Systemy operacyjne

Laboratorium 2

Dowiązania

Jarosław Rudy
Politechnika Wrocławska

28 lutego 2017

Laboratorium obejmuje umiejętność tworzenia i obsługi dowiązań symbolicznych oraz, w mniejszym stopniu, twardych. Laboratorium obejmuje także znajomość uniksowego systemu plików: katalogów, drzew katalogowych i ścieżek.

1 Pliki zwykłe i katalogi

Pliki zwykłe (zwane też plikami regularnymi) są podstawowym typem plików. Zawierają dane użytkowników. Pliki te mogą być tekstowe lub binarne. Mogą posiadać rozszerzenie (fragment nazwy po kropce, rozszerzenie może mieć więcej niż 3 znaki). Pliki mogą nie posiadać rozszerzeń lub posiadać więcej niż jedno rozszerzenie (np. `file.tar.gz`). Ponadto, jeśli pierwszym znakiem nazwy jest kropka, to plik jest traktowany jako ukryty tzn. nie jest domyślnie raportowany przez polecenie `ls`.

Katalogi są również typem plików, innym jednak niż pliki regularne. Również mogą posiadać rozszerzenia (choć nie mają one dla katalogów specjalnego znaczenia). Katalogi nie zawierają danych użytkownika, a ich zawartość nie może być wyświetlona poleceniem `cat`. Zawartością katalogu jest niczym innym jak listą wpisów. Każdy wpis jest parą składającą się z nazwy i tak zwanego numeru i-węzła (*ang.* i-node number).

I-węzły to struktury przechowywane przez system operacyjny, które są wykorzystane do dostępu do plików – w i-węźle umieszczone są odwołania do fizycznego pliku. Każdy plik ma przypisany numer i-węzła. I-węzły przechowują też szereg innych informacji o pliku (czas ostatniej modyfikacji, typ, rozmiar, prawa dostępu, liczba dowiązań twardych). I-węzły nie przechowują jednak nazw plików – te są częścią wpisów katalogowych.

Aby to zobrazować załóżmy katalog `DIR`, w którym znajdują się 3 pliki nazwane `FILE1`, `FILE2` i `FILE3`. Przykładowe wpisy tego katalogu mogą być postaci

takiej jak w poniższej tabeli:

nazwa pliku	numer i-węzła
`FILE1`	1033
`FILE2`	2480
`FILE3`	6200

Do listowania wpisów danego katalogu służy polecenie `ls`. Domyślnie wyświetla ono tylko nazwy plików, ale numery i-węzłów mogą zostać wyświetlone po dodaniu opcji `-i`. Komenda `stat` pozwala wyświetlić numer i-węzła podanego pliku (oraz większość informacji zawartych w tym i-węźle).

2 Drzewo systemu plików i ścieżki do pliku

System plików uniksowego systemu operacyjnego przyjmuje podstawowo postać drzewa. Korzeniem drzewa jest `"/"` (plik `root`). Plik ten ma dwie szczególne cechy: (1) katalogiem nadrzędnym dla `"/"` jest również `"/"` oraz (2) numer i-węzła tego pliku jest znany systemowi operacyjnemu.

Załóżmy teraz, że chcemy uzyskać dostęp do pliku danego ścieżką `/sciezka/jakiegos/pliku`. W tym przypadku system operacyjny (SO) najpierw sprawdzi wpisy katalogu `/`, a może to zrobić bo i-węzeł korzenia systemu plików jest znany SO. Jeśli wśród wpisów nie ma nazwy `sciezka`, to wystąpi błąd. Jeśli jednak wpis o nazwie `sciezka` istnieje, to SO dostaje się do odpowiedniego pliku i cykl się powtarza – tym razem SO poszukuje wpisu `jakiegos` w katalogu `sciezka`. Ostatecznie, system znajdzie wpis `pliku` w katalogu `jakiegos` i otworzy ten plik.

Podobny proces do tego opisanego powyżej ma miejsce w przypadku każdego dostępu do pliku na podstawie ścieżki. W ogólności wyróżniamy dwa podstawowe rodzaje ścieżek:

- Ścieżka bezwzględna (*ang.* absolute path). Ścieżka ta prawie zawsze[1] zaczyna się od ukośnika (korzeń systemu plików). Ścieżka bezwzględna do danego pliku jest niezależna od naszego aktualnego katalogu roboczego (CWD). Szczególnym przypadkiem ścieżki bezwzględnej jest ścieżka kanoniczna (o której później).

- Ścieżka względna (*ang.* relative path). Ścieżka zaczyna się od innego znaku niż ukośnik. Ścieżka ta liczona jest względem pewnego punktu systemu plików. Normalnie tym punktem jest CWD. System operacyjny albo pamięta i-węzeł związany z chwilowym CWD, albo dołącza ścieżkę CWD do ścieżki względnej, tworząc ścieżkę bezwzględną. Oczywistym jest, że ścieżka względna do danego pliku jest różna w zależności od CWD.

[1] Jednym z wyjątków jest sytuacja, gdy korzystamy ze znaku tyldy. Zostanie on jednak i tak przetłumaczony na ścieżkę bezwzględną na poziomie `basha`.

Przykładowo, żałóżmy że ścieżką naszego obecnego CWD jest `/aktualny/katalog`. W takiej sytuacji użycie ścieżki względnej `wzgledna/plik` zostanie przez system operacyjny automatycznie zrozumiane jako ścieżka `/aktualny/katalog/wzgledna/plik`. Zauważmy, że został dodany ukośnik przy łączeniu ścieżek. Bez niego ścieżka wskazywała by na zupełnie inne miejsce w systemie plików. W ogólności pliki wskazywane przez ścieżki nie muszą istnieć. Nie muszą nawet istnieć żadne katalogi pośrednie (z wyjątkiem korzenia systemu plików).

Ponadto, zauważmy też, że poniższe trzy ścieżki bezwzględne:

```
/sciezka/bezwzgledna/pliku
/sciezka/../sciezka/bezwzgledna/pliku
///sciezka/./././sciezka/bezwzgledna/pliku
```

wskazują na to samo miejsce systemu plików (ich rozwiązanie doprawdzi do uzyskania tego samego numeru i-węzła). Widzimy więc, że porównanie ścieżek bezwzględnych (jako łańcuchów tekstowych) nie jest dobrym pomysłem, bo różne ścieżki bezwzględne mogą wskazywać na ten sam plik.

Wśród wszystkich ścieżek bezwzględnych istnieje jednak jedna ścieżka szczególna zwana ścieżką kanoniczną (*ang.* canonical)[2]. Ścieżka kanoniczna jest najkrótszą (pod względem liczby ukośników, nie liczby znaków!) ścieżką do pliku. Powstaje m.in. poprzez usunięcie (rozwiązanie) ze ścieżki bezwzględnej wszystkich dowiązań symbolicznych (o których później) i odwołań typu `"."` i `".."`. Jeśli dwie ścieżki kanoniczne (porównywane jako tekst) są sobie *równe*, to na pewno wskazują na *ten sam* plik.

Aby utworzyć ścieżkę kanoniczną można wykorzystać polecenie `realpath`, które powinno być dostępne w laboratorium (ewentualnie można je zainstalować). Polecenie przekształca podaną ścieżkę na kanoniczną. Wadą tego polecenia jest fakt, że każdy komponent podanej ścieżki musi istnieć. Alternatywą do `realpath` jest komenda `readlink` *użyta z jedną z opcji* `-e`, `-f` *lub* `-m`. Opcje te sterują, które z komponentów ścieżki mogą nie istnieć (dla `-e` wszystkie komponenty muszą istnieć, co daje taki sam efekt jak komenda `realpath`). Użycie `readlink` bez opcji spowoduje zupełnie inny efekt (patrz dalej).

Polecenia `basename` i `dirname` służą do uzyskania, odpowiednio, ostatniego elementu ścieżki (po ostatnim ukośniku) i wszystkich elementów przed ostatnim (wszystko przed ostatnim ukośnikiem). Komendy te są prostymi programami modyfikującymi podany łańcuch tekstowy na podstawie znalezionych ukośników, w szczególności podana ścieżka nie musi naprawdę istnieć.

3 Dowiązania symboliczne

Dowiązania/linki symboliczne (zwane też dowiązaniami/linkami miękkimi lub "symlinkami") są, po plikach zwykłych i katalogach, trzecim rodzajem plików. Zawartością dowiązań symbolicznym jest po prostu ścieżka. Próba otwarcia dow.

[2]Manual często nazywa ją po prostu ścieżką bezwzględną.

symb. spowoduje próbę otwarcia pliku danego ścieżką zapisaną w tym dowiązaniu.

Ścieżka zapisana w dow. symb. może być bezwzględna lub względna. Dowiązania ze ścieżką *bezwzględną* działają tak jak w przypadku każdej innej ścieżki bezwzględnej – ścieżka jest liczona od korzenia systemu plików. W przypadku ścieżki *względnej* następuje pewna różnica. Ścieżka ta jest bowiem liczona *NIE* od aktualnego katalogu CWD, ale od *aktualnego* położenia samego dowiązania. Oznacza to, że jeśli treścią linku jest `zawartosc/linku`, zaś ścieżką samego linku jest `sciezka/do/linku`, to próba otwacia linku skutkuje próbą otwarcia pliku danego ścieżką:

```
sciezka/do/linku/zawartosc/linku
```

W związku z powyższym powstaje pytanie: czy lepiej w dow. symb. jest używać ścieżek względnych czy bezwzględnych? Prostej odpowiedzi nie ma, gdyż oba przypadki mają swoje dobre i złe strony. Ścieżka bezwzględna ma tę zaletę, że pozostanie poprawna, nawet jeśli przeniesiemy dowiązanie w inne miejsce (oczywiście w ramach tego samego systemu plików). Zaletą ścieżki względnej jest to, że zachowuje ona położenie względem celu dowiązania. Innymi słowy, jeśli ścieżka od dowiązania do celu jest zawarta w pewnym poddrzewie katalogowym, to przeniesienie całego poddrzewa (wraz z *celem dowiązania*) w inne miejsce nie sprawi, że dowiązanie stanie się niepoprawne.

To jednak nie koniec właściwości dowiązań symbolicznych. Zacznijmy od tego, że dowiązania istnieją, by zastąpić pewne operacje na pliku wskazywanym przez dane dowiązanie. Tak więc próba zapisu lub odczytu dowiązania spowoduje tak naprawdę zapis/odczyt ze wskazywanego pliku (o ile taki istnieje). Nie dotyczy to jednak wszystkich operacji – próba usunięcia lub zmiany nazwy dowiązania symbolicznego spowoduje usunięcie lub zmianę nazwy jedynie samego dowiązania, a nie wskazywanego przez niego pliku. Dowiązania symboliczne pozwalają na szybki dostęp do ważnych plików (i katalogów!) a przy tym mają zwykle mały rozmiar (ogólnie równy liczbie znaków w przechowywanej ścieżce).

Ścieżki w dowiązaniach symbolicznych przechowywane są dosłownie – dokładnie tak jak podano przy tworzeniu (lub modyfikacji) dowiązania. Dowiązania same z siebie nigdy nie zmieniają zawartej w niej ścieżki na bezwzględną, ani tym bardziej kanoniczną.

Warto zaznaczyć, że dowiązania symboliczne są fizycznymi plikami – mają więc własną nazwę i numer i-węzła. Co więcej, nie ma przeszkód, by utworzyć dowiązanie symboliczne, które wskazuje na inne dowiązanie symboliczne. W ten sposób powstaje łańcuch dowiązań. Nie jest to problem, pod warunkiem, że w łańcuchu nie występuje cykl. Ogólnie dowiązania symboliczne można tworzyć do dowolnych plików, także do katalogów. Co więcej, wskazywany plik nie musi istnieć (taki link na nieistniejący plik nazywany jest linkiem niepoprawnym, zepsutym lub wiszącym). Zepsuty link można zidentyfikować przy wykorzystaniu polecenia `ls`, o ile polecenie to odpowiednio koloruje swój wynik – zepsute linki będą wyświetlone na czerwono lub na czerwonym tle.

Możliwe jest, że poprawny link przestanie działać, jeśli jego cel zostanie usunięty, przeniosiony lub zmieni nazwę. Możliwa jest też sytuacja odwrotna –

niedziałający link może stać się poprawny, gdy wcześniej nieistniejący cel tego dowiązania zostanie stworzony. Można więc stworzyć dowiązanie symboliczne do pliku na nośniku USB – link będzie poprawny, gdy nośnik będzie zamontowany (w odpowiednim miejscu). Co więcej, nie ma przeszkód by dowiązanie istniejące na jednym urządzeniu (partycja główna, dysk zewnętrzny, nośnik USB) wskazywało na plik na całkowicie innym urządzeniu.

Przejdziemy teraz do zarządzania symlinkami i komend z tym związanych. Do odczytywania zawartości symlinku służy komenda `readlink` (bez dodatkowych opcji!). Komenda ta zwraca ścieżkę zawartą w symlinku bez *żadnych* zmian. Użycie `readlink` na pliku, który nie jest symlinkiem spowoduje otrzymanie pustego łańucha znaków. Uwaga! Nie należy tego użycia komendy `readlink` mylić z jej użyciem z opcjami `-f`, `-e` lub `-m`!

Usunięcie symlinku odbywa się standardowo z użyciem komendy `rm`. Tworzenie symlinków jest bardziej skomplikowane i często sprawia problemy podczas laboratorium. Do tworzenia symlinków służy komenda `ln` *z opcją* `-s`!. Bez opcji `-s` komenda utworzy dowiązanie twarde, zamiast symbolicznego! Po drugie, nie należy mylić komendy `ln` (od "link") z komendą `ls` (od "list"). Jednakże głównym problemem z użyciem komendy `ln` jest jej składnia i kłopoty z podaniem odpowiednich ścieżek. Komenda ta w najczęściej używanej postaci wymaga podania po opcji `-s` dwóch argumentów:

1. Pierwszym argumentem jest zawartość dowiązania (cel, ścieżka do celu). Praktycznie każdy łańcuch tesktowy może być celem dowiązania, więc system nie ostrzeże nas jeśli podamy zły link. Jeśli cel podajemy jako ścieżkę bezwzględną, to zwykle problemu nie ma. Problemem jest ścieżka względna. Ścieżka ta musi być podana względem *miejsca, w którym utworzone zostanie dowiązanie*!

2. Drugim argumentem jest ścieżka, w której należy utworzyć dowiązanie. Oczywiście ścieżka ta musi być poprawna. Tzn. jeśli argumentem tym jest `miejsce/tworzenia/linku`, to katalogi `miejsce` i `tworzenia` muszą istnieć w odpowiednich miejscach, zaś dowiązanie będzie miało nazwę `linku`. Utworzenie nie uda się, jeśli plik `linku` w `miejsce/tworzenia` już istnieje.[3] Oczywiście argument ten może być podany jako ścieżka względna lub bezwzgledna. Jeśli jednak podajemy ścieżkę względną, to pamiętajmy, że podajemy ją względem naszego obecnego położenia (CWD).

Niezrozumienie powyższych zasad zwykle prowadzi do tego, że link "znika" (został utworzony w złym katalogu) lub jest niepoprawny (wskazuje na zły lub nieistniejący plik). Częściowym ułatwieniem jest tworzenie symlinków wtedy, gdy nasze aktualne położenie jest takie same jak katalog, w którym będzie tworzony link (trzeba jednak umieć do takiej sytuacji doprowadzić i potem z niej wyjść).

[3] Chyba, że użyjemy opcji `-f`, która wymusi nadpisanie istniejącego linku.

4 Dowiązania twarde

Dowiązania twarde (zwane też po prostu dowiązaniami lub linkami, zwłaszcza przez manuale) nie są osobnym typem plików. Tak naprawdę dowiązania twarde zostały już częściowo opisane, bo są one niczym innym jak wpisami katalogowymi omówionymi we wcześniejszej części instrukcji. Spójrzmy na poniższą tabelkę z wpisami katalogowymi:

nazwa pliku	numer i-węzła
`FILE1`	1033
`FILE2`	2480
`FILE3`	1033

Zauważmy, że wpisy `FILE1` oraz `FILE3` odwołują się do tego samego i-węzła (czyli do tego samego pliku fizycznego). Oznacza to, że dla i-węzła 1033 istnieje więcej niż jedna "ścieżka dostępu" (ściślej, istnieje więcej niż jedna ścieżka kanoniczna do tego pliku. Ścieżki kanoniczne nie zawierają katalogów, które są dow. symbolicznymi). Mówimy, że wpisy `FILE1` oraz `FILE3` są *dowiązaniami twardymi* na ten sam plik. Można też powiedzieć, że `FILE1` jest dow. twardym na `FILE3` *i w drugą stronę*.

Dany plik nie zostanie fizycznie usunięty przez system operacyjny, dopóki istnieje co najmniej jedno dowiązanie twarde odnoszące się do tego pliku (tj. do i-węzła tego pliku). Jeśli liczba dow. twardych osiągnie 0, to plik będzie mógł zostać usunięty. Zauważmy, że nie ma znaczenia, które dowiązanie zostało utworzone jako pierwsze – `FILE1` czy `FILE3` – obie "ścieżki dostępu" są dla systemu operacyjnego równoważne. Oczywiście `FILE1` i `FILE3` nie muszą być w jednym katalogu, by być dowiązaniami twardymi na ten sam plik – wystarczy, że odwołują się do tego samego i-węzła, niezależnie od tego w którym katalogu się znajdują. Jeśli są jednak w jednym katalogu, to muszą mieć różne nazwy.

Informacja o liczbie dowiązań twardych do danego pliku jest zawarta w jego i-węźle. Dla dow. symbolicznych nie ma takiego udogodnienia (nie można w łatwy sposób stwierdzić ile jest dow. symb. do danego pliku, zwłaszcza, że mogą one pochodzić z innych urządzeń).

Dowiązania twarde rządzą się nieco innymi prawami niż dowiązania symboliczne. Po pierwsze, jak sprawdzić czy dane dwa pliki są dowiązaniami twardymi na siebie? Porównanie numerów i-węzłów (gdy już uda się to zrobić) nie jest, wbrew pozorom, wystarczające. Numery i-węzłów są bowiem unikalne *tylko w obrębie urządzenia (partycji)*. Należałoby więc sprawdzić zarówno numer urządzenia jak i numer i-węzła. Na szczęście komenda `test` posiada specjalną opcję, która pozwala sprawdzić czy podane dwa pliki są na siebie dowiązaniami twardymi. Opcją tą jest `-ef`. Należy jednak zaznaczyć, że jest to *jedna nietypowa* opcja krótka, a nie połączenie opcji `-e` i `-f`!

Jak wiemy, wpisy katalogwe oparte są jedynie o i-węzły, a nie numery urządzeń. Dlatego też nie można tworzyć dowiązań twardych pomiędzy różnymi urządzeniami. W teorii nie ma przeszkód, żeby tworzyć dowiązania twarde do katalogów (niektóre systemy uniksowe dopuszczają takie rozwiązanie), ale w sys-

temie linuks możliwość ta jest dla użytkownika zablokowana. Zauważmy jednak, że tylko *użytkownik* nie może tworzyć dow. twardych do katalogów – system ma taką możliwość i dokonuje tego w jednym konkretnym przypadku – podczas tworzenia nowego katalogu.

Załóżmy, że istnieje katalog `PARENT`. Teraz tworzymy w nim katalog `CHILD` z użyciem standardowej komendy `mkdir`. Wydawałoby się, że nowo utworzony katalog `CHILD` będzie pusty. Tak jednak nie jest! W katalogu tym zostaną automatycznie utworzone dwa dowiązania twarde: `"."` oraz `".."`. Pierwszy z tych wpisów wskazuje na ten sam i-węzeł co właśnie utworzony katalog `CHILD`, drugi zaś wskazuje na katalog nadrzędny dla `CHILD`, czyli `PARENT`. Teraz widać, dlaczego użycie wielu "kropek" w ścieżce ma sens:

```
./sciezka/./do/./pliku
```

Każda "kropka" będzie bowiem sprawdzana we wpisach innego katalogu (pierwsza w katalogu aktualnym, druga w `sciezka` a trzecia w `sciezka/do`. Podobnie jest przy konstruowaniu ścieżek typu `"../.."` i podobnych.

Wpisy `"."` oraz `".."` nie są więc "magicznymi" ciągami znaków interpretowanymi przez komendy, lecz nieco specjalnymi wpisami katalogowymi. Ich szczególna właściwość jest jednak bardzo użyteczna przy określaniu liczby *bezpośrednich* podkatalogów danego katalogu. Bowiem jeśli katalog `DIR` w typowym systemie linuksowym ma N dowiązań twardych, to oznacza, że ma dokładnie $N - 2$ bezpośrednich podkatalogów. Dzieje się tak, ponieważ przy tworzeniu `DIR` zostały automatycznie utworzone dwa dowiązania: jedno w rodzicu `DIR` (po to, żeby można było się dostać do `DIR` z katalogu rodzica) a drugie w samym katalogu `DIR` (jako wpis `"."`). Pozostałe N dowiązań do `DIR` to wpisy `".."` w każdym z N bezpośrednich podkatalogów `DIR`.

Jedyną pozostałą kwestią jest sposób tworzenia dowiązań twardych. Jest on podobny, ale nie identyczny do tworzenia dowiązań symbolicznych. Po pierwsze, przy tworzeniu linków twardych nie podajemy opcji `-s`. Po drugie, przy tworzeniu dowiązań twardych co prawda podajemy ścieżke, do celu dowiazania, ale system nie zapisuje tej ścieżki w dowiązaniu, a jedynie używa jej do znalezienia odpowiedniego numeru i-węzła, do którego dowiązanie będzie się odwoływać. Konsekwencje tego są dwie: 1) cel dowiązania musi istnieć oraz 2) jeśli cel dowiązania jest podany ścieżką względną, to jest ona liczona standardowo względem aktualnego CWD (czyli inaczej niż dla dowiązań symbolicznych).