Zadanie programistyczne nr 3
z Sieci komputerowych

1 Opis zadania

Napisz program `client-udp`, który będzie łączyć się z określonym serwerem i wysyłając pakiety UDP (zgodnie z opisanym niżej protokołem) pobierze od serwera plik o określonej wielkości. Program powinien działać w trybie tekstowym.

Program powinien przyjmować trzy argumenty: *port*, *nazwa_pliku_wynikowego* i *rozmiar_pliku* w bajtach. Celem jest pobranie od specjalnego serwera UDP nasłuchującego na porcie *port* komputera `aisd.ii.uni.wroc.pl` zawartości pierwszych *rozmiar_pliku* bajtów pliku. Następnie ta zawartość powinna zostać zapisana w pliku *nazwa_pliku_wynikowego*. W celu komunikacji z serwerem UDP można mu wysyłać datagramy zawierające następujący napis:

`GET` *start* *długość*`\n`

Wartość *start* powinna być liczbą całkowitą z zakresu $[0, 1\,000\,000]$, zaś *długość* z zakresu $[1, 1\,000]$. Znak `\n` jest uniksowym końcem wiersza, zaś odstępy są pojedynczymi spacjami. Jedyną zawartością datagramu musi być powyższy napis: serwer zignoruje datagramy, które nie spełniają tej specyfikacji. W odpowiedzi serwer wyśle datagram, na którego początku będzie znajdować się napis:

`DATA` *start* *długość*`\n`

Wartości *start* i *długość* są takie, jakich zażądał klient. Po tym napisie znajduje się *długość* bajtów pliku: od bajtu o numerze *start* do bajtu o numerze *start*+*długość*-1. Uwaga: bajty pliku numerowane są od zera!

Program powinien obsługiwać błędne dane wejściowe, zgłaszając odpowiedni komunikat. Program nie powinien wypisywać niepotrzebnych komunikatów diagnostycznych, ale może wypisywać postęp w pobieraniu kolejnych części pliku.

Serwer UDP oczekuje na portach od 40001 do 40010; na takich portach można testować swój program. Nie należy ograniczać działania programu do powyższych portów: podczas oddawania programu będzie on testowany na osobnej instancji serwera UDP działającej na innym porcie.

1.1 Przykład

Jeśli program zostanie uruchomiony w następujący sposób:

```
$> ./client-udp 40001 output 1000
```

to może wysłać w do portu 40001 serwera `aisd.ii.uni.wroc.pl` dwa datagramy o treściach:

```
GET 0 600\n
```

oraz

```
GET 600 400\n
```

następnie zaczekać na odpowiedzi serwera i zapisać odpowiednie 1000 bajtów do pliku `output`.

1.2 Zawodna komunikacja

Należy pamiętać, że datagramy UDP mogą zostać zagubione, zduplikowane lub nadejść w innej kolejności niż były wysyłane. Wykorzystywany serwer UDP sprzyja nauczeniu się tych reguł, działając w następujący sposób:

1. Odpowiedź na dane żądanie zostaje wysłana z prawdopodobieństwem ok. 1/2.
2. Każda wysyłana odpowiedź zostanie zduplikowana z prawdopodobieństwem ok. 1/5. W tej definicji duplikat też jest traktowany jako odpowiedź, więc może pojawić się też duplikat duplikatu (z prawdopodobieństwem ok. 1/25) itd.
3. Każdy wysyłany datagram jest wysyłany z losowym opóźnieniem wynoszącym od 0 do 2 sekund.
4. Serwer utrzymuje kolejkę co najwyżej 100 datagramów, które ma wysłać.

Do tych cech należy dodać również zawodność wprowadzaną przez sieć oraz efekty związane z przepełnianiem kolejki odbiorczej serwera. Jednak w porównaniu z powyższymi cechami, efekty te są zaniedbywalne.

1.3 Uwagi implementacyjne

Konieczne jest sprawdzanie, czy adres źródłowy i port źródłowy datagramu jest prawidłowy. Możesz założyć, że jeśli otrzymujesz dane od adresu IP i portu na które dane wysłałeś, to są one zgodne ze specyfikacją i nie musisz tego sprawdzać.[1]

Twój program będzie testowany pod kątem poprawności i wydajności. W najprostszym wariancie można zaprogramować go jako algorytm typu *stop-and-wait*. Do odczekiwania i sprawdzania, czy gniazdo zawiera datagram, można wykorzystać funkcję `select()`. Tak zaimplementowany został program `client-udp-slower` (statycznie skonsolidowaną wersję 32-bitową i 64-bitową można pobrać ze strony wykładu). Za poprawną implementację takiego podejścia można dostać ok. 6 punktów.

Podejście typu *stop-and-wait* jest bardzo nieefektywne. Aby je poprawić, możesz zaimplementować odpowiednik algorytmu przesuwnego okna, utrzymujący odpowiednie liczniki czasu dla wszystkich otrzymanych segmentów. Tak zaimplementowany został program `client-udp-faster` (wersje również do pobrania ze strony wykładu). Za taką implementację można dostać maksymalną liczbę punktów.

Oczywiście Twój program nie musi pobierać danych zgodnie z opisanymi wyżej schematami. Pamiętaj, że program będzie testowany w pracowni 109, czyli parametry wpływające na jego efektywność (czas odczekiwania i rozmiar okna) należy dobrać eksperymentalnie w sieci instytutowej. Napisanie programu, który będzie dynamicznie dostosowywał się do parametrów łącza nie jest wymagane. Również programy `client-udp-slower` i `client-udp-faster` mogą się kiepsko zachowywać (i zapychać łącze), jeśli zostaną uruchomione spoza sieci lokalnej instytutu.

[1] W prawdziwych zastosowaniach byłby to bardzo zły pomysł.

2 Uwagi techniczne

Pliki Swojemu ćwiczeniowcowi należy dostarczyć jeden spakowany plik zawierający katalog z programem. Katalog powinien zawierać:

- 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`.

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 przeprowadzane zostaną w 64-bitowym środowisku Linux.

Kompilacja w przypadku C ma wykorzystywać standard ISO C99 z ewentualnymi rozszerzeniami GNU (opcja kompilatora `-std=c99` lub `-std=gnu99`).

Kompilacja powinna wykorzystywać opcje `-Wall` i `-W`. Podczas kompilacji nie powinny pojawiać się ostrzeżenia.

3 Sposób oceniania programów

Poniższe uwagi służą ujednoliceniu uceniania w poszczególnych grupach. Napisane są jako polecenia dla ćwiczeniowców, 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 punkta. 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.

Do testów zostanie uruchomiona osobna instancja serwera, działająca na innych portach; porty te zostaną podane ćwiczeniowcom. Chodzi o to, żeby podczas oceniania zadań zminimalizować efekty związane z jednoczesnym dostępem do serwera przez wiele osób. Przed testami należy wyłączyć zaporę (`netmode lab`) i pobrać adres IP (`dhclient eth0`).

2 pkt. Uruchomić program do pobrania ok. 3 000 bajtów. Liczba bajtów nie powinna być wielokrotnością 1 000: chodzi o sprawdzenie jak program studenta radzi sobie z taką sytuacją. Zmierzyć czas instrukcją `time`. Na takich samych danych uruchomić program `client-udp-slower-32`; niech t będzie czasem jego działania. Program studenta otrzymuje punkty, jeśli jego czas działania jest nie większy niż $4 \cdot t + 5$ sek. a generowane przez oba programy pliki są identyczne.

2 pkt. Jak wyżej, ale dla ok. 15 000 bajtów.

2 pkt. Uruchomić program do pobrania ok 15 000 bajtów. Zatrzymać go w trakcie wykonywania; sprawdzić wiresharkiem jaki jest jego port źródłowy. Następnie poleceniem `nc` wysłać do tego portu źródłowego datagram zawierający (a) śmieci, (b)

błędną odpowiedź na wysłane zapytanie `GET start dlugosc\n`, czyli przykładowo `DATA start dlugosc\nblah blah`. Wznowić działanie programu i sprawdzić, czy generowany jest poprawny plik.

2 pkt. Jak w pierwszym punkcie, ale pobieramy ok. 50 000 bajtów i porównujemy czas z czasem działania programu `client-udp-faster-32`.

2 pkt. Jak w pierwszym punkcie, ale pobieramy ok. 900 000 bajtów i porównujemy czas z czasem działania programu `client-udp-faster-32`.

Punkty karne Punkty karne przewidziane są za następujące usterki.

do -3 pkt. Zła / nieczytelna struktura programu: wszystko w jednym pliku, brak modularności i podziału na funkcjonalne części, niekonsekwetne wcięcia, powtórzenia kodu.

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

-1 pkt. Zły plik `Makefile` lub jego brak: program powinien się kompilować poleceniem `make`, polecenie `make clean` powinno czyścić program z tymczasowych obiektów (plików `*.o`), polecenie `make distclean` powinno usuwać skompilowane programy i zostawiać tylko pliki źródłowe.

-1 pkt. Niewłaściwa kompilacja: nietrzymanie się opcji podanych w zadaniu, ostrzeżenia wypisywane przy kompilacji, kompilacja bezpośrednio do pliku wykonywalnego bez tworzenia obiektów tymczasowych `*.o`.

-1 pkt. Nietrzymanie się specyfikacji wejścia i wyjścia. Przykładowo: wyświetlanie nadmiarowych informacji diagnostycznych, oczekiwanie na podanie parametrów na stamdardowym wejściu zamiast jako argumentów.

-3/-6 pkt. Kara za wysłanie programu z opóźnieniem: -3 pkt. za opóźnienie do 1 tygodnia, -6 pkt. za opóźnienie do 2 tygodni. Programy wysyłane z większym opóźnieniem nie będą sprawdzane.

Marcin Bieńkowski