Príklady:

spojená:
- rozhovor
- telefonický hovor
- poslanie listov

nespojená:
- prednáška
- rádiové vysielanie
- rozoslanie letákov

Spôsob prenosu
- prúdový prenos - po jednotlivých znakoch (USB)
- blokový prenos - po väčších blokoch
  - blok sa prenesie alebo neprenesie ako celok
  - bloky sa nasýpajú rôzne: - paket
    - datagram
    - rámec (frame)
  - každý takýto blok má:
    - metainformáciu (adresa, kontrolný súčet, adresy, ...)
    - vlastný obsah

1) Prepájanie paketov
2) — " — okruhov

EN: - Packet switching
- ~~Packet~~ circuit switching

2) - spojané
- sieť je „múdra", drží v sebe kontext spojenia } drahé, spoľahlivé
- príklad: telefón
  ↳ klasická tel. sieť

1) - nespojané
- sieť nedrží kontext spojenia, iba smeruje pakety } lacné, nespoľahlivé
- koncové zariadenie drží kontext
  ↳ INTERNET

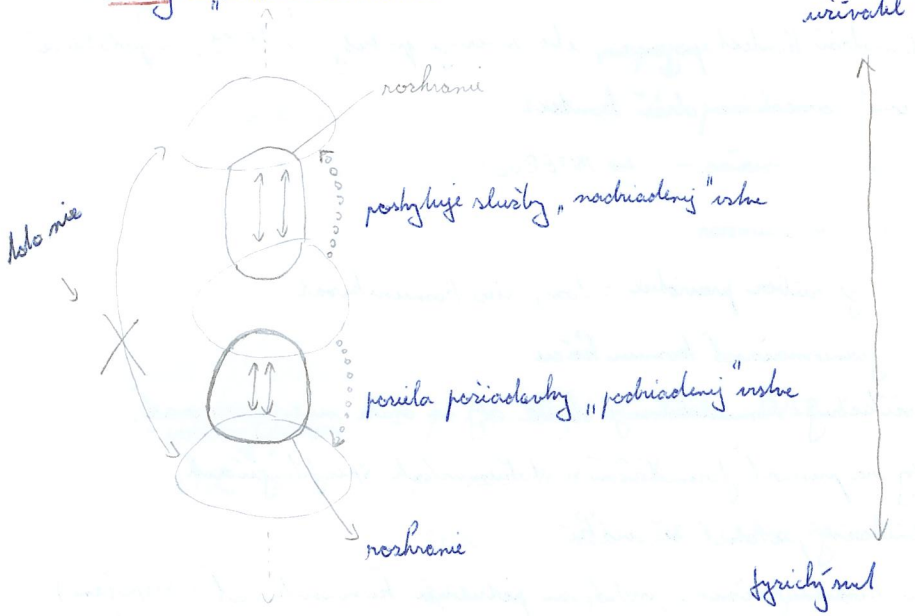
Protokol je súbor pravidiel o tom, ako komunikovať.
- účel: jednoznačnosť komunikácie
  Pri počítačovej komunikácii je treba aby sa dala naprogramovať.
- Nároky na presnosť formulácií v dokumentoch špecifikujúcich komunikačný protokol sú veľké.
  Autor komunikačného protokolu potrebuje komunikovať (nespojane) s ľuďmi, ktorí budú implementovať protokol
- Vzniká potreba komunikačného protokolu o špecifikácii komunikačného protokolu.
- Úloha č. 1 bolo vyjednotenie terminológie

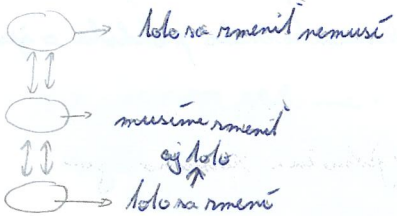
↓

Referenčný model ISO/OSI
- Abstraktný model dátovej siete
- Autori: svetoví telekomunikační giganti (197x ≤ roky ≤ 198x)

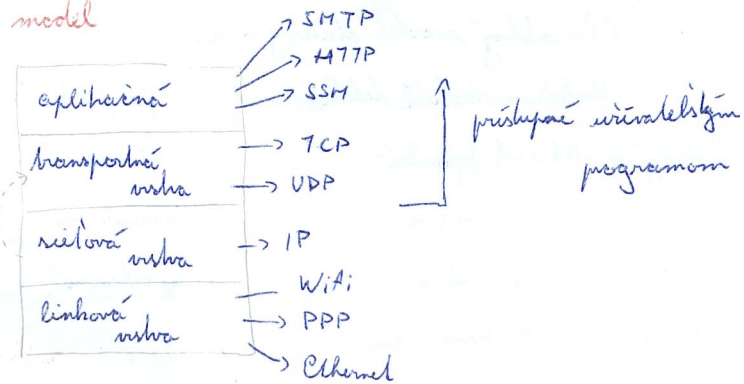
Vrstvový model: 1) fyzická
2) spojová
3) sieťová
4) transportná
5) relačná
6) prezentačná
7) aplikačná

Vrstvy sú „úrovne abstrakcie"





TCP/IP model



- Prvé verzie internetových protokolov boli vymyslené v DARPA
(Defense Advance Research Project Agency)
- riadenie: komunikácia v nespoľahlivom, rušenom prostredí
- robustnosť, redundantnosť
- scenár: vojnový konflikt

„Normy" internetu:

RFC: Request For Comments

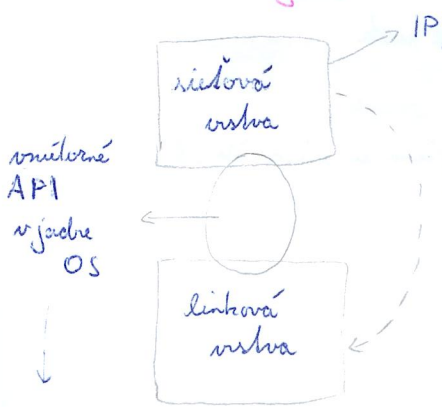
Vydávané:

IETF - Internet Engineering Task Force

RFC dokumenty - písané jednoznačným, zrozumiteľným jazykom
- každý ho môže navrhnúť

IP protokol, adresácia, smerovanie paketov

TCP/IP vrstvy



požadovaná služba:

* doručenie paketu po fyzickom médiu
- ethernet kábel (sieť)
- frekvenčný rozsah rádiových vĺn

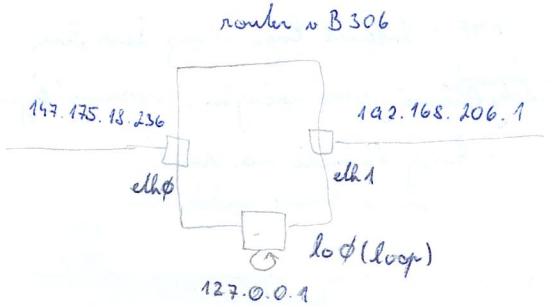
(abstrahujeme od typu média)

- sériová linka
- telefónny kábel

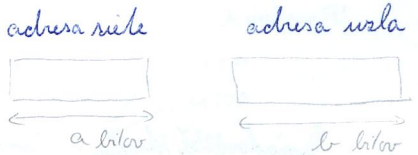
* zariadenie je určené IP adresou (v rámci média)

IP adresácia (v4)

IP adresa: - 4 bajty (oktety) = 4 × 8 = 32 bitov
- zápis 0 - 255. _ . _ . _ . _
- jednoznačne určuje rozhranie uzla v rámci internetu



IP adresa má 2 časti:



IP adresy sa udeľujú po sieťach

IP adresy sú zadarmo

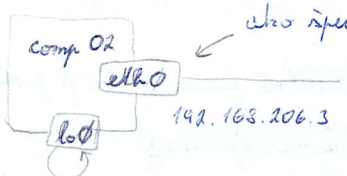
Triedy adries

Trieda	prefix	adresa siete	veľkosť siete	
A	0	a	2^{24} uzlov	$0 < a < 128$
B	10	a.b	2^{16} uzlov	$128 < a < 192$
C	110	a.b.c	2^{8} uzlov	$192 < a < 255$

Sieť sa delí na podsiete: (KM FEI STU)

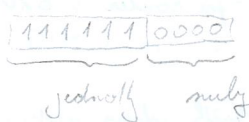
147.175. 96.

Z hľadiska uzla sa IP adresy delia na 2 disjunktné množiny:
- dosiahnuteľné priamo (na tom istom fyzickom médiu, ale niektoré z rozhraní uzla)
- nedosiahnuteľné priamo



ako špecifikovať ktré IP adresy sú dosiahnuteľné priamo?

Každé rozhranie má - IP adresu
- netmask „adresa" tvaru



IP 192.168.206.3 =: a

netmask 255.255.255.0 =: nm

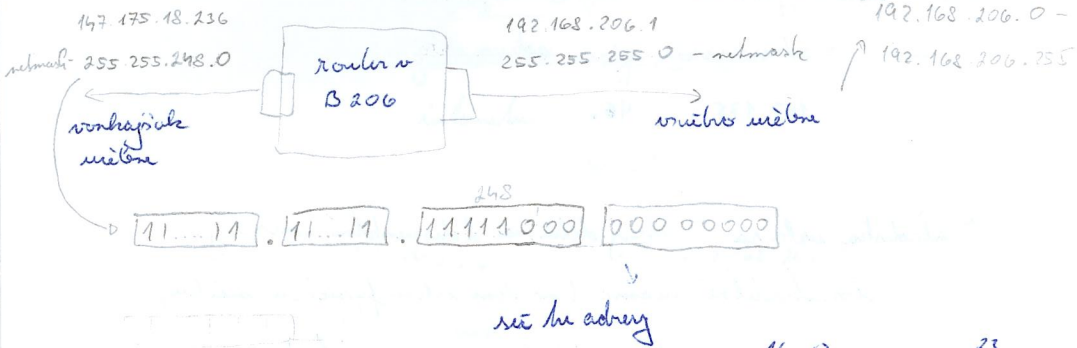
je adresa x na tomto rozhraní?

ÁNO, ak

x & nm = a & nm

↓

pobitoh



128 64 32 16 8 4 2 1

sieťové adresy

147.175. ~~16.0~~ 18.232 - 147.175. 23 ~~18~~.255

Ak vznikne potreba komunikovať

- ak je adresa dosiahnuteľná priamo, nie je problém
- inak sa paket pošle na gateway

(zjednodušenie (gateway môže byť viac ako 1.))

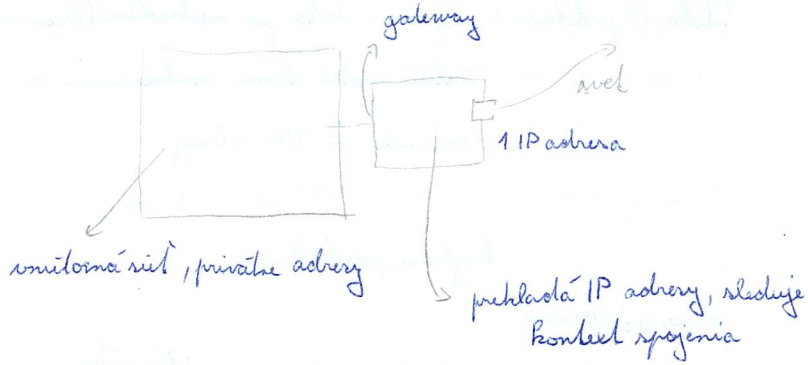
- pre uzly v miešni je gateway 192.168.206.1
- pre router v B206 je gateway 147.175.16.1

Všetky adresy nedosiahnuteľné priamo idú na gateway

Pre vnútorné účely organizácií sú vyhradené privátne siete

- CLASS B: 192.168.0.0
- musí sa zaberpečiť, aby pakety s týmito adresami nepreniknú na „vonkajší internet"
- automaticky sa zahadzujú na prechode von

- spôsob fungovania bežného pripojenia domácnosti



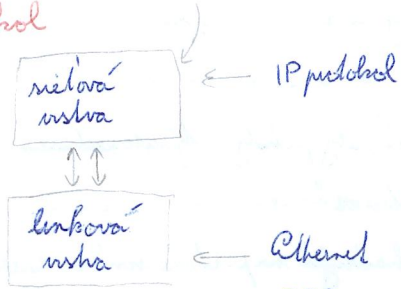
Dynamické pridelovanie IP adries

(DHCP) - Dynamic Host Configuration Protokol

- rozhraniu je možné pripojiť IP adresu napevno
- dynamický: po pripojení rozhrania sa vyšle do celej siete naraz požiadavka na konfiguračný paket
- tá prebieha na linkovej úrovni
- príde jeden paket obsahujúci (okrem DHCP servera) informácie o adrese, netmask, gateway, DNS server, ...

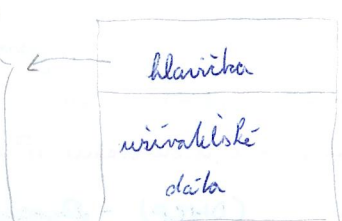
hranice - program, ktorý nájde cestu k internetu

IP protokol



Úloha IP protokolu: doručiť dáta pre nadradenú (transportnú) vrstvu medzi dvomi rozhraniami na internete ≅ IP-adresy

Obsah IP paketu (vybrané položky)



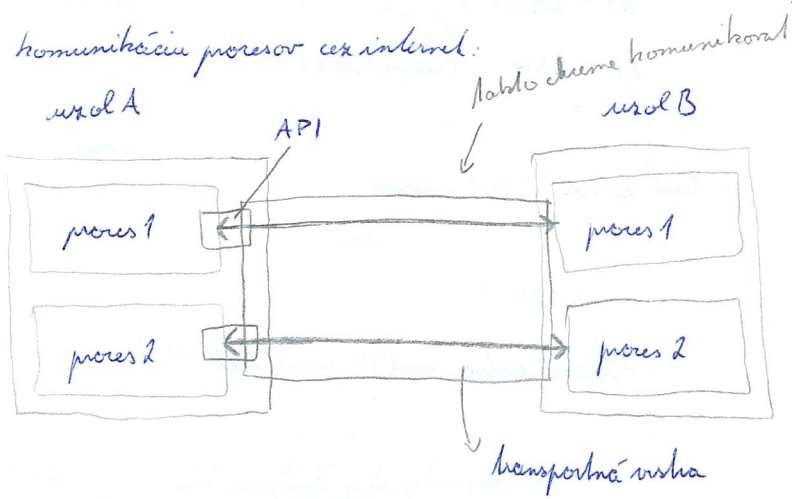
- verzia IP
- celk. dĺžka hlavičky
- celková dĺžka paketu
- zdrojová adresa
- cieľová adresa
- TTL
- protokol (transportnej vrstvy)
- kontrolný súčet

TTL (Time To Live) je číslo, od ktorého každý router odčíta 1; ak je rovné 0, paket sa zahodí

Použitie IP protokolu spravidla vyžaduje práva superužívateľa (priamo)

Transportná vrstva

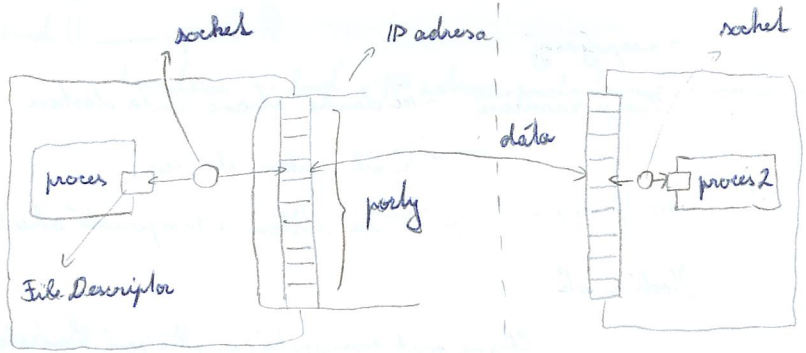
- Rieši komunikáciu procesov cez internet:



- Rozdeľuje komunikačnú kapacitu na menšie časti, ktoré budú:

1) smerom „dnu" napojené na procesy
2) smerom „von" bude každá časť kapacity adresovateľná

(v UNIX-e)



porty → čísla 1 - 65535

- Komunikácia cez porty 1 - 1024 je vyhradená pre superužívateľa.
- Rôzne aplikačné protokoly bežia na rôznych portoch;

  80 - HTTP (web)

  21 - SMTP (pošta)

  22 - SSH (secure shell)

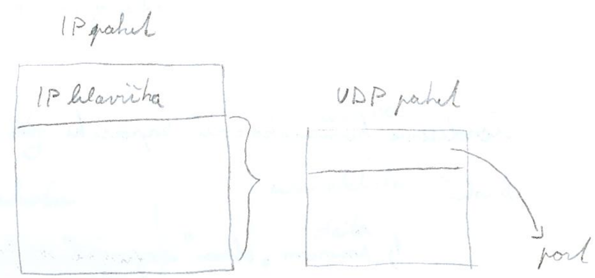
  ⋮

- Viď súbor /etc/services

UDP protokol

- tenká vrstva nad IP protokolom

-



Vlastnosti UDP protokolu

- nespojený

Nie je zaručené
- že druhá strana dáta dostane
- že ich dostane iba raz
- že ich dostane v tom poradí ako boli vyslané

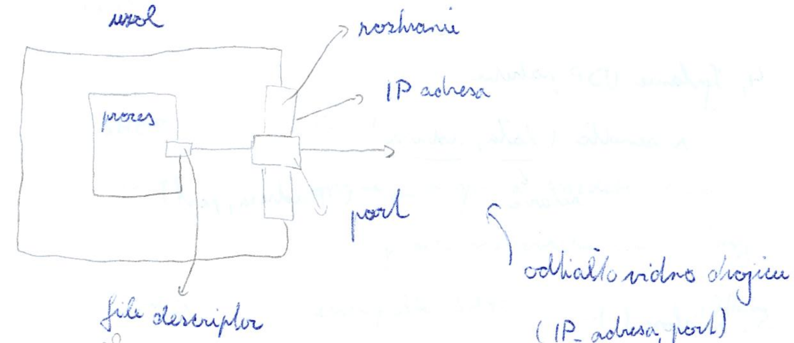
Hodí sa ak:

- Chceme mať komunikáciu plne pod kontrolou
- Chceme vysielať a prijímať iba malé množstvá dát naraz (DNS) -> prevod doménového mena na IP adresu

- Potrebujeme komunikovať v reálnom čase - multimédiá
  (sťahovanie cez Torrent, video, ...)

7.10

Dres: Chat server + klient (UDP)



programové celky

- rozhranie pre programovanie aplikácií, ktorého funkcie/objekty je možné spôsobom na mieste volať zo zdrojového kódu

Pythonovské API (v iných jazykoch je to veľmi podobné)

1, vytvorenie socketu

  s = socket.socket (socket.AF_INET, socket.SOCK_DGRAM)

2, (nie povinný), len keď chcem pripojiť na konkrétny port

  s.bind ((____, port)) -> umiestnená schránka - dvojica

  ↳ reťazec -> buď - IP adresa prislúchajúca k nejakému rozhraniu

  -> alebo - prázdny reťazec (všetky rozhrania)

3, Prijímanie UDP paketu

  s.recvfrom (____) - čaká, kým nepríde paket

  ↓ maximálna akceptovateľná veľkosť paketu

- vráti dvojicu
  - dáta
  - adresa = (IP adresa, port)
    (IP adresa = reťazec)

4, Vyslanie UDP paketu

s.sendto (dáta, adresa)

dáta = reťazec; adresa → (IP adresa, port)

5, s.close ( )

uvoľnenie portu, „zatvorenie" socketu

Pre non-root procesy je bind povolený len na porty < 1024

mpm.nf.stuba.sk - povolené porty na firewalle 9995-9999

UDP chat server + klient

- port servera 9999

PROTOKOL (pravidlá komunikácie)

- každý paket vyzerá takto

___ : ___

príkaz (neobsahuje :) : argument (znaky UTF-8 kódovanie)

4 príkazy

HELLO (nastav mi nick)
ISAY - ja hovorím
SAYS - hovorí niekto iný
ERROR

PRÍKAZ	SMER	POPIS
HELLO: nick	C -> S	povinne prvá správa každého klienta; nick je reťazec neobsahujúci medzery
ISAY: správa	C -> S	server pošle SAYS: nick␣správa všetkým klientom ktorých pozná
SAYS: nick␣správa	S -> C	klient vypíše správu na svoju obrazovku
ERROR: číslo␣popis	S -> C	uvidíme

python

```
import re
data = "HELLO:vtipnynick"
re.match("[^:]*:.*", data)
m = re.match("^([^:]*):(.*)$", data)
m.start()
m.end()
m.group(0) - všetko
m.group(1)
    'HELLO'
m.group(2)
    'vtipnynick'
```

nez vyjasníme ako SHELL

Program netcat (nc)

- vytvorenie socketu na príkazovom riadku

nc -u 127.0.0.1 9999

localhost (môže byť aj iná adresa)

HELLO by malo vytvoriť priradenie (adresa, port) ⟼ nick
- slovníky!

clients = {} - prázdny slovník

TCP protokol

- spoľahlivá ~~spojová~~ komunikácia (viď prvá prednáška)
- spojaná komunikácia
- prúdový spôsob prenosu

nad IP

TCP protokol:

- nadviazanie a rušenie spojenia
- opakovanie dát v prípade straty
- zaberpečenie prispôsobenia rýchlosti (komunikácie) výmeny dát rýchlosti spojenia

skrytié pred aplikačnou vrstvou

API pre TCP protokol

A) server

„serverový socket"

1) s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

s.setsockopt (socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

kvôli možnosti sa ihneď znovu navviazať na (IP_adresa, port)

2) s.bind ((IP_adresa, port))

3) s.listen (n) $1 \leq n \leq 5$ - koľkých žiadostí o spojenie môže brať vo fronte v 1 chvíli

4) s.accept() - blokuje chod programu kým sa nepripojí klient

vráti dvojicu (connected_socket, adresa)

(IP_adresa, port)

nový socket pripojený na klienta → connected_socket.send, recv

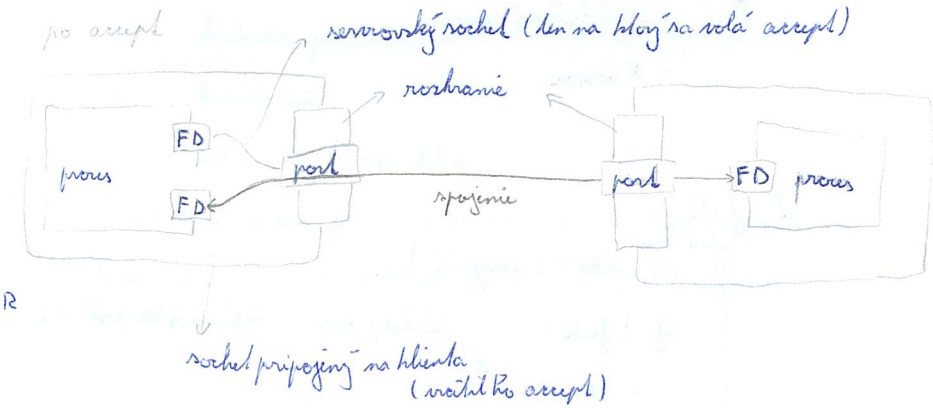
B) klient

1) ako server, nebreba setsockopt

2) s.connect ((IP_adresa, port))

3) Posielanie: s.send (reťazec) - pošle dáta

s.recv (max_dĺžka) - prijme dáta



TCP server

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

(margin: mam select)

- riešenia:

1, select - umožňuje sledovať na viacerých file descriptoroch naraz či sa niečo nedeje
- umožňuje zistiť či je možné čítať / zapisovať na nejaký filedescriptor (= socket teraz)
  - select(...)
  ↓
  - zistím, či klient niečo poslal
    - je chce sa pripojiť nový klient
  - vybavím

používa sa niekedy, ak klienti pristupujú ku zdieľanému zdroju, ktorý môžu meniť

Pr.: - databázové servre
- X server

2, fork

```
while True:
    CS, addr = s.accept()
    if !fork:          child proces - má na starosti cs
        s.close()      - zavrieme file descriptor prislúchajúci k ch procesu
        :
        : komunikuje so klientom cez CS
        cs.close()     - nemusí byť, sys.exit to zavrie
        sys.exit()
    else:
        cs.close()     ← parent proces
```

Textovo orientovaný protokol

Po pripojení:

S → C SUMATOR 1.0 → server vynuluje počítadlo

Riadky:

C → S CISLO␣(prirodzené číslo)\n

S → C 100 OK\n (ak je všetko v poriadku)

200 Zle cislo\n (ak nie je dobré číslo)

server pripočíta k počítadlu číslo

C → S Suma\n

S → C 100 OK\n
hodnota_počítadla\n } + počítadlo vynuluje

C → S hocičo iné

201 Zlý príkaz\n

Pozn.: Pri vracaní statusu je dôležité iba číslo, nie popisujúci reťazec za ním. 201 Bad command\n je v poriadku

14.10.

KLIENT k sumátoru

sčítaj server číslo$_1$ .... číslo$_n$

- pošle príslušné príkazy na server
- vypíše výsledok

DNS

Domain Name — Systém
\ Service

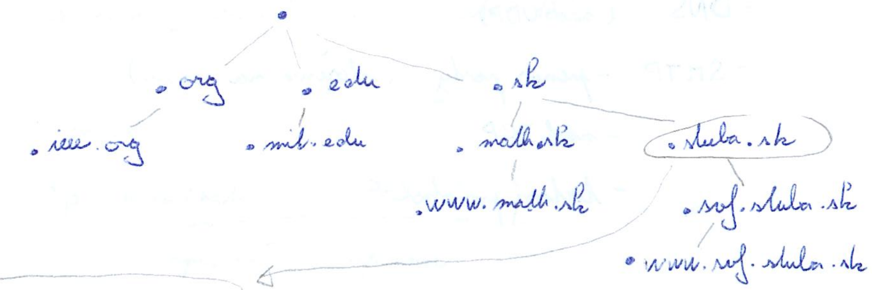
- Principiálny pôvodný účel: mapovanie symbolických mien na IP adresy
- Neskôr ďalšie použitie
- Princíp implementácie: distribuovaná hierarchická databáza
- Rieši problém mapovania
  www.stuba.sk —?→ IP adresa

- Pôvodne bol na každom uzle jeden textový súbor HOSTS
- Dnes /etc/hosts
- Riešenie (zjavne) nebolo dobre škálovateľné

Ako to funguje:

- Každý na internete, kto chce zistiť aká IP adresa patrí danému menu sa to musí spýtať DNS servera
- Klientsky kód je súčasťou C knižnice

Hierarchia mien



tabuľky záznamov
TYP : HODNOTA
→ pre podriadené domény
→ - kto má na starosti Pošta
- IPv4 adresa
- IPv6 adresa
- ďalšie záznamy

- Každý (bod) uzol v hierarchii môže mať na starosti iný server.
- Každý počítač môže riešiť DNS záznamy sám, alebo to delegovať na iný stroj: to je obvyklé

DOMA:

náš počítač → DNS server vášho providera
- cachuje záznamy

.y.z → tu sú uložené záznamy pre .x.y.z
\ .x.y.z

TYP

A - IP adresa

NS - IP adresa DNS servera

MX - IP adresa poštového servera

Aplikačné protokoly

- DNS (nad UDP)
- SMTP - prenos pošty (uloženie na serveri)
  - nad TCP
  - textový protokol

HTTP protokol

- vymyslený v 80tych rokoch v CERN (Ženeva)
- T. Berners-Lee
- aplikačný protokol siete www
- vymyslený dobre
- súčasná verzia: HTTP 1.1
  - nové vlastnosti oproti 1.0
    1, viac požiadaviek na 1 spojenie
    2, virtual hosts - umožňuje viac webov zdielať 1 IP adresu - líšia sa len menom

URI - uniform resource identifier

- syntax identifikácie zdroja

scheme : hierarchical_part ? query # fragment

„TYP": určuje ako vyzerajú ďalšie

mailto: username@example.com ? subject = Hello

URL - podtrieda URI

http:// host / path ? .... # ....
(nepovinné, nepovinné)

file: /// home / user .....
(súčasť cesty)

ftp: // ....

HTTP URL

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

(username: nemusí byť; password: nemusí byť; port: číslo portu (nemusí byť, default = 80); / : absolútna cesta)

Niekedy (ak je známy kontext 1) = 1) host + ...
2) - 11 - + prefix cesty) sa udáva

ila 1, absolútna cesta
2, relatívna cesta (- vzhľadom na známu adresu dokumentu)

absolútna - začína /
relatívna - nezačína /

Tvar správ HTTP protokolu

C -> S (požiadavka)

metóda ␣ URL ␣ verzia_protokolu
(GET) (adresa) (HTTP/1.1)

[
Headre - slúžia na bližšiu špecifikáciu požiadavky
]

- prázdny riadok

[
obsah požiadavky (nemusí byť)
]

S -> C (odpoveď)

verzia_protokolu ␣ status ␣ popis_statusu
(status ↓ číslo)

[
Headre : špecifikácia vlastností odpovede (čas, posledná modifikácia, X-privatne (Chrome posiela)?, server, typ obsahu, ...)
]

- prázdny riadok

[
obsah odpovede (nemusí byť)
]

C -> S podrobne

zdroj - je určený URL

metódy:

najčastejšie:
- GET
- PUT
- POST

GET, PUT => idempotentné metódy

POST - nie idempotentná metóda

DELETE

HEAD

OPTIONS

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

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

POST - pridaj k zdroju (zmeň zdroj)

DELETE - zmaž zdroj

HEAD - ako GET, ale vráti iba headre nie obsah

OPTIONS - nevieme

Headre

Každý header má tvar:

názov-headra: obsah
↓
nezáleží na veľkosti písma

- headre končia prázdnym riadkom

Accept: (typy dokumentov, ktoré vieme spracovať)

Accept-Language: preferované jazyky

Host: DNS-meno: port
↓ jediný povinný header; port ↓ nemusí byť

↓

toto vyplnil užívateľ v adresovom riadku

Môžeme mať viac „virtuálnych webov" na 1 IP (šetrí IP adresy)

www.math.sk
↓
rovnaká IP → web server ich rozlíši pomocou Host: headra
↑
nanyzotapanie.sk

21.10.
P5

C → S

metóda url verzia

headre

—— prázdny riadok

obsah requestu

S → C

verzia status-kód popis-statusu

headre

—— prázdny riadok

obsah

Headre C → S

Accept: MIME-type(s), oddelené čiarkou

MIME - Multimedia Internet Mail Extensions
- súčasť systému RFC zavádzajúceho to, čo poznáme (nemo zalesntu)
ako „prílohy pošty"
- MIME type:

x / y
type ← → subtype

x ∈ { image, text, audio, video, application, ... }

- MIME types sú centrálne prideľované organizáciou IANA
- Neutrálny MIME type - nevieme aký
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

Cookie : [bude o týždeň]

Content_Length: ___ ← dĺžka obsahu zasielaného klientom v bytoch

Content_Type: MIME_type ← typ zasielaného obsahu

Date: ← čas posielania requestu

Host: ___ : ___ ← identifikácia požadovaného virtuálneho host

doménové meno : port → toto nemusí byť

If-* : ___ ← „podmienená požiadavka"

špecifikácia — kvôli šetreniu zdrojov

If-Modified-Since: ___ ← server pošle normálnu odpoveď ak

dátum, čas — bol zdroj zmenený od daného času;

ak nie, pošle status 304 Not Modified,

bez obsahu

If-None-Match : ETAG

- dlhé hexadecimálne číslo, v praxi: hash obsahu

- klient sa ho dozvie z Etag: ___ headra posielaného

serverom pri predošlej požiadavke

- v praxi Etag posielajú „reverzné proxy"

↑ ak sa obsah zmenil oproti hodnote identifikovanej ETAG-om, server ho

pošle, inak 304 Not Modified

Referer: URL ← URL, ktoré linkovalo na požadovanú stránku

User-Agent: identifikácia browsera (klienta)

S → C headre

Content-Type: MIME type obsahu

Content-Length: dĺžka obsahu

Date: ___ ←

Etag: ___ ← identifikácia obsahu (pre if-none-match)

Location: URL ← pri presmerovaní

Server: ___ ← identifikácia servera

Status kódy v odpovedi: trojciferné čísla

1xx : informačné

2xx : úspech

3xx : presmerovanie

4xx : chyba klienta

5xx : chyba servera

200 OK

201 Created (odpoveď na PUT)

301 Moved permanently } teoreticky rôzne, v praxi synonymá

302 Found

↓

tento zdroj nájdeš na URL, ktorú je v Location: ___ headri

304 Not modified ; odpoveď na if-... headra

400 Bad request

404 Not found

403 Forbidden

408 Request timeout

418 I'm a teapot (1. aprílový žart)

500 Internal Server Error

502 Bad Gateway ← padnú reverzné proxy

22.10.

HTTP S → C

Content-length: ___ (bytes)
⋮
___
obsah

Transfer-encoding: chunked \r\n
⋮
hexadecimálny počet bytov \r\n
↓ obsah
⋮ znovu
⋮
0 – koniec dát

28.10.
P6

HTTP Autentizácia

Server: pošle status 401 Not authorised
- v headroch:

WWW-Authenticate: Basic realm = "___"

v ktorom na akú rolu sa ~~autentifikuje~~ autentizuje

Klient:
- vypýta si meno a heslo od užívateľa
- vytvorí reťazec meno: heslo
- zakóduje ho pomocou algoritmu base 64
- pošle request znovu, pričom uvedie v headroch

~~Authentication~~ 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 uložiť niekoľko krátkych dvojíc meno = hodnota
- Každá dvojica môže mať uvedenú platnosť (do kedy platí)

Mechanizmus:

S → C sa pošle header Set-Cookie: $meno_1$ = $hodnota_1$;
môže byť viacero takýchto headrov.

Od tejto chvíle klient posiela v každom requeste
C → S
header Cookie: $meno_1$ = $hodnota_1$; ...

~~V praxi sú hodnot~~

V praxi sú hodnoty cookies nezrozumiteľné kódy, ktoré referencujú na nejakú položku v databáze.

V

Set-Cookie: meno = hodnota; Expire = ...

Dátum, čas (podrobnosti viď RFC)

je možné uviesť explicitnú platnosť;

Ak nie je uvedená, cookie sa poriadi až kým neskončí beh browsera

Cookies sa používajú na - autentifikáciu (heslá k účtom)

- uloženie informácií počas nedenia ("nákupný košík")

4.11.
P7

HTML

- SGML - všeobecný jazyk formátovania dokumentov -
  - ISO norma (80-te roky)
  - 2 časti - syntax dokumentu (generická)
    - spôsob špecifikácie typu dokumentu (DTD document type definition)
  - SGML dokument má rôzne časti
    - obrah
    - tagy < meno_tagu > < / meno_tagu >
    - entity &entita;
    - pomenty

DTD hovorí - aké tagy sú povolené

- v akom poradí, ...

- Je možné skontrolovať, či SGML dokument spĺňa daný DTD (programom)

- Začiatok 90 tych rokov

Timothy Berners - Lee - fyzik pracujúci v CERNe

- prevzal: HTML neformálna „analógia" SGML (nebol DTD)

- úloha: - prenos a editácia dokumentácie v CERNe
- dokumenty: „hypertext" - obsahujúce linky
- vymyslel HTMP + HTTP

- vytvoril prvý browser/editor pre počítače NeXT cube
- NeXTy boli drahé
- Prvá implementácia browsera v X windows: NCSA Mosaic
- Prvý browser nekontroloval dôsledne správnosť syntaxe dokumentu, lebo sa vedelo s tým, že hocikto vytvárať HTML ručne
- Pri chybných dokumentoch sa zachoval „nátvorne"

< b > x < i > abc < / b > d < / i > (- zlá syntax)

- Toto „nátvorné" správanie bolo nutné reimplementovať v iných browseroch
- Dodnes: browser na chybný dokument vrátiť chybu, ale vrátiť „niečo"

- Prvá verzia špecifikácie: HTML 2.0 (ako DTD SGML) 1996
  - vo chvíli, keď vyšla, bola už zastaraná
- 1994-2000 „browser wars"
  - na trhu sú 2 browsery
    - Internet Explorer
    - Netscape Navigator
- Konkurujú si divokým, nepremysleným zavádzaním nových tagov
  + nedôslednou implementáciou tagov konkurencie
- pôvodne HTML - špecifikácia obsahu („sémantické tagy")
- počas browser wars: prezentačné tagy <font> <blink> <marquee>...
- 199x - vzniká konzorcium W3C - normalizačný orgán
  - pokus zaviesť poriadok zlyháva (HTML 3.2 1997)
- 1998 - HTML 4.0 - oddelenie obsahu (HTML) a prezentácie (CSS)
  - presadilo sa, lebo CSS je výhodné pre tvorcov obsahu.
- postupný ústup prezentačných tagov
- súbežne: XML - ideový nástupca SGML (1996-1999)
  - jednoduchšia, 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 IE 6.0 (internet explorer)
  - Netscape krachuje / je tým hriešny
  - vývoj je dlhodobo ustrnutý na IE 6.0
  - Netscape uvoľňuje kód Navigatora pod Open Source licenciou
    ↓
    Firefox - vyvíjaný nezávislou nadáciou

- My budeme používať XHTML 1.0 + CSS 2
- Odporúčaný zdroj: htmldog.com

XHTML dokument (viď kostra.html v repozitári)

<!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 = "hodnota 1" atribút2 = "hodnota 2" > — element
  (vždy úvodzovky)
  ↑ obsah tagu — obsah
  </tag>
  → môže obsahovať vnorené elementy alebo text

- okrem elementov a textu obsahuje XML dokument tzv. entity
  - © ©
  - > >

Elementy bez obsahu

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

XHTML: elementy v obsahu <head>

<title> ..... </title> - nadpis dokumentu
(text)
- typicky: v rámci okna / tabu

<meta http-equiv = "Content type" content = "text/html; charset = UTF-8" />
(UTF-8 ↓ dôležité)

→ chovaj sa, ako keby server poslal takýto mime typ v HTTP headeri

<link rel = "..." /> → externé zdroje týkajúce sa dokumentu (stylesheet, licencia obsahu, ikona, názov záložky, ...)

5. 11.

<body>
↕ vlastný dokument
</body>

- Normálny text
  - viacero bielych znakov (␣, TAB, newline) = 1 biely znak
  - ak chceme použiť medzeru ako „prázdny znak", je to entita   (non-breakable space)
  -   je určené na zabránenie zalomenia: v dome

- Odstavce (pomocou p)
  <p>
  ⋮
  </p>

Tvrdé zalomenie riadku

 ==
</br>
↑ prečo tu je medzera?

Pôvodne boli povolené nepárové tagy, napr.
 (SGML)
- v XML sa to nesmie, každý tag musíme ukončiť

- Riešenie bolo zaviesť
  ↓ bez medzery

- Táto vec ale nebola starými browsermi chápaná ako koniec riadku
- Ale ! <br␣/> ako koniec riadku vyhlásovali
  ergo, medzera je v XML „norme"

- <br␣/> sa aj tak má používať len vo veľmi špeciálnych prípadoch, keď je ukončenie riadku súčasťou textu (verše, ...)

Zdôrazňovanie textu

 . . . . – „emphasize" (default: šikmo)

 . . . . – „strongly emphasize" (default hrubo)

Zastaralé tagy na to isté <i> </i> (italic)

  (bold)

Nadpisy

<h1> </h1> – najvyššia úroveň (najväčší)

⋮

<h6> </h6> → najnižšia

Zoznamy

- → „ordered list" default: číslovaný zoznam (arabské číslice)

  jediné povolené

   elementy tu sú <li> . . .
  ↓
  „list item"

Do vnútri <li> sa smie čokoľvek použiť.

- – unordered list

  ⋮ ↳ miesto čísel bodky

Každý zoznam čokoľvek by mal byť <ul> alebo

Definičné zoznamy

<dl> „description list"

↕ } povolené ~~[illegible]~~ : <dt> . . . </dt> → čo popisujem

</dl> <dd> . . . </dd> → popis

Linky

 text linku
↓
URL a) plné URL (http://...)

b) plná cesta (/

↓ začínajúce /)

„na tom istom serveri ako je tento dokument"

URL dokumentu

referencuje sa
http://x.y.z/g1/g2.html

{ http://x.y.z/P1/P2/P3.html
{ href="/g1/g2.html" toto sa nahradí dodnodou ...

c) relatívna cesta

URL dokumentu { "http://x.y.z/p1/p2.html"
href = "p10.html"

referencuje na "http://x.y.z/p1/p10.html"

ktorá nahradí obsahom src

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

11.11.
P8

Obrázky

 ← minimálna verzia

↓ v pixeloch

↑ kvôli zjednodušeniu vykreslenia

↓ popis obrázka pre slepcov a vyhľadávanie stroje

všeobecný atribút → title = "..." ← popisok (po nabehnutí myšou)

Tabuľky

<table>

</table>

→ „starý" spôsob

- jediný povolený prvok: <tr> </tr> (table row) riadky

v stĺpci

<tr> } → <th> </th> - hlavičková bunka

</tr> → <td> </td> - normálna bunka

↳ horiace

→ „nový" spôsob: → tu sú <tr> </tr>

<thead> </thead>

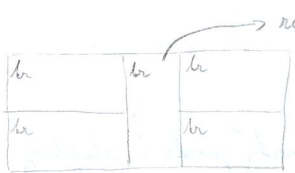
<tbody> </tbody>

<tfoot> </tfoot>

(alebo th)

<td rowspan = "__" colspan = "__" />

číslo číslo



CSS

- Cascade Style Sheets

- Vizuálna prezentácia HTML
  - „ako majú veci vyzerať"

3 spôsoby aplikácie:

1) v atribúte style prvku
2) vo vnútri head <head> </head> dokumentu
3) v externom CSS dokumente

v <head> <link rel = "stylesheet" href= "URL" />
(v html dokumente)

↓ CSS je na tom URL

CSS sa skladá z pravidiel

PRAVIDLO:

```
selektor {
    vlastnosť : hodnota;
    vlastnosť : hodnota;
        ⋮
}
```

(vlastnosti) – ako zmeniť vzhľad

(selektor) → výber množiny elementov

- V jednom 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` – typový selektor (všetko s daným tagom) (p, li, ...)
- `.názov_triedy` – podľa class atribútu
- `tag.názov_triedy` – daný typ (tag) s danou triedou (class)
- `#hodnota_id` – podľa id atribútu (id má byť unikátne v rámci dokumentu)

...

Grouping

```
sel1, sel2, sel3 {
    ⋮
}
```

– zjednotenie množín

Nesting

```
sel1 ␣ sel2 {
    ⋮
}
```

- vyberie prvky rodiane so sel_1
- v ich vnútri vyberie prvky rodiare so sel_2

`sel 1 > sel 2`
- vyberie sel 1
- len z bezprostredných potomkov, ktoré spĺňajú sel 2

`sel 1 + sel 2`
- vyberie sel 1
- len z bezprostredne nasledujúcich (na tej istej úrovni), ktoré sedia s sel 2.

<....> </...> ← tento sel 1
<...> </...> ← tento sel 2

Pseudotriedy

```
sel : hover {
}
```
ak je nad elementom kurzor myši

```
sel : first-letter {
}
```
prvý znak písmena

```
sel : first-line {
}
```
prvý riadok

a : link → nenavštívený link
a : visited → už navštívený link
a : active → „link na kliknutie sťahuje"

```
: before {
    content : "____" ;
}
```
HTML

↓ vopchá pred

```
: after {
}
```
↓ vopchá za

Generické elementy

< div > < /div > → na celú šírku (alebo h1) *
< span > < /span > → v riadku (alebo em) □

* display : block ;
□ display : inline ;

< div > sa používa na rozdelenie stránky na logické časti (do html)

```
<body>
<div id = "hlavicka" >
  ... < h1 > ... < /h1 >
</div>
<div id = "navigacia" >
  ...
</div>
:
</body>
```

Hodnoty atribútov

1) DĹŽKY

1 pixel - 1 cm, 1 px, 1 pt } závislé od média

1 em - šírka veľkého M

10% → relatívne k „normálnej" veľkosti

2, FARBY

- názvom (16 farieb)
- rgb (x_1%, x_2%, x_3%) – miešajú sa svetlá
  - červená, zelená, modrá zložka (zelená + červená = žltá)
- rgb (0-255, 0-255, 0-255)
- #_ _ _ → hexadecimálne dvojcifry 00 - FF
- #_ _ _ → hexadecimálne 1 cifry

Vlastnosti textu / fontu

font-family: "....", "....", "....", "___";
- mená fontov → klesá priorita
- serif alebo sans-serif → vždy k dispozícii

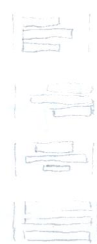
font-size: ___ dĺžka ;

font-weight: bold / normal ; – hrubý

font-style: – normal / italic ; – šikmé

text-decoration: – overline / line-through / underline / none ;

text-align: – left / right / center / justify



line-height: ___ dĺžka ; – treba zadávať aj jednotku ! (1,5em), alebo percento (150%) – výška riadku

Farby

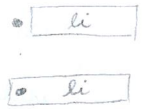
color: farba;

background-color: farba;

Zoznamy (ol, ul) veľa

list-style-type:
- circle ○
- disc •
- decimal 1. 2. 3.
- lower-alpha a. b. c
- upper-roman I. II. III.
- none (nič)

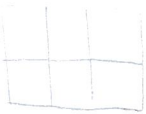
list-style-position: outside; / inside;

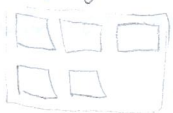


list-style-image: url("....");
URL

Tabuľky

border-collapse: collapse; / separate;





18.11.
P9

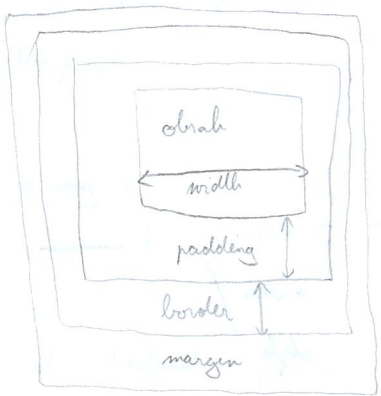
Obrázky na pozadí

background-image: url(___);

background-repeat: repeat-x / repeat-y / no-repeat / repeat (alebo x alebo y) / default

background-attachment: scroll → default / fixed → obsah scrolluje „ponad" pozadie

Škatuľový model (box model)



margin, padding sú „priesvitné" - iba šírka (nie farba)
border: môže mať vlastnú farbu a „tvar"

margin: dĺžka;
padding: dĺžka;

margin / padding - left / right / top / bottom : dĺžka;

border-color: farba;
border-width: dĺžka;
border-style: solid; / dotted; / dashed; / inset; / outset; / ... / none;

Skratka:
border: width style color;

Pr.: border: 2px solid red;

border - left / right / top / bottom : width style color;

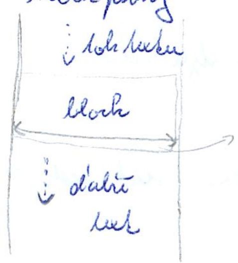
border - left / right / top / bottom - color / width / style : ——— ;

margin má farbu pozadia „okolia"
padding má farbu pozadia „prvku"

Ovládanie rozloženia prvkov na stránke

- display — block / inline ;

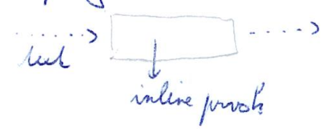
block prvky



celá dostupná šírka

Pr.: p, h1, ..., h6, ul, ol, li
div

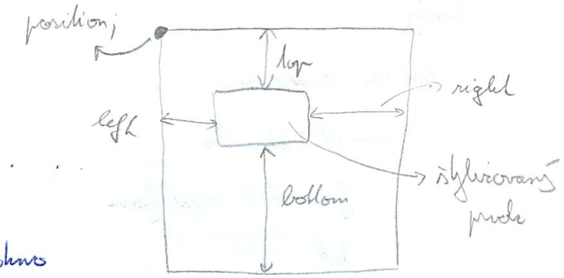
inline prvky



Pr.: em, strong, a, img, table
span

- position: static; (default
  relative;
  absolute; (...
  fixed;
  (vzhľadom na okno browsera)

top:
bottom:
left:
right:

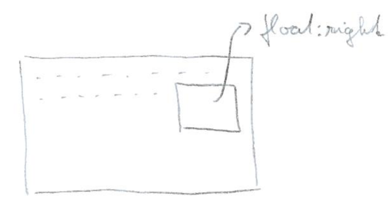


relative - vzhľadom na normálnu pozíciu
absolute - vzhľadom na obaľujúci box

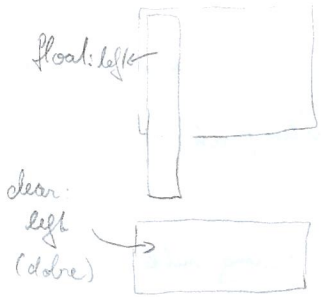
width: dĺžka; → pevné nastavenie šírky
height: dĺžka; → -II- výšky
ignorované pri inline prvku

float: left; right; none;

→ text „obteká"



clear: left
right
both



10.11.

body

div id = container

div id = hlavicka

form id = searchform

h1

div id = navigacia

ul

div id = obsah

div id = hlmul

div id = hlpovedby

div id = footer

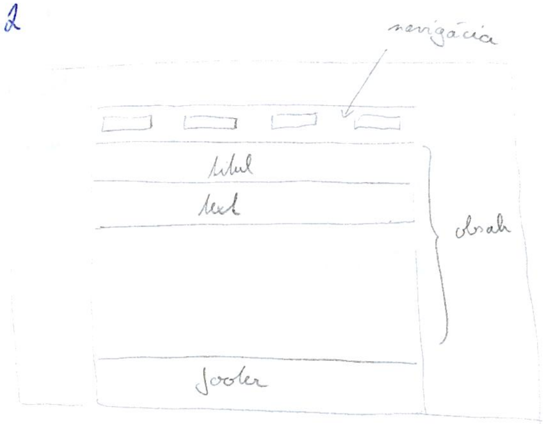
Drazji 1


```
* {
    margin: 0; padding: 0
  }

# container {
    width: 80%;
    margin-left: auto;
    -11 -right: -11-;
  }

# searchform {
    float: right;
  }

# navigacia {
    float: left;
    width: 10em;
    background-color: #fff8;
  }

# obsah {
    border-left: 10em solid #fff8;
  }
```

obe, aby to bolo pekne zarovno

objekt button reagoval na link → v a zarovnat vlastnost display na block

Dizajn 2




```
#navigacia {
    width:
    float: left;
    background-color;
    padding:
}
```

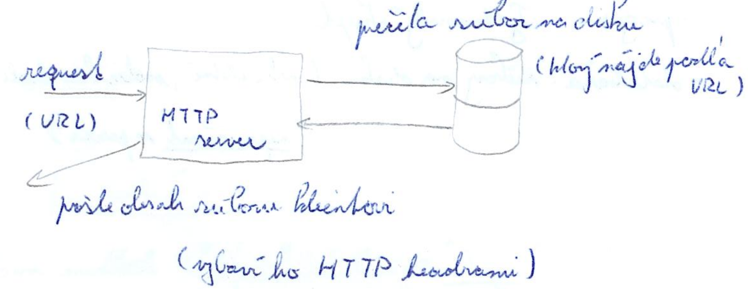
→ + prepracovať dĺžku šírky a výšky

Dynamické webové stránky

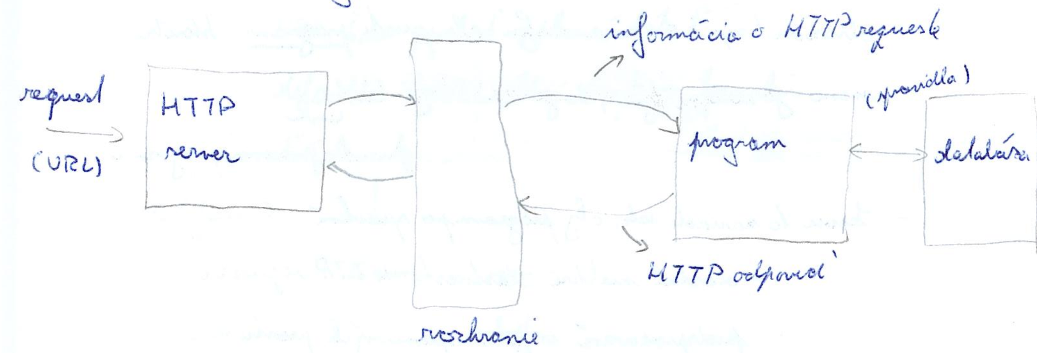
- CGI skripty
  (namiesto pevného HTML sa spúšťa program, ktorý HTML generuje na výstupe)
- pôvodne: spracovanie formulárov, prístup do databáz, ...
- PHP - Fractal of bad design
- Frameworky (Ruby on Rails (RUBY), Django (PYTHON), Code Ignition (PHP), Apache Struts (JAVA))

Dynamický obsah na webe

Statické stránky = súbory na disku



Dynamické stránky (v praxi)



program - píše sa v skriptovacom jazyku (PHP, Perl, Python, Ruby) alebo v Jave

V našom prípade:

- HTTP server: zabudovaný v Pythone
- rozhranie: CGI - Common Gateway Interface
  - najstaršie

- najpomalšie
- vždy fungujúce

- program: Pythonovský skript
- databáza: súbory na disku (nekorektné, vedie ku kolíziám – nepoužívať v praxi)

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/... (tu je meno programu)

- Server to zariadi tak, aby program po spustení
  - dostal niektoré vlastnosti HTTP requestu predspracované vo forme premenných prostredia (podrobnosti neskôr)
  - obsah HTTP requestu (to, čo je po headroch) dostane program na stdin

Program spracuje údaje z requestu a zapíše na svoj stdout
- headre odpovede (nie status!)
- obsah odpovede

- Server preberie odpoveď, skontroluje ju (?), zloží status a pošle klientovi

CGI premenné

HTML formuláre

<form method = "___" action = "...." />

- method: "get" alebo "post" (get = default)
- action: URL, kam poslať vyplnené data; default: to URL, ktoré obsahuje formulár

ľubovoľné elementy (aktívne sú input, select, ...)

</form>

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 typu ?var_1=val_1&var_2=val_2&... (var = názvy vstupov, val = hodnoty vstupov)

– toto je HTML (HTTP) norma

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

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

HTML / HTTP norma:

- 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 headeroch a prázdnom riadku)
- Content-type je application/x-www-form-urlencoded
- Content-length je spravidla tiež uvedená

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

HTML elementy reprezentujúce vstupy vo formulároch

<input type = "...." name = "..." value = "..." />

- type → povolené hodnoty: text; password; radio; checkbox; submit
- name → meno poľa
- value → nepovinné; spravidla iniciálna hodnota

type = "text"

maxlength = "___" maximálna možná dĺžka

size = "___" šírka v znakoch (viditeľná)

type = "password"

(ako text, len zobrazuje *)

type = "submit"

- odosielací butón
- name nepovinné
- value je popiska butónu

type = "radio"

- radio button: ○ OFF ⊙ ON
- browser umožní vyznačiť najviac 1 radio butón, ktorý má danú hodnotu atribútu name; posiela v obsahu name = value

predvolene zapnutý: checked = "checked"

type = "checkbox"

- ako radio, ale z jedného name je možné vyznačiť viac naraz

2.12. P11

Select a Textarea

<textarea name = "...." rows = "...." cols = "...." > (rows, cols → čísla)

↕ počiatočná hodnota

ako text input, len viac riadkov

</textarea>

<select name = "...." .... > ← môže byť multiple = "multiple"

↕ iba <option> elementy

</select>

<select name="...">
<option> .... </option>
→ text
⋮
<option> .... </option>
</select>

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 (miesto try a except)
- Používa sa počas vývoja

import cgitb
cgitb.enable()

Spracovanie formulárov

Ak máme:

<form method="post"> (riadny atribút action)

- browser pošle dáta z formulára na tú ~~url~~, ktorá obsahuje formulár.

Kód, ktorý 1) generuje formulár:
2) spracúva

if metóda je GET:
→ vrát nevyplnený formulár (HTML)
elsif metóda je POST:
→ spracuje dáta a buď uloží, alebo vráti chyby

formulár → <form method="post">
⋮
</form>

Modul cgi

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

a uloží obsah do seba

FieldStorage() podporuje špeciálnymi metódami: ~~testovanie~~

- testovanie prítomnosti premennej cez in;

  "meno" in fs

- iterovanie cez premenné:

  for var in fs:

Ako získať hodnoty koncepčne správne

fs.getlist("___")

názov premennej

vráti zoznam hodnôt, ktoré prišli s týmto menom

POZOR v rôznych tutoriáloch je to zle.

(ak žiadna, vráti prázdny zoznam)

9.12.
P12

Javascript

- vyvinutý v 90-tych rokoch pre Netscape Navigator počas browser wars
- Pôvodný účel: - jednoduché animácie, kontrola vstupov, ...
- Princíp: skript zahrnutý v rámci stránky, meniaci stránku
- Dnes: existuje norma (ECMA Script)

  použitie: - na serveri (node.js) couchdb)

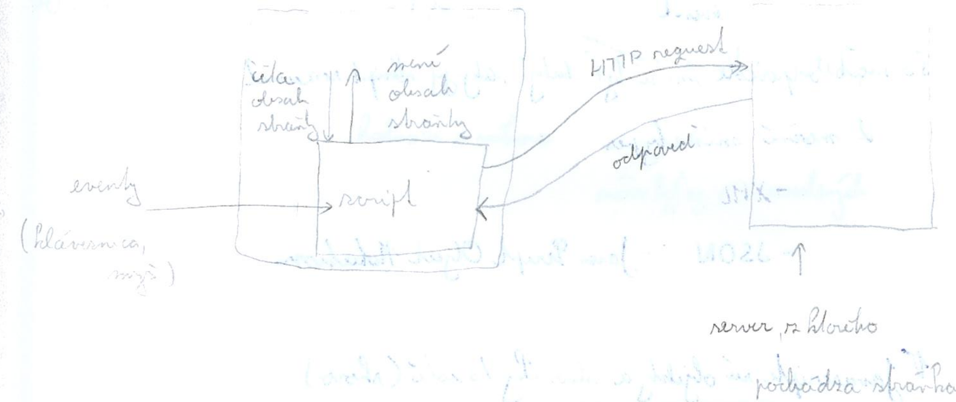
  - na desktope - widgety (Apple, KDE)

Z hľadiska taxonómie jazykov Javascript je:

- funkcionálny
- objektovo orientovaný
- ~~j~~ nemá pojem triedy (ale dajú sa naprogramovať)

Použitie v browseri

HTML stránka



Dnes sa používajú JavaScriptové „frameworky"

bohaté: Dojo, gooxdoo, ...

písané v iných jazykoch, preložené do JavaScriptu: GWT - Google

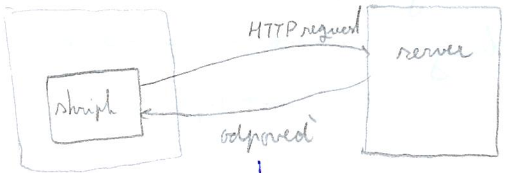
Pyjamas - Python → JS

My si budeme ukazovať jquery.

AJAX

Asynchronous JavaScript and XML

STRÁNKA



(ten istý, v ktorom sa nachádza stránka)

„asynchrónne" – nečaká sa na ňu, ale skript dostane event.

Formát odpovede musí byť taký, aby mu skript rozumel

2 možné mimetypes

- XML
- JSON – Java Script Object Notation

V Javascripte sú objekty a slovníky to isté (skoro)

x = { a: 1,
b: [4, 5, 6] ⟶ JSON syntax
c: "ahoj"
}

– objekt

môžeme robiť x["a"], aj x.a

api.jquery.com

GET

/cgi-bin/student?id=3

↓

vráti JSON objekt { priezvisko: ... ,
meno: ,
email
}

16.12.
P13

POST / . . . . ? n=1&email= . . . .

nastaví email na daný → aj ostatné polia;
hodnotu v súbore ↓
môže byť aj viac uvedených