Dwa oblicza FUSE

Jakub Juszczakiewicz

23 października 2026

O mnie...

Jakub Juszczakiewicz

Siewca / inż. Sieffcuś
Rycerz Porytego Kodu

członek alumni ASI
absolwent PWr
pracownik etatowy w Enamor Int.

https://juszczakiewicz.net

Spis rzeczy

1. Czym jest FUSE? Jego zalety i wady
2. Przykłady systemów plików korzystających z FUSE
3. Squashfs w akcji
4. Programowanie z wykorzystaniem libfuse
5. Podsumowanie

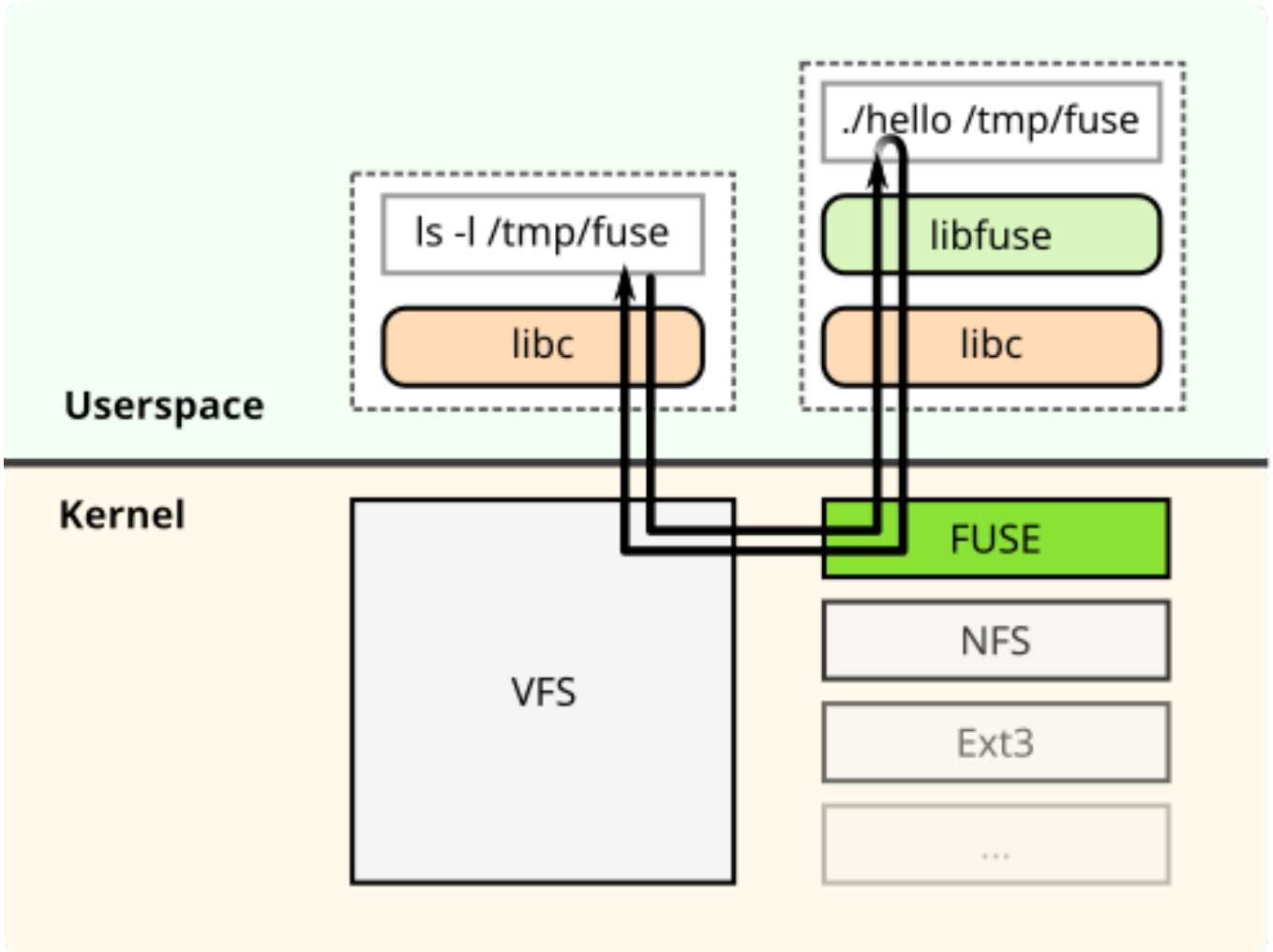
Część 1

FUSE - wprowadzenie

File System in UserSpace

- dostawca systemu plików z przestrzeni użytkownika
- wspierany bezpośrednio przez uniwersalny sterownik w jądrze
- pozwala podmontować system plików zwykłemu użytkownikowi
- kod takiego dostawcy nie wymaga kodu przestrzeni jądra
- przykładowym interfejsem jest biblioteka C (libfuse)

FUSE - zasada działania



Źródło: https://en.wikipedia.org/wiki/Filesystem_in_Userspace

FUSE - przykłady

- sshfs
- rsshfs
- smb-fuse
- ntfs3g
- exfat-fuse
- CurlFtpFS
- davfs2
- ldap-fuse
- squashfs-fuse

FUSE - jak używać?

Montowanie

montowanie systemu plików dostarczanego przez fuse odbywa się przez aplikację, która dostarcza dany system plików

Odmontowanie

```
fusermount -u <katalog>
umount <katalog>
```

Przykłady

sshfs

- sieciowy system plików przez SSH
- pod spodem lata SFTP
- obsługuje kompresję
- obsługuje wiele wątków

przykłady

```
sshfs hosting: www
sshfs -C backup@nas:/var/storage/backup files
sshfs -C storage:/var/data /home/user/data -oro
```

Przykłady

rsshfs

- odwrócone sshfs
- pozwala po sshfs podmontować zdalnie lokalny katalog
- przydatne gdy chcemy udostępnić lokalne pliki na jakiejś maszynie
- nie daemonizuje się - zamknięcie aplikacji wymusza przerwanie udostępniania

przykłady

```
rsshfs projekt builder@worker01:sources/project
rsshfs sources worker02:project -o ro
```

Przykłady

smb-fuse

- sieciowy system plików bazujący na SMB/CIFS
- pozwala podmontować zdalne zasoby samby
- napisany w RUST

przykłady

```
smbfuse mount //SERVER/SHARE  /mnt/nas -u USER -p
smbfuse umount  /mnt/nas
```

Przykłady

ntfs-3g

- pierwsza pełna implementacja obsługi NTFS na *nixy
- wraz ze sterownikiem FUSE są dostarczone narzędzia np. do naprawiania systemu plików
- montowanie wymaga uprawnień super użytkownika ze względu na konieczność dostępu blokowego do woluminu
- powstał na długo przed sterownikiem w jądrze ze względu na ograniczenia prawne

przykłady

```
ntfs-3g /dev/sda1 /mnt/windows
umount /dev/sda1
```

Przykłady

exfat-fuse

- implementacja obsługi exFAT (FAT64)
- jest to sam sterownik - narzędzia do exFAT są osobnymi pakietami
- montowanie wymaga uprawnień super użytkownika ze względu na konieczność dostępu blokowego do woluminu

przykłady

```
mount.exfat-fuse /dev/sdd1 /mnt/pendrak
mount.exfat-fuse -oro /dev/sdd1 /mnt/pendrak
mount -t exfat-fuse /dev/sde1 /mnt/pendrak2
umount /dev/sdd1
```

Przykłady

CurlFtpFS

- sieciowy system plików bazujący na FTP
- do obsługi połączeń z serwerem wykorzystuje bibliotekę libcurl
- obsługuje szyfrowanie i pośredników sieciowych
- pozwala na dostęp z logowaniem
- ostatnia wersja jest z kwietnia 2007r

przykłady

```
curlftpfs ftp.example.com /mnt/ftp/
curlftpfs ftp /mnt/ftp/ -o user=username:password
```

Przykłady

davfs2

- sieciowy system plików bazujący na WebDAV
- obsługuje proxy, szyfrowanie i uwierzytelnienie
- posiada własną implementację klienta WebDAV
- bazuje na lokalnej pamięci podręcznej
- nadal rozwijany (ostatni commit w tym roku)

przykłady

```
mount -t davfs http://dav.example.com/ /media/dav
mount.davfs http://dav.server/ /media/dav -oro
```

Przykłady

ldap-fuse

- wirtualny system plików pozwalający przeglądać katalogi LDAP
- wspiera tylko odczyt
- pozwala przeglądać wszystkie pola
- projekt obecnie nierozwijany

przykłady

```
ldapfuse ldap://ldap.server/dc=company ~/ldap
ldapfuse -D cn=admin,dc=company -w P@$$WORD \
  ldap://ldap.server/dc=company ~/ldap
```

Przykłady

squashfs-fuse

- implementacja sqaushfs w fuse
- są 2 wersje: „zwykła“ i „low-level”
- do podmontowania pliku nie potrzeba uprawnień root
- wersja LL jest dodatkowo wielowątkowa
- obsługuje tylko odczyt

przykłady

```
squashfuse backup.squashfs ~/mnt/squashfs
squashfuse_ll archive.squashfs ~/mnt/squashfs
```

Wydajność - wstęp

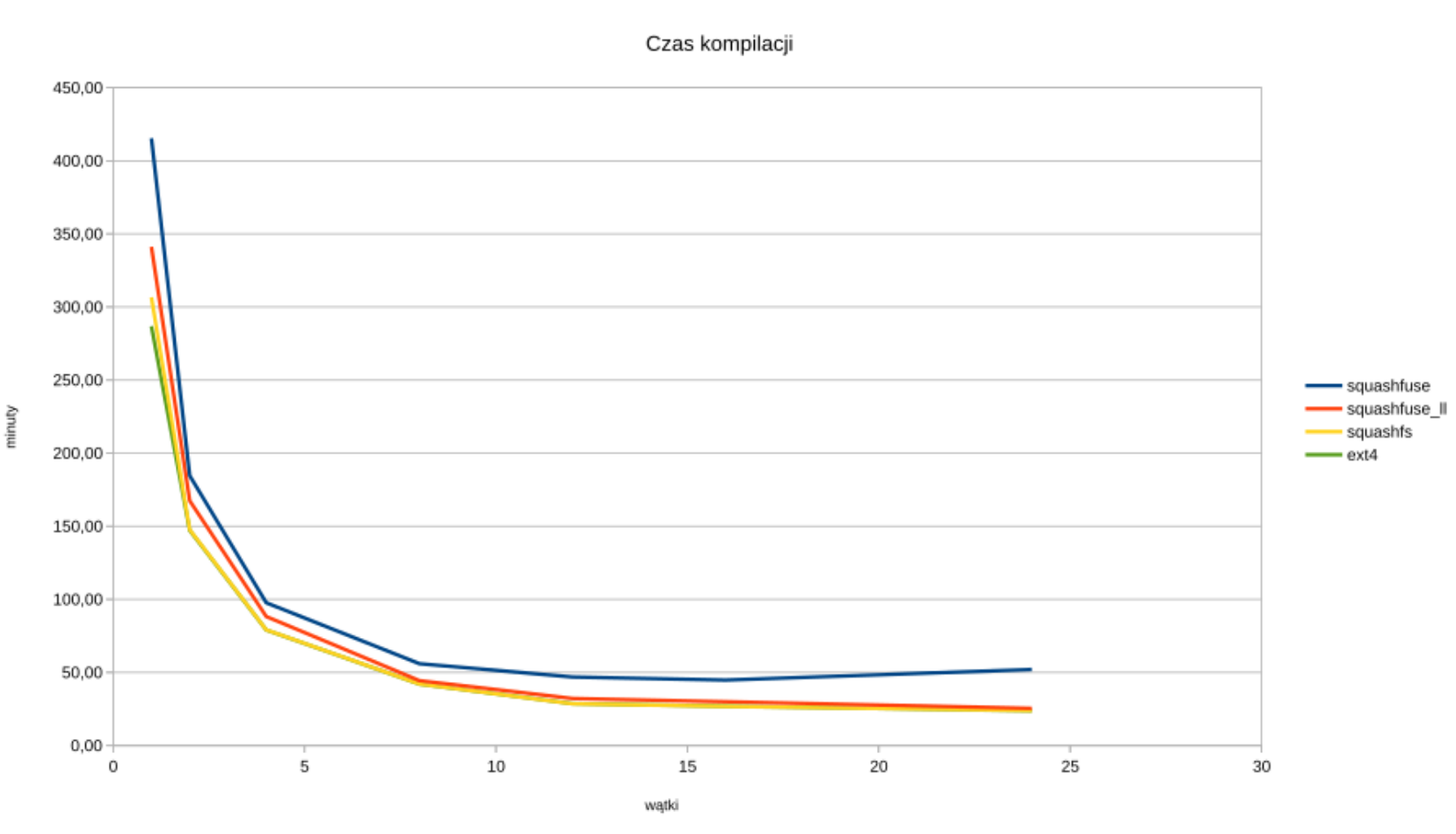
Pomiary

- kompilacja linuksa 6.18.44 z konfigiem ze Slackware64-current
- sprawdzanie dla 1, 2, 4, 8, 12, 16 i 24 wątków na raz
- sprawdzanie dla squashfuse, squashfuse_ll, squashfs i ext4
- kompilacja na maszynie z 24 wątkowym CPU

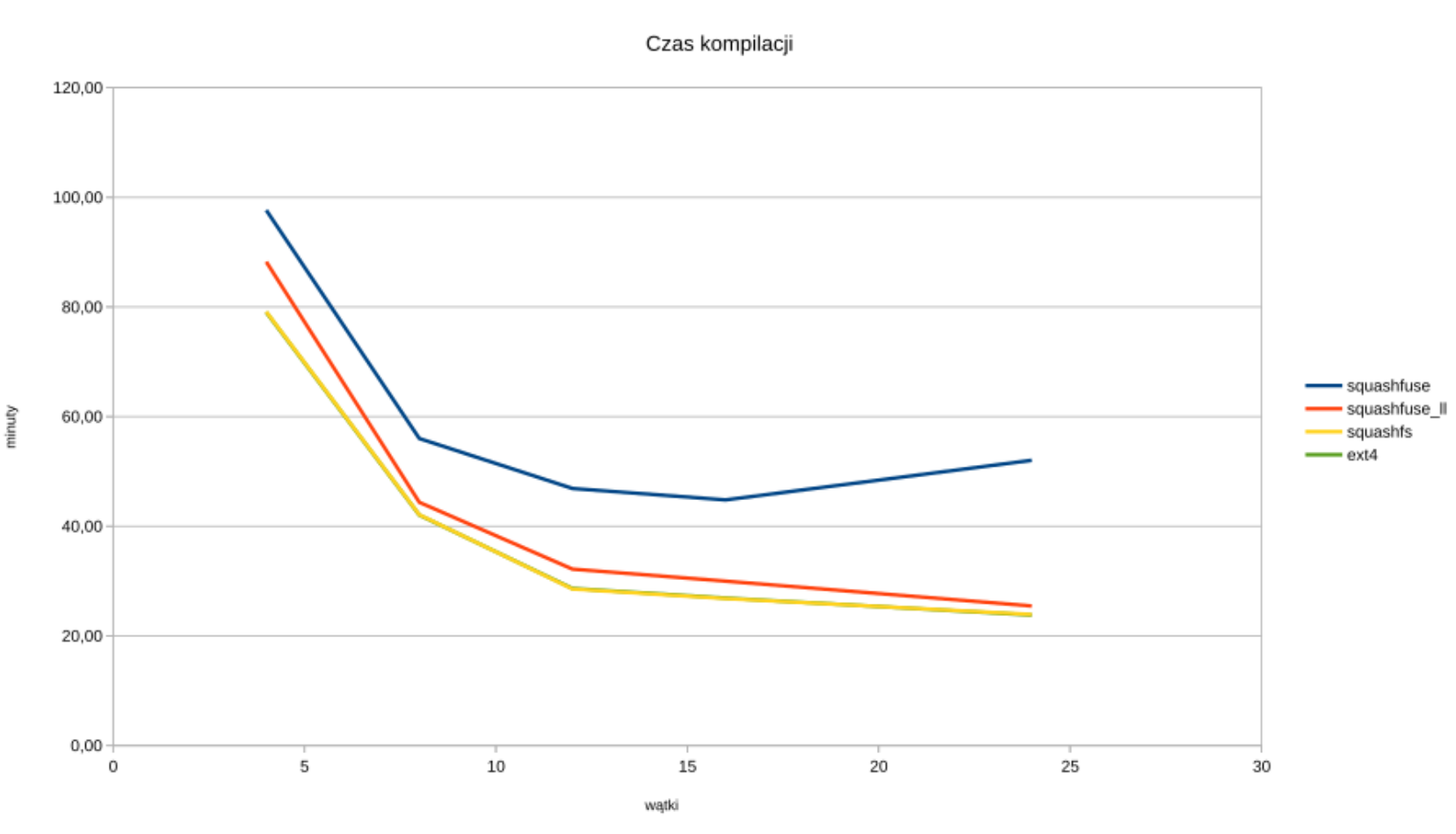
Przygotowanie

```
cat linux-6.18.44.tar.xz |pixz -d |mksquashfs -
linux-6.18.44.squashfs -tar -exports -b 1M -comp
zstd -Xcompression-level 22
```

Wydajność - wyniki



Wydajność - wyniki (2)



SquashFS

Cechy

- skompresowany, blokowy system plików tylko do odczytu
- dedykowany dla systemów wbudowanych
- obsługuje: gzip, lzo, lz4, xz i zstd
- uwzględnia różne stopnie kompresji i rozmiary bloków
- `mksquashfs` potrafi czytać katalogi/pliki, TARy
- obsługuje również xattr i duplikację plików
- kompresja jest wielowątkowa
- jest narzędzie `unsquashfs`

- potrafi pełnić rolę archiwum
- nadaje się do kopii zapasowych

SquashFS c.d.

SquashFS jako archiwum

- dostęp blokowy z kompresją
- uszkodzenia w pliku nie propagują się
- zawartość można przeglądać bez wypakowywania całości
- sprawdzenie poprawności zawartości wymaga sprawdzenia 1 pliku
- nie nadaje się na taśmy
- dobrze będzie się sprawdzał na urządzeniach z dostępem blokowym, takich jak dyski, czy płyty
- można go zaszyfrować LUKSem i nagrać na płytę

SquashFS na backup

Jak zrobić kopię zapasową?

```
nice mksquashfs katalog backup.squashfs -b 1M \
  -comp zstd -Xcompression-level 22
truncate -s +8M backup.squashfs
nice cryptsetup -q reencrypt --encrypt \
  --type luks2     --resilience none \
  --disable-locks --reduce-device-size 8M \
  backup.squashfs <<< 'P@$$WORD'
```

- powyższa procedura nie wymaga uprawnień super użytkownika
- tak utworzony plik można bezpiecznie zapisać
- odczytanie pliku już wymaga dodatkowych uprawnień

SquashFS - statystyki

linux-6.18.44

algorytm	tar	sqaushfs	$\frac{tar}{squashfs}$
gzip -9	239,19MiB	236,76MiB	101,03%
xz	147,42MiB	171,68MiB	85,87%
lz4 -12	284,98MiB	284,50MiB	100,17%
zstd -22	143,43MiB	177,72MiB	80,71%

Backup /home (ok. 115,31GiB)

algorytm	tar	sqaushfs	$\frac{tar}{squashfs}$
gzip -9	99,12GiB	91,54GiB	108,28%
xz	94,97GiB	90,00GiB	105,53%
lz4 -12	100,29GiB	92,46GiB	108,48%
zstd -22	94,57GiB	90,25GiB	104,78%

Część 2

Własny system plików

Jak zacząć?

- na początek trzeba znać API POSIX dotyczące systemu plików
- do wykorzystania FUSE jest potrzebna biblioteka libfuse
- warto zacząć od zapoznania się z przykładami dostępnymi w repo projektu (https://github.com/libfuse/libfuse/tree/master/example)
- informację o zależnościach i flagach kompilacji wyciągniemy z pkg-config (libfuse3)
- warto zacząć od przykładów helloworld i passthrough - na początek w „zwykłej wersji“ (nie low-level)
- na potrzeby prezentacji napisałem własny, wirtualny system plików kit-tag-fuse dostępny na mojej stronie

Własny system plików c.d.

Założenia

- libfuse ma własną pętlę główną programu
- całość komunikacji odbywa się na bazie zestawu callbacków
- aplikacja samodzielnie się daemonizuje
- odmontowanie udostępnionego systemu plików zamyka aplikację
- nie ma potrzeby implementować wszystkich callbacków

Własny system plików c.d.

Przykład - kit-tag-tool

- 2 narzędzia:
  1. tworzące bazę danych sqlite3 z zawartością przeczytanych TAGów z plików multimedialnych
  2. przeglądarka bazy w postaci aplikacji FUSE
- użycie polega na przeczytaniu przez pierwsze narzędzie wszystkich obsługiwanych plików z podanego podkatalogu, a wyciągnięte z nich znaczniki są zapisywane do pliku z bazą danych
- przeglądanie bazy danych odbywa się przez „jej podmontowanie" przy użyciu FUSE jako zestaw plików i katalogów
- struktura katalogów to `/artysta/album/piosenka/info`.

Własny system plików - main

```
#define FUSE_USE_VERSION FUSE_MAKE_VERSION(3, 18)

// ...

int main(int argc, char * argv[])
{
  struct stat stbuf;
  if (stat(argv[1], &stbuf) != 0)
    exit(1);

  memcpy(&g_atime, &stbuf.st_atim, sizeof(g_atime));
  memcpy(&g_mtime, &stbuf.st_mtim, sizeof(g_mtime));
  memcpy(&g_ctime, &stbuf.st_ctim, sizeof(g_ctime));
```

Własny system plików - main (2)

```
  sqlite3_open(argv[1], &g_db);
  if (!g_db)
    exit(1);

  char *args[2] = {argv[0], argv[2]};
  return fuse_main(sizeof(args) / sizeof(args[0]),
      args, &db_oper, NULL);
}
```

Własny system plików - callbacki

```
static const struct fuse_operations db_oper =
{
   .getattr  = db_getattr,
   .access   = db_access,
   .readlink = db_readlink,
   .readdir  = db_readdir,
   .open     = db_open,
   .read     = db_read,
   .release  = db_release,

   .init     = db_init,
   .destroy  = db_destroy,
};
```

Własny system plików - opis piosenki

```
struct song
{
  char * title;
  char * album;
  char * artist;
  char * comment;
  char * genre;
  int year;
  int track;
  char * path;
};
```

Własny system plików - funkcje pomocnicze c.d.

```
long get_album(const char * album);

long get_artist(const char * artist);

int is_artist_album(const char * album,
    const char * artist);

long is_song(const char * album,
    const char * artist, const char * title);
```

Własny system plików - funkcje pomocnicze

```
long get_song(const char * album,
    const char * artist, const char * title,
    struct song * song);

void free_song(struct song * song);

int get_song_from_path(const char * path,
    struct song * song);
```

Własny system plików - mięsko

```
static int db_stat(const char * path,
    struct stat * stbuf)
{
  memset(stbuf, 0, sizeof(struct stat));
  stbuf->st_uid = getuid();
  stbuf->st_gid = getgid();

  memcpy(&stbuf->st_atim, &g_atime, sizeof(g_atime));
  memcpy(&stbuf->st_mtim, &g_mtime, sizeof(g_mtime));
  memcpy(&stbuf->st_ctim, &g_ctime, sizeof(g_ctime));

  if (strcmp(path, "/") == 0) {
    stbuf->st_mode = S_IFDIR | 0555;
    stbuf->st_nlink = 1;
    return 0;
  }
```

Własny system plików - mięsko c.d.

```
    if (strcmp(ptr3, "path") == 0) {
      stbuf->st_mode = S_IFLNK | 0444;
      stbuf->st_nlink = 1;
      stbuf->st_size = strlen(info.path);
      free_song(&info);
      free(paths);
      return 0;
    }

    free_song(&info);
    free(paths);
    return -ENOENT;
}
```

Własny system plików - getattr

```
static int db_getattr(const char * path,
    struct stat * stbuf,
    struct fuse_file_info *fi)
{
  return db_stat(path, stbuf);
}
```

Własny system plików - access

```
static int db_access(const char * path, int mask)
{
  struct stat stbuf;
  int r = db_stat(path, &stbuf);
  if (r < 0)
    return r;

  if ((mask == F_OK) || (mask == R_OK))
    return 0;

  if ((mask == X_OK) && ((stbuf.st_mode & S_IFMT)
        == S_IFDIR))
    return 0;

  if (mask == W_OK)
    return -EACCES;
```

Własny system plików - readdir

```
static int db_readdir(const char *path, void *buf,
    fuse_fill_dir_t filler,
    off_t offset, struct fuse_file_info *fi,
    enum fuse_readdir_flags flags)
{
  if (strcmp(path, "/") == 0) {
    filler(buf, ".", NULL, 0,
        FUSE_FILL_DIR_DEFAULTS);
    filler(buf, "..", NULL, 0,
        FUSE_FILL_DIR_DEFAULTS);

    const char * sql = "SELECT name FROM artist;";
    int r;
    sqlite3_stmt * stmt = NULL;
```

Własny system plików - readdir c.d.

```
  while (sqlite3_step(stmt) == SQLITE_ROW) {
    filler(buf, sqlite3_column_text(stmt, 0),
      NULL, 0, FUSE_FILL_DIR_DEFAULTS);
  }

  sqlite3_finalize(stmt);
  return 0;
}
```

Własny system plików - readdir c.d.

```
filler(buf, ".", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "..", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "title", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "artist", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "album", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "comment", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "genre", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "year", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "track", NULL, 0, FUSE_FILL_DIR_DEFAULTS);
filler(buf, "path", NULL, 0, FUSE_FILL_DIR_DEFAULTS);

free(paths);
return 0;
```

Własny system plików - open

```
static int db_open(const char *path,
    struct fuse_file_info *fi)
{
  struct stat stbuf;
  int r = db_stat(path, &stbuf);
  if (r < 0)
    return r;

  if ((fi->flags & O_ACCMODE) != O_RDONLY)
    return -EACCES;

  if ((stbuf.st_mode & S_IFMT) != S_IFREG)
    return -EINVAL;

  return 0;
}
```

Własny system plików - destroy

```
void db_destroy(void * private_data)
{
  sqlite3_close(g_db);
}
```

Pomysły

Czy tylko system plików?

- FUSE sprawdza się do fizycznych systemów plików
- nic nie stoi na przeszkodzie napisania wirtualnego systemu plików
- można sterować aplikacją przez FUSE i np. unix socket
- można napisać edytor partycji bazując na FUSE

Podsumowanie

Zastosowania

- FUSE pozwala przygotować sterownik do systemu plików bez zagnieżdżania się we wnętrzności jądra
- może służyć do pisania prototypów zanim kod zacznie się integrować z kernelem
- Wydajność jest niższa niż przy natywnych sterownikach, ale dobrze napisany program wykorzystujący interfejs low-level nie odstaje w znaczący sposób
- System plików przygotowany przy użyciu FUSE nie musi odzwierciedlać fizycznych plików - może to być interfejs do aplikacji, czy bazy danych

Koniec!

Pytania?

Dziękuję za uwagę!