**Programowanie aplikacji klient-serwer**

Projekt 2 Wzajemne udostępnianie plików

**Projekt**

(wersja 1.0)

**AUTOR PROJEKTU:**

Ivan Napolskykh

Spis treści

1 Wstęp 3

1.1 Treść zadania 3

1.2 Dodatkowe założenia 3

1.2.1 Serwer 3

1.2.2 Klient 3

2 Słownik pojęć 3

3 Użyte protokoły 3

3.1 Protokół komunikacji klienta z serwerem 3

3.1.1 Wstęp 3

3.1.1.1 Technologia 3

3.1.1.2 Funkcjonalność: 3

3.1.2 Dokładny opis funkcjonalny poszczególnych metod/komend 4

3.1.3 Spis komunikatów błędów 4

3.2 Protokół przesyłania plików pomiędzy klientami(o ile występuje) 4

3.2.1 Wstęp 4

3.2.2 Żądanie 5

3.2.3 Odpowiedź 5

3.2.4 Spis komunikatów błędów 5

4 Przypadki użycia 5

4.1.1 Klient tworzy nowego użytkownika w bazie serwera 5

4.1.2 Klient loguje się do serwera 5

4.1.3 Klient wylogowuje się z serwera 5

5 Aplikacja klienta 5

6 Aplikacja serwera 5

# Wstęp

## Treść zadania

Aplikacja służąca do wzajemnego udostępniania plików. W projekcie należy opracować zasady takiego udostępniania. Na przykład rozdzielenie zadań między serwer i klienta. Zadania serwera: przyjmowanie nowych klientów oraz logowanie ich w sieci, prowadzenie bazy danych o udostępnianych plikach, odłączanie klientów od sieci. Zadania klienta: dołączanie/odłączanie się od sieci, tworzenie listy udostępnianych plików i przekazywanie jej serwerowi, wyszukiwanie pliku na serwerze i składanie zamówienia na wybrany plik, realizacja złożonych zamówień – przesyłanie pliku

## Dodatkowe założenia

### Serwer

Serwer wykonany został w języku C.

### Klient

Klient będzie aplikacją napisaną w języku C.

# Słownik pojęć

Sesja – abstrakcja połączenia zaufanego klienta z serwerem. Wszystkie komunikaty są wysyłane w ramach sesji. Wszystkie próby komunikacji z serwerem poza aktywną sesją są odrzucane przez serwer.

Test – pytanie z odpowiedziami do niego.

# Użyte protokoły

## Protokół komunikacji klienta z serwerem

### Wstęp

File Transfer Protocol (FTP) klient-serwer aplikacja (RFC 959) zaimplementowana za użyciem C Stream Sockets biblioteki w języku C.

#### Technologia

https://www.rfc-editor.org/rfc/rfc959.txt

#### Funkcjonalność:

* ftp>put <filename> (służy do zaladowania pliku o nazwie filename do serwera)
* ftp>get <filename> (służy do pobrania pliku o nazwie filename z serwera)
* ftp>ls (służy do wyswietłenia listy plików z bieżącego folderu na serwerze)
* ftp>!ls (służy do wyswietłenia listy plików z bieżącego folderu na kliencie)
* ftp>cd <directory> (służy do zmiany bieżącego folderu na serwerze)
* ftp>!cd <directory> (służy do zmiany bieżącego folderu na kliencie)
* ftp>pwd (służy do wyświetłenia ścieżki do bieżącego folderu na serwerze)
* ftp>!pwd (służy do wyświetłenia ścieżki do bieżącego folderu na kliencie)
* ftp>quit (służy do zamknięcia FTP sesji na kliencie i zwrotu do macierzynskiego środowiska)

ZAKONCZENIE DZIALANIA

* Można zamknąć serwer za pomocy zwykłego sygnalu Ctrl + C w oknie terminalu z uruchomionym serwerem.
* Można zamknąć serwer za pomocy wywolania komendy "quit" w oknie terminalu z uruchomionym klientem.

OGRANICZENIA

* Zaimplementowane są wyłącznie wyżej zaznaczone polecenia.
* Maksymalna liczba jednocześnie zalogowanych clientów nie może przekraczać 8.
* W nazwach ścierzek, które są używane jako bierzący katalog roboczy, nie może zaistnieć żadnych spacji (jest to określono tym, że cd <directory> jest zaimplementowana za użyciem chdir(<directory>), która nie pozwala na użycie białych symboli).
* Komenda get przepisywuje istniejący plik, jeżeli taki istnieje w katalogu roboczym klienta. Podobnie, put komenda przepisywuje istniejący plik, jeżeli taki istnieje w katalogu roboczym serwera.

WŁAŚCIWOŚCI

* Przyjmuje każdą z powyżej opisanych komend. Ponad tego, odpowiada komunikatem o blędzie w przypadku niepoprawnego polecenia: "An invalid FTP command".
* Dopasowuje polecenia i wypisywuje odpowiednie dane w terminalu.
* Sprawdzenie konca pliku w przypadku get i put komend jest zaimplementowane za pomocy odwolania w odpowiednim procesie(server/client) do rozmiaru(czy liczbie bloków) pliku.
* Działa w trybie połączeniowym i współbieżnym.

### Spis komunikatów błędów

|  |  |  |
| --- | --- | --- |
| Nr błędu | Komunikat | Znaczenie komunikatu |
| 1 | Usage | Nie poprawnie został wywołany program |
| 2 | Problem in creating the socket | Problem przy utworzeniu soketa |
| 3 | Problem in connecting to the server | Problem przy nawiązaniu połączenia z serwerem |
| 4 | Problem in creating data channel | Problem przy utworzeniu kanału |
| 5 | Port number must be greater than 1024 | Numer portu musi być większy od 1024 |
| 6 | Problem in binding the data socket | Problem przy bindowaniu soketu danych |
| 7 | Problem in accepting the data socket | Problem przy akceptowaniu soketu danych |

* Zaimplementowane są wyłącznie wyżej zaznaczone polecenia.
* Maksymalna liczba jednocześnie zalogowanych clientów nie może przekraczać 8.
* W nazwach ścierzek, które są używane jako bierzący katalog roboczy, nie może zaistnieć żadnych spacji (jest to określono tym, że cd <directory> jest zaimplementowana za użyciem chdir(<directory>), która nie pozwala na użycie białych symboli).
* Komenda get przepisywuje istniejący plik, jeżeli taki istnieje w katalogu roboczym klienta. Podobnie, put komenda przepisywuje istniejący plik, jeżeli taki istnieje w katalogu roboczym serwera.

# Przypadki użycia

### Klient tworzy nowego użytkownika w bazie serwera

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **~$./server 9999**  **Server running...waiting for connections.**  **Received request...**  **Child created for dealing with client requests** | **~$./client 127.0.0.1 9999**  **ftp>** |

### Klient wykonuje polecenia na serwerze

1. ls

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: ls** | **ftp>ls**  **server**  **server.c**  **test2** |

1. !ls

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: !ls** | **ftp>!ls**  **client**  **client.c**  **test** |

1. put

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: put test**  **Filename given is: test**  **File download done.** | **ftp>put test**  **File upload done.** |

1. get

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: get test2**  **Filename given is: test2**  **File upload done.** | **ftp>get test2**  **File download done.** |

1. pwd

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: pwd** | **ftp>pwd**  **/home/eouser/FTP\ Manager\ C/Server** |

1. !pwd

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: !pwd** | **ftp>!pwd**  **/Users/napolsky/FTP\ Manager\ C/Client** |

1. cd

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: cd ../Client**  **Path given is: ../Client** | **ftp>cd ../Client**  **Path given is: ../Client** |

1. !cd

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: !cd ../Server**  **Path given is: ../Server** | **ftp>!cd ../Server**  **Path given is: ../Server** |

### Klient wylogowuje się z serwera

1. quit

|  |  |
| --- | --- |
| **Strona serwera** | **Strona klienta** |
| **String received from client: quit**  **The client has quit** | **ftp>quit** |

# Aplikacja klienta

* Tworzy soket i dowiązuje jego do serweru za pomocy IP adresu serweru i portu(określonych jako argumenty, przy wywolaniu procesu).
* Dopasowuje FTP polecenie i sprawdza jego poprawność. Jeżeli wynika błąd podczas zapytu, wysyła komunikat o tym do klienta.
* Wszystkie polecenia są podawane w terminalu. Są konwerowane do poprawnej postaci i wywolane(klientem/serwerem w zależności od rządania).
* W przypadku uzyskania komendy get, uzyskuje plik przez połączenie danych (port danych jest generowany serwerem) w postaci blokowej.
* W przypadku uzyskania komendy put, sprawdza określony plik i wysyła jego do serwera przez połączenie danych (port danych jest generowany serwerem) w postaci blokowej.
* W przypadku uzyskania komendy ls, uzyskuje spis plikow przez połączenie danych (port danych jest generowany serwerem) w postaci blokowej(lista plików może być dużego rozmiaru).
* W przypadku uzyskania komendy !ls, użyte zostaje polecenie system("ls"), zaczym wypisywany spis plikow z katalogu bierzącego clienta.
* W przypadku uzyskania komendy !cd <directory>, wywolane zostaje polecenie chdir(<directory>), zaczym zmieniany katalog roboczy klienta.
* W przypadku uzyskania komendy pwd, wywolane zostaje polecenie \_getcwd, jeżeli serwerem jest maszyna pod systemem Windows, czy getcwd w przeciwnym przypadku, żeby uzyskać katalog roboczy klienta.
* Wraca się do macierzynskiego środowiska z klienta, jeżeli użyska polecenie quit.

# Aplikacja serwera

* Tworzy soket i dowiązuje jego do portu(określonego jako argument, przy wywolaniu procesu).
* Zaczyna nasłuchiwać na sokecie.
* Przyjmuje połączenie od klienta.
* Uzyskuje FTP polecenie od klienta.
* Dopasowuje FTP polecenie i sprawdza jego poprawność. Jeżeli wynika błąd podczas zapytu, wysyła komunikat o tym do klienta.
* Wszystkie polecenia są podawane w terminalu. Są konwerowane do poprawnej postaci i wywolane(klientem/serwerem w zależności od rządania).
* W przypadku uzyskania komendy get, sprawdza określony plik i wysyła jego do klienta przez połączenie danych (port danych jest generowany serwerem) w postaci blokowej.
* W przypadku uzyskania komendy put, uzyskuje określony plik od klienta przez połączenie danych (port danych jest generowany serwerem) w postaci blokowej.
* W przypadku uzyskania komendy ls, wywolane zostaje polecenie popen("ls","r"), zaczym wysyłany spis plikow przez połączenie danych (port danych jest generowany serwerem) w postaci blokowej(lista plików może być dużego rozmiaru).
* W przypadku uzyskania komendy cd <directory>, wywolane zostaje polecenie chdir(<directory>), zaczym zmieniany katalog roboczy serwera.
* W przypadku uzyskania komendy pwd, wywolane zostaje polecenie \_getcwd, jeżeli serwerem jest maszyna pod systemem Windows, czy getcwd w przeciwnym przypadku, żeby uzyskać katalog roboczy serwera i nadeśłać jego do klienta.
* Zamyka się połączenie z klientem, jeżeli zostało uzyskane polecenie quit od klienta.
* Czeka na nowe połączenia i powtaża cykliczne opisany proces, dopóki nie zostanie wyłączony.