ÚVOD

2 základné spôsoby komunikácie:

1. spájaná komunikácia:
- nadviazanie spojenia
- alokácia zdrojov (vyhradenie zdrojov) potrebných na komunikáciu
- vlastná komunikácia
- rozviazanie spojenia, uvoľnenie zdrojov

napr. rozhovor 2 ľudí, telefonický hovor, posielanie listov

2. nespájaná komunikácia:
- dáta sú posielané „naslepo", nedochádza k nadviazaniu spojenia

napr. prednáška, rádiové a televízne vysielanie, rozosielanie letákov

Spôsob prenosu:
- prúdový – po jednotlivých znakoch (napr. USB)
- blokový – po väčších blokoch

Blok sa prenesie alebo neprenesie ako celok. Nazývajú sa rôzne: paket, datagram, frame (rámec). Každý blok má metainformáciu (dĺžka, kontrolný súčet, adresy, ...)

Vďaka tejto informácii sa prenesie dobre

1. Prepájanie okruhov (circuit switching)
- spájané, sieť je „múdra", drží v sebe kontext spojenia napr. telefón
- drahé, spoľahlivé

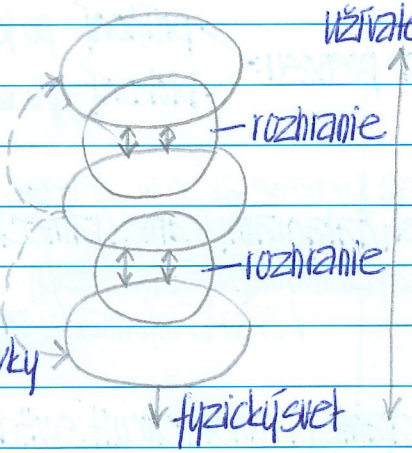
2. Prepájanie paketov (packet switching)
- nespájané, sieť nedrží kontext spojenia, iba smeruje pakety
- koncové zariadenie drží kontext napr. internet
- lacné, nespoľahlivé

Protokol je súbor pravidiel o tom, ako komunikovať.
- účelom je **jednoznačnosť komunikácie**
- pri počítačovej komunikácii je treba, aby sa dala **naprogramovať**
- nároky na presnosť formulácií v dokumentoch špecifikujúcich komunikačný protokol sú veľké
- autor komunikačného protokolu potrebuje komunikovať (nespájane) s ľuďmi, ktorí budú implementovať protokol
→ vzniká potreba **komunikačného protokolu o špecifikácii komunikačného protokolu**

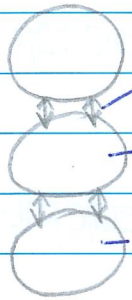
Nútená č.1 zjednotenie terminológie
→ toto viedlo k vytvoreniu referenčného modelu ISO/OSI
- abstraktný model dátovej siete
- autori: svetoví telekomunikační giganti → vymysleli **vrstvový model**

Vrstvy (úrovne abstrakcie):
1. fyzická
2. spojová
3. sieťová
4. transportná
5. relačná
6. prezentačná
7. aplikačná

poskytuje služby → „nadriadenej" (abstraktnejšej) vrstve
„podriadenej" posiela požiadavky



Účel



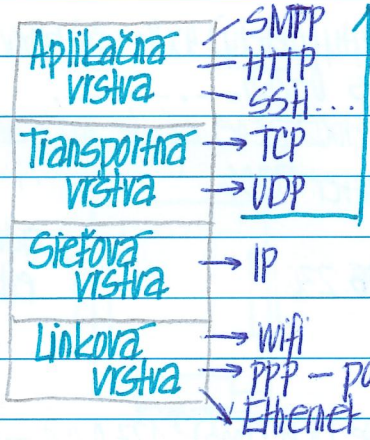
toto rozhranie zostane nedotknuté (nemusíme vš. preprogramovať)
musíme zmeniť aj toto
zmeníme túto časť

ISO/OSI – International Organization for Standarization / Open Systems Interconnection

TCP/IP MODEL

Prvé verzie internetových protokolov boli vymyslené v DARPA (Defense Advanced Research Project Agency)

- zadanie: komunikácia v nespoľahlivom, rušenom prostredí
- robustnosť, redundantnosť
- scenár: vojnový konflikt



SSH (secure shell)
SMTP (simple mail transfer protocol)

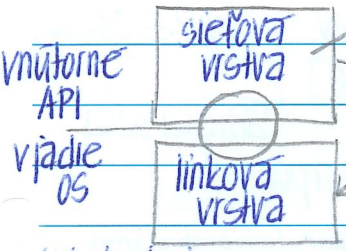
„Normy" internetu: RFC (Request For Comments)

- normy sú vydávané a schvaľované IETF (Internet Engineering Task Force)
- RFC dokumenty – písané jednoznačným, zrozumiteľným jazykom
- každý ho môže navrhnúť

RFC 3501

IP PROTOKOL, ADRESÁCIA, SMEROVANIE PAKETOV

TCP/IP vrstvy



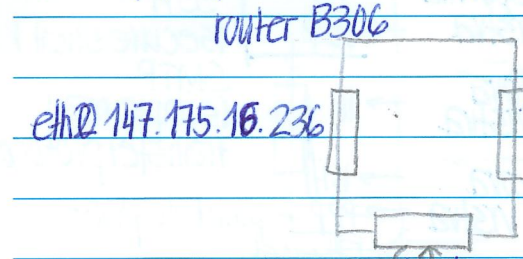
IP protokol

požadovaná služba: doručenie paketu po fyzickom médiu napr. ethernet kábel (sieť), frekvenčný rozsah rádiových vĺn, sériová linka, telefónny kábel

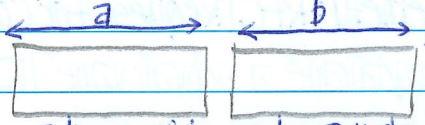
- zariadenie je určené IP adresou (v rámci jedného média)

IP Adresácia (V4)

IP adresa – 4 Byty (oktety) $4 \times 8 = 32$ bitov
– zápis 0–255.—.—.—.
– úlohou je, že jednoznačne určuje rozhranie uzla v rámci internetu



rozhranie Lo 0 (loop) 127.0.0.1 – má ho každý počítač aj keď nie je pripojený k internetu (všade rovnaké)



IP adresa má 2 časti: $a + b = 32$

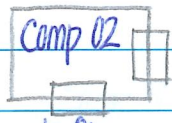
– IP adresy sa pridelujú po sieťach
– IP adresy sú zadarmo

Triedy adries:	prefix	adresa siete	veľkosť siete	Rozsah
trieda A	0...	a	2^{24} uzlov	$0 \leq a < 128$
trieda B	10...	a.b	2^{16} uzlov	$128 \leq a < 192$
trieda C	110...	a.b.c	2^{8} uzlov	$192 \leq a \leq 255$

(23)

Sieť sa delí na **podsiete**: napr. 147.175. 96 . [ ]
adresa siete | adresa podsiete | adresa uzla

Z hľadiska uzla sa IP adresy delia na 2 disjunktné množiny: 1. **dosiahnuteľné priamo** (na tom istom fyzickom médiu ako niektoré z rozhraní uzla) 2. **nedosiahnuteľné priamo**



Ako špecifikovať, aké adresy sú dosiahnuteľné tu?

Každé rozhranie má: /sbin/ifconfig

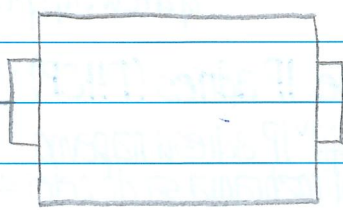
- IP adresu
- netmask „adresa" tvaru

jednotky	nuly
1... 1	0....0

- majme IP adresu a := 192.168.206.3
- majme netmask nm := 255.255.255.0

Je adresa X na tomto rozhraní? Áno, ak $X \wedge nm = a \wedge nm$ (bitwise)

router B306



Vonkajšie učebne:
147.175.16.236
255.255.248.0
(netmask)
sú tu adresy 147.175.16.0 – 147.175.23.255

Vnútro učebne:
192.168.206.1
255.255.255.0
(netmask)
sú tu adresy:
192.168.206.0 – 192.168.206.255

Ak vznikne potreba komunikovať:
- ak je adresa dosiahnuteľná priamo, nie je problém
- inak sa paket prepošle na gateway (gatewayov môže byť viac)
- pre uzly v učebni je gateway 192.168.206.1

Gateway pre B206 je 147.175.16.1 → všetky adresy nedosiahnuteľné priamo, idú na gateway.

Privátne adresy

- pre privátne účely organizácií sú vyhradené privátne siete
- Class B sieť 192.168.0.0 musí zabezpečiť, aby pakety s týmito adresami neprenikli na „vonkajší internet"
- automaticky sa zahadzujú na prechode von
- spôsob fungovania bežného pripojenia domácnosti:



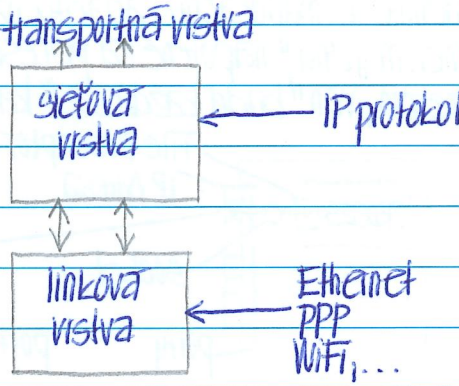
Dynamické prideľovanie IP adries (DHCP) — Dynamic Host Configuration Protokol

- rozhraniu je možné pripojiť IP adresu napevno
- dynamicky: po pripojení rozhrania sa do celej siete vyšle požiadavka na konfiguračný paket → toto prebieha na linkovej úrovni
- príde 1 paket od DHCP servera (obvykle router)

(paket obsahuje informácie o adrese, netmask, gateway, DNS server)

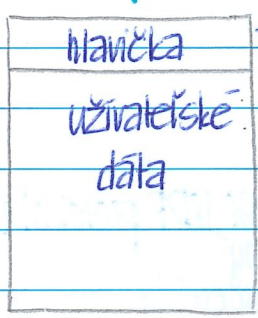
IP PROTOKOL

Úloha: Doručiť dáta pre nadradenú (transportnú) vrstvu medzi dvomi rozhraniami na internete ≅ IP adresami



Použitie IP protokolu spravidla vyžaduje práva superužívateľa (priamo).

Obsah IP paketu:

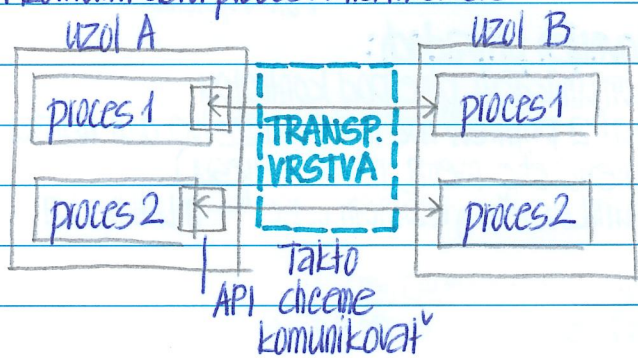


- verzia IP
- celková dĺžka hlavičky
- celková dĺžka paketu
- zdrojová adresa
- cieľová adresa
- TTL
- protokol transportnej vrstvy
- kontrolný súčet

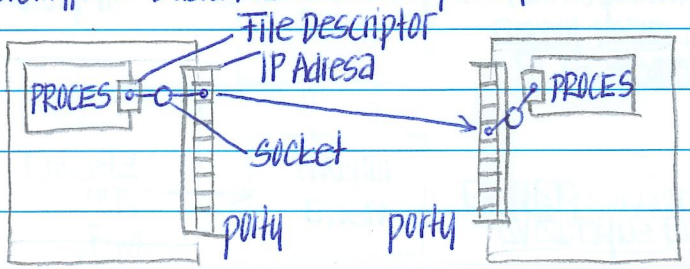
TTL (time to live)
- číslo, od kt. každý router odpočíta 1 ak TTL = 0 paket sa zahodí

TRANSPORTNÁ VRSTVA

- rieši komunikáciu procesov na internete



Rozdelíme komunikačnú kapacitu na menšie časti, ktoré budú:
1. smerom „dnu" napojené na procesy
2. smerom „von" budú každá časť kapacity adresovateľná

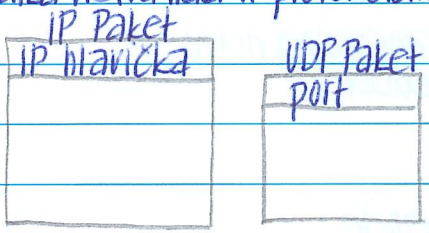


V UNIXE sa dáta posielajú cez write

Porty – čísla 1–65 535
Komunikácia cez porty 1–1024 je **vyhradená pre superužívateľa**
– rôzne aplikačné protokoly bežia na rôznych portoch napr.
80 – HTTP, 21 – SMTP, 22 – SSH /etc/services
(web) (pošta) (secure shell)

UDP PROTOKOL

– tenká vrstva nad IP protokolom

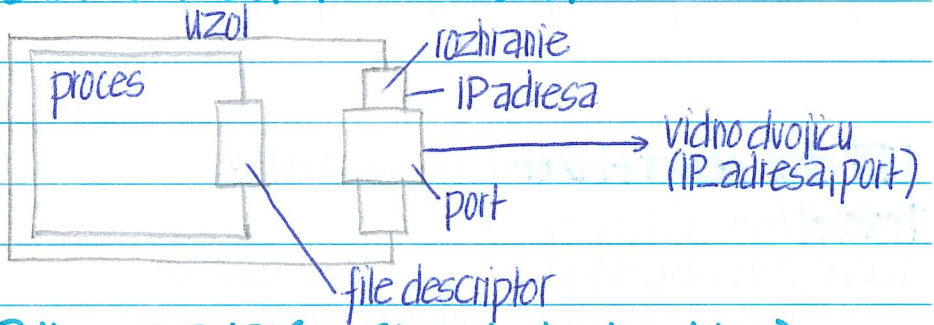


Vlastnosti UDP protokolu
– nespájaný
Nie je zaručené, že:
– druhá strana dáta dostane
– ich dostane iba raz
– ich dostane v tom poradí, ako boli vyslané

Hodí sa v nasledovných prípadoch:
– ak chceme mať komunikáciu plne pod kontrolou
– ak chceme vysielať a prijímať iba malé množstvo dát naraz
DNS (prevod doménového mena na IP adresu)
– potrebujeme komunikovať v „reálnom čase" – multimédia

TCP – nesymetrický klient ↔ server
UDP – server sa od klienta líši tým, že udáva číslo portu

CHAT: SERVER + KLIENT (UDP)



Pythonovské API (v iných jazykoch je to podobné)

1) **Vytvorenie socketu** `s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)` – konštanty == 2
– musíme importnúť socket
dir(socket) – všetky typy socketov: bluetooth, ...

2) **Nadviazanie** (nepovinné) `s.bind((__, port))`
– pred číslom portu vo dvojici je reťazec → buď IP adresa prislúchajúca k nejakému rozhraniu, alebo prázdny reťazec (∀ rozhrania)

3) **Prijímanie UDP paketu** `s.recvfrom(__)`
`s.recvfrom(n)` – maximálna akceptovateľná veľkosť paketu (tak je väčší, tak sa jednoducho zahodí)
– čaká, kým nepríde paket a vráti dvojicu:
– dáta
– adresa (IP_adresa, port) – tiež tam už musí byť neprázdny reťazec, lebo ide o konkr. adresu

4) **Vysielanie UDP paketu** `s.sendto(dáta, adresa)`
– toto vysielanie je slepé (nevieme, čo sa nám s tým stalo)
adresa = (IP_adresa, port)

5) `s.close()` (uvoľnenie portu, zavretie socketu)

- pre non root procesy je bind povolený na porty < 1024

UDP chat server + klient (port servra 9999)

Protokol (pravidlá komunikácie):
každý paket vyzerá takto: _____ : _____
príkaz (neobsahuje :) argument (znaky, UTF-8)

4 príkazy: HELLO (nastav mi nick), ISAY (hovorím ja), SAYS (hovorí niekto), ERROR (chyba)

Príkaz	Smer	Popis
HELLO: nick (tzv. prihlásenie)	C→S	- povinne prvý príkaz (prva správa v klienta) - nick - retazec neobsahujúci ":" "—"
ISAY: správa	C→S	- server pošle SAYS: nick: správa všetkým klientom (ktorých pozná)
SAYS: nick_správa	S→C	- klient vypíše správu na obrazovku
ERROR: číslo_popis	S→C	- možné chyby, kt. môžu nastať

import re → regulerne vyrazy

program netcat (nc)
- vytvorenie socketu na príkazovom riadku (napr. nc www.math.sk 80)

HELLO by malo vytvoriť priradenie (IP_adresa, port) → nick
(ideálnou štruktúrou je slovník)
- klient bude bežať na príkazovom riadku

- na UDP protokole bezi DNS server, peer to peer siete
- na TCP protokole bezia napr. youtube videa a iné multimédiá

TCP PROTOKOL

- spoľahlivé spojenia (viď prednáška 1)
- spájaná komunikácia
- prúdový spôsob prenosu
- nad IP

- zabezpečí nadviazanie a rušenie spojenia
- zabezpečí opakovanie dát v prípade straty
- zabezpečenie prispôsobenia rýchlosti výmeny dát, rýchlosti spojenia

→ skryté pred aplikačnou vrstvou

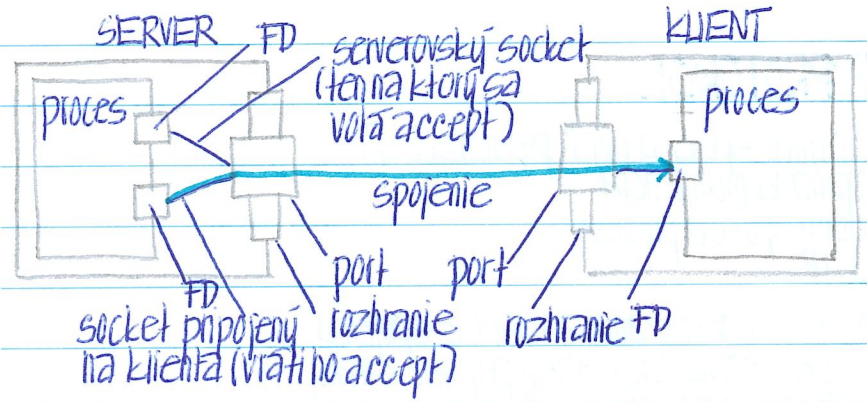
API - pre TCP protokol

A) SERVER

1. s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   s.setsockopt(socket.SOL_SOCKET, socket.SOCK_REUSEADDR, 1
   → kvôli možnosti sa ihneď znovu nadviazať na (IP_adresa, port)
2. s.bind((IP_adresa, port))
3. s.listen(n) - koľko žiadostí o spojenie môže čakať vo fronte v 1 chvíli
4. s.accept() - blokuje, kým sa nepripojí klient, vráti dvojicu (connected_socket, addr) → (IP_adresa, port)
   connected socket - nový socket pripojený na klienta (pripojené sockety) connected_socket.send/recv

B) KLIENT

1. rovnaký postup ako server, ale netreba setsockopt
2. s.connect((IP_adresa, port))
3. posielanie: s.send(reťazec) - pošle dáta
   s.recv(maximálna dĺžka) - prijme dáta



Funguje to na forku: zabezpečí komunikáciu s viacerými klientmi naraz.

TCP SERVER

- máme server so socketom v stave „accept"
- je blokovaný až kým klient nenadviaže spojenie cez connect
- v tejto chvíli treba naraz robiť 2 veci:
  - ísť znovu na accept, aby sme obslúžili ďalšieho klienta
  - vybaviť požiadavku klienta

Riešenia:

1. SELECT

- umožňuje zisťovať (sledovať), či sa niečo nedeje a to na viacerých filedescriptoroch naraz
- umožňuje zistiť, či je možné čítať/zapisovať na nejaký filedescriptor == socket
  - select(...)
    zistím, či klient niečo poslal
    zistím, či sa chce pripojiť nový klient
    vybavím
- používa sa niekedy, ak klienti pristupujú k zdieľanému zdroju, ktorý môžu meniť napr. databázové servery, X server

2. FORK

```
while true
    if !fork: child proces
        s.close()
        komunikuje s klientom cez cs
        cs.close()
        sys.exit()
    else: parent proces
        cs.close()
```

- budeme mať textovo orientovaný protokol
- po pripojení: S→C sumátor 1.0 → server znuluje počítadlo
- riadky:

C→S	CISLO_0\n 0→ prirodzené číslo
S→C	100_OK\n (ak je všetko v poriadku) 200_BAD_NUMBER\n (ak nieje dobré číslo)
C→S	SUMA\n
S→C	100_OK\n hodnota počítadla\n + znuluje počítadlo
C→S	hocičo iné
S→C	201_BAD_COMMAND\n

Poznámka: pri vracaní statusu je dôležité iba číslo, nie popisujúci reťazec za ním

KLIENT TCP

- príkaz má tvar: sčítaj server číslo1, číslo2, číslo3, ...

Úloha: - pošle na server príslušné príkazy
- vypíše výsledok

Domáce zadanie → na stránke predmetu

DNS SERVER (Domain Name System/Service)

- principiálny pôvodný účel - mapovanie symbolických mien na IP adresy, neskôr mal aj ďalšie využitie
- princíp implementácie: distribuovaná hierarchická databáza

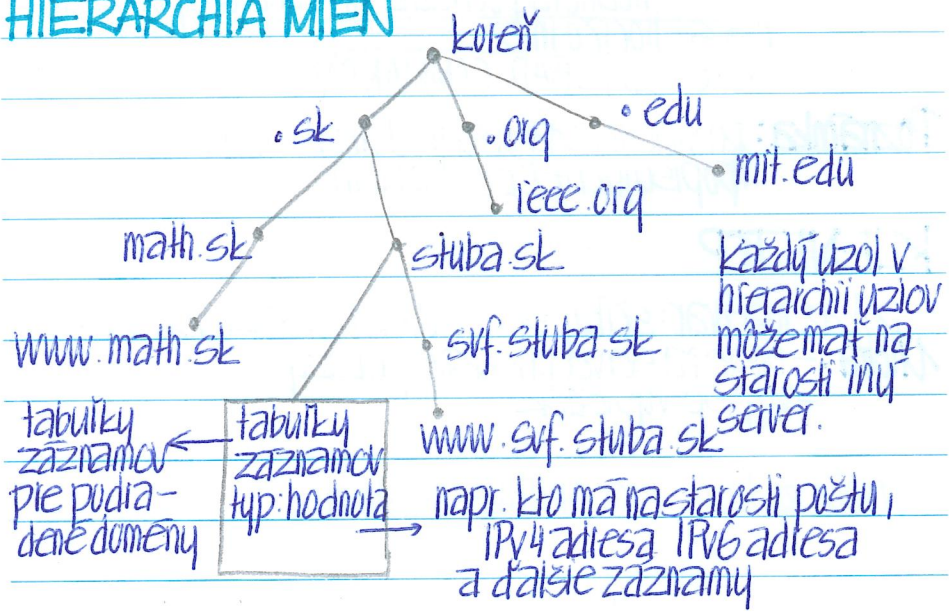
www.stuba.sk $\xrightarrow{?}$ IP adresa

Pôvodne bol na každom uzle jeden textový súbor, kt. sa volal HOSTS. Dnes /etc/hosts → môžeme priradovať mená nejakým IP adresám → riešenie zjavne nebolo dobre škálovateľné.

Ako to funguje:

- každý uzol na internete, ktorý chce zistiť aká IP adresa patrí danému uzlu menu sa to musí spýtať DNS servra. Klientský kód je súčasťou C knižnice.

HIERARCHIA MIEN



Každý počítač môže zisťovať DNS záznamy sám, alebo to delegovať na iný stroj (to je obvyklé)
DOMA: váš počítač → DNS server vášho providera
→ cacheuje záznamy

Typy záznamov: „y.z – tu sú uložené záznamy pre x.y.z

A	IP_adresa
NS	IP_adresa DNS serva
MX	IP_adresa poštového klienta

APLIKAČNÉ PROTOKOLY

- **DNS** (nad UDP) (spôsob fungovania TCP protokolu)
- **SMTP** – prenos pošty (uloženie na serveri)
  - nad TCP
  - textový protokol

HTTP PROTOKOL

Európ. ústav pre jadrový výskum

- vymyslený v 80. rokoch v Cerne (Ženeva), autor T. Berners-Lee
- aplikačný protokol siete WWW
- je vymyslený dobre
- súčasná verzia: HTTP 1.1

Nové vlastnosti oproti HTTP 1.0:

1. viac požiadaviek na 1 spojenie
2. virtual hosts – umožňuje viac webom zdieľať 1 IP adresu (odlišujú sa len doménovým menom)

URI – uniform resource identifier (syntax identifikácie zdroja)

scheme : hierarchical_part ? query # fragment

„TYP": určuje ako vyzerajú tieto

napr. mailto:username@example.com?subject%Hello

URL – podtrieda URI

http://host/path?...%....

nepovin. nepovinná časť

file:///home/user ftp://...

HTTP URL

http://username:password@hostname:port/?query%fragment

nemusí byť (username:password@)

absolútna cesta (/)

port: číslo portu (nemusí byť / default 80)

Niekedy, ak je známy kontext (= host + ..., host + ... + prefix cesty) sa uvádza: – v prvom prípade iba absolútna cesta
– v druhom prípade relatívna cesta
– absolútna cesta začína „/" relatívna nie

TVAR SPRÁV HTTP:

C → S (požiadavka)

metóda␣URL␣verzia protokolu

[hlavičky – bližšia špecifikácia požiadavky

– prázdny riadok

[obsah požiadavky (nemusí byť)

S → C (odpoveď)

verzia_prot.␣status␣popis st. (číslo)

[hlavičky – bližšia špecifikácia vlastností odpovede

– prázdny riadok

[obsah odpovede (nemusí byť)

CLIENT → SERVER podrobne:

- **metódy:** GET, PUT, POST, DELETE, HEAD, OPTIONS
  najčastejšie používané zdroj = je určený URL

GET „daj mi obsah zdroja"
- 2 rovnaké GET by mali dať to isté → server by nemal meniť zdroj na základe GET metódy → **idempotentné metódy**

PUT „zmeň obsah zdroja"
- 2 rovnaké PUT by mali dať to isté

POST „pripoj k zdroju" (zmení zdroj)
- nie idempotentná metóda

DELETE „zmaž zdroj"

HEAD – funguje rovnako ako GET, ale vráti iba headre, nie obsah

OPTIONS

HEADRE

Každý header má nasledovný tvar: **názov_headra : obsah**
(v názve headra nezáleží na veľkosti písma)
- headre končia prázdnym riadkom

Client → Server:

Accept: typy dokumentov, ktoré vieme spracovať
Accept-Language: preferované jazyky
Host: DNS-meno : port (nemusí byť) → jediný povinný header
- toto vypĺňa užívateľ v príkazovom riadku
- môžme mať viac virtuálnych webov na 1 IP adrese
- webserver ich rozlíši pomocou Host: headra

Client → Server:

Accept: MIME_type(s), oddelené čiarkou

MIME – Multipurpose Internet Mail Extensions
- súčasť sústeme RFC zavádzajúceho to, čo poznáme ako „prílohy pošty"
- **MIME type**: x/y (typ / subtyp) x ∈ {image, text, audio, video, application, ...}

MIME types sú centrálne pridelované organizáciou IANA. (Internet Assign. Numbers Authority)
Neutrálny MIME typ application/octet_stream
MIME type je možné „bližšie špecifikovať"
Pr. text/html; charset=utf-8

Accept: _____ – takéto typy som schopný spracovať
v praxi: signalizácia preferencie HTML/XHTML/TXT (slepecké čítačky)

Accept-charset: _____ – aká znaková sada môže byť
pr. utf-8, ISO-8859-2, CP-1250

Content-length: _____ – dĺžka obsahu (zasielaného klientom) v Bytoch

Content-type: MIME_type – typ zasielaného obsahu

Date: čas posielania requestu

Host: _____ : _____ – identifikácia požadovaného virtual host
doménové meno : port (nepovinné)

If-*: _____ – „podmienená požiadavka"
špecifikácia kvôli šetreniu zdrojov

napr. If-Modified-Since: _____ (dátum, čas)
– server pošle normálnu odpoveď (200 OK + obsah) ak bol zdroj zmenený od daného času, ak nie, pošle stavu 304 Not Modified, bez obsahu

napr. If-None-Match: ETAG - dlhé hexadecimálne číslo
v praxi: hash obsahu

ETAG - klient sa ho dozvie z Etag headra (Etag: ...) posielaného servrom pri predošlej požiadavke
- v praxi ETAG posielajú „reverzné proxy"
- ak sa obsah zmení oproti hodnote definovanej ETAG, server ho pošle, inak 304 Not Modified

Referer: URL - URL, ktoré linkovalo na požadovanú stránku (napr. reklamy)

User-Agent: identifikácia browsera (klienta)

Server → Client:

Content-Type: MIME type obsahu
Content-Length: dĺžka obsahu
Date:
Etag: ... identifikácia obsahu (pre If-None-Match)
Location: URL - pri presmerovaní
Server: ... identifikácia servra

STATUS KÓDY V ODPOVEDI: trojciferné čísla

1xx informačné
2xx úspech
3xx presmerovanie
4xx chyba klienta
5xx chyba servra

100 Continue 101 Switching protocols

200 OK, 201 Created (odpoveď na PUT)
301 Moved permanently } teoreticky rôzne, v praxi synonymá
302 Found } znamená: toto nájdeš na URL, ktoré je v Location: ... header

304 Not Modified - odpoveď na If- header (refresh)

400 Bad request
404 Not Found
403 Forbidden

Klient neposlal požiadavku včase, keď ju server očakával
408 Request Timeout
418 I'm a tea pot
401 Not Authorised

500 Internal server error
502 Bad gateway (padnú reverzné proxy)

HTTP: Server → Client

1. Content-Length: ___ (Bytes)
   ⋮
   ___ — posielame celý obsah naraz
   obsah
2. Transfer-encoding: chunked\r\n → využíva sa najmä pri stránkach, ktoré vznikajú dynamicky
   ⋮
   hexadecimálny počet_bytov\r\n
   ↓ obsah (množstvo dát je daný hexadecimálnym počtom_bytov)
   znovu

∅ — koniec dát (pošleme „chunk" veľkosti nula)

HTTP AUTENTIZÁCIA

Server: pošle status 401 Not authorized
pošle v headroch: www-Authenticate: Basic realm="*"
*vzhľadom na akú zdroj sa autentizuje

Klient: vypýta si meno a heslo od užívateľa
vytvorí reťazec meno:heslo
zakóduje ho podľa algoritmu base64
pošle request znovu, pričom uvedie v headroch:
Authorization: Basic zakódované meno_heslo

Server: dekóduje skontroluje, vráti 401 alebo normálny status
POZOR! Basic autentizáciu používať iba ak je šifrované spojenie na úrovni socketu ("https:")

COOKIES:

- server si môže u klienta odložiť niekoľko krátkych dvojíc meno=hodnota
- každá dvojica môže mať uloženú platnosť (dokedy platí)

Mechanizmus: zo servra na klienta sa pošle header **Set-Cookie**:
meno1=hodnota1 (môže ich byť aj viac)
Od tejto chvíle klient posiela v každom requeste header **Cookie**:
meno1=hodnota1; meno2=hodnota2; ...

- v praxi sú hodnoty cookies nezrozumiteľné kódy, kt. referencujú na nejaký položku v databáze
- **Set-Cookie: meno=hodnota; Expiration=_____**
  (dátum a čas - podrobne napísané v RFC)
- priamo v headri možno explicitne napísať platnosť
- ak nie je uvedená, cookie sa posiela až kým sa neukončí beh browsera

Cookies sa používajú:
1. autentizácia
2. udržiavanie informácii počas sedenia napr. nákupný košík

```
import Cookie
c = Cookie.SimpleCookie()
(Morsel object)
c["názov_cookie"].value = hodnota
c.output() → Set-Cookie: '___'
```

HTML

- **SGML** - všeobecný jazyk formátovania dokumentov
  - ISO norma (80. roky)
  - má 2 časti: syntax dokumentu (generická)
    spôsob špecifikácie typu dokumentu
    (DTD - document type definition)
- **SGML dokument má rôzne časti** - obsah
  - tagy <meno_tagu> </meno_tagu>
  - entity &entita
  - poznámky
- DTD hovorí, aké tagy sú povolené, v akom poradí atď, ...
- je možné skontrolovať programom, či SGML dokument spĺňa daný DTD.

História:

- začiatok 90. rokov Timothy-Berners-Lee fyzik pracujúci v CERNE
- pôvodne HTML - neformálna analógia SGML (nebol DTD)
- účel: prezeranie a editácia dokumentácie v CERNe, dokumenty boli **hypertextové** (obsahujúce linky)
- vymysleli **HTML+HTTP**
- vytvorili 1 browser/editor a pre počítače NeXT cube
- NeXTy boli drahé
- prvá implementácia browsera v X Windows: NCSA Mosaic
- prvý browser nekontroloval dôsledne správnosť syntaxe dokumentu, lebo sa nerátalo s tým, že niekto bude vytvárať HTML ručne
- pri chybných dokumentoch sa zachoval „náhodne"
  xabcd
- toto náhodné správanie bolo nutné reimplementovať v iných browseroch

- browser na chybný dokument nezahlási chybu, ale urobí „niečo" dodnes
- prvá verzia špecifikácie HTML 2.0 (ako DTD SGML)
- vo chvíli keď vyšla, už bola zastarala

1994-2000 - browserwars

- na trhu sú 2 browseri: Internet Explorer
  Netscape Navigator
- konkurujú si nepremysleným zavádzaním nových tagov + nedôslednou implementáciou tagov konkurencie
- pôvodne HTML - špecifikácia obsahu („sémantické tagy")
- počas browserwars: **prezentačné tagy** <font>, <blink>, ...
- 1997 vzniká konzorcium W3C - normalizačný orgán
- pokus zaviesť poriadok zlyhával (HTML 3.2, 1997)

1998 HTML 4.0 - oddelenie obsahu a prezentácie
(obsah - HTML, prezentácia - CSS)

- presadilo sa, lebo CSS je výhodné pre tvorcov obsahu
- postupný ústup prezentačných tagov
- súbežne: XML - ideový nasledovník SGML
  - jednoduchší, striktnejšia syntax
  - všeobecný spôsob prezentácie dát

XHTML 1.0 - HTML, ktoré je typom XML dokumentu
- umožňuje lepšie automatizované spracovanie HTML

HTML 4.0 - ako-tak implementované v Internet Explorer 6.0
- Netscape - niekým kúpený/krachuje
- vývoj je dlhodobo ustrnutý na Internet Explorer 6.
- Netscape uvoľňuje kód Navigátora pod OPEN Source licenciou → Firefox

Firefox - vyvíjaný nezávislou nadáciou

- momentálne XHTML 1.0 + CSS2

XHTML dokument

```
<!DOCTYPE ...>
<html xmlns="..." xml:lang="sk" lang="sk">
<head>
  hlavička
</head>
<body>
  obsah
</body>
</html>
```

Z čoho sa skladá XML dokument:

1. `<!DOCTYPE ...>` deklarácia typu (nemusí byť)
2. **Obsah dokumentu**

`<tag atribút1="hodnota1" atribút2="hodnota2">`
obsah tagu – môže obsahovať ďalšie vnorené elementy alebo text
`</tag>` – koniec elementu

- okrem elementov a textu obsahuje dokument tzv. entity napr. `©` ©, `&qti;` >

Elementy bez obsahu:

`<tag atribúty></tag>` sa v XML píše skrátene ako `<tag atribúty />` – povinná medzera

XHTML – elementy v obsahu `<head>`

- `<title>text</title>` – nadpis dokumentu – typicky v rámci okna (tabu)
- `<meta http-eqiv="content type" content="text/html; charset="UTF-8">`

↓ veľmi dôležité; chová sa, ako keby server poslal takýto mime typ v HTTP headroch

`<link rel="..." />` – externé zdroje týkajúce sa dokumentu (stylesheet, licencia, ...)

<body>
vlastný dokument
</body>

- **normálny text** - viacero bielych znakov (medzera, tab, newline) = 1 biely znak
  - ak chceme z nejakých dôvodov použiť medzeru ako prázdny znak je to entita   (non-breakable space)
  - je určené na zabránenie zalomenia napr. v dome

Odstavce:
Tvrdé zalomenie riadku:
 ==
</br> (XML)
- prečo je tu medzera?
  Pôvodne boli povolené nepárové tagy napr.
 (v SGML)
- v XML sa to nesmie (tag musíme ukončiť)

- riešením by bolo zaviesť
 (bez medzery)
- táto vec ale nebola staršími browsermi chápaná ako koniec riadku, ale <br /> ako koniec riadku vykresľovali
- **medzera je v XML norme**

 sa môže využívať len **vo veľmi špeciálnych prípadoch** keď je ukončenie riadku súčasťou textu (veršu)
!nie ako náhrada za iné prvky formátovania

Zdôraznenie textu:
... - „emhasize" (default: šikmo)
... - „strongly emphasize" (default: hrubo)
- zastarané tagy na to isté (italic)
   (bold)
- nadpisy <h1>...</h1> - najvyššia úroveň (najväčší)
  <h6>...</h6> - najnižšia úroveň

Zoznamy:

 - ordered list (default: číslovaný zoznam)
  - jediné povolené elementy tu sú <li> ... "list item"

- vnútri v <li> sa môže používať už čokoľvek

 - unordered list (odrážky → default: body)
  :

každý zoznam čohokoľvek by mal byť <ul> alebo

Definičné zoznamy:

<dl> - description list
  - povolené elementy <dt> ... </dt> - čo popisujem
                      <dd> ... </dd> - popis
</dl>

Linky

 text linku default: modrý text a podčiarknuté
          URL

a) **plné URL** http://...
b) **plná cesta** (začínajúce /) na tom istom servri ako je tento dokument. URL dokumentu: http://x.y.z/p1/p2/p3.html
href="/q1/q2.html" referencuje na http://x.y.z/q1/q2.html
c) **relatívna cesta** URL dokumentu http://x.y.z/p1/p2.html
href="p10.html" referencuje na http://x.y.z/p1/p10.html

Ak používate ©, štruktúra odkazov je presunuteľná pod iný URL prefix (na iný server do iného adresára)

Obrázky:

 minimálna verzia

 - v pixeloch

- zjednodušenie vykresľovania (stránka sa pri načítavaní neposúva)

alt=" " - popis obrázka pre slepcov a vyhľadávacie stroje

title=" " - generický atribút → popisok, ktorý sa zobrazuje po nabehnutí myšou

Tabuľky:

<table>

starý spôsob - jediný povolený prvok bol <tr>...</tr> (table row)

vo vnútri: <th> hlavičková bunka </th>

<td> normálna bunka </td>

nový spôsob

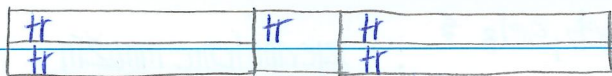
<thead></thead> – vo vnútri tagov sa nachádzajú tablerow elementy

<tbody></tbody>

<tfoot></tfoot>

</table>

<tr rowspan="_" colspan="_" />



rowspan="2"

CSS (Cascade Style Sheets)

- vizuálna prezentácia HTML - ako majú veci vyzerať

3 spôsoby aplikácie:

1) v atribúte style (veľmi neefektívny spôsob)
2) v tagu <head></head> dokumentu - tag <style></style>
3) v externom CSS dokumente <link rel="stylesheet" href="URL" /> (v tagu head)

miesto, kde je uložený súbor .css

CSS sa skladá z pravidiel:

PRAVIDLO: selektor { vlastnosť1 = hodnota1 — popis toho, ako vlastnosť2 = hodnota2 zmeniť daný element }

výber množiny element.

- v CSS môžu 2 selektory vybrať rovnaký prvok (selektory nemusia vyberať disjunktné množiny) - v prípade konfliktu má prednosť špecifickejší selektor (existuje číslo „špecifickosť")

TYPY SELEKTOROV

* — všetko

tag (p, li, ...) — typový (tag) selektor — vyberie všetko s daným tagom a vlastnosti aplikuje na to

.názov_triedy — podľa class atribútu <názov_tagu class="názov_triedy" .../>

tag.názov_triedy — daný typ (tag) s danou triedou (class)

#hodnota_id — podľa id-atribútu (id má byť unikátne vrámci dokumentu)

Grouping: sel1, sel2, sel3 { } — zjednotenie množín

Nesting: sel1 sel2 { } — vyberie prvky sediace so sel1 a vnútri vyberie prvky sediace so sel2 (nekomutatívne)

sel1 > sel2 — vyberie sel1 a tie z bezprostredných potomkov, ktoré spĺňajú sel2

sel1 + sel2 — vyberie sel1 — tie z bezprostredne nasledujúcich (na tej istej úrovni), ktoré sedia so sel2

PSEUDOTRIEDY

sel:hover – ak je nad elementom kurzor myši
sel:first-letter – prvý znak textu podobne sel:first-line
a:link – nenavštívený link
a:visited – už navštívený link
a:active – link na túto stránku

sel:before { content:"..." } HTML sel:after – podobne

GENERICKÉ ELEMENTY

 – na celú šírku dostupného priestoru ako h1
 – v riadku (ako em) display:block
default display:inline

 sa používa na rozdelenie stránky na logické časti napr.

```
<body>
<div id="hlavička">
    <h1>...</h1>
</div>
<div id="navigácia">
</div>
</body>
```

– neskôr nám umožňuje odkazovať sa na tieto logické časti pomocou id, za predpokladu, že tento atribút je nastavený
– v samotnom členení textu nehrá žiadnu úlohu (z vizuálneho hľadiska)

Hodnoty atribútov:

1. dĺžky: 1 cm 1 px 1 pt – sú závislé od média (pt – typografická miera 1 bod) 1 em – šírka veľkého M alebo v % k normálnej veľkosti

2. farby: – názov (Red, Blue, Black, atď. – len predefinované)
– rgb (x1%, x2%, x3%) percentuálne zastúpenie farieb (červená, zelená a modrá)
– rgb (0–255, 0–255, 0–255)
– # hexadecimálne dvojčíslice 00–FF #⌴⌴⌴
– # hexadecimálne jednočíslice 0–F #⌴⌴⌴

Vlastnosti textu/fontu:

font-family: "—", "—", "—", "serif/sans-serif"
– ten poslednú je vždy k dispozícii
– ostatné určujú, akým fontom sa má text zobraziť s klesajúcou prioritou

font-size: – dĺžka;
font-weight: < normal; bold;
font-style: < normal; italics;

text-decoration: < overline; XX
line-through; XX
underline; XX
none; XX

tex-align: < left; right; center; justify; – prispôsobí množstvo medzier medzi slovami

line-height: – dĺžka; (normálne 120% textu)

Farby:

color: farba;
background-color: farba;

Štylizácia zoznamov: (pivky <li> a)

list-style-type: < circle ○; disk ●; decimal 1. 2. 3.; alpha (lower-alpha) a. b. c.; roman (upper-roman) I II III

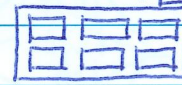
list-style-position: < outside
inside

list-style-image: url("...") - URL s obrázkom odrážky

Tabuľky:

border-collapse: < collapse (pozlučujú sa hranice)
separate (by default)





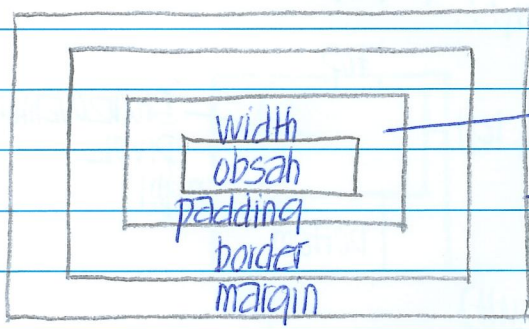
Obrázky na pozadí

background-image: url(____);
background-repeat: - repeat-x; (opakovanie na šírku)
- repeat-y; (opakovanie na výšku)
- repeat (default); (opakovanie x aj y)
- no-repeat;

background-attachment: < scroll (defaut)
fixed (obsah sa skroluje ponad pozadie)

- pozadie môžeme nastaviť ľubovoľnému elementu

ŠKATUĽOVÝ MODEL (BOX MODEL)

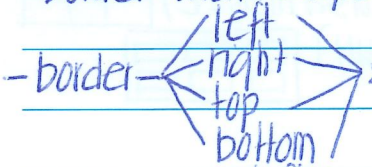


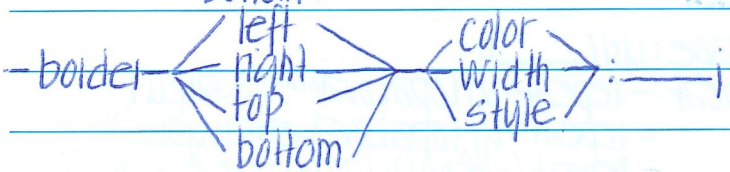
- margin má farbu pozadia „okolia"
- padding má farbu pozadia prvku
- margin môžeme zafarbiť ak ho vložíme do div

- margin: dĺžka, padding: dĺžka

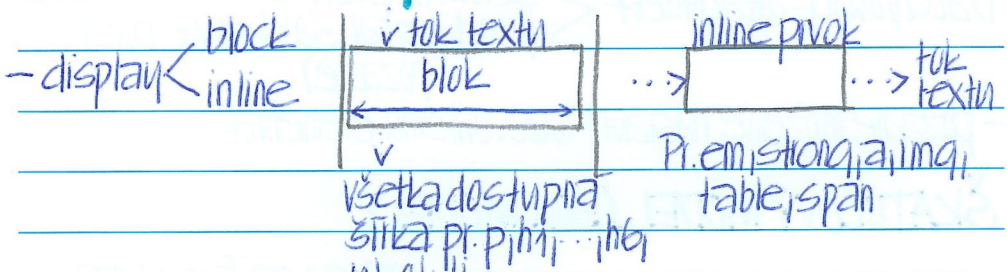
margin:
padding:
right
left
top
bottom
dĺžka

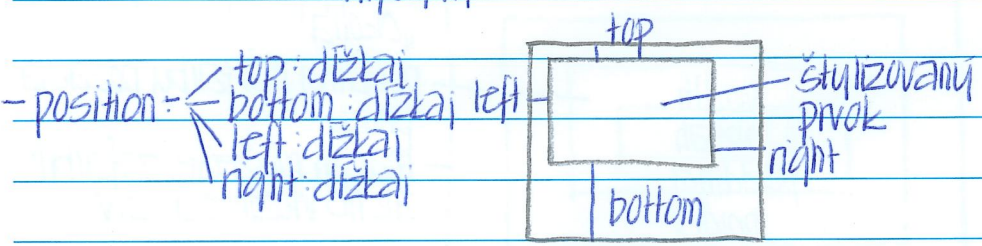
- border-color: farba;
- border-width: dĺžka;
- border-style: solid, dotted, dashed, inset, outset, none, ...
- border: width style color; napr. border: 2px solid red;





Ovládanie rozloženia prvkov na stránke

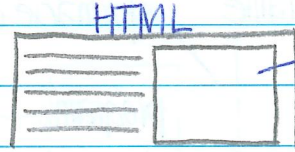




- position
  - static; (default)
  - relative; (vzhľadom na pozíciu, kde by to stálo normálne)
  - absolute; (vzhľadom na obsah - box)
  - fixed; (vzhľadom na okno browsera napr. reklamy)

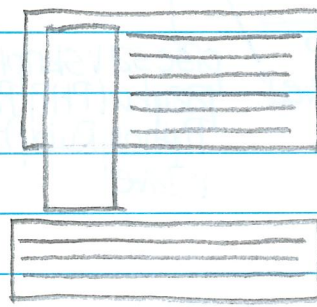
- k Box modelu: width: dĺžka; height: dĺžka; pri inline elementoch ignorovaná

- float: left; right; none;



float: right
text, kt. nasleduje po ňom ho obteká

- clear: left; right; both;



float: left
- ak presahuje cez veľkosť elementu, v ktorom sa nachádza
- nasledujúci element sa posunie až pod

DYNAMICKÝ WEB

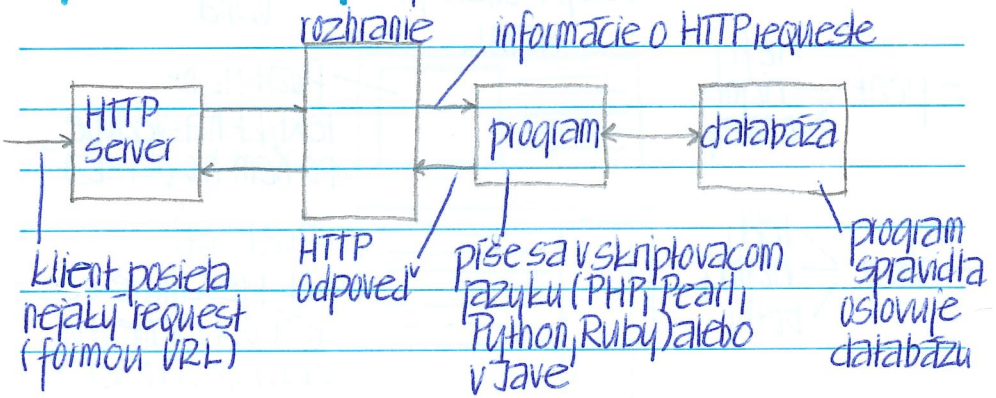
Dynamické webové stránky:
- CGI skripty (namiesto pevného html sa spúšťa program, ktorý html generuje na výstupe)
- pôvodne: spracovanie formulárov, prístup do databáz
- PHP (fractal of Bad Design)
- Framework (Ruby on Rails, Django, CodeIgnition, Apache Struts)

RUBY PYTHON PHP JAVA

- statické stránky = súbor na disku



Dynamické stránky: (v praxi)



V našom prípade:

- HTTP server: zabudovaný v Pythone
- interface (rozhranie): CGI (Common Gateway Interface)
  - → najstaršie
  - → najpomalšie
  - → vždy fungujúce (dostupné)
- program: pythonovský skript
- databáza: súbory na disku (nekorektné - došlo by ku kolíziam, pri súčasnom prepise súboru 2 inštanciami programu)

Ako funguje CGI (pod Unixom):

- máme vyhradený nejaký prefix URL, obvykle /cgi-bin
- ak server dostane request s týmto prefixom, nehľadá dokument/obrázok/mp3/... na disku ale spustí program, ktorého meno je určené tým, čo je za /cgi-bin/... - meno programu

Server to zariadi tak, aby program po spustení
- dostal niektoré vlastnosti HTTP requestu predspracované vo forme premenných prostredia

- obsah HTTP requestu (to, čo je pod headrami) dostane program na stdin
- program spracuje údaje z requestu a zapíše na svoj stdout: 1. headre odpovede (nie status!) 2. obsah odpovede

Server prebehne odpoveď, skontroluje ju, vybaví stavom a pošle klientovi.

HTML FORMULÁRE

```
<form method="__" action="__">
    ľubovoľné elementy (aktívne sú input, select)
</form>
```

method="get"/"post" ("get" je nastavené ako default)
action = URL - kam poslať vyplnené dáta (default: URL, ktoré obsahuje formulár)

Čo sa deje po zaslaní dát, ak je method="get":

Browser pošle HTTP GET request na URL uvedené v action, pričom za to URL pripojí reťazec **?var1=val1&var2=val2&...**
var - názvy vstupov - toto je HTML (HTTP) nemá
val - hodnoty vstupov

Ak je na URL CGI-skript, server mu odovzdá časť URL za otáznikom v premennej QUERY_STRING

Čo sa deje po zaslaní dát, ak je method="post":

Browser pošle HTTP POST request na URL uvedené v action atribúte <form.../> elementu, pričom obsah formulára je obsahom requestu (t.j. je uvedený po HTTP headroch a prázdnom riadku)

Content-type: application/x-www-form-urlencoded
Content-lenght: je spravidla tiež uvedené

Ak je na URL CGI-skript, server mu odovzdá dáta na štandardnom vstupe.

HTML elementy reprezentujúce vstupy vo formulároch:

<input type="__" name="__" value="__" />
(meno prvku) (nepovinné) - spravidla iniciálna hodnota
povolené hodnoty: text, password, radio, checkbox, submit, ...

type="text"
maxlength="__" maximálna možná dĺžka
size="__" šírka v znakoch (viditeľná)

type="password" - rovnako ako text, ale zobrazuje *

type="submit" - odosielací button
- name je nepovinné
- value - popiska na buttone (default: submit)

type="radio" - radio button ⊙ ○
- browser umožňuje vyznačiť ON OFF
- ~~nanajvýš 1 radio button, ktorý má danú hodnotu atribútu name="__"~~
- posiela v obsahu name=value
- predvolene zapnutý checked="checked"

type="checkbox"
- ako radio button, ale z jedného name je možné vyznačiť viac naraz

Select a Textarea

<textarea name="__" rows="__" cols="__">
počiatočná hodnota - umožňuje viacriadkový text
</textarea>
<select name="__" ...> - môže byť multiple="multiple"
option elementy → <option> text </option>
</select> - môžeme odosielať viac hodnôt naraz

Skrytý vstup: <input type="hidden" name="__" value="__" />
- nezobrazí sa ale browser ho pošle

PYTHONOVSKÉ MODULY PRE PODPORU CGI A FORMULÁROV

Modul cgitb
- slúži na zachytenie výnimky a jej zobrazenie v browseri
- používa sa počas vývoja
- použitie: import cgitb cgitb.enable()

Spracovanie formulárov:

Ak máme <form method="post" /> (žiadny atribút action)
Browser pošle dáta z formulára na tú URL, ktorá obsahuje formulár.
Máme kód, ktorý a) generuje formulár b) spracúva formulár

if metóda je GET:
    vráť nevyplnený formulár (HTML) - <form method="POST
elif metóda je POST: </form>
    spracuj dáta a buď ich ulož, alebo vráť chybu

Modul cgi

cgi.FieldStorage - trieda, ktorá pri vytváraní svojej inštancie (fs = cgi.FieldStorage) vytiahne obsah formulára
- z premennej prostredia QUERY_STRING (pre metódu GET)
- z obsahu requestu (sys.stdin) (pre metódu POST) a uloží obsah do seba

FieldStorage podporuje špeciálnymi metódami testovanie prítomnosti premennej cez: if "meno" in fs a iterovanie cez premenné for var in fs:

Ako získať hodnoty koncepčne správne:

fs.getlist("__") - vráti zoznam hodnôt premennej, kt. prišli s týmto menom (ak žiadna vráti prázdny zoznam)

POZOR! V rôznych tutoriáloch je to zle!

JAVASCRIPT

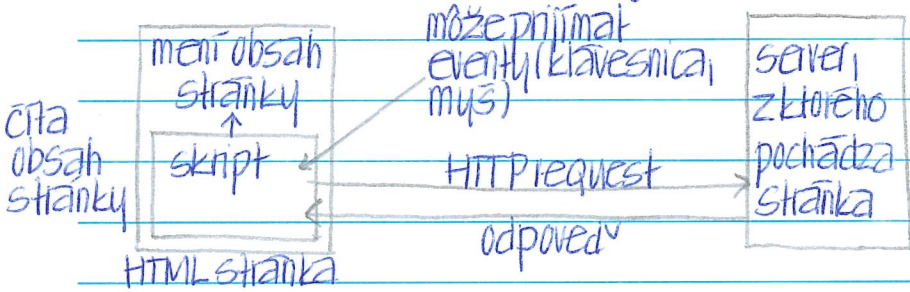
- vyvinutý v 90. rokoch pre Netscape Navigator počas browser wars

Pôvodný účel: jednoduché animácie, kontrola vstupov
Princíp: skript zahrnutý v rámci stránky, mení stránku

- existuje norma (ECMA Script)
- použitie: na serveri (node.js, couchdb)
  na desktope (widgety - Apple, KDE)

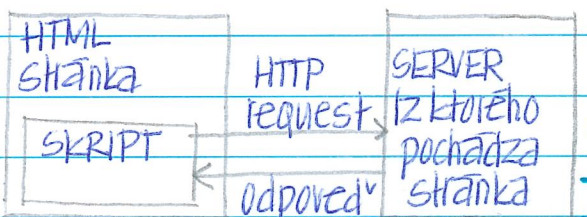
Z hľadiska taxonómie je Javascript jazyk: funkcionálny, objektovo orientovaný, nepozná pojem triedy

Použitie v browseri:



Dnes sa používajú Javascriptové "frameworky".
- bohaté: Dojo, QOOXDOO
- písané v iných jazykoch, prekladané do Javascriptu:
  GWT - Google, Pyjamas - Python - Javascript
- my si budeme ukazovať JQUERY

AJAX: Asynchronous Javascript and XML



Formát odpovede musí byť taký, aby jej skript rozumel
2 možné mimetype:
- **XML**
- **JSON - Javascript Object Notation**

"asynchrónne" - nečaká na ňu, ale skript dostane event (nezasahuje do vzhľadu a správania existujúcej stránky)

V Javascripte sú objekty a slovníky skoro to isté.

```
x = {
    a: 1
    b: [4,5,6]
    c: "ano"
}
```

- JSON syntax
- objekt, môžeme robiť x["a"] (ako slovník) aj x.a (ako objekt)