Virtuální souborový systém
(*Virtual File System, VFS*)

operační systém musí poskytovat prostředek pro perzistentní uložení dat a jejich správu

soubor kontejner pro data

souborový systém umožňuje organizaci souborů a přístup k nim

tradiční souborový sytém System V file system, s5fs, původně i v BSD systémech, byl součástí jádra

4.2 BSD Fast File System, FFS

souborový systém byl dále součástí monolitického jádra a ani s5fs a FFS nemohly koexistovat v jednom operačním systému

navíc cenná data můžou být uložena i v souborových systémech jiných operačních systému MS-DOS, později Windows, ...

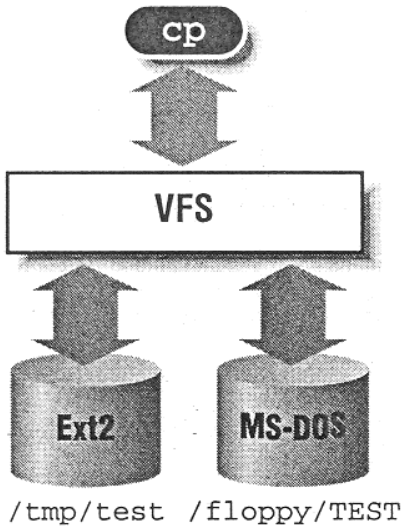
operace nad soubory, systémová volání, se "jenom" různě vykonávájí

VFS je vrstva jádra obsluhující všechna systémová volání pro souborový systém a poskytuje rozhraní současně pro různé souborové systémy (SunOS 1986)

příklad

`$ cp /floppy/TEST /tmp/test`

`/floppy` je bod začlenění diskety se souborovým systémem operačního systému MS-DOS



(a)

```
inf = open("/floppy/TEST", O_RDONLY, 0);
outf = open("/tmp/test",
           O_WRONLY|O_CREATE|O_TRUNC, 0600);
do {
    l = read(inf, buf, 4096);
    write(outf, buf, l);
} while (l);
close(outf);
close(inf);
```

(b)

souborové systémy podporované VFS můžeme rozdělit do třech skupin

diskové souborové systémy
- s5fs, FFS, Ext2(Linux)
- MS-DOS, Windows
- ISO9660 CD-ROM souborový systém
- ostatní

síťové souborové systémy

- Network File System, NFS (Sun)
- SMB (Microsoft)
- NCP, NetWare Core Protocol (Novell)

speciální souborové systémy

- nespravují diskový prostor, devfs (Linux)

Linux VFS

zavádí obecný souborový model schopný reprezentovat všechny podprorované souborové systémy

zrcadlí tradiční souborový systém operačního systému Unix s cílem minimální režie pro nativní souborový systém

jádro nemůže přímo obsahovat kód pro jednotlivé funkce jako je **`read()`**, namísto toho pro každou operaci použije ukazatel na imlementující funkci pro daný souborový systém

souborový systém je po otevření ve VFS reprezentován datovou strukturou **`file`**, která obsahuje položku **`f_op`**, která obsahuje ukazatel na funkce specifické pro konrétní typ souborového systému

pro operaci **`read()`** z příkladubude položka **`f_op`** údajové struktury **`file`** obsahovat ukazatel na funkce pro MS-DOS a volání funkce **`read`** je nepřímé

```
file->f_op->read(...);
```

pro operaci **`write()`** položka **`f_op`** údajové struktury **`file`** bude obsahovat ukazatel na funkce pro Ext2

na model můžeme nahlížet objektově

objekty jsou implementovány jako záznamy s položkami obsahující data a položkami obsahující ukazatele na funkce, odpovídající metodám objektu

obecný souborový model se skládá z objektů následujících typů

objekt superblok
uchovává globální informace o souborovém systému

objekt iuzel (vuzel)
uchovává informace o jednotlivém souboru, číslo iuzlu jednoznačně identifikuje soubor v souborovém systému

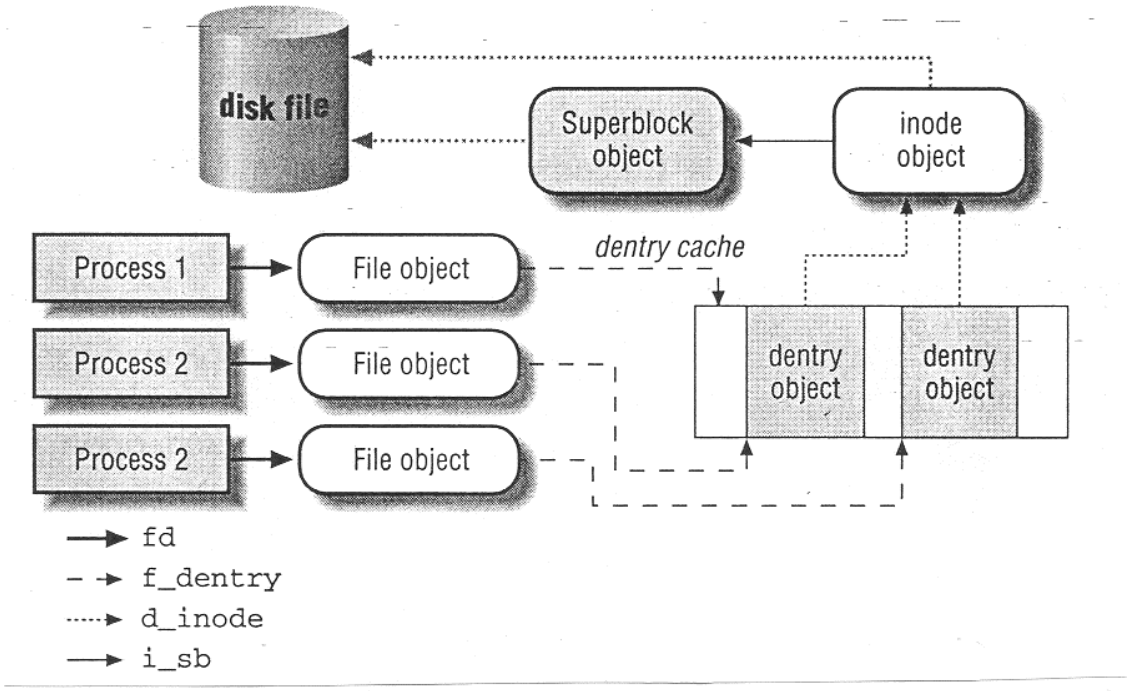
objekt soubor
uchovává informace o interakci mezi otevřeným souborem a procesem

objekt položka adresáře (*directory entry, dentry*)

uchováva odkaz na soubor odpovídající položce adresáře, uložení této informace na disku se pro jednotlivé typy souborových systémů liší

příklad

tři procesy otevřely tentýž soubor, dva z nich použily stejný odkaz (*hard link*)



VFS obsahuje mezipaměť nedávno použitých položek adresáře (*dentry cache*), urychluje převod cesty v adresáři na iuzel poslední součásti cesty

některá systémová volání nevyžadují volání specifických funkcí konkrétního systémového souboru

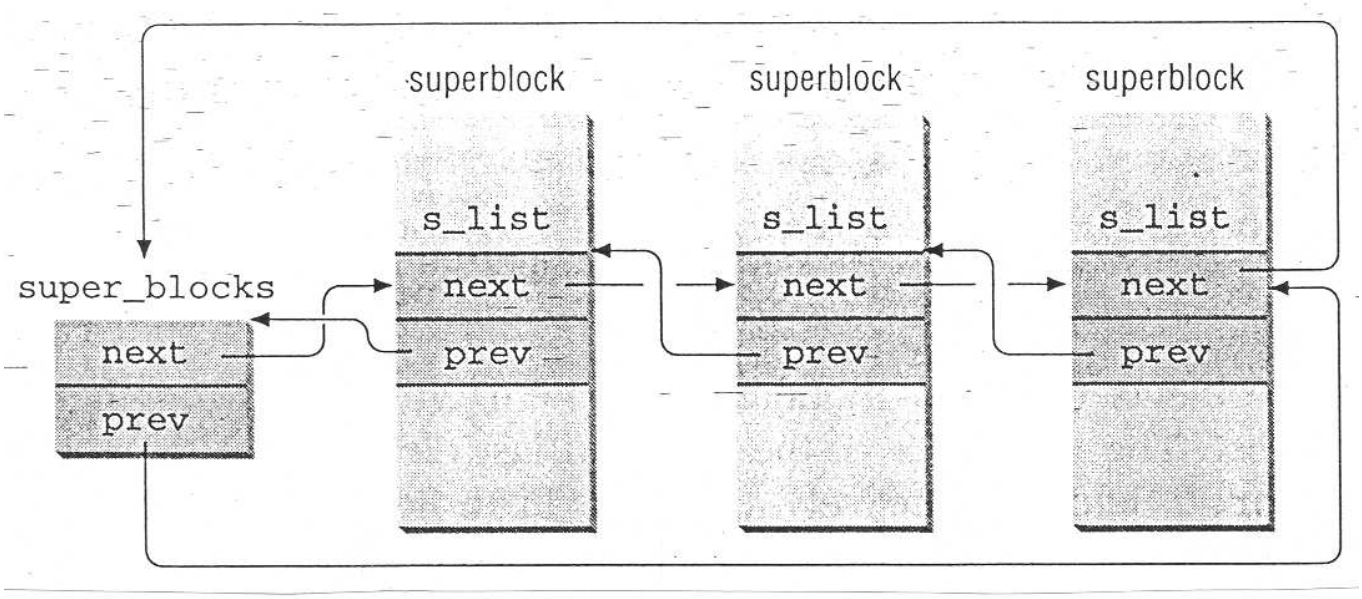
například **`lseek()`**, které nastavuje pozici v souboru pro další operaci, co je atribut, který se vztahuje k interakci otevřeného souboru a procesu, vyžaduje modifikaci jenom odpovídajícího objektu typu soubor a je tedy nezávislé na typu souborového systému

objekty superblok

záznam **`super_block`**

`s_list`	ukazatelé pro seznam superbloků
`s_blocksize`	velikost bloku v bytech
`...`	
`s_dirt`	příznak modifikování
`s_type`	typ souborového systému
`s_op`	metody superbloku
`...`	
`s_root`	položka adresáře adresáře začlenění
`...`	
`s_dirty`	seznam modifikovaných iuzlů
`u`	specifické informace pro souborový systém

všechny objekty superblok (jeden pro začleněný systém) jsou spojeny v obousměrném spojovém seznamu



údaje v položce **u**, například bitová mapa přidělených bloků, jsou kopírovány do paměti, jsou-li tyto údaje změněny superblok na disku se musí aktualizovat, co se zaznamená v příznaku **s_dirt**

metody objektu superblok jsou v záznamu **super_operations**, kterého adresa je v poli **s_op**

VFS potřebné operace, například přečtení iuzlu
`read_inode()`, volá

```
sb->s_op->read_inode();
```

příklady operací superbloku

`read_inode(inode)`
čte údaje z iuzlu na disku a vyplní položky objektu iuzel

`write_inode(inode)`
aktualizuje iuzel na disku z obsahu položek objektu iuzel

`delete_inode(inode)`
odstraní datové bloky obsahující soubor, diskový iuzel a VFS iuzel

`put_super(super)`
uvolní objekt superblok (odpovídající souborový systém je odčleněn)

`...`

v operacích se nenachází metoda **`read_super`** pro čtení superbloku z disku

uvedené metody jsou metody objektu superblok pro začleněný souborový systém

metoda **`read_super`** je metodou reprezentující typ souborového systému, která je volaná při začleňovámí souborového systému systém

objekty iuzel

všechny informace o souboru potřebné pro práci souborového systému se souborem jsou v iuzlu, speciálně jde o ukládání a vybírání informací uložených v souboru

jméno souboru je více nebo méně náhodné označení souboru

jedinečná reprezentace souboru je iuzel

v paměti má tvar záznamu **`inode`** s položkami

`i_hash`	ukazatelé na rozptýlený (*hash*) seznam iuzlů
`i_list`	ukazatelé na seznam iuzlů
`i_dentry`	ukazatelé na seznam položek adresáře
`i_ino`	číslo iuzlu
`i_count`	počítadlo použití
`i_mode`	typ souboru a přístupová práva
`i_nlink`	počet odkazů
`i_uid`	identifikátor vlastníka
`i_gid`	identifikátor skupiny
`i_size`	velokost souboru v bytech
`...`	

i_blksize	velikost bloku v bytech
...	
i_op	operace iuzlu
i_sb	ukazatel na objekt superblok
...	
i_state	příznak stavu objektu
...	
u	specifická informace pro typ suborového systému

pokud se změní hodnoty položek, které jsou v diskovém uzlu, hodnota položky **i_state** bude I_DIRTY

každý inode objekt je vždycky v jednom ze tří obousměrných spojových seznamů

- seznam nepoužívaných (volných) iuzlů
- seznam používaných iuzlů
- seznam modifikovaných iuzlů

vytvořených položkou **i_list**

objekty inode v seznamech používaných a modifikovaných iuzlů jsou také v rozptýlené tabulce

položka **i_op** obsahuje adresu záznamu **inode_operations** s metodami objektu iuzel

`create(dir, dentry, mode)`

vytvoří nový diskový iuzel pro obyčejný soubor sdružený s objektem položka adresáře

`lookup(dir, dentry)`

hledá v adresáři iuzel odpovídající jménu souboru v objektu položka adresáře

`link(old_dentry, dir, new_dentry)`

vytvoří nový odkaz na soubor specifikovaný parametrem **`old_dentry`** v adresáři **`dir`** ,

jméno nového odkazu je v **`new_dentry`**

`unlink(dir, dentry)`

`symlink(dir, dentry, symname)`

`mkdir(dir, dentry, mode)`

`rmdir(dir, dentry)`

`mknod(dir, dentry, mode, rdev)`

`...`

objekty soubor

objekt soubor opisuje interakci procesu s otevřeným souborem

je vytvořen když je soubor otevřen a vytváří ho záznam **`file`** s položkami

`f_next`	ukazatel na další objekt soubor
`f_pprev`	ukazatel na předházející objekt soubor
`f_dentry`	ukazatel na združený objekt položka adresáře
`f_op`	ukazatel na tabulku operací
`f_mode`	mód přístupu k souboru
`f_pos`	pozice v souboru
`f_count`	počet použití objektu
`...`	

objekty soubor nemají odpovídající obraz na disku a proto nemají položku, do které se zaznamená jejich modifikace

každý objekt soubor je na jednom ze dvou obosměrných kruhových seznamů

- seznam nepoužívaných objektů soubor, položka **`f_count`** je nulová
- seznam používaných objektů soubor

položka **`f_op`** obsahuje adresu záznamu s operacemi nad souborem

```
llseek(file, offset, whence)

read(file, buf, count, offset)
```

write(file, buf, cont, offset)

...

objekty položka adresáře

v modelu je adresář soubor obsahující seznam souborů a adresářů

po přečtení položky adresáře je transfomována na objekt položka adresáře

objekt položka adresáře se vytvoří pro každou část cesty v adresáři

pro /tmp/test se vytvoří tři objekty položka adresáře

objekt nemá odpovídající strukturu na disku, neobsahuje tedy položku pro zaznamenání modifikace

položky záznamu **dentry** jsou

d_count	počítadlo použití objektu
d_inode	iuzel sdružený se jménem souboru
d_parent	objekt položka adresáře rodiče
d_mounts	objekt položka adresáře pro kořen začleněného souborového systému

d_covers objekt položak adresáře pro bod začlenění souborového systému
d_hash ukazatelé na rozptýlený seznam
d_lru seznam nepoužitých objektů položka adresáře
d_alias ukazatelé pro seznam pro seznam objektů položka adresáře
združených s iuzlem
d_name jméno souboru
d_op metody objektu položka aresáře
d_sb objekt superblok souboru

objekty položka adresáře se uchovávají v mezipaměti objektů položka adresáře (*dentry cache*)

nepoužívané objekty položka adresáře mají v položce **d_count** hodnotu nula, přičemž **d_inode** ukazuje na združený iuzel a jsou pomocí položky **d_lru** uchovávány v obousměrném kruhovém seznamu

používané objekty položka adresáře združené s objektem iuzel jsou uchovávány v obousměrném seznam specifikovaném položkou **i_dentry** objektu iuzel pomocí položky **d_alias**

používaný objekt položka adresáře se stane negativní, když je zrušen poslední odkaz na odpovídající soubor a objekt položka adresáře se vloží do seznamu nepoužívaných

pro zrychlení přístupu k objektům položka adresáře se používá rozptýlená tabulka (*hash table*), přičemž hodnota rozptylové funkce se vytváří z adresy objektu položka adresáře adresáře a jména souboru, objekty položka adresáře se stejnou se stejnou hodnotou rozptylové funkce jsou v seznamu vytvořeném pomocí položky **`d_hash`**

metody objektu položka adresáře jsou v záznamu **`dentry_operation`**, kterého adresa je v položce **`d_op`**

metody jsou například

`d_hash(dentry, hash)`
vypočítá hodnotu rozptylové funlce
`d_compare(dir, name1, name2)`
porovná jméno **`name1`** v adresáři **`dir`** se jménem **`name2`**, pro MS-DOS 'xxx' = 'XXX'
`d_delete(dentry)`
`...`

soubory sdružené s procesem

každý proces má svůj pracovní adresář a kořenový adresář

adresa záznamu s touto informací je v položce **`fs`** deskriptoru procesu

```
struct fs_struct {
    atomic_t count;
```

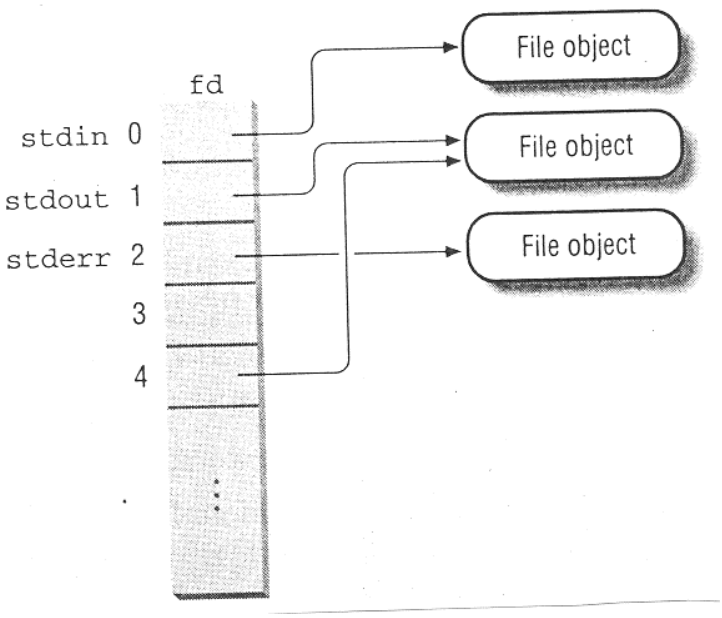
```
    int umask;
    struct dentry *root, *pwd;
};
```

položka **count** specifikuje počet procesů sdílejících záznam **fs_struct** a položka **umask** slouží k nastavení začátečních přístupových práv k nově vytvářeným souborům

v položce **files** deskriptoru procesu je adresa na záznam osahující položku **fd** co je ukazatel na pole ukazatelů na objekty soubor

pro každý soubor s prvkem v poli **fd** je index prvku deskriptor souboru

vzhledem např. na systémové volání dup() můžou dva deskriptory souboru odkazovat na stejný objekt soubor



připojení souborového systému

před začátkem používání souborového systému se musí vykonat dvě operace

- registrace
- připojení (začlenění)

registrace se vykoná buď při zavádění operačního systému (*boot*) nebo při zavedení modulu implementujícího souborový systém

po registraci souborového systému má jádro k dispozici jeho specifické funkce a souborový systém takového typu může být připojen

souborový systém, kterého kořenový adresář je kořenem systémového stromu adresářů se nazývá kořenový souborový systém

ostatní souborové systémy můžou být připojeny k systémovému stromu adresářů

adresáře, na které jsou připojeny souborové systémy se nazývají body připojení

registrace souborového systému

Linux je možné konfigurovat tak, aby rozeznával všechny potřebné typy souborových systémů při překladu jádra

kód implementující souborový systém může být zaveden také dynamicky jako modul

každý typ souborového systému je reprezentován objektem **`file_system_type`**, který má položky

`name`	jméno souborového systému
`fs_flags`	příznaky připojení, např. **`FS_REQUIRES_DEV`**
`read_super`	metoda pro čtení superbloku
`next`	ukazatel na následující prvek seznamu

připojení kořenového souborového systému

z (diskového) zařízení se pokouší přečíst superblok voláním metody **`read_super`** registrovaných objektů **`file_system_type`**

pro / vytvoří objekt iuzel a objekt položka adresáře

nastaví položky **`root`** a **`pwd`** procesu **`init`** na objekt položka adresáře /

vloží první prvek do seznamu připojených souborových systémů

připojení všeobecného (generického) souborového systému

standardní tvar příkazu pro připojení

`mount –t typ zařízení adresář`

odevzdá jádru fyzické zařízení, na kterém je souborový systém a jeho typ a adresář kam bude ve stromu adresářů připojen nový souborový systém

předcházející obsah tohoto adresáře (pokud nějaký byl) se stane neviditelný dokud nový souborový systém zůstane připojen

příklad

`mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom`

jádro prohledá seznam registrovaných typů souborových systémů a získá ukazatel na objekt **`file_system_type`** s požadovaným typem souborového systému

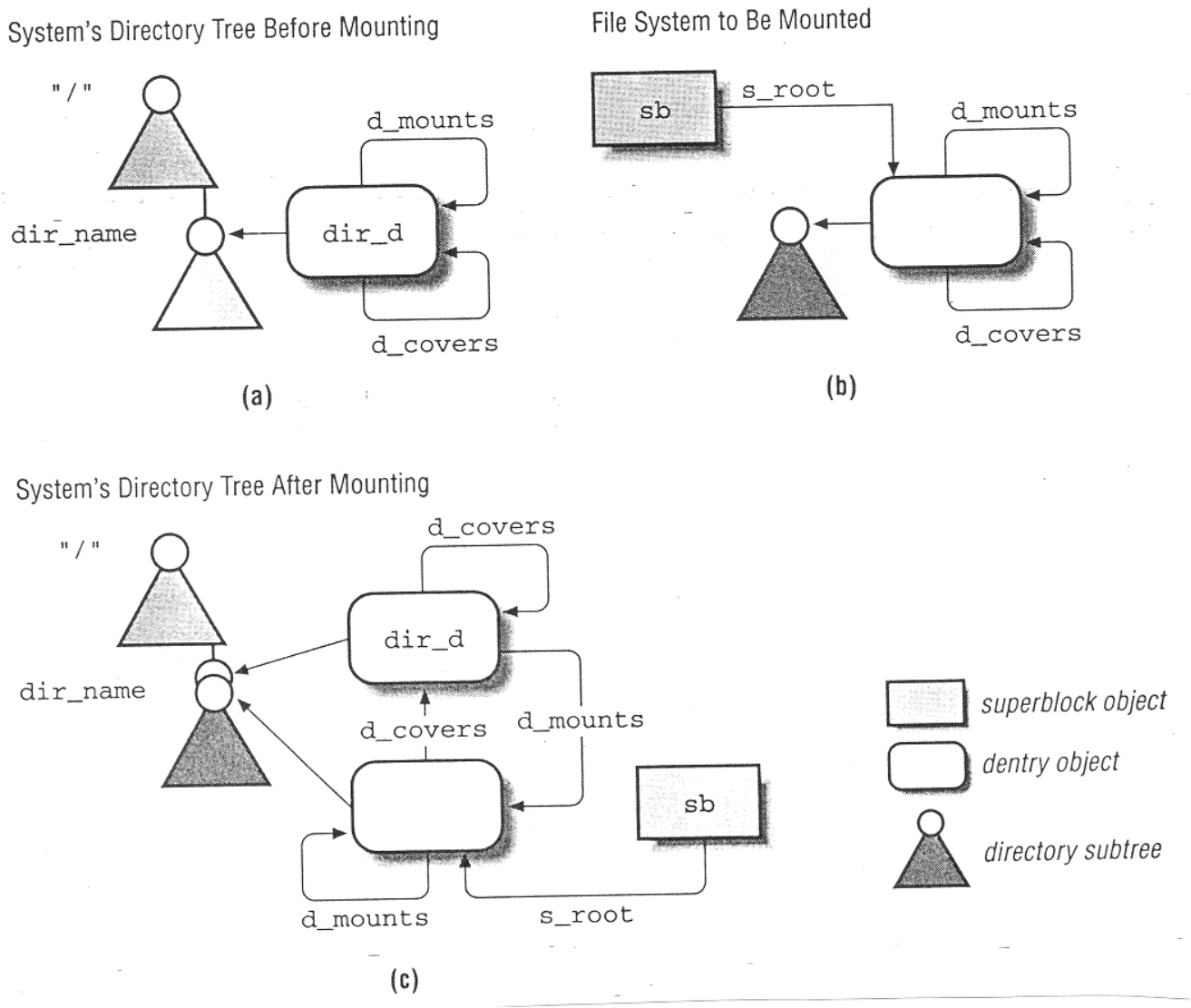
nalezne nebo vytvoří objekt **`dir_d`** položka adresáře pro specifikovaný adresář (a)

zavolá metodu **`read_super`** na získání objektu superblok nového souborového systému, přičemž položka **`s_root`** objektu superblok ukazuje na objekt položka adresáře souborového systému, který má být připojen (b)

přidá další prvek do seznamu připojených souborových systémů

položku **`d_mounts`** objektu **`dir_d`** nastaví na hodnotu položky **`s_root`** , tj. kořenový adresář připojovaného souborového systému

položku **`d_covers`** objektu položka adresáře kořenového adresáře připojovaného souborového systému nastaví na objekt **`dir_d`** (c)



odpojení souborového systému

postup je v zásadě opačný
nelze odpojit souborový systém, kterého soubory jsou používány
nelze odpojit kořenový systém souborů
změněné objekty se zapíšou na disk

prohledávání cesty k souboru

cílem je, aby VFS ze zadané cesty k souboru určil odpovídající iuzel

cesta se rozdělí na posloupnost jmen souborů, které všechny, kromě posledního, musí být adresáře

je-li začáteční znak /, cesta je absolutní a prohledávání začne adresářem

`běžící->fs->root`

jinak prohledávání začne v adresáři

`běžící->fs->pwd`

následně se hledá v adresáři položka s prvním jménem v cestě, čím se získá iuzel prvního adresáře v cestě

postup se opakuje až projdeme celou cestu

celý naznačený postup značně urychluje mezipaměť objektů položka adresáře

přitom nutno vzít v úvahu

- přístupová páva pro každý adresář
- jméno může být symbolický odkaz a postup musí pokračovat pro všechny části cesty v symbolickém odkazu
- vznikne-li symbolickými odkazy kruh, musí být identifikován a prohledávání skončit chybou
- jméno může být bod připojení souborového systému a prohledávání musí pokračovat v novém souborovém systému

zamykání souborů (file locking)

operační sytém UNIX byl navržen se souběžným přístupem k souborům více procesy

obdobně jako u sdílených proměnných vzniká problém synchronizace

POSIX požaduje mechanizmus zamykání umožňující zamknout libovolnou část souboru – od jednoho bytu až celý soubor

jelikož je možno zamykat soubor po částech proces může vlastnit několik zámků

pokud je nějaká část souboru zamknuta a jiný proces nekontroluje její zamčení může jiný proces k zamčené části přistoupit

takovéto zámky se nazývají poradní (*advisory locks*) a vyžadují spolupráci procesů

jsou implementovány na bázi systémového volání **`fcntl()`**

tradiční varianty BSD implementují poradní zámky systémovým voláním **`flock()`**, které však neumožňuje zamykání častí souboru, ale jenom celý soubor

tradiční varianty Systému V poskytují systémové volání **`lockf()`**, co je jenom rozhraní k **`fcntl()`**

navíc System V R3 zavedl mandatorní (*mandatory*) zámky, kdy jádro kontroluje zamčení souboru při každém volání **`open(), read(), write()`**

soubor je označen pro mandatorní zmaykání nastavením bitu **`SGID`** a nulováním oprávnění x pro skupinu, co jinak nedává smysl

mandatorní zámky jsou aktivovány a deaktivovány pro souborový systém příznakem **`MS_MANDLOCK`** při připojení souborového systému systému

když je příznak **`MS_MANDLOCK`** nastaven, **`flock()`** vytváří poradní zámky a **`fcntl()/lockf()`** vytváří zámky mandatorní

když příznak **`MS_MANDLOCK`** není nastaven obě systémová volání vytváří poradní zámky

bez ohledu na to, jestli procesy používají poradní nebo mandatorní zámky mohou využívat

- sdílené (*shared*) zámky pro čtení
- výhradní (*exclusive*) zámky pro psaní

libovolný počet procesů může vlastnit sdílené zámky, ale jenom jeden proces může vlastnit výhradní zámky

okamžitý zámek	požadovaný zámek	
	čtení	zápis
žádný	ano	ano
čtení	ano	ne
psaní	ne	ne