Data przekazania: 17.01.2021

Skład zespołu:
Krzysztof Blankiewicz
Jan Wasilewski
Rafał Rostkowski
Rafał Sztelmach

Dokumentacja projektu z przedmiotu TIN

Treść zadania:

Napisać program obsługujący uproszczoną wersję protokołu NFS.
Projekt będzie składał się z implementacji serwera, biblioteki pozwalającej na połączenie się z serwerem jako klient oraz programów klienckich przeznaczonych do testowania działania systemu.
Założenia:

- obsługa blokowania dostępu do plików (tryb “jeden pisarz lub wielu czytelników”),
- możliwość równoległego dostępu do wielu serwerów przez jednego klienta,
- udostępnienie funkcji biblioteki klienckiej:
    - mynfs_open
    - mynfs_read
    - mynfs_write
    - mynfs_lseek
    - mynfs_close
    - mynfs_unlink
    - mynfs_fstat

Interpretacja treści zadania:

- Komunikacja pomiędzy klientami a serwerem bazuje na protokole TCP.
- Klienci komunikują się z serwerem tylko poprzez użycie udostępnionych funkcji bibliotecznych.
- Serwer jest realizowany wieloprocesowo.
- Obsługa wielu klientów odbywa się poprzez przekazywanie komunikacji z klientami do procesów potomnych (jeden proces potomny na połączenie).
- Synchronizacja dostępu do plików w trybie “jeden pisarz lub wielu czytelników” jest realizowany z wykorzystaniem funkcji flock().
- Po wywołaniu mynfs_unlink() plik zostanie usunięty od razu, jeżeli jest zamknięty. W przeciwnym przypadku zostanie usunięty dopiero wtedy, kiedy zostanie zamknięty.

- Klienci operują na deskryptorach plików (innych niż te zwracane przez funkcje systemowe) otrzymanych od serwera, które są mapowane 1 do 1 z lokalnymi deskryptorami plików serwera.
- Serwer wykonuje operacje na lokalnych plikach poprzez użycie odpowiadających żądaniom klientów funkcji systemowych.
- Uwierzytelnianie zostanie zrealizowane w postaci whitelisty adresów IP na serwerze. Klienci których adresy znajdują się na liście mają prawo dostępu do systemu plików serwera.

Opis funkcjonalny:

Funkcje udostępnione klientowi nawiązują i kończą połączenie z serwerem w zależności od potrzeby, bez udziału użytkownika.
Możliwe następujące operacje:

- otwarcie pliku w trybach:
    - O_RDONLY - tryb tylko do odczytu
    - O_WRONLY - tryb tylko do zapisu
    - O_RDWR - tryb do zapisu i odczytu
    - O_CREAT - utworzenie pliku jeśli nie istnieje
- czytanie z pliku
- zapis do pliku
- usuwanie pliku
- zmiana pozycji w pliku
- zamknięcie pliku
- pobranie atrybutów otwartego pliku

Udostępnione funkcje:

int mynfs_open(char ***host**, char ***path**, int **oflag**, int **mode**);

Działa jak funkcja systemowa **open**() wywołana na rzecz pliku o ścieżce **path,** znajdującego się na serwerze o adresie **host**, z parametrami **oflag** i **mode**. Ta funkcja w razie potrzeby nawiąże połączenie TCP z serwerem. Zwraca deskryptor pliku przydzielony przez serwer. W przypadku błędu zwracana jest wartość -1, a kod błędu umieszczany jest w zmiennej globalnej **mynfs_error**.

int mynfs_read(char ***host**, int **fds**, void ***buf**, int **count**);

Działa jak funkcja systemowa **read**() wywołana na rzecz pliku o deskryptorze **fds**, znajdującego się na serwerze o adresie **host**, z parametrami **buf** i **count**. Zwraca ilość odczytanych danych (tak jak funkcja systemowa).
W przypadku błędu zwracana jest wartość -1, a kod błędu umieszczany jest w zmiennej globalnej **mynfs_error**.

int mynfs_write(char ***host**, int **fds**, void ***buf**, int **count**);

Działa jak funkcja systemowa **write**() wywołana na rzecz pliku o deskryptorze **fds**, znajdującego się na serwerze o adresie **host**, z parametrami **buf** i **count**. Zwraca ilość zapisanych danych (tak jak funkcja systemowa).
W przypadku błędu zwracana jest wartość -1, a kod błędu umieszczany jest w zmiennej globalnej **mynfs_error**.

int mynfs_lseek(char ***host**, int **fds**, int **offset**, int **whence**);

Działa jak funkcja systemowa **lseek**() wywołana na rzecz pliku o deskryptorze **fds**, znajdującego się na serwerze o adresie **host**, z parametrami **offset** i **whence**. Zwraca nową pozycję w pliku (tak jak funkcja systemowa).
W przypadku błędu zwracana jest wartość -1, a kod błędu umieszczany jest w zmiennej globalnej **mynfs_error**.

int mynfs_close(char ***host**, int **fds**);

Działa jak funkcja systemowa **close**() wywołana na rzecz pliku o deskryptorze **fds**, znajdującego się na serwerze o adresie **host**.
Ta funkcja w razie potrzeby zakończy połączenie TCP z serwerem. Zwraca 0 w przypadku powodzenia. W przypadku błędu zwracana jest wartość -1, a kod błędu umieszczany jest w zmiennej globalnej **mynfs_error**.

int mynfs_unlink(char ***host**, char ***path**);

Działa jak funkcja systemowa **unlink**() wywołana na rzecz pliku o ścieżce **path,** znajdującego się na serwerze o adresie **host**.
Ta funkcja w razie potrzeby nawiąże oraz zakończy połączenie TCP z serwerem. Zwraca 0 w przypadku powodzenia. W przypadku błędu zwracana jest wartość -1, a kod błędu umieszczany jest w zmiennej globalnej **mynfs_error**.

int mynfs_fstat(char ***host**, int **fds**, struct stat ***buf**);

Działa jak funkcja systemowa **fstat**() wywołana na rzecz pliku (potencjalnie folderu) o deskryptorze **fds,** znajdującego się na serwerze o adresie **host**. Zwraca 0 w przypadku powodzenia. W przypadku błędu zwracana jest wartość -1, a kod błędu umieszczany jest w zmiennej globalnej **mynfs_error**.

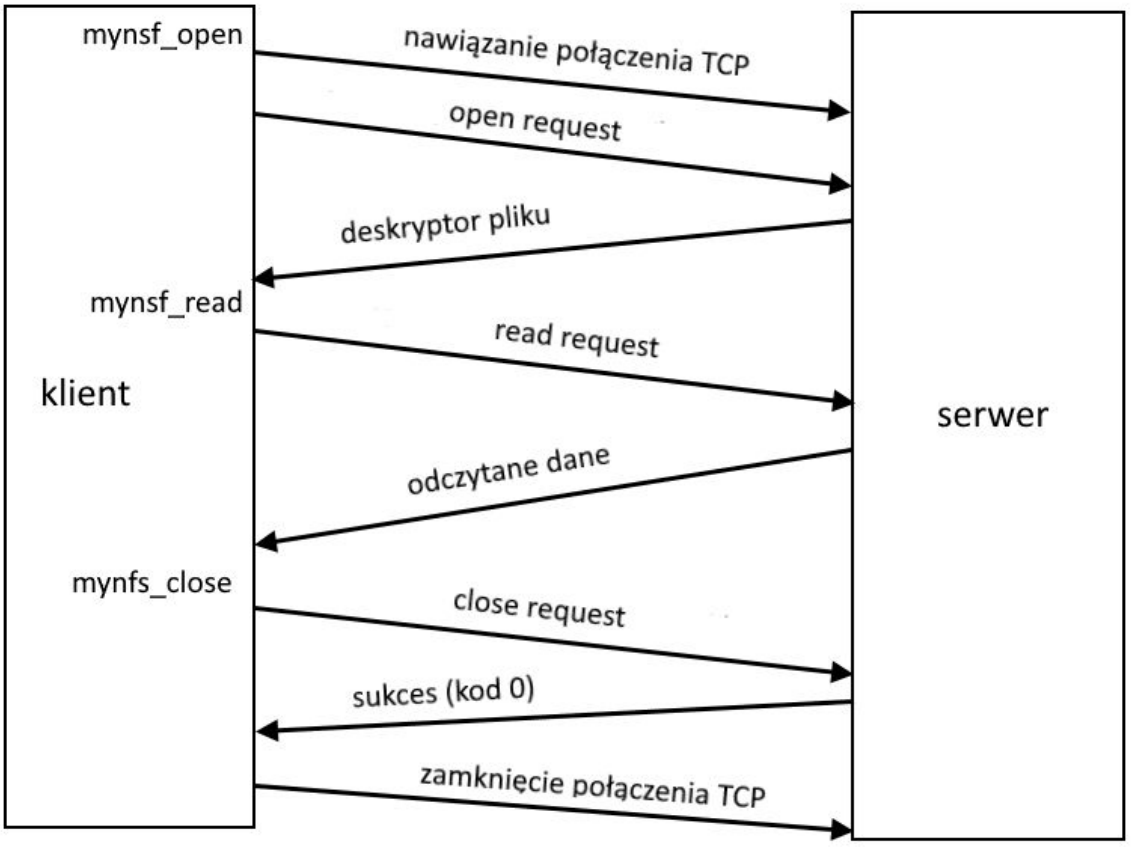
Deskryptor **fds** oznacza deskryptor zwrócony przez serwer po wywołaniu **mynfs_open**.

Serwer przechowuje listę aktywnych deskryptorów za pomocą klasy własnej na bazie tablicy stałej szerokości. Wynika z tego limit maksymalnie 10 jednocześnie otwartych plików na danym serwerze na klienta. (Klient może otworzyć dowolną ilość plików, ale maksymalnie 10 naraz na jednym serwerze. Serwer może mieć otwartą dowolną ilość plików, ale maksymalnie 10 dla jednego klienta.) Serwer przechowuje pary postaci: deskryptor_klienta - deskryptor_serwera.

Użytkownikowi nie zostanie udostępnione API blokowania plików. Blokowanie odbywa się automatycznie, bez udziału użytkownika, i na rzecz każdego otwieranego pliku. Dla plików otwieranych w trybie READ nakładane są zamki czytelnicze (wielokrotne współdzielone), a dla wszystkich pozostałych trybów nakładane są zamki pisarskie (na wyłączność). Przy zamykaniu pliku następuje zdjęcie odpowiadającego mu zamka.

Opis protokołów komunikacyjnych:

Na poniższym diagramie znajduje się przykład przebiegu komunikacji pomiędzy serwerem i klientem:



3) Każdy użytkownik w danej chwili jest połączony z serwerem co najwyżej jedną sesją TCP. Nawiązywanie sesji TCP i rozłączanie wykonywane jest automatycznie przez program kliencki, w sposób przezroczysty dla użytkownika.

Klient może znajdować się w 2 stanach:
A - ma obecnie zero otwartych plików na serwerze
B - ma obecnie co najmniej jeden otwarty plik na serwerze

w stanie A klient nie jest połączony sesją TCP
w stanie B klient jest połączony sesją TCP

polecenie mynfs_open() może wywołać przejście A -> B
w takiej sytuacji nawiązane zostanie połączenie TCP

polecenie mynfs_close() może wywołać przejście B -> A
w takiej sytuacji zakończone zostanie połączenie TCP

W przypadku wywołania polecenia mynfs_unlink() w stanie A nastąpi nawiązanie połączenia, usunięcie pliku, oraz zakończenie połączenia.

Format komunikatów:

Format komunikatu klienta (pola):

- identyfikator wywoływanej funkcji (uint8_t)
- odpowiednia struktura request

Format komunikatu serwera:

- odpowiednia struktura response

```
struct mynfs_open_request {
        uint16_t path_length;          //długość ścieżki pliku
        int16_t oflag;                 //parametr oflag funkcji
        int16_t mode;                  //parametr mode funkcji
        char *path;                    //ścieżka pliku
        }
```

```
struct mynfs_open_response {
        int32_t return_value;           //deskryptor pliku lub kod błędu
        }

struct mynfs_read_request{
        int32_t file_descriptor;        //deskryptor pliku
        int16_t data_size;              //ilość danych do przeczytania (parametr
                                        count funkcji)
        }

struct mynfs_read_response {
        int16_t return_value;       //liczba przeczytanych bajtów danych lub kod błędu
        int16_t data_size;          //rozmiar przesyłanych przez serwer danych
        void *data;                 //przesłane przez serwer dane
        }

struct mynfs_write_request{
        int32_t file_descriptor;    //deskryptor pliku
        int16_t data_size;          //ilość danych do zapisania (parametr
                                    count funkcji)
        void *data;                 //dane do zapisania
        }

struct mynfs_write_response {
        int16_t return_value;       //liczba zapisanych bajtów lub kod błędu
        }

struct mynfs_lseek_request{
        int32_t file_descriptor;    //deskryptor pliku
        int16_t offset;                 //parametr offset funkcji
        int16_t whence;             //parametr whence funkcji
        }

struct mynfs_lseek_response {
        int16_t return_value;       //nowa pozycja w pliku lub kod błędu
        }

struct mynfs_close_request{
        int32_t file_descriptor;    //deskryptor pliku
        }
```

```
struct mynfs_close_response {
        int16_t return_value;       //0 w przypadku powodzenia lub kod błędu
        }


struct mynfs_unlink_request{
        uint16_t path_length;       //długość ścieżki pliku
        char *path;                 //ścieżka pliku
        }

struct mynfs_unlink_response {
        int16_t return_value;       //0 w przypadku powodzenia lub kod błędu
        }


struct mynfs_fstat_request{
        int32_t file_descriptor;    //deskryptor pliku
        }

struct mynfs_fstat_response {
        int16_t return_value;       //0 w przypadku powodzenia lub kod błędu
        struct stat file_stats;     //struktura zawierająca informacje o pliku
        }
```

Podział i organizacja:

Projekt dzieli się na następujące części:

- program serwer (lub serwerów)
- biblioteka kliencka
- testowe programy klienckie

Główny proces programu serwera nasłuchuje na gnieździe pod kątem nowych połączeń. W momencie gdy zostanie nawiązane połączenie z nowym klientem zostaje utworzony nowy proces, który przejmuje komunikację z klientem oraz realizuje również żądania przez niego dostarczone. Lista dopuszczonych adresów IP (whitelist) jest przechowywana w pliku .txt, który jest odczytywany przez serwer przy uruchomieniu.

Zarys implementacji:

Projekt będzie realizowany w środowisku Linux w języku C++. Do komunikacji między serwerami a klientami posłużą gniazda BSD.
Do przechowywania par deskryptorów (serwer - klient) zostanie wykorzystany kontener *map* z biblioteki <*map*>.

Korzystanie i testowanie:

Należy upewnić się, że w katalogu z aplikacją serwera znajduje się plik ip_addresses.txt. Ten plik to lista adresów uprawnionych do łączenia się z serwerem.
Jeden adres na wiersz, format 255.255.255.255
Należy zadbać o umieszczenie adresu, z którego będzie łączył się klient w tym pliku.
Komunikacja odbywa się przez port 9999.

Należy uruchomić serwer, następnie uruchomić program kliencki. Można napisać własny korzystając z zapewnionych bibliotek, albo skorzystać z gotowych programów testowych.

Funkcje dostępne w bibliotece klienckiej przyjmują adres serwera oraz argumenty analogiczne do odpowiadających im funkcji systemowych.

Kody błędu umieszczane są w zmiennej **mynfs_error**. Kody dodatnie mają identyczne znaczenie z kodami umieszczanymi w zmiennej errno. Kody ujemne to kody błędu naszych funkcji.

-1	błąd tworzenia gniazda (socket())
-2	błąd połączenia (connect())
-3	błąd wysyłania (write())
-4	błąd odbioru (read())
-5	serwer odrzucił połączenie (adres klienta nie jest na liście dozwolonych adresów)
-6	próba wykonania polecenia na serwerze, z którym nie zostało nawiązane połączenie (np. próba użycia read na serwerze, na którym klient nie ma żadnego otwartego pliku.)
-7	nieoczekiwane zakończenie połączenia z serwerem
-8	próba otwarcia 11 pliku na serwerze (limit 10)

Kompilacja i uruchomienie programu serwera oraz klienta:

- kompilacja programu serwera:

```
g++ -c server.cpp -o server.o
g++ -o server server.o -L. -l_nfs
```

- uruchomienie programu serwera:

```
./server
```

- kompilacja testowego programu klienta (test#.cpp):
  # - numer 1, 2, 3, 4 lub 5

```
g++ -c test#.cpp -o test#.o
g++ -o test# test#.o -L. -l_nfs
```

- uruchomienie programu klienta (test#.cpp)

```
./test#
```

- opcjonalnie można dokonać kompilacji wszystkich plików źródłowych przy pomocy załączonego skryptu (comp.sh) za pomocą komendy:

```
sh comp.sh
```

Opis rozwiązań funkcjonalnych:

Po uruchomieniu serwer tworzy gniazdo i nasłuchuje.
Serwer tworzy nowy proces dla każdego zaakceptowanego połączenia. Utworzonemu procesowi zostaje przekazany deskryptor gniazda do komunikacji z klientem. Każdy proces przechowuje własny słownik deskryptorów, mapujący deskryptory używane przez serwer na deskryptory udostępnione użytkownikowi.

Klient z danym serwerem łączy się za pomocą co najwyżej jednego gniazda. Biblioteka przechowuje informację o deskryptorze gniazda powiązanego z danym adresem serwera, o ile takowe istnieje. Przechowuje również liczbę otwartych plików na tym serwerze. Na podstawie tych informacji decyduje o nawiązywaniu i zamykaniu połączenia. W momencie utraty lub zamknięcia połączenia gniazda po obu stronach zostają zamknięte.

Testowanie:

Spodziewany wynik testu 1:

Utworzony zostanie plik o nazwie “mynfs_testing_plik.txt”.
W pliku zostaje zapisane zdanie “Ala ma kota”.

Uzyskany wynik testu 1:

Sukces

Utworzony został plik o nazwie “mynfs_testing_plik.txt”.
W pliku znajduje się zdanie “Ala ma kota”

Spodziewany wynik testu 2:

Za pomocą poleceń lseek() oraz write() zawartość pliku o nazwie “mynfs_testing_plik.txt” zostaje zmieniona na “Ala ma wielkiego psa”.

Uzyskany wynik testu 2:

Sukces
W pliku znajduje się zdanie “Ala ma wielkiego psa”

Spodziewany wynik testu 3:

Otwarty zostanie plik o nazwie “mynfs_testing_plik.txt”.
Zawartość zostanie odczytana i wypisana na konsolę klienta.

Uzyskany wynik testu 3:

Sukces.

Spodziewany wynik testu 4:
Pobrane zostaną informacje o pliku o nazwie “mynfs_testing_plik_txt” (polecenie fstat()).
Na konsolę klienta zostanie wypisany jego rozmiar (st_size) oraz ID właściciela (st_uid).

Uzyskany wynik testu 4:

Sukces.

Spodziewany wynik testu 5:
Plik o nazwie “mynfs_testing_plik.txt” zostanie usunięty.

Uzyskany wynik testu 5:

Sukces.
Plik został usunięty.