Správa súborov

- Súbory
- Adresáre
- Implementácia súborového systému
- Kódovanie znakov
- Narábanie so súbormi a adresármi v Pythone

Autor: Peter Tomcsányi

Použité obrázky z učebnice:
Andrew. S. Tanenbaum, Structured Computer Organization
http://www.cs.vu.nl/~ast/books/

Požiadavky na súbory

Priestor na dlhodobé uloženie údajov

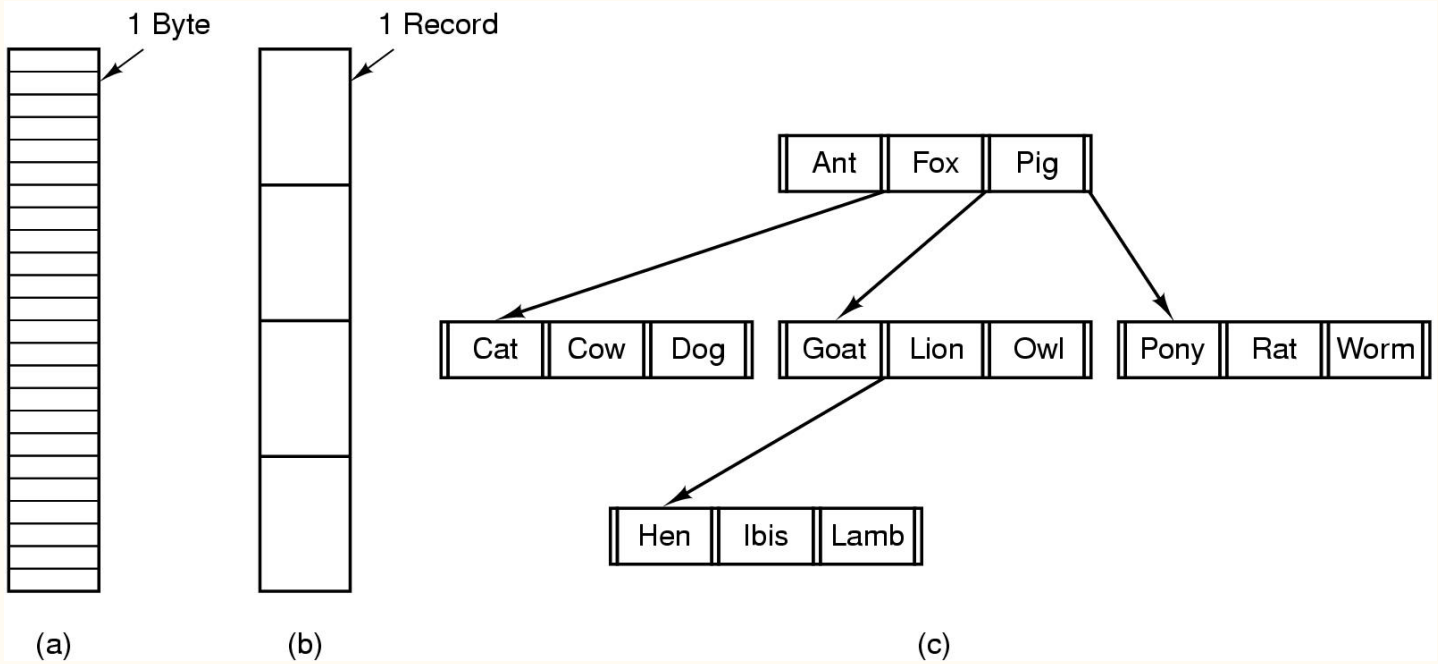
- Musia sa dať uložiť veľké množstvá údajov
- Uložené údaje musia existovať aj keď proces, ktorý ich zapísal, skončí
- K uloženým údajom musia mať súčasný prístup viaceré procesy

Pomenovanie súborov

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Typické mená a prípony (typy) súborov

Štruktúra súborov



Operačný systém môže poskytovať pohľad na súbor ako na:

- (a) postupnosť bajtov
- (b) postupnosť viet (záznamov)
- (c) stromová štruktúra - indexovaný súbor

Moderné operačné systémy väčšinou poskytujú len pohľad (a).

(b) a (c) je nechané na nadstavby operačného systému - napr. knižnice programovacích jazykov alebo databázové systémy.

Typy súborov

- Textové
  - Obsahujú riadky textu v nejakom dohodnutom kódovaní
  - Riadky sú oddelené oddeľovačom riadkov - môže byť závislý od operačného systému napr. Windows používa dva znaky CR a LF, Linux len jeden znak LF, Mac OS jeden znak CR
  - Dajú sa priamo vytlačiť
- Binárne
  - nie sú textové - nedajú sa priamo vytlačiť
  - Majú definovanú vnútornú štruktúru
  - Vnútornú štruktúru rozoznávajú príslušné aplikácie
  - Operačný systém musí rozonávať minimálne jeden typ súboru - vykonateľný program (napr. exe súbor vo Windows)

Prístup k súboru

- Sekvenčný prístup
  - súbor sa číta alebo zapisuje postupne od začiatku po koniec
  - nedá sa v ňom “skákať”
- Priamy prístup (náhodný prístup)
  - bajty alebo vety sa dajú čítať v ľubovoľnom poradí
  - operácia Seek nastaví pozíciu v súbore (na dané poradové číslo vety alebo bajtu) a od nej možno čítať sekvenčne
- Indexsekvenčný prístup
  - umožňuje vyhľadávať vety podľa kľúča a potom od danej vety čítať sekvenčne
  - v súčasnosti ich implementujú databázové systémy, ale v minulosti boli súčasťou operačných systémov

V starších systémoch bol spôsob prístupu atribútom súboru od jeho vytvorenia (hovoríme potom o sekvenčných súboroch, súboroch s priamym prístupom a indexsekvenčných súboroch).
Moderné operačné systémy implementujú väčšinou len priamy prístup a ostatné možnosti nechávajú na užívateľských programoch

Atribúty súboru

Definujú vlastnosti súboru, nie sú súčasťou samotného súboru

- Delia sa zhruba na tri druhy
  - Základné vlastnosti súboru - druh prístupu (sekvenčný/priamy), dátum vytvorenia, dátum poslednej zmeny, dátum posledného prístupu, veľkosť
  - Vlastnosti definujúce spôsob použitia pre ľubovoľného používateľa - Read/only, System, Hidden
  - Ochrana súborov - definujú možný spôsob použitia pre každého používateľa alebo skupinu používateľov zvlášť - vlastník, heslo, práva vlastníka, práva skupín, práva jednotlivých používateľov

Súborové operácie

Create
Delete
Open
Close
Read
Write
Append
Seek
Get attributes
Set Attributes
Rename

Operácie Open a Close umožňujú operačnému systému zrýchliť vykonanie ďalších operácií Read, Write a Seek ako aj implementovať zdieľanie súborov medzi procesmi.

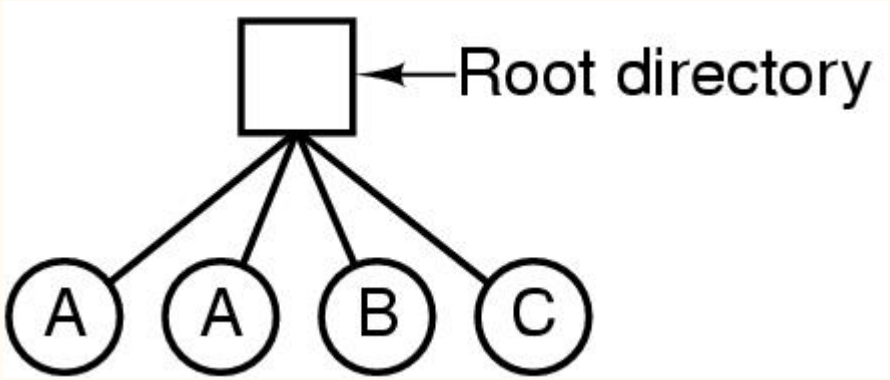
Zoznam uvádza typické mená operácií. Ich skutočné mená v konkrétnom operačnom systéme ako aj ich parametre sa môžu líšiť

Mapovanie súboru do pamäte

- Umožňuje prideliť súboru adresu vo virtuálnom adresnom priestore procesu a pristupovať k jeho obsahu akoby to bola pamäť - teda pole bajtov
- Pre mapovanie musí operačný systém implementovať dve ďalšie operácie - mapovanie do pamäti a zrušenie mapovania
- Pri stránkovaní si mapovanie súboru si môžeme predstaviť tak, že časť virtuálnej pamäte procesu sa odkladá do užívateľom definovaného súboru (teda nie do sytémového stránkovacieho súboru)
- Vo Windows je mapovanie súboru do pamäte možnosť ako zdielať pamäť medzi dvomi procesmi

Adresáre (1)
Jednoúrovňová štruktúra

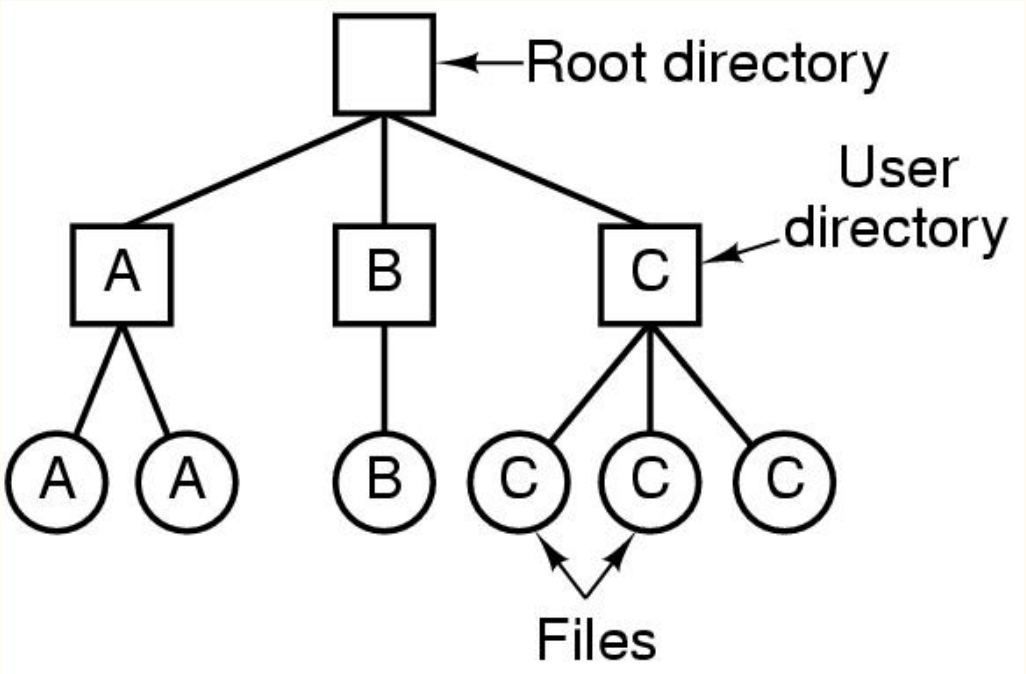
- Na každom disku je len koreňový adresár



Súbory troch používateľov A, B a C sú “pomiešané” v rámci jedného adresára

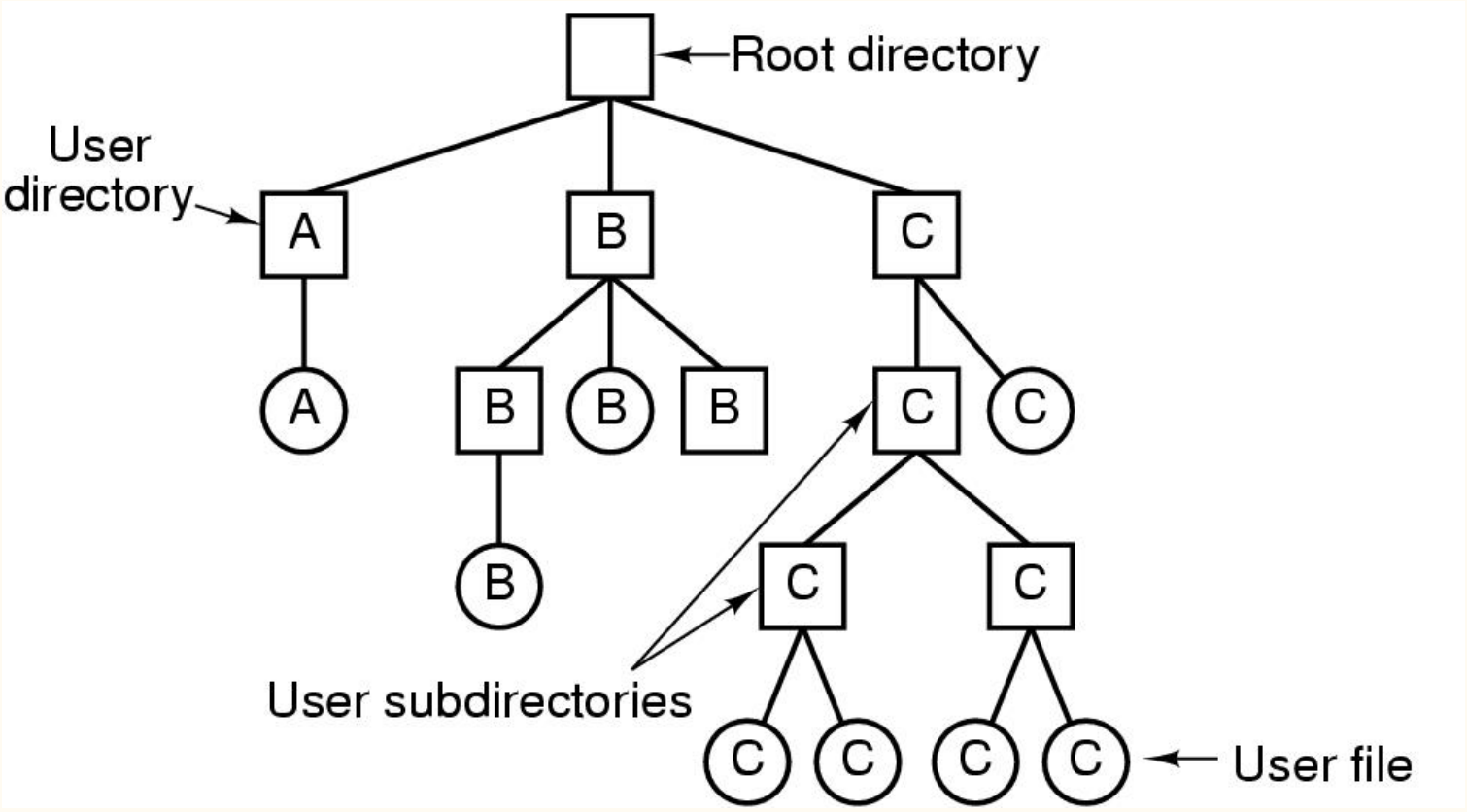
Adresáre (2)
Dvojúrovňová štruktúra

- Koreňový adresár každého disku obsahuje záznamy len o adresároch druhej úrovne
- Každý používateľ má svoj adresár druhej úrovne

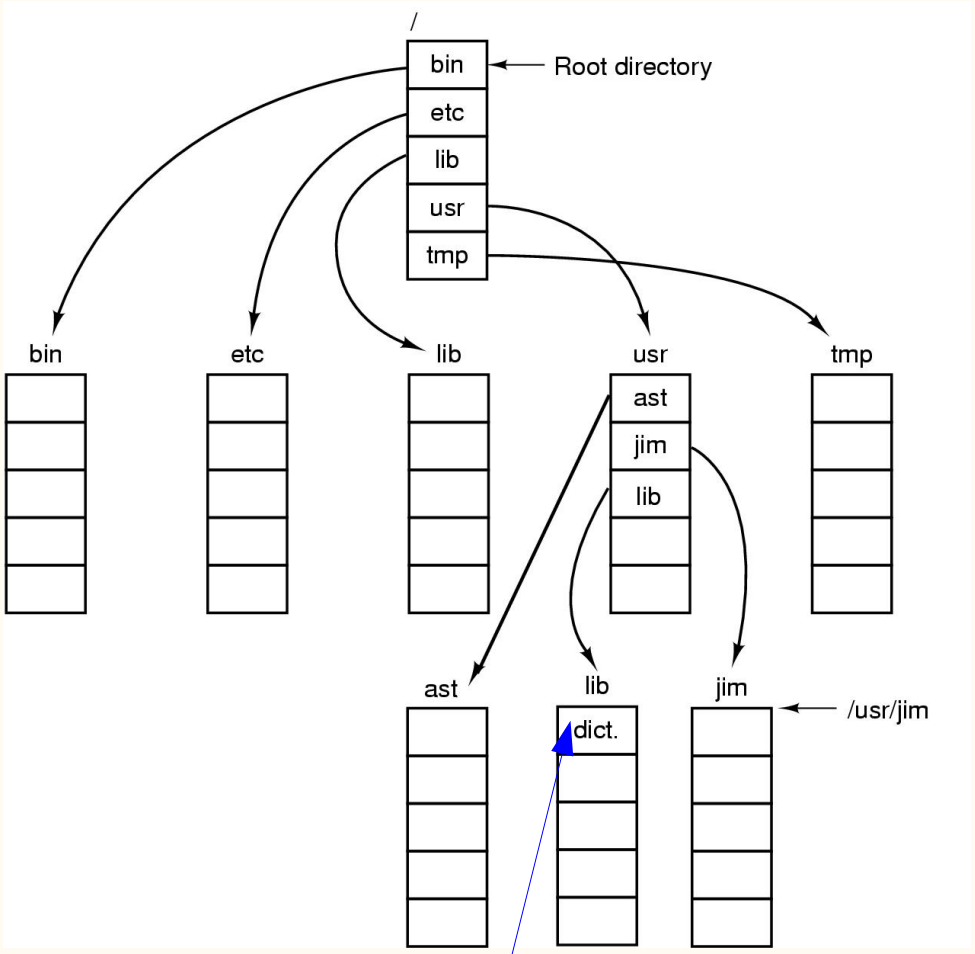


Adresáre (3)
Hierarchická štruktúra

- Adresáre a súbory sú umiestnené v stromovej štruktúre
- Moderné operačné systémy používajú tento prístup
- Každý disk má koreňový adresár (Windows) alebo existuje jediný koreň a všetky disky integrované do jediného stromu (Unix)



Cesty k súborom (Path names)



Príklad z Unixu: /usr/lib/dict

Adresárové operácie

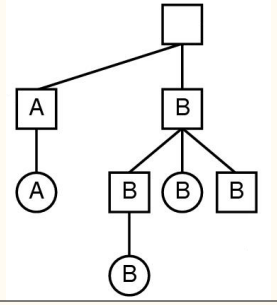
Create	Opendir	Link
Delete	Closedir	Unlink
Rename	Readdir	

Zoznam uvádza typické mená operácií. Ich skutočné mená v konkrétnom operačnom systéme ako aj ich parametre sa môžu líšiť

Opendir, CloseDir a ReadDir umožňujú procesom dozvedieť sa mená existujúcich súborov a adresárov v konkrétnom adresári a tak implementovať operáciu na viacerých súboroch v danom adresári alebo v celom podstrome adresárov. V Lazaruse sú implementované volaniami FindFirst, FindClose a FindNext.

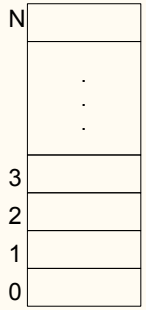
Operácie Link a Unlink umožňujú dať jednému súboru alebo adresáru niekoľko alternatívnych mien, ktoré sa nachádzajú v iných adresároch Vo Windowse sa linky nazývajú Shortcuts (do slovenčiny preložená ako zástupcovia).

Implementácia súborového systému



Disk ako ho prezentuje správa súborov - stromová štruktúra adresárov a súborov

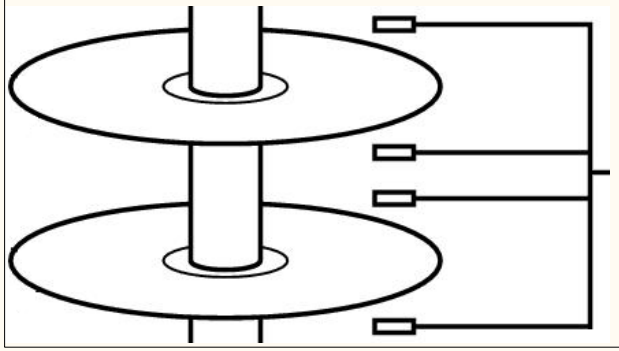
Implementovať súborový systém znamená implementovať túto šípku,
teda vyriešiť ako zobraziť nelineárnu dynamickú štruktúru adresárov a súborov do lineárneho priestoru sektorov
Existuje veľa spôsobov, ako to spraviť, teda existuje veľa **súborových systémov**.



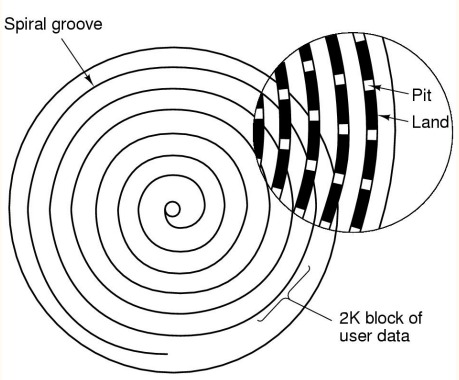
Logický disk ako ho prezentuje správa zariadení - lineárny adresný priestor sektorov s jednoduchými operáciami čítania a zápisu

Toto implementuje správa zariadení

Disk - niekoľko povrchov, na nich stopy a sektory.



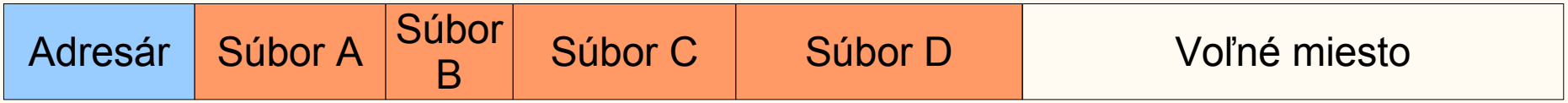
CD - jedna špirálová stopa s blokmi údajov



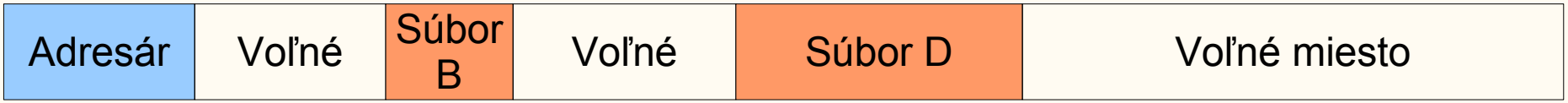
SSD,
SD karta, ...

Toto je hardware diskových zariadení

Spojité umiestnenie súborov



- Používalo sa v starších operačných systémoch
- Každému súboru je vyhradený spojitý úsek na disku
- Keď vytvárame súbor, musíme vedieť aký veľký bude
- Ak súbor prerastie zadanú veľkosť, a za ním nie je voľný úsek, tak ho musíme premiestniť

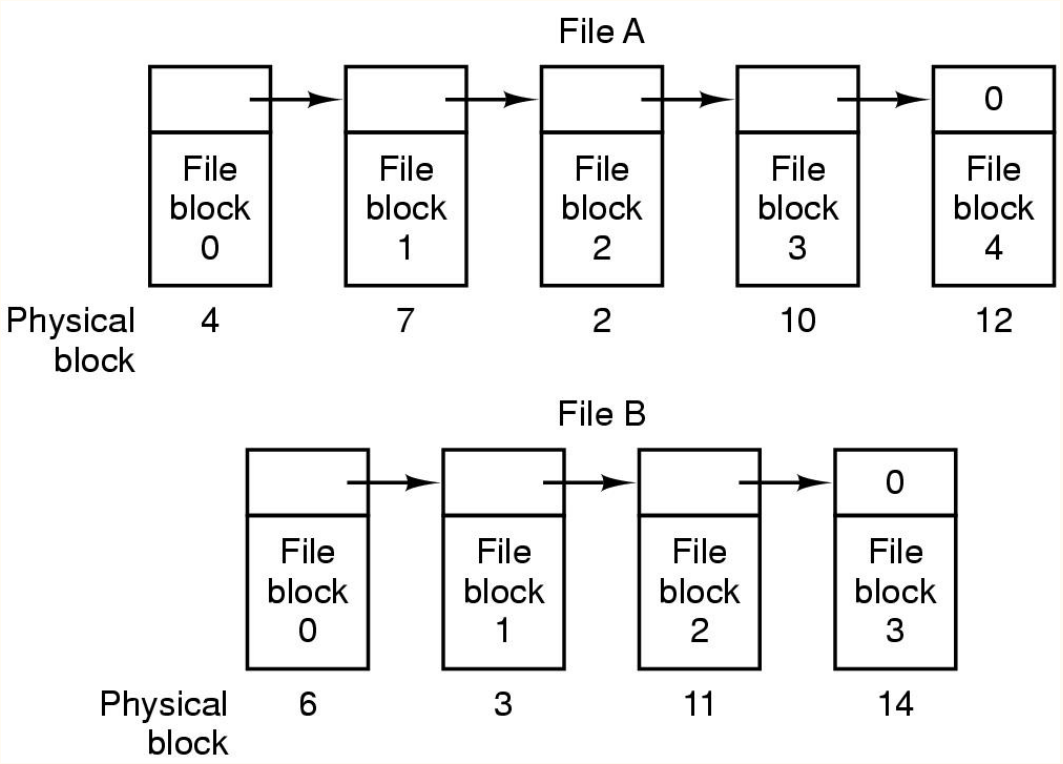


Po zrušených súboroch zostávajú diery - problém vonkajšej fragmentácie (rovnaký ako u jednoduchej správy pamäti)
Kompaktácia disku je oproti pamäti veľmi pomalá

Nespojité umiestnenie súborov Alokačný blok (klaster)

- Súbor môže byť rozdelený na viac kúskov a tie môžu byť umiestnené na rôznych miestach disku
- Aby vonkajšia fragmentácia nenechávala nepoužiteľne malé kúsky, disk je rozdelený na úseky jednotnej veľkosti - alokačné bloky
- **Alokačný blok** (**blok** alebo **klaster,** anglicky **cluster**) je najmenšia jednotka diskového priestoru, ktorá môže byť pridelená jednému súboru
- Alokačný blok je niekoľko sektorov, typická veľkosť alokačného bloku je 4KB, na disketách to bolo bežne 512B (= jeden sektor), môže byť až 32KB alebo 64KB
- Súbor vždy zaberá celý počet alokačných blokov
- Posledný blok nie je plne obsadený - tým vzniká **vnútorná fragmentácia**

Nespojité umiestnenie súborov Možné implementácie (1)

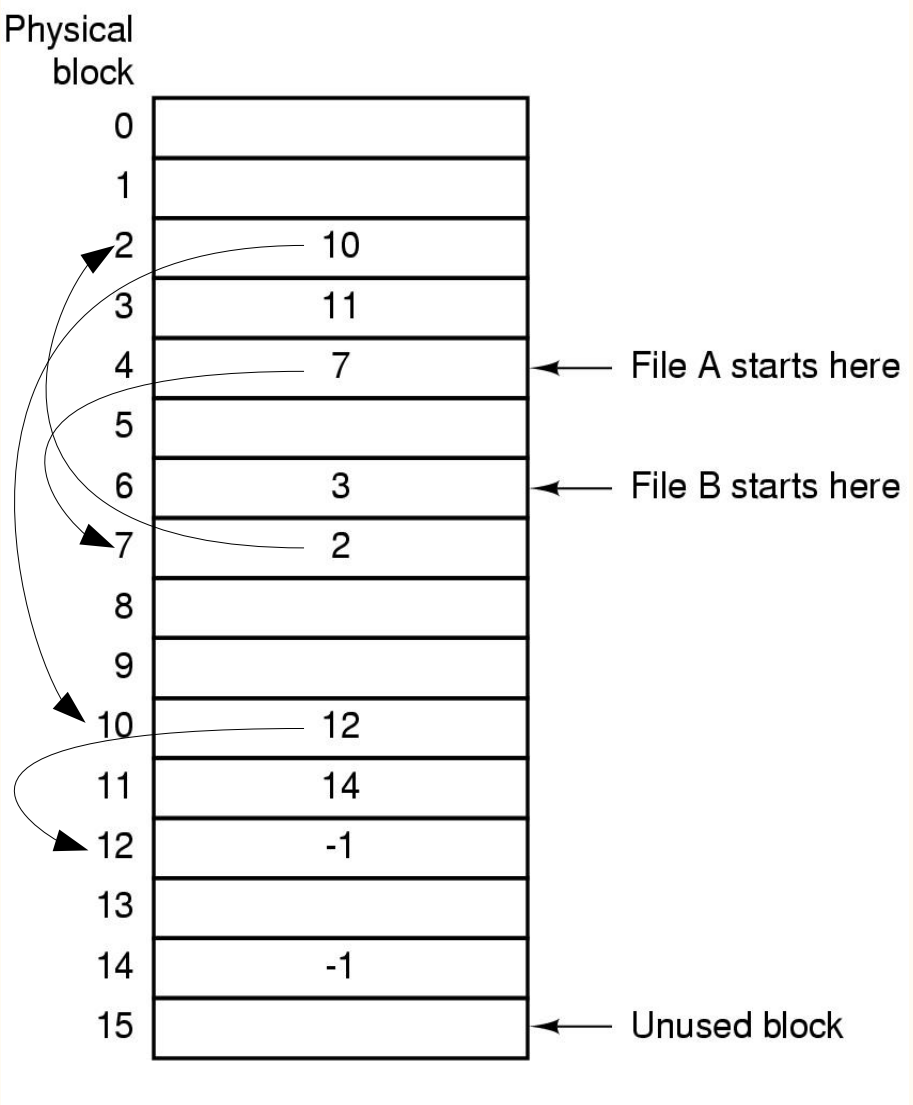


Spájaný zoznam blokov na disku - každý blok súboru obsahuje číslo nasledujúceho bloku

Smerníky sú roztrúsené po disku, preto môže byť pomalý prístup a ľahšie sa poškodia pri poruche disku

Smerníky by bolo lepšie mať koncentrované na jednom mieste disku

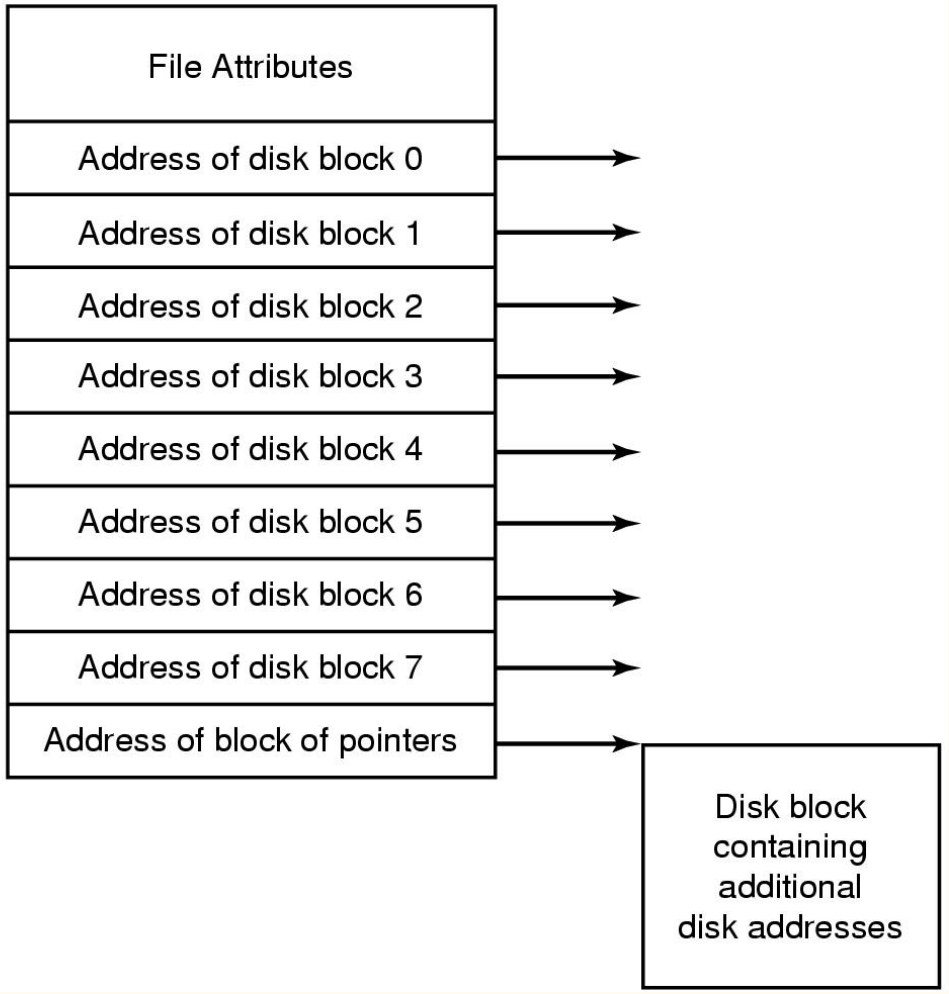
Nespojité umiestnenie súborov Možné implementácie (2)



- Tabuľka pridelenia (Alocation table) - kombinuje bitovú mapu a spájaný zoznam (ale inak ako v správe pamäte)
- Každému bloku disku zodpovedá jedno miesto v tabuľke
- prázdne miesto v tabuľke znamená, že blok je voľný (v reálnej implementácii by to bolo nejaké špeciálne číslo)
- Kladná hodnota určuje číslo ďalšieho bloku súboru
- -1 znamená, že daný blok je posledný v súbore
- Každé miesto v tabuľke musí mať toľko bitov, aby sa do nich dalo zobraziť najväčšie možné číslo bloku

Táto myšlienka je základom súborových systémov FAT12, FAT16 a FAT32, ktoré vznikli v MS-DOSe a vo Windows a používajú sa dodnes

Nespojité umiestnenie súborov Možné implementácie (3)



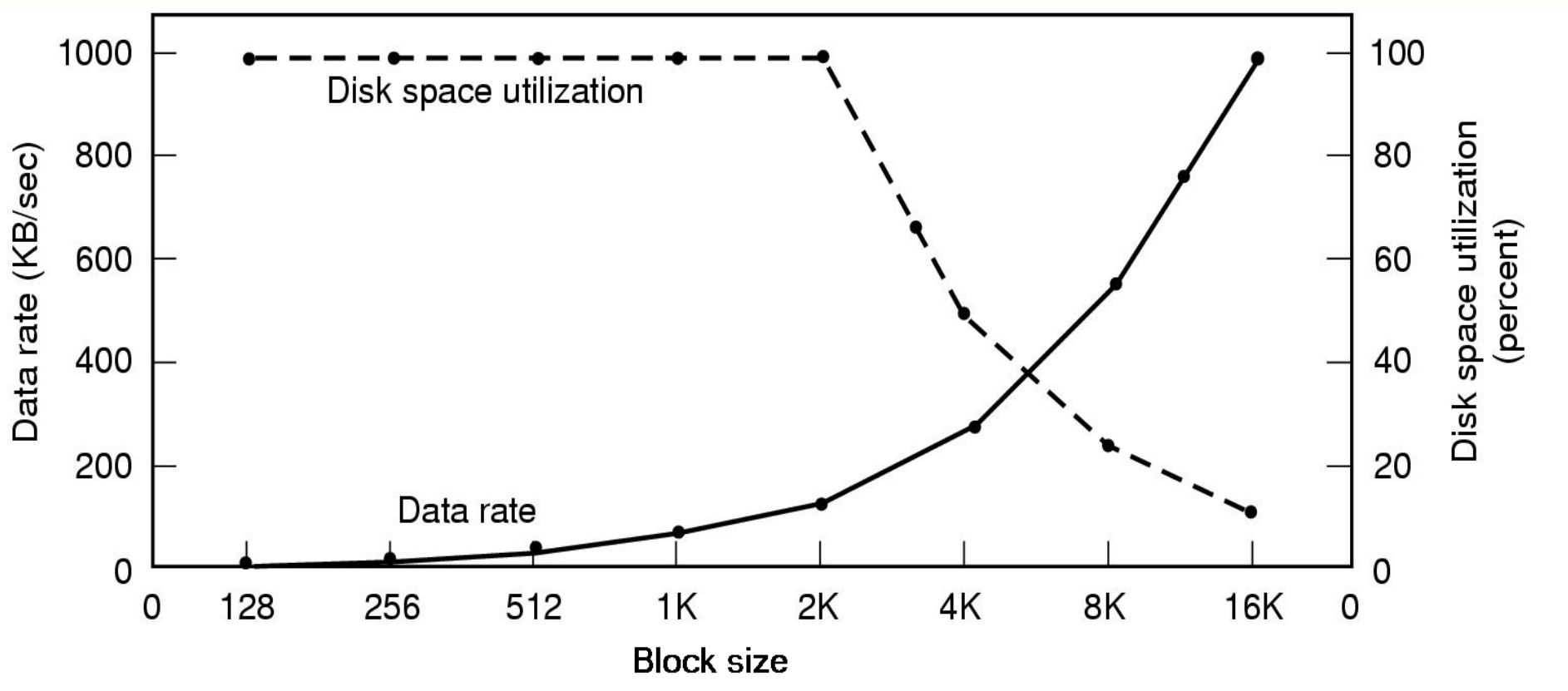
- V adresári je len meno súboru a číslo špeciálneho bloku tzv. **i-node** (index node), obsahujúce atribúty súboru a niekoľko čísiel blokov
- Keď nestačia čísla v jednom i-node, tak niektoré číslo bloku (na obrázku je to posledné) ukazuje na ďalší i-node

Táto myšlienka je základom súborových systémov ext (ext, ext2, ext3, ext4), ktoré sa bežne používajú v Linuxe.

Nespojité umiestnenie súborov Malé vs. veľké alokačné bloky (1)

- Malé alokačné bloky:
  - zmenšujú vnútornú fragmentáciu, teda zlepšujú využitie disku
  - zhoršujú rýchlosť čítania dlhších súborov
  - pri tej istej kapacite disku sú čísla blokov väčšie, a teda potrebujeme viac bitov na každé číslo bloku v dátových štruktúrach
- Veľké alokačné bloky
  - zvyšujú vnútornú fragmentáciu
  - zlepšujú rýchlosť čítania súborov
  - stačí menej bitov na číslo bloku v dátových štruktúrach

Nespojité umiestnenie súborov Malé vs. veľké alokačné bloky (2)



- Graf využitia disku a rýchlosti čítania pre rôzne veľkosti blokov a súbory veľkosti 2KB

Kódovanie znakov ASCII

- Pre uloženie písmen a iných znakov abecied do počítača musíme vymyslieť ich **zakódovanie** do čísiel.
- Kód je definovaný tabuľkou číslo – znak
- **Hodnota (Code point)** – číslo, ktoré jednoznačne kóduje nejaký znak.

USASCII code chart

b_7 b_6 b_5 → / Bits	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

http://en.wikipedia.org/wiki/File:ASCII_Code_Chart-Quick_ref_card.jpg

- **ASCII** (1963) – 7-bitový kód obsahujúci všetky anglické písmená, číslice, bežné interpunkčné znamienka ako aj 32 netlačiteľných riadiacich znakov (napr. CR, LF, FF, BS, BEL, TAB, ...)

8-bitové rozšírenia ASCII

- 7-bitový ASCII obsahuje len anglickú abecedu
- Novšie počítače majú pamäť organizovanú po bajtoch
- 7-bitový znak je uložený v 8-bitovom bajte
- Využime na kódovanie všetkých 8 bitov - **pridajme** do kódu napr. znaky s nadpunkciou, ale **zachovajme** pôvodné ASCII kódovanie pre kódy 0..127.
- Vznikajú navzájom nekompatibilné **rozšírenia** pre rôzne jazyky ale aj pre tie isté jazyky, napríklad:
  - Kód bratov Kamenických (Československo)
  - ISO/IEC 8859-2 (Stredná Európa)
  - Kódová stránka 852 pre MS-DOS (Stredná Európa)
  - Kódová stránka 1250 pre Windows (Stredná Európa)
  - Celá Európa potrebuje 6 kódových stránok pre Windows http://msdn.microsoft.com/en-us/library/cc195051.aspx

Unicode

- Veľa rôznych 8-bitových kódov pre rôzne jazyky od rôznych firiem a organizácií
- Niektorým jazykom ani nestačí 8-bitov na zakódovanie všetkých znakov – vznikli napr. kódy s premenlivou dĺžkou
- Riešenie: použiť viac bitov pre jeden znak
- Unicode (1991) – kódovanie znakov, ktoré má ambíciu obsiahnuť všetky znaky všetkých abecied
- Súčasný stav pridelených kódov http://unicode.org/charts/
- Používa **hodnoty** (anglicky **code point**), od 0 do 10FFFF – 17 „stránok“, 1 114 112 hodnôt.
- Hodnoty 0 až 127 sú zhodné s pôvodným ASCII

Zobrazenie Unicode hodnôt do bajtov

- **UTF-32** alebo **UCS-4** je priame zobrazenie každej hodnoty do 32-bitového (4-bajtového) čísla. Každý štvorbajt v súbore alebo v pamäti uchováva jednu Unicode hodnotu.
- Pretože pre európske jazyky je väčšina hodnôt menšia než 256, vzniká **dojem**, že sa **plýtva miestom**
- **UCS-2** používa len tie hodnoty, ktoré sa zmestia do 16 bitov a ukladá každú do jedného dvojbajtu.
  V súčasnosti sa považuje za zastaralý.
- **UTF-8** a **UTF-16** zobrazujú jednu hodnotu do postupnosti bajtov resp. dvojbajtov, ktorej dĺžka závisí od veľkosti hodnoty.
  Jeden bajt nie je jeden znak!
  Ani jeden dvojbajt nie je jeden znak!

UTF-8

Unicode hodnota	**UTF-8 sekvencia**
U+00000000 – U+0000007F	0xxxxxxx
U+00000080 – U+000007FF	110xxxxx 10xxxxxx
U+00000800 – U+0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+00010000 – U+001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U+00200000 – U+03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U+04000000 – U+7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

- Pre hodnoty 0 až 127 je rovnaký ako ASCII
- Používa sa na zobrazenie Unicode hodnôt v súboroch, na Webe, ...
- Unixovské operačné systémy ho používajú aj na kódovanie znakov v pamäti (Windows používa UTF-16 alebo 8-bitové kódy)
- Python používa UCS-2 alebo UCS-4 ako vnútornú reprezantáciu reťazcov (podľa toho, aké hodnoty obsahujú) a UTF-8 pre ukladanie programov
  Ale: pre čítanie textových súborov to nie je štandard.
  Ale 2: knižnica tk nedokáže zobraziť kódy nad FFFF

Práca s UTF-8 súbormi v Pythone

- Pre správne prečítanie/zápis súboru musíme použiť parameter encoding v open:
  `open('unicode.txt', encoding='utf-8')`
- Musíme pritom ešte dať pozor na prečítanie prvého znaku, ktorý môže (ale nemusí) byť BOM (\uFEFF)
- Znaky v reťazcoch môžeme zapisovať aj pomocou sekvencií \uXXXX, \UXXXXXXXX alebo \N{meno}.
- Modul unicodedata obsahuje databázu znakov, pomocou neho vieme zistiť, či je znak písmeno, cifra, aký je jeho popis v Unicode a pod.
- Ďalšie detaily:
  https://docs.python.org/3.9/howto/unicode.html

Vytváranie a rušenie súborov a adresárov, aktuálny adresár

Zistenie existencie

```
os.path.isfile(path)
os.path.isdir(path)
os.path.islink(path)
```

Vytváranie, rušenie, kopírovanie, prenášanie

```
os.remove(path, *, dir_fd=None) # súbor
os.rename(src, dst, *, src_dir_fd=None, dst_dir_fd=None)
  # dst môže byť celá cesta v takom prípade sa súbor prenesie (Move)
os.mkdir(path, mode=0o777, *, dir_fd=None)
os.makedirs(path, mode=0o777, exist_ok=False)
  # Vytvorí všetky adresáre tak, aby existovala cela zadaná cesta v path
os.rmdir(path, *, dir_fd=None)
os.removedirs(path)
shutil.copyfile(src, dst)
```

Aktuálny adresár (radšej nepoužívať a pracovať s kompletnými cestami)

```
os.getcwd()
os.chdir(path)
os.path.abspath(path)
  # ak je path relatívna cesta, tak výsledkom je úplná cesta
  #   (od aktuálneho adresára)
  # výsledok je vždy "normalizovaný", napr. neobsahuje . ani ..
```

Atribúty súborov

Čas a dátum súboru alebo adresára

```
os.path.getatime(path)    # čas posledného prístupu
os.path.getmtime(path)    # čas poslednej zmeny
os.path.getctime(path)    # čas vytvorenia
os.utime(path, times=None, *, ns=None, dir_fd=None, follow_symlinks=True)
        # nastaví čas posledného prístupu a čas modifikovania
        # times je dvojica (atime,mtime) v sekundách
        # ns je je dvojica (atime,mtime) v nanosekundách
        # ak je times aj ns None, tak sa nastaví súčasný čas
```

Čas je vyjadrený počtom sekúnd od začiatku „epoch“. Niekedy to môžu byť aj necelé čísla (teda aj zlomky sekúnd). Na prevody časov možno použiť funkcie modulov time a calendar:

```
time.ctime(secs)          # prevod času v sekundách na reťazec
time.gmtime(secs)         # prevod sekúnd na struct_time v UTC
time.localtime(secs)      # prevod sekúnd na struct_time v lokálnom čase
calendar.timegm(struct)   # prevod struct_time na sekundy v UTC
time.mktime(struct)       # prevod struct_time na sekundy v lokálnom čase
```

Veľkosť súboru

```
os.path.getsize(path)
```

Rozoberanie a skladanie mien súborov, normalizovanie a porovnávanie

```
os.path.sep                # oddelovač cesty - '\\' vo Windows, '/' v Unixe
os.path.join(path1, path2, ...)
os.path.split(path)        # (cesta, meno)
os.path.basename(path)     # len meno súboru
os.path.dirname(path)      # len cesta
os.path.splitdrive(path)   # (disk, zvyšok), v linuxe disk je "mount point"
os.path.splitext(path)     # (meno, prípona), splitext('a.txt') = ('a','.txt')
```

Pomôcky pri porovnávaní ciest:

```
os.path.normpath(path)     # normalizuje cestu tak, aby na nej neboli symboly
                           # odkazy na adresáre . A .. ani redundantné
                           # oddeľovače, vo Windows zmení aj / na \
os.path.normcase(path)     # normalizuje malé/veľké písmená tak, že vo Windows
                           # prevedie všetko na malé písmená a zmení / na \
                           # v Linuxe o v Mac OS nespraví nič (v Mac OS je
                           # preto väčšinou nepoužiteľná)
```

Prechádzanie stromom adresárov

```
os.listdir(path='.')
os.walk(top, topdown=True, onerror=None, followlinks=False)

def prechadzaj(cesta,level = 0):
    print(' '*2*level+os.path.basename(cesta))
    level+=1
    for fn in os.listdir(cesta):
        fpath = os.path.join(cesta,fn)
        if os.path.isfile(fpath):
            print(' '*2*level+fn)
        elif os.path.islink(fpath):
            print(' '*2*level+'--> ',fn)
        else:
            prechadzaj(fpath,level)


def prechadzaj2(cesta):
    toplevel = cesta.count(os.path.sep)
    for dirpath,subdirs,files in os.walk(cesta):
        level = dirpath.count(os.path.sep)-toplevel
        print(' '*level*2+os.path.basename(dirpath))
        for fn in files:
            print(' '*(level+1)*2+fn)
```