Witold Oleszkiewicz, Marcin Bryk, Dawid Dominiak, Dawid Wojciechowski

TECHNIKI INTERNETOWE - DOKUMENTACJA WSTĘPNA PROJEKTU

1. Treść zadania

Celem projektu jest implementacja aplikacji do transferu zasobów pomiędzy węzłami w sieci, z wykorzystaniem protokołu UDP.

- Zasób to plik identyfikowany nazwą.
- W sieci może jednocześnie funkcjonować wiele różnych zasobów o tej samej nazwie, można przyjąć, że w jednym węźle sieci może istnieć tylko jeden zasób o danej nazwie.
- Nazwa to dowolny napis w kodzie ASCII z arbitralnie ograniczoną długością.
- Wielkość zasobu może być ograniczona, ale powinny być obsługiwane zasoby o "dużych rozmiarach" np. 2^32 -1 B.
- Lokalizacja zasobu:
    - w celu zlokalizowania zasobu, komputer "poszukujący" rozgłasza w sieci zapytanie czy istnieje plik o danej nazwie
    - na zapytanie może przyjść 0, 1 lub więcej odpowiedzi, brak odpowiedzi w określonym czasie (timeout) oznacza brak zasobu
- Transfer zasobu:
    - transfer zasobu z węzła źródłowego odbywa się w protokole UDP
    - należy uwzględnić gubienie datagramów (potwierdzenia, retransmisje)
    - możliwość równoległego transferu zarówno po stronie nadawcy jak i odbiorcy
- Interfejs:
    - prosty interfejs tekstowy
    - transfery zasobów przebiegają asynchronicznie w stosunku do interakcji z użytkownikiem
    - informacje o przebiegu transferu logowane do pliku
- Sytuacje wyjątkowe:
    - usunięcie zasobu w czasie transferu
    - timeout-y komunikacji
- Do doprecyzowania:
    - format komunikatów
    - sposób funkcjonowania i polecenia UI
    - obsługa błędów

2. Założenia wstępne

2.1. Interpretacja i założenia wynikające wprost z treści:

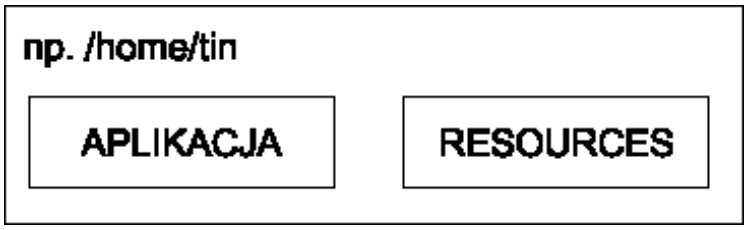
a) Aplikacja umożliwi transfer zasobów w obrębie węzłów sieci lokalnej.
b) W pojedynczym węźle nazwa zasobu w jednoznaczny sposób identyfikuje plik.
c) Długość nazwy zasobu ograniczona do 50 znaków, nazwy zasobów o dłuższej nazwie będą nie obsługiwane, (użytkownikowi zostanie wyświetlony stosowny komunikat o błędzie.)
d) Maksymalna wielkość pliku 2^32 - 1 B.
e) Każdy pojedynczy transfer będzie logowany do pliku tekstowego o nazwie w postaci: *nazwa_pliku_id_wątku_timestamp_początku_pobierania/wysyłania.log* zarówno po stronie nadawcy jak i odbiorcy. W tym pliku będą umieszczane na bieżąco informacje na temat czasu rozpoczęcia pobierania, komunikaty na temat powodzenia lub niepowodzenia wysłania poszczególnych datagramów i przekroczonych timeoutów oraz wszystkich innych sytuacjach wyjątkowych.

2.2. Założenia dodatkowe:

a) Każdy transfer otrzyma unikalny identyfikator, który będzie znany dla odbiorcy, będzie on służył do rozróżniania poszczególnych transferów. Identyfikator będzie tworzony według wzoru: *nazwa_pliku_id_wątku*
Np. dla pobierania pliku przykład.txt, inny_przykład.txt identyfikatory przyjmą kolejno wartości:

```
przykład.txt_0   inny_przykład.txt_1
```

b) Zasoby dostępne do wysyłania będą znajdowały się w folderze `Resources`, który z kolei znajdował się będzie w tej samej lokalizacji co aplikacja. Jeśli któryś z zasobów nie będzie spełniał ustalonych ograniczeń (na długość nazwy, rozmiar zasobu), to aplikacja nie będzie go udostępniać. Węzeł po otrzymaniu zapytania o zasób, sprawdzi fizyczne istnienie pliku. Jeśli plik zostanie usunięty już po wysłaniu odpowiedzi, aplikacja obsłuży taki przypadek jako usunięcie pliku w trakcie połączenia. Także pliki pobierane będą umieszczane we wspomnianym folderze. Aplikacja przez rozpoczęciem wyszukiwania zadanego pliku w sieci, najpierw sprawdzi czy znajduje się on we własnych zasobach, a jeśli zostanie on wykryty, to nie będzie go pobierać i wyświetli odpowiedni komunikat dla użytkownika.



Rys. 1. Konwencja lokalizowania aplikacji i zasobów plikowych na dysku

c) Brak uwierzytelniania pomiędzy węzłami przed rozpoczęciem transferu.

3. Funkcjonalny opis "black box", interfejs użytkownika.

Po uruchomieniu aplikacji użytkownikowi zostanie wyświetlony znak zachęty $. Po wpisaniu komendy *help* użytkownikowi wyświetlony zostanie zestaw komend dostępnych w programie:

a) **Wyszukiwanie zasobu w sieci i transfer:**
`get [nazwa_pliku]` – polecenie wysłania zapytania do wszystkich węzłów sieci w celu znalezienia zasobu. Żądanie zasobu będzie rozgłaszane do wszystkich węzłów sieci, przy czym polecenie find poprzedzi sprawdzenie czy rzeczywiście użytkownik potrzebuje tego zasobu (być może plik o takiej nazwie już istnieje w tym węźle). Węzeł pytający po otrzymaniu kilku odpowiedzi arbitralnie wybiera pierwsze, które przyszło. W przypadku braku odpowiedzi w ustalonym czasie, przyjmuje się, że zasobu w sieci nie ma o czym użytkownik będzie poinformowany odpowiednim komunikatem wypisanym na konsolę. Jeżeli operacja wyszukania zasobu w sieci powiedzie się, to rozpocznie się transfer zasobu, zostanie wypisany na konsolę komunikat, który będzie zawierał identyfikator transferu. Po zakończeniu transferu pojawi się komunikat o skutecznym pobraniu zasobu.

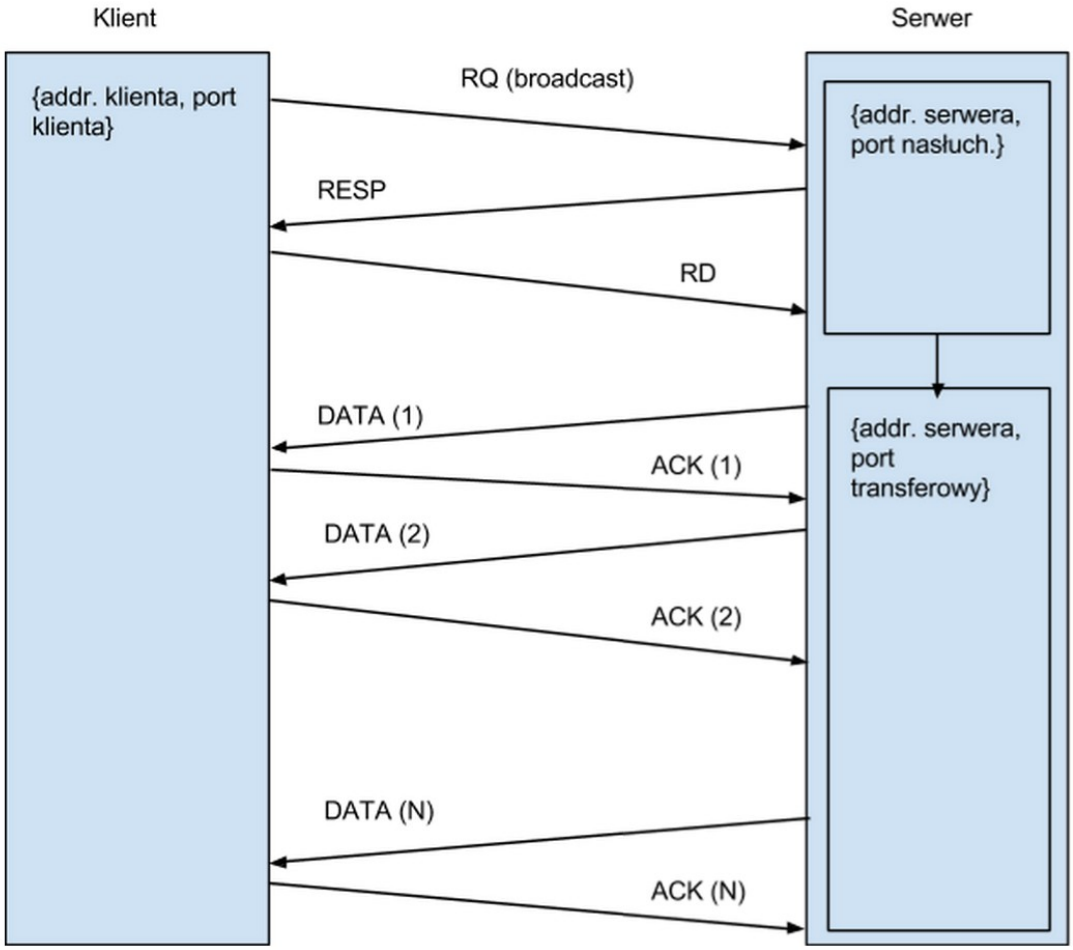
b) **Przerwanie transferu**:
`cancel [ID_transferu]` – anuluj transfer o danym identyfikatorze. W trakcie transferu użytkownik będzie mógł wydać polecenie jego zaprzestania.

c) **Prezentacja aktualnych pobierań**:
`show` – pokaż aktualnie trwające transfery - wyświetla identyfikatory aktualnie trwających pobrań zasobów.

d) **Prezentacja statusu danego transferu**:
`transferinf [ID_transferu]` - wyświetl status pobierania danego transferu.

4. Protokół komunikacyjny

Zgodnie z treścią zadania zaprojektowany protokół wykorzystuje w warstwie transportowej UDP, natomiast w warstwie sieciowej IPv4.



Rys. 3. Schemat wymiany pakietów między serwerem a klientem.

1) W odpowiedzi na żądanie odnalezienia pliku o zadanej nazwie, uruchomiony zostanie wątek klienta obsługi żądania, który rozgłasza wszystkim węzłom pakiet typu `RQ` z nazwą pliku na port nasłuchujący serwerów.
2) Jeśli serwer udostępnia zadany plik, odpowiada wysłaniem pakietu `RESP`. W przypadku nie otrzymania żadnego pakietu `RESP` w określonym czasie, wątek klienta stwierdza brak poszukiwanego pliku i kończy działanie informując o tym użytkownika.
3) Jeśli nadeszła odpowiedź, następuje wybór jednego z serwerów (strategia "pierwszy lepszy") i wysłanie do niego pakietu `RD` z żądaniem przesłania pliku o podanej nazwie. W wyniku tego serwer uruchamia wątek obsługi transferu na nowym gnieździe, który inicjuje transmisję wysłaniem pierwszego pakietu danych (`DATA`).
4) Kolejny pakiet zostanie wysłany przez serwer po otrzymaniu potwierdzenia dostarczenia poprzedzającego go pakietu danych (pakiet typu `ACK`).
5) Koniec transmisji następuje po potwierdzeniu otrzymania przez klienta porcji danych o rozmiarze mniejszym niż maksymalny rozmiar pola danych określany przez protokół tj. 4 KB bądź też pakietu z pustym polem danych, gdy rozmiar pliku jest podzielny przez 4 KB.

W przypadku zagubienia się któregoś z pakietów typu `DATA`, `ACK` lub `RD` (rozpoznawanym po braku odpowiedzi drugiej strony), po określonym czasie TO1 (lub TO2 w przypadku `RD`) następuje jego retransmisja. Gdy po czasie TO3 > TO1 (TO3 > TO2) nie uzyskamy odpowiedzi, transmisja zostaje zakończona niepowodzeniem. W przypadku wystąpienia błędu po którejkolwiek ze stron, wysłany zostaje jednokrotnie specjalny pakiet błędu `ERR` zawierający informację o zaistniałym problemie (np. błąd dostępu do pliku), który również kończy transmisję. Pakiety zduplikowane (w czasie przesyłania lub w wyniku retransmisji) są odrzucane i niepotwierdzane. Kolejność otrzymywania pakietów nie stanowi problemu, ze względu na to, iż nadanie kolejnego komunikatu z sekwencji następuje w wyniku otrzymania wcześniejszego, nadanego przez drugą stronę, a pakiety otrzymane poza właściwą kolejnością są odrzucane.

Wartości TO1, TO2, TO3 zostaną wyspecyfikowane w ramach testowania programu.

Struktura pakietu:

- pole TYPE - typ pakietu {RQ, RESP, RD, DATA, ACK, ERR} – pole obowiązkowe;
- pole NUMBER - numer bloku danych lub potwierdzenia otrzymania bloku danych lub numer błędu
- pole DATA SIZE – rozmiar bloku danych w bajtach lub całkowity rozmiar żądanego pliku
- pole DATA - blok danych
- pole STR LENGTH – długość nazwy pliku w bajtach
- pole FILENAME – nazwa pliku (w ASCII)

Pakiet danych - DATA

TYPE	NUMBER	DATA SIZE	DATA
8 bitów	20 bitów	12 bitów	do 4 kB

Pakiet potwierdzenia – ACK

TYPE	NUMBER
8 bitów	20 bitów

Pakiet żądania zasobu - RQ

TYPE	STR LENGTH	FILENAME
8 bitów	8 bitów	do 50 bajtów

Pakiet odpowiedzi na żądanie zasobu - RESP

TYPE	DATA SIZE
8 bitów	32 bity

Pakiet inicjujący pobieranie zasobu - RD

TYPE	STR LENGTH	FILENAME
8 bitów	8 bitów	do 50 bajtów

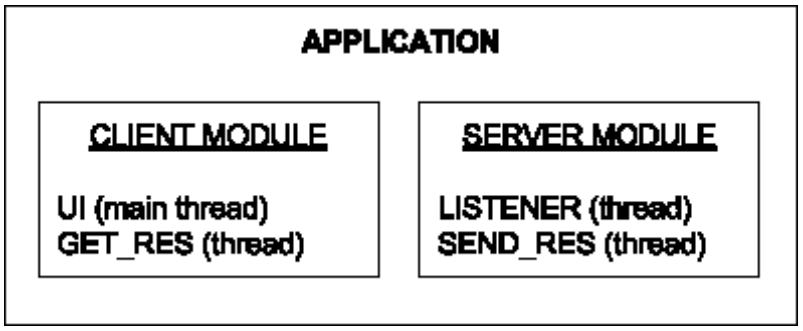
Pakiet błędu - ERR

TYPE	NUMBER
8 bitów	8 bitów

Możliwe sytuacje wyjątkowe i ich obsługa

Sytuacja wyjątkowa	Akcja
Zgubienie pakietu	Retransmisja pakietów DATA, ACK, RD, RESP po timeoucie T1 (T2 dla RESP). Komunikat o niepowodzeniu przy wyszukiwaniu pliku dla pakietów RQ i RD.
Zerwanie połączenia (nie docierają żadne pakiety)	Po time oucie TO3: zakończenie transferu po stronie klienta (serwera), uzupełnienie logów, usunięcie otrzymanych dotychczas bloków, zamknięcie otwartych deskryptorów plików, zakończenie wątku GET_RES (SEND_RES) wraz z powiadomieniem użytkownika odbierającego pliki.
Powielenie pakietów DATA, ACK	Klient pamięta numer ostatniego otrzymanego bloku danych, jeśli nowy pakiet danych ma numer mniejszy lub równy zapamiętanemu – odrzucamy pakiet. Analogiczna sytuacja ma miejsce po stronie serwera.
Usunięcie lub błąd w dostępie do pliku po stronie serwera w trakcie transferu	Obsługa błędu funkcji czytania, wysłanie pakietu ERR, zakończenie transferu, uzupełnienie logów, usunięcie otrzymanych dotychczas bloków, zamknięcie otwartych deskryptorów plików, zakończenie wątku GET_RES wraz z powiadomieniem użytkownika odbierającego pliki.
Duplikacja pakietu RD	W tej sytuacji mogłoby dojść do duplikacji transferu do wielokrotności duplikacji pakietu RD. W celu uniknięcia tego problemu klient powinien się połączyć z serwerem za pomocą funkcji connect() dla gniazda UDP w celu zapamiętania adresu docelowego wysłanych datagramów.

5. Struktura programu.



Rys. 2. Architektura aplikacji.

Aplikacja podzielona będzie na dwa główne moduły obsługi sieciowej oraz część obsługi interfejsu tekstowego i związanych z nim poleceń.

Klient:
W tym module zaimplementowane zostaną dwie podstawowe funkcje z punktu widzenia klienta - wyszukiwanie zasobu oraz odbieranie danych. Obie funkcje będą wykonywane w wątku `GET_RES`, uruchamiany z głównego wątku (wątek UI, które będzie odpowiadać za interakcję z użytkownikiem w konsoli).

Serwer:
W części serwerowej znajdą się funkcje usługowe. Kluczowym elementem będzie wątek nasłuchujący nadchodzących zgłoszeń. Wątek ten będzie uruchamiany jako wątek potomny wątku głównego (`UI`), tworzony zaraz po starcie aplikacji. Wysyłanie zasobu będzie realizowane w nowym wątku.

Każdy transfer będzie realizowany w nowym wątku wysyłającym (po stronie nadawcy) oraz odbierającym (po stronie odbiorcy). Wątki te uruchamiane będą z poziomu wątku Listenera dla serwera (uruchamia `SEND_RES`) i wątku `UI` klienta (uruchamia `GET_RES`). Wątek wyszukujący będzie uruchamiany z poziomu głównego procesu aplikacji, odpowiedzialnego za interakcje z użytkownikiem. W ten sposób spełnione zostanie wymaganie zachowania asynchroniczności transmisji względem interakcji na linii aplikacja - użytkownik.

Komunikacja i synchronizacja pomiędzy wątkami:
Przyjęta architektura aplikacji ogranicza komunikację pomiędzy wątkami do dwóch sytuacji:

1. Polecenie `trasferinf`. Wątek główny odpowiedzialny za interakcję z użytkownikiem na podstawie danych zawartych w globalnej liście struktur wyświetli pasek postępu. Prototyp struktury przechowującej wiadomości dla wątków:

```
struct RunnigTask
{
      char* filename;       // nazwa pobieranego pliku
      int last_block;       // nr ostatnio pobranego bloku danych
      int all_blocks;       // liczba wszystkich bloków danych pliku
      int id_thread;        // id wątku GET_RES
      bool interrupt_req;   // flaga żądania przerwania wątku
      bool dirty;           // flaga zakończenia wątku
}
```

   Dane w liście będą aktualizowane przez wątki pobierające, a dostęp do elementów tablicy będzie synchronizowany semaforem binarnym.

2. Polecenie `cancel`, wykonywane z poziomu wątku głównego w przedstawionej powyżej strukturze `RunningTask` ustawia flagę `interrupt_req`. Wątek realizujący transfer po odczytaniu takiej flagi zakończy działanie.

Dodatkowo dostęp wątków do wypisywania komunikatów na konsoli będzie synchronizowany muteksem.

6. API.

1. Wyszukanie pliku o zadanej nazwie:

   ```
   int find_by_name(char* filename, struct timeval timeout)
   ```

2. Wysłanie odpowiedzi przez wątek `Listener` po znalezieniu pliku:

   ```
   int respond_to_find()
   ```

3. Pobranie pliku:

   ```
   int download_file(char* filename, struct sockaddr *saddr, socklen_t
   addrlen)
   ```

4. Wysłanie pliku:

   ```
   int send_file(char* filename, struct sockaddr *saddr, socklen_t
   addrlen)
   ```

7. Zarys koncepcji implementacji (język, biblioteki, narzędzia)

Projekt zostanie zrealizowany w języku C++ z użyciem API Gniazd BSD (sockets). Środowiskiem implementacyjnym i uruchomieniowym będzie środowisko Linux Mint. Budowanie i kompilacja aplikacji przy użyciu GNU Compiler Collection oraz makefile. W celu sprawnej pracy całego zespołu jako system kontroli wersji wybrany został Git, a repozytorium kodu powstało na portalu GitHub.