Techniki Internetowe (TIN)- 2020Z

Projekt: Aplikacja NFS - Sprawozdanie wstępne

Zespół w składzie:
– Mateusz Kalinowski
– Mateusz Kiljan
– Paweł Szafrański
– Paweł Wieczorek

Data przekazania: 26.11.2020

Spis treści

Treść zadania

Celem projektu jest zaimplementowanie sieciowego dostępu do zewnętrznego systemu plików poprzez stworzenie biblioteki klienckiej, programu serwera oraz aplikacji klienckiej. Komunikacja zgodnie z treścią zadania powinna bazować na TCP. Przygotowane rozwiązanie ma udostępniać podstawowe funkcje do operacji na plikach, tj. otwarcie, odczyt, zapis, przesunięcie (*lseek*), usuwanie oraz pobieranie właściwości (*fstat*). Ponadto, powinno zapewniać blokowanie dostępu do plików w trybie „jeden pisarz albo wielu czytelników” (W12) oraz umożliwiać jednoczesny dostęp do wielu niezależnych serwerów z poziomu aplikacji klienckiej (W32).

Doprecyzowanie treści

Blokowanie w trybie „jeden pisarz albo wielu czytelników” działa następująco: za każdym razem, gdy klientowi udało się otworzyć plik sprawdza czy musi zablokować ten plik wywołując systemową funkcję *flock()* na nim. Przy pomocy odpowiednich instrukcji warunkowych zarządza wtedy blokadą tak, żeby w żadnym momencie nie zaistniała jedna z poniższych sytuacji:
1) Pewien klient ma otwarty plik X w trybie dopuszczającym pisanie, a innemu udaje się otworzyć plik w jakimkolwiek trybie.
2) Pewien klient ma otwarty plik X w trybie O_RDONLY, a innemu udaje się otworzyć plik w trybie innym niż O_RDONLY.

Serwer przechowuje deskryptory lokalnie otwartych plików, a użytkownik dysponuje identyfikatorem pliku. Serwer przechowuje słownik (std::map) realizujący mapowanie między deskryptorami lokalnie otwartych plików serwera, a identyfikatorami przekazywane klientom.

Ponadto, serwer przechowuje listę z informacjami o otwartych plikach dla sesji klienta, tak żeby móc automatycznie zamknąć wszystkie deskryptory po jej zakończeniu. Dzięki temu poprawnie zostanie obsłużona sytuacja, gdy klient zerwie połączenie przed zamknięciem swoich plików.

Opis funkcjonalny

W ramach przygotowywanego projektu planujemy zaimplementować następujące funkcjonalności:

Po stronie klienta:

1. **Odczyt z oraz zapis do plików** - odpowiedniki systemowych read() oraz write()
2. **Usuwanie plików** - w sytuacji, gdy użytkownik próbuje usunąć otwarty plik, operacja ta zostanie wykonana dopiero gdy dany plik zostanie zamknięty u każdego użytkownika
3. **Przesunięcie kursora** - odpowiednik funkcji systemowej lseek()
4. **Pobieranie właściwości pliku** - odpowiednik funkcji systemowej fstat()
5. **Otwieranie plików** – można otworzyć plik w trybach *O_RDONLY, O_WRONLY, O_RDWR, O_CREAT*

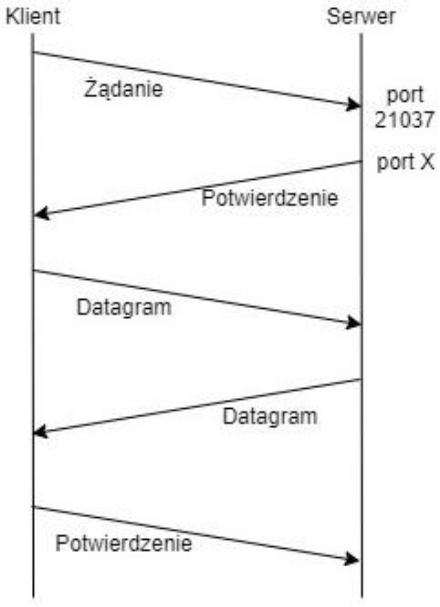
Po stronie serwera:

1. **Autoryzacja użytkowników z wykorzystaniem mechanizmu *Linux-PAM*** – takie rozwiązanie pozwala na wykorzystanie mechanizmu uprawnień obecnego w systemie Linux do prostego ograniczenia uprawnień dostępu do katalogów i plików
2. **Wsparcie dla prostego pliku konfiguracyjnego** – parametry działania aplikacji serwerowej jak: ścieżka do udostępnianego katalogu, maksymalna ilość jednocześnie otwartych deskryptorów czy lokalizacja pliku z logami będą podawane poprzez prosty plik konfiguracyjny o składni zbliżonej do tej stosowanej w pliku *.profile* (i.e. *nazwa=wartość*).
3. **Logowanie zdarzeń na terminal oraz do pliku lokalnego** – wszystkie istotne zdarzenia (w szczególności: logowanie nowego użytkownika, otwarcie czy usunięcie pliku, błędy etc.) będą odnotowane na wyjściu programu (*stdout*) oraz w osobnym pliku wraz z informacją o dacie wystąpienia oraz identyfikatorze użytkownika, który je spowodował.

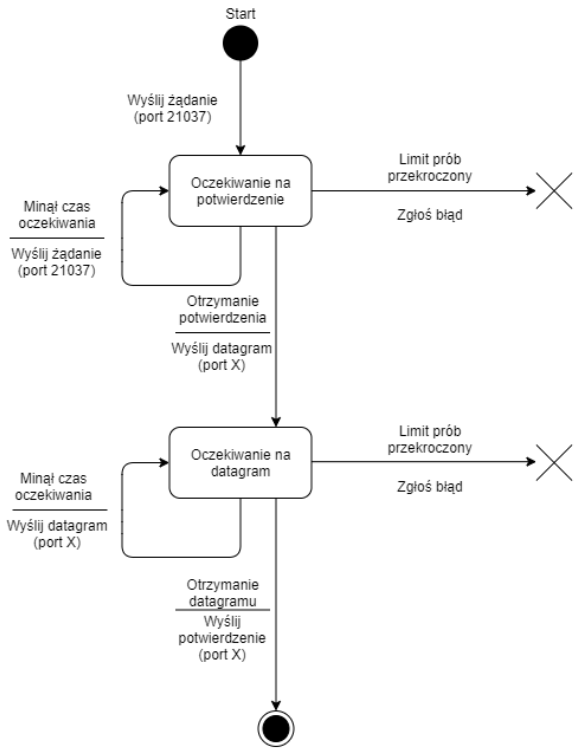
Protokół komunikacyjny

Komunikacja pomiędzy serwerem, a klientem obywać się będzie z wykorzystaniem socketa głównego TCP stale otwartego na porcie 21037 oraz socketów indywidualnych dla każdego zgłaszającego się klienta, które podczas pracy serwera będą otwierane i zamykane. Komunikacja z klientem zostaje przerwana po upływie zadanego okresu (5 minut) bezczynności.

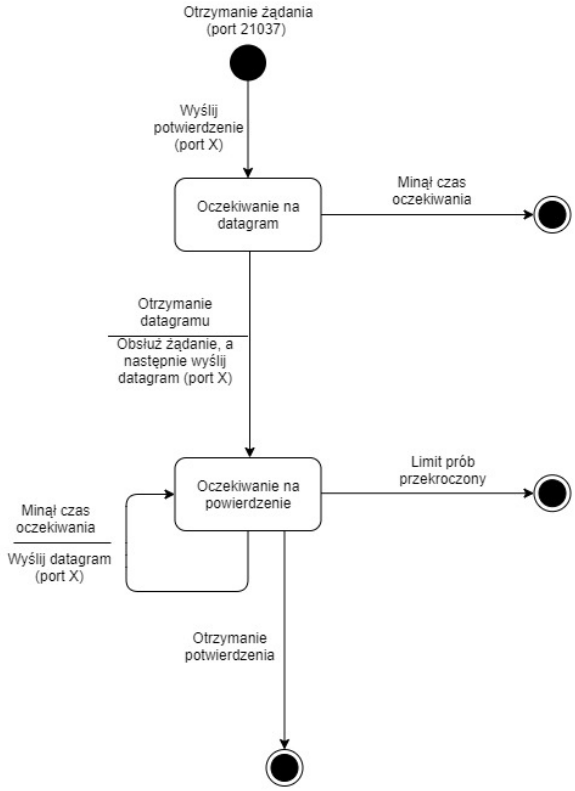
Przepływ informacji pomiędzy klientem a wybranym serwerem wygląda w następujący sposób:



Rys. 1. Schemat komunikacji pomiędzy klientem a serwerem

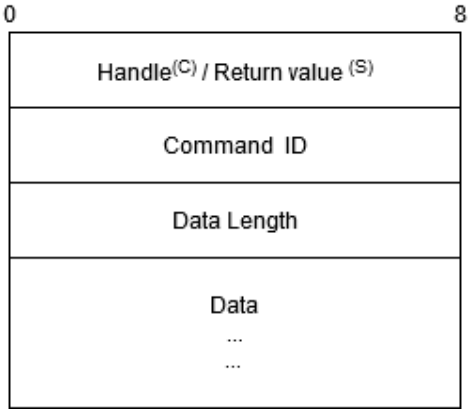


Rys. 2. Diagram stanów klienta



Rys. 3. Diagram stanów serwera

Każdy pakiet danych będzie enkapsulowany w następujący format:



Rys. 4 Budowa nadrzędnego datagramu
(S) – pole używane w danych zwracanych z serwera,
(C) – w danych od klienta

Odpowiednikiem powyższego pakietu jest następująca struktura w języku C:

Lst. 1. – Budowa datagramu

```
struct mynfs_datagram_t {
   union {
      int64_t handle;
      int64_t return_value;
   };
   uint64_t cmd;
   size_t data_length;
   char data[0];
}
```

Rys. 5 Budowa nadrzędnego datagramu jako struktury w języku C

Opis datagramu

W ogólności, pole *handle* przechowuje identyfikator pliku, na którym ma zostać wykonana żądana operacja, pole *return_value* wartość zwróconą przez funkcję serwera, *cmd* kod operacji do wykonania, a *data_length* długość danych przekazywanych w tablicy *data*. Format danych zawartych w polu *data* zależny jest od rodzaju wywoływanej funkcji – szczegóły dla każdej z planowanych komend są przedstawione w poniższej tabeli:

CMD	Nazwa	Dane wejściowe – format C	Dane wyjściowe

0	mynfs_open	struct mynfs_open_t{
		int oflag;
		int mode;
		size_t path_length;
		char name[0];
		}
			brak – uchwyt zwracany jest w polu *return_value* datagramu nadrzędnego

1	mynfs_read	struct mynfs_read_t {
		size_t length;
		}
			tablica *data* zawiera odczytane bajty o długości określonej w *data_length*

2	mynfs_write	struct mynfs_write_t {
		size_t length;
		char buffer[0];
		}
			brak – ilość zapisanych bajtów zwraca jest w polu *return_value* (analogicznie jak w funkcji *read* z stdlib)

3	mynfs_lseek	struct mynfs_read_t {
		size_t offset;
		int whence;
		}
			brak

4	mynfs_close	**brak** – identyfikator jest przekazywany w polu *handle*	**brak**
5	mynfs_fstat	**brak** – identyfikator jest przekazywany w polu *handle*	tablica *data* zawiera strukturę *stat* zdefiniowaną w *sys/stat.h*

6	mynfs_unlink	struct mynfs_open_t{
		size_t path_length;
		char name[0];
		}
			brak

Budowa projektu

Projekt składa się z trzech części: aplikacji serwera, biblioteki klienckiej z rozszerzeniem .so oraz aplikacji klienckiej korzystającej z tej biblioteki. Implementacja każdej z tych części będzie podzielona na wydzielone moduły.

Aplikacja kliencka będzie składała się z modułu odpowiedzialnego za interfejs użytkownika i z modułu realizującego komunikację z serwerem (wykorzystanie funkcji z biblioteki). Podczas korzystania z aplikacji, klient będzie posiadał możliwość wyboru jednego z wielu niezależnych systemów plików znajdujących się na oddzielnych architekturach (maksymalnie 3). Możliwe będą zarówno inicjalizacja połączenia jak i korzystanie z tychże systemów w tym samym momencie. Po stronie klienta przechowywane będą informacje o serwerach tzn.: adresy IP oraz numery socketów, na których dany serwer nasłuchuje żądań.

Aplikacja serwera będzie składała się z trzech modułów:

1. **Moduł obsługi żądań.** Odpowiada on za nasłuchiwanie na głównym porcie (21037) żądań od klientów. Po otrzymaniu prawidłowego żądania tworzy nowy socket i przypisuje go do nowego portu, który będzie służył do dalszej komunikacji z konkretnym klientem. Zwraca klientowi nowy numer portu.
2. **Moduł komunikacji z klientem.** Odpowiada za przyjmowanie wszystkich nadchodzących poleceń od klienta na jego indywidualnym porcie oraz wysyłaniu mu odpowiednich danych i komunikatów. W celu wykonania tych poleceń przekazuje je do modułu obsługi plików.
3. **Moduł obsługi plików.** Odpowiada za operacje na lokalnych plikach serwera oraz kontrolowanie blokowania w trybie „jeden pisarz albo wielu czytelników”. Zwraca modułowi komunikacji z klientem odpowiednie dane oraz komunikaty diagnostyczne.

Implementacja

Rozwiązanie zostanie przygotowane w języku C++/C z wykorzystaniem implementacji socketów BSD dostępnej w *stdlib*, systemu kontroli wersji git oraz systemu budowania *make*.