Systemy plików UNIX-a

Zespół w składzie:
Urszula Herman-Iżycka,
Paweł Kępska,
Elżbieta K rępska,
Piotr Witusowski

19 stycznia 2004

Spis treści

Rozdział 1

System plików

1.1 Czym jest system plików

UNIXowy system plików to zbiór plików i katalogów, który mą następujące własności:

- Ma główny katalog (/), w którym znajdują się pozostałe pliki i katalogi. Większość dyskowych systemów plików posiada również katalog `lost+found`, gdzie przechowywane są pliki odzyskane po załamaniu się systemu, a którym nie udało odnaleźć się rodziców.
- Każdy plik lub katalog jest jednoznacznie identyfikowany przez: nazwę, katalog, w którym się znajduje oraz jednoznaczny identyfikator zwykle nazywany `inode'm`.
- Zgodnie z konwencją `inode` głównego katalogu ma numer 2, a `lost+found` 3. `Inode`'y o numerach 0 i 1 są nie używane. Informacje o numerach `inode`'ów moża uzyskać za pomocą polecenia `ls -i`.

```
[witus@222-mo3-2 usr]# ls -i
     44 bin
    596 etc
    597 games
    598 include
  89715 java
      8 lib
    602 local
    103 sbin
    635 src
    636 tmp
    590 X11R6
```

- Jest zamknięty: nie ma żadnych zależności pomiędzy nim a jakimkolwiek innym systemem plików.

1.2 Tworzenie nowych systemów plików

Nowe systemy plików mogą być tworzone w partycjach lub dyskach logicznych. Najczęściej używanym poleceniem do tworzenia nowych dysków logicznych jest `mkfs`, chociaż na niektórych platformach dostępne jest polecenie `newfs` o bardziej przyjaznym interfejsie. Typ tworzonego systemu plików przekazywany jest jako argument polecenia `mkfs`. Na przykład aby stworzyć system plików *VxFS* na większości systemów UNIXowych należy wykonać `mkfs -F vxfs`. W Linuxie to polecenie będzie wyglądało tak: `mkfs -t vxfs`.

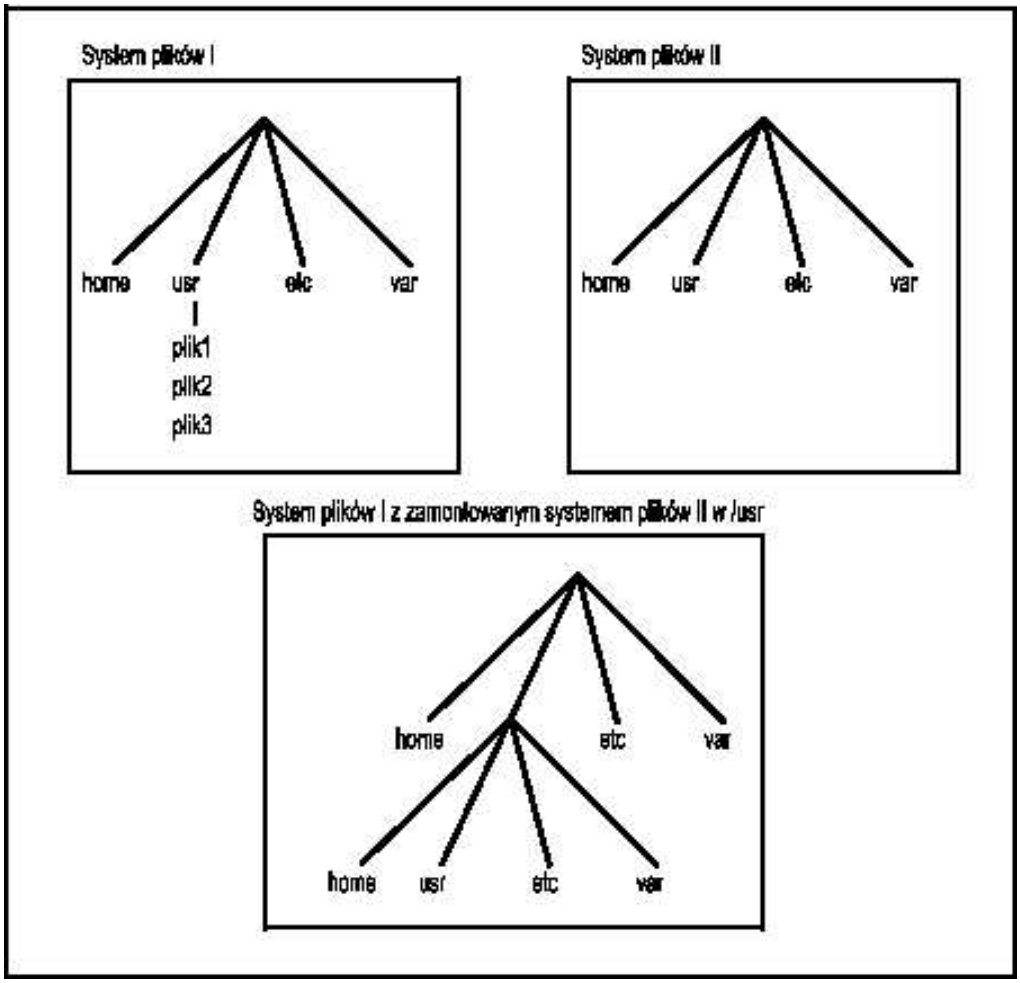
```
[root@222-mo3-2 fs]# mkfs -t ext3 /dev/ram14 10000
mke2fs 1.32 (09-Nov-2002)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
2512 inodes, 10000 blocks
500 blocks (5.00%) reserved for the super user
First data block=1
2 block groups
8192 blocks per group, 8192 fragments per group
1256 inodes per group
Superblock backups stored on blocks:
        8193
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.

[root@222-mo3-2 fs]# mkfs -t ext3 /dev/ram14
mke2fs 1.32 (09-Nov-2002)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
8000 inodes, 32000 blocks
1600 blocks (5.00%) reserved for the super user
First data block=1
4 block groups
8192 blocks per group, 8192 fragments per group
2000 inodes per group
Superblock backups stored on blocks:
        8193, 24577
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

Zależnie od rodzaju tworzonego systemu plików czas trwania operacji może się bardzo zmieniać. Polecenie wykonane w przykładzie zakończyło się natychmiast, ale stworzenie na przykład systemu plików `UFS` o rozmiarze 25GB może zająć 15-25 minut.

1.3 Montowanie i odmontowywanie systemów plików

System operacyjny montuje swój główny system plików (root) rozruchu. Każdy inny system plików może zostać zamontowany w jakimkolwiek katalogu głównego systemu plików, poza /. Kiedy montowanie następuje w niepustym katalogu, jego zawartość jest ukrywana. Pliki i katalogi są niewidoczne dopóki jest tam zamontowany system plików.



Montowanie w niepustym katalogu.

Gdy chcemy zamontować system plików musimy podać poleceniu `mount` jego typ, urządzenie, na którym on się znajduje i punkt montowania.

```
[root@222-mo3-2 fs]# mount -t vfat /dev/hda1 /mnt/win_c/
[root@222-mo3-2 fs]# mount | grep win_c
/dev/hda1 on /mnt/win_c type vfat (rw)
```

Kiedy system plików jest zamontowany informacja o nim jest dodawana do pliku `/etc/mnttab` (w Linuxie `/etc/mtab`).

```
[root@222-mo3-2 fs]# cat /etc/mtab
/dev/hdb1 / reiserfs rw,notail 0 0
none /proc proc rw 0 0
none /proc/bus/usb usbdevfs rw 0 0
none /dev devfs rw 0 0
none /dev/pts devpts rw,mode=0620 0 0
/dev/hdb6 /home reiserfs rw,notail 0 0
/dev/hdb7 /usr reiserfs rw,notail 0 0
/dev/hda1 /mnt/win_c vfat rw 0 0
```

Wszystkie wersje UNIXa mają funkcje umożliwiające manipulację na tablicy montowania. Najczęściej są to dwie funkcje, których nagłówki znajdują sie w `<sys/mnttab.h>`:

1. `int getmnttent(FILE *fp, struct mnttab *mp);`

2. `int putmnttent(FILE *iop, struct mnttab *mp);`

Jak łatwo się domyślić pierwsza służy do odczytywania z tablicy montowania, a druga do zapisywania. Struktura `mnttab` ma następujące składowe:

```
char *mnt_special;     /* Urządzenie, na którym znajduje się system plików */
char *mnt_mountp;      /* Punkt montowania */
char *mnt_fstypee;     /* Typ systemu plików */
char *mnt_mntopts;     /* Operacje montowania */
char *mnt_time;        /* Czas montowania */
```

W Linuxie format tablicy montowań jest trochę inny i funkcje operują na strukturze `mntent`, która różni się nieco od `mnttab`. Oto prosty przykład programu odczytującego dane z tablicy:

```
#include <stdio.h>
#include <mntent.h>

main()
{
  struct mntent *mt;
  FILE *fp;

  fp = fopen("/etc/mtab", "r");

  printf("%-20s%-20s%-30s\n",
         "punkt montowania", "typ systemu plikow", "urzadzenie");
  while ((mt = getmntent(fp)) != NULL) {
    printf("%-20s%-20s%-30s\n",
   mt->mnt_dir, mt->mnt_type, mt->mnt_fsname);
  }
}
```

I wynik jego działania:

```
[root@222-mo3-2 fs]# ./mymount
punkt montowania    typ systemu plikow  urzadzenie
/                   reiserfs            /dev/hdb1
/proc               proc                none
/proc/bus/usb       usbdevfs            none
/dev                devfs               none
/dev/pts            devpts              none
/home               reiserfs            /dev/hdb6
/usr                reiserfs            /dev/hdb7
/mnt/win_c          vfat                /dev/hda1
```

Odmontowuje system plików polecenie `umount`, której można podać zarówno urządzenie, na którym system plików się znajduje, jak i punkt jego montowania.

1.4 Automatyczne montowanie

Zwykle jak stworzymy system plików, chcemy aby był on montowany automatycznie podczas rozruchu systemu operacyjnego. Tablice wirtualnych systemów plików (`/etc/vfstab` dla systemów z rodziny System V, `/etc/fstab` w rodzinie BSD) zawiera wszystkie niezbędne informacje potrzebne do zamontowania systemu plików. Plik ten jest tworzony podczas instalacji systemu operacyjnego, administrator w wypadku pojawienia się nowego systemu plików, może dodawać tam nowe pozycje.

```
[root@222-mo3-2 fs]# cat /etc/fstab
/dev/hdb1  /         reiserfs   notail      1 1
none       /dev/pts  devpts     mode=0620   0 0
/dev/hdb6  /home     reiserfs   notail      1 2
none       /proc     proc       defaults    0 0
/dev/hdb7  /usr      reiserfs   notail      1 2
/dev/hdb5  swap      swap       defaults    0 0
```

Pola w pliku to:

1. urządzenie,
2. punkt montowania,
3. typ systemu plików,
4. opcje przekazywane do polecenia `mount`,
5. pole związane z poleceniem `dump`,
6. pole używane przez polecenie `fsck` do ustalenia kolejności w jakiej będą sprawdzane systemy plików przy rozruchu systemu operacyjnego.

Podczas rozruchu systemu operacyjnego po załadowaniu jądra do pamięci wykonywane są liczne zadania inicjalizacji. Jednym z takich zadań jest zamontowanie głównego systemu plików (/). Zwykle jest jest to jedyny system plików montowany przed wykonaniem skryptów startowych `rc`. Program `init` uruchamiany przez jądro jako pierwszy proces, na podstawie pliku `inittab` ustala jakie czynności na wykonywać. Przebieg czynności jest różny zależnie od systemu operacyjnego. W tych opartych o System V skrypty inicjalizujące znajdują się w `/etc/rcX.d`, gdzie `X` odpowiada poziomowi na którym `init` pracuje. Zainteresowanie powinien wzbudzić skrypt `SO1MOUNTSYS` znajdujący się w `/etc/rc2.d`. Tam okazuje się, że za montowanie wszystkich systemów plików jest odpowiedzialny skrypt `mountall`. W Linuxie systemy plików są montowane w skrypcie `/etc/rc.d/rc.sysinit`. Może to wyglądać tak:

```
# Mount all other filesystems (except for NFS and /proc,
# which is already mounted).
action "Mounting local filesystems: "
  mount -a -t nonfs,smbfs,ncpfs -O no_netdev,noloop,noencrypted
```

1.5 Reperowanie uszkodzonego systemu plików

Zwykle system plików może znajdować się w jedym z dwóch stanów: `clean` lub `dirty`. Żeby zamontować system plików musi on się znajdować w stanie `clean`, co znaczy że jego struktura jest poprawna. Kiedy systemy plików są zamontowane do odczytu/zapisu, są oznaczane jako `dirty`, żeby było wiadomo, że operacje są przeprowadzane na systemie plików. W czasie załamania systemu operacyjnego może się zdarzyć, że struktura systemu plików zostanie uszkodzona. W takiej sytuacji niebezpieczne byłoby zamontowanie systemu. Aby przywrócić system do stanu `clean` używamy programu `fsck`. Zależnie od systemu plików różne czynności są podejmowane.

```
[root@222-mo3-2 fs]# fsck -t ext3 -V /dev/ram14
fsck 1.32 (09-Nov-2002)
[/sbin/fsck.ext3 (1) -- /dev/ram14] fsck.ext3 /dev/ram14
e2fsck 1.32 (09-Nov-2002)
/dev/ram14: clean, 11/2512 files, 1366/10000 blocks
```

Możliwe jest również odpluskwianie (debug) systemu plików. Służy do tego polecenie `fsdb`. Jednak istnieje możliwość wyrządzenia nieodwracalnych szkód, dlatego zaleca się, żeby próbowały robić to tylko osoby bardzo dobrze znające się na systemach plików (Podobno jest to jedno z najrzadziej używanych poleceń w systemach UNIXowych).

1.6 Funkcje `statvfs()` i `statfs()`

Tak samo jak funkcja `stat()` służy do wydobywania informacji o pliku, `statvfs()` może być wykorzystywane aby otrzymać informacje o systemie plików. Nagłówek tej funkcji (z pliku `<sys/statvfs.h>`) wygląda tak:

```
int satvfs(const char *path, struct statvfs *buf);.
```

Funkcja operuje na strukturze `statvfs` o następujących polach:

```
u_long f_bsize;           /* rozmiar bloku w systemie plików */
u_long f_frsize;          /* rozmiar podstawowego bloku (jeżeli jest wspierany) */
fsblkcnt_t f_blocks;      /* liczba bloków w systemie plików w f_frsize */
fsblkcnt_t f_bfree;       /* liczba wolnych bloków */
fsblkcnt_t f_bavail;      /* liczba wolnych dostępnych bloków */
fsfilcnt_t f_files;       /* liczba węzłów plików (inodów) */
fsfilcnt_t f_ffree;       /* liczba wolnych węzłów plików */
fsfilcnt_t f_favail;      /* liczba wolnych dostępnych węzłów plików */
u_long f_fsid;            /* identyfikator systemu plików */
char f_basetype[FSTYPSZ]; /* nazwa systemu plików */
u_long f_flag;            /* maska fagi */
u_long f_namemax;         /* maskymalny rozmiar pliku */
char f_fstr[32];          /* napis specyficzny dla systemu operacyjnego */
```

W Linuxie nie ma funkcji `statvfs()`, jej zadania wykonuje `statfs()` operująca na strukturze `statfs`. Pola tej struktury są bardzo podobne do pól `statvfs`.

1.7 Ograniczanie przestrzeni dla użytkowników

Systemy plików mogą być używane przez wielu użytkowników. Może się jednak zdarzyć, że jeden użytkownik wyczerpie cały zasób przestrzeni w systemie plików. Do ograniczania miejsca w systemie plików jakie może zająć użytkownik lub grupa użytkowników każdy system plików ma własne polecenie. Do sprawdzania ograniczeń używane jest polecenie `quota`. System ten opiera się na ilości plików i bloków jakie może zająć użytkownik (niektóre systemy plików ograniczają ilość dostępnych użytkownikowi `inodów`). Są dwa rodzaje limitów, które może ustanowić administrator:

Soft limit. Kiedy użytkownik osiągnie limit jest on powiadamiany o tym i rozpoczyna się okres, kiedy może on jeszcze zwolnić zajmowane miejsce. Po upływie określonego czasu (domyślnie 7 dni) użytkownik nie może już zaalokować więcej plików i bloków. Ten limit jest użyteczny, gdy użytkownik chce korzystać z aplikacji tworzących bardzo dużo plików tymczasowych, które są potrzebne tylko podczas działania aplikacji. Niektóre systemy plików alokują dużo przestrzeni aby pliki, które powstaną były ciągłe. Po tej operacji niewykorzystywane zasoby są zwalniane. W takich wypadkach **soft limit** jest również bardzo przydatny.

Hard limit. Kiedy osiągany jest ten limit użytkownik natychmiast traci prawo do alokowania plików i bloków.

Rozdział 2

Niezależność od systemu plików

2.1 File System Switch

SVR3 z zaimplementowanym *File System Switch* (FSS) oferował środowisko obsługujące jednocześnie wiele różnych systemów plików. Tak jak wcześniejsze wersje UNIXa SVR3 zachowywał mapowanie pomiędzy deskryptorem pliku po stronie użytkownika a tablicą plików i `in-core inode'ami`. Różnica między tym systemem a wersjami zależnymi od systemu plików polegała głównie na nowej implementacji `in-core inode'ów`. Nowe `inode'y` posiadały pola wspólne dla wszystkich systemów plików (identyfikatory użytkownika i grupy, rozmiar pliku itp.) i dodatkowo możliwość związania z nimi danych specyficznych dla systemu plików. Dodatkowe pola to:

- `i_fsptr`. Wskaźnik na prywatne pola systemu plików nie widoczne dla reszty jądra. W dyskowych systemach plików pole zwykle wskazywało na kopię dyskowego `inode'u`.
- `i_fstyp`. Identyfikator systemu plików.
- `i_mntdev`. Wskaźnik na strukturę `mount` charakterystyczną dla systemu plików.
- `i_mton`. Pole używane w czasie przekształcania ścieżki do pliku. Jeżeli katalog związany z `inode'm` był punktem montowania innego systemu plików, pole to wskazywało na jego strukturę `mount`.
- `i_fstypp`. Wskaźnik do struktury z funkcjami realizującymi wywołania funkcji niezależnych od systemu plików.

Zestaw operacji charakterystycznych dla danego systemu plików znajdował się w strukturze `fstypsw`. Tablice o tej samej nazwie przechowywała te struktury dla każdego obsługiwanego systemu plików.

W momencie otwierania pliku pole `i_fstypp` było ustawiane tak, aby wskazywało na element w tablicy `fstypsw` związane z odpowiednim systemem plików. Wywołanie funkcji odbywało się przez makra np.:

```
#define FS_READI(ip) (fstypsw[(ip)->i_fstyp].fs_readi)(ip)
```

Wszystkie systemy plików musiały przestrzegać nazewnictwa FSS i ilości argumentów przekazywanych do funkcji.

FSS był dużym krokiem naprzód w porównaniu do tradycyjnych obsługujących jeden system plików UNIXów. Jednak został bardzo szybko zastąpiony przez lepszy interfejs Sun VFS/vnode zaimplementowany w SVR4.

2.2 VFS

2.2.1 Sun VFS/vnode

Architektura

Po raz pierwszy idea vnode'ów została przedstawiona publicznie w artykule Steve'a Kleimana „Vnodes: An Architecture for Multiple System Types in Sun UNIX". Zakładano realizację czterech celów:

- Implementacja systemu plików powinna być podzielona na dwie warstwy: zależna i niezależną od systemu plików. Należy zdefiniować interfejs komunikacji między tymi warstwami.
- Obsługa lokalnych systemów plikowych zarówno UNIXowych i obcych (MS-DOS), zdalnych systemów plików (NFS, RFS).
- Obsługa serwera zdalnych systemów plików.
- Operacje na systemach plików powinny być atomowe, aby unikać blokowania i danych globalnych.

Zaproponowano dwa interfejsy pomiędzy warstwą niezależną i zależną od systemu plików: `vfsops` z operacjami na systemie plików i `vnops` z operacjami na poszczególnych plikach. Ponieważ obsługiwane miały być też systemy plików nie mające nic wspólnego z UNIXem i/lub dyskami `in-core inode` nie mógł dalej pełnić swojej funkcji. Jego miejsce zajął `vnode`. Nowa struktura zawierała wszystko potrzebne warstwie niezależnej, a jednocześnie pozwalała przechowywać połączenia z charakterystycznymi dla systemu plików danymi (`inode'ami`, `rnode'ami` itp.). Pola struktury `vnode` to:

- `v_flag` – flagi: `VROOT` `vnode` jest głównym katalogiem systemu plików, `VNOMAP` wskazuje, że plik nie może być zamapowany w pamięci, `VNOSWAP` wskazuje, że plik nie może być użyty jako urządzenie wymiany, `VNOMOUNT` mówi, że plik nie może być montowany i `VISSWAP` ustawiany, gdy plik jest częścią wirtualnego użądzenia wymiany.
- `v_count` – liczba otwartych dowiązań do pliku.
- `v_shlockc` – liczba współdzielonych blokad założonych na plik.
- `v_exlockc` – liczba wyłącznych blokad założonych na plik.
- `v_vfsmountedhere` – jeżeli inny system plików jest zamontowany w katalogu związanym z `vnode'm` pole to wskazuje strukturę `vfs` tego systemu.
- `v_op` – wskaźnik do operacji na pliku związanym z `vnode'm`.
- `v_vfsp` – określa typ pliku reprezentowanego przez `vnode`.
- `v_data` – wskaźnik na dane charakterystyczne dla systemu plików (np.: `inode`.

Jeżeli system plików nie ma UNIXowego interfejsu to odpowiednie pola `vnodeops` są ustawiane na `fs_nosys`. Funkcja ta zwraca w wypadku wywołania `ENOSYS`.

Struktura `vfs`

Zamontowane systemy plików przechowane są w postaci listy struktur `vfs`. Podobnie jak `vnode` `vfs` jest niezależny od systemu plików. Jedynie pole `vfs_data` może być użyte do przechowania dowiązania do danych charakterystycznych dla systemu plików. Z każdym systemem plików związana jest struktura `vfsops` zawierająca operacje, które mogą być na nim wykonywane. Dowiązanie do tej struktury znajduje się w polu `vfs_op` z `vfs`. Dostępne są następujące operacje:

- `vfs_mount` – funkcja używana do montowania systemu plików.
- `vfs_unmount` – funkcja używana do demontowania systemu plików.

- `vfs_root` – funkcja zwraca główny `vnode` systemu plików.
- `vfs_statfs` – zwraca informacje potrzebne funkcji systemowej `statfs()`.
- `vfs_sync` – funkcja zrzuca dane pliku i dane strukturalne systemu plików na dysk. Używana jest aby zminimalizować straty w czasie załamania systemu.
- `vfs_fid` – funkcja używana przez NFS dla zapewnienia obsługi pliku.
- `vfs_vget` – funkcja używana przez NFS do konwersji funkcji zwróconej przez wywołanie `vfs_fid`.

Operacje na plikach

Wszystkie operacje, które można przeprowadzić na plikach zdefiniowane są w strukturze `vnodeops`. Są to następujące funkcje:

- `vop_open` – funkcja używana dla plików specjalnych urządzeń. Używana tylko kiedy `vnode` został zwrócony z wcześniejszego wywołania `vop_lookup`.
- `vop_close` – podobnie do `vop_open`.
- `vop_rdwr` – czytanie i zapis do pliku.
- `vop_ioctl` – wywołanie `ioctl` na pliku.
- `vop_select` – implementacja funkcji `select()`.
- `vop_getattr` – używana do wydobywania informacji potrzebnych funkcjom takim, jak `stat()`.
- `vop_setattr` – używana do ustawiania tych informacji.
- `vop_access` – sprawdzenie pozwolenia na odczyt z pliku, zapis do pliku i wykonanie pliku.
- `vop_lookup` – zastępuje funkcję `namei()`.
- `vop_create` – utworzenie nowego pliku w katalogu związanym z `vnode'm`.
- `vop_remove` – usunięcie katalogu.
- `vop_link` – impementacja wywołania systemowego `link()`.
- `vop_rename` – impementacja wywołania systemowego `rename()`.
- `vop_mkdir` – impementacja wywołania systemowego `mkdir()`.
- `vop_rmdir` – impementacja wywołania systemowego `rmdir()`.
- `vop_readdir` – odczytuje podkatalogi z katalogu związanego z `vnode'm`. Funkcja wywoływana przez `getdents()`.
- `vop_symlink` – impementacja wywołania systemowego `symlink()`.
- `vop_readlink` – odczytuje zawartość dowiązania symbolicznego.
- `vop_fsync` – zrzuca zawartość pliku na dysk. Używana przez `fsync()`.
- `vop_inactive` – funkcja wywoływana gdy nikt już nie korzysta z `vnode'u` i może być już on zwolniony.
- `vop_bmap` – funkcja wywołana przez wirtualną pamięć przy stronicowaniu.
- `vop_strategy` – przeniesienie bloków do pamięci po wywołaniu `vop_bmap()`.

- vop_bread – funkcja odczytuje logiczny blok z vnode'u i zwraca bufor z cache'a buforów, który zawiera odczytane dane.
- vop_brelse – funkcja zwalnia bufor zwrócony przez vop_bread().

Jeżeli system plików nie obsługuje operacji to pole powinno być ustawione na fs_nosys(), która zwraca ENOSYS. Operacje wywoływane są przez makra np.:

```
#define VOP_INACTIVE(vp, cr) \
  (*(vp)->v_op->vop_inactive)(vp, cr)
```

2.2.2 SVR4 VFS/vnode

Architektura

SVR4 powstał w wyniku połączenia SVR3 i SunOS. Jednym z celów projektu było połączenie interfejsu VFS/vnode z opracowanym przez AT&T *File System Switch*. Pojawiła się więc tablica budowana dynamicznie w czasie kompilacji jądra *virtual system switch table* podpięta pod vfssw[] zawierająca pozycję dla każdego systemu plików, z którym jądro mogło współpracować. Elementami tablicy są struktury vfssw:

```
struct vfssw {
  char *vsw_name; /* nazwa systemu plików */
  int (*vsw_init)(); /* funkcja wywoływana w czasie inicjalizacji jądra */
  struct vfsops *vsw_vfops;
}
```

Operacje, które można wykonać na systemie plików przechowane są zarówno w vsw_vfops i w polu vfs_ops struktury vfs:

- vfs_mount – montowanie systemu plików.
- vfs_unmount – demontowanie systemu plików.
- vfs_root – zwraca główny vnode sytemu plików.
- vfs_statvfs – zwraca statystyki związane z systemem plików.
- vfs_sync – zrzucenie zmodyfikowanych danych na dysk.
- vfs_vget – funkcja wykorzystywana przez NFS, zwraca vnode obsługujący podany plik.
- vfs_mountroot – Pole wykorzystywane w systemach plików, które mogą być montowane jako główny system plików.

Vnode i operacje na plikach

Vnode'y uległy małym modyfikacjom. Usunięte zostały pola v_shlockc i v_exlockc. Dodano następujące pola:

- v_stream – jeżeli otwarty plik wskazuje na urządzenie strumieniowe, pole to wskazuje na początek (head) strumienia.
- v_filocks – dowiązanie do blokad założonych na plik.
- v_pages – dowiązanie do wszystkich stron związanych z vnode'm w cache'u.

Większych zmian dokonano w vnodeops. Usunięto operacje: vop_bmap(), vop_bread(), vop_brelse(), vop_strategy(), vop_rdwr() i vop_select(). Dodano następujące operacje:

- vop_read – odczyt z pliku.

- `vop_write` – zapis do pliku.
- `vop_setfl` – sprawdzenie przekazywanej przez system flagi.
- `vop_fid` – utworzenie obsługi pliku, z której NFS może później korzystać.
- `vop_rwlock` – blokowanie zapisu/odczytu pliku (`LOCK_SHARED`, `LOCK_EXCL`).
- `vop_rwunlock` – odblokowanie zapisu/odczytu pliku.
- `vop_seek` – sprawdzenie czy w systemie plików dozwolone jest wywołanie `lseek()` z podanymi argumentami (niektóre systemy plików nie umożliwiają zapisu za końcem pliku).
- `vop_cmp` – funkcja porównująca `vnode'y`
- `vop_frlock` – funkcja implementująca blokowanie plików i rekordów.
- `vop_space` – funkcja używana w systemach plików umożliwiających zwolnienie przestrzeni w trackie pracy nad plikiem.
- `vop_realvp` – wydobywanie `vnode'a` w systemach plików ukrywających je.
- `vop_getpage` – przeczytanie danych w pliku w przypadku błędu braku strony.
- `vop_putpage` – zrzucenie zmodyfikowanych danych na dysk.
- `vop_map` – funkcja używana do implementacji mapownia pamięci.
- `vop_addmap` – dodanie mapowania.
- `vop_delmap` – usunięcie mapowania.
- `vop_poll` – implementacja funkcji systemowej `poll()`.
- `vop_pathconf` – implementacja wywołań systemowych `pathconf()` i fpatchconf().

Podobnie jak poprzednio operacje są dostępne poprzez makra.

2.2.3 BSD

Architektura

Ważną zmianą była modyfikacja operacji `namei()`, która w BSD korzysta z następującej struktury danych (nie korzysta z danych ustawionych przez wcześniejsze wywołanie w jądrze `namei()`):

```
struct nameidata {
  caddr_t ni_dirp          /* dowiązanie do ścieżki */
  enum uio_seg ni_seg;     /* lokalizacja ścieżki */
  short ni_nameiop;        /* operacja do wykonania */
  struct vnode *ni_cdir;   /* aktualny katalog roboczy */
  struct vnode *ni_rdir;   /* główny katalog */
  struct ucred *ni_cred;   /* dane związane z wywołującym */
  caddr_t ni_pnbuf;        /* bufor nazwy ścieżki */
  char *ni_ptr;            /* aktualna pozycja w nazwie ścieżki */
  int ni_pathlen;          /* ilość znaków pozostała w nazwie ścieżki */
  short ni_more;           /* flaga informująca czy pozostało coś do przekształcenia */
  short ni_loopcnt;        /* liczba odnalezionych symbolicznych dowiązań */
  struct vnode *nivp       /* zwracany vnode */
  struct vnode *nidvp      /* vnode nadkatalogu */
}
```

Operacje, które można wykonać (ni_nameiop) to:

- LOOKUP – wyszukanie;
- CREATE – przygotowanie do stworzenia pliku;
- DELETE – przygotowanie do usunięcia pliku;
- WANTPARENT – zwrócenie vnode'a nadkatalogu;
- NOCACHE – usunięcie nazwy z cache'a;
- FOLLOW – podążanie za dowiązaniami symbolicznymi;
- NOFOLLOW – niepożądanie za dowiązaniami symbolicznymi;

VFS i vnode

Większość struktur pozostała taka sama jak w Sun VFS (dokonano bardzo małych modyfikacji). Struktura vfs zyskała pole vfs_bsize (optymalny rozmiar bloku), do statfs dodano f_bsize, natomiast vnode powiększył się o v_text używany przez pliki wykonywalne.

Przybyło kilka operacji na plikach:

- vn_mknod – obsługa funkcji systemowej mknod(S);
- vn_read – obsługa wywołania systemowego read(S);
- vn_write – obsługa wywołania systemowego write(S);
- vn_seek – obsługa wywołania systemowego lseek(S);
- vn_abortop – funkcja wywoływana gdy poprzednio wywołano namei() z CREATE lub DELETE, ale zrezygnowano z przeprowadzenia operacji;
- vn_lock – blokowanie operacji na pliku;
- vn_unlock – odblokowanie operacji na pliku;

2.2.4 HP-UX

VFS i vnode

Budowa VFS i vnode'ów w HP-UX jest bardzo podobna do SVR4. Struktura vfs zachowuje większość składowych z SVR4 z kilkoma małymi dodatkami. Operacje VFS też nie różnią się zbytnio od SVR4 (podobnie jest kilka dodatkowych funkcji). Strukturę vnode odróżniają od tej z SVR4 dwie listy: z niezmienionymi buforami v_cleanblkhd i z buforami zwierającymi modyfikacje v_dirtyblkhd. Konstrukcja ta jest podobna do v_pages, ale zapewnia proste rozróżnienie, które bufory były modyfikowane.

Rozdział 3

Najpopularniejsze systemy plików dostępne pod Linuxem

3.1 Streszczenie

Niniejsza część dokumentu stara się po krótce scharakteryzować najpopularniejsze dziś systemy plików używane pod Linuxem, podać ich najważniejsze cechy oraz historię ich postania.

3.2 Wstęp

Dzięki dostępności kodu źródłowego systemu Linux oraz wielkiej liczbie osób zaangażowanych w jego rozwój, powstało, lub zostało na niego przeniesionych, co najmniej kilkadziesiąt różnych systemów plików. Są wśród nich systemy eksperymentalne, o znaczeniu dziś już tylko czysto historycznym, ale można tam także znaleźć efektywne, nowoczesne i praktyczne systemy przeznaczone do różnego rodzaju zastosowań.

Postaram się poniżej przedstawić te najpopularniejsze.

3.3 Extended Filesystem 3

3.3.1 Historia

MINIX

Korzenie Extended Filesystem 3 (Ext3) sięgają początków Linuxa, a właściwie nawet wcześniej i są związane z systemem operacyjnym MINIX. Dla MINIXa został opracowany specjalny system plików o tej samej nazwie. Wraz z pojawieniem się Linuxa zaadoptowano ten system na jego potrzeby. W ten sposób MINIX stał się pierwszym systemem plików przeznaczonym Linuxa.

System MINIX posiadał wiele ograniczeń, które odziedziczyła po nim jego Linuxowa wersja. Jednym z nich było ograniczenie długości nazwy pliku do 14 znaków (wkrótce rozluźniono je do 30 znaków). Poza tym wielkość partycji nie mogła przekraczać 64Mb.

Mimo tych niedogodności system MINIX pozostawał w użyciu przez bardzo długi czas zyskując sobie miano najbardziej godnego zaufania systemu plików dla Linuxa.

Ext

Następca MINIXa stał się system ExtFS (Extended Filesystem). Pozwalał on na stosowanie nazw plików o długości nie większej niż 255 oraz plików i partycji do 4Gb. Nie zyskał on sobie wielkiego uznania ze względu na sposób spamiętywania wolnych bloków i i-węzłów (stosowane w tym celu były listy), który to prowadził, przy dłuższym użytkowaniu, do znacznej fragmentacji dysku i poważnego spadku wydajności.

Ext2

Po Ext pojawił się, dość nieoczekiwanie, system Ext2, bazujący poza Ext także na FFS (BSD Fast Filesystem). Z systemu FFS zaczerpnięto między innymi podział partycji na bloki alokacji oraz ich strukturę.

Ext2 zyskał sobie znacznie większe uznanie od Ext i wkrótce stał się podstawowym systemem plików dla Linuxa.

Dość szybko zaczęły się pojawiać różnego rodzaju nakładki (*patches*) na Ext2. Po pewnym czasie zostały one zebrane tworząc w ten sposób system Ext3.

3.3.2 Cechy

Ze względu na fakt, że system Ext2 był bardzo dokładnie omówiony na wykładzie z Systemów Operacyjnych, nie będę tu podawał szczegółów jego budowy. Wspomnę jedynie o cechach, które zostały dodane do niego w wersji 3:

- księgowanie (*journaling*) operacji dyskowych zapewniające szybkie odzyskanie spójności danych w razie załamania się systemu, z możliwością wyboru jednego z trzech poziomów bezpieczeństwa (od najbardziej do najmniej bezpiecznego):
  - journal – zapisuje do dziennika wszystkie operacje dyskowe (zarówno na danych jak i metadanych systemu plików)
  - ordered – zapisuje wszystkie zmiany metadanych systemu plików dopiero po dokonaniu zapisu samych danych (tryb domyślny)
  - writeback – zapisuje asynchronicznie wszystkie zmiany metadanych systemu plików
- fragmentacja bloków (podział bloku dyskowego na mniejsze jednostki równej wielkości, umożliwiające zapis w jednym bloku kilku małych plików)
- opcjonalna obsługa skompresowanych i zaszyfrowanych plików
- opcjonalna obsługa list kontroli dostępu (*ACL – Access Control List*) do pliku
- opcjonalna obsługa zamazywania zawartości bloków dyskowych, przy usuwaniu plików do których te bloki przynależały

Ext3 nie jest być może systemem bardzo innowacyjnym w porównaniu z chociażby ReiserFS, XFS czy JFS, lecz mimo to zapewnia wciąż w porównaniu z konkurentami dobrą wydajność, wystarczająca do większości zastosowań.

3.4 Reiser Filesystem

3.4.1 Omówienie

ReiserFS jest bardzo ciekawym system plików rozwijanym od samego początku na Linuxie. Ze względu na innowacyjność wielu rozwiązań tam zastosowanych (zwłaszcza jeśli chodzi o nadchodzącą wersję 4), oraz jego rosnąca popularność zostanie on omówiony nieco dokładniej w następnej części.

3.5 Extended Filesystem

3.5.1 Wstęp

System XFS został stworzony przez firmę SGI (Silicon Graphics Incorporation) jako system plików dla Irixa (domyślny system plików dla tego systemu od wersji 6.0). Zastąpił w tej roli system Extent Filesystem – EFS (który skądinąd jest także obsługiwany przez Linuxa).

Od maja 2001 roku jest dostępna implementacja XFS dla Linuxa stworzona przez SGI (jako producent sprzętu firma powoli wycofuje się z produkcji oprogramowania, chcąc ostatecznie przestawić się na sprzedaż komputerów działających pod nadzorem systemu Linux).

Mimo, iż od rozpoczęcia pracy nad XFS minęło już 11 lat (prace rozpoczęto w 1993 roku), pozostaje on nadal bardzo wydajnym i nowoczesnym systemem plików, mającym zastosowanie zwłaszcza w większych systemach komputerowych (mocne stacje robocze i serwery).

3.5.2 Cechy

Do najważniejszych cech systemu XFS można zaliczyć:

- obsługa nawet dużych plików (9Tb) i partycji (18TB)
- dynamiczna alokacja i-węzłów (system stara się je umieszczać możliwie blisko danych)
- efektywne wyszukiwanie i-węzłów, wolnego miejsca na dysku oraz plików w katalogach oparte na odpowiednich B+ drzewach
- możliwość rezerwacji minimalnego pasma transmisji danych dla procesu rzędu np. 2MB/sekundę (użyteczne zwłaszcza dla aplikacji przetwarzających dane wideo itp.)
- zaawansowany asynchroniczny system kronikowania operacji wykonywanych na metadanych (i-węzłach, drzewach systemowych, katalogach) z możliwością przywracania stanu systemu w trakcie jego normalnego działania (w locie)
- duża skala współbieżności wykonywanych operacji
- zastosowanie przedziałów bloków dyskowych (*Extent*) w celu zmniejszenia fragmentacji danych i zwiększenia efektywności odwołań do nich
- udostępnienie list uprawnień (*ACL - Access Control List*) o wielkości do 64kb (zgodnych z normą POSIX)
- przechowywanie małych plików (linków symbolicznych) oraz katalogów bezpośrednio w i-węzłach co znacznie zwiększa efektywność odwołań do nich

3.6 Journaled Filesystem

3.6.1 Wstęp

System Journaled Filesystem (JFS) powstał na początku lat 90 w laboratoriach firmy IBM jako system przeznaczony głównie dla dużych serwerów tej firmy korzystających z systemu operacyjnego AIX (obecnie działa także pod HP-UX, OS/2 od wersji 5 oraz Linuxem).

W wersji dla Linuxa system jest dostępny na licencji GPL jako ładowalny moduł jądra.

3.6.2 Cechy

Do najważniejszych cech systemu JFS należą:

- obsługa bardzo dużych plików i partycji (rzędu kilku tysięcy TB)
- dynamiczna alokacja i-węzłów
- efektywne wyszukiwanie i-węzłów, wolnego miejsca na dysku oraz plików w katalogach oparte na odpowiednich B+ drzewach
- zaawansowany system kronikowania operacji dyskowych wbudowany bezpośrednio w system plików, a nie dodany do niego później jak w przypadku innych systemów plików

- zastosowanie przedziałów bloków dyskowych (*Extent*) w celu zmniejszenia fragmentacji danych i zwiększenia efektywności odwołań do nich
- udostępnienie list uprawnień (*ACL - Access Control List*)
- przechowywanie małych plików (linków symbolicznych) oraz katalogów bezpośrednio w i-węzłach co znacznie zwiększa efektywność odwołań do nich
- efektywna obsługa rozrzedzonych plików

3.7 Inne systemy plików

3.7.1 BSD FastFS

Jest to natywny system plików używany przez BSD (*Barkley System Distribution*) oraz większość systemów operacyjnych opartych na nim jak: OpenBSD, FreeBSD, NetBSD, Sun Solaris. Na jego podstawie został także stworzony system Ext2 dla Linuxa, z którym FFS ma bardzo wiele wspólnych cech.

3.7.2 VxFS

VxFS (Veritas Filesystem) jest komercyjnym systemem plików firmy Veritas, zajmującej się tworzeniem oprogramowania do przechowywania, zabezpieczania i odzyskiwania danych. Dostępne są wersje tego systemu między innymi na: HP-UX, SCO UnixWare, Sun Solaris i Linuxa (nie są one w pełni ze sobą kompatybilne).

Do najważniejszych cech tego systemu można zaliczyć:

- obsługa dużych plików (do 2TB) i partycji
- alokacja danych oparta na przedziałach bloków dyskowych (*extents*)
- kronikowanie (*journaling*) z możliwością przywracania spójności danych w trakcie działania systemu
- ograniczenia przestrzeni dyskowej w stylu BSD (*BSD style quotas*)
- obsługa dynamicznej alokacji i-węzłów

Istnieje wolnodostępny zestaw narzędzi lini poleceń do odczytu partycji VxFS (bez możliwości zapisu).

3.7.3 NTFS

System NTFS jest systemem opracowanym i rozwijanym przez firmę Microsoft. Jest on tu wspomniany głównie ze względu na powszechność systemu Windows i wynikającą z tego względu częstą konieczność korzystania z partycji założonych przez ten system pod Linuxem.

Historia NTFS sięga systemu HPFS (High Performance Filesystem) powstałego ze współpracy firm IBM i Microsoft. Po jej zerwaniu Microsoft wykorzystał wiele rozwiązań z HPFS w nowo tworzonym przez siebie dla WindowsNT systemie plików i tak powstał NTFS.

Ze względu na politykę firmy Microsoft nie jest dostępna pełna specyfikacja NTFS co znacznie utrudnia jego przeniesienie na Linuxa. W chwili obecnej sterownik NTFS w jądrze obsługuje poprawnie jedynie odczyt z tego rodzaju partycji. Zapis jest możliwy, ale wiąże się z ryzykiem rozspójnienia, lub wręcz całkowitej utraty danych, tak więc włączenie tej opcji nie jest zalecane.

3.7.4 FAT

System FAT jest starym systemem plików uznawanym obecnie za dość przestarzały. Nie zapewnia on właściwie żadnej ochrony zgromadzonych nań danych przed dostępem przez nieuprawnionych użytkowników. Linux obsługuje pewne rozszerzenia tego systemu zapewniające między innymi obsługę uprawnień, łączy symbolicznych, plików urządzeń itp.

Obecnie FAT stracił swoją pozycje jako lokalny system plików jest jednak nadal wykorzystywany, głównie ze względu na jego prostotę i powszechność obsługi przez systemy operacyjne, przez: urządzenia do przenoszenia danych takie jak np. PenDrive, cyfrowe dyktafony, aparaty fotograficzne, kamery, FDD itp.

3.8 Źródła

Ciekawe informacje dotyczące omawianego tematu można znaleźć w:

1. **Filesystems-HOWTO** - dokument omawiający po krótce większość używanych pod Linuxem systemów plików; niestety dość nieaktualny; zawiera ciekawe odnośniki do innych materiałów
2. katalog **/usr/src/linux/fs** i odpowiednie podkatalogi - kody źródłowe odpowiednich systemów plików; jest to najlepsza metoda poznania sposobu funkcjonowania systemu, niestety bardzo pracochłonna i nie zawsze owocna ze względu na brak odpowiednich komentarzy
3. **http://oss.software.ibm.com/developerworks/opensource/jfs/** - Strona domowa projektu JFS
4. **http://oss.sgi.com/projects/xfs/** - Strona domowa projektu XFS
5. **http://namesys.com/** - Strona domowa projektu ReiserFS

Rozdział 4

Sieciowe systemy plików

4.1 Streszczenie

Rozproszony system plików, implementowany za pomocą modelu klient-serwer, pozwala użytkownikom na współdzielenie plików poprzez sieć komunikacyjną.

Podstawowe pojęcia związane z DFS zostały przedstawione w wykładzie 12, skupię się zatem w niniejszej części na przedstawieniu konkretnych rozwiązań i przykładów DFS, obejmujących NFS (ang. Network Filesystem) firmy Sun Microsystems (a zwłaszcza wersję 4.0), RFS (ang. Remote File Sharing) firmy AT&T oraz AFS (ang. Andrew File System) stworzony na Carnegie-Mellon University, który przekształcił się w DFS (ang. Distributed File Service), jedną ze składowych środowiska DCE (ang. Distributed Computing Environment).

4.2 NFS

4.2.1 Wprowadzenie

Krótki wstęp

Większość materiału umieszczonego w danym rozdziale poświęcona jest NFS w wersji 4 (na razie znany jest sam protokół, bez jakiejkolwiek implementacji). Wersja 3.0 była omawiana na wykładzie.

Od początku swego istnienia, w latach 80., system NFS był ogromnym sukcesem. Sprawiły to zapewne przenośność i prostota przyjętych rozwiązań. Podstawowe cele, jakie postawili sobie projektanci firmy Sun to:

1. niezależność od sprzętu i systemu operacyjnego
   Osiągnięta dzięki prostocie protokołu przesyłania danych pomiędzy klientem a serwerem.

2. odporność na awarie serwera
   Serwer NFS został pomyślany jako bezstanowy, dzięki czemu klient nie był w stanie rozróżnić czy serwer uległ awarii czy że wolno działa. Awaria serwera nie ma wpływu na działanie klienta.

3. przezroczystość dostępu do plików
   Aplikacje mają dostęp do plików poprzez NFS w taki sam sposób, jak do plików na lokalnym dysku.

4. UNIXowa semantyka

5. szybkość działania

NFS v4

NFS v4 ma być rozwinięciem poprzednich wersji, będzie się jednak znacząco różnić w kilku podpunktach od swych poprzedników. Prace nad wersją 4.0 prowadzone są od 1998 roku przez IETF (ang. Internet Engineering Task Force), kiedy to Sun Microsystems zrzekł się pełnej kontroli nad opracowywaniem nowej wersji. Generalnie, można powiedzieć, że wersja 4.0 ma rozwiązać większość problemów i zastrzeżeń, jakie są do wersji 3.0, a więc:

- szybkość działania w sieciach rozległych i Internecie
- zwiększone bezpieczeństwo poprzez szyfrowanie danych
- łatwa rozszerzalność i rozbudowywalność protokołu
- niezależność sprzętowa
- zwiększona elastyczność systemu plików

Oczywiście protokół w wersji 4.0 pozostanie kompatybilny z poprzednimi wersjami.
Jak widać podstawowy problem, to przystosowanie NFS do działania w Internecie.

Podstawowe cechy

Podstawowe cechy NFSv4 oraz różnice z NFSv3:

- niezależność od systemu operacyjnego

  Niezwykle ważne założenie, które ma zostać jeszcze pełniej niż wcześniej zrealizowane.

- komunikacja klientów i serwerów poprzez wywołania procedur zdalnych RPC (ang. Remote Procedure Call)

  Protokół NFSv4 nadal będzie oparty na protokole RPC, jednak nie będą już implementowane konkretne procedury, lecz operacje (ang. operation). Dzięki temu znacząco zmniejszy się liczba pakietów RPC przesyłanych przez sieć, a poprawi szybkość i wydajność działania.

- dane kodowane metodą rozszerzonej reprezentacji danych XDR (ang. Extended Data Representation)

  Dzięki zastosowaniu jednolitego kodowania możliwe jest wymienianie danych w komputerach stosujących różną reprezentację danych.

- brak pomocniczych protokołów

  Ważną rolę odgrywały w NFSv3 protokół Mount oraz sieciowy zarządca blokad NLM (ang. Network Lock Manager). W NFSv4 obsługa danych aspektów systemu została wbudowana w protokół.

- jeden port 2049

  NFS miał zawsze przydzielony port o numerze 2049, jednak protokół Mount za każdym razem dostawał inny numer przydzielany dynamicznie. Jest to sytuacja niezwykle niekorzystna dla użytkowników firewall. W NFSv4 problem rozwiązano, wbudowywując wszystkie procedury w protokół oraz decydując się na użycie TCP do przesyłania danych w Internecie.

- blokowanie zintegrowane z NFSv4

  Zapobiega to rozspójnianiu danych.

- bezpieczeństwo przesyłania danych po sieci

  Zabezpieczenia z wcześniejszych wersji, pomyślanych na małe lokalne sieci typu LAN, są dalece niewystarczające dla aplikacji sieciowych. W NFSv4 zadbano zatem o bezpieczeństwo używając GSS-API.

- nazewnictwo plików

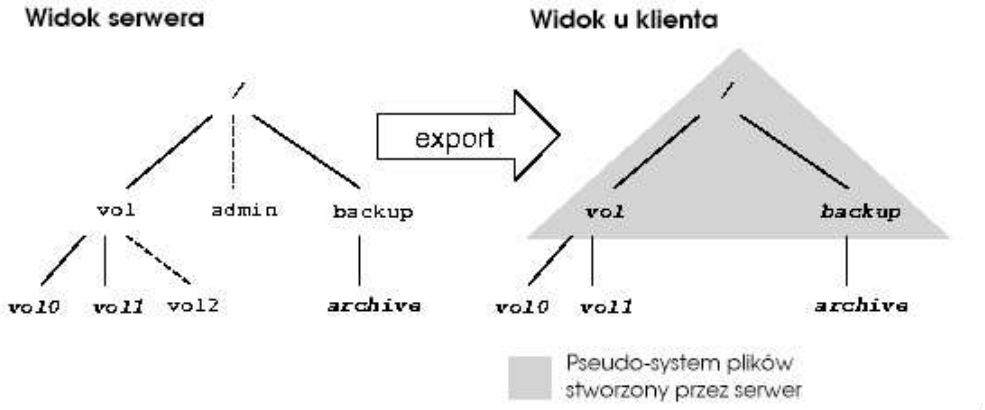
  Kodowanie nazw plików odbywa się w standardzie UTF-8, dzięki czemu możliwe jest kodowanie długich i międzynarodowych nazw.

Wymienione tu pokrótce aspekty NFSv4 zostaną szerzej omówione w dalszych rozdziałach.

4.2.2 Omówienie NFSv4

Eksportowanie systemu plików

W NFSv4 jak w poprzednich wersjach serwer eksportuje różne systemy plików, jednak dla klienta jest to jeden spójny obraz, co bardzo ułatwia przechodzenie po drzewie systemu plików.



Eksportowanie plików w systemie NFS.

RPC i procedury `COMPOUND`

Wprowadzenie procedur `COMPOUND` jest znaczącą innowacją wprowadzoną w wersji 4.0. Tego typu procedury pozwalają na grupowanie tradycyjnych operacji na plikach w jedno żądanie, które zostanie wysłane do serwera po sieci. Tak więc protokół NFSv4 nie implementuje już każdej akcji jako odrębnej procedury RPC, lecz działa na operacjach, które składają się na część procedury `COMPOUND`. Została oczywiście zachowana zgodność funkcjonalna z poprzednimi wersjami. Obsługa błędów po stronie serwera również nie jest trudna: serwer wykonuje polecenia dopóki w wykonaniu któregoś nie nastąpi błąd, jeśli tak to przerywa obsługę konkretnego żądania i zwraca błąd klientowi. Można powiedzieć, że de facto jedynymi procedurami RPC pozostały procedury `NULL` oraz `COMPOUND`.
Nowymi operacjami wprowadzonymi dopiero w wersji 4.0 są stanowe operacje `OPEN` i `CLOSE`. Jest to tym większa nowość, gdyż, jak wiadomo, serwer NFS był wcześniej bezstanowy i była to jedna z jego głównych zalet (łatwość w usuwaniu awarii). Dzięki wprowadzeniu owej stanowości osiągnięto atomowość operacji na plikach i zgodność z Windows'ową semantyką. Ponadto możliwe jest wprowadzenie delegowania, a co za tym idzie także „agresywnego" cache'owania po stronie klienta (klient na czas delegacji dostaje wyłączność na działania na danym pliku, dzięki czemu może on wykonywać wszystkie operacje lokalnie, co jest szybsze i nie zajmuje czasu pracy serwera).

Aby zilustrować wygodę i zwiększenie szybkości dzięki użyciu funkcji `COMPOUND` i pozbyciu się protokołu Mount prześledźmy przykład.

```
mount bayonne:/export/vol0 /mnt
dd if=/mnt/home/data bs=32k count=1 of=/dev/null
```

Chcemy zamontować zdalny system plików i przeczytać 32KB z pliku.
NFSv3

```
PORTMAP C GETPORT (MOUNT)
PORTMAP R GETPORT
MOUNT C Null
MOUNT R Null
MOUNT C Mount /export/vol0
MOUNT R Mount OK
PORTMAP C GETPORT (NFS)
PORTMAP R GETPORT port=2049
NULL
NULL
FSINFO FH=0222
FSINFO OK
GETATTR FH=0222
GETATTR OK
LOOKUP FH=0222 home
LOOKUP OK FH=ED4B
LOOKUP FH=ED4B data
LOOKUP OK FH=0223
ACCESS FH=0223(read)
ACCESS OK (read)
READ FH=0223 at 0 for 32768
READ OK (32768 bytes)
```

NFSv4

```
PUTROOTFH
  LOOKUP "export/vol0"
   GETFH
   GETATTR
PUTROOTFH OK CURFH
   LOOKUP OK CURFH
   GETFH OK
   GETATTR OK
PUTFH
   OPEN "home/data"
   READ at 0 for 32768
PUTFH OK CURFH
   OPEN OK CURFH
   READ OK (32768 bytes)
```

Struktury danych i atrybuty

Jak wiadomo protokół NFS związuje z każdym plikiem i katalogiem uchwyt pliku (ang. file handle). Zazwyczaj uchwyt pliku zawiera identyfikator systemu plików (ang. file system ID), numer i-węzła oraz numer pokolenia (ang. generation number). Rozwiązanie to jest niewystarczające w nie-UNIXowych systemach plików, gdzie identyfikacja odbywa się poprzez ścieżkę. Dlatego też w NFSv4 wprowadzono drugi rodzaj uchwytu- zmienny (ang. volatile handler).
W związku ze stanowością serwera i wbudowaniem blokad w protokół wprowadzono odpowiednie struktury danych: identyfikator klienta (ang. client ID) oraz stanu (ang. state ID). Dzięki temu możliwa jest

identyfikacja klientów jak też powrót do normalnego działania po awarii.
Atrybuty dla operacji zostały w NFSv4 określone dużo elastyczniej niż w poprzednich wersjach, a mianowicie wprowadzono 3 rodzaje atrybutów:

- obowiązkowy (ang. mandatory)

  Są to np. rozmiar, typ pliku.

- zalecany (ang. recommended)

  Są to np. właściciel, prawa UNIXowe.

- nazwane (ang. named)

  Są to ciągi bitów. Pozwajają na zwiększenie elastyczności systemu plkiów (nie musi być już UNIXowy).

Blokady i wyłączność

Blokady zostały w wersji 4 wbudowane w protokół NFS. Powrót po awarii będzie dzięki temu ułatwiony (w przeciwieństwie do używanego wcześniej NLM). Działanie blokad zasadza się na okresach wyłączności kontrolowania stanu pliku (ang. leases). Serwer daje wyłączność klientowi na pewien czas, podczas którego nie może udzielić jej innemu klientowi. Jeśli klient chce ponowić blokadę musi się jeszcze raz zwrócić do serwera.
Klientom udostępniono również żądanie, podczas otwierania pliku, wyłączności na możliwość jego otwierania (ang. share reservation).

Bezpieczeństwo

Zwiększenie bezpieczeństwa jest jednym z podstawowych zadań, jakie stoją przed projektantami NFSv4, gdyż w obecnie stosowanym rozwiązaniu mechanizmy uwierzytelniające (AUTH_NULL, AUTH_UNIX, AUTH_KERB) nie sprawdzają się na poziomie sieci rozległych. Problem powinno rozwiązać użycie RPCSEC_GSS oparte na GSS-API (Generic Security Service), które pozwala na dodawanie nowych mechanizmów bezpieczeństwa bez konieczności tworzenia na nowo aplikacji NFS. Ponadto NFSv4 wymusza tworzenie aplikacji zgodnych z mechanizmem Kerberos wersja 5 oraz LIPKEY (The Low Infrastructure Public Key).

Użytkownicy i grupy są identyfikowani w NFSv4 jako:

- uzytkownik(at)domena
- grupa(at)domena

gdzie domena jest zarejestrowana w DNS.
W protokole NFSv4 wprowadzono również listę dostępu ACL (ang. Access Control List) opartą na modelu Windows NT, gdzie dostęp jest określany następującymi parametrami:

- ALLOW

  Jednostka ma dostęp do zadanego pliku bądź katalogu.

- DENY

  Jednostka nie ma dostępu.

- AUDIT

  Próby dostępu są kronikowane.

- ALARM

  Próba dostępu generuje systemowo określony alarm.

4.3 RFS

4.3.1 Wprowadzenie

Krótki wstęp

System RFS został stworzony w AT&T. Początkowo działał on na systemie SVR3 UNIX, lecz szybko RFS przebudowano i zintegrowano z interfejsem węzłów wirtualnych (w wersji SVR4). Tutaj omówimy właśnie tę implementację RFS.

Cele projektowe

Podstawowe założenia to:

- przezroczysty dostęp do plików i urządzeń zdalnych
- zachowanie pełnej semantyki UNIXowej

  Wspierane są wszystkie typy plików i urządzeń jak np. kolejki FIFO. Możliwe jest nakładanie blokad na pliki.

- zapewnienie zgodności binarnej

  Nie trzeba modyfikować istniejących aplikacji.

- niezależność sieciowa

  RFS można używać zarówno w sieciach lokalnych jak i rozległych.

4.3.2 Budowa systemu RFS

Architektura RFS

Właściwości architektury RFS:

- model klient-serwer

  Serwer eksportuje pliki, które klilenci montują.

- serwer jest stanowy

  Jest to konieczne do zapewnienia UNIXowej semantyki plików otwartych. Serwer utrzymuje tablicę wszystkich klientów i przechowuje informacje o stanie plików.

- zastosowano usługę transportową typu obwodu wirtualnego, TCP/IP

  Każda para klient-serwer stosuje jeden obwód, który jest tworzony podczas operacji *mount*.

- zastosowano mechanizm strumieni i interfejs dostawcy transportu TPI

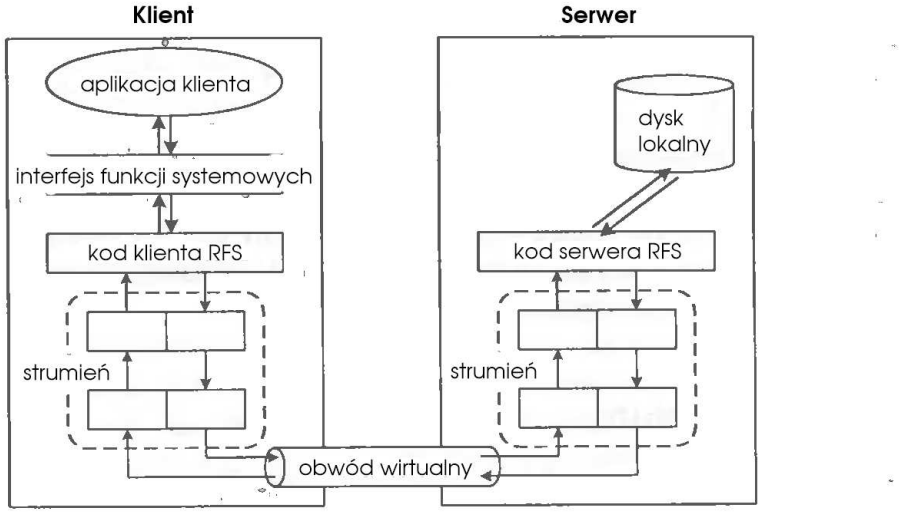
  W RFS można stosować wiele strumieni, może być zatem wielu dostawców transportu na jednym komputerze.

- mamy niezależność położenia pliku dzięki nazwom zasobów

  Każdy eksportowany katalog plików dostaje symboliczną *nazwę zasobu*, które są odwzorowywane na faktyczne położenie sieciowe przez scentralizowany *serwer nazw*.

- są różne protokoły RFS

  Początkowo (wersja 1.0) używano modelu zdalnej funkcji systemowej, gdzie serwer odtwarzał środowisko klienta. Następnie (wersja 2.0) wprowadzono zestawy zleceń bezpośrednio odpowiadającym operacjom na v-węzłach. Na początku klient i serwer muszą uzgodnić, którą wersję obsługują.



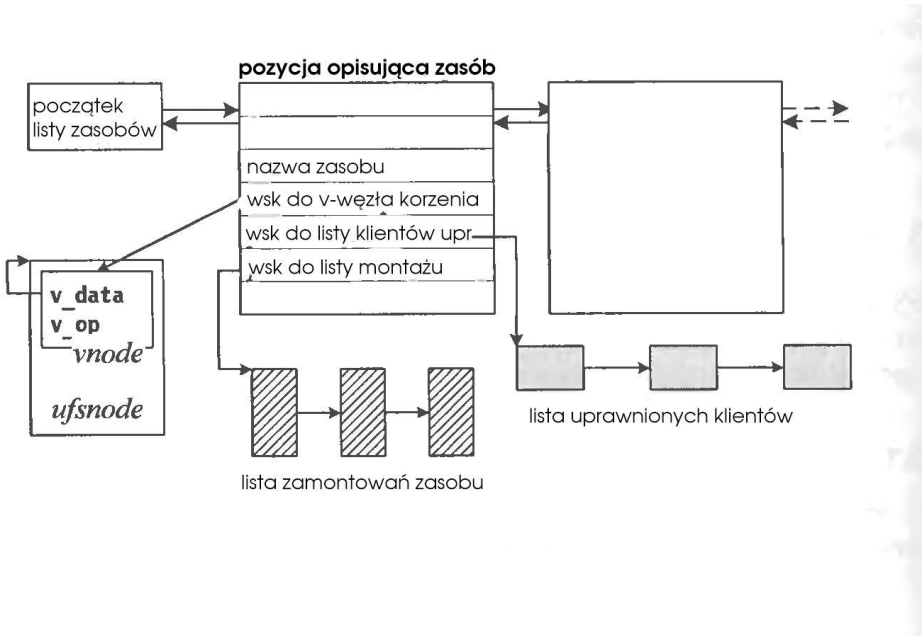
Komunikacja w RFS.

Implementacja systemu RFS

1. Montowanie

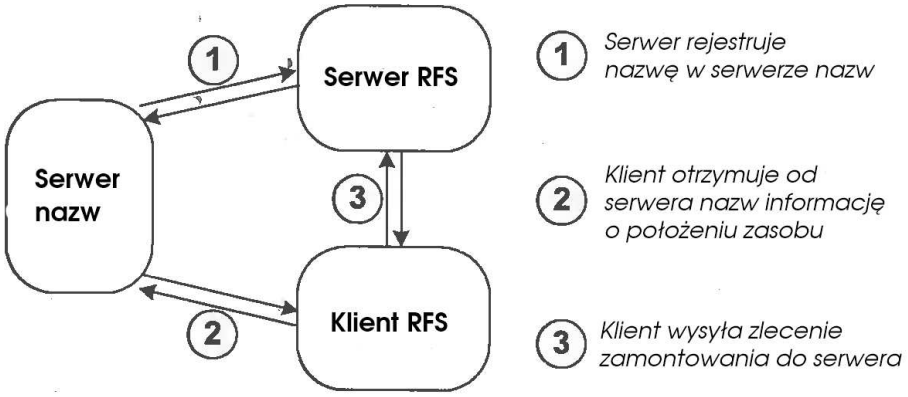
   Operację montowania obsługuje się oddzielnie za pomocą *montowania zdalnego*. Używany jest wtedy serwer nazw.

   Aby klienci mogli zamontować katalog, serwer musi go najpierw wyeksportować, robi to przy użyciu funkcji *advfs* (w nowszych wersjach podfunkcji funkcji *rfsys*). Wywołanie tej funkcji stworzy w liście zasobów jądra odpowiednią pozycję dla zadanego katalogu.



Lista zasobów w systemie RFS.

Teraz klient może wywołać polecenie *mount*, w którym najpierw zostanie odpytany serwer nazw o położenie pliku w sieci, a potem zostanie wysłane do zadanego serwera żądanie zamontowania danego pliku bądź katalogu. Jeśli klient ma prawo do korzystania z danego zasobu serwer zwróci identyfikator zamontowania. Klient z kolei, kończy montowanie poprzez konfigurację v-węzła, gdzie zostaje zapamiętana struktura *deskryptor przesyłu* (ang. send descriptor). W deskryptorze przesyłu pamiętany jest uchwyt pliku i informacje o obwodzie wirtualnym.



Montowanie systemu plików RFS.

2. Klient i serwer RFS

   Klient może odwoływać się do pliku RFS za pomocą nazwy ścieżkowej bądź deskryptora pliku. Ponieważ protokół RFSv2 pozwala na montowanie innych systemów plików na katalogach RFS, to podczas analizy ścieżkowej trzeba analizować każdą składową osobno. Gdy klient uzyska już uchwyt pliku, będzie go zawsze przekazywać przy żądaniu operacji na pliku.

W serwerze utrzymywana jest dynamiczna pula procesów-demonów, które obsługują każde zlecenie osobno. Gdy przybywa nowe zlecenie tworzony jest nowy proces, gdy nie ma dostępnych. Serwer działa jako jeden lub kilka demonów, które wykonują się w trybie jądra. Demony są szeregowane tak jak inne procesy.

3. Sygnały

Ponieważ system RFS wspiera całą semantykę UNIXową wprowadzono również możliwość przesłania sygnału. Jądro klienta po otrzymaniu sygnału wysyła do serwera *zlecenie sygnału* (ang. signal request). Serwer po jego otrzymaniu przekazuje sygnał do właściwego demona.

4. Pamięć podręczna

Ze względu na wydajność system RFS wspiera buforowanie danych po stronie klienta. Ponieważ jednak semantyka systemu RFS jest UNIXowa, należało zadbać, aby pamięć podręczna klientów była zawsze aktualna. Osiągnięto to dwojako:

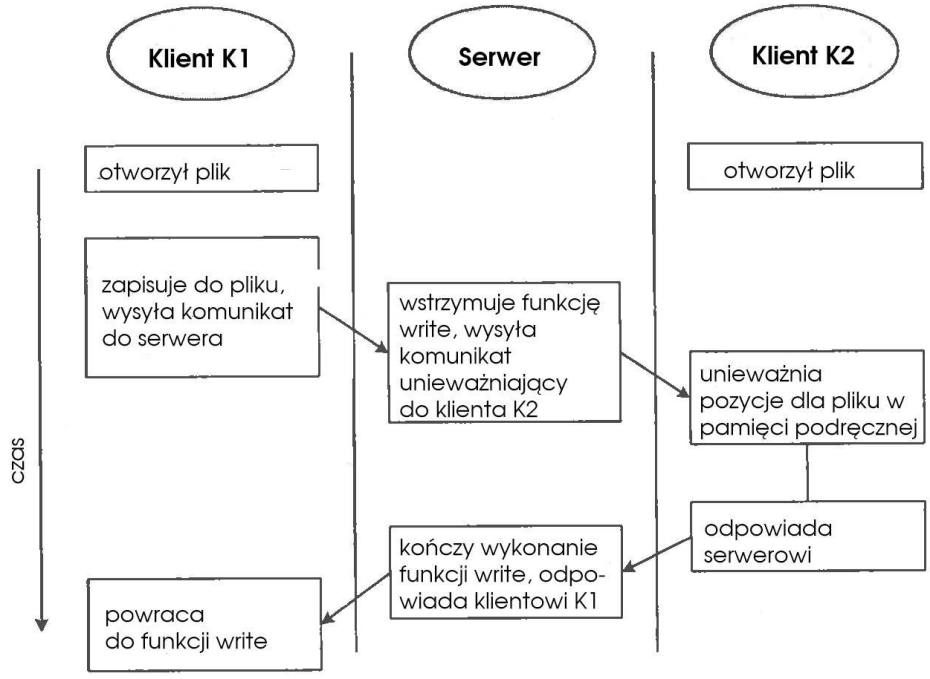
- zapisywanie natychmiastowe (ang. write-through)

  Choć pogarsza wydajność jest niezbędne do utrzymania spójności danych.

- czytanie wszystkich bloków z pamięci podręcznej lub żadnego

  Aby zagwarantować niepodzielność funkcji *read* dane przekazywane są z buforów podręcznych jedynie wtedy, gdy wszystkie bloki są po stronie klienta. Jeśli kawałka pliku brakuje, to cały żądany fragment jest pobierany z serwera.

O spójność danych pamięci podręcznej dba serwer (jeśli plik jest współdzielony przez wielu klientów) lub klient (jeśli mamy do czynienia ze współdzieleniem przez różne procesy tego samego klienta). Gdy jakiś klient wykona operację *write*, wtedy rozsyłany jest komunikat do wszystkich innych klientów mających ten plik otwarty, że buforowane dane są nieaktualne. Pozostali klienci czytają dane bezpośrednio z serwera, dopóki plik będzie modyfikowany. Natomiast klient mający plik zamknięty może używać buforowanych danych, gdy *numer wersji* pliku w buforze zgadza się z tym zwróconym przez serwer w wyniku operacji otwierania.



Zapewnianie spójności pamięci podręcznej w systemie RFS.

4.4 AFS

4.4.1 Wprowadzenie

Wstęp

Rozproszony system plików AFS stworzono na Canergie-Mellon University we współpracy z IBM (Information Technology Center ITC). ITC stworzyło kilka wersji, w tym ostateczną AFS 3.0. Dalsze prace prowadzono w Transarc Corporation, gdzie system AFS przekształcono w DFS, składową środowiska DCE.

Cele projektowe

Najważniejsze założenia dla systemu AFS:

- skalowalność
- zgodność z systemem UNIX (pliki binarne wykonywane bez zmian w klientach)
- jednolita niezależna od położenia przestrzeń nazw dla plików współdzielonych
- pliki można przenosić bez wyłączania systemu
- awaria składowej sieci nie czyni całej niedostępną
- wydajność prównywalna z systemem podziału czasu

4.4.2 Omówienie systemu AFS

Architektura systemu AFS

W systemie AFS sieć podzielona jest na pewną liczbę niezależnych klastrów. W systemie AFS używa się serwerów dedykowanych. Komputer w sieci jest albo serwerem, albo klientem, nie może pełnić obu funkcji. Do każdego klastra należy pewna liczba kllientów oraz serwer, który przechowuje pliki dla tych klientów. Chociaż użytkownicy mają dostęp do wszystkich plików w sieci, najszybciej otrzymają te, które znajdują się na serwerze należącym do tego samego segmentu sieci.
Aby zwiększyć wydajność w systemie AFS zastosowano „agresywne" techniki cache'owania po stronie klientów. Zapamiętywany jest cały plik lub 64-kilobajtowe kawałki. O ważność danych u klientów dbają serwery.
Aby zmniejszyć obciązenie serwerów analiza nazw plików odbywa się u klientów.

Organizacja pamięci

Pliki współdzielone są przechowywane przez zestaw serwerów *Vice*. Zorganizowane są one w logiczne jednostki *woluminy* (ang. volume). Wolumin jest to zestaw związanych ze sobą plików i katalogów, przy czym jest to jednostka różna od partycji dysku. Użycie woluminów ma następujące zalety:

- duży wolumin może być położony na wielu dyskach
- woluminy można przenosić bez wpływu na aktywnych użytkowników
- można tworzyć pliki większe niż rozmiar dysku
- można powielić woluminy tylko do odczytu
- można tworzyć kopie zapasowe woluminów

W systemie przestrzeń nazw jest jednolita i niezależna od położenia, co jest realizowane przez bazę danych położenia woluminów. Aby system działał sprawnie, baza danych znajduje się na każdym serwerze.
Dla każdej stacji roboczej klienta wymagany jest dysk lokalny, gdzie przechowywane są pliki lokalne i katalog do montowania hierarchii plików współdzielonych. Zatem każdy klient widzi tę samą współdzieloną przestrzeń nazw. Dodatkowo, dysk lokalny pełni funkcję pamięci podręcznej.
W początkowych wersjach systemu AFS stosowano semantykę sesji, potem zmieniono sprawdzanie ważności danych na każdy odczyt i zapis, co nadal nie spełnia jednak UNIXowej semantyki. Wprowadzono ją dopiero w systemie DFS, następcy systemu AFS.

Implementacja systemu AFS

Zestaw serwerów w systemie AFS nazywany jest *Vice*, zaś stacje klientów *Virtue*. W jądrze klienta działa *zarządca pamięci podręcznej AFS*, natomiast w serwerze większość jego funkcji wykonywanych jest na poziomie jądra przez procesy-demony.

1. Pamięć podręczna

   Implementacją operacji v-węzła u klienta zajmuje się zarządca pamięci podręcznej. O spójność danych w pamięciach podręcznych klientów dba serwer, przy użyciu mechanizmu *obietnicy powiadomienia* (ang. callback), co oznacza że jeśli inny klient zmodyfikuje plik, to pozostali są o tym powiadamiani. Wiąże się to oczywiście z faktem, iż serwer AFS musi być stanowy, by przechowywać informacje o otwartych plikach.

2. Analiza nazw ścieżkowych

   Analiza nazw ścieżkowych jest wykonywana po stronie klienta, aby odciążyć serwer. Klient pamięta więc dowiązania symboliczne, katalogi jak i pozycje położenia woluminów z bazy danych. Analiza odbywa się po jednej składowej na raz.

3. Bezpieczeństwo

   W systemie AFS korzysta się z systemu uwierzytelniania Kerberos opracowanego na MIT, przy czym uwierzytelnianie klientów odbywa się poprzez odpowiedzi na zaszyfrowane komunikaty serwera. Klucz do szyfrowania zna klient i serwer.
   Ponadto wykorzystywana jest *lista kontroli dostępu* ACL (ang. Access Control List) do poszczególnych katalogów.

4. Wnioski i wady

   Pomimo sukcesu pod koniec lat 80. system AFS miał kilka znaczących wad, które wymusiły zmiany w jego następcy, systemie DFS, a mianowicie:

   - stosunkowo niska wydajność klienta
   - problemy z implementacją stanowości serwera
   - rozbieżności z semantyką UNIXową i problemy z utrzymaniem spójności
   - znajomość przez klienta formatu zapisu katalogu na serwerze

4.5 DFS

4.5.1 Wprowadzenie

Wstęp

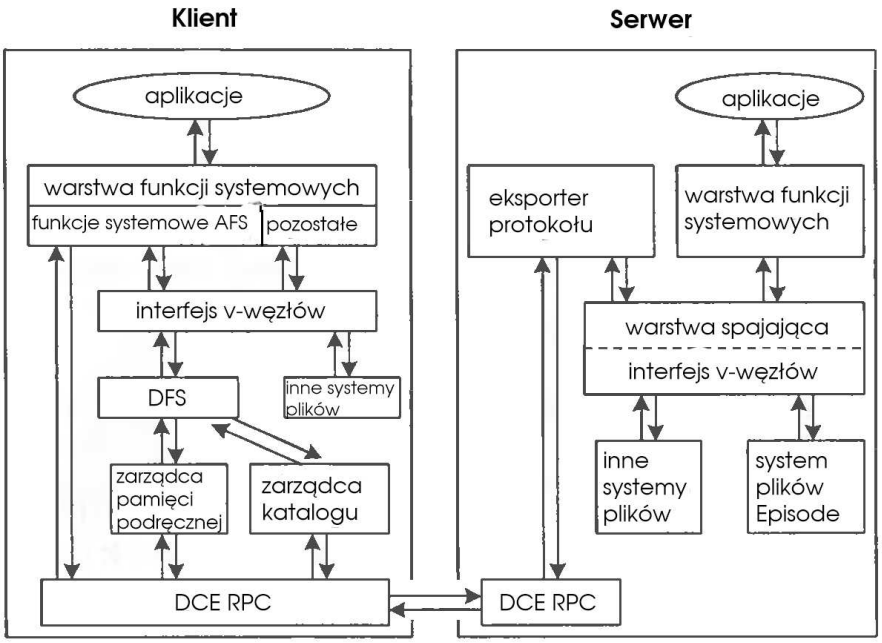
System DFS, będący kontynuacją systemu AFS, był tworzony od 1989r. przez firmę Transarc. System DFS został przyjęty przez Open Software Foundation za podstawę rozproszonego systemu plików w środowisku DCE (ang. Distributed Computing Environment). Główne ulepszenia w stosunku do systemu AFS to:

- ten sam komputer może być jednocześnie serwerem i klientem
- UNIXowa semantyka
- lepsza współpraca z innymi systemami plików (choć najlepiej współpracuje z systemem plików Episode, lokalnym dla serwerów DFS)

4.5.2 Omówienie systemu DFS

Architektura DFS

Architektura systemu DFS podobna jest do architektury AFS (stanowy model klient-serwer, lokalny system plików klienta jako pamięć podręczna, baza danych położenia woluminów).
Nowością w systemie DFS jest zastosowanie interfejsu v-węzłów w klientach i serwerach, co umożliwia lepszą komunikację z innymi systemami plików. Ponadto użytkownicy lokalni na serwerze mają dostęp do systemu plików DFS. Porozumiewanie pomiędzy klientami a serwerami odbywa się za pomocą protokołu DCE RPC (możliwość używania trybu synchronicznego i asynchronicznego, uwierzytelnianie przy użyciu mechanizmu Kerberos).



Architektura DFS.

Klienci działają podobnie jak w systemie AFS, znacząco różni się natomiast działanie serwerów, gdyż oddzielono od siebie protokół dostępu do plików z samym systemem plików. Dzięki temu możliwe jest eksportowanie rodzimego systemu plików serwera jak też dostęp do eksportowanego systemu plików aplikacjom na serwerze. Dodatkowe funkcje wspierające woluminy i listy kontroli dostępu dostarcza rozszerzony interfejs *vfs*, nazywany VFS+, który ma pełne wsparcie systemu Episode.
Eksporter protokołu zajmuje się obsługą zleceń klientów DFS i utrzymuje informacje o stanie każdego klienta, natomiast *warstwa spajająca* w interfejsie vfs ma za zadanie utrzymywanie zgodności pomiędzy eksporterem protokołu a innymi metodami dostępu do pliku.

1. Pamięć podręczna

   Zarządca żetonów na serwerze DFS, znajdujący się w warstwie spajającej interfejsu v-węzłów (właśnie tam, aby dobrze działało współistnienie z innymi metodami dostępu do pliku np. przez NFS), zajmuje się utrzymywaniem spójności pamięci podręcznej klientów, a tym samym wspiera UNIXową semantykę dostępu do plików. Są 4 rodzaje żetonów, każdy związany z innymi operacjami na plikach:

   - żetony danych

     Są ich 2 rodzaje: odczytu i zapisu. Jeśli np. klient ma żeton zapisu, to może modyfikować dane w swojej pamięci podręcznej bez konieczności przesyłania ich do serwera.
   - żetony stanu

     Są 2 rodzaje: odczytu i zapisu. Dotyczą atrybutów plików będących w pamięci podręcznej. Jeśli np. klient ma żeton zapisu, to inni klienci nie mogą zmienić atrybutów pliku ani nic do niego zapisać.
   - żetony blokad

     Klient posiadający dany żeton ma gwarancję, że inny klient nie założy konfliktowej blokady.

- żetony otwarcia

  Są 4 rodzaje: odczytu, zapisu, wykonania i zapisu wyłącznego. Jeśli np. klient ma żeton wykonania to ma gwarancję, że inny klient nie zmodyfikuje pliku.

2. Inne usługi

   Dodatkowe usługi oferowane przez system DFS oprócz eksportera protokołu i zarządcy pamięci to:

   - baza danych położenia zestawów plików

     Jak w AFS globalna replikowalna baza danych położeń woluminów.

   - serwer zestawu plików

     Implementacja operacji na zestawach plików, czyli np. migracji.

   - serwer uwierzytelniający

     Realizacja uwierzytelniania mechanizmu Kerberos.

   - serwer zwielokrotniania

     W DFS można replikować (zwielokrotniać) zestawy plików. Są 2 formy zwielokrotniania: wersji (klienci jawnie uaktualniają repliki) oraz planowane (automatyczne uaktualnianie).

3. Wnioski

   System DFS ma wiele udogodnień i korzystnych założeń, które jednak są trudne i skomplikowane w implementacji np. w porównaniu z protokołem NFS (zwłaszcza wcześniejsze wersje).

4.6 Źródła

Dalsze informacje na dany temat można znaleźć:

1. Na stronie **http://www.nfs4.org**
2. **http://www.nfs4.org**
3. U.Vahalia *Jądro systemu UNIX* rozdział 10. Dokładne omówienie NFSv2 i rozszerzenie o NFSv3. Opis systemu RFS oraz AFS i DFS.
4. S.Pate *UNIX Filesystems: Evolution, Design and Implementation* rozdział 13.

Rozdział 5

Systemy plików specjalnego przeznaczenia

5.1 Streszczenie

Niniejsza część prezentacji opisuje różnorodne systemy plików specjalnego przeznaczenia, m.in systemy plików tymczasowych, pseudosystemy plików. Główny nacisk został położony na pokazanie ogólnych idei funkcjonowania tych systemów. Uwzględniono także niektóre niekoniecznie będące popularne w użyciu, ale nietypowe pomysły.

5.2 Tymczasowe systemy plików

Wiele programów (np. kompilatory i zarządcy okienek) intensywnie używa plików tymczasowych do zapisywania wyników pośrednich faz wykonania. Pliki te są tworzone na krótko (usuwane przy zakończeniu działania aplikacji) i nie muszą być trwałe (nie muszą przetrwać awarii komputera). Dlatego potrzebny jest system plików z szybkim tworzeniem i dostępem do plików.

5.2.1 Wykorzystanie pamięci podręcznej

Jeśli nic nie zrobimy, to i tak nie jest źle. Jądro wykorzystuje buforową pamięć podręczną, aby opóźnić zapis danych na dysk. Pliki tymczasowe zazwyczaj są usuwane zanim zajdzie potrzeba zapisania ich na dysk.

Tworzenie plików tymczasowych nadal jest jednak wolne, bo zadania te wymagają wykonania synchronicznych operacji na metadanych (np. na katalogach).

5.2.2 *RAM-dyski*

RAM-dyski są systemami plików, które „udają” fizyczne partycje dysku, lecz w rzeczywistości umieszczone są całkowicie w pamięci operacyjnej. Na RAM-dysku można zainstalować dowolny system plików w standardowy sposób (`newfs()`).

Główna wadą takiego rozwiązania jest poświęcenie dużej ilości pamięci na *RAM-dysk* i słabe wykorzystanie zasobów systemowych. Ponadto, przy projektowaniu tego systemu nie zwrócono uwagi na problem „potrójnych kopii” danych: gdy wykonujemy operację `read()` dane są kopiowane z dysku do bufora pamięci podręcznej, a następnie do RAM-dysku.

5.2.3 System plików *mfs*

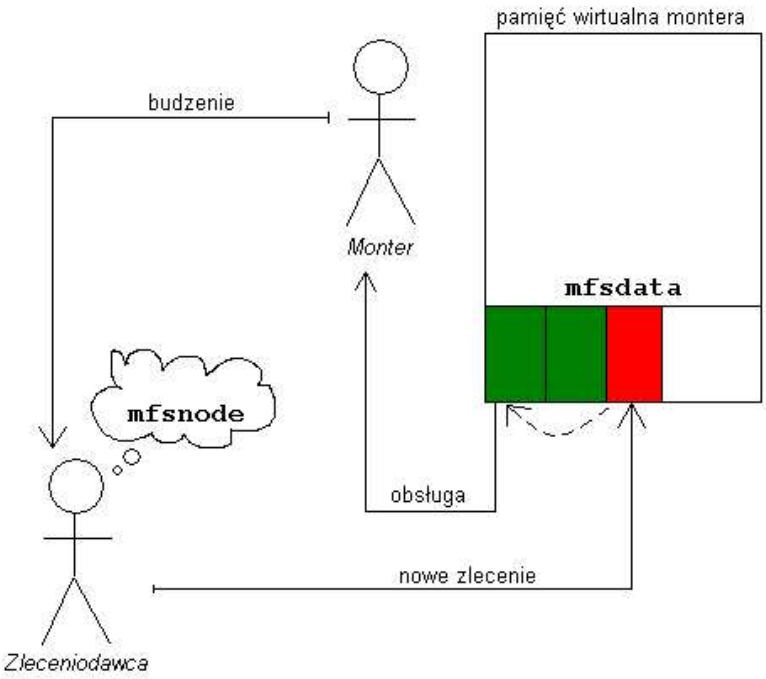
Mfs - Memory file System opracowano na Uniwersytecie w Berkeley. Cały system plików tworzy się w

wirtualnej przestrzeni adresowej procesu, który wykonał operację jego montowania. Proces ten nie wraca z funkcji `mount`, ale działa jak serwer operacji wejścia-wyjścia dla tego systemu plików.

Gdy proces chce wykonać operację we-wy na pliku systemu *mfs*:

1. Proces wstawia zlecenie do kolejki `mfsdata`
2. i budzi proces montujący i usypia. (Proces zna pid procesu montującego, który jest zapisany w strukturze `mfsnode`)
3. Proces montujący spełnia żądanie (kopiuje dane)
4. i budzi proces zleceniodawcy.

Strony systemu *tmpfs* rywalizują o pamięć i mogą być przeniesione do obszaru wymiany. Pozwala to na obsługę plików tymczasowych o rozmiarze większym, niż rozmiar pamięci operacyjnej.



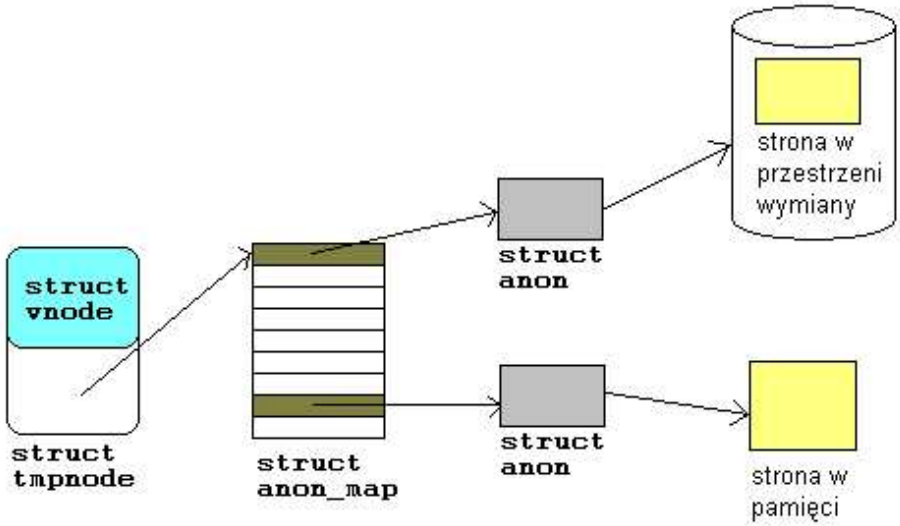
Schemat działania systemu plików *mfs*.

Główną wadą tego systemu plików jest konieczność wykonywania dwóch przełączeń kontekstu przy każdej operacji we-wy. Badania wykazały jednak, iż ten system jest dwukrotnie bardziej efektywny przy operacjach we-wy oraz wielokrotnie szybszy przy przy wykonywaniu dużej liczby operacji na metadanych.

5.2.4 System plików *tmpfs*

System plików *tmpfs* opracowano w Sun Microsystems.

Tmpfs jest zaimplementowany w całości w jądrze. Wszystkie metadane plików przechowywane są w niestronicowanej pamięci przydzielanej ze sterty. Dane z plików znajdują się w pamięci stronicowanej i są reprezentowane jako *strony anonimowe* (ang. *anonymous pages*). Każda strona jest odwzorowana za pomocą *obiektu anonimowego* (ang. *anonymous object*, `struct anon`), w którym pamięta się położenie strony. Każdy *tmp-węzeł* zawiera wskaźnik do *mapy anonimowej* (ang. *anonymous map*, `struct anon_map`) wszystkich obiektów anonimowych pliku.



Schemat działania systemu plików *tmpfs*.

Proces może odwzorować plik systemu *tmpfs* za pomocą funkcji `mmap` w swoją przestrzeń adresową. Daje to bardzo szybki i bezpośredni dostęp do danych w plikach tymczasowych.

5.2.5 System plików z opóźnieniem montowania

System plików z *opóźnieniem montowania* (ang. *File System by Delay Mount Option*) jest udoskonaleniem najprosteszego wariantu szybkiej obsługi plików plików tymczasowych - korzystania z pamięci podręcznej.

W funckji `mount` udostępnia się dodatkową opcję *opóźniania* (ang. *delay*), która powoduje ustawienie specjalnego znacznika dla stron z tego systemu. Programy uaktualniające synchronicznie dysk (np. funkcja systemowa `sync`) trzeba tak zmodyfikować, by nie usuwały strony od razu, ale najpierw zmieniały jej stan na *dirty* opóźniając w ten sposób zapis strony na dysk.

To rozwiązanie jest efektywne, ale wymaga modyfikacji programach zapisujących dane na dysk.

5.3 Pseudosystemy plików

5.3.1 System plików `/proc`

Po raz pierwszt system plików `/proc` został włączony do 8-ej edycji Uniksa, a później do SVR4. W zamyśle, miał zastąpić niewygodną funkcję `ptrace` dając dostęp do przestrzeni adresowej procesu, na której można byłoby działać jak na zwykłym pliku. System rozwinął się w elegancki interfejs do przestrzeni adresowej procesów.

Implementacja

Wczesne implementacje. We wczesnych implementacjach każdy proces reprezentowano jako plik w katalogu `/proc`. Nazwą pliku był pid procesu, rozmiarem - rozmiar przestrzeni adresowej procesu, prawami - prawa właściciela procesu. Na pliku można było wykonywać operacje `read()`, `write()`, `lseek()` i `ioctl()`.

Zmiany w późniejszych impelemtacjach. Zmiany w implementacji były powodowane przede wszystim wprowadzeniem do Uniksa wielowątkowości na poziomiwe użytkownika (*lwp - light weight process*). Każdy proces w systemie jest reprezentowany jako podkatalog katalogu `/proc`.

Zawartość katalogu `proc`

Każdy katalog w `/proc` zawiera m.in. pliki:

Nazwa	Tryb	Struktura (patrz `procfs.h`)	Informacje w pliku
status	r	`pstatus`	Stan procesu.
psinfo	r	`psinfo`	Informacje dla `ps`.
ctl	w	–	Umożliwia wykonywnie operacji sterujących.
map	r	zaw. `prmap`	Mapa adresów wirtualnych procesu.
as (adress space)	rw	–	Przestrzeń wirtualna procesu.
sigact	r	zaw. `sigaction`	Informacje związane z obsługą sygnałów.
cred (credentials)	r	`prcred`	Informacja uwierzytelnijąca.
object	kat.	–	Zawiera o jednym pliku dla każdego obiektu odwzorowywanego na przestrzeń adresową procesu.
lwp	kat.	–	Zawiera informacje o wątkach - po jednym katalogu dla każdego wątku. Każdy taki katalog zawiera trzy pliki: **lwpstatus**, **lwpsinfo** oraz **lwpctl** (znaczenie analogiczne).

Operacja na plikach **ctl** i **lwpctl**:

`PCSTOP`	Zatrzymuje wszystkie wątki.
`PCWSTOP`	Czeka na zatrzymanie wszystkich wątków procesu.
`PCRUN`	Wznawia zatrzymane procesy lekkie.
`PCKILL`	Wysyła podany sygnał do procesu.
`PCSENTRY`	Nakazuje zatrzymanie się wątku przy wejściu do określonych funkcji systemowych.
`PCSEXIT`	Nakazuje zatrzymanie się wątku przy wyjśiu z określonych funkcji systemowych.

Przykład

Przykładowy wydruk dla procesu `bash` (w Linuxie 2.4.21):

```
[ela@karamba /]$ ps
  PID TTY          TIME CMD
13762 pts/6    00:00:00 bash
14065 pts/6    00:00:00 ps

[ela@karamba /]$ ll /proc/13762
razem 0
```

```
-r--r--r--    1 ela      ela             0 sty  1 04:19 binfmt
-r--r--r--    1 ela      ela             0 sty  1 04:19 cmdline
lrwxrwxrwx    1 ela      ela             0 sty  1 04:19 cwd -> /proc/
-r--------    1 ela      ela             0 sty  1 04:19 environ
lrwxrwxrwx    1 ela      ela             0 sty  1 04:19 exe -> /bin/bash*
dr-x------    2 ela      ela             0 sty  1 04:19 fd/
-r--r--r--    1 ela      ela             0 sty  1 04:19 maps
-rw-------    1 ela      ela             0 sty  1 04:19 mem
-r--r--r--    1 ela      ela             0 sty  1 04:19 mounts
lrwxrwxrwx    1 ela      ela             0 sty  1 04:19 root -> //
-r--r--r--    1 ela      ela             0 sty  1 04:19 stat
-r--r--r--    1 ela      ela             0 sty  1 04:19 statm
-r--r--r--    1 ela      ela             0 sty  1 04:19 status

[ela@karamba /]$ cat /proc/13762/cmdline
/bin/bash
```

Pliki katalogu `/proc` nie są fizycznymi plikami z danymi. Operacje na tych plikach są tłumaczone na odpowiednie akcje podejmowane na procesach lub ich przestrzeniach adresowych. Wiele z tych plików jest interfejsem do pewnych struktur procesu, np. `pstatus`, które opsane są w pliku `procfs.h`.

Pseudosystem plików `/proc` jest bardzo użyteczny w szczególności w programach odpluskwiających i śledzących (dzięki plikom **ctl**).

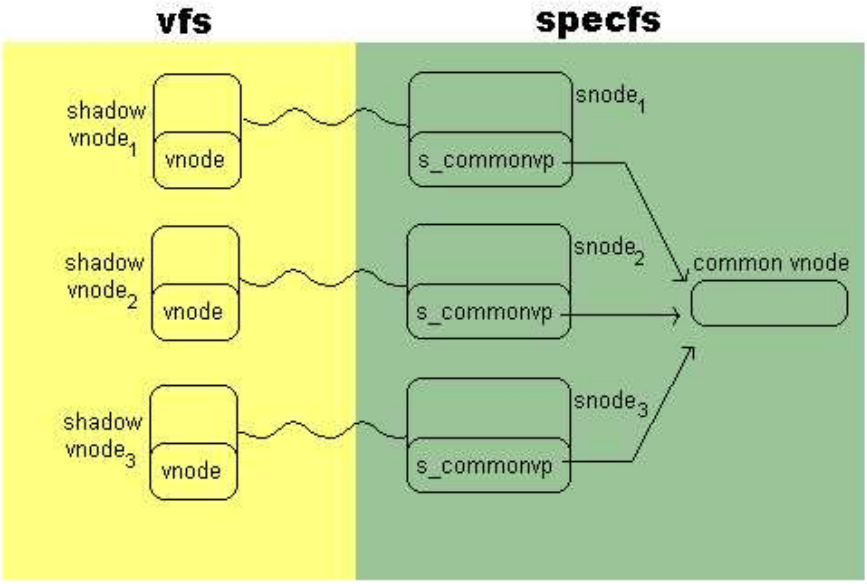
5.3.2 System plików *specfs*

Powstanie. Gdy w Uniksie był tylko jeden system plików, zaczęło powstawać bardzo wiele różnych rodzajów plików, w szczególności plików urządzeń. Powodowało to problemy z narzutem czasowym przy dostępie, narzutem pamięciowym (ten sam blok w dwóch cache'ach). Pseudosystem plików *specfs*, wprowadzony po raz pierwszy do SVR4, rozwiązał te problemy.

Interfejs urządzeń. System plików *specfs* udostępnia jednolity interfejs do urządzeń. System ten jest niewidoczny dla użytkowników i nie może być montowany. Eksportuje interfejs, który może być stosowany przez dowolny system plików wspierający pliki specjalne. Jego podstawowym znaczeniem jest przechwycenie operacji we-wy do plikw urządzeń i przetłumaczenie ich na wywołania odpowiednich programów obsługi tych urządzeń.

Gdzie jest haczyk? Na pierwszy rzut oka wydaje się to ławtwym zadaniem – dla każdego pliku możemy sprawdzić czy jest on plikiem urządzenia i , jeśli tak, pobrać numery urządzenia. Wówczas możemy znaleźć podprogramy obsługi za pomoca tablic rozdzielczych urządzeń. Problemy pojawiają się, gdy z tym samym urządzeniem związanych jest wiele plikow urządzeń i korzysta z nich wielu użytkowników. Jądro musi wiedzieć, które vwęzły reprezentują to samo urządzenie.

Implementacja. *Specfs* tworzy *v-węzeł przesłaniający* (ang. *shadow vnode*) dla każdego pliku urządzenia. Jego dane w warstwie zależnej od systemu plików reprezentuje s-węzeł (ang. *snode*). Operacja wyszukiwania pliku urządzenia daje w wyniku wskaźnik do v-węzła przesłaniającego, a nie do węzła rzeczywistego. Węzeł rzeczywisty można uzyskać za pomocą operacji `vop_realvp()`. S-węzeł ma pole o nazwie `s_commonvp`, które wskazuje na *węzeł wspólny* dla tego urządzenia. Może na wiego wskazywać wiele węzłów, ale jednemu urządzeniu odpowiada jeden węzeł wspólny. Wszystkie operacje wymagające synchronizacji oraz odczyty i zapisy do urządzenia kierowane są przez węzeł wspólny.



Schemat działania systemu plików *specfs*.

5.3.3 System plików procesora

System plików procesora (ang. *Processor File System*) udostępnia interfejs do poszczególnych procesów na komputerze wieloprocesorowym. Jest zamontowany w katalogu `/system/processor` i ma po jednym pliku dla każdego procesora. Nazwy plików są numerami procesorów, są tylko do odczytu. Zawierają następujące informacje:

- stan (aktywny / odłączony)
- typ procesora
- szybkość (MHz)
- rozmiar pamięci podręcznej (KB)
- informacja czy zawiera jednostkę arytmetyczną
- podprogramy obsługi zw. z procesorem
- czas ostatniej zmieny stanu

Ponadto system plików procesora zawiera plik **ctl**, do którego może pisać tylko superużytkownik. Zapis do tego pliku powoduje wykonanie operacji sterujących na procesorze, np. odłączenie go.

Informacja o procesorze w Linuksie

```
[ela@karamba /]$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 6
model           : 10
model name      : AMD Athlon(tm) XP 2500+
stepping        : 0
cpu MHz         : 1830.035
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
  mca cmov pat pse36 mmx fxsr sse syscall mmxext 3dnowext 3dnow
bogomips        : 3643.80
```

5.3.4 System plików TFS

Powstanie. System plików TFS (ang. *Translucent File System* (Półprzezroczysty)) powstał w Sun Microsystems. Celem było stworzenie systemu do współdzielenia plików bez duplikowania ich. System wspiera tworzenie dużego oprogramowania:

- utrzymywanie całości (wersjonowanie, kontrola konsolidacji)
- utrzymywanie prywatnych hierarchii odizolowanych od innych

Współdzielony katalog. TFS implementuje *warstwy* projektu, każda warstwa jest fizycznym katalogiem. Warstwy są połączone w listę za pomocą plików ukrytych – *dowiązań wyszukiwania* (ang. *search links*). Pliki widoczne w katalogu TFS są sumą plików ze wszystkich warstw. Dodatkowo TFS umożliwia tworzenie warstw wariantowych, np. dla różnych platform.

Prywatne hierarchie. Każdy użytkownik może pobrać pliki z dowolnej warstwy. Domyślnie dostęp jest do najnowszej, zewnętrznej warstwy. Jeśli potrzebne są wcześniejsze warstwy, trzeba przejść jawnie po łańcuchu wyszukiwania. Gdy użytkownik zmodyfikuje plik w swojej hierarchii, zmiany stają się widoczne dla innych w momencie zapisania go do systemu plików (*kopiowanie przy zapisie*). Plik zostanie zapisany do najbardziej zewnęrznej warstwy.

Wydajność. System TFS nie jest bardzo wydajny, gdyż pobranie pliku wymaga zwykle przeszukania kilku warstw. Aby przyspieszyć wyszukiwanie korzysta się z pamięci podręcznej.

5.3.5 System plików *namefs*

Powstanie. System plików *namefs* został wprowaddzony do SVR4. Jego celem jest umożliwienie związania pliku z otwartym *nazwanym strumieniem* (ang. *named STREAM*) i sworzenie w ten sposób „kanału komunikacyjnego".

Implementacja. Wprowadzono funkcję `fattach()`, która monutuje system *namefs* na pliku. Związuje się „głowę" strumienia z punktem montowania, co pozwala przekierowywać operacje we-wy na pliku do strumienia.

Przykład, Za pomocą `IP STREAM` można implementować protokł TCP/IP lub UDP/IP.

5.3.6 Systemy plików *fifofs, fdfs, swapfs, lofs*

fifofs

Powstanie. System plików *fifofs* został wprowadzony do SVR4, działa analogicznie do *specfs*, lecz dla łącz nazwanych (kolejek FIFO).

Implementacja Gdy użytkownik próbuje się „dobrać" do pliku, który jest łączem nazwanym, dochodzi do wywołania funkcji `fifovp()`, która zwraca v-węzeł systemu *fifofs* i odpowiednio inicjalizuje pole `v_op` na `fifo_vnodeops`.

fdfs

Implementacja. System plików *file descriptor filesystem*, zwykle montowany na `/dev/fd`, jest wygodnym interfejsem do otwartych plikow (proces ma dostęp do swoich otwartych plików). Używa się tego głównie w językach skryptowych, np. bash, perl, awk, itd.

Przykład. Dwa wywołania:

```
fd = open ("/dev/fd/n", mode);
fd = dup (n)
```

są identyczne.

swapfs

Swapfs został wprowadzony do systemu operacyjnego Solaris, aby rozwiązać problem z nieużywaną pamięcią operacyjną (ponieważ każda strona pamięci miała mieć odpowiadającą stronę przestrzeni wymiany; problem powstawał, gdy przestrzeń wymiany była za mała).

System plików *swapfs* umożliwia zamontowanie wirtualnej przestrzeni wymiany, która obejmowała:

- zwykłą przestrzeń wymiany
- fragment dysku - strony anonimowe, dla których *swapfs* dostarcza nazwy

lofs

System plików *Loopback File System* tworzy wirtualny system pliów, którym można nakryć lub zduplikować pliki. Są one wówczas dostępne z obu ścieżek.

5.4 Zadania

5.4.1 Treść

1. Do jakiego innego systemu plików jest podobny system plików procesora?
2. Jakie są zalety stosowania systemów plików tymczasowych?

5.4.2 Odpowiedzi

1. System plików procesora jest odpowiednikiem systemu `/proc` tylko, że zamiast interfejsu przestrzeni procesu daje namiastkę interfejsu procesora.
2. Mamy możliwość niezwykle szybkiego tworzenia plików i dostępu do nich, programy korzystające intensywnie z tego rodzajów plików będą działały bardziej wydajnie.

5.5 Źródła

Informacje dotyczące systemów plików spejalnego przeznaczenia można szukać:

1. Dokumentacji systemu. (dla Linuxa – `/usr/src/linux/Documentation/`)
2. Podręczniku systemowym (`man`)
3. Prezentacje z zeszłych lat
4. Uresh Vahalia „Jądro systemu UNIX” - rozdz. 9

Rozdział 6

Złożone systemy plików

6.1 System plików z klastrami - *Sun-FFS*

Powstanie. Systemu plików rozszerzony o *klastrowanie* plików (ang. *files clustering*) został wprowadzony w systemie SunOS, a później włączony do SVR4 i 4.4BSD, nazywany jest również *Sun-FFS*. Jego celem było przyspieszenie wykonywania operacji wejścia-wyjścia poprzez grupowanie ich (ang. *clustering*).

Implementacja. Przydzielanie pamięci odbywa się tak samo jak w systemie FFS - po bloku i sprytnie (z przewidywaniem kolejnych alokacji). *Sun-FFS* ustawia współczynnik opóźnienia obrotowego na 0 i wykorzystuje nieużywane wówczas pole `maxconfig` superbloku do przechowywania żądanego rozmiaru klastra.

Odczyt. System stara się odczytać jak największy fragment klastra (ciągłego obszaru danych) na dysku. Zmodyfikowano funkcję `bmap()` tak, by pobierała logiczny numer bloku na dysku, a zwracała jego numer fizyczny i okreslała rozmiar ciągłego obszaru, który można wczytać (czego wcześniej nie robiła).

Zapis. Podprogram zapisujący strony na dysk `ufs_putpage` oczekuje aż nie nazbiera się cały klaster danych do zapisu sekwencyjnie. Korzystając z funkcji `bmap()` stara się je zapisać możliwie sekwencyjnie (jeżeli funkcja `bmap()` zwróci za mało, trzeba dzielić zapis na dwa, itd.)

Efektywność Badania wykazały, iż odczyty i zapisy sekwencyjne zostały przyspieszone dwukrotnie, szybkość dostępu bezpośredniego natomiast pozostała bez zmian.

6.2 Systemy plików z kroniką

6.2.1 Kronika

Cel. Aby zlikwidować część wad tradycyjnych systemów pligków w wielu nowoczesnych systemach zastosowano technikę *kronikowania* (ang. *journalling* lub *logging*). Kronika jest specjalnym plikiem przeznaczonym tylko do dopisywania. Zapisuje się ją sekwencyjnie, dużymi partiami, zwykle całą ścieżkę naraz. Kronika umożliwia szybkie postawienie systemu po awarii.

Rodzaje kronik Jest wiele rodzajów kronik:

Co zapisywać w kronice?	Są dwie możliwości zapisywania: 1. wszystkich modyfikacji 2. zmian wybranych metadanych (można wybrać tylko te operacje, które wpływają na spójność systemu)
Zapisywać operacje czy wartości?	Można zapisywać operacje albo ich wyniki.

Kronika jako dodatek czy zastąpienie	Można stosować kronikę jako uzupełanienie (*Sysytemy plików rozszerzone o kronikę* (ang. *Logging enhanced file systems*)) albo jako jedyną reprezentację systemu plików na dysku (*Systemy plików o strukturze kroniki* (ang. *log-structured file systems*)).
Kroniki z informacją do przywracanmia i anulowania	Są dwa typy kronik: • *tylko do przywracania* (ang. *redo-only*), w której zapamiętuje się tylko zmodyfikowane dane. Dla tej kroniki wymagane jest zachowywanie porządku zapisywania danych. • *do anulowania przywracania* (ang. *undo-redo*), w której zapamiętuje się zarówno stare, jak i nowe dane.
Odśmiecanie	Kronikę zwykle traktuje się jako plik cykliczny. Odśmiecać można wykonywać w czasie normalnego działania systemu lub za pomocą odrębnej operacji.
Grupowanie	Konieczny jest kompromis między rozmiarem zapisywanych porcji danych oraz częstością zapisu (więcej $\Rightarrow$ szybciej, częściej $\Rightarrow$ w razie awarii tracimy mniej danych).
Pobieranie danych	Konieczne jest zastosowanie efektywnego mechanizmu indeksowania, umożliwiającego odnalezienia w kronice dowolnego pliku (aby obsłużyć błędy braku strony).

Zalety. Zapisy na dysk dużych porcji - bardzo szybkie. Usuwanie skutków awarii jest również szybkie.

Wady. Odczyt jest wolniejszy (trzeba wyszukiwać ostatnio zmodyfikowane dane w kronice). W systemie, który działa już jakiś czas można z łatwością osiągnąć współczynnik trafień ponad 90%, jednak na początku, w wypadku bloków, które muszą być odczytane z dysku, trzeba zapewnić sposób odnajdowania ich w kronice.

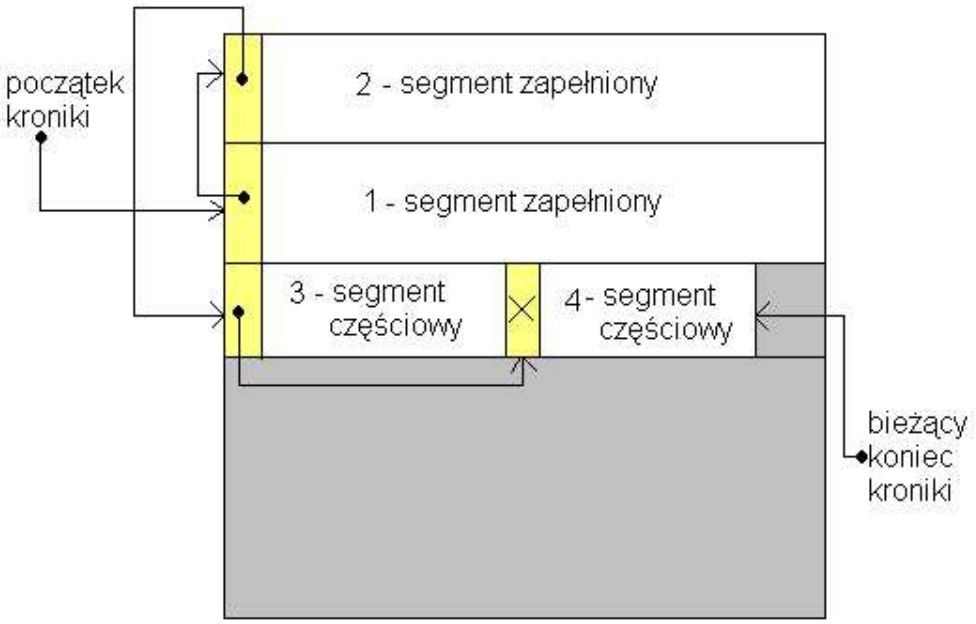
6.2.2 System o strukturze kroniki: BSD-LFS

Opis. System 4.4BSD, znany pod nazwą BSD-LFS, zaprojektowano na podstawie wyników prac nad systemem *Sprite*. System BSD-LFS przeznacza cały dysk na kronikę, która jest jedyna trwałą reprezentacją systemu plików. Wszystkie zapisy umieszcza się na końcu kroniki, a odśmiecanie wykonuje proces *cleaner*. Kronika podzielona jest na segmenty stałego rozmiaru (zwykle 500 KB) połączone w listę.

Segmenty częściowe

Nie zawsze można zapisywać całkowicie wypełnione segmenty (np. gdy zabraknie pamięci podręcznej), dlatego wprowadza się *segmenty częściowe* (ang. *partial segments*). Każdy segment częściowy ma *nagłówek*, który zawiera następujące informacje:

- sumy kontrolne
- adresy dyskowe każdego iwęzła w segmencie częściowym
- numer i-węzła i jego wersji oraz numery logiczne bloków dla każdego pliku, który ma bloki danych położone w segmencie
- czas utworzenia, znaczniki, etc.



Schemat działania systemu plików *BSD-LFS*.

Mapa i-węzłów i tablica użycia segmentów

Zachowano strukturę katalogów i i-węzłów. Utrzymuje się *mapę i-węzłów* (ang. *inode map*), która przechowuje adresy dyskowe wszystkich i-węzłów. Mapa ta jest w pamięci fizycznej, ale zapisuje się ją do kroniki w okresowych *punktach kontrolnych* (ang. *checkpoints*).

System utrzymuje także *tablicę użycia segmentów* (ang. *segment usage table*), która przechowuje liczbę nieprzestarzałych bajtów i czas ostatniej modyfikacji dla każdego segmentu.

Operacje wejścia-wyjścia

Zapis. System BSD-LFS zbiera zabrudzone dane tak długo, aż będzie ich wystarczająco do zapełnienia segmentu. Wówczas wszystkie zabrudzone dane są zapisywane, a kronika zawiera wszystkie dane do pełnego usunięcia skutków awarii.

Odczyt. W celu umożliwienia sprawnego odczytu, potrzebna jest duża pamięć podręczna, tak aby większość zleceń można było realizować bez dostępu do dysku. W pamięci podręcznej bloki są identyfikowane i haszowane za pomocą v-węzła i logicznego numeru bloku. Ważna jest także obsługa błędu braku strony w sensownym czasie. I-węzły wyszukuje się w mapie i-węzłów.

Usuwanie skutków awarii

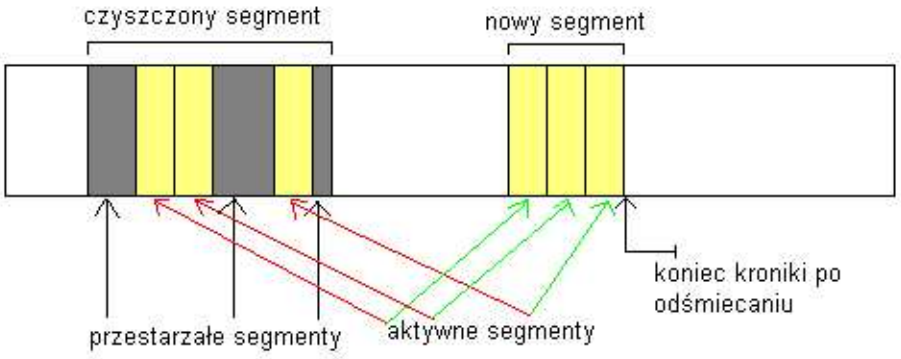
Usuwanie skutków awarii przebiega następująco:

1. odnalezienie ostatniego punktu kontrolnego
2. zainicowanie mapy i-węzłów i tablicy użycia segmentów
3. przeglądanie i wykonywanie tych zmian, które nastapiły po zapisaniu ostatniego punktu kontrolnego.
4. suma kontrolna w nagłówku zapewnia, że segment częściowy jest kompletny. Jeśli nie, to jest to ostatni segment i jest on odrzucany.

Cleaner

Zadaniem procesu *cleaner* jest gromadzenie aktywnych danych z segmentów, przenoszenie ich w nowe miejsce i w ten sposób odzyskiwanie semgnetów. *Cleaner*

1. wybiera segment do czyszczenia (korzysta z tablicy użycia segmentów)
2. wczytuje segment do czyszczenia do swojej przestrzeni adresowej
3. wybiera i-węzły i bloki, które są aktywne (korzysta z mapy i-węzłów)
4. zapisuje je w innym segmencie częściowym
5. zaznacza segment czyszczony jako wolny



Schemat działania procesu *cleaner* w systemie plików *BSD-LFS*.

6.2.3 Systemy z kronikowaniem metadanych

Zyski i wady. Kronikowanie tylko metadanych:

- upraszcza implementację, gdyż nie trzeba zmieniać implementacji operacji, które nie zmieniają struktury systemu plików. Jedyne, co trzeba zrobić, to zapisywać do kroniki.
- umożliwia szybkie odtworzenie systemu po awarii
- przyspiesza operacje na metadanych dzięki gromadzeniu operacji, które wykonują wiele zmian metadanych (np. *mkdir*) w jeden zapis w kronice i zmniejszeniu liczby synchronicznych zapisów na dysk. Ta oszczędność powinna niwelować narzut związany z prowadzeniem kroniki.
- nie ogranicza strat w danych z plików

Organizacja kroniki. Jest wiele wariantów organizackji kroniki:

- **System *Cedar*.** Plik zwykły, cykliczny, o ustalonym rozmiarze.
- **System *Calaveras*.** Wszystkie systemy plików współdzielą jedną kronikę, która znajduje się na osobnym dysku.
- **System *Veritas.*** Kronika oddzielona od systemu plików, może być na osobnym dysku.

Badania. System plików z kroniką jest znacznie szybszy, gdy wykonuje się wiele operacji na metadanych. W przeciwnym przypadku szybkość jego działania jest porównywalna do zwykłych systemów plików.

6.2.4 System plików *Episode*

Powstanie. System plików *Episode* (ang. *Episode File System*) powstał jako rozwinięcie systemu plików *Andrew* (AFS). Stał się on lokalnym systemem plików w rozproszonym środowisku obliczeniowym (ang. DCE - *Distributed Computing Environment*).

Możliwości. System udostępnia kilka złożonych możliwości, których nie mają tradycyjne uniksowe systemy plików. Są to m.in:

- rozszerzone mechnizmy zabezpieczeń
- wsparcie dla dużych plików
- kronikowanie
- oddzielenie abstrakcji dotyczących pamięci masowej od logicznej struktury systemu plików

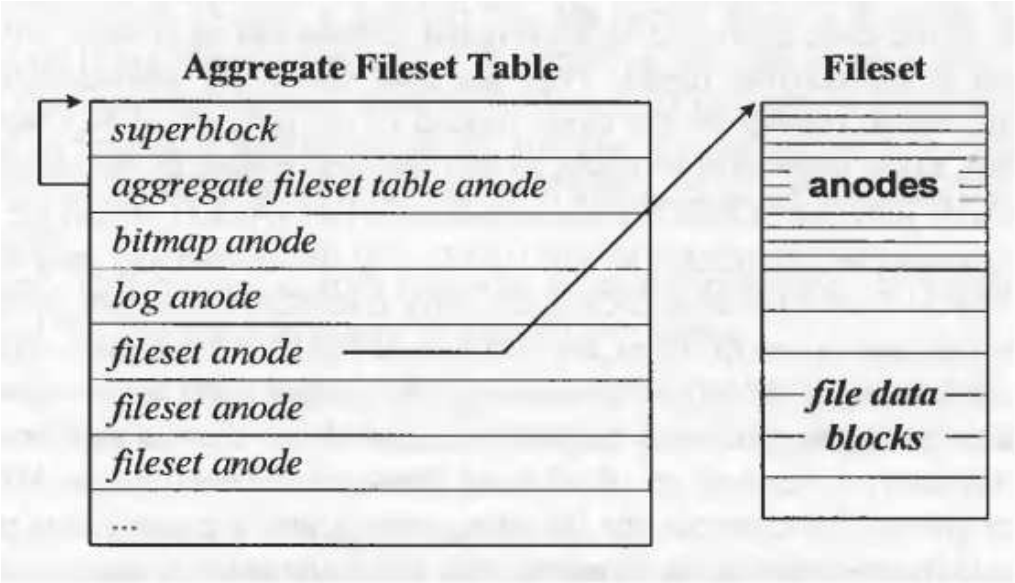
Organizacja. Do systemu plików *Episode* wprowadzono pojęcia:

- *agregaty* (ang. *aggregates*) - zbiór bloków na dysku, logicznie ciągły (uogólnienie partycji). Fizycznie może składać się z kilku partycji. Zatem umożliwia tworzenie plików o rozmiarze większym niz rozmiar dysku.
- zestawy plików (ang. *filesets*) - logiczny system plików, drzewo katalogów. Jeden agregat może zawierać wiele zestawów plików, które można niezaleznie montować, demontować, przenosić do innego agregatu.
- a-węzły (ang. *anodes*) - odpowiedniki i-węzłów. W *Episode* mamy dokładnie po jednym a-węźle dla każdego pliku i każdego kontenera.
- kontenery (ang. *containers*) - obiekt przechowujący dane. Składa się z bloków. Każdy zestaw plików mieści się w kontenerze, który gromadzi wszystkie metadane i dane tego zestawu. Konterery nie zajmują ciągłych obszarów w agregacie, mogą więc rosnąć i kurczyć się. Wszystkie a-węzły są umieszczone w *tablicy a-węzłów zestawu plików* (ang. *fileset anode table*) znajdującej się na początku kontenera. Za nimi znajduja się dane i bloki pośrednie. Każdy agregat ma trzy dodatkowe kontenery:

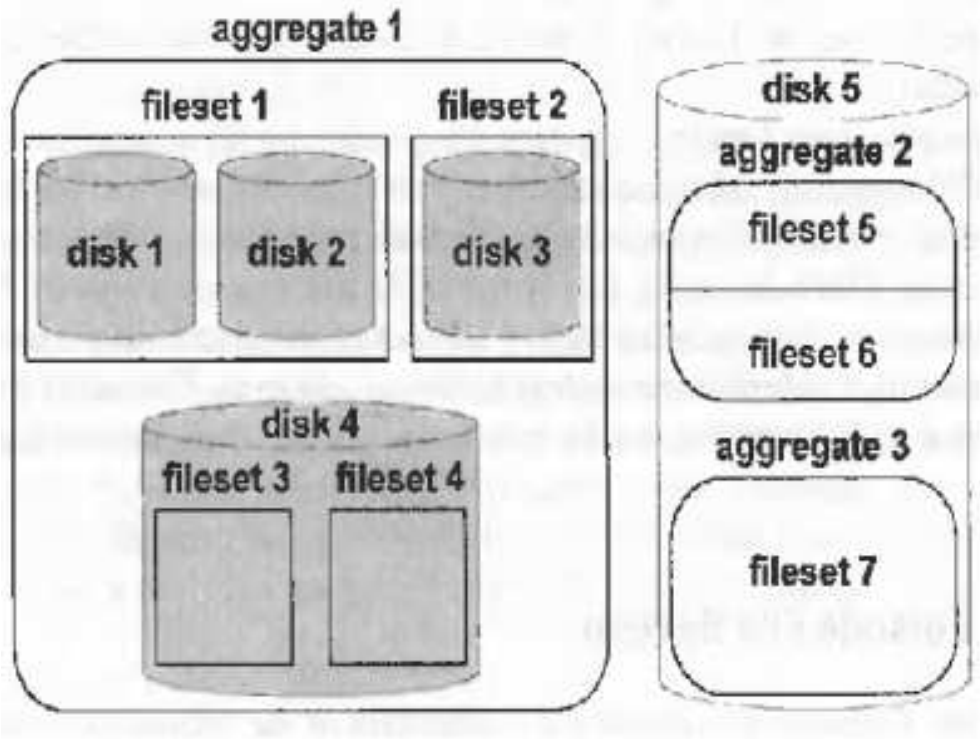
  1. **Kontener na mapę bitową**. Informacja dla każdego fragmentu - czy jest przydzielony i czy jest kronikowany.
  2. **Kontener na kronikę**. Zawiera kronikę metadanych dla tego agregatu. Zwykle znajduje się w agregacie (ale nie musi tak być).
  3. **Kontener na tablicę zestawów plików**. Tablica zawiera superblok i a-węzły dla każdego zestawu plików w agregacie.

  Kontenery zezwalają na trzy tryby przechowywania danych:

  1. *wplecione* (ang. *inline*) - każdy węzeł ma dodatkową przestrzeń, przydatne dla linków symbolicznych, kontroli dostępu, itd.
  2. *fragmentowe* (ang. *fragmented*) - kilka małych kontenerów moze współdzielic jeden blok dysku
  3. *blokowe* (ang. *blocked*) - wspiera cztery poziomy pośredniości



Schemat działania systemu *Episode* **.**



Schemat działania systemu *Episode***.**

Kronika w systemie plików *Episode* Kronika zawiera informacje o przywracaniu i anulowaniu (tj. stare i nowe wartości), która zapewania silna transakcyjną (albo cała operacja albo nic) gwarancję spójności. To rozwiązanie wymaga *dwufazowego protokołu nakładania blokady* (ang. *Twopfase locking*): nakłada się blokadę na wszystkie obiekty, których dotyczy transakcja, dopóki cała transakcja nie zostanie zatwierdzona. Dzięki temu żadna inna transakcja nie jest w stanie odczytać niezatwierdzonych danych.

Klonowanie Można klonować zestawy plików (kopiuje się metadane, a dane plików są dzielone). Operacja ta jest szybka. Sklonowany zestaw plików jest tylko do odczytu (jest jednak aktualizowany) i przydaje się np. w narzędziach administracyjnych.

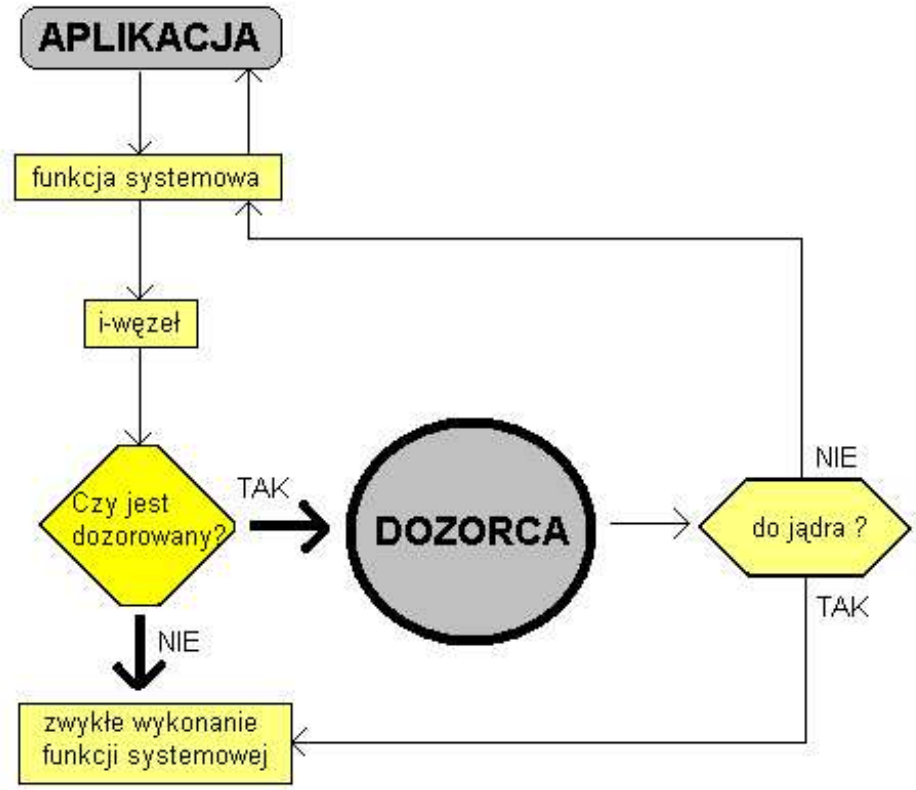
6.2.5 Systemy plików z dozorcą

Zarys problemu. Chcemy w jednolity sposób związać z pewnymi plikami specjalne sposoby obługi dostępu do nich. Na przykład:

1. podejmowanie automatycznych akcji przy odbiorze poczty
2. przechowywanie danych w postaci skompresowanej i autmatyczne rozpakowywanie przy odczytcie
3. kronikowanie / nadzorowanie dostępów do konkretnegio pliku

Wprowadzenie programów dozorujących.Z każdym plikiem związujemy (na żądanie) proces przechwytujący operacje wejscia-wyjscia na tym pliku, zwany *procesem dozorcy* (ang. *watchdog*). Wykonuje się to za pomocą funkcji systemowej: `wdlink (nazwa pliku, nazwa programu dozorcy w katalogu /wdogs)`. Plik staje się *plikiem dozorowanym* (ang. *guarded file*).

Gdy dowolny proces zechce dobrać się do pliku dozorowanego, jądro przekazuje sterowanie do procesu dozorcy (uruchamiając go być może), który podejmuje decyzję. Jeśli dozorca pozwala na otworzenie pliku, pokazuje jakich operacji nalezy użyć (mogą to być nawet różne operacje dla różnych otwartych egzemplarzy tego samego pliku).



Schemat działania systemu z dozorcą.

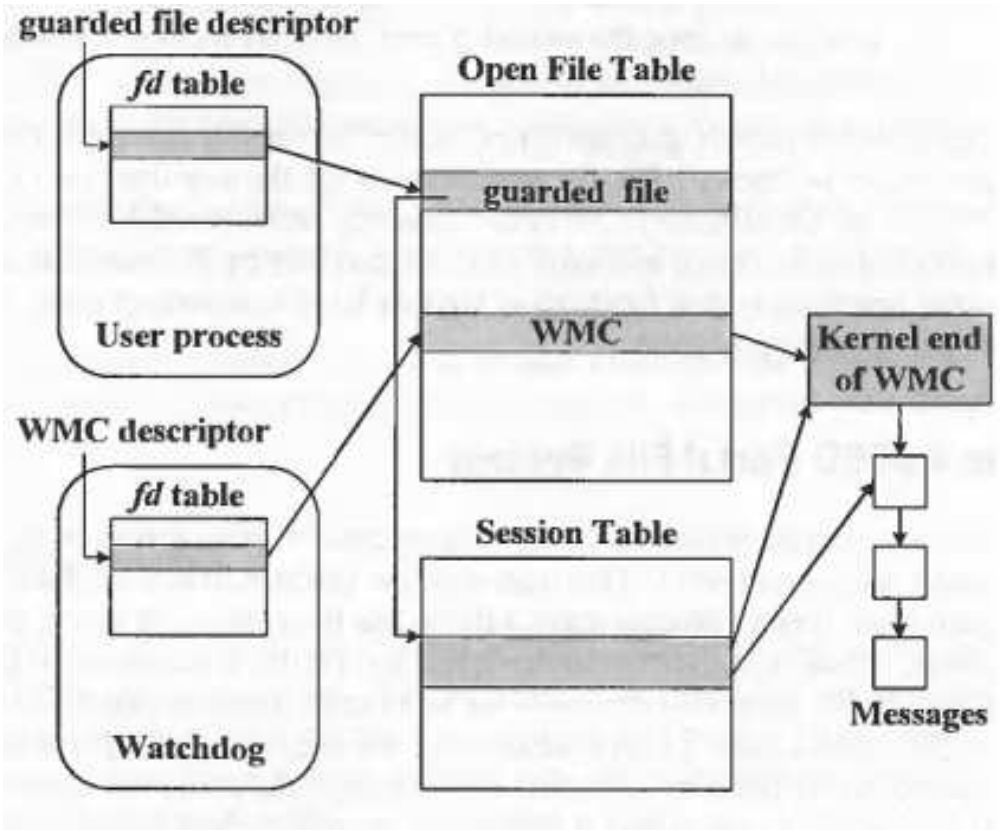
Dozorowanie katalogów. Można dozorować również katalogi. Daje to możliwość tworzenia iluzji plików, których nie ma.

Kanały komunikacyjne. Komunikację między dozorcą obsługuje się za pomocą *kanałów komunikacyjnych* (ang. WMC - *Watchdog Message Channel*), tworzony za pomocą funkcji systemowj *createwmc*,

która zwraca deskryptor kanału. Każdy komunikat ma pole typu, identyfikator sesji i właściwą treść. Każdy otwarty plik rozpoczyna unikatową sesję z dozorcą.

Przebieg komunikacji:

1. Wywołanie funckji wystemowej *createwmc*. Tworzy kanał WMC (plik) i zwraca jego deskryptor.
2. Dozorca odczytuje komunikaty i wysyła odpowiedzi używając deskryptora kanału. Ma dostęp do kolejki komunikatów poprzez ten deskryptor.
3.



Schemat działania systemu z dozorcą.

Zastosowania W pierwotnej wersji wprowadzono kilka ciekawych zastosowań:

wdacl	Związuje z plikiem listę kontroli dostępu (ACL).
wdcompact	Kompresja / dekompresja w locie.
wdbiff	Automatyczne akcje dla skrzynki pocztowej.
wdview	Prezentuje różnym użytkownikom różną postać tego samego katalogu.
wddate	Pozwala na odczytanie bieżącego czasu i daty.

6.2.6 Systemy plików z portalem

Funkcjonalność System plików z portalem (ang. *Portal File System*) oferuje podobną funkcjonalność jak dozorcy. W systemie działa *demon portalu* (ang. *portal daemon*), który obsługuje zlecenia otwarcia plików i przydziela deskryptory. Demon portalu ma unkisowe gniazdgo, na którym nasłuchuje (wywołuje

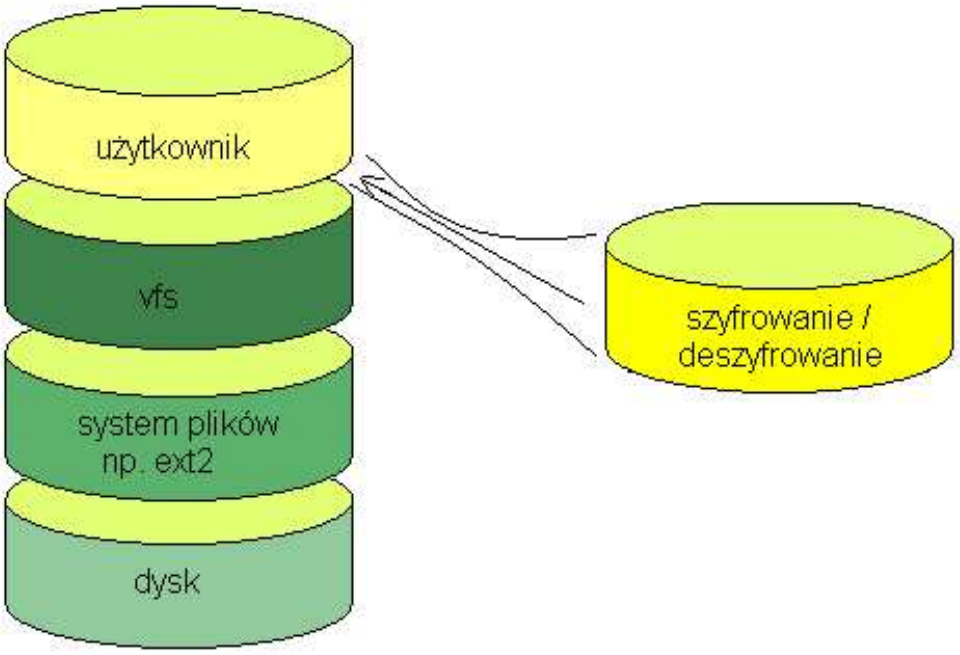
accept-y).

Przebieg komunikacji Proces użytkownika zechce otworzyć nowy plik w systemie plików z portalem. Prosba zostaje przekazana do demona portalu, który określa i zwraca deskryptor nowego pliku.

6.2.7 Stosy systemów plików

Analogia z Wirualnym Systemem Plików

Tworzeniu *Stosowych warst systemu plików* przez UCLA (University of California) i firmę SunSoft przyświecała analogia z przykryciem konkretnych systemów plików przez wirtualny.



Schemat działania systemu z dozorcą.

Możliwości. Mechanizm stosowego nakładania systemów plików pozwala montować wiele systemów plików na drugim. Wirtualny v-węzeł jest - fizycznie - stosem węzłów kolejnych warstw systemów plików.

Przykład. Producent chce dostarczyć moduł szyfrowania-deszyfrowania. Wystarczy umieścić go na wierzchu systemu plików - moduł będzie przechwytywał wsystkie operacje we-wy i przekazywał (po zaszyfrowaniu lub zdeszyfrowaniu) operacje warstwę niżej.

4.4BSD. W systemie 4.4.BSD funkcja systemowa *mount* odkłada nową warstwę systemu plików na stos v-węzła, a *umount* zdejmuje. Każdą operację przekazuje się najpierw do warstwy znajdującej się na wierzchu. Każda warstwa może albo zakończyć wykonywanie się operacji albo przekazać ją do niższej warstwy. Jeżeli warstwa nie rozpoznała operacji przekazuje ją do warstwy niższej za pomocą funckji `bypass`

6.2.8 System plików *nullfs*

System plików *nullfs* pozwala zamontować dowolne poddrzewa hierarchii plików w dowolnym miejscu w systemie plików. Powoduje to skutek taki, jak dostarczenie drugiej nazwy ścieżkowej dla każdego pliku w poddrzewie. *Nullfs* przekazuje operacje do piewrwotnego systemu plików.

6.2.9 System plików *union mount*

Oferuje funkcjonalność podobną do TFS-a. Umożliwia uzyskanie łącznego obrazu (unii) wszystkich systemów plików zamontowanych pod warstwą *union mount*. Wierzchnia warstwa jest najnowasza i tylko do niej można zapisywać. Gdy użytkownik szuka pliku, system przeszuka warstwy poczynając od wierzchniej.

6.3 Zadania

6.3.1 Treść

1. Z jakich powodów system BSD-LSD może być zmuszony do zapisu segmentów częściowych (a nie pełnych) do kroniki?
2. Czy system plików z kroniką jest w 100 % zabezpieczony przed awarią? Jeśli tak, to dlaczego? Jeśli nie to podać przykład, kiedy tak nie jest.
3. Czy w razie utraty tablicy użycia segmentów lub mapy i-węzłów w BSD-LFS nastąpi utrata danych z systemu?
4. Jakie wady ma zastosowanie *dwufazowego protokołu nakładania blokady* w systemie plików *Episode*?

6.3.2 Odpowiedzi

1. Mogą to być: niedobór pamięci podręcznej, wykonanie funkcji systemowej *sync* lub zlecenia NFS.
2. Oczywiście kronika nie daje całkowitego bezpieczeństwa. Jeśli awaria nastąpi przed zapisaniem segmentu w pamięci operacyjnej lub w czasie zapisania (tak, że nie zapiszemy całego) to dane z ostatniego segmentu zostaną utracone.
3. Nie. Te struktury przechowują informacje redundantne, potrzebne do efektywnego działania systemu. Można je odzyskać z kroniki, ale trwa to długo.
4. Takie blokowanie zmniejsza współbieżność systemu, a zatem obniża jego wydajność. W systemie *Episode* niweluje się ten problem wprowadzając *klasy równoważności* transakcji. Równoważne są transakcje działające na tych samych obiektach - albo wszystkie zostaną wykonane albo żadna.

6.4 Źródła

Dodatkowe informacje można znaleźć:

1. Książka Uresha Vahalii *Jądro systemu UNIX*, rozdz. 11 „Złożone systemy plików” - był podtawą tworzenia tej części prezentacji.
2. Prezentacja z 2001 roku dotycząca systemu `/proc` w Linuxie:

   `http://rainbow.mimuw.edu.pl/SO/LinuxASD/index.html/T66/proc.ps`

3. Fragment podręcznika Linuxa:

   `http://rainbow.mimuw.edu.pl/SO/LinuxPodrecznik/PLIKI/`
   `PREZENTACJA/index.html#11_proc`

4. Fragment listy dyskusyjnej dotyczący systemu *tmpfs*:

   `http://lwn.net/2001/1206/a/tmpfs.php3.`

Zawiera informaceje informacje praktyczne. Podobnie strona IBM dotycząca *tmpfs*:

`http://www-106.ibm.com/developerworks/library/l-fs3.html.`

5. Strona poświęcona Unixowi, w tym sporo o systemach plików:

   `http://www.lagmonster.org/docs/unix/sysadm-65.html.`

Rozdział 7

ReiserFS

7.1 Streszczenie

Wiele osób ze zbliżającym się ukazaniem wersji 4 systemu plików Reiser wiąże duże nadzieje. Poza znaczną poprawą wydajności (zwłaszcza dla dużych plików), zostanie w niej zaprezentowany nowy, potężny system wtyczek, nowy system transakcji, alternatywna metoda dostępu do plików poprzez specjalnie do tego celu przygotowaną funkcję systemową i wiele, wiele innych.

Niniejsza część stawia sobie za cel prezentację podstawowych cech i architektury tego ciekawego systemu.

7.2 Wstęp

7.2.1 Powstanie projektu

ReiserFS (*Reiser File System*, w skrócie Reiser) powstał w roku 1996. W odróżnieniu od większości tego typu projektów, które są rozwijane na innych systemach operacyjnych i dopiero z czasem adaptuje się je na potrzeby „pingwinka", Reiser został napisany zupełnie od zera specjalnie dla Linuxa. Jego twórcą związanym z projektem przez cały czas jest Hans Reiser (jak łatwo się domyślić od jego nazwiska pochodzi nazwa systemu).

7.2.2 Licencja i finansowanie

ReiserFS jest rozpowszechniany jako opensource na licencji GPL.

Projekt jest finansowany przez kilka firm tworzących oprogramowanie dla Linuxa. Najważniejszym z nich jest SuSe, w której to firmy dystrybucji Reiser jest domyślnym systemem plików (podobnie jak w Lindows i Gentoo). Poza tym sponsorowany jest także przez DARPA (*The Defense Advanced Research Projects Agency*). Wymusza to na twórcach Reisera (zwłaszcza współfinansowanie przez wojsko) zachowanie najwyższych standardów niezawodności, rozszerzalności oraz przejrzystości kodu.

Ciekawą informacją jest, iż dysponując odpowiednią sumą pieniędzy można współfinansując projekt, zlecić stworzenie np. dodatkowej wtyczki (*plugin*) realizującej użyteczną dla nas funkcjonalność, która jeśli wyrazimy taką chęć zostanie włączona do głównego wydania systemu.

7.2.3 Obecna wersja

Obecnie najnowszą dostępną wersją Reisera jest 3.6. Wersja to została oficjalnie uznana za zamkniętą, w związku z czym nie są już w niej dokonywane żadne zmiany poza usuwaniem zgłaszanych błędów (od połowy 2003 roku ilość tych modyfikacji jest znikoma, co świadczy o stabilności i niezawodności tej wersji). W związku z zamknięciem wersji nie jest przewidywane dostosowanie jej dla potrzeb jądra Linuxa z serii 2.6. Można za to spodziewać się ukazania już wkrótce finalnej 4 wersji ReiserFS. Prace nad nią

są w obecnej chwili właściwie ukończone (są dostępne wyczerpujące testy wydajnościowe porównujące omawiany system z Ext2, Ext3, JFS oraz XFS).

Według zapowiedzi autorów ma to być najwydajniejszy system plików ogólnego zastosowania dla Linuksa.

7.3 Podstawy

7.3.1 Założenia

Głównym założeniem przeświecającym twórcom Reisera było stworzenie możliwie najefektywniejszego systemu plików ogólnego przeznaczenia dla Linuxa odpornego na załamania się systemu operacyjnego, mającego stanowić „poligon" doświadczalny dla różnego rodzaju rozwiązań.

Aby to osiągnąć przejęto do niego wiele rozwiązań z systemów zarządzania bazami danych (*DBMS Data Base Managment System*). Podobne rozwiązania są stosowane np. w XFS czy JFS. O ile jednak wspomniane systemy są nadal przeznaczone głównie do zastosowań serwerowych, to Reiser może być z powodzeniem stosowany zarówno na niewielkich stacjach roboczych zastępując tu Ext, jak również na małych i średniej wielkości serwerach.

7.3.2 Cechy systemu

Z podstawowych cech ReiserFS można wymienić:

- obsługę nawet bardzo dużych plików (do 2^{60}B) oraz partycji
- bardzo efektywny sposób przechowywania wszystkich informacji o plikach i katalogach w pojedynczym drzewie „tańczącym" (w poprzednich wersjach Reisera stosowano do tego celu normalne B+ drzewa)
- zaawansowany system transakcji zapewniający pełną atomowość wykonywanych operacji dyskowych oraz spójność wszystkich danych zapisanych w systemie
- wyraźne rozgraniczenie pomiędzy warstwą semantyczną (odpowiedzialną za organizację danych oraz ich interpretację) oraz fizyczną (odpowiedzialną za ich przechowywanie)
- kompresja wielu małych plików oraz tzw. ogonów (końcówek plików o rozmiarze mniejszym od wielkości bloku) w jednym bloku dyskowym pozwalająca w znacznym stopniu zminimalizować fragmentację wewnętrzną
- efektywna obsługa nawet dużych katalogów (stosowana jest w tym przypadku tablica haszująca, dla której klucze są generowane na podstawie nazwy pliku – bardzo szybkie wyszukiwanie pliku)
- zaawansowany system wtyczek (*plugins*) pozwalający niemal dowolnie modyfikować zachowanie warstwy semantycznej systemu (w przeszłości także w pewnym stopniu fizycznej)

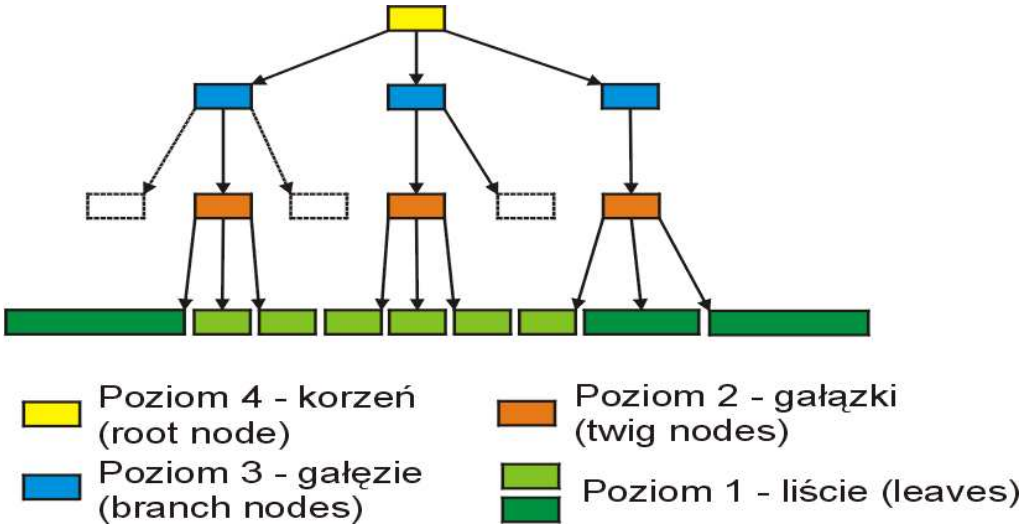
7.4 Budowa

7.4.1 Struktura partycji

W ReiserFS najważniejszą częścią systemu jest pojedyncze drzewo służące do indeksowania bloków dyskowych. Używaną do tego celu strukturą jest tzw. drzewo „tańczące" (*dancing tree*), bardzo podobne do B+ drzewa (na którym oparta była wersja 3 systemu).

Podobnie jak w B+ drzewie dane w drzewie „tańczącym" zapisane są wyłącznie w liściach (wersja 4 w przeciwieństwie do poprzedniczki jest pod tym względem konsekwentna), w węzłach znajdują się wyłącznie klucze. To co odróżnia obie struktury to brak ścisłych wymagań co do liczby pozycji w węzłach wewnętrznych dla tej drugiej (nie jest wymagana zajętość co najmniej 50% pozycji). Jak twierdzi autor

systemu zmiana struktury danych ma zapewnić wzrost wydajności, kosztem być może nieco większej zajętości przestrzeni dyskowej.



Struktura drzewa.

7.4.2 Struktura bloków

Bloki dyskowe w przypadku ReiserFS mogą mieć wielkość od 1 do 64kb. Domyślną wartością jest 4kb (głównie ze względu na mechanizmy buforujące Linuxa), jednak jak wynika z testów zmiana wielkości bloku nie ma większego wpływu zarówno na wydajność działania systemu jak również na efektywność wykorzystania przestrzeni dyskowej.

Postać obiektu (*item*) w drzewie.

Blok korzenia oraz gałęzi (poziomy 3 i 4 drzewa z rysunku)

Blok gałązki (poziom 2 drzewa z rysunku)

Blok liścia (poziom 1 drzewa z rysunku)

7.4.3 Przedziały bloków dyskowych

W wersji 4 wprowadzono wreszcie pojęcie przedziału bloków dyskowych (*extent*) obecne od długiego czasu w innych systemach plików (XFS, JFS, VxFS).

Dzięki temu ma zostać znacznie poprawiona efektywność obsługi dużych plików (z czym wersja 3 miała poważne problemy).

7.4.4 Zarządzanie wolną przestrzenią dyskową

ReiserFS do zarządzania wolną przestrzenią dyskową posługuje się mapami bitowymi zajętości bloków dyskowych. Wyszukiwanie wolnego obszaru rozpoczyna się od lewego sąsiada w drzewie i odbywa się w tym samym kierunku, w którym było wykonywane przy ostatniej operacji.

7.4.5 Kronikowanie

W wersji 4 ReiserFS zrezygnowano ze zwykłego dziennika zmian, który jest najczęściej stosowanym rozwiązaniem w innych systemach (Ext3, NTFS, XFS, JFS, VxFS) i zastosowano rozwiązanie hybrydowe oparte na pewnych elementach systemu WAFL (*Write Anywhere File Layout filesystem*), zapewniające łatwy sposób odzyskania pełnej spójności danych na dysku w przypadku załamania się systemu.

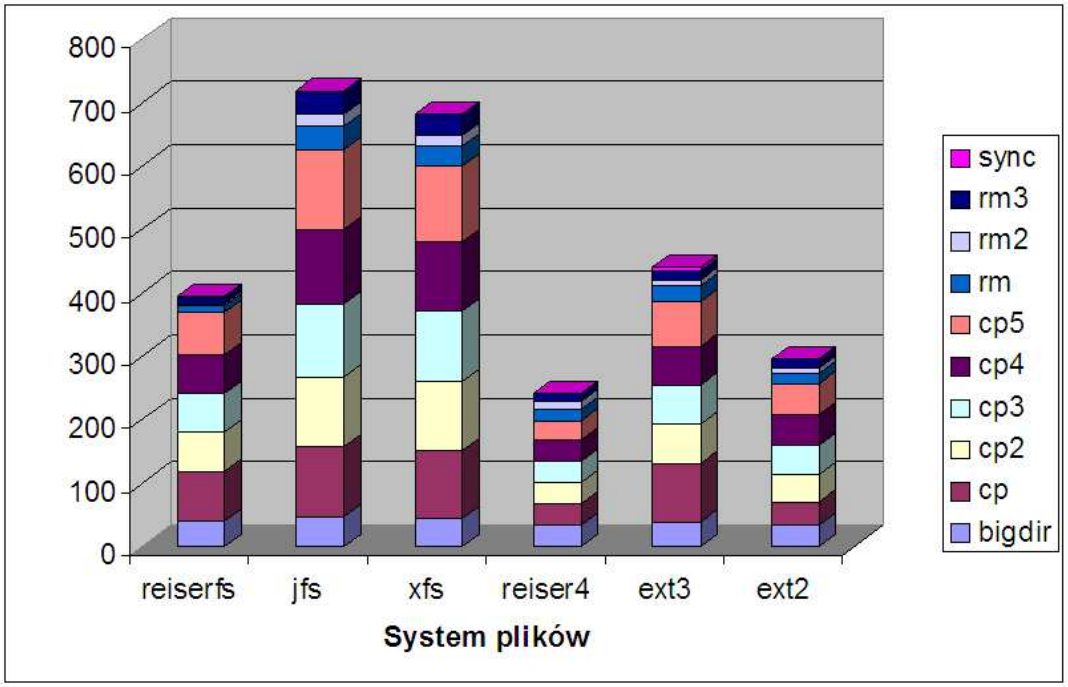
Sposób działania WAFL opiszemy zakładając, że chcemy dokonać w jednej atomowej trasakcji, zapisu do dwóch plików. W tym celu wyszukujemy wpisy opowiadające tym plikom. Zapisujemy zlecone bloki w wolnej obszarze dysku, a następnie kopiujemy bloki, w których znajdują się wskaźniki na nadpisane części, przed zapisem aktualizując odpowiednie wskaźniki. Podążając w ten sposób w górę drzewa dojdziemy do sytuacji gdy wystarczy zmiana pojedynczego wskaźnika w celu zatwierdzenia zmian na dysku (być może będzie trzeba zmienić wskaźnik korzenia drzewa systemu plików w superbloku na której znajduje się partycja).

Podejście, które zastosowano ostatecznie w ReiserFS jest podejściem hybrydowym. Cześć operacji wykonywanych jest jak to opisano powyżej, część jest odnotowywana w dzienniku w celu późniejszego zatwierdzenia (dziennik ma postać listy jednokierunkowej do której wskaźnik znajduje się w superbloku).

7.5 Efektywność

7.5.1 Wyniki przykładowych testów

Poniżej zamieszczam wyniki testów porównujących czas wykonania serii podstawowych operacji dyskowych dla najpopularniejszych systemów plików dostępnych pod Linuxem (im niższy słupek tym lepiej).



Wyniki przykładowych testów.

Jak widać ReiserFS v4 uzyskuje w tych testach znaczącą przewagę nad konkurentami. Wyniki te robią wrażenie jednak nie należy popadać w zbytni zachwyt. ReiserFS v3 w testach próbujących oddać rzeczywiste obciążenie systemu plików wypadał na ogół gorzej od Ext3, czego w powyższym teście nie widać. Należy więć te wyniki traktować raczej jako sygnał postępu, który dokona się wraz z wydaniem nowej wersji Reisera, a nie jako rzeczywiste odzwierciedlenie wydajności poszczególnych systemów.

7.6 Planowane rozwinięcia

Istnieją już pewne wytyczne co do tego co ma się pojawić w kolejnych wersjach Reisera.

7.6.1 Wersja 4.1

Ma się pojawić wbudowany w system plików „repacker" umożliwiający dokonanie czegoś w rodzaju defragmentacji plików. Jest to związane po części z użyciem WAFL, który to może prowadzić do fragmentacji zwłaszcza dużych plików.

7.6.2 Wersje 5 i 6

O tym, która wersja, 5 czy 6 zostanie zaimplementowana jako pierwsza mają zadecydować sponsorzy.

Wersja 5

Ma być rozproszonym systemem plików. Niestety w chwili obecnej brak jest dokładnych szczegółów na temat funkcjonalności tego systemu.

Wersja 6

Ma być systemem opartym na rozszerzonej semantyce, z której szczegółami można się zapoznać na stronie Reisera.

Główną zmianą w tym systemie ma być odejście od typowo hierarchicznej, drzewiastej struktury katalogów na rzecz dowolnych grafów. Takie podejście spowoduje oczywiście spadek wydajności, lecz według twórców umożliwi, wraz z pewnymi dodatkowymi rozszerzeniami, osiągnięcie znacznie bardziej intuicyjnej i precyzyjnej reprezentacji danych.

7.7 Źródła

Ciekawe informacje dotyczące omawianego tematu można znaleźć w:

1. **http://namesys.com/** , zawierająca dokładne omówienie filozofii oraz założeń przyjętych przy jego tworzeniu; brak na niej niestety dokładniejszych szczegółów technicznych

2. katalog **/usr/src/linux/fs/reiserfs/** oraz pliki **/usr/src/linux/include/linux/reiser.h** - kod źródłowy oraz pliki nagłówkowe zawierające ciekawe szczegóły dotyczące systemu, napisane w sposób dość czytelny i przejrzysty