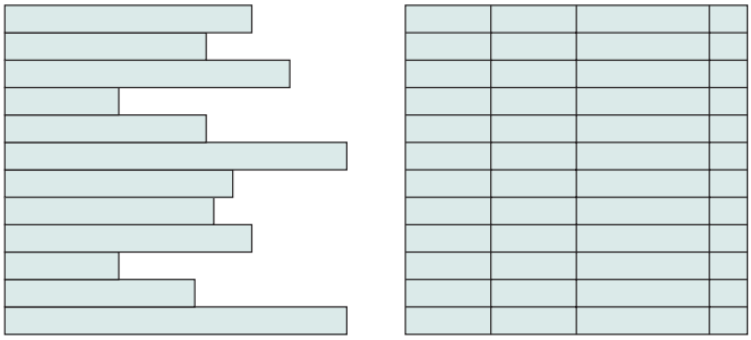
L11.Z1. Na podstawie rozdziału §12.2 podręcznika Stallings wyjaśnij różnice między organizacją **sterty** i **pliku sekwencyjnego**. Rozważmy poniższe przypadki użycia:

- Zaproponuj binarny format przechowywania dziennika operacji składających się z listy par klucz-wartość złożonych ze znaków alfanumerycznych. Podaj semantykę kodów ASCII o numerach 28–31.
- Dodajemy rekordy do pliku przechowującego użytkowników systemu. Jak zachować porządek używając pliku dziennika lub listy dwukierunkowej? Jak przyspieszyć wyszukiwanie z użyciem indeksu?

Wyjaśnij różnice między organizacją sterty i pliku sekwencyjnego.

sterta –

plik sekwencyjny –



1
Sterta - sposób organizacji danych gdy są one składowane po kolei w polach o różnej wielkości odpowiadających ich rozmiarowi. Każde pole zawier opis (rozmiar, typ danych itd.). Wyszukiwanie pojedynczego rekordu wymaga sprawdzenia po kolei. Dobre do składowania dużych ilości danych o różniącym się rozmiarze, które będziemy wczytywać sekwencyjnie

Plik sekwencyjny - każdy rekord w pliku ma tę samą długość i jest podzielony na takie same pola nie trzeba więc trzymać informacji o jego strukturze. Pierwsze pole to identyfikator - klucz. Aby znaleźć rekord, trzeba (?) po kolei i szukać pierwszego pasującego klucza. Zmiana wielkości rekordu jest również bardzo nieefektywna. Rekordy trzymane są w kolejności (np. alfabetyczna kolejnosć kluczy), dodawanie nowych rekordów też może być problematyczne i zajmować dużo czasu.

plik dziennika - chronologiczny zapis zawierający informacje o zdarzeniach i działaniach użytkowników

indeks - osobny plik zawierający klucz - odnośnik do grupy rekordów z kluczami posortowanymi w sposób ułatwiający szukanie

ASCII:
28 - FS File Separator - EOF or between concatenation of what otherwise could be separate files
29 - GS Group Separator - between sections/groups of data
30 - RS Record Separator - end of record row
31 - US Unit Separator - between fields of a single record

binarny format: ciąg znaków przedzielany tymi ascii powyżej

lista dwukierunkowa - dodanie na koniec i przepisanie wskaźników (patrz lista 7, zad 3)

plik dziennika - trzymanie userów w kolejności logowania i nadawanie im w odpowiedni sposób kluczy

indeks pozwala (w przypadku pliku dziennika) trzymać klucze odpowiadające grupom rekordów, czyli zamiast jednego przejścia po ogromnym zbiorze (np. 10^6 rekordów) mamy dwa po małych (np. po 1000) po indeksie i bloku rekordów. możliwe jest wielopoziomowe indeksowanie

L11.Z2. Rozważamy **hierarchiczną strukturę katalogów**. Czym różnią się **ścieżka absolutna**, **relatywna** i **znormalizowana**? Względem którego katalogu obliczana jest ścieżka relatywna? Jak zmienić ten katalog? Wyjaśnij czym są **punkty montażowe**, a następnie na podstawie `mount`(8) wyjaśnij znaczenie i zastosowanie następujących atrybutów punktów montażowych: «`noatime`», «`noexec`» i «`sync`».

systemy katalogów

- jednopoziomowe
  - jest jeden katalog główny, który zawiera wszystkie pliki
  - zalety: prostota, możliwość szybkiej lokalizacji plików – jest tylko jedno miejsce, gdzie można ich szukać
  - wady: w nowoczesnych komputerach zawierających tysiące plików znalezienie czegokolwiek byłoby niemożliwe, gdyby wszystkie pliki były zapisane w pojedynczym katalogu
- hierarchiczne
  - ustalamy hierarchię – drzewo katalogów, grupujemy pliki. W węzłach są katalogi, w liściach pliki.

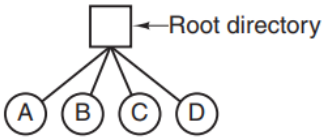


Figure 4-6. A single-level directory system containing four files.

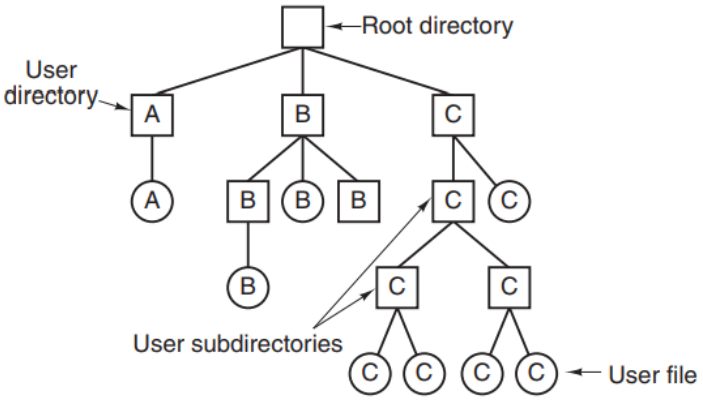


Figure 4-7. A hierarchical directory system.

Czym różnią się ścieżka absolutna, relatywna i znormalizowana? Względem którego katalogu obliczana jest ścieżka relatywna? Jak zmienić ten katalog?

Kiedy system plików jest zorganizowany w postaci drzewa katalogów, potrzebny okazuje się sposób specyfikacji nazw plików. Stosowane metody: podanie ścieżki bezwzględnej lub podanie ścieżki względnej.

rozwiązywanie nazw – proces odwzorowania ścieżki na zasób. Ścieżka składa się z komponentów i znaków separatora „/”. Rozwiązywanie nazw należy do zadań wirtualnego systemu plików (ang. *Virtual File System*). Ścieżkę odwzorowujemy na v-węzeł, czyli reprezentację i-węzła w pamięci niezależną od systemu plików.

ścieżka bezwzględna (absolutna) – ścieżka z katalogu głównego do pliku. Jest unikatowa. W przypadku gdy pierwszym znakiem nazwy ścieżki jest separator (/ lub \), to ścieżka jest bezwzględna.

ścieżka względna (relatywna) – wszystkie nazwy ścieżek, które nie rozpoczynają się od katalogu głównego, są interpretowane w odniesieniu do katalogu roboczego (bieżącego)

ścieżka znormalizowana – nie zawiera . (bieżący katalog), .. (katalog nadrzędny) i dowiązań symbolicznych

katalog bieżący - katalog wskazany w ostatnim poprawnie wykonanym poleceniu zmiany katalogu (cd, chdir); katalog, w którym obecnie pracujemy. Zmiana katalogu roboczego: `chdir` *`ścieżka`* `lub fchdir` *`fd`*.

Wyjaśnij czym są punkty montażowe.

montowanie – systemu plików urządzeń zewnętrznych nie można używać, ponieważ nie ma sposobu zdefiniowania ścieżki, która by do nich prowadziła. Montowanie systemu plików pozwala na dołączenie systemu plików na urządzeniu zewnętrznym do głównego systemu plików.

punkt montażowy – katalog w systemach uniksowych, do którego jest montowane drzewo systemu plików rozpoznanego właśnie urządzenia (płyty, pendrajwa).

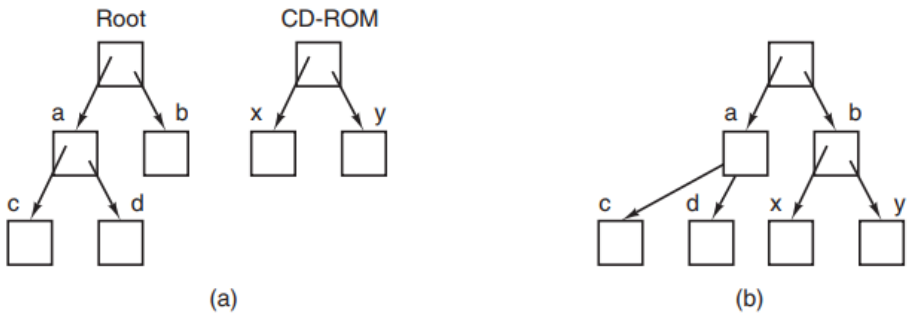


Figure 1-15. (a) Before mounting, the files on the CD-ROM are not accessible. (b) After mounting, they are part of the file hierarchy.

Na podstawie `mount`(8) wyjaśnij znaczenie i zastosowanie następujących atrybutów punktów montażowych: «`noatime`», «`noexec`» i «`sync`».

`mount` - mount a filesystem
All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at /. These files can be spread out over several devices. The **mount** command serves to attach the filesystem found on some device to the big file tree. Conversely, the umount(8) command will detach it again.

- `noatime` – nie aktualizuj czasu dostępu do i-węzłów w tym drzewie, działa też dla katalogów (ważne przy optymalizacjach dostępu; pomocne, gdy chcemy zmodyfikować coś na dysku, ale nie chcemy, żeby ktoś kto będzie to potem czytał zorientował się, że coś zrobiliśmy, poprzez odczyt access time)
- `noexec` – brak możliwości bezpośredniego exec-owania binarek na montowanym systemie plików (np. gdy złośliwy kolega przynosi nam pendrive z plikami wykonywalnymi, nie wiemy, co one mogą robić; chronimy się przed odpalaniem tego)
- `sync` – wszystkie I/O do systemu plików powinny być wykonane synchronicznie (program musi czekać na zakończenie poprzedniego dostępu). Może to spowodować skrócenie życia nośnika, jeśli ma on ograniczoną liczbę cykli zapisu.

L11.Z3. Wymień wszystkie **typy plików** w systemie Linux. Na podstawie strony *File system calls* podaj listę istotnie różnych wywołań systemowych realizujących operacje na **plikach zwykłych** i **plikach urządzeń**. Które z operacji nie mają sensu dla plików o **dostępie sekwencyjnym** i dlaczego? Wyjaśnij znaczenie wszystkich **atrybutów pliku,** które można odczytać wywołaniem `stat`(2).

Wymień wszystkie typy plików w systemie Linux.

typ pliku - to rodzaj zasobu jądra, które zostało udostępnione użytkownikowi przez interfejs interakcji z plikiem.

- plik zwykły → ciąg bajtów o swobodnym dostępie
- katalog → ciąg rekordów opisujących zawartość katalogu
- potok → ciąg bajtów o dostępie sekwencyjnym zawierający
- dane wysłane z innego procesu
- urządzenie znakowe → ciąg bajtów o dostępie sekwencyjnym
- odpowiadający urządzeniu (np. port szeregowy, drukarka, …)
- urządzenie blokowe → ciąg bajtów o swobodnym dostępie,
- najefektywniejszy dostęp blokowy (dysk twardy, pendrive)
- gniazdo domeny uniksowej → jak potok, ale dwukierunkowe
- dowiązanie symboliczne → "wskaźnik" na inny plik

Na podstawie strony File system calls podaj listę istotnie różnych wywołań systemowych realizujących operacje na plikach zwykłych i plikach urządzeń.
http://linasm.sourceforge.net/docs/syscalls/filesystem.php

plik urządzenia - device file or special file is an interface to a device driver that appears in a file system as if it were an ordinary file

Syscall	Description
CLOSE	Close a file descriptor
OPEN	Open and possibly create a file
NAME_TO_HANDLE_AT	Obtain handle for a pathname
MKNOD	Create a special or ordinary file
RENAME	Rename a file
TRUNCATE	Truncate a file to a specified length
FALLOCATE	Manipulate file space

The **name_to_handle_at**() and **open_by_handle_at**() system calls split the functionality of openat(2) into two parts: **name_to_handle_at**() returns an opaque handle that corresponds to a specified file; **open_by_handle_at**() opens the file corresponding to a handle returned by a previous call to **name_to_handle_at**() and returns an open file descriptor.

The system call **mknod**() creates a filesystem node (file, device special file, or named pipe) named *pathname*, with attributes specified by *mode* and *dev*.

Które z operacji nie mają sensu dla plików o dostępie sekwencyjnym i dlaczego?

dostęp sekwencyjny - operacja dostępu ma charakterystykę temporalną, po wykonaniu operacji nie możemy cofnąć się w czasie np. pakiety sieciowe, drukarka, potok

fallocate() allows the caller to directly manipulate the allocated
disk space for the file referred to by *fd* for the byte range starting
at *offset* and continuing for *len* bytes.

`pread` i `pwrite` – read / write from a file descriptor at a given offset
`lseek` - reposition read/write file offset
`truncate` – truncate a file to a specified length
`fallocate` – manipulate file space

Wyjaśnij znaczenie wszystkich atrybutów pliku, które można odczytać wywołaniem `stat`(2).

stat() - służy do pobierania statusu pliku przechowywanego w jego i-węźle. Przekazuje takie informacje jak typ pliku, właściciel pliku, prawa dostępu, rozmiar pliku, liczba dowiązań, numer i-węzła i czas dostępu do pliku.

```
struct stat {
    dev_t     st_dev;       numer urządzenia zawierającego dany plik
    ino_t     st_ino;       numer i-węzła
    mode_t    st_mode;      tryb pliku
    nlink_t   st_nlink;     licznik dowiązań twardych
    uid_t     st_uid;       identyfikator wlasciciela pliku
    gid_t     st_gid;       identyfikator grupy
    dev_t     st_rdev;      numer (ID) urządzenia
    off_t     st_size;      rozmiar w bajtach
    blksize_t st_blksize;   gives the "preferred" blocksize for efficient file system I/O.
                            (Writing to a file in smaller chunks may cause an inefficient read-modify-
                            rewrite.)
    blkcnt_t  st_blocks;    liczba zaalokowanych bloków w jednostkach po 512B
    time_t    st_atime;     czas ostatniego dostępu
    time_t    st_mtime;     czas ostatniej modyfikacji
    time_t    st_ctime;     czas ostatniej zmiany statusu
};
```

L11.Z4. Jak zrealizować poniższe scenariusze wykorzystując flagi wywołania systemowego `open`(2)?

open, openat, creat - open and possibly create a file

Dopisywanie do pliku komunikatów diagnostycznych z wielu procesów z uniknięciem wyścigów.

O_APPEND

The file is opened in append mode. Before each write(2), the file offset is positioned at the end of the file, as if with lseek(2). The modification of the file offset and the write operation are performed as a single atomic step.

Nowy program nie może otrzymać otwartego pliku od procesu, który wywołał `execve`(2).

By default, the new file descriptor is set to remain open across an execve(2) (i.e., the **FD_CLOEXEC** file descriptor flag described in fcntl(2) is initially disabled); the **O_CLOEXEC** flag, described below, can be used to change this default. The file offset is set to the beginning of the file (see lseek(2)).

Dwie możliwości:

1. otwieramy plik z flagą `O_CLOEXEC`

By default, the new file descriptor is set to remain open across an `execve`(2); the `O_CLOEXEC` flag can be used to change this default.

2. otwieramy plik, manipulujemy fdem z flagą `FD_CLOEXEC`

```
int fd = open(...);
fcntl(fd, F_SETFD, FD_CLOEXEC);
```

`fcntl` – manipulowanie fdem
`F_SETFD` – ustawianie flag fda
`FD_CLOEXEC` – zapobiega dziedziczeniu fdów

Umożliwiamy otworzenie nowego pliku dopiero wtedy, gdy zakończymy do niego zapis.

O_TMPFILE

Create an unnamed temporary regular file. The *pathname* argument specifies a directory; an unnamed inode will be created in that directory's filesystem. Anything written to the resulting file will be lost when the last file descriptor is closed, unless the file is given a name.

O_TMPFILE must be specified with one of **O_RDWR** or **O_WRONLY** and, optionally, **O_EXCL**. If **O_EXCL** is not specified, then linkat(2) can be used to link the temporary file into the filesystem, making it permanent, using code like the following:

```
char path[PATH_MAX];
fd = open("/path/to/dir", O_TMPFILE | O_RDWR,
          S_IRUSR | S_IWUSR);

snprintf(path, PATH_MAX,  "/proc/self/fd/%d", fd);
linkat(AT_FDCWD, path, AT_FDCWD, "/path/for/file",
         AT_SYMLINK_FOLLOW);
```

Istnienie danego pliku wyraża fakt założenia blokady na zasób współdzielony przez procesy.

O_CREAT | O_EXCL

Flaga `O_EXCL` – zapewnia, że wywołanie tworzy plik. W połączeniu z `O_CREAT`, jeśli *pathname* już istnieje, `open`() zawodzi i powoduje error `EEXIST`.

The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing `open`() naming the same filename in the same directory with `O_EXCL` and `O_CREAT` set.

L11.Z5. Zawartość pliku katalogu składa się z rekordów (**i-węzeł**, nazwa, typ-pliku). Czemu użytkownik nie ma bezpośredniego dostępu do pliku katalogu przy pomocy wywołań `read` i `write`? Podaj listę istotnie różnych wywołań systemowych realizujących operacje na katalogach. Czy zawartość katalogu jest posortowana? Kiedy wykonywana jest operacja **kompaktowania**?

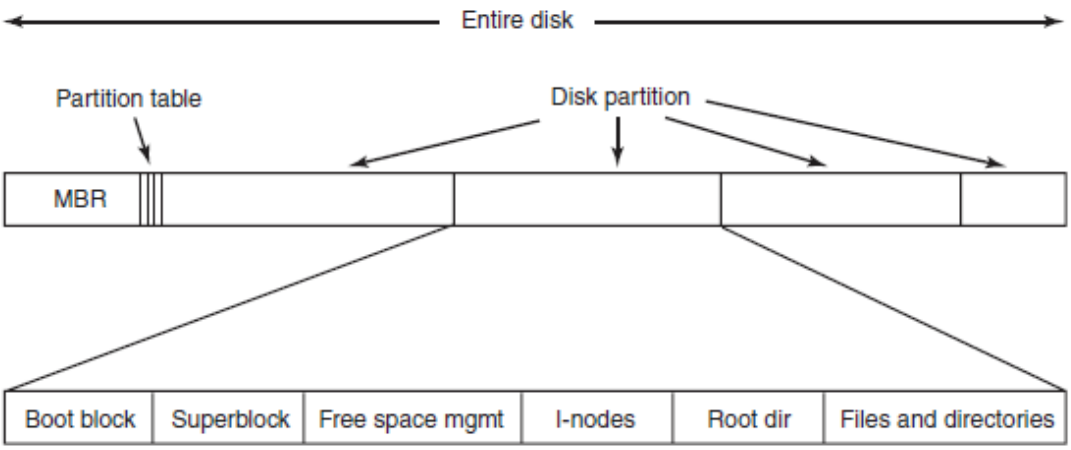


Figure 4-9. A possible file-system layout.

Our last method for keeping track of which blocks belong to which file is to associate with each file a data structure called an i-node (index-node), which lists the attributes and disk addresses of the file's blocks. Given the i-node, it is then possible to find all the blocks of the file.

Struktura i-node'a składa się z atrybutów pliku i wskaźników na bloki, z których składa się plik. Tannenbaum, jako przykładową realizację takiej struktury danych podaje:

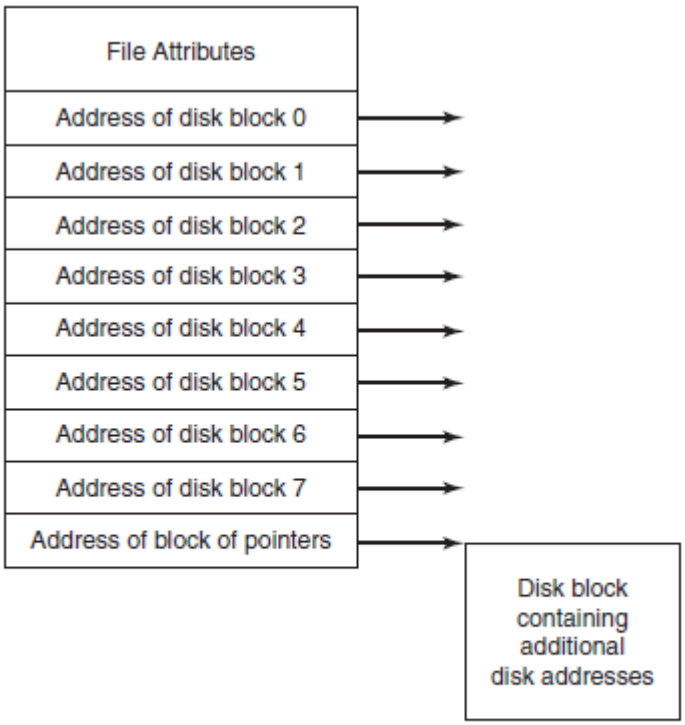


Figure 4-13. An example i-node.

Długość listy bloków rośnie proporcjonalnie do liczby bloków, które zajmuje plik, czyli w najgorszym razie proporcjonalnie do wielkości dysku. Co jeśli plik przekroczy sfixowaną liczbę adresów, którą zawiera nasz i-node? Wtedy możemy zarezerwować ostatni wskaźnik na blok na dysku, który zawiera dodatkowe wskaźniki na kolejne bloki.

Ważne:

- i-node nie zawiera nazwy pliku.

- Katalogi zawierają listę struktur, które przechowują jedną nazwę pliku i odpowiadający mu i-node
- W celu znalezienia pliku system musi znaleźć jego nazwę w katalogu i przekonwertować ją na numer i-node

Nie ma wymogu, co do tego, żeby zawartość folderów była w jakikolwiek sposób posortowana. Istnieją za to sposoby, żeby przyspieszyć wyszukiwanie pliku, jak pisze Tannenbaum „For extremely long directories, linear searching can be slow. One way to speed up the search is to use a hash table in each directory. […] A different way to speed up searching large directories is to cache the results of searches. Before starting a search, a check is first made to see if the file name is in the cache. If so, it can be located immediately."

Czemu użytkownik nie ma bezpośredniego dostępu do pliku katalogu przy pomocy wywołań read i write?

Katalogi przechowują swoje dane w blokach wskazywanych przez i-node, więc bezpośrednie pisanie do tych bloków skutkowałoby uszkodzeniem struktury systemu plików.

Podaj listę istotnie różnych wywołań systemowych realizujących operacje na katalogach.

MKDIR – Make dat dir
MKDIRAT - Create a directory relative to a directory file descriptor
RMDIR – Delete directory (dir must be empty to succeed)
GETCWD - Get current working directory
CHDIR - Change working directory
CHROOT – Change root directory
GETDENTS - Get directory entries (pobranie wpisów z katalogu)

Kompaktowanie – konsolidacja wolnego miejsca na dysku poprzez przesuwanie danych z dalszych części dysku do pustego miejsca w we wcześniejszych częściach. Kiedy? Kiedy poprosisz.

Jako ciekawostka i-node w ext2:

```
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        unsigned short i_uid;           /* Owner Uid */
        unsigned long i_size;           /* Size in bytes */
        unsigned long i_atime;          /* Access time */
        unsigned long i_ctime;          /* Creation time */
        unsigned long i_mtime;          /* Modification time */
        unsigned long i_dtime;          /* Deletion Time */
        unsigned short i_gid;           /* Group Id */
        unsigned short i_links_count;   /* Links count, max 32000 */
        unsigned long i_blocks;         /* Blocks count */
        unsigned long i_flags;          /* File flags (append only,
                                        immutable, etc.) */
        unsigned long i_reserved1;
        unsigned long i_block[15];      /* Pointers to blocks */
        unsigned long i_version;        /* File version (for NFS) */
        unsigned long i_file_acl;       /* File ACL */
        unsigned long i_dir_acl;        /* Directory ACL */
        unsigned long i_faddr;          /* Fragment address */
        unsigned char i_frag;           /* Fragment number */
        unsigned char i_fsize;          /* Fragment size */
        unsigned short i_pad1;
        unsigned long i_reserved2[2];
};
```

L11.Z6. Wyjaśnij różnice w sposobie implementacji **dowiązań twardych** (ang. *hard link*) i **symbolicznych** (ang. *symbolic link*). Podaj listę istotnie różnych wywołań systemowych realizujących operacje na dowiązaniach. Jak za pomocą dowiązania symbolicznego stworzyć w systemie plików pętlę? Kiedy jądro systemu operacyjnego ją wykryje? Czemu nie można utworzyć dowiązania twardego do pliku znajdującego się w innym systemie plików?

6
Dowiązania twarde to wskaźniki na i-węzły (licznik referencji!) plików → różne nazwy tego samego pliku w obrębie jednego systemu plików.

Dowiązania symboliczne kodują ścieżkę do której należy przekierować algorytm rozwiązywania nazw.

A soft link does not contain the data in the target file.
A soft link points to another entry somewhere in the file system.
A soft link has the ability to link to directories, or to files on remote computers networked through NFS.
Deleting a target file for a symbolic link makes that link useless.
A hard link preserves the contents of the file.
A hard link cannot be created for directories, and they cannot cross filesystem boundaries or span across partitions.
In a hardlink you can use any of the hardlink names created to execute a program or script in the same manner as the original name given

“Underneath the file system files are represented by inodes
A file in the file system is basically a link to an inode.
A hard link then just creates another file with a link to the same underlying inode.
When you delete a file it removes one link to the underlying inode. The inode is only deleted (or deletable/over-writable) when all links to the inode have been deleted.
A symbolic link is a link to another name in the file system.
Once a hard link has been made the link is to the inode. deleting renaming or moving the original file will not affect the hard link as it links to the underlying inode. Any changes to the data on the inode is reflected in all files that refer to that inode.
Note: Hard links are only valid within the same File System. Symbolic links can span file systems as they are simply the name of another file.”

syscalle: link/linkat - tworzy nowy hard link (make a new name for a file)

	Number	Description
LINK	86	Create a hard link to a file
LINKAT	265	Create a hard link to a file relative to directory file descriptors
SYMLINK	88	Create a symbolic link to a file
SYMLINKAT	266	Create a symbolic link to a file relative to a directory file descriptor
UNLINK	87	Delete a name and possibly the file it refers to
UNLINKAT	263	Delete a name and possibly the file it refers to relative to a directory file descriptor
READLINK	89	Read value of a symbolic link
READLINKAT	267	Read value of a symbolic link relative to a directory file descriptor

pętla:
$ mkdir a a/b
$ cd a/b
$ ln -s .. a
cd a/b/a/b/a/b/a/b/a/

kiedy wykryje?
nie do końca tu o pętle chodzi ale:
tako rzecze man path_resolution:

If the component is found and is a symbolic link (symlink), we first resolve this symbolic link (with the current lookup directory as starting lookup directory). Upon error, that error is returned. If the result is not a directory, an ENOTDIR error is returned. If the resolution of the symlink is successful and returns a directory, we set the current lookup directory to that directory, and go to the next component. Note that the resolution process here involves recursion. In order to protect the

kernel against stack overflow, and also to protect against denial of service, there are limits on the maximum recursion depth, and on the maximum number of symbolic links followed. An ELOOP error is returned when the maximum is exceeded ("Too many levels of symbolic links").

czyli zwróci jak przekroczy liczbę symlinków

czemu nie można utworzyć dowiązania twardego:
The File system is composed by a directory structure composed for directory entries to organize files. Each directory entry associates a file-name with an inode.
As the inode is a data structure used to represent a file-system object, it's internal to the File system, and you can't point to an inode of another file system.

L11.Z7. Jaką rolę pełnią **uprawnienia** «`rwx`» dla katalogów w systemach uniksowych? Opisz krótko zastosowanie dodatkowych uprawnień: «`set-uid`», «`set-gid`» i «`sticky`». Przedstaw algorytm określania dostępu do pliku dla zadanego **właściciela** i **grupy**. Podaj przykład, w którym standardowy system kontroli dostępu jest zbyt ograniczony i należy użyć **ACL** (ang. *access control list*).

uprawnienia – mechanizm umożliwiający określenie uprawnień odczytu, edycji i uruchamiania zawartości systemu plików dla poszczególnych użytkowników.

właściciel – użytkownik, który stworzył plik / folder

grupa – kolekcja użytkowników, każdy nowy użytkownik należy do jakiejś grupy.

Standardowym symbolicznym sposobem zapisu uprawnień jest:
t uuu ggg ooo

t - oznacza typ pliku (- zwykły, d katalog, l dowiązanie symboliczne, s gniazdo, f FIFO, c urządzenie znakowe, b urządzenie blokowe)
u - uprawnienia właściciela
g - uprawnienia grupy
o - uprawnienia pozostałych

Każda trójka oznacza:
- pierwszy symbol – r: readable
- drugi symbol – w: writable
- trzeci symbol –
  - x: executable
  - s lub t – setuid/setgid lub sticky (executable)
  - S lub T – setuid/setgid lub sticky (nie executable)

Zapis liczbowy praw dostępu polega na przedstawieniu uprawnień za pomocą trzycyfrowej lub czterocyfrowej liczby ósemkowej, której kolejne cyfry oznaczają: uprawnienia specjalne (w liczbie trzycyfrowej nie występuje), uprawnienia użytkownika, uprawnienia grupy, uprawnienia pozostałych
- rwx r-x r-- → 111 101 100 $_{(2)}$ → 4+2+1 4+1 4 $_{(8)}$ → 754 $_{(8)}$

Jaką rolę pełnią uprawnienia rwx dla katalogów w systemach uniksowych?

- r: odczyt; pozwala wyświetlić zawartość folderu, tj. nazwy plików, ale żadnych innych informacji (np. typu pliku, zawartości, rozmiaru, właściciela)
- w: modyfikacja; można w nim tworzyć i kasować pliki/katalogi -> modyfikacja tego i-node'a, danych w tym i-nodzie
- x: dostęp interpretowany jako „pozwolenie na szukanie" – jeśli znamy nazwę pliku, to daje możliwość dostępu do jego zawartości i meta-info na jego temat; nie daje możliwości wylistowania nazw plików, jeśli nie ma uprawnienia r

Opisz krótko zastosowanie dodatkowych uprawnień: «`set-uid`», «`set-gid`» i «`sticky`».

- set-uid / set-gid dla plików - w momencie ładowania pliku nadawane są uprawnienia właściciela / grupy pliku; umożliwia uruchamianie programów, które do poprawnej pracy wymagają wyższych uprawnień niż te, które typowy użytkownik systemu zazwyczaj posiada, np. passwd, ping potrzebują uprawnień roota.
- set-uid dla katalogów - atrybut ignorowany.
- set-gid dla katalogów - wszystkie nowo tworzone wpisy (pliki, katalogi itp.) stają się własnością grupy, będącej właścicielem katalogu. Atrybut ten jest zazwyczaj dziedziczony przez nowo tworzone podkatalogi.

- sticky
  - pliki: w przypadku pliku wykonywalnego powoduje wymuszenie przechowywania jego kodu w pamięci (ustawianie Sticky na plikach wykonywalnych jest obecnie rzadko stosowane)
  - katalogi: uniemożliwia usuwanie, przenaszanie, zmianę nazwy przez nie-właścicieli (poza rootem), nawet jeśli mają write permission.

Przedstaw algorytm określania dostępu do pliku dla zadanego właściciela i grupy.

1. Jeśli właścicielem pliku jest dany proces, używane są bity uprawnień usera.
2. W przeciwnym przypadku, jeśli grupa posiadająca plik ma efektywny identyfikator grupy procesu (lub jednego z dodatkowych grup), wtedy używane są bity uprawnień grupy.
   (Otherwise, if the file's owning group is the process's effective GID or one of the process's supplementary group ID, then the group permission bits are used.)
3. W przeciwnym przypadku, używane są bity uprawnień dla reszty.

Supplementary groups [edit]

In Unix systems, every user must be a member of at least one group, the **primary group** which is identified by the numeric GID of the user's entry in the group database, which can be viewed with the command `getent passwd` (usually stored in `/etc/passwd` or LDAP). This group is referred to as the *primary group ID*. A user may be listed as member of additional groups in the relevant entries in the group database, which can be viewed with `getent group` (usually stored in `/etc/group` or LDAP); the IDs of these groups are referred to as *supplementary group IDs*.

Effective vs. real [edit]

Unix processes have an effective (EUID, EGID), a real (UID, GID) and a saved (SUID, SGID) ID. Normally these are identical, but in `setgid` processes they are different.

Podaj przykład, w którym standardowy system kontroli dostępu jest zbyt ograniczony i należy użyć ACL.

Jednak co zrobić, jeżeli chcemy dać jakiemuś konkretnemu użytkownikowi (użytkownikom) dostęp do naszego pliku? Jeśli do dyspozycji mamy tylko standardowy mechanizm uprawnień, to mamy właściwie dwie możliwości: dodanie go do naszej grupy i umożliwienie wszystkim jej członkom dostępu do pliku lub udostępnienie tego pliku wszystkim. Tutaj z pomocą przychodzi ACL - listy kontroli dostępu do plików.

ACL – listy kontroli dostępu do plików, pozwalają (oprócz tego, co zwykłe uprawnienia), także na ustawienie praw dostępu dla dowolnego użytkownika / grupy.

Algorytm sprawdzania uprawnień

1. jeśli użytkownik jest właścicielem pliku - zastosuj uprawnienia właściciela,
2. jeżeli użytkownik jest na liście nazwanych użytkowników - zastosuj efektywne uprawnienia nazwanego użytkownika,
3. jeżeli jedna z grup użytkownika jest grupą właściciela i posiada odpowiednia efektywne prawa - dopuść,
4. jeżeli jedna z grup użytkownika występuje jako grupa nazwana i posiada odpowiednie efektywne prawa - dopuść,
5. jeżeli jedna z grup użytkownika jest grupą właściciela lub należy do grup nazwanych, ale nie posiada dostatecznych efektywnych uprawnień - dostęp jest zabroniony,
6. następnie uprawnienia pozostałych określają możliwość dostępu.

Do wyświetlenia atrybutów ACL jakiegoś pliku służy polecenie:
`getfacl nazwa_pliku`, np.:

```
$ getfacl plik.txt
# file: plik.txt
# owner: user
# group: group
user::rw-
group::r--
other::r—
```

Aby ustawić ACL dla jakiegoś pliku, korzystamy z polecenia:
`setfacl nazwa_pliku`, np.

```
% setfacl -m u:testowy:rwx plik.txt
```

Jeżeli natomiast wypiszemy zawartość katalogu, to przy atrybutach pliku, który ma ACL pojawi się plusik:

```
% ls -l
-rw-rwxr--+ 1 user group 1000 2009-09-09 09:09 plik.txt
```