POLITECHNIKA BIAŁOSTOCKA

Wydział Informatyki

Paweł Żukowski

Multifile Renaming Utility
Program narzędziowy do grupowej zmiany nazw plików na podstawie metadanych w nich zawartch

PRACA INŻYNIERSKA

Promotor
dr inż. Marcin Skoczylas

Białystok 2013

Streszczenie

Zażółć gęślą jaźń

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius. Claritas est etiam processus dynamicus, qui sequitur mutationem consuetudium lectorum. Mirum est notare quam littera gothica, quam nunc putamus parum claram, anteposuerit litterarum formas humanitatis per seacula quarta decima et quinta decima. Eodem modo typi, qui nunc nobis videntur parum clari, fiant sollemnes in futurum.

Słowa kluczowe: *Zażółć gęślą jaźń*

Spis treści

Rozdział 1.

Wstęp

Z każdym rokiem ludzie oraz same komputery generują coraz większą ilość informacji. Mimo że duża część z nich jest przechowywana w dobrze strukturyzowanych bazach danych, to ciągle, większość ludzi ma bezpośredni dostęp jedynie to tego co przechowuje w systemie plików własnego komputera. <Rodzaje danych, ich zastosowanie>

Od dziesięcioleci dysk twardy pozostaje głównym kontenerem dla danych użytkowników komputerów na całym świecie. <Dane przechowywane w systemach plików>

<Systemy plików, ich cechy wspólne, ograniczenia, metadane>

<problem z identyfikatorami>

1.1 Cel i zakres pracy

Niniejsza praca ma na celu stworzenie programu narzędziowego pozwalającego na automatyczne generowanie identyfikatorów (nazw) plików na podstawie metadanych w nich zawartych. <także danych generowanych przez sam program - CRC np>

Zakres pracy obejmuje:

- Przegląd istniejących rozwiązań - programów i technik wspomagających masową zmianę identyfikatorów plików.
- Porównanie funkcjonalności istniejących narzędzi i ich ograniczeń.
- Projekt oraz implementacja wieloplatformowej architektury modułów.
- Stworzenie parsera wyrażeń zawierających metatagi.
- Projekt graficznego interfejsu użytkownika opartego na bibliotece wxWidgets.
- Implementacja backendu do systemu plików opartego na bibliotece boost::filesystem.
- Implementacja przykładowych modułów metatagów.
- Testy aplikacji.

1.2 Założenia

Gotowa aplikacja powinna być niezależna od systemu operacyjnego w stopniu w jakim pozwalają na to zależności użytych bibliotek. Dzięki modułowej budowie powinna także udostępniać interfejs pozwalający na jej łatwą rozbudowę.

1.3 Plan pracy

<Co w jakim rozdziale się znajduje>

Rozdział 2.

Teoria

W niniejszym rozdziale postaram się przybliżyć obraz problemu identyfikatorów (zwanych również nazwami) plików opisując środowisko i w którym występuje.

2.1 Dane w systemie komputerowym

Jednym z podstawowych elementów systemu komputerowego jest jego pamięć. Od początku istnienia komputerów istniała potrzeba składowania danych wymaganych przy praktycznie każdych operacjach wykonywanych przez jednostkę centralną komputera. Jako że pierwsze systemy komputerowe były wykorzystywane do obliczeń typowo matematycznych, algorytmy na nich uruchamiane nie wymagały wielkich kontenerów na dane. W tych czasach wbudowane rejestry oraz ulotna pamięć RAM zaspokajały potrzemy rynku. Jednak wraz z rozwojem sprzętu i algorytmów na nim uruchamianych pojawiła się potrzeba przechowywania coraz to większej ilości danych jak i (dzięki zastosowaniu architektury von Neumanna) samych programów przez coraz dłuższy czas. Pojawiła się idea nieulotnej oraz pojemnej pamięci - dysku twardego. <!sprawdzić!>

Pojemności pierwszych dysków twardych stanowiły promil dzisiejszych jednostek toteż nie wymagały stosowania systemów plików - były po prostu nieulotnym rozszerzeniem pamięci operacyjnej RAM.

Wraz ze zwiększeniem ich pojemności oraz generalizacją oprogramowania, pojawiła się

potrzeba standaryzowania, kategoryzacji przechowywanych na dyskach danych - tak powstały systemy plików.

??

2.2 Systemy plików i identyfikacja danych

System plików stanowi warstwę abstrakcji między programami, a danymi zapisanymi na nośniku — dysku twardym, karcie pamięci czy też płycie CD. System plików jest metodą zapisu danych — schematem dzięki któremu, programy nie muszą operować na surowych blokach danych lecz mogą korzystać z bardziej wysokopoziomowych deskryptorów plików - węzłów bądź ścieżek dostępu.
Zwykle systemem plików zarządza system operacyjny — to on udostępnia API, a także blokuje lub pozwala na dostęp do danych ze względu na uprawnienia użytkownika, programu lub samego zasobu.

Istnieje wiele typów oraz implementacji systemów plików, które można podzielić na dwie kategorie:

- tradycyjne - znajdujące zastosowanie przy przechowywaniu dowolnych (ogólnych) danych w postaci plików
- specjalne - dostosowane do specyficznych rozwiązań (jak na przykład bazy danych)

Oddzielną kategorię mogą stanowić zdobywające coraz większą popularność wirtualne systemy plików - różnią się one od tradycyjnych i specjalistycznych tym że nie przechowują danych fizycznie na nośniku, a są raczej aplikacjami udostępniającymi (generującymi) struktury danych na żądanie użytkownika/programu. Przykładem takich systemów mogą być: `procfs` - udostępniający dostęp do procesów systemowych i ich atrybutów w systemach rodziny GNU/Linux oraz *BSD, czy też `NFS` (Network File System) — pozwalający na dostęp do systemów plików znajdujących się na innych komputerach w sieci.

2.2.1 Katalogi i ścieżki do plików

Niniejsza praca skupia się na problemie opisywania danych w tradycyjnych systemach plików za pomocą tak zwanych ścieżek do plików.

Tradycyjne systemy plików pozwalają na przechowywanie danych w drzewiastej strukturze danych zwanej drzewem katalogów. W większości implementacji każdy węzeł takiego drzewa może być katalogiem, plikiem lub dowiązaniem do innego węzła. Dodatkowo węzły katalogów jako jedyne mogę posiadać węzły podległe — podkatalogi.
Każdy węzeł prócz węzła-korzenia jest identyfikowany przez unikalny względem węzła-rodzica identyfikator zwany nazwą pliku/katalogu.
Warto zauważyć iż struktura drzewa katalogów nie wymusza sposobu rozkładu danych w systemie plików — tak długo jak identyfikatory pozostają unikalne, pliki przez nie opisywane mogą znajdować się w tym samym katalogu[1].

2.2.2 Różnice w identyfikacji plików wśród różnych systemów operacyjnych

Format ścieżki do pliku narzucany jest niezależnie od zastosowanego systemu plików przez system operacyjny.

Systemy kompatybilne ze standardem `POSIX`, wywodzące się z Unixów takie jak Apple MacOS czy rodzina BSD, a także rodzina GNU/Linux używają drzew katalogów z pojedynczym, nienazwanym korzeniem oznaczanym symbolem prawego ukośnika (slash) — '`/`'.
Symbol prawego ukośnika jest również używany jako separator elementów (poziomów) ścieżki i nie może stanowić elementu identyfikatora węzła w wymienionych środowiskach.
Przykład ścieżki zgodnej ze standardem `POSIX`:

```
/home/idlecode/projects/mru/doc/main.tex
```

[1] W praktyce ilość plików które mogą należeć do jednego węzła zależy od rozmiaru licznika użytego w implementacji.

Systemy operacyjne z rodziny Windows korporacji Microsoft[2] wykorzystują natomiast lewy ukośnik (backslash) — '\' — jako separator komponentów ścieżki oraz uniemożliwiają stosowanie większej ilości symboli w nazwach.
System plików systemu Windows może posiadać kilka korzeni (po jednym dla każdego wolumenu/dysku) oznaczanych pojedynczymi, zwykle dużymi literami alfabetu łacińskiego. Litera dysku wraz z symbolem dwukropka poprzedza właściwą ścieżkę do pliku.
Przykład ścieżki używanej w systemach operacyjnych Windows korporacji Microsoft:

```
C:\Users\idlecode\My Documents\Projects\MRU\doc\main.tex
```

Dodatkowo w przypadku obu[3] wyżej wymienionych schematów, nazwy elementów nie mogą zawierać znaku zerowego (NUL — o kodzie heksadecymalnym `0x00`), który może zostać zinterpretowany jako koniec łańcucha znaków.

Większość implementacji pozwala zawrzeć pełen zakres symboli (znaków) w ścieżce za pomocą kodowań z rodziny `UTF` przy czym pojedynczy identyfikator może mieć maksymalną długość 255 bajtów. Ograniczenie długości całkowitej ścieżki, jeśli istnieje jest wynosi ustawione na poziomie. Warto tu zauważyć iż systemy z rodziny Windows zachowują wielkość liter w identyfikatorach lecz przy interpretacji ścieżek — rozwijaniu ich do odpowiadających węzłów — nie gra ona znaczenia co nie ma miejsca w systemach klasy `POSIX`. Istnieje również możliwość stosowania ukośników prawych do rozdzielania komponentów ścieżki tak jak to ma miejsce w systemach `POSIX`-owych.

Istnieje jeszcze kilka schematów zapisu ścieżek, które nie zostały przybliżone ze względu na zakres niniejszej pracy.

[2]Istnieje wiele więcej systemów operacyjnych używających podobnego schematu
[3]System MacOS nie posiada tego ograniczenia

2.3 Metadane

Metadane z definicji są danymi opisującymi inne dane. Metadane stosowane są w przypadkach gdy nie istnieje fizyczna możliwość dołączenia lub dodatkowe informacje są zbyt luźno powiązane z opisywanymi danymi. Przykładem metadanych mogą być karty biblioteczne — informują one o statusie i historii np. książki nie będąc jej integralną częścią.

W systemach plików, metadane dostarczają informacji o plikach zapisanych w drzewie katalogów. Przykładem komputerowych metadanych może być wspominana wcześniej nazwa czy ścieżka do pliku, która nie jest jego integralną częścią — może zostać zmieniona bez naruszania struktury przechowywanego dokumentu. Dodatkowo systemy plików często dostarczają ogólnych atrybutów — meta-informacji możliwych do uzyskania z dowolnego typu pliku takich jak jego rozmiar, czas utworzenia/ostatniej modyfikacji czy prawa dostępu. Ciekawym przykładem metadanych są rozszerzenia nazw plików — sufiksy rozpoczynające się od ostatniego znaku kropki w nazwie. Rozszerzenia odgrywały ważną rolę w systemach operacyjnych korporacji Microsoft gdzie stanowiły integralną część nazwy i pozwalały systemowi skojarzyć typ pliku z programem go obsługującym. W systemach `POSIX`-owych informacja o typie pliku jest zwykle przekazywana wraz z kontekstem uruchomienia aplikacji operującej na pliku (za pomocą linii komend) lub pomijana całkowicie — wiele aplikacji takich systemów operuje na plikach jako ciągu bajtów i nie rozróżnia ich typów.

Niektóre formaty plików (szczególnie multimedialnych) pozwalają na integracje metadanych z samym plikiem. Jako że pliki (szczególnie binarne) mogą stosować dowolną strukturę zapisu, nie istnieje ogólny algorytm wyciągnięcia zawartych w ten sposób informacji. Istnieje natomiast wiele bibliotek umożliwiających odczyt informacji z określonego typu pliku.

Rozdział 3.

Implementacja

Rozdział ten zawiera opis środowiska które zostało użyte do stworzenia implementacji, a także architektury samej aplikacji.

3.1 Środowisko pracy

3.1.1 Język C++

Do implementacji aplikacji `MRU` został użyty język C++ w standardzie z roku 2003 (ISO/IEC 14882:2003). Język C++ jest dojrzałym, wieloplatformowym językiem programowania średniego poziomu, używanym od wielu lat przez programistów na całym świecie do tworzenia aplikacji, sterowników czy nawet systemów operacyjnych. Dzięki kompatybilności z C[1] pozwala na wykorzystanie wielu istniejących bibliotek napisanych zarówno w C jak i C++.

3.1.2 LLVM Clang

`LLVM` — Low Level Virtual Machine — jest modułową architekturą do budowy kompilatorów. Pozwala ona na oddzielenie parserów różnych języków programowania (frontendów)

[1] C++ nie jest całkowicie kompatybilny z C, jednak różnice w obu tych językach są na tyle małe że zwykle nie wpływają negatywnie na kompatybilność (szczególnie na poziomie ABI).

od modułu optymalizacji (wspólnych dla wszystkich języków kompilowalnych) i emiterów kodu bajtowego (backendów) dla różnych platform.

Clang jest frontendem języków C i C++ dla architektury `LLVM`. Projekt jest otwarty (wydawany na licencji BSD) i zdobywa coraz większą popularność[2] dorównując i przewyższając w niektórych testach GCC.

3.1.3 System operacyjny FreeBSD

System FreeBSD jest systemem operacyjnym z rodziny BSD wywodzącej się z kolei z rodziny `UNIX`-ów. Podobnie do dystrybucji GNU/Linux, sam w sobie wraz w wieloma, otwartymi bibliotekami tworzonymi przez społeczność stanowi środowisko przyjazne programistom.

3.1.4 Mercurial

Do zarządzania plikami źródłowymi oraz kopią zapasową został wykorzystany rozproszony system kontroli wersji Mercurial. Wraz z serwisem bitbucket.org pozwala on na synchronizację kodów źródłowych między wieloma maszynami, ułatwiając tym samym pracę nad pojedynczym projektem wielu programistów.

W odróżnieniu od scentralizowanych systemów kontroli wersji takich jak SVN, Mercurial, podobnie jak Git nie wymaga pojedynczego serwera, ani serwera w ogóle. Pełne repozytorium jest trzymane na każdej maszynie z której korzysta programista, a praca różnych programistów (zmiany w kodzie) może być synchronizowana między nimi samymi.

3.1.5 CMake

Aby projekt był jak najbardziej przenośny i niezależny od platformy, ważne jest aby jego proces budowania był również taki był. W celu zapewnienia łatwego wsparcia dla

[2] Od listopada 2012 Clang stał się domyślnym kompilatorem dla systemu FreeBSD

budowania projektu na wielu platformach i wielu łańcuchach narzędziowych, MRU stosuje CMake — narzędzie do zarządzania procesem kompilacji i zależnościami.

CMake pozwala programiście określić z jakich elementów składa się program i jakich zewnętrznych zasobów (bibliotek) wymaga. Narzędzie następnie interpretuje skryptowy plik konfiguracyjny i tworzy natywne dla danej platformy pliki projektowe zawierające odpowiednią do zbudowania projektu konfigurację.

Rozdział 4.

Definicje

ABI - Application Binary Interface, binarny interfejs aplikacji — pozwala dwóm programom komputerowym na wymianę informacji na najniższym, niezależnym od języka programowania poziomie abstrakcji.

I - macierz opisująca obraz wejściowy

w - szerokość obrazu wejściowego, liczba kolumn macierzy I

Lista publikacji

C. Zet, H. Kiesewetter, M. Skoczylas, L. Westerberg, R. Spohr. A system for irradiating polymer films with a preset number of ions. GSI Scientific Report, Gesellschaft für Schwerionenforschung mbH (GSI), 154, 2003

C. Zet, C. Foslau, M. Skoczylas, L. Westerberg, R. Spohr. System for irradiating polymer films with a preset number of ions. SIELMEN, 4th International Conference on Electromechanical and Power Systems vol. 2, 167-170, 2003

M. Skoczylas, K. Andrzejewski. Krzyżtopór. Telewizja Polska, wyd. Rzeczpospolita, Akademia Filmu i Telewizji Warszawa, 2005