Akademia Górniczo-Hutnicza
Wydział Informatyki, Elektroniki i Telekomunikacji

KATEDRA INFORMATYKI

Program do tworzenia kopii zapasowych na wzór Dropboxa

Kierunek, rok studiów:
Informatyka, rok II
Przedmiot:
Systemy Operacyjne
Prowadzący:
dr inż. Sławomir Bieniasz
Autor:
Mateusz Jaje

SPIS TREŚCI

1. Tematyka i opis projektu

1.1 Polecenie

Program kliencki powinien monitorować zadany folder i synchronizować zawartość z serwerem, a także pobierać zmiany wysłane przez inne programy klienckie na serwer. Serwer powinien tworzyć wersje zmienionych plików tak, aby można było cofnąć zmiany w pliku do wybranej wersji z zadanego dnia.

1.2 Opis zagadnień

W ramach projektu stworzono aplikację, która umożliwia synchronizację plików z aplikacji klienckich z serwerem oraz zachowywanie historii zmian,tak, że możliwe może być cofanie zmian do zadanej wersji.

1.3 Implementacja

System składa się z dwóch odrębnych modułów: klienta i serwera.

1.3.1 Serwer

Serwer po uruchomieniu zaczyna nasłuchiwać na połączenia od klientów. Po zaakceptowaniu połączenia sprawdza czy jest w stanie je obsłużyć (jeśli może jeszcze przyłączyć więcej klientów), i jeśli może, to tworzy nowy wątek dla klienta oraz wysyła mu informację o powodzeniu logowania lub niepowodzeniu. W komunikacji z klientem jest stroną bierną, tzn, że sam nie wysyła klientowi żadnych informacji pierwszy, realizuje tylko rozkazy klienta.

1.3.1 Klient

Klient po uruchomieniu próbuje się połączyć z serwerem i jeśli połączenie zostanie nawiązane, oraz serwer odeśle pozytywną odpowiedź na logowanie, klient rozpoczyna w pętli nieskończonej na zmianę synchronizować swoje pliki lokalne z serwerem a potem prosi serwer o synchronizację plików z serwera. Klient w komunikacji z serwerem jest stroną aktywną, tzn, że to klient wysyła rozkazy do serwera.

2. Technologie i biblioteki

2.1 Komunikacja

Komunikacja między serwerem a klientami odbywa się za pośrednictwem socketów z dziedziny internetu. Jest ona prowadzona w trybie połączeniowo-strumieniowym, wykorzystując protokół TCP/IP. Serwer jest aplikacją wielowątkową dlatego też dla każdego nowego połączenia tworzony jest osobny socket.

2.2 Wielowątkowość

Serwer obsługuje wiele połączeń jednocześnie, więc aby przebiegało to sprawnie, zastosowałem wielowątkowość, z pomocą biblioteki pthread. Każde połączenie jest obsługiwane w osobnym wykonującym się rozłącznie wątku, dzięki czemu synchronizacja plików z różnymi klientami może przebiegać jednocześnie.

2.3 Synchronizacja

Aby zapewnić spójność danych na serwerze, zastosowałem blokady systemowe z użyciem funkcji flock, którą to blokadę na wyłączność musi pobrać wątek który chce używać danego pliku.

3. Szczegółowy opis procedur

3.1 Serwer

3.1.1 prepare_data

Zajmuje się przygotowaniem danych do działania serwera, m.in. inicjalizacją mutexów, tablic, stworzenie klucza danych własnych.

3.1.2 parse_arguments

Parsuje argumenty wywołania sprawdzając ich poprawność pod względem formatu (używając wyrażeń regularnych) oraz przygotowuje je do używania w programie. W przypadku błędnego formatu zwraca błąd.

3.1.3 set_work_directory

Ustawia katalog roboczy procesu wykonującego program serwera, jest to dla wygody możliwości używania ścieżek względnych plików.

3.1.4 create_and_connect_socket

Zajmuje się stworzeniem oraz bindowaniem gniazda przeznaczonego do nasłuchiwania na nowe połączenia. Tworzone jest gniazdo strumieniowe protokołu INET, nieblokujące, zbindowane na port podany w argumencie wywołania.

3.1.5 create_connection

Zajmuje się logowaniem klienta, który połączył się z serwerem. Jeśli serwer może przyjąć jeszcze nowych klientów, procedura tworzy nowy wątek dla nowego klienta, jeśli nie może już przyjąć więcej połączeń lub nie udało się stworzyć wątku dla połączenia, odsyła informację o niemożliwości przyjęcia logowania.

3.1.6 connection_thread

Wątek obsługi połączenia z klientem. Po uruchomieniu wysyła klientowi informację o powodzeniu logowania oraz zapisuje socket klienta w danych własnych wątku. Zajmuje się, w pętli nieskończonej, nasłuchiwaniem na rozkazy wydawane przez klienta oraz realizacją ich, aktualnie obsługuje rozkazy CHECK_FILE(porównanie wersji pliku klienta z serwerem), SYNC(wysyłanie listy plików z serwera do klienta), TEST_CONNECTION, LOGOUT.

3.1.7 check_file

Zajmuje się sprawdzeniem wersji pliku wskazanego przez klienta. Odbiera od klienta nazwę pliku oraz datę ostatniej modyfikacji, a następnie porównuje wersję klienta (datę modyfikacji) z wersją na serwerze oraz podejmuje decyzję czy:

1. wysłać klientowi informację, że klient ma starą wersję i że za chwilę nastąpi wysyłanie pliku od serwera do klienta, żeby klient nastawił się na odbieranie pliku
2. wysłać klientowi informację, że klient ma nowszą wersję i że powinien wysłać teraz ten plik do serwera, serwer w tej sytuacji zaczyna czytać z gniazda plik, który klient powinien wysyłać
3. wysłać klientowi informację, że ten plik został usunięty, że klient powinien go usunąć z lokalnego folderu.

Na czas operacji na pliku jest on blokowany funkcją flock.

3.1.8 synchronize_with_client

Ma za zadanie wysyłanie listy plików do klienta sterowane przez klienta, który po odebraniu ścieżki decyduje, czy pominąć i przejść do kolejnego pliku, czy wysłać dany plik klientowi, czy zaznaczyć go, jako usunięty.

3.1.9 get_file

Ma za zadanie odczytać najnowszą przechowywaną na serwerze wersję pliku danego w argumencie, zwraca 0 jeśli takiego pliku nie ma, i czas w formacie time_t jeśli plik istnieje oraz informację, czy dany plik został usunięty, wtedy czas określa datę zarejestrowania usunięcia.

3.1.10 get_file

Wysyła informację do klienta o tym, że powinien rozpocząć wysyłanie pliku , następnie odbiera plik z socketu (używając receive_file) i zapisuje go pod ścieżką daną w argumencie, pod wersją daną w argumencie. Po odebraniu pliku aktualizuje informację w plikach bazodanowych o najnowszej wersji.

3.1.11 give_file

Zajmuje się samym wysłaniem pliku do klienta. Wysyła mu informację, że powinien zacząć odbierać plik, następnie wysyła ten plik (używając send_file).

3.1.12 receive_file

Zajmuje się samym odbieraniem, za pomocą bufora, pliku przez socket i zapisywaniem na dysk.

3.1.13 send_file

Zajmuje się samym wysyłaniem, za pomocą bufora, pliku przez socket.

3.2 Klient

3.2.1 prepare_data

Zajmuje się przygotowaniem danych do działania serwera, m.in. ustawia wartości zmiennej sterującą nieskończoną pętlą.

3.2.2 parse_arguments

Prasuje argumenty wywołania sprawdzając ich poprawność pod względem formatu (używając wyrażeń regularnych) oraz przygotowuje je do używania w programie. W przypadku błędnego formatu zwraca błąd.

3.2.3 set_work_directory

Ustawia katalog roboczy procesu wykonującego program serwera, jest to dla wygody możliwości używania ścieżek względnych plików.

3.2.4 create_and_connect_socket

Zajmuje się stworzeniem oraz nawiązaniem połączenia i zalogowaniem się klienta do serwera, jeśli serwer po nawiązaniu połączenia zwróci stałą CONNECTION_OK, funkcja wraca deskyptor - socket, a jeśli nie udało się nawiązać połączenia, zwracana jest wartość -1.

3.2.6 synchronize

Procedura ma za zadanie przeprowadzić synchronizację plików z serwerem, używa do tego dwóch procedur:

– synchronize_and_send_local_directory

– synchronize_and_download_virtual_directory

3.2.6 synchronize_and_send_local_directory

Procedura używa bibliotecznej procedury ftw() do zbadania całego drzewa katalogów od aktualnego roboczego. Jako argument ftw jest użyta procedura look, która każdy regularny plik synchronizuje z serwerem w sposób:

1. wysłanie informacji o chęci sprawdzenia pliku wraz z długością ścieżki
2. wysłanie ścieżki pliku
3. odebranie informacji, czy wysłać plik (i rozpoczęcie wysyłania), odebrać plik (i rozpoczęcie odbierania), usunąć (i usunięcie pliku).

3.2.7 synchronize_and_download_virtual_directory

Ma za zadanie przeprowadzić synchronizację z wirtualnym folderem, jego zadaniem jest tylko pobieranie plików, które nie istnieją w lokalnym folderze (te istniejące zostały zsynchronizowane procedurze synchronize_and_send_local_directory) wysyła informację do serwera, że chce przeprowadzić tą synchronizację i przestawia się w tryb odbierania informacji, tzn czeka na dodatnią liczbę od serwera (długość ścieżki do pobrania), następnie pobiera ścieżkę od serwera, sprawdza, czy taki plik istnieje na maszynie lokalnej, jeśli istnieje, odsyła serwerowi informację, żeby pominął ten plik, jeśli nie istnieje, to wysyła serwerowi prośbę o wysłanie wersji pliku, następnie sprawdza, czy ta wersja jest nowsza od czasu ostatniej synchronizacji i jeśli jest, to znaczy, że plik został dodany do folderu i następuje pobieranie pliku, a jeśli serwer ma starszą wersję, to to znaczy, że plik został usunięty, klient w takiej sytuacji wysyła informację serwerowi, że plik został usunięty.

4. Uruchomienie

Programy powinny być uruchamiane z następującymi parametrami:
Serwer:

port katalog

gdzie:

port – liczba, port na którym serwer będzie nasłuchiwał na nowe połączenia klienckie

katalog – katalog który ma być korzeniem drzewa katalogów, które ma być synchronizowane

Klient:

port host katalog

gdzie:

port – port na którym serwer nasłuchuje na logowania klientów

host – nazwa hosta serwera np. „localhost”, „student.agh.edu.pl”

katalog – katalog, który ma być korzeniem drzewa katalogów, które będzie synchronizowane

5. Bibliografia

- W. Richard Stevens - Programowanie w środowisku systemu Unix
- Linux Manual
- Prezentacje studentów Informatyki rocznika 2011r. z przedmiotu Systemy Operacyjne