **Hasła jednorazowe** - lista haseł jednorazowych generowanych przez generator kodów (token), z których hasło raz użyte nie może być użyte ponownie. Do generacji wykorzystywany jest jednokierunkowy łańcuch skrótu, który wykorzystuje jednokierunkowe funkcje zwracające wynik w taki sposób, że nie ma możliwości rozpoznania danych wejściowych, a tym samym przewidzieć jakie będzie następne hasło
- **Uwierzytelnianie kryptograficzne** - uwierzytelnianie za pomocą podpisanego klucza prywatnego
- **Karty magnetyczne** - legitymowanie się kartą magnetyczną, czy inteligentną. Bardzo bezpieczna metoda, lecz wymaga posiadania przez użytkownika karty oraz specjalnego czytnika po stronie systemu
- **Techniki biometryczne** - najbezpieczniejsze istniejące obecnie metody uwierzytelniania, polegają na porównywaniu linii papilarnych, długości palców, cech twarzy, czy też tęczówek oka. System taki jest bardzo drogi, dlatego rzadko stosowany

18 S8 – Spójność sieciowego systemu operacyjnego

Sieciowy system operacyjny (NOS – Network Operating System) – umożliwia komunikacje miedzy wieloma urządzeniami a także zasobami sieci. Sieciowy system operacyjny tworzy środowisko, w którym użytkownicy maszyn mają dostęp zasobów sieciowych. Ma on zastosowanie zarówno na serwerach jak i stacjach roboczych.

Do najważniejszych funkcji sieciowego systemu operacyjnego jest zapewnienie komunikacji, poprzez wykorzystanie protokołów komunikacyjnych takich jak:

- TCP/IP - sieć Internet
- IPX/SPX - sieć Novell

Popularne sieciowe systemy dostępne na rynku to:

- Windows Server 2012 R2
- NetWare
- Unix
- Linux

Sieciowy system operacyjny można opisywać na podstawie systemu UNIX oraz zgodnych.

Cechy systemu unixowego:

- wieloużytkownikowy,
- wielozadaniowy,
- podział czasu,
- wywłaszczanie procesów,
- podsystem plików,
- zarządzanie pamięcią,
- biblioteki systemowe,
- urządzenia dostępne jako pliki specjalne.

System powinien posiadać interfejsy programistyczne zgodne z ANSI/C lub Posix:

- Procesy tworzone przez funkcję fork(), kończone przez wait() i exit().
- Komunikacja między procesami przy użyciu sygnałów.
- Obsługa strumieni FIFO i pipe.
- Komunikacja przy użyciu pamięci dzielonej.

Komunikacja sieciowa w oparciu o sieć magistrali, pierścienia lub gwiazdy. Obsługa komunikacji zgodnie z modelem ISO/OSI (siedem warstw) przy użyciu TCP/IP. Większość komunikacji odbywa się w modelu klient-serwer.

Ważnym z punktu widzenia spójności jest system plików używany w takim systemie. Zakłada on model rozproszony – DFS (Distributed File System), który przechowuje pliki na wielu serwerach.

Zalety DFS to:

- Zwielokrotnienie – (z ang. replication) wiele kopii danych.
- Niezawodność – wielokrotność egzemplarzy może uchronić przed ich zaginięciem.
- Efektywność – szybszy i łatwiejszy dostęp do danych.

Wady DFS:

- Problemy ze spójnością.
- Edycja danych na jednym serwerze nie gwarantuje ich zmiany na pozostałych.
- Konieczny jest mechanizm zapewniający aktualizację na pozostałych serwerach.
- Optymalne rozmieszczenie serwerów sprzyja spójności danych.

Sposoby dostępu do danych w systemach rozproszonych:

Dostęp zdalny:

- Współdzielony plik zlokalizowany jest na serwerze.
- Regularne przesyłanie komunikatów pomiędzy klientem a serwerem – możliwe opóźnienia.
- Prosty sposób, brak problemów ze spójnością.

Relokacja:

- Przeniesienie pliku do pamięci lokalnej hosta.
- Zmniejszenie ruchu w sieci.
- Pozostałe hosty odwołują się do nowej lokalizacji pliku.

Powielanie:

- Kopiowanie pliku do pamięci lokalnej klienta.
- Lokalne odwołanie do pliku – pełna prędkość.
- Potencjalna niespójność danych.

Metody propagowania zmian do serwera:

- Natychmiastowe przepisywanie.
- Opóźnione przepisywanie.
  - Zastosowanie polityki aktualizacji.
  - Okresowa aktualizacja.
  - Aktualizowanie przy zamykaniu pliku.

Walidacja danych pomiędzy serwerem a klientem może zostać zrealizowana na 2 sposoby. Pierwszy z nich polega na zainicjalizowaniu tego procesu przez klienta:

- Przy każdym dostępie do danych lub co stały okres czasu.
- Klient wysyła dużo komunikatów do serwera – skutkuje zwiększeniem się ruchu sieciowego.

- Możliwe obciążenia serwera.

Drugi sposób polega na zainicjalizowaniu procesu walidacji przez serwer:

- Serwer monitoruje stan wszystkich plików.
- Klient informuje o zmianach w plikach.
- Wszystkie kopie można odczytywać w tym samym czasie.
- Tylko jedna kopia może być modyfikowalna.

Przykłady sieciowych systemów plików:

- NFS – protokół oparty na UDP lub TCP. Jest standardowym sieciowym systemem plików w systemach uniksowych. Z NFS wiąże się wiele problemów -– przede wszystkim bardzo trudno zapewnić, że dana operacja została wykonana.
- SMB – protokół służący udostępnianiu zasobów komputerowych, m.in. drukarek czy plików. SMB jest protokołem typu klient-serwer, a więc opiera się na systemie zapytań generowanych przez klienta i odpowiedzi od serwera.
- Microsoft DFS – implementacja rozproszonego systemu plików firmy Microsoft. Do wymiany danych korzysta on z protokołu SMB. DFS był już obecny w Windows NT 4.0, obecnie jest dostępny też w Windows Server 2000/2003/2008 oraz Samba 3.0.

19 S9 – Charakterystyka mikrokontrolerów

Mikrokontroler to układ scalony mikroprocesorowy, który zawiera elementy, takie jak: jednostkę centralną (CPU), pamięć (danych, programu) oraz układy wejścia/wyjścia. Jego nazwa pochodzi od obszaru zastosowań – sterowanie urządzeniami elektronicznymi. Bloki funkcjonalne, jakie może zawierać to:

- kontrolery przerwań,
- generator sygnału zegarowego (np. oscylator),
- układy licznikowe,
- kontrolery transmisji,
  - równoległej,
  - szeregowej,
- przetworniki analogowo-cyfrowe oraz cyfrowo-analogowe,
- zegar czasu rzeczywistego (RTC),
- watchdog (układ wykrywający błędne działanie systemu, zapobiega awariom – zabezpiecza przed zawieszeniami),
- etc.

Elementy te odróżniają mikrokontroler od mikroprocesora. Układ ten jest zdolny do autonomicznej pracy.

Mikrokontrolery wykorzystują następujące rodzaje pamięci:

- EPROM – nieulotna, można je podzielić na jednorazowo programowalne (OTP) lub wielokrotnie, z możliwością kasowania dotychczasowej zawartości (EPROM), używana jest do pamięci programu (ale rzadko)
- EEPROM – nieulotna, kasowana elektrycznie, podobnie jak powyższa, stosowana jest do pamięci programu
- flash – nieulotna, można ją programować bezpośrednio w mikrokontrolerze, również zawiera pamięć programu
- SRAM – statyczna pamięć RAM, czyli ulotna, używana jest do pamięci danych
- FRAM – nieulotne pamięci ferroelektryczne, zastosowane np. w moich „ulubionych” MSP430

Architektury mikrokontrolerów można podzielić ze względu na dostęp do pamięci lub typu listy instrukcji. Zgodnie z wiedzą z Architektury Komputerów wiemy, że istnieją architektury podzielone ze względu na dostęp do pamięci jak: von Neumanna, harvardzka, mieszana.

Architektura von Neumanna cechuje się tym, że ta sama pamięć przechowuje program oraz dane. Dostęp do nich odbywa się przez jedną, wspólną magistralę, a każda komórka posiada unikatowy adres. Kolejnym elementem jest jednostka sterująca, która jest odpowiedzialna za pobieranie i przetwarzanie instrukcji oraz danych trzymanych w tej pamięci.

Architektura harvardzka wyróżnia dwa rodzaje pamięci – na program oraz na dane. W odróżnieniu od poprzedniej architektury, ta pozwala na równoległą komunikację z pamięciami programu oraz danych, ponieważ występują tu osobne magistrale.

Architektura mieszana wyróżnia dwa rodzaje pamięci, jak w architekturze harvardzkiej, jednak z jedną magistralą, jak w architekturze von Neumanna.

Drugim podejściem do podziału na architektury jest podział na listy instrukcji. Pierwszą z nich jest CISC (*Complex Instruction Set Computing*). Zbiór rozkazów jest złożony, wykonanie instrukcji wymagają od kilku do kilkunastu cykli zegara. Duża liczba trybów adresowania oraz liczba rozkazów powodują, że układy dekoderów są skomplikowane. W tej architekturze rozkazy operują bezpośrednio na pamięci.

Innym podejściem jest architektura RISC (*Reduced Instruction Set Computing*). Posiada ona więcej rejestrów w stosunku do CISC, aby zmniejszyć narzut komunikacji z pamięcią. Tryby adresowania zostały zredukowane, aby zmniejszyć stopień skomplikowania dekoderów.

Mikrokontrolery są zdolne do autonomicznej pracy, dzięki dostępnym peryferiom, które wspomagają kontrolę urządzeń zewnętrznych. Można do nich zaliczyć:

- układ licznika w najprostszej postaci jest rejestrem przesuwnym, ważną cechą tego układu jest jego pojemność (np. 8 bit)
- oscylatory generujące sygnały zegarowe, oprócz nich można podłączyć do μC zewnętrzne układy
- watchdog – ważnym elementem pracy μC, ponieważ przywraca działanie układu do normalnego trybu w razie, gdy mikrokontroler przestanie działać poprawnie
- przetworniki analogowo-cyfrowe pozwalające na pracę z analogowymi peryferiami (np. czujnikami temperatury); przetworniki cyfrowo-analogowe pozwalają generować analogowe sygnały na podstawie cyfrowych danych

System przerwań opiera się na sygnałach, które powodują zmianę przepływu sterowania, niezależnie od wykonywanego programu. Przerwanie potrafi wstrzymać aktualnie wykonywany program i wykonać odpowiedni kod do obsługi danego przerwania. Pozwala to na interakcję z podłączonymi urządzeniami (np. przetwarzanie danych, gdy się pojawią), lub obsługę licznika.

Mikrokontrolery pozwalają na komunikację z zewnętrznymi urządzeniami. Niektóre z nich mogą mieć konkretny interfejs komunikacyjny. W μC najważniejszym interfejsem są piny GPIO, które umożliwiają komunikację, ich zachowanie programowane jest przez programistę, domyślnie nie mają określonego zastosowania. Mogą być zastosowane do implementacji interfejsów – np. USART, SPI, I^2C.

USART pozwala na szeregową komunikację z zewnętrznymi urządzeniami. Komunikacja ta może być synchroniczna lub asynchroniczna. Ta forma transmisji jest wykorzystywana w standardach RS232, RS485 (różnią się poziomami logiki). W trybie synchronicznym wykorzystywana jest dodatkowa linia zegara synchronizującego. Magistrala ta wymaga dwóch linii transmisyjnych (RX – odbiór, TX – wysyłanie).

I^2C jest inną magistralą szeregową, pozwala na podłączenie wielu urządzeń. Urządzenia podzielone są na dwie klasy – master oraz slave. Master generuje sygnał zegarowy, rozpoczyna komunikację z podwładnymi urządzeniami. Węzły slave odbierają sygnały zegarowe oraz odpowiadają, gdy ko-

munikacja jest zaadresowana do nich. Magistrala ta wymaga dwóch linii sygnałowych (dane, sygnał zegara).

Inną magistralą komunikacyjną jest SPI. Ona również zezwala na podłączenie wielu urządzeń. Zezwala na komunikację full-duplex, czyli urządzenia mogą się ze sobą komunikować równocześnie. Podobnie jak I^2C, posiada podział na master oraz slave. Wymaga trzech linii komunikacyjnych (sygnał zegara, wyjście danych układu master, wejście danych układu master) oraz jedną linię komunikacyjną na urządzenie (master posiada linie wyboru układu slave).

Programowanie mikrokontrolerów może odbywać się:

- szeregowo w systemie (ISP), poprzez SPI
- przez bootloader – w układach, które mają możliwość modyfikacji programu, przesyłanie może odbyć się np. przez UART
- przez JTAG – wykorzystywane do programowania i debugowania, początkowo służył do weryfikacji poprawności montażu układów scalonych

Układy można skonfigurować przy pomocy lock bitów (zabezpieczają przed nieuprawnionym dostępem do pamięci programu) oraz fuse bitów (konfiguracja układu, źródeł sygnału zegarowego, watchdoga, etc.).

Mikrokontrolerów zazwyczaj nie posiadają systemu operacyjnego, chociaż istnieją rozwiązania RTOS (np. uClinux, freeRTOS). Programowanie zwykle wykorzystuje asemblera bądź C ze wstawkami asemblera. Konwencja tworzenia programów na mikrokontrolery opiera się na tym, że procesor powinien wykonywać zadania, mimo wystąpienia zdarzeń losowych, czy też podczas oczekiwania na jakieś przerwanie. W niektórych zastosowaniach może być ważne również usypianie i wybudzanie mikrokontrolera w celu oszczędzania zasilania, gdy μC oczekuje na np. dane.

20 S10 – Systemy wbudowane w strukturach programowalnych

Przypomnienie – **struktury programowalne** to układy elektroniczne, które mają programowalną strukturę, mogą działać jak dowolny układ cyfrowy (o ile zasoby na to zezwalają). Funkcjonalność jest definiowana przez programistę, a nie producenta, co pozwala zmniejszyć koszty, bo z punktu widzenia producenta, produkowane układy są identyczne. PLD są programowane za pomocą języków opisu sprzętu (najpopularniejsze – Verilog, VHDL), a narzędzie syntezy przetwarza kod programu na fizyczne połączenia. Wyróżnia się układy PLD takie jak:

- SPLD – PLA, PLE, PAL; najprostsze, mają ograniczone możliwości funkcjonalne i logiczne,
- CPLD – koncepcyjnie podobne do SPLD, ich architektura opiera się na makrokomórkach, które tworzą bloki funkcyjne, łączone są przez matrycę połączeń,
- FPGA – najbardziej zaawansowane, składają się z CLB łączonych w bloki, łączone są przez linie traktów połączeniowych.

Więcej informacji można znaleźć w punkcie K9, nie ma się co tu rozpisywać więcej, bo ileż tu można upchnąć ciekawych informacji `:-)))`.

System wbudowany – system komputerowy o specjalnym przeznaczeniu, jest integralną częścią obsługiwanego przez niego sprzętu. System ten spełnia wymagania, które zostały określone do zadań, które ma wykonywać. Tak więc zgodnie z tą definicją komputer osobisty nie jest systemem wbudowanym. Systemy wbudowane wykorzystują mikroprocesory bądź mikrokontrolery, które są zaprogramowane do wykonywania tylko i wyłącznie tychże zadań (mają dedykowane oprogramowanie) – dlatego są nazwane systemami szczególnego przeznaczenia. Systemy te można także oprzeć na strukturach programowalnych. Dzięki wyspecjalizowaniu tych systemów, pobierają one mniej energii, mają mniejsze rozmiary w stosunku do systemów ogólnego przeznaczenia. Systemy te także cechują się wysoką niezawodnością – są odporne na awarie i zakłócenia.

Z systemami wbudowanymi można się spotkać na każdym kroku. Występują one m.in. w:

- komputerach pokładowych (w aucie, samolocie, etc.),
- systemach naprowadzania rakiet (`;-)`),
- sprzęcie medyczny,
- bankomatach,
- „yntelygentnych” urządzeniach (telewizory, lodówki, boję się pomyśleć co jeszcze),
- routerach,
- systemach alarmowych,
- drukarkach,
- odtwarzaczach DVD/Blu-Ray/etc.,
- można tak wymieniać do wieczora, ale chyba komisji to się zbytnio nie spodoba.

System wbudowany składa się z trzech głównych części:

- obwody wejściowe (czujniki, przetworniki A/C lub C/A, etc.) – zbierają dane,

- jednostka centralna – pobiera dane, przetwarza je przez swój algorytm oraz przekazuje na obwody wyjściowe,
- obwody wyjściowe – mogą to być porty komunikacyjne, czy też wyjścia sterujące.

Realizacja systemów wbudowanych na strukturach programowalnych może się odbyć na CPLD lub FPGA (SPLD nie są wystarczająco duże).

Układy CPLD oraz FPGA można zaprogramować programowymi procesorami. Jednym z nich, dosyć popularnym jest PicoBlaze, (8-bitowy). Jest on stworzony w HDLu. Posiada on cechy podobne do istniejących na rynku mikrokontrolerach (takich jak: 8051, ATtiny13, ST6215). W odróżnieniu od nich, pozwala na dowolną liczbę peryferii, ograniczeniem jest liczba bloków wybranego PLD.

Istnieją także inne mikroprocesory, którymi można zaprogramować układy PLD, jednakże wymagają one FPGA (nie CPLD). Są to: MicroBlaze, OpenRISC, ARM Cortex-M1 (oczywiście, jest ich więcej). MicroBlaze jest podobny do PicoBlaze, jednak jest to procesor programowy 32-bitowy, posiada większą liczbę zasobów, oraz wyposażony w jednostkę FPU (czyli obsługuje nasze ulubione liczby zmiennoprzecinkowe). Z ciekawostek, obsługa MicroBlaze jest zawarta w kodzie źródłowym jądra Linuksa.

Programowanie takich procesorów programowych odbywa się w dwóch etapach. Pierwszy z nich polega na definicji układu przy pomocy HDLa – tutaj jest ustalana „budowa" procesora. Następnie jest to przekształcane na sprzętową implementację jako układy kombinacyjne lub sekwencyjne. Kolejny krok to stworzenie kodu programu w Asemblerze bądź C/C++, a następnie zaprogramowanie procesora.

Łączenie mikroprocesorów i układów programowalnych może się odbyć na cztery sposoby:

1. **Mikroprocesor/mikrokontroler**

   Połączenie mikroprocesora z peryferiami w jeden lub kilka układów scalonych.

2. **μprocesor/μkontroler + PLD**

   Rozwiązanie to pozwala rozszerzać systemy procesorowe o dodatkowe możliwości, które dają układy programowalne.

3. **PLD zintegrowane z μprocesorem / inaczej: μkontroler z wbudowaną częścią FPGA w jednym układzie scalonym**

   Układy PLD z wbudowanym procesorem, np. Xilinx Virtex II + PowerPC. Rozwiązanie to nie przyjęło się ze względu na wysoką cenę.

4. **Układy PLD z zaimplementowanym μprocesorem**

   Rozwiązanie używane do projektowania i weryfikacji projektów przez firmy produkujące układy scalone. Wraz ze wzrostem wydajności i pojemności tańszych układów ta konfiguracja zaczyna być konkurencyjna dla mikrokontrolerów. Rozwiązanie to wydaje się być optymalne dla sprzętowych procesorów (PicoBlaze, MicroBlaze, etc.).

Układy programowalne dają elastyczność, struktura systemu wbudowanego o taki układ może być dostosowana ściśle do potrzeb oraz może być w prosty sposób zmieniona. Gotowe mikroprocesory nie pozwalają na manipulację liczbą portów czy możliwych peryferii do podłączenia. Można zbudować system oparty o wiele procesorów, gdzie każdy ma przydzielone inne zadanie do wykonania.

Wadą takiego rozwiązania jest większy pobór mocy.

Układy programowalne pozwalają na zrealizowanie dowolnych peryferii wewnątrz układu, np. układy DVB, szyfrujące, bloki przetwarzania sygnałów, porty szeregowe i równoległe, magistrale, etc.

Część III

Pytania specjalnościowe – INS

21 S1 – Konfiguracja sieciowa systemów operacyjnych (sterowniki urządzeń sieciowych, ustawienia parametrów sieci lokalnej i TPC, automatyzacja konfiguracji)

Sieciowy system operacyjny to system, którego jądro ma zintegrowany stos sieciowy. Mądrze brzmi, ale co to znaczy? Znaczy to tyle, że jądro takiego systemu posiada

- sterowniki obsługujące karty sieciowe,
- oprogramowanie obsługi co najmniej trzech najniższych warstw, tj. łącza danych, sieciowej i transportowej
- funkcje biblioteczne wspierające gniazdka Berkeley, Windows bądź interfejs TLI.

21.1 Urządzenia

W systemie możemy wyróżnić między innymi następujące trzy rodzaje urządzeń.

- Znakowe – są to urządzenia, które obsługują odczyt i zapis znak po znaku. Nie są buforowane i interakcja jest natychmiastowa. Przykład: drukarka.
- Blokowe – urządzenia te umożliwiają odczyt i zapis całych bloków. Są buforowane i umożliwiają dostęp do danych w nich przechowywanych. Przykład: dysk, plik.
- Interfejsy sieciowe – zamiast znaków czy bloków, obsługują pakiety.

Każde z tych urządzeń jest obsługiwane przez odpowiedni sterownik – część jądra systemu, odpowiedzialną za obsługę urządzeń. Sterowniki można dołączać do jądra **statycznie** – w trakcje kompilacji – oraz **dynamicznie** – w formie ładowalnych modułów.

Interfejsy można konfigurować lokalnie, za pomocą plików konfiguracyjnych i narzędzi typu ifconfig. Jednak istnieją również rozwiązania pozwalające na automatyczną konfigurację systemu w momencie jego uruchomienia i włączenia do sieci.

21.2 BOOTP

Jest to protokół służący do konfiguracji komputerów. Taki komputer po uruchomieniu rozsyła na adres rozgłoszeniowy zapytanie BOOTP. Kiedy serwer odbierze to zapytanie, wysyła klientowi odpowiedź BOOTREPLY zawierającą konfigurację.

Tego protokołu używano również do pobierania obrazu systemu operacyjnego. Pozwalało to na używanie prostych terminali, które nie posiadały własnych dysków, a całość konfiguracji ściągały z sieci i ładowały do pamięci operacyjnej.

21.3 DHCP – Dynamic Host Configuration Protocol

Następcą BOOTP, jest protokół DHCP. W odróżnieniu do BOOTP, protokół ten jest przeznaczony dla maszyn posiadające własne dyski i skupia się na automatycznej konfiguracji komputerów, które często mogą zmieniać swoją fizyczną lokalizację. Potrafi komunikować się z komputerem również po uruchomieniu się systemu, a nie tylko przed, dzięki czemu łatwiejsze staje się odnawianie dzierżawy adresu – komputera nie trzeba uruchamiać ponownie.

21.4 PXE – Preboot eXecution Environment

Protokół ten jest następcą BOOTP jeśli chodzi o bootowanie systemu z sieci. Jest on zbudowany na bazie DHCP (do uzyskania adresu i maski) oraz TFTP (do transferu danych). Gdy komputer uzyska konfigurację sieciową, zaczyna odpytywać sieć w poszukiwaniu serwera TFTP. Gdy go znajdzie to pobiera z niego obraz systemu. Protokół ten jest aktualnie standardem pozwalającym na szybkie i proste skonfigurowanie komputerów, np. pracowników dużych firm, których komputery powinny posiadać pewną określoną polityką firmy konfigurację.

Ze względu na konieczność bycia łatwymi do zaimplementowania, podane wyżej protokoły opierają się na UDP, a nie TCP.

22 S2 – Mechanizmy zdalnego dostępu do zasobów sieciowych (dyski sieciowe, mapowanie uprawnień dostępu, sieciowe zarządzania użytkownikami NIS/LDAP)

Udostępnianie zasobów w sieci jest bardzo wygodnym rozwiązaniem. Zwalnia nas z potrzeby wymieniania się nośnikami danych i kopiowania danych między komputerami. Rozwiązań jest wiele i od naszych wymagań czy możliwości zależy to, które wybierzemy.

22.1 FTP

Jedną z najbardziej rozpowszechnionych usług tego typu są serwery FTP oraz ich bezpieczne odpowiedniki SFTP (tunelowanie ssh) oraz FTPS (ssl/tls).

FTP (File Transfer Protocol) jest to usługa, która pozwala nam na udostępnienie miejsca na serwerze, z którego ludzie mogą pobierać lub wysyłać dane. Może ona działać w dwóch trybach, z których oba potrzebują dwóch połączeń – do przesyłania poleceń oraz danych. Tryby te to

- aktywny – gdy klient łączy się do serwera i wymienia polecenia, a dopiero później ustanawia nowe połączenie na porcie N+1 w celu przesłania danych,
- pasywny – gdy klient od samego początku ustanawia oba połączenia.

22.2 NAS

Będąc w temacie zasobów sieciowych nie można oczywiście pominąć dysków sieciowych. Są to urządzenia wpinane do sieci – zazwyczaj kablem ethernetowym – lecz dla użytkowników są widoczne jak zasoby lokalne. Daje nam to możliwość udostępnienia zasobów raz – nie musimy wymieniać się nośnikami i kopiować danych.

22.3 NFS

Jednak nie trzeba w celach udostępniania danych w sieci kupować dedykowanych urządzeń. Można skorzystać z protokołu NFS, który umożliwia udostępnienie swoich danych w sieci. Udostępnione pliki można zamontować w swoim systemie i operować jak na zasobach lokalnych.

22.4 SMB

Innym protokołem jest SMB.

22.5 Uprawnienia

NFS musi oczywiście sprawdzić, czy mamy uprawnienia dostępu do zasobów, co sprowadza nas do tematu uprawnień i zarządzania użytkownikami. W przypadku NFSa, system sprawdza czy nasz użytkownik i jego uprawnienia zgadzają się z tym co jest na serwerze – jeśli spróbujemy odczytać

coś do czego nasz użytkownik nie ma dostępu to się nam to nie uda. Konfigurację przeprowadza się za pomocą pliku /etc/exports.

22.6 Zarządzanie użytkownikami

Dla zarządzania użytkownikami również istnieją różne rozwiązania, m.in. *NIS, LDAP* czy *Active Directory.*

NIS (Yellow Pages) jest protokołem umożliwiającym współdzielenie informacji o użytkownikach. Działa on w architekturze serwer-klient, gdzie serwer posiada bazę z informacjami na temat użytkowników, a klient z tej bazy może skorzystać. Można więc tak ustawić sieć, że gdy użytkownik próbuje się zalogować do komputera, a w lokalnym systemie nie widzimy takiego użytkownika, komputer odpytuje serwer z bazą danych i na tej podstawie weryfikuje czy użytkownik może uzyskać dostęp.

NIS ma jednak tę wadę, że klient może pobrać całą bazę, a następnie przeprowadzić atak siłowy celem złamania haseł.

Następnikiem NISa jest LDAP. LDAP podobnie jak NIS działa w architekturze serwer-klient, jednak różnica polega na tym, że weryfikacja odbywa się po stronie serwera.

Przykładem, gdzie takie rozwiązanie jest wdrożone może być sama politechnika. Studenci posiadają konta na poczcie, którego można użyć do zalogowania się do sieci Eduroam. Takie rozwiązanie jest również wdrożone w niektórych laboratoriach.

23 S3 – Metody rozwiązywania problemu martwego punktu (impasu) w systemach i sieciach komputerowych

Impas, inaczej zakleszczenie, jest pojęciem znanym z programowania współbieżnego. Jest to sytuacja, w której procesy, bądź wątki czekają na siebie nawzajem i żadna z tych operacji nie może zostać ukończona.

Do takiej sytuacji może dojść np. w przypadku, gdy proces A żąda zasobu, który jest zajęty przez proces B, a proces B żąda zasobu zajętego przez proces A. Wychodzi wtedy cykliczna zależność, która blokuje system.

Pojęciem powiązanym z zakleszczeniem jest *zagłodzenie.* Przykładem jest *problem ucztujących filozofów.* Problem polega na tym, że procesy wymagają do działania dwóch zasobów – widelców. Jeśli jeden filozof zacznie jeść, to dwaj obok niego automatycznie nie mogą jeść.

Metody walki z tym problemem dzielą się na

- wykrywanie i likwidację,
- zapobieganie,
- unikanie.

23.1 Wykrywanie i likwidacja

Polega na cyklicznym sprawdzaniu stanu systemu i wykrywaniu czy nie pojawiły się w nim blokady. W przypadku ich wykrycia, system może odłączać poszczególne zasoby od puli zasobów zajętych, aż do momentu usunięcia blokady. Takie podejście jest oczywiście zarezerwowane dla systemów, w których istnieje możliwość wywłaszczenia zasobów.

23.2 Zapobieganie (statyczne)

Do wystąpienia blokady, konieczne jest spełnienie czterech warunków

- wyłączność – zasoby nie mogą być współdzielone (w danej chwili zasób może być wykorzystywany tylko przez jeden proces,
- niewywłaszczalność – zasób nie może zostać odebrany procesowi przed zakończeniem operacji,
- przetrzymywanie i czekanie – procesy zajmują zasób i oczekują na kolejny,
- cykliczne oczekiwanie – istnieje cykl procesów oczekujących na siebie wzajemnie.

Aby zapobiegać blokadom, należy tak projektować system, by przynajmniej jeden z tych warunków nie został spełniony, tj.

- pozwolić na jednoczesny dostęp do zasobów lub sprawić, by procesy nie korzystały z tych samych zasobów,
- pozwolić na wywłaszczenie zasobów,
- wymuszenie, by proces zajął wszystkie potrzebne mu zasoby albo nic,

•

23.3 Unikanie (dynamiczne)

Podczas gdy zapobieganie blokadom opiera się na rozwiązaniach statycznych, tj. implementowanych przy projekcie systemu, unikanie blokad wykorzystuje aktualne dane o stanie systemu, co pozwala odpowiednio zareagować.

23.3.1 Algorytm bankiera

Pozwalamy procesowi na zajęcie zasobu, jeśli po jego zajęciu będzie istniała możliwość uporządkowania procesów tak, by uniknąć blokady.

Porządkowanie zasobów dla procesów w ogólnym przypadku jest problemem **NP-zupełnym,** co znaczy, że nie jest to łatwe zadanie, a heurystyczne algorytmy mogą odrzucać część prawidłowych stanów.

23.4 Scentralizowane i rozproszone

Algorytmy można dzielić ze względu na różne kryteria. Jednym z takich podziałów jest podział na algorytmy scentralizowane oraz rozproszone.

23.4.1 Scentralizowane

Są to algorytmy opierające się na znajomości stanu globalnego systemu.

23.4.2 Rozproszone

Są to algorytmy opierające

23.5 Optymalne i suboptymalne

Algorytmy można również dzielić na

- optymalne – algorytm doskonale sobie radzi i odrzuca jedynie stany niebezpieczne,
- suboptymalne – jest możliwe, że algorytm odrzuci część stanów bezpiecznych.

24 S4 – Metody równoważenia obciążeń w systemach i sieciach komputerowych

Równoważenie obciążenia w systemach nie jest banalnym tematem. Wiele procesów wymaga dostępu do zasobów, w tym do czasu procesora.

Za kolejkowanie procesów odpowiada **planista.** Jest to proces, którego zadaniem jest między innymi przydzielenie procesom miejsca w kolejce.

24.1 Rodzaje planistów

Wyróżniamy planistów

- krótkoterminowych – odpowiadających za procesy oznaczone jako *gotowe,*
- średnioterminowych – odpowiadających za ładowanie i usuwanie procesów z pamięci (swap),
- długoterminowych – odpowiadających za uruchamianie nowych procesóœ.

24.2 Algorytmy planowania

Same algorytmy, wg. których działać może planista możemy podzielić na wywłaszczeniowe oraz niewywłaszczeniowe.

24.2.1 Algorytmy niewywłaszczeniowe

To takie, które nie dopuszczają sytuacji, w której proces, który jest aktualnie obsługiwany, zostaje wywłaszczony – przestaje się go obsługiwać przed zakończeniem.

Do takich algorytmów można zaliczyć kolejkowanie

- SJF - Shortest Job First,
- FIFO - First In First Out,
- LIFO - Last In First Out.

24.2.2 Algorytmy wywłaszczeniowe

Te algorytmy pozwalają, a nawet opierają się na tym, że proces zostaje wywłaszczony. Można tutaj wspomnieć między innymi o

- Round-robin – każdy proces dostaje kwant czasu, po którego upływie zostaje wywłaszczony,
- Shortest Remaining Time – proces zostaje wywłaszczony kiedy zjawi się proces, o krótszym czasie obsługiwania
- Podejście priorytetowe – obsługiwany jest proces o najwyższym priorytecie.

25 S5 – Źródła zagrożeń bezpieczeństwa systemów i usług informatycznych

25.1 Natura

Jednym z zagrożeń dla systemów, które istnieje, a nie zawsze się o nim myśli jest natura. Woda może zalać sprzęt, piorun uszkodzić instalacje, a zwierzątka poprzegryzać kable i należy się przed tym zabezpieczać.

25.2 Użytkownicy

Jednak głównym źródłem zagrożeń dla systemów są ludzie. Nawet jeśli zaprojektujemy system, który w naszej opinii jest wyjątkowo dobrze zabezpieczony, to ktoś zawsze znajdzie sposób, by nam udowodnić, że się mylimy (czy to przez przypadek, czy umyślnie).

Użytkownicy ustawiają słabe hasła, które łatwo złamać, bądź też udostępniają je na lewo i prawo. Robią coś i nie pomyślą nawet, że ich działania mogą przyczynić się do ujawnienia tajnych danych czy zakażenia komputera wirusem.

25.3 Socjotechniki

Ludzie są łatwowierni. Jeśli wejdziemy do jakiegoś miejsca odpowiednio ubrani i będziemy sprawiać wrażenie, że dokładnie wiemy co robimy to nikt nas nie zatrzyma. Można więc w okolice firmy podrzucić odpowiednio zakażony pendrive lub wysłać wiadomość podając się za przedstawiciela banku i poprosić o udostępnienie danych.

25.4 Haksy

Również ludzie w postaci programistów są odpowiedzialni za błędy w oprogramowaniu, które mogą pozwalać na nieautoryzowany dostęp do systemu. Programiści mogą chcieć ułatwić sobie pracę tworząc tak zwane *backdoory* i później ich nie usuwają (ponownie, specjalnie bądź przez przypadek).

Oczywiście poza przyczynami losowymi i czynionymi nieumyślnie mamy też do czynienia z niebezpieczeństwem ze strony ludzi, którzy umyślnie chcą spowodować szkody. Do zagrożeń ze strony takich osób należą

25.4.1 Sniffery

Oprogramowanie podsłuchujące dane przepływające w sieci korzystając z faktu, że niestety internet był projektowany w czasach, gdy bezpieczeństwo w sieci nie było wielkim tematem i dane w niższych warstwach są przesyłane w postaci jawnej. Wykrywanie ich polega na wykorzystaniu zasady działania snifferów – wykorzystują one kartę sieciową przełączoną w tryb *promiscuous,* aby przechwytywać pakiety, które nie są do nich adresowane.

Można więc taką kartę wykryć wysyłając do podejrzanego hosta pakiet ICMP ECHO z jego adresem IP, ale cudzym adresem MAC. Praworządna karta nie powinna takiego pakietu przyjąć, lecz karta podsłuchująca wyśle odpowiedź i w ten sposób można poznać kto jest kto.

25.4.2 Denial of Service

Atak typu DoS polega na zajęciu zasobów systemowych (czas CPU, miejsce na dysku) w taki sposób, że inne procesy i użytkownicy nie są w stanie z nich korzystać. Atak taki można również przeprowadzić z sieci, np. poprzez zasypywanie serwera pakietami.

Walczyć z takim atakiem można poprzez ustawienie limitów w systemie, np. limit na miejsce przeznaczone dla jednego użytkownika, czy liczbę procesów mogących operować jednocześnie.

25.4.3 Spoofing

Podszywanie się pod kogoś innego.

- DNS – wysłanie do serwera DNS fałszywej informacji kojarzącej domenę z adresem IP.
- ARP – rozsyłanie w sieci spreparowanych pakietów ARP zawierających fałszywe adresy MAC, co powoduje przesłanie danych w złe miejsce.

25.4.4 Reszta

- SQL injection
- keyloggery
- przepełnienia bufora
- wirusy psotnicy

26 S6 – Metody i mechanizmy zapewniania bezpiecznego dostępu i bezpiecznej komunikacji sieciowej w systemach komputerowych

Temat bezpieczeństwa systemów i sieci należy zacząć od wspomnienia o najważniejszym elemencie bezpiecznego systemu, którym jest **użytkownik.**

Możemy zaprojektować najlepszy system, który będzie wykorzystywał najlepsze metody szyfrowania danych, lecz jeśli użytkownik nie będzie postępował zgodnie z zasadami, to system nigdy nie będzie bezpieczny.

Przykładowo możemy wymusić na użytkownikach, by używali długich i skomplikowanych haseł, jednak najprawdopodobniej doprowadzi to do zapisania takiego hasła gdzieś przy komputerze – w takiej sytuacji tego hasła mogłoby równie dobrze nie być.

Opisane wyżej postępowanie nazywa się mianem *bezpieczeństwa kosztem używalności.* Takie postępowanie bardzo często prowadzi do zmniejszenia bezpieczeństwa systemu, a nie do jego ulepszenia.

Zanim przejdziemy dalej, powiedzmy sobie o trzech ważnych pojęciach.

- Identyfikacja – przedstawienie systemowi swojej tożsamości, którą będzie on weryfikował w następnym kroku.
- Uwierzytelnianie – polega na przedstawieniu systemowi pewnych dowodów, które przekonają go, że „rozmawia" z odpowiednią osobą.
- Autoryzacja – jest osobnym procesem i polega na otrzymaniu dostępu do zasobów lub usług systemu, a co za tym idzie, jest ściśle powiązana z uwierzytelnianiem.

26.1 Hasła

Podstawowym mechanizmem zabezpieczania systemów są hasła.

26.1.1 Hasła wielokrotnego użytku

Najczęściej są one przechowywane w postaci zaszyfrowanej, a podczas uwierzytelniania hasło podane przez użytkownika przepuszczane jest np. przez funkcję skrótu i porównywane z wersją przechowywaną w systemie. Jeśli system uzna, że hasła są zgodne to może udzielić dostępu.

Hasła tradycyjne mają tę wadę, że można je złamać, wykraść czy zgadnąć. Dlatego, choć są wygodne, nie są najlepszą metodą zabezpieczania systemów.

26.1.2 Hasła jednorazowe

Alternatywą dla haseł tradycyjnych są hasła jednorazowe.

Przykładem mogą być hasła odczytywane z listy, która zawiera wcześniej wygenerowane hasła, wygasające po jednym użyciu. Takie rozwiązanie obok haseł SMS często spotkać można w bankach.

Również w bankach można natknąć się na hasła generowane za pomocą tokenów. Takie hasła zmieniają się automatycznie np. co 5 minut i ich ważność wygasa w momencie użycia.

26.2 Szyfrowanie

Równie ważnym tematem jest szyfrowanie danych.

Polega ono na zamianie danych z postaci jawnej do postaci zaszyfrowanej przy pomocy pewnego algorytmu szyfrującego i klucza, który będzie służył do odkodowania danych.

Metody szyfrowania możemy podzielić na symetryczne oraz asymetryczne.

- Symetryczne – istnieje jeden klucz używany zarówno do szyfrowania i odszyfrowania.
- Asymetryczne – istnieje para kluczy, z których jeden służy do szyfrowania, a drugi do odszyfrowania.

Rozwiązanie asymetryczne stosuje się między innymi w GPG (Gnu Privacy Guard), które służy między innymi do szyfrowania i podpisywania korespondencji. Kiedy dwie osoby chcą się komunikować za pomocą tej metody, generują dla siebie po parze kluczy, z których jeden jest określany mianem *prywatnego* i jest znany tylko właścicielowi, a drugi jest kluczem *publicznym,* który zostaje udostępniony. Gdy ktoś zaczyna tworzyć maila, szyfruje go za pomocą klucza publicznego adresata. Dzięki temu tylko osoba posiadająca klucz prywatny, pasujący do klucza publicznego, którym podpisana zostałą wiadomość, będzie w stanie ją odczytać.

26.3 Podpis

Podpis cyfrowy ściśle wiąże się z opisanym wcześniej szyfrowaniem asymetrycznym.

Wykorzystuje się tutaj fakt, że wiadomość, którą da się odszyfrować kluczem publicznym, musiała zostać zaszyfrowana pasującym kluczem prywatnym. Co za tym idzie, została wysłana przez osobę posiadającą właśnie ten klucz prywatny.

Podpis cyfrowy służy do potwierdzania autentyczności pochodzenia wiadomości, ale także pozwala wykryć nieautoryzowane próby zmiany dokumentu.

27 S7 – Różnice pomiędzy introspekcją i odzwierciedleniem – metodami stosowanymi do rozpoznania własności klas lub zmodyfikowania zachowania się aplikacji działających na wirtualnej maszynie Java

Języki wysokiego poziomu umożliwiają tworzenie programów, które są w stanie w **czasie wykonania** dokonać inspekcji i modyfikacji innych programów, bądź też samych siebie.

Pojęcia introspekcji i refleksji (odzwierciedlenia) nie są unikalne dla języka Java, jednak właśnie na tym języku się skupię (taka natura pytania).

Są to pojęcia ściśle ze sobą powiązane, jednak nie są tym samym.

27.1 Introspekcja

Introspekcja umożliwia programowi inspekcję kodu programów, które już się wykonują (w tym też samego siebie). Dzięki niej, możemy uzyskać informacje o używanych typach i zdefiniowanych metodach.

Przykładem ze środowiska Javy, gdzie jest używana introspekcja, mogą być narzędzia służące do graficznego projektowania interfejsu użytkownika.

Korzystają one z mechanizmu introspekcji do prześwietlania kodu i wydobywania informacji z ziarenek Javy (JavaBeans), aby z nich skorzystać.

27.2 Refleksja

Refleksja korzysta z mechanizmu introspekcji i idzie o krok dalej. Oprócz samego wydobywania informacji z programu, refleksja umożliwia modyfikowanie go.

Możemy więc wydobyć informacje z jakiegoś obiektu, o którym nic nie wiemy, a następnie wybrać, które metody wywołać i w jaki sposób.

Przykładem, gdzie przydaje się refleksja może być testowanie. Za pomocą mechanizmu refleksji, framework JUnit wyszukuje metody oznaczone za pomocą adnotacji @Test i w ten sposób wie, które metody służą do testowania i może je wywołać.

W odpowiednich rękach jest to potężne narzędzie, które umożliwia nam pracę z kodem, który mógł jeszcze nie zostać nawet napisany. Należy jednak pamiętać, że ze względu na to, że wszystko dzieje się w czasie wykonania, nadmierne wykorzystywanie mechanizmu refleksji może negatywnie wpłynąć na wydajność programu.

28 S8 – Sposoby budowy i zarządzania aplikacjami rozproszonymi za pomocą pakietów należących do standardowej dystrybucji Javy

Aplikacje rozproszone to takie, które są wykonywane na wielu maszynach celem rozłożenia obciążenia.

Takie aplikacje można zrealizować za pomocą zdalnego wywoływania metod obiektów, bądź przesyłania komunikatów.

28.1 Gniazda

Przesyłanie komunikatów wiąże się z wykorzystaniem gniazd sieciowych, a budowa aplikacji opiera się na architekturze klient-serwer. Samą komunikację można podzielić na strumieniową (TCP) oraz datagramową (UDP).

Wygląda to tak, że zarówno klient jak i serwer otwierają gniazda sieciowe. Następnie serwer oczekuje na nadejście nowych połączeń. Klient podaje z jakim adresem IP chce się połączyć, a konkretna usługa jest rozpoznawana po numerze portu. Gdy serwer odbierze połączenie, otwiera nowe gniazdo, przez które jest kontynuowana wymiana, a stare gniazdo istnieje w celach odbierania nowych połączeń.

28.2 RMI

Innym sposobem realizacji takiej aplikacji, który poznaliśmy w czasie studiów jest RMI. RMI dostarcza mechanizmy pozwalające na udostępnianie oraz wywoływanie metod zdalnych obiektów.

Obiekty, których metody będą wywoływane zdalnie, przede wszystkim muszą implementować pewien konkretny interfejs. Następnie, namiastka tych obiektów musi zostać zarejestrowana w rejestrze RMI. Będzie ona służyła do wywoływania ich metod.

Kiedy klient chce wywołać metodę zdalnie, komunikuje się z rejestrem RMI i pobiera informacje o namiastkach i ich metodach. Po pobraniu namiastek, może on działać na nich tak jakby to były zwyczajne obiekty lokalne – komunikacja jest więc dla użytkownika niewidoczna.

29 S9 – Dostęp do internetowych baz danych na przykładzie wybranej technologii

Internetowa baza danych to oczywiście baza danych dostępna w Internecie. Bazy danych istnieją po to, by przechowywać dane w celu ich późniejszego wykorzystania.

29.1 ORM

Pojęciem niezwykle ważnym w tym temacie jest Mapowanie Obiektowo Relacyjne. Pozwala ono na przetworzenie danych z postaci przechowywanej w bazie relacyjnej do postaci obiektów w wybranym języku programowania.

Pozwala to na dostęp do danych z zachowaniem wysokiego poziomu abstrakcji przy użyciu języka, z którym programiści są najlepiej obeznani. Nie musimy więc pisać skomplikowanych zapytań SQLowych. Wystarczy jedynie odwołać się do zmapowanych obiektów i ich metod, co jest wyjątkowo wygodne i pozwala na znaczne przyśpieszenie pisania aplikacji.

Ma to również tę zaletę, że odpowiedzialność za takie rzeczy jak odporność na SQL Injection przynajmniej częściowo zrzucamy na bibliotekę i jednocześnie w pewien sposób uniezależniamy się od konkretnej implementacji bazy danych – to biblioteka dba o to, by odpowiednio przetłumaczyć nasze polecenia.

Jako że pytanie wyraźnie mówi o dostępie na przykładzie wybranej technologii, wypowiem się o tym jak to wygląda w Pythonie.

29.2 DjangoORM

Jednym z dostępnych rozwiązań w świecie Pythona jest DjangoORM. Jest to system mapowania stworzony na potrzeby Frameworka webowego Django jednak po odpowiedniej konfiguracji można go używać bez części webowej.

Działa on w taki sposób, że po skonfigurowaniu, komunikacja z bazą danych odbywa się za pośrednictwem obiektów utworzonych w Pythonie. Stworzenie tabeli jest tak proste jak zdefiniowanie klasy. Tworzymy więc klasę o takiej nazwie jak nowa tabela i umieszczamy w niej pola, które odpowiadają kolumnom.

Możemy zatem napisać

```
class Person(models.Model):
        first_name = models.CharField(max_length=30)
```

i po przeprowadzeniu migracji – operacji uaktualnienia schematu bazy danych – na podstawie tej klasy zostanie utworzona nowa tabela.

Wykonywanie zapytań jest równie łatwe. Wystarczy wywołać odpowiednią metodę jak *all()*, a stworzenie nowego rekordu jest tak proste jak utworzenie nowego obiektu i zapisanie go.

29.3 Inne

Poza DjangoORM istnieją również inne rozwiązania takie jak Peewee czy SQLAlchemy.

29.4 Wady

Jednak żadna technologia nie jest idealna. Mimo swoich niewątpliwych zalet, tj. przyśpieszenie pisania i praca na wyższym poziomie abstrakcji, ORM posiada kilka wad.

System nie zawsze jest w stanie wygenerować optymalne zapytania – dla aplikacji, dla których kluczowa jest szybkość działania warto zastanowić się nad ręcznym pisaniem SQL.

Tak jak w każdym przypadku – przede wszystkim należy zastanowić się czy technologia pasuje do naszych wymagań, a nie traktować jej jak *srebrny pocisk.*