Zadanie programistyczne nr 1
z Sieci komputerowych

1 Opis zadania

Napisz program `traceroute`, wyświetlający adresy IP routerów na ścieżce do docelowego adresu IP. Program powinien działać w trybie tekstowym i jego jedynym argumentem powinien być adres IP komputera docelowego.

Program powinien wysyłać pakiety ICMP *echo request* o coraz większych wartościach TTL (podobnie jak robi to wywołanie `traceroute -I`. Dla każdej wartości TTL $\in [1, 30]$ program powinien wykonać następujące operacje.

1. Wysłać 3 pakiety ICMP *echo request* z ustalonym TTL (jeden za drugim, bez czekania na odpowiedź).

2. Odebrać z gniazda odpowiedzi na te pakiety, ale nie czekać na ich nadejście dłużej niż sekundę. Jeśli wszystkie 3 odpowiedzi przyjdą szybciej niż po sekundzie, należy od razu przejść do kolejnego punktu. Ewentualne odpowiedzi na pakiety z poprzednich iteracji (czyli tych iteracji, w których wysyłaliśmy pakiety z mniejszymi martościami TTL) można zignorować.

3. Wyświetlić adres IP routera, od którego nadejdą komunikaty i średni czas odpowiedzi w milisekundach. W przypadku braku odpowiedzi od jakiegokolwiek routera należy wyświetlić `*`. W przypadku odpowiedzi od więcej niż jednego routera należy wyświetlić wszystkie odpowiadające. W przypadku nieotrzymania trzech odpowiedzi w ustalonym czasie zamiast średniego czasu odpowiedzi należy wyświetlić `???`.

Po iteracji, w której otrzymamy odpowiedź od docelowego komputera, należy przestać zwiększać TTL i zakończyć program.

Przykładowy wynik działania programu może wyglądać następująco:

```
> ./traceroute 156.17.254.113
1. 156.17.4.254 40ms
2. 156.17.252.34 ???
3. *
4. *
5. 156.17.254.113 156.17.254.114 50ms
6. 156.17.254.113 65ms
```

Program powinien obsługiwać błędne dane wejściowe, zgłaszając odpowiedni komunikat.

1.1 Uwagi implementacyjne

1. Do wysyłania i odbierania komunikatów ICMP wykorzystaj gniazda surowe. Pamiętaj, że wymagają one uprawnień administratora (programy będą uruchamiane na maszynie wirtualnej z domyślną konfiguracją sieciową, por. dokument *Maszyna wirtualna Virbian* na stronie wykładu).

2. Wykorzystaj fakt, że na podstawie odpowiedzi można zidentyfikować do jakiego pakietu należą: komunikaty ICMP *echo reply* zawierają te same pola *identifier* i *sequence number*, zaś komunikaty ICMP *time exceeded* zawierają oryginalny nagłówek IP i 8 bajtów oryginalnego pakietu IP (czyli w w przypadku odpowiedzi na ICMP *echo request* cały oryginalny nagłówek ICMP).

3. Do oczekiwania na pakiety możesz wykorzystać funkcję `select()`. W szczególności Twój program nie powinien wykonywać aktywnego czekania.

4. Możesz wykorzystywać fragmenty kodu podane na tegorocznym wykładzie, w szczególności kod obliczający sumę kontrolną nagłówka ICMP.

2 Uwagi techniczne

Pliki Sposób utworzenia napisu oznaczanego dalej jako *imie_nazwisko*: Swoje (pierwsze) imię oraz nazwisko proszę zapisać wyłącznie małymi literami zastępując litery ze znakami diakrytycznymi przez ich łacińskie odpowiedniki. Pomiędzy imię i nazwisko należy wstawić znak podkreślenia.

Prowadzącemu ćwiczeniopracownię należy dostarczyć plik *imie_nazwisko*`.tar.xz` z archiwum (w formacie `tar`, spakowane programem `xz`) zawierającym pojedynczy katalog o nazwie *imie_nazwisko* z następującymi plikami.

- Kod źródłowy w C lub C++, czyli pliki `*.c` i `*.h` lub pliki `*.cpp` i `*.h`. Każdy plik `*.c` i `*.cpp` na początku powinien zawierać w komentarzu imię, nazwisko i numer indeksu autora.
- Plik `Makefile` pozwalający na kompilację programu po uruchomieniu `make`.
- Ewentualnie plik `README.txt` lub `README.md`

W katalogu tym **nie** powinno być żadnych innych plików, w szczególności skompilowanego programu, obiektów `*.o`, czy plików źródłowych nie należących do projektu.

Kompilacja Kompilacja i uruchamianie przeprowadzone zostaną w 64-bitowym środowisku Linux. Kompilacja w przypadku C ma wykorzystywać standard C99 lub C17 z ewentualnymi rozszerzeniami GNU (opcja kompilatora `-std=c99`, `-std=gnu99`, `-std=c17` lub `-std=gnu17`), zaś w przypadku C++ — standard C++11, C++14 lub C++17 z ewentualnymi rozszerzeniami GNU (opcje kompilatora `-std=c++11`, `-std=gnu++11`, `-std=c++14`, `-std=gnu++14`, `-std=c++17` lub `-std=gnu++17`). Kompilacja powinna korzystać z opcji `-Wall` i `-Wextra`. Podczas kompilacji nie powinny pojawiać się ostrzeżenia.

3 Sposób oceniania programów

Poniższe uwagi służą ujednoliceniu oceniania w poszczególnych grupach. Napisane są jako polecenia dla prowadzących, ale studenci powinni **koniecznie się** z nimi zapoznać, gdyż będziemy się ściśle trzymać poniższych wytycznych. Programy będą testowane na zajęciach w obecności autora programu. Na początku program uruchamiany jest w różnych warunkach i otrzymuje za te uruchomienia od 0 do 10 punktów. Następnie obliczane są ewentualne punkty karne. Oceniamy z dokładnością do 0,5 punktu. Jeśli ostateczna liczba punktów wyjdzie ujemna wstawiamy zero. (Ostatnia uwaga nie dotyczy przypadków plagiatów lub niesamodzielnych programów).

Testowanie: punkty dodatnie Rozpocząć od kompilacji programu. W przypadku programu niekompilującego się, stawiamy 0 punktów, nawet jeśli program będzie ładnie wyglądał.

3 pkt. Uruchomić program na jednym z adresów należących do instytutu np. `156.17.4.1` a następnie na dwóch adresach zewnętrznych, np. `216.58.207.78` (google.com) i `94.23.242.48` (wikipedia.pl). Porównać wyniki programu z wykonaniem `traceroute -I`. Do następnych punktów wybrać taki adres X, na którym program działał poprawnie. Jeśli nie działał nigdzie poprawnie, to za pozostałe punkty program również otrzymuje zero punktów.

1 pkt. Uruchomić Wiresharka i sprawdzić, czy program faktycznie wysyła określone w zadaniu 3 pakiety na każdy TTL.

1 pkt. Uruchomić jednocześnie na tej samej maszynie program `traceroute -I` X i program na adresie X.

1 pkt. Uruchomić na tej samej maszynie pinganie adresu X. Uruchomić program na adresie X.

2 pkt. Uruchomić jednocześnie dwie instancje programu, obie na adresie X.

2 pkt. Uruchomić jednocześnie dwie instancje programu, jedną na adresie X, jedną na jakimś innym.

Punkty karne Punkty karne przewidziane są za następujące usterki.

-1 pkt. Brak sprawdzania poprawności danych na wejściu.

do -3 pkt. Zła / nieczytelna struktura programu: brak modularności i podziału na funkcjonalne części, niekonsekwentne wcięcia, powtórzenia kodu.

-2 pkt. Aktywne czekanie zamiast zasypiania do momentu otrzymania pakietu.

-1 pkt. Brak sprawdzania poprawności wywołania funkcji systemowych, takich jak `recvfrom()`, `write()` czy `bind()`.

-1 pkt. Nietrzymanie się specyfikacji wejścia i wyjścia. Przykładowo: wyświetlanie nadmiarowych informacji diagnostycznych, inna niż w specyfikacji obsługa parametrów.

-1 pkt. Zły plik `Makefile` lub jego brak: program powinien się kompilować poleceniem `make`: poszczególne pliki `*.c` i `*.cpp` powinny kompilować się do obiektów tymczasowych `*.o` a następnie powinny być konsolidowane do wykonywalnego programu. Polecenie `make clean` powinno czyścić katalog z tymczasowych obiektów (plików `*.o`), zaś polecenie `make distclean` powinno usuwać te obiekty i wykonywalny program pozostawiając tylko pliki źródłowe.

-3 pkt. Kara za wysłanie programu po terminie; opóźnienie nie może być większe niż 1 tydzień.

Materiały do kursu znajdują się w systemie SKOS: https://skos.ii.uni.wroc.pl/.

Marcin Bieńkowski