Struktura jądra UNIX

Wykład 15: Wirtualny system plików (NetBSD)

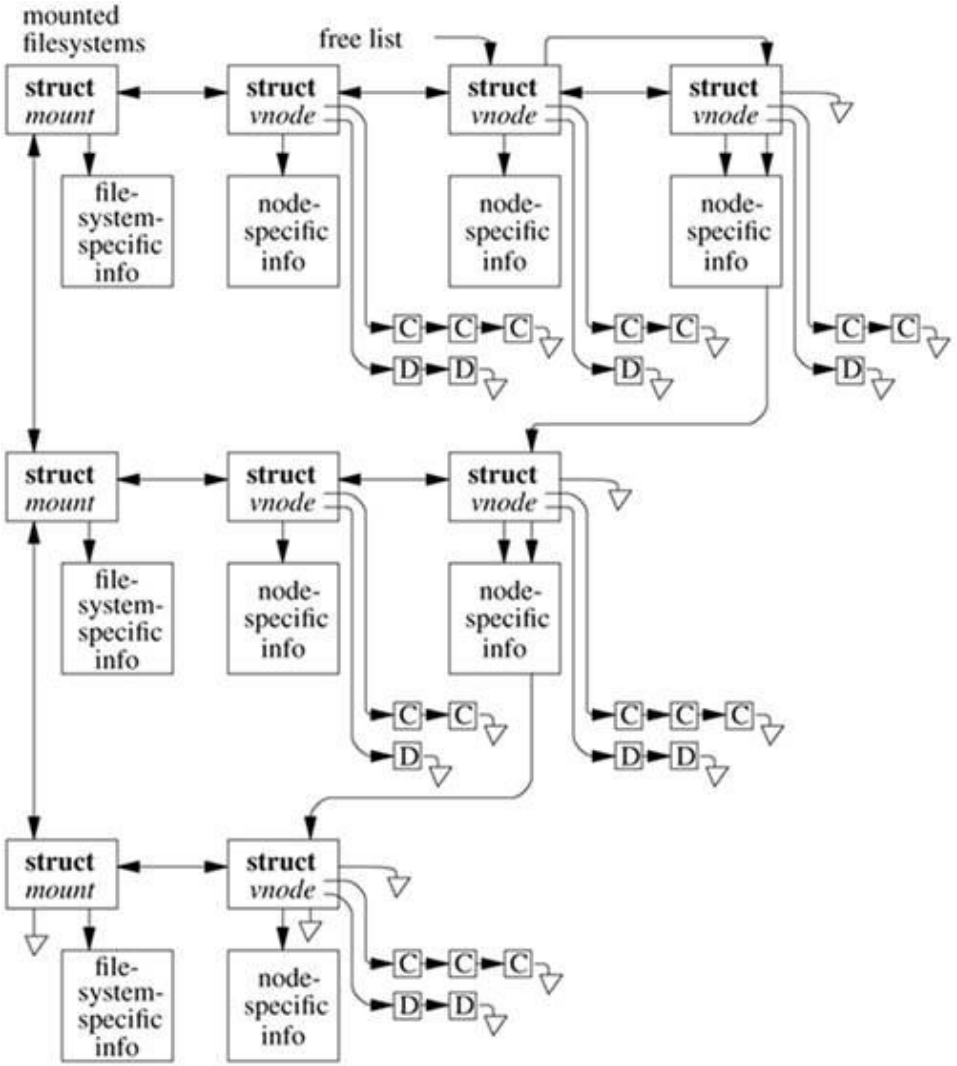
VFS: struktury danych jądra

`mount` reprezentuje zamontowany system plików

- lista v-węzłów z tego sys. plików
- przesłonięty v-węzeł
- wskaźnik na `vfsops`
- prywatne dane systemu plików

`vnode` reprezentuje obiekt sys. plików (plik, katalog, …)

- lista czystych i brudnych bloków
- liczniki referencji
- typ pliku (`VDIR`, `VREG`, `VLNK`, ...)
- wskaźnik na `vnodeops`
- odpowiadający `vm_object`



Składowe vnode(9)

v_usecount	liczba aktywnych użytkowników, zarządzane vref(9), vrele(9), vput(9)
v_holdcnt	liczba użytkowników zależnych od istnienia tego v-węzła, zarządzane: vhold(9), holdrele(9)
v_writecount	liczba użytkowników posiadająca plik do zapisu, zarządzane przez kod obsługujący open / close
v_data	skojarzone dane należącego do konkretnego FS
v_*blkhd	listy bloków dyskowych
v_uobj	obiekt wspierający pamięci wirtualnej

v-węzły vnode(9)

Każdy buf unikatowo identyfikowany przez adres v-node i offset względem początku pliku.

Pamięć podręczna vcache przechowuje nieużywane v-węzły. Część z nich posiada bufory, które zostaną zsynchronizowane z dyskiem przez program syncer i odpięte przez program pager. Żeby v-węzeł użyć z innymi system plików najpierw trzeba go wypiąć ze struktur danych bieżącego FS → VOP_RECLAIM.

Lista zarządzana zgodnie z polityką LRU, jeśli jest na niej zbyt dużo elementów to zostaje wybudzony vdrain_thread.

Flagi vnode(9)

Stan przechowywany w ***v_iflag*** i ***v_vflag***, zmienany przy założonych blokadach systemu plików albo ***v_interlock***.

VV_ROOT v-węzeł jest korzeniem swojego FS

VV_SYSTEM używane przez jądro (np. plik quota(1))

VI_EXECMAP v-węzeł ma wykonywalne odwzorowania (***execve***)

VI_ONWORKLST w trakcie przetwarzania przez demon syncer

VI_CLEAN v-węzeł został oswobodzony, żaden system plików już się do niego nie odnosi

vnodeops: operacje na v-węzłach

rozwiązywanie ścieżek	VOP_LOOKUP, VOP_ACCESS
tworzenie, usuwanie i zmiana nazwy obiektów	VOP_CREATE, VOP_MKNOD, VOP_LINK, VOP_SYMLINK, VOP_MKDIR VOP_REMOVE, VOP_RENAME, VOP_RMDIR
obsługa atrybutów	VOP_GETATTR, VOP_SETATTR
interpretacja obiektów	VOP_OPEN, VOP_READ, VOP_WRITE, VOP_FSYNC, VOP_SEEK, VOP_CLOSE VOP_READDIR, VOP_READLINK, VOP_FALLOCATE, VOP_FDISCARD
zarządzanie obiektami	VOP_LOCK, VOP_UNLOCK, VOP_INACTIVE, VOP_RECLAIM, VOP_REVOKE
interakcja z procesami	VOP_ADVLOCK, VOP_FCNTL, VOP_POLL, VOP_IOCTL, VOP_KQFILTER
zarządzanie buforami	VOP_BWRITE, VOP_STRATEGY
odwzorowania i pager	VOP_MMAP, VOP_GETPAGES, VOP_PUTPAGES

Gdzie są istotne różnice między tym a interfejsem wywołań systemowych?

Brak pojęcia kursora i otwartego pliku!

vattr(9) atrybuty v-węzła

Struktura zawiera atrybuty zwracane przez stat(2).

Przechowuje wynik tłumaczenia atrybutów systemu plików, który ma bogatszy albo skromniejszy zestaw niż uniksowe FS.

```
mode_t          va_mode;        /* files access mode and type */
nlink_t         va_nlink;       /* number of references to file */
uid_t           va_uid;         /* owner user id */
gid_t           va_gid;         /* owner group id */
u_quad_t        va_size;        /* file size in bytes */
long            va_blocksize;   /* blocksize preferred for i/o */
struct timespec va_atime;       /* time of last access */
struct timespec va_mtime;       /* time of last modification */
struct timespec va_ctime;       /* time file changed */
u_quad_t        va_bytes;       /* bytes of disk space held by file */
...
```

Ścieżki i rozwiązywanie nazw

Rozwiązywanie nazw to proces odwzorowania ścieżki na v-węzeł. Ścieżka składa się z **komponentów** i znaków **separatora** “/”. Ścieżka **względna** zaczyna się w katalogu roboczym procesu (`chdir`, `getcwd`). Ścieżka **bezwzględna** zaczyna się w katalogu głównym. Jeśli nie zawiera “`.`”, “`..`” i dowiązań symbolicznych to jest dodatkowo **znormalizowana**.

Rozwiązywanie nazw jest przeprowadzana przez moduł `namei(9)`. Proces jest kosztowny, więc wprowadzono pamięć podręczną `namecache(9)`. W trakcie rozwiązywania ścieżek VFS sprawdza także **uprawnienia dostępu** na podstawie `kauth(9)`.

Przygotowanie struktury ***nameidata***

NDINIT(*struct nameidata *ndp, u_long op, u_long flags,*
*struct pathbuf *pathbuf*);

Inicjuje stan struktury **nameidata**, w tym:

```
struct componentname {
  uint32_t       cn_nameiop;  // namei operation
  uint32_t       cn_flags;    // flags to namei
  kauth_cred_t   cn_cred;     // credentials
  const char *   cn_nameptr;  // pointer to looked up name
  size_t         cn_namelen;  // length of looked up component
  size_t         cn_consume;  // chars to consume in lookup
};
```

Tryby pracy ***namei***

Każda z operacji ma za zadanie znaleźć węzeł i udzielić systemowi plików wskazówek co do przyszłych operacji.

- **LOOKUP** dla stat(2) i open(2) bez **O_CREATE**
- **CREATE** dla mkdir(2) i open(2) z **O_CREATE**
- **DELETE** dla unlink(2) i rmdir(2)
- **RENAME** dla rename(2)

W trakcie wyszukiwania sprawdzane są uprawnienia na podstawie ***cn_cred*** i modyfikowana pamięć podręczną namecache(9).

Interakcja z systemem plików przebiega przy pomocy: VOP_LOOKUP(9), VOP_ACCESS(9) i VOP_READLINK(9).

namei(9) rozwiązywanie nazw

1. Przydziel miejsce na stan algorytmu ***nd*** typu ***nameidata***.
2. Zainicjuj stan ***nd*** przy pomocy ***NDINIT()*** specyfikując parametry działania algorytmu rozwiązywania nazw.
3. Wywołaj ***namei()*** i obsłuż potencjalny błąd (`res != 0`).
4. Odczytaj znaleziony v-węzeł z ***nd.ni_vp***. Jeśli flagi zawierały ***LOCKPARENT***, to odczytaj v-węzeł katalogu z ***nd.ni_dvp***.
5. Dla operacji na katalogu (zakładanie pliku), użyj obiektu typu ***componentname*** zapisanego w ***nd.ni_cnd***.

Flagi operacji ***namei***

FOLLOW Trawersuj dowiązania symboliczne.

LOCKLEAF Załóż blokadę na znaleziony v-węzeł.

LOCKPARENT Załóż blokadę na katalog zawierający v-węzeł.

NOCACHE Nie twórz wpisów w p. podręcznej nazw.

NOCROSSMOUNT Nie przekraczaj bariery systemów plików.

ISWHITEOUT Czy dany v-węzeł jest wpisem przesłaniającym.

Wpisy przesłaniające służą do kasowania plików i katalogów w hybrydowych systemach plików `mount_union(8)`.

namecache(9): pamięć podręczna nazw

Pozwala ***namei*** na przyspieszenie przetwarzania zapytań.

Przechowuje rekordy ***namecache*** w tablicy mieszającej list utrzymywanych zgodnie z polityką LRU.

`{(dir-vnode, filename) → [v-node|NULL]}`

Wpis może być w jednym z następujących stanów:

- ***active***: utworzony ***cache_lookup*** lub ***cache_revlookup***
- ***queued***: unieważniony ***cache_invalidate***
- ***nonexistent***: zwolniony ***cache_reclaim***

Operacje na pamięci podręcznej nazw

Trafienia w p. podręczną mogą być pozytywne (`v-węzeł`) lub negatywne (`NULL`). Istnieją wpisy przesłaniające (`whiteout`).

Wpisy negatywne przydają się dla powłoki uniksowej, która odpytuje katalogi w zmiennej ***`PATH`*** w poszukiwaniu pliku wykonywalnego danego polecenia.

Zapytanie o nazwę w katalogu ***`cache_lookup`***, o katalog zawierający dany v-węzeł ***`cache_revlookup`***.

Dodawanie wpisów ***`cache_enter`***. Usuwanie wpisów ***`cache_purge`*** skojarzonych z v-węzłem, p. montażowym ***`cache_purgevfs`***.

vfsops: operacje na punktach montażowych

Wykorzystywane przez algorytm rozwiązywania ścieżek, implementację wywołań systemowych (do celów administracyjnych).

VFS_ROOT pobierz v-węzeł głównego katalogu systemu plików

VFS_VGET pobierz v-węzeł o zadanym numerze

VFS_MOUNT zamontuj system plików w danym punkcie uwzględniając opcje specyficzne dla systemu plików

VFS_UNMOUNT demontuje system plików

VFS_STATFS pobiera metadane i statystyki (statfs)

VFS_SYNC uspójnia brudne bufory skojarzonych v-węzłów (sync)

VFS_QUOTACTL operacje na limitach systemu plików (quotactl)

Algorytm rozwiązywania nazw

1. Kopiujemy ścieżkę do wewnętrznego bufora.
2. Wyznaczamy punkt startowy, tj. v-węzeł katalogu głównego (***`VFS_ROOT`***) lub bieżącego (***`p->p_cwdi`***).
3. Dla każdego komponentu ścieżki:
   a. sprawdź uprawnienia do katalogu,
   b. zapytaj system plików o komponent ***`VOP_LOOKUP`***,
   c. jeśli znaleziony v-węzeł to dowiązanie symboliczne, to zmodyfikuj wyszukiwaną ścieżkę i ponów operację (3),
   d. ostatni komponent → sprawdź uprawnienia i zakończ (3),
   e. v-węzeł nie jest katalogiem → zakończ z ***`ENOTDIR`***,
   f. sprawdź czy katalog jest punktem montażowym ***`vu_mountedhere`***.

Pytanie: Jak zakładać blokady, żeby uniknąć zakleszczeń?

Implementacja v-węzłów

Każdy v-węzeł zawarty w dodatkowej strukturze `vnode_impl`, która przechowuje stan i węzły struktur:

1. pamięć podręczna nazw (***`vi_*nclist`***),
2. bufory v-węzłów (***`vi_hash`***) i wartość skrótu (***`vi_key`***),
3. lista brudnych v-węzłów (***`vi_synclist`***),
4. v-węzły punktu montażowego (***`vi_mntvnodes`***),
5. lista LRU do której należy (***`vi_lrulist`***, ***`vi_lrulisthd`***)

`lru_vrele_list` oczekujące na asynchroniczne zwolnienie
`lru_free_list` v-węzły czyste bez buforów i referencji
`lru_hold_list` v-węzły z ***`v_holdcnt > 0`***

Buforowanie v-węzłów

W NetBSD v-węzły przechowywane w tablicy mieszającej list ***vcache_hashtab***. Skrót obliczany z adresu p. montażowego ***mp*** i numeru ***i-node***. V-węzły pobieramy ***vcache_{get,new}***.

vrele: zwalnia v-węzeł, wkłada na listę ***free*** / ***hold***

VOP_RECLAIM zwalnia z v-węzła informacje specyficzne dla danego systemu plików, pozwala podpiąć pod inny FS

Usunięcie systemu plików bez odmontowania

Należy najpierw wstrzymać operacje na systemie plików *vfs_suspend(9)*. Wszystkie v-węzły należące do punktu montażowego oznaczyć jako martwe *vgone(9)*.

vgone wewnętrznie woła ***VOP_RECLAIM***, które podczepia v-węzeł pod punkt montażowy systemu plików deadfs. Zmienia wskaźnik na ***vnodeops*** na ***dead_nodeop_entries***.

Dzięki temu operacje na pliku zwracają wartości ***errno***, albo takie jakby plik miał długość 0.

Podobnej techniki można użyć, gdy ponownie montujemy system plików z flagą *readonly*.

Buforowanie zawartości v-węzłów

VOP_INACTIVE informuje system plików, że plik nie jest już używany i należy wypisać na dysk wewnętrzne bufory

vflush	wywołanie *sync(2)* wymaga wyczyszczenia wszystkich buforów v-węzłów systemu plików
vflushbuf	*fsync(2)* wypisz na dysk brudne bufory
vtruncbuf	*ftruncate(2)* zapomnij bufory leżące za nowym końcem pliku

Po skasowaniu pliku można od razu unieważnić wszystkie jego bufory i anulować operacje zapisu.

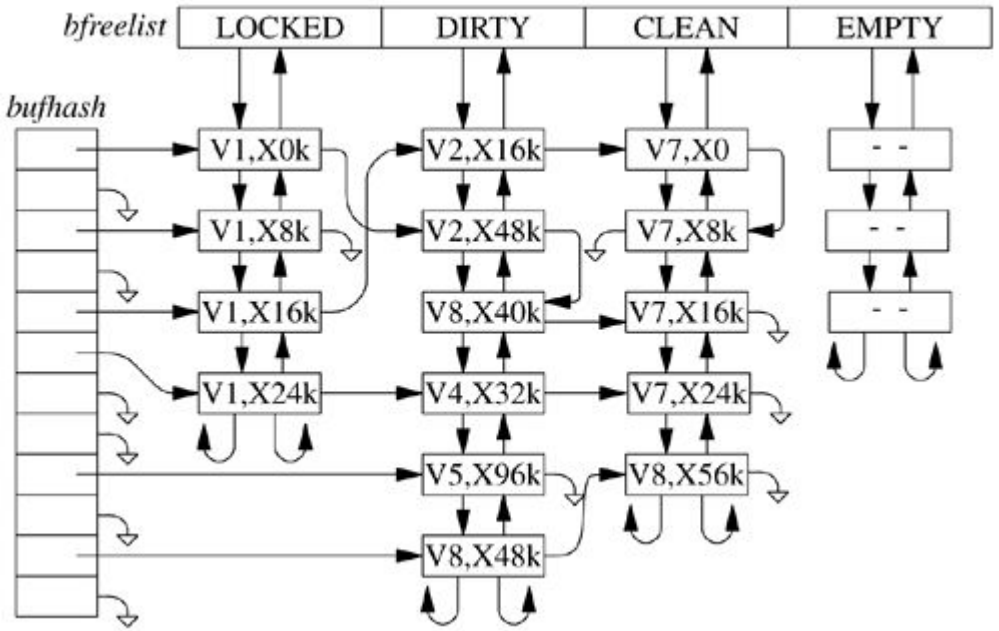
Buforowanie bloków dyskowych

Niegdyś system posiadał osobne bufory dla stron procesów oraz bloków dyskowych (tj. plików). Problemy z implementacją `mmap(2)`. Strony i bloki dyskowe są traktowane tak samo → page cache.

Q: Jak wyznaczyć położenie strony należącej do pliku?
A: Potrzebujemy identyfikator niezależny od systemu plików (vnode) i pozycję strony w pliku.

`bufhash` kubełki adresowane parą (**`vnode`**, **`offset`**)

`LOCKED` → na zawartości wykonywane operacje wej.-wyj.



buffercache(9): zarządzanie buforami (1)

Poniższe funkcje wykorzystują VOP_STRATEGY i VOP_BWRITE.

bread: tworzy, wczytuje bufor i oznacza go jako zajęty; wątki, które chcą dostępu do tego samego bloku muszą poczekać

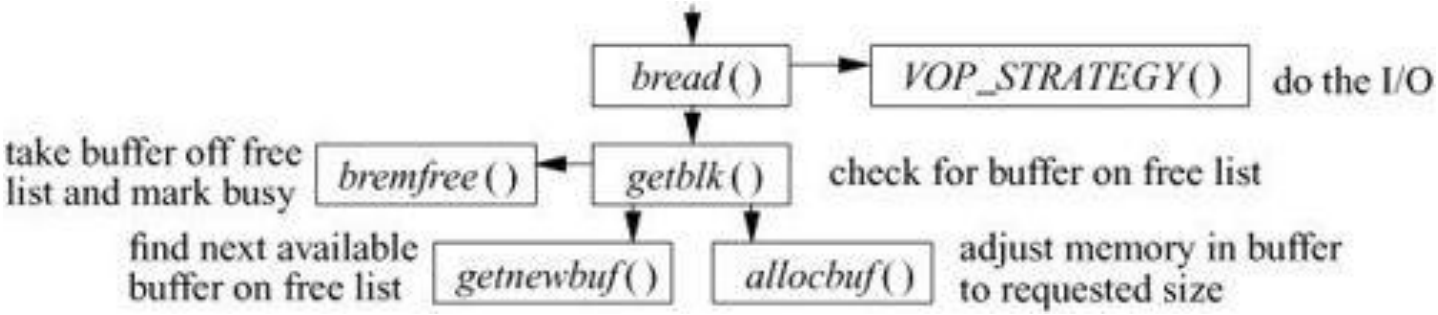
bdwrite: oznacza bufor jako brudny i odblokowuje, za jakiś czas przyjdzie syncer i go zapisze

bawrite: zapis asynchroniczny

bwrite: zapis synchroniczny

brelse: zwalnia blokadę i wkłada czysty bufor na kolejkę FREE, jeśli nie ma oczekujących wątków

buffercache(9): zarządzanie buforami (2)



Pytania?