Rozdział 1

Algorytmy `link` i `unlink`

Autor: Paweł Wolff
Tematem tego rozdziału jest opis algorytmów funkcji sytemowych `link` i `unlink`, służących do tworzenia i usuwania **twardych dowiązań** (ang. *hard links*) do plików oraz opis struktur danych używanych przez te funkcje. Opisy te dotyczą jądra systemu Linux w wersji 2.2.17 z systemem plików *ext2*.

1.1 Wprowadzenie

Niniejszy podrozdział, oprócz ogólnego zarysu sposobu i efektów działania ww. funkcji systemowych, zawiera także wstępne informacje na temat struktur danych systemu Linux, na których powyższe funkcje operują. Omówiona zostanie organizacja systemu plików, w zakresie wyjaśniającym m.in. sposób reprezentacji dowiązań. Natomiast szczegóły dotyczące struktur danych systemu plików, w tym także opisy fragmentów implementacji tych struktur, znajdują się w podrozdziale 1.2.

1.1.1 Organizacja systemu plików

System plików systemu Linux reprezentuje pliki w hierarchicznej strukturze, która niekoniecznie jest drzewem. Sciśle rzecz biorąc, jest to graf skierowany z wyróżnionym wierzchołkiem reprezentującym katalog główny, z którego osiągalne są wszystkie pozostałe wierzchołki (czyli pliki).

Pliki służące do organizowania takiej struktury to katalogi oraz **dowiązania symboliczne** (ang. *symbolic links*). Katalogi umożliwiają organizacje plików w strukturę drzewiastą, natomiast dowiązania symboliczne — dodawanie do tej struktury krawędzi niedrzewowych.

Każdy plik (a więc również katalog i dowiązanie symboliczne) w omawianym systemie plików jest jednoznaczenie reprezentowany przez strukturę danych zwaną **i-węzełem** (ang. *i-node*). Struktura ta opisuje m.in. następujące atrybuty pliku:

– identyfikator użytkownika i grupy użytkowników pliku

– czas ostatniej modyfikacji i ostatniego dostępu do pliku

– prawa dostępu do pliku

– typ pliku (określa np. czy dany plik jest zwykłym plikiem, katalogiem, dowiązaniem symbolicznym, itp.)

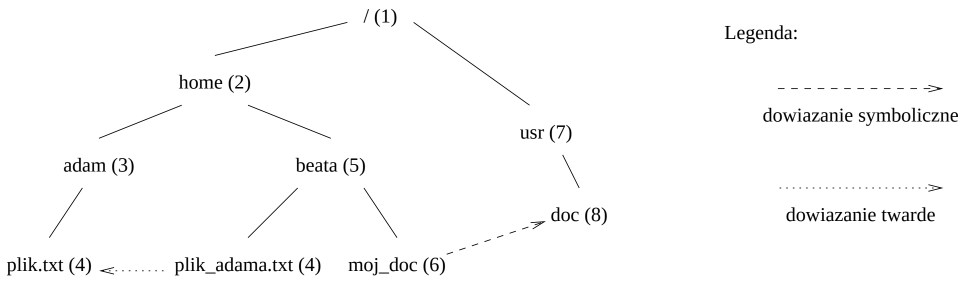
– lista bloków dyskowych, na których są zapisane dane zawarte w pliku

Ponadto, każdy i-węzeł posiada unikatowy numer, co pozwala na jednoznaczne kojarzenie pliku z numerem i-węzła.

W tym miejscu warto wyjaśnić, jakie informacje zawiera plik, który jest katalogiem lub dowiązaniem symbolicznym. Otóż w przypadku katalogu informacją tą jest lista wpisów katalogowych. Każdy z takich wpisów zawiera nazwę pliku i numer i-węzła, jaki ma być kojarzony z tą nazwą. Natomiast plik, będący dowiązaniem symbolicznym, zawiera nazwę scieżki (bezwzględną lub względną) do dowiązywanego pliku.

1.1.2 Dowiązania twarde

Dowiązanie twarde (zwane także trwałym) jest reprezentowane przez wpis katalogowy, należący do katalogu, w którym znajduje się dane dowiązanie. Wpis ten zawiera nazwę dowiązania oraz numer i-węzła pliku, na który dane dowiązanie wskazuje.



Rysunek 1.1: Przykładowa struktura plików

Przykład
Na rysunku 1.1 została przedstawiona przykładowa struktura plików (ściślej: struktura wpisów katalogowych). Numery w nawiasach oznaczają numery i-węzłów, do których odnoszą się poszczególne wpisy.

Zawartością i-węzła nr 2 jest lista dwóch wpisów:

```
adam  3
beata 5
```

natomiast zawartością i-węzła nr 6 jest wpis:

```
/usr/doc
```

Podstawowa różnica między dowiązaniem twardym a symbolicznym polega na tym, że w przeciwieństwie do dowiązania symbolicznego, dowiązanie twarde nie jest reprezentowane przez osobny i-węzeł (informacja o nim zawarta jest tylko w katalogu, zawierającym to dowiązanie). Drugą różnicą jest brak możliwości tworzenia twardych dowiązań do katalogów[1]. Trzecia — to brak możliwości utworzenia twardego dowiązania do nieistniejącego pliku.

Warto jeszcze wspomnieć o fakcie nierozróżnialności pierwotnego wpisu katalogowego danego pliku (tzn. wpisu powstałego w wyniku utworzenia tego pliku w danym katalogu) od twardych dowiązań do tego pliku. Tak więc wszystkie wpisy katalogowe, odnoszące się do danego i-węzła, można traktować jako twarde dowiązania do pliku reprezentowanego przez ten i-węzeł. W związku z tym, w przypadku usunięcia wszystkich twardych dowiązań do danego pliku, nie ma już do niego dostępu i może on zostać usunięty wraz ze swoim i-węzłem[2].

1.1.3 Działanie funkcji systemowej `link`

Funkcja systemowa `link`, dla zadanych dwóch parametrów `oldpath` i `newpath`, tworzy twadre dowiązanie, o nazwie określonej w parametrze `newpath`, do pliku o nazwie `oldpath`. Plik, do którego tworzone jest dowiązanie musi istnieć i nie może być katalogiem, natomiast nie może istnieć plik o nazwie `newpath`.

Tworzenie twardego dowiązania odbywa się w następujący sposób (pomijając na razie wiele szczegółów):

1. znajdowany jest i-węzeł reprezentujący katalog, w którym ma zostać umieszczone dowiązanie

2. znajdowany jest i-węzeł reprezentujący plik o nazwie `oldpath`

3. do katalogu (i-węzła) znalezionego w punkcie 1 dodawany jest wpis katalogowy o nazwie określonej przez parametr `newpath`, wskazujący na i-węzeł znaleziony w punkcie 2

Szczegółowy opis algorytmu znajduje się w podrozdziale 1.3.

1.1.4 Działanie funkcji systemowej `unlink`

Funkcja systemowa `unlink`, dla zadanego parametru `pathname`, usuwa twarde dowiązanie o nazwie określonej w tym parametrze. Jeśli usuwane dowiązanie jest ostatnim dowiązaniem do pewnego pliku, to plik ten jest usuwany, o ile żaden inny proces z tego pliku nie korzysta (w przeciwnym razie zostanie on usunięty dopiero wtedy, gdy wszystkie procesy przestaną z niego korzystać).

Usuwanie twardego dowiązania odbywa się w następujący sposób (pomijając na razie wiele szczegółów):

1. znajdowany jest i-węzeł reprezentujący katalog, z którego ma zostać usunięte dowiązanie

2. z katalogu (i-węzła) znalezionego w punkcie 1 usuwany jest wpis katalogowy o nazwie określonej przez parametr `pathname`

[1] dokumentacja polecenia `ln` (patrz: `man ln`) zawiera błedną informację, dotyczącą możliwości tworzenia dowiązania twardego do katalogu przez nadzorcę systemu

[2] o ile żaden proces nie korzysta z tego pliku; w przeciwnym razie usuwanie następuje dopiero wtedy, gdy wszystkie procesy przestaną z niego korzystać

3. jeśli było to ostatnie dowiązanie do pewnego i-węzła, to ten i-węzeł jest w miarę możliwości usuwany, a wskazywne przez niego bloki dyskowe — zwalniane

Szczegółowy opis algorytmu znajduje się w podrozdziale 1.4.

1.2 Wybrane struktury danych systemu plików

W tym podrozdziale znajduje się opis dwóch struktur danych, używanych przez opisywane funkcje systemowe — struktury `inode` oraz struktury `dentry`. Pierwsza z nich opisuje i-węzły, druga — wpisy katalogowe. Warto od razu wspomnieć, że w przeciwieństwie do danych typu `inode`, dane typu `dentry` nie są zapisywane na trwałym nośniku danych (np. na dysku twardym), a jedynie w pamięci jądra. Jednak informacje, jakie ze sobą niosą dane typu `dentry` (tj. informacje o strukturze plików w systemie) są takie same, jak te zapisane na dysku w katalogach i dowiązaniach symbolicznych. Wobec tego przechowywanie w pamięci jądra struktur `dentry` ma jedynie na celu przyspieszenie operacji przeglądania katalogów (por. sekcja 1.5.2). Oczywiście struktury te są tworzone sukcesywnie, wraz z kolejnymi odczytami z dysku tych katalogów, o których nie ma informacji w pamięci.

1.2.1 Struktura `inode`

Poniżej został zamieszczony fragment deklaracji struktury `inode` (z dodanymi komentarzami), opisującymi znaczenia niektórych pól. Całość deklaracji znajduje się w [include/linux/fs.h].

```
struct inode {
  struct list_head  i_dentry;
  unsigned long     i_ino;
  unsigned int      i_count;
  kdev_t            i_dev;
  umode_t           i_mode;
  nlink_t           i_nlink;
  uid_t             i_uid;      /* id. właściciela pliku */
  gid_t             i_gid;      /* id. grupy właściciela pliku */
  off_t             i_size;     /* rozmiar pliku w bajtach */
  time_t            i_atime;    /* czas ostatniego dostępu do pliku */
  time_t            i_mtime;    /* czas ostatniej modyfikacji pliku */
  time_t            i_ctime;    /* czas utworzenia pliku */
  unsigned long     i_blocks;   /* liczba bloków dyskowych zajmowanych
                                   przez plik */
  struct semaphore  i_sem;
  struct inode_operations *i_op;
};
```

Opis znaczenia najważniejszych pól (z punktu widzenia przedstawianych tutaj funkcji systemowych) znajduje się w tablicy 1.1

Tablica 1.1: Opis ważniejszych pól struktury `inode`

Typ	**Nazwa**	**Znaczenie**
`struct list_head`	`i_dentry`	lista wpisów katalogowych, odnoszących się do danego i-węzła
`unsigned long`	`i_ino`	unikatowy numer i-węzła
`unsigned int`	`i_count`	licznik odwołań procesów do i-węzła
`kdev_t`	`i_dev`	identyfikator systemu plików, do którego dany i-węzeł należy
`umode_t`	`i_mode`	określa typ i-węzła (np. czy jest katalogiem) oraz prawa dostępu
`nlink_t`	`i_nlink`	licznik twardych dowiązań do i-węzła
`struct semaphore`	`i_sem`	semafor binarny służący do blokowania i-węzła
`struct inode_operations`	`*i_op`	wskaźniki do funkcji chrakterystycznych dla odpowiedniego systemu plików (np. *ext2*), operujące na danym i-węźle

Przykład
Poniżej znajdują się wartości niektórych pól struktury `inode`, reprezentującej i-węzeł o numerze 4 (por. rysunek 1.1).

Nazwa	**Wartość pola**
`i_dentry`	lista wpisów: `/home/adam/plik.txt`, `/home/beata/plik_adama.txt`
`i_ino`	4
`i_nlink`	2

1.2.2 Struktura `dentry`

Poniżej został zamieszczony fragment deklaracji struktury `dentry` (wraz z oryginalnymi komentarzami). Całość deklaracji znajduje się w [include/linux/dcache.h].

```
struct dentry {
  int                 d_count;
  unsigned int        d_flags;
  struct inode       *d_inode;    /* Where the name belongs to -
                                     - NULL is negative */
  struct dentry      *d_parent;   /* parent directory */
  struct list_head    d_child;    /* child of parent list */
  struct list_head    d_subdirs;  /* our children */
  struct list_head    d_alias;    /* inode alias list */
  struct qstr         d_name;
  struct dentry_operations *d_op;
  struct super_block *d_sb;       /* The root of the dentry tree */
  void * d_fsdata;                /* fs-specific data */
```

```
};
```

Opis znaczenia najważniejszych pól (z punktu widzenia przedstawianych tutaj funkcji systemowych) znajduje się w tablicy 1.2

Tablica 1.2: Opis ważniejszych pól struktury `dentry`

Typ	**Nazwa**	**Znaczenie**
`int`	`d_count`	licznik odwołań procesów do danego wpisu katalogowego
`struct inode`	`*d_inode`	wskaźnik na i-węzeł, do którego dany wpis się odnosi
`struct dentry`	`*d_parent`	wskaźnik na ojca w strukturze katalogów
`struct list_head`	`d_subdirs`	lista wpisów katalogowych, które zapisane są w i-węźle wskazywanym przez `d_inode`
`struct list_head`	`d_alias`	lista wpisów katalogowych, które odnoszą się do tego samego i-węzła, co dany
`struct qstr`	`d_name`	tu jest zapisana m.in. nazwa danego wpisu katalogowego

Przykład
Poniżej znajdują się wartości niektórych pól struktury `dentry`, reprezentującej wpis `plik_adama.txt` znajdujący się w katalogu `/home/beata` (por. rysunek 1.1).

Nazwa	**Wartość pola**
`d_inode`	wskaźnik do i-węzła o numerze 4
`d_parent`	wskaźnik do wpisu o nazwie ścieżki `/home/beata`
`d_subdirs`	lista pusta
`d_alias`	lista wpisów: `/home/adam/plik.txt`, `/home/beata/plik_adama.txt`
`d_name`	zawiera m.in. nazwę tego wpisu, tj: `plik_adama.txt`

1.3 Algorytmu funkcji systemowej `link`

Algorytm nr 1 zawiera opis działania algorytmu `link`. Ze względu na czytelność tego opisu, funkcje w nim wymieniane zostaną skomentowane nieco bardziej szczegółowo w podrozdziale 1.5.

Kod źródłowy tego algorytmu można znaleźć w:

[fs/namei.c] — funkcje `sys_link(...)`, `do_link(...)`,

[fs/ext2/namei.c] — funkcja `ext2_link(...)`

1.4 Algorytmu funkcji systemowej `unlink`

Algorytm nr 2 zawiera opis działania algorytmu `unlink`. Podobnie jak poprzednio, funkcje w nim wymieniane zostaną skomentowane w podrozdziale 1.5.

Algorytm 1 link

Dane wejściowe: `newpath`, `oldpath`

Wynik: tworzy twarde dowiązanie o nazwie scieżki `newpath` do pliku o nazwie ścieżki `oldpath`

zablokuj wejście do jądra — funkcja `lock_kernel()`
`old_dentry` := wpis katalogowy o nazwie `oldpath` — funkcja `lookup_dentry(oldpath, ...)`
`new_dentry` := wpis katalogowy o nazwie `newpath` — funkcja `lookup_dentry(newpath, ...)`
zablokuj i-węzeł reprezentujący katalog, w którym został znaleziony wpis `new_dentry` — funkcja `lock_parent(new_dentry)`
`dir` := wpis katalogowy reprezentujący zablokowany katalog
jeśli `dir` ≠ `new_dentry->d_parent` **to**
  zakończ z błędem
koniec jeśli
jeśli brak praw zapisu w katalogu `dir` lub niewłaściwy typ tego katalogu (sprawdzanie flag `dir->d_inode->i_mode`) **to**
  zakończ z błędem
koniec jeśli
`inode` := `old_dentry->d_inode`
jeśli `dir->d_inode->i_dev` ≠ `inode->i_dev` **to**
  zakończ z błędem
koniec jeśli
jeśli `inode->i_nlink` ≥ `EXT2_LINK_MAX` **to**
  zakończ z błędem
koniec jeśli
do i-węzła `dir->d_inode` dopisz informację o wpisie `newpath`, wskazującym na `inode` — funkcja `ext2_add_entry(...)`
`inode->i_nlink++`
`inode->i_count++`
`new_dentry->d_inode` := `inode`
dołącz `new_dentry` do listy `inode->i_dentry` i przypisz tę listę pod `new_dentry->d_alias`
odblokuj i-węzeł, na który wskazuje wpis `dir` — funkcja `unlock_dir(dir)`
zwolnij wpis `new_dentry` — funkcja `dput(new_dentry)`
zwolnij wpis `old_dentry` — funkcja `dput(old_dentry)`
odblokuj wejście do jądra — funkcja `unlock_kernel()`

Algorytm 2 unlink

Dane wejściowe: `pathname`

Wynik: usuwa twarde dowiązanie o nazwie scieżki `pathname`

```
zablokuj wejście do jądra — funkcja lock_kernel()
dentry := wpis katalogowy o nazwie pathname — funkcja lookup_dentry(pathname, ...)
zablokuj i-węzeł reprezentujący katalog, w którym został znaleziony wpis dentry — funkcja lock_parent(dentry)
dir := wpis katalogowy reprezentujący zablokowany katalog
jeśli dir ≠ dentry->d_parent to
    zakończ z błędem
koniec jeśli
jeśli brak praw zapisu w katalogu dir lub niewłaściwy typ tego katalogu (sprawdzanie flag dir->d_inode->i_mode) to
    zakończ z błędem
koniec jeśli
inode := dentry->d_inode
usuń z i-węzła dir->d_inode wpis pathname — funkcja ext2_delete_entry(...)
inode->i_nlink--
usuń dentry z listy inode->i_dentry oraz z listy inode->i_dentry->d_alias — funkcja d_delete(dentry)
jeśli inode->i_nlink = 0 — funkcja iput(inode) wywołana przez funkcję d_delete(dentry) z poprzedniego kroku to
    jeśli inode->i_count = 1 to
        usuń inode oraz zwolnij zajmowane przez niego bloki dyskowe
    koniec jeśli
w przeciwnym przypadku
    inode->i_count--
koniec jeśli
odblokuj i-węzeł, na który wskazuje wpis dir — funkcja unlock_dir(dir)
zwolnij wpis dentry — funkcja dput(dentry)
odblokuj wejście do jądra — funkcja unlock_kernel()
```

Kod źródłowy tego algorytmu można znaleźć w:

[fs/namei.c] — funkcje `sys_unlink(...)`, `do_unlink(...)`, `vfs_unlink(...)`,

[fs/ext2/namei.c] — funkcja `ext2_unlink(...)`

1.5 Opis funkcji używanych w algorytmach `link` i `unlink`

1.5.1 Funkcje `lock_kernel()` i `unlock_kernel()`

Funkcje te dostarczają mechanizm wzajemnego wykluczania się procesów w rejonie kodu jądra. Mechanizm taki jest potrzebny tylko w wypadku systemu wieloprocesorowego. W przypadku systemu z jednym procesorem, proces wykonujący kod jądra nie może zostać wywłaszczony, a jedynie sam może oddać procesor, gdy oczekuje na jakieś zdarzenie. Wobec tego nie dojdzie do nieporządanych przeplotów instrukcji jądra. Zatem w przypadku jądra Linuksa dla systemu jednoprocesorowego funkcje te wykonują instrukcję pustą.

Kod źródłowy tych funkcji można znaleźć w:

[include/linux/smp_lock.h]

[include/asm-i386/smplock.h]

1.5.2 Funkcja `lookup_dentry(pathname, ...)`

Funkcja ta przegląda strukturę katalogów w poszukiwaniu wpisu katalogowego o nazwie ścieżki `pathname`. Jeśli ścieżka zaczyna się od znaku „/”, to rozpoczynamy przeszukiwanie od katalogu głównego, w przeciwnym razie — od bieżącego katalogu procesu, który tego przeszukiwania dokonuje. Następnie kolejno każdy element ścieżki jest dopasowywany do pewnego wpisu w katalogu, w którym aktualnie odbywa się przeszukiwanie. Dokładniej rzecz biorąc na liście `d_subdirs` odszukiwana jest struktura `dentry` o takiej nazwie, jak rozpatrywany element ścieżki (w przypadku braków jakichś struktur `dentry`, informacje o nich są dynamicznie tworzone na podstawie danych pochodzących z dysku). Warto jeszcze zaznaczyć, że jeśli ostatni element ścieżki nie zostanie znaleziony wśród struktur `dentry`, zostanie dla niego utworzona nowa struktura `dentry`, odpowiednio podczepiona pod resztę.

Każdorazowo przed przeszukiwaniem wpisów katalogowych w aktualnym katalogu, jest on blokowany, tzn. opuszczany jest semafor `i_sem` i-węzła reprezentującego ten katalog. Po dopasowaniu wpisu (w ostatnim kroku może to oznaczać utworzenie nowego), wykonywana jest na nim funkcja `dget` (patrz sekcja 1.5.6), a następnie odblokowywany jest katalog, w którym ten wpis został znaleziony (lub utworzony). Teraz (o ile pozwalają na to prawa dostępu) aktualnym katalogiem staje się ostatnio znaleziony wpis i wyżej opisana procedura jest powtarzana, dopóki nie zostanie przetworzona cała ścieżka `pathname`.

Kod źródłowy tej funkcji można znaleźć w [fs/namei.c].

1.5.3 Funkcja `lock_parent(dentry)`

Funkcja ta wywołuje funkcję `dget(dentry->d_parent)`, czyli zwiększa licznik odwołań do katalogu, w którym znajduje się wpis `dentry` (por. sekcja 1.5.6), a także blokuje ten katalog (opuszcza semafor i-węzła `dentry->d_parent->d_inode`).

Kod źródłowy tej funkcji można znaleźć w [fs/namei.c]

1.5.4 Funkcja `unlock_dir(dir)`

Funkcja ta odblokowuje katalog `dir` (podnosi semafor `dir->d_inode->i_sem`) oraz wywołuje funkcję `dput(dir)` (patrz sekcja 1.5.6).

Kod źródłowy tej funkcji można znaleźć w [fs/namei.c].

1.5.5 Funkcje `ext2_add_entry(...)` i `ext2_delete_entry(...)`

Są to niskopoziomowe funkcje systemu plików *ext2*, służące odpowiednio do dodawania i usuwania wpisów katalogowych, zapisanych w blokach dyskowych (chodzi o bloki dyskowe wskazywane przez i-węzeł, reprezentujący katalog, do którego dodawane lub z którego usuwane są wpisy).

Kod źródłowy tych funkcji można znaleźć w [fs/ext2/namei.c].

1.5.6 Funkcje `dget(dentry)` i `dput(dentry)`

Funkcja `dget(dentry)` zwiększa licznik odwołań we wpisie `dentry` (zwiększa `dentry->d_count`), natomiast funkcja `dput(dentry)` — zmniejsza ten licznik, lecz nie tylko dla wpisu `dentry`, ale także rekurencyjnie dla przodków tego wpisu (w sensie struktury katalogowej). Jeśli w wyniku zmniejszenia licznika w pewnym wpisie `d`, licznik ten będzie miał wartość równą 0, to wpis `d` zostanie usunięty z pamięci jądra (poprzez wywołanie funkcji `d_delete(d)` — patrz 1.5.7).

Kod źródłowy tych funkcji można znaleźć w:

[include/linux/dcache.h] — funkcja `dget(...)`

[fs/dcache.c] — funkcja `dput(...)`

1.5.7 Funkcja `d_delete(dentry)`

Funkcja ta usuwa z pamięci jądra strukturę `dentry`, która nie jest przez żaden proces używana (tzn. `dentry->d_count` $= 0$), a także wywołuje funkcję `iput(dentry->d_inode)`.

Kod źródłowy tej funkcji można znaleźć w [fs/dcache.c].

1.5.8 Funkcja `iput(inode)`

Funkcja ta zmniejsza licznik odwołań do i-węzła `inode` (`inode->i_count`) oraz w przypadku, gdy licznik ten spadł do zera i `inode->i_nlink` $= 0$ — usuwa z dysku (a także z pamięci jądra) i-węzeł `inode`. Zwalniane są też bloki dyskowe, które zajmował.

Kod źródłowy tej funkcji można znaleźć w [fs/inode.c].

1.6 Zadania

1. Jakie prawa dostępu musi posiadać użytkownik o katalogu domowym `/home/beata`, aby stworzyć twarde dowiązanie jak na rysunku 1.1 ?
   Jeśli już takie dowiązanie stworzy, to jakie prawa dostępu będzie miał do tego pliku?

2. W jakim celu w algorytmach `link` i `unlink` blokowany jest katalog, w którym tworzone bądź usuwane jest dowiązanie?

3. Czy potrzebne jest sprawdzenie dokonywane w kroku 6 algorytmu 1 (a także w kroku 5 algorytmu 2)?

4. Jeśli przy usuwaniu ostatniego dowiązania do pewnego i-węzła, jakiś inny proces korzysta z niego, to kiedy i w jaki sposób ten i-węzeł zostanie usunięty?

1.7 Rozwiązania zadań

1. Użytkownik ten, oprócz prawa wejścia i zapisu do swojego katalogu, musi posiadać prawo wejścia i czytania katalogu `/home/adam`.
   Do utworzonego dowiązania będzie miał te same prawa, co do pliku `/home/adam/plik.txt`, gdyż obydwa są wpisami wskazującymi na ten sam i-węzeł (a przecież w i-węźle zapisane są prawa dostępu do pliku).

2. Jest to robione w celu wykluczenia możliwości czytania bądź pisania w katalogu, którego zawartość jest modyfikowana. W przypadku, gdyby taka blokada nie została założona, mogłoby się zdarzyć, że proces wykonujący np. algorytm `link` zawiesi się na operacji wejścia/wyjścia. Wówczas inny proces mógłby zapisać (lub odczytać) informację z tego samego katalogu. Nie jest to pożądana sytuacja, gdyż w tym momencie mogą tam znajdować się niespójne dane (ze względnu na to, że algorytm `link` nie został jeszcze w całości wykonany).

3. Tak, jest potrzebne. Może się np. zdarzyć sytuacja, w której proces wykonujący algorytm `link` musi czekać na możliwość zablokowania katalogu (patrz: krok 4 algorytmu 1), gdyż w tym czasie inny proces jest np. w trakcie usuwania tego katalogu. Wówczas po obudzeniu proces musi sprawdzić czy istotnie udało mu się zablokować ten katalog. Podobny scenariusz można podać dla algorytmu `unlink`.

4. Usunięcie nastąpi w momencie, gdy ostatni proces korzystający z danego i-węzła `inode`, przestanie go używać. Wówczas wywołana przez niego funkcja `iput(inode)` spowoduje usunięcie tego i-węzła, gdyż liczba twardych dowiązań już wcześniej spadła do zera, zaś licznik odwołań — jest właśnie zmieniany z 1 na 0.

Literatura

[Bach] Bach J. M.: *Budowa systemu operacyjnego UNIX.*

[Linux] Źródła jądra systemu Linux w wersji 2.2.17, w tym:

[include/linux/fs.h]

[include/linux/dcache.h]

[include/asm-i386/smplock.h]

[include/linux/smp_lock.h]

[fs/namei.c]

[fs/inode.c]

[fs/dcache.c]

[fs/ext2/namei.c]