Zadanie 1

Zadanie polega na napisaniu programu binarnego `testhttp_raw` zawierającego oprogramowanie klienta oraz skryptu `testhttp`. Zadaniem tych programów jest zapewnienie narzędzia do prostego testowania stron WWW. Używać będziemy protokołu warstwy aplikacji HTTP w wersji 1.1, protokołu warstwy transportu TCP i warstwy sieci IPv4.

1. Program klienta

Klient po zinterpretowaniu argumentów wiersza poleceń łączy się ze wskazanym adresem i portem, wysyła do serwera HTTP żądanie podania wskazanej strony, odbiera odpowiedź od serwera, analizuje wynik i podaje raport.

Opis komunikacji

Techniczny opis formatu żądań i odpowiedzi HTTP znajduje się w dokumencie RFC7230. Klient ma połączyć się ze wskazanym adresem i portem, a następnie wysłać odpowiednie żądanie HTTP. Adres i port połączenia nie muszą się zgadzać z adresem i portem wskazanym w adresie testowanej strony. Żądanie HTTP ma zawierać odpowiednio umieszczone w swojej treści:

- adres serwera, z którego będziemy ściągać zasób;
- wskazanie zasobu, który ma zostać pobrany;
- określone przez parametry w wierszu poleceń ciasteczka;
- wskazanie, że po zakończeniu przesyłania zasobu połączenie ma zostać przerwane.

Jeśli odpowiedź serwera jest inna niż `200 OK` (np. `202 Accepted`) klient ma podać raport w postaci zawartości wiersza statusu uzyskanej odpowiedzi. Jeśli odpowiedź serwera jest `200 OK`, raport ma składać się z dwóch części: zestawienia ciasteczek oraz rzeczywistej długości przesyłanego zasobu. Techniczny opis formatu pola nagłówkowego `Set-Cookie` znajduje się w dokumencie RFC2109. Należy tutaj pamiętać, że jedna odpowiedź HTTP może zawierać wiele pól `Set-Cookie`. Należy przyjąć, że jedno pole nagłówkowe `Set-Cookie` ustawia jedno ciasteczko. Jeśli w jednym polu ustawiane jest wiele ciasteczek należy ciasteczka poza pierwszym pominąć. Jeśli implementacja przyjmuje ograniczenia na liczbę przyjmowanych ciasteczek i ich długość, to ograniczenia te powinny zostać dobrane zgodnie z założeniami przyjętymi w standardach HTTP dla rozwiązań ogólnego przeznaczenia. Dodatkowo przy liczeniu długości przesyłanego zasobu należy uwzględnić możliwość, że zasób był wysłany w częściach (kodowanie przesyłowe *chunked*).

Opis wypisywanego raportu

Jak wspomnieliśmy, raport ma składać się z dwóch następujących bezpośrednio jedna po drugiej części: zestawienia ciasteczek oraz rzeczywistej długości przesyłanego zasobu. Zestawienie ciasteczek ma składać się z liczby linii równej liczbie ciasteczek. Każde ciasteczko ma być wypisane w osobnym wierszu w formacie *klucz=wartość* (bez spacji naokoło znaku =). Rzeczywista długość przesyłanego zasobu ma składać się z jednego wiersza postaci *Dlugosc zasobu:* , gdzie to zapisana w systemie dziesiętnym długość zasobu. Po znaku *:* ma znajdować się jedna spacja.

Opis wiersza poleceń

Wywołanie programu klienta ma ogólną postać:

```
testhttp_raw <adres połączenia>:<port> <plik ciasteczek> <testowany adres http>
```

gdzie

- `<adres połączenia>` to adres, z którym klient ma się połączyć;
- `<port>` to numer portu, z którym klient ma się podłączyć;
- `<plik ciasteczek>` to plik zawierający ciasteczka wysyłane do serwera HTTP, format opisany poniżej;
- `<testowany adres http>` to adres strony, która ma być zaserwowana przez serwer HTTP.

W pliku zawierającym ciasteczka każde ciasteczko jest zawarte w osobnym wierszu w formacie *klucz=wartość* (bez spacji naokoło znaku =).

Przykładowe wywołania:

```
./testhttp_raw www.mimuw.edu.pl:80 ciasteczka.txt http://www.mimuw.edu.pl/

./testhttp_raw 127.0.0.1:30303 ciasteczka.txt https://www.mimuw.edu.pl/
```

Należy przyjąć, że `<testowany adres http>` po prefiksie `http://` lub `https://` zawiera uproszczony format pierwszego pola w postaci `<adres serwera>[:<numer portu>]`, przy czym ujętą w nawiasy kwadratowe część `:<numer portu>` można pominąć, przykładowe adresy to `http://www.mimuw.edu.pl/plik`, `http://www.mimuw.edu.pl:8080/plik`, `https://www.mimuw.edu.pl/plik`.

2. Skrypt

Wywołanie skryptu ma mieć ogólną postać:

```
./testhttp &lt;plik ciasteczek&gt; &lt;testowany adres http&gt;
```

Zadaniem skryptu jest zapewnienie programowi `testhttp_raw` możliwości testowania stron HTTPS. Skrypt ten ma rozpoznawać, czy `<testowany adres http>` jest adresem czystego HTTP, czy HTTPS. W tym pierwszym przypadku ma przekształcać wywołanie skryptu na wywołanie programu `testhttp_raw`. W tym drugim przypadku ma za pomocą programu `stunnel` stworzyć tunel, z miejscem wejściowym na adresie pętli zwrotnej i wybranym porcie, a następnie wywołać program `testhttp_raw` tak, aby klient połączył się z tym adresem i portem oraz testował żądanie na adres `<testowany adres http>`.

Program `stunnel` działa w ten sposób, że na podstawie danych konfiguracyjnych (mogą być podane w pliku lub przez standardowe wejście) łączy się on z odległym serwerem *SO* pod wskazanym adresem i portem, tworząc połączenie, poprzez które przesyłane są protokołem SSL/TLS zaszyfrowane dane. Połączenie to nazywane jest *tunelem*. Dane do przesyłania program `stunnel` uzyskuje w sposób następujący. Tworzy on na wskazanym adresie lokalnym i porcie serwer. Jeśli jakiś klient połączy się z tym lokalnym adresem i portem, to dane przekazywane tym połączeniem będą trafiały w postaci zaszyfrowanej do serwera *SO*. Na przykład dla pliku konfiguracyjnego o zawartości

```
[service]
client = yes
accept = 127.0.0.1:3333
connect = www.mimuw.edu.pl:443
```

połączenie na adres `127.0.0.1` z portem `3333` spowoduje, że dane tam przesyłane trafią połączeniem SSL/TLS do serwera pod adresem `www.mimuw.edu.pl` na porcie `443`.

Chcielibyśmy jeszcze zwrócić uwagę Państwa na to, że w niektórych konfiguracjach program `stunnel` zapisuje do wskazanego pliku swój *pid*. Sposobem tworzenia tego pliku można sterować poprzez zawartość pliku konfiguracyjnego.

3. Dodatkowe wymagania

Katalog z rozwiązaniem powinien zawierać plik źródłowy `testhttp_raw.c` oraz plik `Makefile` zapewniający automatyczną kompilację i linkowanie. Można też umieścić tam inne pliki potrzebne do skompilowania i uruchomienia programu, jeśli to jest konieczne. Dodatkowo w katalogu powinien znajdować się wykonywalny skrypt `testhttp`.

Jeśli skrypt `testhttp` tworzy pliki tymczasowe lub inne obiekty w systemie operacyjnym, muszą one być kasowane przed zakończeniem działania skryptu, także w wyniku przerwania sygnałem.

Nie wolno używać dodatkowych bibliotek, czyli innych niż standardowa biblioteka C.

Każdy komunikat błędu (np. o niemożliwości połączenia się ze wskazanym serwerem HTTP) musi być wypisywany na standardowy strumień błędów i zaczynać się od prefiksu `ERROR:`.

4. Termin i ocenianie

Termin

Termin oddania zadania: piątek 24 kwietnia 2020 godzina 23:59. Za każde rozpoczęte 12 godzin spóźnienia odejmujemy 1,5 pkt.

Format wysyłanego rozwiązania

Jako rozwiązanie należy wysłać poprzez moodle plik `ab123456.tgz`, gdzie `ab123456` to login na students. W wysłanym pliku ma być katalog `ab123456` zawierający pliki źródłowe i plik `Makefile`. Napisane programy powinny dawać się skompilować i uruchomić na students (ale nie oznacza to, że będą one na students przez nas testowane).

Nie wolno umieszczać tam plików binarnych ani pośrednich powstających podczas kompilacji. W wyniku wykonania polecenia `make` ma powstać plik wykonywalny: `testhttp_raw`.

Ponadto `Makefile` powinien obsługiwać cel `clean`, który po wywołaniu kasuje wszystkie pliki powstałe podczas kompilacji.

Punktacja

- Do uzyskania jest maksymalnie 6 punktów.
- Za prawidłową realizację funkcjonalności mierzoną liczbą testów, przez które poprawnie przeszła implementacja, do 5 punktów.
- Za kulturę pisania kodu (czytelność kodu, komentarze, sensowność rozwiązań itp.) do 1 punktu.
- Niespełnienie jakiegokolwiek formalnego wymagania z sekcji *Format wysłanego rozwiązania* oraz *Dodatkowe wymagania* oznacza odjęcie 1 punktu.