**Treść zadania**

Napisać program obsługujący prosty kanał P2P w oparciu o protokół BitTorrent.

* Należy wykorzystać protokoły IPv6 oraz TCP.
* Zasób to plik identyfikowany pewną nazwą, za takie same zasoby uważa się zasoby o takich samych nazwach. Początkowo dany zasób znajduje się w jednym węźle sieci, następnie może być propagowany do innych węzłów w ramach inicjowanego przez użytkownika ręcznie transferu – 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).
* System ma informować o posiadanych lokalnie (tj. w danym węźle) zasobach i umożliwiać ich pobranie.
* Program powinien umożliwiać współbieżne: o wprowadzanie przez użytkownika (poprzez interfejs tekstowy) nowych zasobów oraz poleceń pobrania nazwanego zasobu ze zdalnego węzła o pobieranie zasobów (także kilku jednocześnie) o 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, zawsze występuje jeden oryginał i potencjalnie wiele kopii. Można założyć, że oryginał nie zmienia się nigdy. Istnienie kopii jest rozgłaszane tak samo jak oryginału. Oryginał nigdy nie zmienia swojej lokalizacji. Właściciel oryginału może go unieważnić wysyłając odpowiedni komunikat rozgłaszany. Szczegółowe zachowanie się posiadaczy lokalnych kopii w przypadku unieważnienia oryginału pozostaje do decyzji implementacyjnej i powinno być szczegółowo przeanalizowane i opisane w projekcie wstępnym.
* Nie jest wymagane implementowanie szyfrowania.
* Interfejs użytkownika nie musi być szczegółowy i mocno rozbudowany, powinien być prosty i przejrzysty, najważniejszy w implementacji jest protokół.

**Krótki opis funkcjonalny**  
 Program został napisany przy wykorzystaniu protokołu IPv6 oraz TCP, jest on obsługiwany przy użyciu terminala tekstowego.

Aplikacja umożliwia:

* udostępnienie przez użytkownika zasobu (oryginału)
* polecenie pobrania nazwanego zasobu ze zdalnego węzła
* pobranie zasobu ze zdalnego węzła (utworzenie lokalnej kopii zasobu)
* rozsyłanie informacji o lokalnie posiadanych zasobach.

Użytkownik ma możliwość unieważnić swój zasób (w przypadku, gdy posiada oryginał zasobu). W przypadku unieważnienia przez użytkownika oryginalnego zasobu system losowo wybiera użytkownika, którego kopia zasobu od tej pory będzie uznawana za oryginał.

Rozgłaszania posiadania zasobu (oryginału i kopii) oraz unieważnienie zasoby będzie rozgłaszane odpowiednim komunikatem.

Zasób jest identyfikowany poprzez nazwę, dlatego próba dodania do sieci zasobu o już istniejącej nazwie będzie sygnalizowana błędem.

**Pełen opis funkcjonalny**

Program obsługuje prosty protokół komunikacyjny P2P w oparciu o protokół Bittorrent. Wykorzystuje o tego protokół TCP i IPv6.

Użytkownikami systemu są osoby chcące pobierać zasoby znajdujące się w innych węzłach sieci oraz udostępniać swoje pliki do pobrania.

Aplikacja umożliwia:

* Rozsyłanie informacji o lokalnie posiadanych zasobach,

Użytkownik za pomocą interfejsu wybiera opcję “Share File”. W odpowiedzi program pyta o ścieżkę do lokalnego zasobu, który użytkownik chce udostępnić. Ścieżka ta jest następnie przekazywana jako argument do funkcji *sock\_PostFileToTracker()*.

Klient otrzymuje z kolejki wiadomość “1”, potem oczekuje wiadomości z rozmiarem pliku .torrent oraz nazwy udostępnianego zasobu. Następnie wysyła trackerowi wiadomość “1”, tracker odpowiada i odbiera plik .torrent od klienta. Tracker przetwarza otrzymany plik, informacje wyłuskane z niego dodaje do listy zasobów możliwych do pobrania przez innych użytkowników. Jeżeli zasób o danej nazwie znajduje się już w systemie, Tracker wysyła Klientowi informację o błędzie, jeżeli nie - informację o powodzeniu.

* Pobranie zasobu ze zdalnego węzła,

Użytkownik za pomocą interfejsu wybiera opcję “Download File”, w odpowiedzi program pyta o nazwę zasobu, który użytkownik chce pobrać. Następnie w module interfejsu wołana jest funkcja *downloadFile()*.

Klient otrzymuje z kolejki wiadomość “2”, w następnej wiadomości oczekuje nazwy zasobu, który ma być pobrany. Funkcja *downloadFile()* usypia wtedy wątek interfejsu i daje modułowi klienckiemu chwilę na komunikację z trackerem. Klient następnie wysyła trackerowi wiadomość o treści “2”, otrzymuje wiadomość zwrotną, po czym wysyła nazwę zasobu do pobrania. Tracker przeszukuje listę posiadanych zasobów, po czym zwraca klientowi listę hashy części zasobu oraz listę seed’ów posiadających ten zasób.

Otrzymane dane klient najpierw parsuje, to znaczy wyłuskuje z nich same adresy IP (analogicznie do funkcji getline() w C++). Następnie przekazuje te adresy na kolejkę, z której po chwili zaczyna czytać wybudzona funkcja *downloadFile()*. Próbuje ona nawiązać połączenie z każdym adresem IP odczytanym z kolejki, aż któreś z połączeń nie zakończy się powodzeniem (po każdej próbie moduł kliencki zapisuje na kolejce informację o statusie połączenia).

Klient łączy się z danym peerem i wysyła mu nazwę zasobu, który chciałby pobrać. Inny peer, w odpowiedzi przesyła mu ten zasób. Odebrane dane klient zapisuje do pliku o nazwie wcześniej mu podanej.

Po zakończeniu pobierania, klient wysyła trackerowi wiadomość “4”, tracker odpowiada, a następnie oczekuje wiadomości z nazwą pliku. Po jej otrzymaniu przeszukuje listę zasobów i dodaje adres IP użytkownika do listy seed’ów posiadających dany zasób.

* Pobranie posiadanego lokalnie zasobu przez inne węzły,

Socket nasłuchujący klienta nasłuchuje i czeka na połączenie od innego peera. W momencie pojawienia się połączenia, klient czeka na wiadomość z nazwą zasobu, który ma udostępnić. Po jej odebraniu, wysyła odpowiednie dane. Po ich przesłaniu, połączenie jest zamykane i socket jest ponownie uruchamiany.

* Rozsyłanie informacji o unieważnieniu posiadanego zasobu.

Użytkownik za pomocą interfejsu wybiera opcję “Delete File”, w odpowiedzi program pyta o nazwę zasobu, który użytkownik chce "usunąć”. Wywoływana jest wtedy funkcja *sock\_DeleteFileFromTracker().*

Klient otrzymuje z kolejki wiadomość “3”. Po tej wiadomości odbiera kolejną - z nazwą zasobu. Następnie wysyła wiadomość o treści “3” do trackera. Po niej - nazwę zasobu. Tracker przeszukuje listę zasobów w systemie i usuwa z listy seedów, adres IP klienta, jeżeli nie ma takiego zasobu, albo adres IP klienta nie widnieje na liście seed’ów, wysyła klientowi odpowiednią wiadomość, którą klient wyświetla użytkownikowi na ekran.

Każda z powyższych funkcji realizowana będzie przy użyciu modułów aplikacji:

* Obsługa Metadanych
* Usługa P2P
* Interfejs

**Podział na moduły i struktura komunikacji**

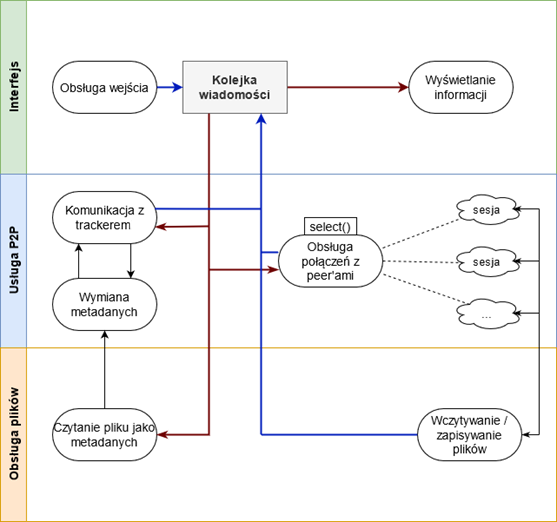
Do realizacji współbieżności wykorzystana jest wielowątkowość. Każdy z modułów programu działa w oddzielnym wątku w celu zapewnienia asynchroniczności wykonywania operacji, natomiast komunikacja pomiędzy modułami/wątkami odbywa się przez prostą kolejkę wiadomości.

W skład modułów wchodzą:

* Interfejs
  + ***Obsługa wejścia*** – nawigacja w menu, wpisywanie nazw plików etc.
  + ***Wypisywanie na ekran*** – wyświetlanie menu, pasków postępu, komunikatów
* Usługa P2P
  + ***Komunikacja z trackerem*** *–* zbieranie informacji o dostępnych zasobach
  + ***Ustanawianie połączeń z peer'ami*** *–* przy pomocy socketów oraz funkcji select(), komunikacja z innymi peer’ami, zarządzanie pobieraniem
* Obsługa Metadanych
  + ***Czytanie pliku jako metadanych*** *–* tworzenie plików .torrent
  + ***Wczytywanie/zapisywanie do pliku*** *–* przy pomocy zwykłego fstreama, współpracujący z *ustanawianiem połączeń*

Jako główny sposób komunikacji międzymodułowej wykorzystana jest wcześniej wspomniana **kolejka wiadomości**. Celem tego rozwiązania jest utworzenie kanału dla komunikatów, których przetworzenie nie jest priorytetem. Jest to prosta struktura, która przechowuje wiadomości w postaci ciągów znaków.

Każda wiadomość posiada informacje o odbiorcy (ID modułu), typie oraz treści. Moduły okresowo skanuję ostatnie puste pole w celu sprawdzenia, czy nadeszła dla nich wiadomość (innymi słowy polling, ale wykonywany sporadycznie by nie zmniejszać wydajności aplikacji).

**Diagram modułów**  


**Najważniejsze rozwiązania funkcjonalne**

**Moduł Metadanych**

Moduł Metadanych wyciąga informacje dotyczące pliku, na podstawie którego ma być sporządzony plik torrentowy. W folderze sharedFiles powinien się znajdować plik oraz folder o takiej samej nazwie (bez rozszerzenia) w którym znajduje się sfragmentowany plik – kolejne fragmenty nazywają się jak liczby początkowe np. 1, 2,3. W folderze metadataFiles będzie tworzony plik o tej samej nazwie co plik tworzący. Znajdujemy rozmiar pierwotnego pliku. Wynikowy plik składa się z następujących fragmentów:

1. Announce – URL trackera
2. Info – zawiera informacje dotyczące pliku
   1. Length - długość pliku w bajtach
   2. Name – proponowana nazwa pliku
   3. Piece length - długość pojedynczego kawała pliku (ostatni kawałek może być krótszy)
   4. Pieces – lista zawierająca hash MD5 każdego kawałka

Całość pliku torentowego jest z

**Moduł usługi P2P**

**Tracker**

Moduł trackera przechowuje informacje o lokalizacjach części poszczególnych zasobów w systemie oraz pełni funkcję serwera TCP, przy połączeniach z poszczególnymi klientami.

Informacje o plikach tracker przechowuje w postaci listy jednokierunkowej, w której węzłami są poszczególne pliki. Każdy węzeł posiada listę hashy części, z których składa się plik oraz listę tzw. seed'ów posiadających lokalnie dany zasób.

Tracker uruchamiany jest oddzielnie, przed wystartowaniem reszty modułów programu i działa przez cały czas działania systemu. Po uruchomieniu inicjalizuje listę jednokierunkową do przechowywania plików, tworzy gniazdo do odbierania połączeń, przypisuje mu nazwę, a następnie nasłuchuje połączeń od klientów. W momencie pojawienia się żądania od klienta, tworzy nowy wątek do jego obsługi, a następnie go odłącza.

Klient może łączyć się z Trackerem za pomocą czerech opcji:

* “1” - taka wiadomość oznacza chęć udostępnienia informacji o posiadanym zasobie. W odpowiedzi Tracker prosi o przesłanie pliku .torrent zasobu, a następnie go odbiera i zapisuje w lokalnym folderze. Plik ten następnie jest otwierany, a odpowiednie funkcje wyłuskują z niego nazwę zasobu oraz hashy jej części. Wszystkie te informacje dodawane są do jednokierunkowej listy w programie.

Jeżeli dany zasób widnieje już na liście, adres IP klienta, z którego przychodzi żądanie dodawany jest na listę seed’ów posiadających lokalnie dany zasób. (Klient zostaje o tym poinformowany odpowiednią wiadomością.)

Jeżeli dany zasób nie widnieje na liście, dodawany jest nowy węzeł ze wszystkimi informacjami pobranymi z piku .torrent, a adres IP klienta, z którego przychodzi żądanie, dodawany jest na listę seed’ów jako właściciel oryginału. (Klient zostaje o tym poinformowany odpowiednią wiadomością.)

Na koniec skocket klienta zostaje zamknięty, a wątek go obsługujący, dzięki wcześniejszemu odłączeniu automatycznie zwraca zasoby do sytemu.

* “2” - taka wiadomość oznacza chęć pobrania listy hashy części określonego zasobu oraz listy seed’ów posiadających lokalnie ten zasób. W odpowiedzi Tracker prosi o podanie nazwy zasobu. Następnie lista przeszukiwana jest w celu jego odnalezienia.

Jeżeli zasób o danej nazwie nie zostanie odnaleziony na liście, klient zostanie o tym powiadomiony odpowiednią wiadomością.

Jeżeli zasób o danej nazwie zostanie odnaleziony na liście, Tracker lokalnie tworzy plik, do którego zapisuje listę hashy części oraz listę seed’ów danego zasobu, a następnie plik ten wysyła klientowi.

Na koniec socket klienta zostaje zamknięty, a wątek go obsługujący, dzięki wcześniejszemu odłączeniu automatycznie zwraca zasoby do sytemu.

* “3” - taka wiadomość oznacza chęć usunięcia informacji o posiadanym przez klienta zasobie. W odpowiedzi tracker prosi o nazwę zasobu do usunięcia. Następnie lista przeszukiwana jest w celu jego odnalezienia.

Jeżeli zasób o danej nazwie nie zostanie odnaleziony na liście, klient zostanie o tym powiadomiony odpowiednią wiadomością.

Jeżeli zasób o danej nazwie zostanie odnaleziony na liście, ale adres klienta, od którego przychodzi żądanie nie zostanie odnaleziony na liście seed’ów posiadających ten zasób, klient zostanie o tym powiadomiony odpowiednią wiadomością.

Jeżeli zasób o danej nazwie zostanie odnaleziony na liście i adres klienta, od którego przychodzi żądanie zostanie odnaleziony na liście seed’ów tego zasobu, adres klienta zostaje usunięty z tej listy. Jeżeli był to seed oznaczony jako posiadacz oryginału, nowym posiadaczem oryginału zostaje pierwszy seed z listy. Jeżeli usuwany seed był jedynym posiadaczem zasobu, wraz z usunięciem jego adresu z listy, usuwane są z systemu wszystkie informacje o zasobie.

Na koniec klient jest informowany o pomyślnym usunięciu klienta z listy seedów, socket klienta zostaje zamknięty, a wątek go obsługujący, dzięki wcześniejszemu odłączeniu automatycznie zwraca zasoby do sytemu.

* “4” - taka wiadomość oznacza chęć poinformowania Trackera o zakończeniu pobierania przez klienta zasobu od innych peer’ów. W odpowiedzi Tracker prosi o nazwę pobranego zasobu. Następnie lista przeszukiwana jest w celu jego odnalezienia.

Jeżeli zasób o danej nazwie nie zostanie odnaleziony na liście, klient zostanie o tym powiadomiony odpowiednią wiadomością.

Jeżeli zasób o danej nazwie zostanie odnaleziony na liście, adres IP klienta zostaje dodany na listę seed’ów posiadających lokalnie dany zasób. Klient zostanie o tym powiadomiony odpowiednią wiadomością.

Na koniec socket klienta zostaje zamknięty, a wątek go obsługujący, dzięki wcześniejszemu odłączeniu automatycznie zwraca zasoby do sytemu.

Jeżeli klient przy pierwszym żądaniu wyśle wiadomość, która nie reprezentuje żadnej z powyższych opcji, zostanie o tym poinformowany odpowiednią wiadomością, socket klienta zostanie zamknięty, a wątek go obsługujący, dzięki wcześniejszemu odłączeniu automatycznie zwraca zasoby do sytemu.

**Klient**

Moduł klienta został podzielony na dwie zasadnicze części.

Pierwsza część wykonuje wyłącznie jedną funkcję tzn. nasłuchuje i czeka na ewentualne połączenie od innego klienta. W momencie pojawienia się połączenia, następuje próba jego akceptacji. Jeśli akceptacja się powiedzie klient czeka na wiadomość z nazwą pliku, który inny peer chce od niego pobrać. Po odebraniu wiadomości w module następuje próba otwarcia pliku i gdy ta się powiedzie przesył danych. Po przesłaniu danych połączenie jest zamykane. Zamykany jest również socket, który został użyty do komunikacji między peerami, a moduł wraca do nasłuchu przy użyciu nowo otwartego socketu.

Druga część modułu przetwarza wiadomości, które odczytuje z kolejki. W zależności od otrzymanej wiadomości moduł przechodzi do odpowiedniej opcji:

* “0” - wiadomość ta oznacza, że kończymy pracę i moduł powinien zostać zamknięty.
* “1” - wiadomość oznacza chęć wysłania pliku torrent do modułu trackera. W kolejnej wiadomości moduł oczekuje rozmiaru pliku, który ma zostać przesłany i nazwy pliku. Następnie podejmowana jest próba otwarcia pliku. Jeśli plik został otworzony następuje wymiana odpowiednich wiadomości między klientem a trackerem, a na końcu przesłanie pliku torrent. Jeśli przesłanie się powiodło, na konsoli wyświetla się odpowiedni komunikat. Socket zostaje zamknięty, a moduł przechodzi do obsługi następnych wiadomości.
* “2” - wiadomość oznacza chęć pobrania od trackera listy seedów, które posiadają dany plik. Po tej wiadomości moduł oczekuje kolejnej wiadomości z nazwą pliku, dla którego ma zostać pobrana lista. Następuje wymiana odpowiednich wiadomości, a następnie pobranie od trackera odpowiedniego pliku. Dane otrzymane od trackera są wysyłane na kolejkę służącą do zapisu informacji.
* “3” - wiadomość sygnalizuje chęć wysłania do trackera informacji o usunięciu pliku. Po tej wiadomości powinna zostać odebrana wiadomość z nazwą pliku, który został usunięty. Moduł przesyła taką informację do trackera i jeśli wszystko się udało, przechodzi do dalszej obsługi wiadomości z kolejki.
* “4” - wiadomość oznacza chęć pobrania pliku od innego peeru. W kolejnej wiadomości powinien zostać wysłany adres, z którego plik ma zostać pobrany, a w następnej informacja o nazwie pliku. Moduł próbuje się połączyć z odpowiednim peerem i jeśli uda mu się nawiązać połączenie wysyła nazwę pliku, który chciałby otrzymać. Inny peer przesyła mu plik (opisana wyżej część nasłuchująca) i po zakończeniu przesyłania zamyka połączenie. Jeśli moduł otrzymał plik zapisuje go lokalnie oraz wysyła do trackera informację o tym, że posiada dany plik. Po zakończonych operacjach klient zamyka otwarte sockety i przechodzi do dalszej obsługi kolejki.

**Kolejka wiadomości**

Sama w sobie kolejka jest de facto tablicą o stałym rozmiarze przekształconą w **bufor cykliczny** dzięki zastosowaniu liczników zapisów oraz odczytów.

Każdy z modułów (metadanych, interfejsu oraz socketów - w sumie 3) ma swoją własną kolejkę mieszczącą 32 wiadomości, każda po 64 bajty / znaki. Wykorzystując zaledwie 2KB pamięci w prosty sposób realizowana jest **asynchroniczna komunikacja między wątkami**. Warto podkreślić, iż wartości te dobrane zostały w celu prostoty obsługi kolejki oraz ułatwienia programowania – zajmowana pamięć mogłaby być o wiele mniejsza. Ze względu na wykorzystanie wielowątkowości, kolejka wiadomości jest deklarowana globalnie, przez co każdy wątek ma do niej łatwy dostęp.

Dla każdego modułu zdefiniowane jest jego ID, które następnie używane jest do wołania funkcji obsługujących kolejkę. Wykorzystywane są dwie funkcje – **writemsg**() oraz **readmsg**(). Każda akceptuje jako parametry ID kolejki, na której powinna operować oraz wskaźnik na char[]. Zawartość wiadomości w kolejce zostanie przekopiowana do wskazanego bufora, po czym wiadomość zostanie usunięta z kolejki. Korzystanie z dowolnej funkcji jest bezpieczne i nie powoduje błędów synchronizacji, ponieważ obie funkcje stosują wewnętrznie **mutexy**. Ze względu na szybkość pisania oraz czytania z kolejki, chwilowe blokady nie spowalniają w znaczącym stopniu programu.

**Szczegółowy opis interfejsu użytkownika**

Inerface umożliwia wykonanie trzech akcji: Pobranie pliku z trackera, udostępnienie pliku trackerowi, usunięcie pliku z trackera. Gdy chcemy pobrać plik z trackera wybieramy za pomocą klawisy a i d opcję Download File po czym klikamy enter. Terminal zapyta nas o nazwę pliku jaki zamierzamy pobrać a następnie pyta trackera o adresy IP maszyn, które ten plik udostępniają i próbuje go od nich pobrać. Możesz udostępnić plik innym użytkownikom, należy wybrać opcję Share File. Terminal zapyta o ścieżkę do pliku, który chcesz udostępnić a następnie zkomunikuje się z trackerem udostępniając mu odpowiednie dane. Gdy chcesz przestać udostępniać dany plik trackerowi czyli niejako usunąć ten plik należy wybrać opcję Delete File. Terminal ponownie spyta o nazwę pliku, który chcesz usunąć a następnie usunie go z Trackera.

**Pliki konfiguracyjne**

* Makefile – plik znajduje się w module trackera i służy do skompilowania i zbudowania modułu trackera. Plik zawiera wszystkie komendy, które są potrzebne by poprawnie utworzyć plik wykonywalny. Plik wykonywalny tworzony jest po wywołaniu komendy “make”. Plik zawiera również konfigurację wspierającą usuwanie plików tymczasowych, aby wywołać takie działanie, należy wykonać polecenie “make clean”.
* CMakeLists.txt - plik wspiera budowanie modułu klienta i interfejsu użytkownika. Plik zawiera odwołania do wszystkich plików oraz bibliotek, które umożliwiają poprawne skompilowanie i uruchomienie powyższych modułów.

**Wykorzystane narzędzia**

Projekt został zrealizowany w języku C z wykorzystaniem standardowego API gniazd BSD/Unix. Do obsługi wielowątkowości wykorzystaliśmy bibliotekę thread. Kod tworzyliśmy przy wykorzystaniu programów Visual Studio Code i Clion. Do zarządzania projektem zastał użyty git i gitlab.

**TUTAJ PROSZĘ NIECH KAŻDY KTO UŻYWAŁ CZEGOŚ JESZCZE, NIECH O TYM NAPISZE (METADANE, TRACKER, INTERFEJS LUB COKOLWIEK INNEGO)**

- Ogólny opis: Ola + każdy sam uzupełnia, jeżeli jakieś specjalne narzędzia były wykorzystywane w jego module

**Opis testów i wyniki testowania**

Testowanie modułu klienta z modułem trackera polegało w ogólności na sprawdzaniu komunikacji i zachowania poszczególnych modułów w przypadku wysłania odpowiednich danych. Sprawdzone zostały reakcje na wysłanie zarówno poprawnych jak i niepoprawnych danych. Odpowiedzi trackera były poprawne, w przypadku wysłania błędnych danych tracker wysyła odpowiednią wiadomość, która jest wyświetlana na konsoli klienta. W przypadku wysłania poprawnych danych poza konkretnym działaniem przesyłane są również odpowiednie wiadomości sygnalizujące poprawność danych.

Testowanie modułu metadanych polegało na sprawdzaniu czy fragmenty zbencodowanego pliku są poprawne. Używaliśmy biblioteki openssl do generowania kodu MD5, więc należało też sprawdzić, czy kod zawiera 32 znaki oraz czy kod zmienia się, gdy zmienimy plik.