TIN P2P torrents

Dokumenacja wstępna

27 kwietnia 2021

Spis treści

Autorzy:

- Igor Kaźmierczak 293118 - **lider**
- Bartłomiej Olber 300237
- Jakub Gałat 300209
- Konrad Kulesza 300247

Treść zadania

Treść: Napisać program obsługujący prosty protokół P2P (Peer-to-Peer). Założenia:

- Zasób to plik identyfikowany pewną nazwą, za takie same zasoby uważa się zasoby o takich samych nazwach.
- Początkowo dany zasób znajduje się w jednym weźle sieci, następnie może być propagowany do innych węzłów w ramach inicjowanego przez użytkownika ręcznie transferu (patrz dalej) – raz pobrany zasób zostaje zachowany jako kopia.
- Tak więc, po pewnym czasie działania systemu ten sam zasób może znajdować się w kilku węzłach sieci (na kilku maszynach).
- Program ma informować o posiadanych lokalnie (tj. w danym węźle) zasobach i umożliwiać ich pobranie.
- Program powinien umożliwiać współbieżne:
  - wprowadzanie przez użytkownika (poprzez interfejs tekstowy):
    - nowych zasobów – z lokalnego systemu plików,
    - poleceń pobrania nazwanego zasobu ze zdalnego węzła ,
  - pobieranie zasobów „w tle” (także kilku jednocześnie),
  - rozgłaszanie informacji o posiadanych lokalnie zasobach.
- W przypadku pobierania zdalnego zasobu system sam (nie użytkownik) decyduje skąd zostanie on pobrany.
- Zasób pobrany do lokalnego węzła jest kopią oryginału, kopia jest traktowana tak samo jak oryginał (są nierozróżnialne) – t.j.: istnienie kopii jest rozgłaszane tak samo jak oryginału. {Duplikat}
- Właściciel zasobu może go unieważnić wysyłając odpowiedni komunikat rozgłaszany. Wszystkie kopie zasobu po unieważnieniu powinny przestać być rozgłaszane. W przypadku gdy trwają transfery zasobu, który został unieważniony powinny się one poprawnie skończyć, dopiero wtedy informacja o zasobie powinna być usunięta. {Kasowanie}

- Należy zwrócić uwagę na różne obsługę różnych sytuacji wyjątkowych – np. przerwanie transmisji spowodowane błędem sieciowym.
- Lokalizacja zasobów ma następować poprzez rozgłaszanie – wskazówka: użyć prot. UDP, ustawić opcje gniazda SO_BROADCAST, wykorzystać adresy IP rozgłaszające (same bity "1" w części hosta).
- Można dodatkowo wprowadzić skrót lub podpis, aby zapewnić jednoznaczność identyfikacji zasobów.
- Interfejs użytownika – wystarczy prosty interfejs tekstowy

Nasz wariant: W2-Powinno być możliwe pobranie zasobu z kilku węzłów na raz (tj. "w kawałkach").

Słowniczek:

- **serwer** - oddzielny wątek oraz jego wątki pochodne, które są odpowiedzialne za wszystko co dzieje się pod spodem: komunikacja z innymi węzłami, pobieranie zasobów, udostępnianie, zarządzanie lokalnymi zasobami, synchronizacja.
- **klient** - wątek odpowiedzialny za komunikację z użytkownikiem korzystającym z naszego systemu i przekazywanie odpowiednich żądań do i od serwera.
- **właściciel zasobu** - osoba(osoby), która jako pierwsza wprowadziła zasób do sieci; osoby znające identyfikator zasobu, na podstawie którego liczony jest hash.
- **katalog roboczy** - katalog, w którym będą przechowywane wszystkie zasoby operacyjne: zarówno udostępnianie jak i pobierane.
- **chunk** - niezależny do pobrania kawałek zasobu.
- **synchronizacja** - uzgodnienie stanu informacji o wzajemnych zasobach pomiędzy dwoma węzłami

Dodatkowe założenia:

- Zasób to plik identyfikowany pewną nazwą, za takie same zasoby uważa się zasoby o takich samych nazwach.
- Kopie zasobów są nierozróżnialne od oryginału
- Nie istnieje możliwość zarejestrowania dwóch zasobów o takiej samej nazwie.
- Tylko właściciel zasobu może zaprzestać rozprzestrzeniania zasobu.
- Udostępnianie zasobu będzie możliwe tylko dla plików znajdujących się w katalogu roboczym. Przyświeca nam idea jak najprostszej obsługi przypadków usunięcia pliku (lub przemieszczenia).
- Pobierane zasoby będą automatycznie zapisywane w katalogu roboczym.
- {Kasowanie}/unieważnienie zasobu rozumiemy jedynie niemożliwość dalszego propagowania zasobu w sieci, a nie konieczność usunięcia go lokalnie przez wszystkie węzły sieci - raz pobrany plik jest do dyspozycji "posiadacza".
- Serwer powinien posiadać informacje o wszystkich plikach udostępnionych w sieci(z dokładnością do opóźnienia)
- Kawałek(chunk) pliku będzie posiadał 512 bajtów - z wyjątkiem ostatniego chunka pliku.

Opis funkcjonalny

Opis typu "black-box" realizacji funkcjonalności:

1. Udostępnianie nowych zasobów:

- Użytkownik chce udostępnić nowy zasób.
- Jeżeli zasób ten znajduje się w katalogu roboczym to polecenie zostanie wysłane do serwera.
- Serwer odbiera polecenie.
- Sprawdza czy zasób o takiej nazwie nie jest już dostępny w sieci
- Rejestruje zasób w strukturze zasobów udostępnianych.
- Rozsyła informacje o dostępności zasobu poprzez rozgłaszanie
- Każdy z węzłów po otrzymaniu komunikatu odnotowuje lokalnie informacje o nowym zasobie.

1. Pobieranie zasobów z sieci:

- Użytkownik żąda wypisania listy dostępnych do pobrania zasobów
- Użytkownik specyfikuje jaki zasób chce pobrać.
- Klient przekazuje zapytanie do serwera, czy taki zasób znajduje się w sieci.
- Jeśli tak to zleca jego pobranie serwerowi.
- Jeśli nie to wypisuje stosowną informację użytkownikowi.
- Serwer decyduje z którym z węzłem nawiązać połączenie w celu pobrania odpowiedniego zasobu/chunków
- Następnie deleguje wątek odpowiedzialny za pobranie odpowiedniego zasobu/chunków.

- Po zakończeniu pobierania zasobu zaktualizowana zostaje struktura posiadanych zasobów.
- Jeżeli w trakcie pobierania nie został odebrany komunikat o jego unieważnieniu to serwer rozgłasza dostępność nowego zasobu u siebie

3. Unieważnienie zasobu:

- Użytkownik specyfikuje jaki zasób chce unieważnić.
- Użytkownik jest proszony o podanie hasła do wyliczenia hasha identyfikującego właściciela
- Jeżeli hash się nie zgadza to powiadom o tym użytkownika i zaniechaj dalszych działań
- Jeżeli się zgadza to zostaje rozgłoszony odpowiedni komunikat unieważniający dany zasób
- Każdy węzeł po otrzymaniu komunikatu o rozgłoszeniu aktualizuje swoją strukturę plików

Opis protokołów komunikacyjnych

Kody komunikatów są przyznawane za według następującej konwencji: - 1xy - komunikaty informacyjne, po ich otrzymaniu węzły będą aktualizować swoją bazę wiedzy - 4xy - błędy synchroniczne(?), wynikające z zawodności UDP(np. zasób jest unieważniony a żądający tego nie wiedział) - 5xy - błędy asynchroniczne, np. usunięcie pliku przez użytkownika

Druga cyfra danych komunikatów określa jeden kontekst. Dla przykładu komunikaty x4y odpowiedzialne są za kontekst przesyłania pliku(żądanie wysłania, synchronizacja, błędy synchronizacji, błędy poczas przesyłania)

KOMUNIKATY ROZGŁASZANE(UDP z opcją gniazda SO_BROADCAST)

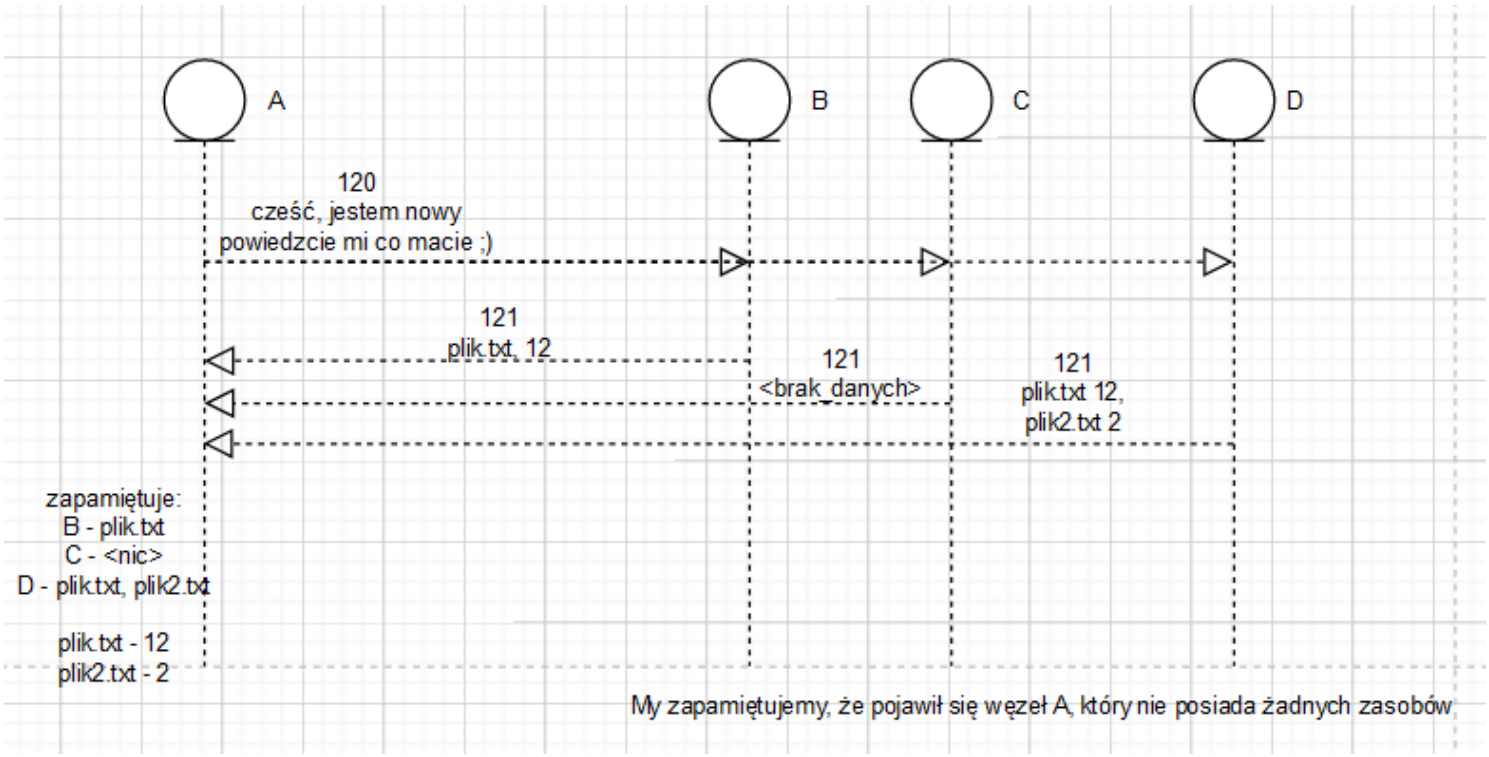
kod	kiedy wysyłany	co przesyłamy	zachowanie po otrzymaniu
100	węzeł wprowadza nowy zasób do sieci	kod, nazwa-zasobu, hash, rozmiar-w-chunkach	dodanie informacji o nowym zasobie w lokalnych strukturach
110	właściciel unieważnia zasób	kod, nazwa-zasobu	Zaktualizowanie informacji o danym zasobie. Komunikat nie wpływa na pobieranie/przesyłanie, które w momencie otrzymania miało miejsce
111	węzeł ogłasza, że nie posiada już danego zasobu(ale go nie unieważnia, np. jak użytkownik usunie dany zasób z lokalnego systemu plików)	kod, nazwa-zasobu	zaktualizowanie lokalnych struktur o otrzymaną informację
120	węzeł loguje się do sieci po raz pierwszy	kod	wysłanie do węzła rozgłaszającego stanu swojej “biblioteki” zasobów
130	węzeł kończy działanie i, co za tym idzie, przestaje udostępniać swoje zasoby	kod, lista zasobów	zaktualizowanie lokalnych struktur o otrzymaną informację

KOMUNIKATY WYSYŁANE DO KONKRETNEGO WĘZŁA

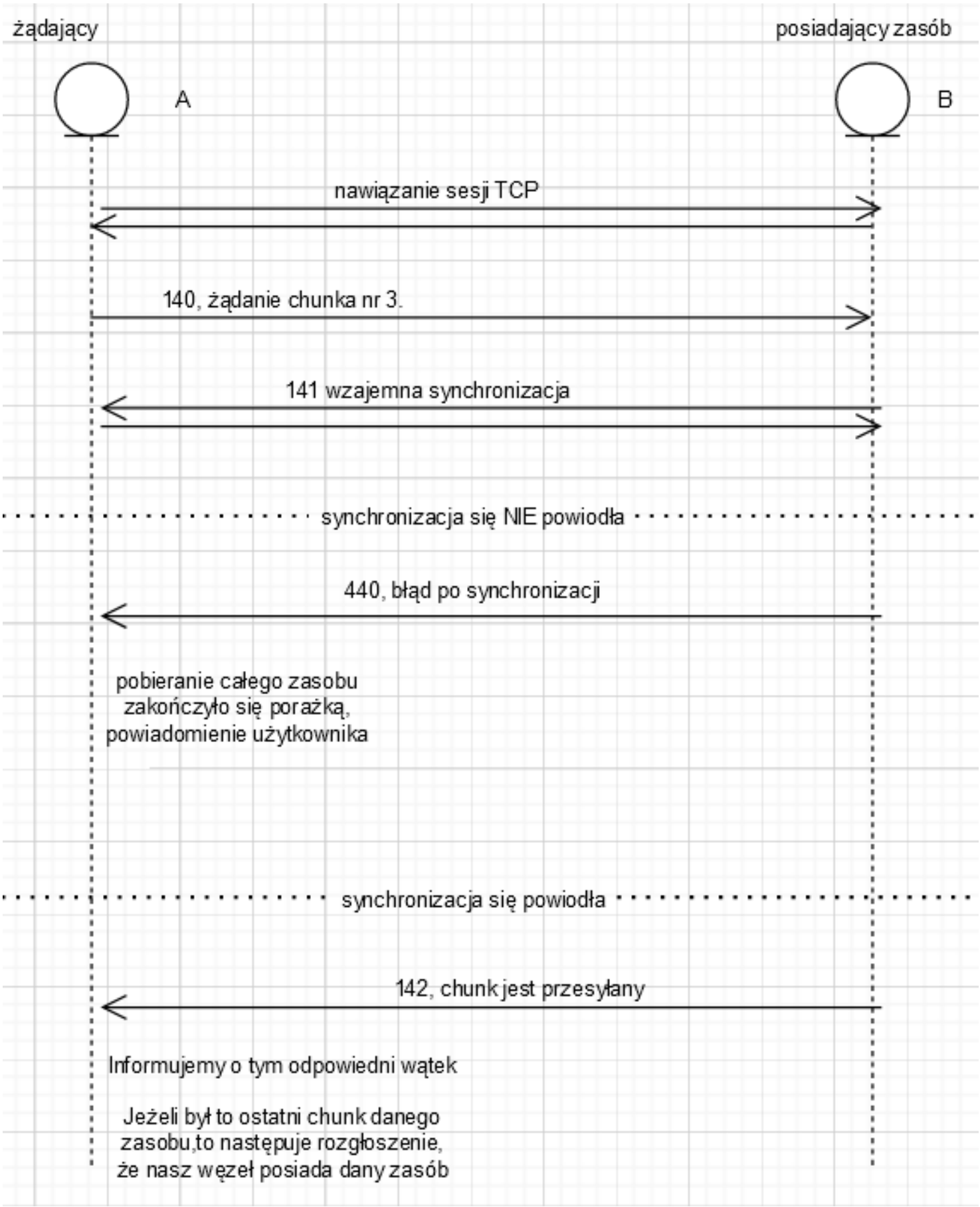
kod	kiedy wysyłamy	co przesyłamy	zachowanie po otrzymaniu	protokół transportowy
121	po otrzymaniu komunikatu o nowym użytkowniku w sieci	kod, krotki(nazwa, rozmiar)	aktualizacja struktur	**UDP!** jedyna taka sytuacja, chcemy uniknąć nawiązywania wielu połączeń z pojedynczym węzłem.

kod	kiedy wysyłamy	co przesyłamy	zachowanie po otrzymaniu	protokół transportowy
140	węzeł żąda dany chunk	kod, nazwa-zasobu, index-chunka	następuje indywidualna synchronizacja węzłów, tj. węzły sprawdzają czy informacje które posiadają zgadza. Jeżeli zakończy się sukcesem to węzeł udostępniający wysyła 142, a jeżeli nie to powinien wysyła 440	TCP
141	potrzeba indywidualnej synchronizacji	kod, krotki(nazwa, rozmiar, czy unieważniony)	aktualizacja struktur, decyzja o udostępnieniu danego węzła	TCP
142	wysyłanie danego chunka	kod, index-chunka, zawartość-zasobu	zapisujemy otrzymany fragment, czekamy na resztę. Jeżeli to wszystko to kończymy sesję TCP i informujemy wątek/i nadrzędny o sukcesie	TCP
440	nie można wysłać chunka po synchronizacji	kod	zaniechanie pobierania całego zasobu, koniec sesji TCP	TCP
540	podczas przesyłania zasobu nastąpił błąd u wysyłającego(np. zasób został usunięty)	kod	Szukamy węzła, który posiada dany zasób, wykonujemy całą pracę synchronizacyjną od nowa. Jeżeli nie znajdziemy to informujemy użytkownika że się nie udało	TCP
541	podczas odbierania zasobu nastąpił błąd(np. użytkownik się wylogował lub plik do którego był zapisywany został usunięty)	kod	zakończenie sesji TCP i zakończenie wątku/ów odpowiedzialnych	TCP

Pojawienie się nowego węzła w sieci:

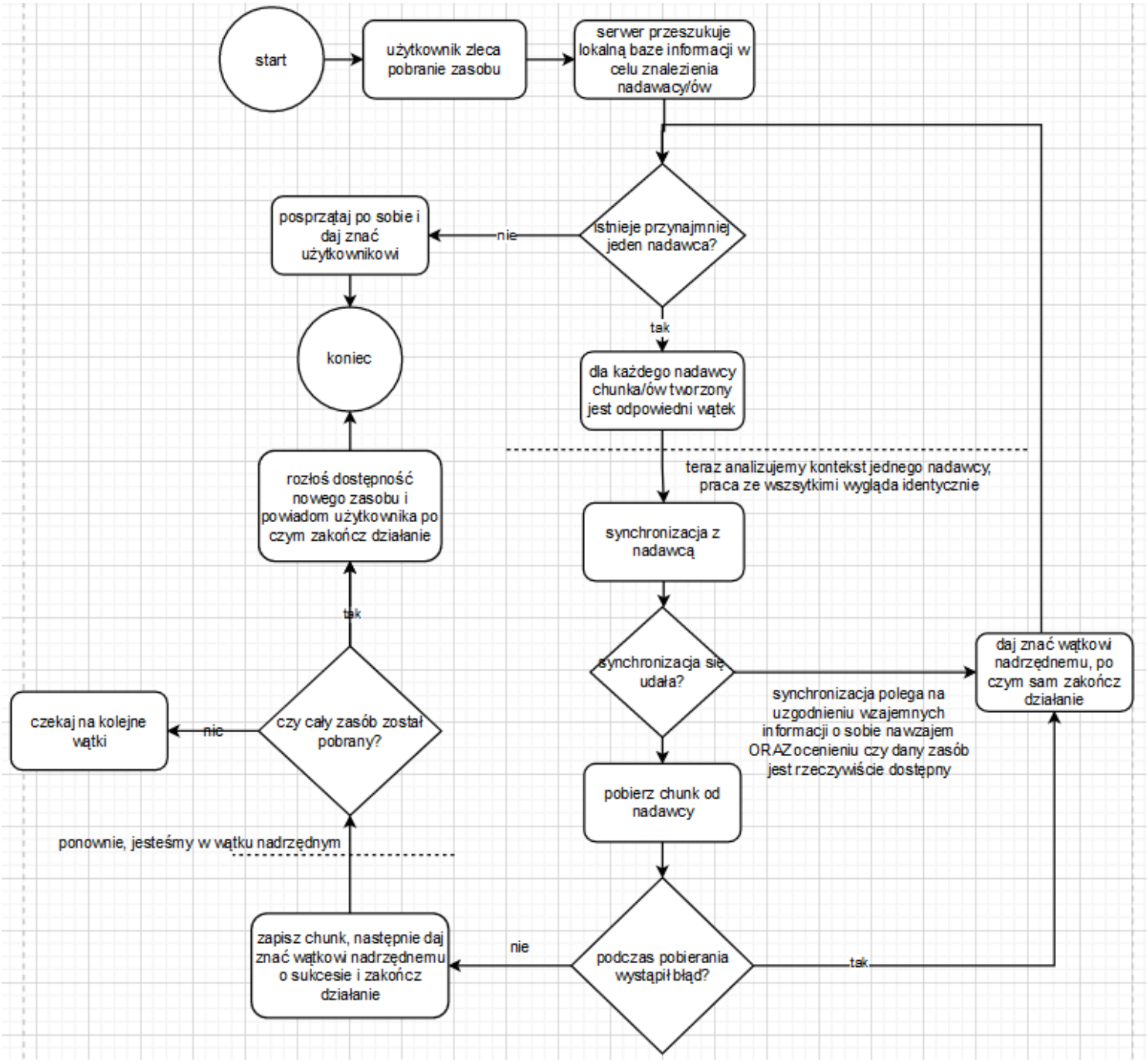


Wymiana chunka/zasobu pomiędzy dwoma węzłami:



Reszta zapytań będzie obsługiwana w analogiczny sposób.

Graf stanów węzła pobierającego zasób:



Sytuacje wyjątkowe

- zerwane połączenie z innym węzłem podczas pobierania/udostępniania - będą obsługiwane w sposób identyczny co błędy podczas przesyłania/odbierania chunka.
- w przypadku protokołu UDP nie mamy gwarancji poprawności dostarczonych danych. Naszym pomysłem na rozwiązanie tego jest dodawanie sumy kontrolnej do zapytania. W momencie otrzymania wadliwego komunikatu po prostu go odrzucamy - każda sesja TCP będzie poprzedzona synchronizacją, tj. uzgodnieniem wzajemnego stanu zasobów węzłów wymieniających.

Zgrubny podział na moduły/wątki

- klient:
  - wątek CLI
  - wątek nasłuchujący komunikatów od lokalnego serwera
- serwer:
  - wątek nasłuchujący CLI
  - wątek nasłuchujący komunikaty od innych węzłów i oddelegowujący zadania na inne wątki
  - wątki robocze:
    - wątki kompletujące dane i przesyłające je do wątków wykonawczych
    - wątki wysyłające komunikaty i przesyłające dane
    - wątki aktualizujące stan informacji
    - wątki zapisujące dane do katalogu roboczego
    - wątki odczytujące dane z katalogu roboczego

Synchronizacja wątków będzie odbywałą się za pomocą mutexów. Wątki będą się komunikować ze sobą na podstawie funkcji create_thread(function, params). Głównym wątkiem będzie wątek serwera, nasłuchujący komunikatów od innych węzłów, dane potrzebne wątkom przekazywane będą za pomocą funkcji tworzącej wątek oraz funkcji join. Komunikaty pomiędzy serwerem a klientem będą wymieniane poprzez unix sockets. Klient wysyła ustrukturyzowane żądania. Struktura żądań będzie zbliżona do protokołu komunikacji między węzłami Serwer wysyła komunikaty o (nie)powodzeniu oraz przesyła informacje o stanie sieci.

Zarys implementacji

Projekt realizować będziemy w języku C/C++. Nie planujemy używać innych bibliotek niż standardowe. Używane przez nas IDE to CLion. W celu synchronizacji i koordynacji projektu użyjemy GitHub. Testowanie wykonamy za pomocą dwóch procesów wysyłających do siebie testowe dane i wymieniają komunikaty. Dla każdego testu jest osobny scenariusz i testuje on wybraną funkcjonalność: czy wyświetla się lista, czy przesłał się przykładowy plik, czy dany plik się unieważnił.

Struktury danych

- mapa informacji o zasobach(klucz:nazwa - wartość:metadane)
- wektor zasobów dostępnych lokalnie(gotowych do udostępnienia)
- wektor zasobów dostępnych w sieci(gotowych do pobrania)
- mapa węzłów w sieci(klucz:socket - wartość:informacje-o-wezle)
- krotka określająca metadane zasodu: <rozmiar_w_chunkach; hash_do_unieważnienia; czy_unieważniony; czy_dostępny_lokalnie>