PSI – Zadania 1.1 + 1.2

21.11.2025

Jakub Kwaśniak

Michał Pędziwiatr

Kacper Górski

Zadanie 1.1

Ustalona została sztywna, międzyplatformowa forma danych datagramu (pliki **datagram.py** oraz **datagram.h**). Jego struktura rozpoczyna się dwubajtowym nagłówkiem, informującym o liczbie znajdujących się po nim par „nazwa – wartość”. Nagłówek (ilość par klucz-wartość w datagramie) kodowany jest w konwencji big endian (**“!”** w py oraz funkcji **htons** w przypadku C) oraz jako wartość unsigned short (**“H”** w py czy unsigned short/ uint16_t w C). Następujące po nim pola mają stałą długość 20 bajtów, są kodowane ASCII i dopełniane “null byte’ami”.

Serwery (pliki **udp-server.c**, **udp-server.py**, **server.py**) w obu językach działają na zasadzie nieskończonej pętli, nasłuchując na danym porcie aż do zabicia procesu. Po otrzymaniu pakietu za pomocą **recvfrom** oraz zdekodowaniu zawartego w nim datagramu, serwer wypisuje w konsoli odkodowany datagram. Odpowiada mu także własnym zakodowanym datagramem (z użyciem funkcji **sendto**), zawierającym informację o statusie komunikacji.

Programy implementujące klientów (**udp-client.py**, **udp-client.c)** w przeciwieństwie do serwera nie działają w sposób ciągły. Po przesłaniu do serwera zakodowanych datagramów, czekają jedynie na odpowiedź zwrotną ze statusem operacji i po jej odebraniu kończą swoją pracę.

Stworzony system przetestowany został w różnych kombinacjach, przede wszystkim sprawdzając kombinacje międzyplatformowe:

Testy odtworzyć można w zapisie sesji terminala dostępnym w przesłanych plikach (demo.cast)

! [wszystkie serwery oraz klienci byli uruchomieni jednocześnie, w jednej powłoce, dlatego wypisywane w strumieniu wyniki mogą nie będą ułożone chronologicznie]

Zadanie 1.2

Na bazie rozwiązania zadania 1.1 stworzono testowy scenariusz, którego celem było określenie maksymalnego rozmiaru przyjętego przez serwer datagramu. Skupia się on na wysyłaniu przez klienta datagramów o przyrastającej wielkości za pomocą metody **_find_max_capacity()** zaimplementowanego w Pythonie klienta.

Jeśli datagram przekracza dozwoloną długość kończy się to wyjątkiem z wiadomością “Message to long”.

Jeśli napotkany zostanie błąd i w funkcji send() ustawiona jest flaga check_overflow=True, wywołana zostanie funkcja **_find_max_capacity(),** która znajdzie granicę ilości bajtów możliwą do przesłania w środowisku testowym.

W 1.1 zaimplementowaliśmy strukturę Datagram, która zgłasza błąd w przypadku stworzenia niewłaściwego/niezgodnego z nią datagramu, czyli z ‘niewyrównaną’ pamięcią struktury - jeśli pamięć struktury nie jest wyrównana do założonej wielkości - na każde pole klucz-wartość przeznaczone jest np 20 bajtów i 20 bajtów musi być zajęte. W ramach znalezienia limitu przesyłanych bajtów w sieci testowej musieliśmy tworzyć niepoprawne datagramy (dlatego serwer wielokrotnie w tych testach zwracał komunikat o błędzie dekodowania - co jest porządanym efektem :)

Po otrzymaniu pakietu przekraczającego dopuszczalny limit - wywoływane jest funkcja **_find_max_capacity(),** która szuka limitu między wielkościami ostatniego przesłanego datagramu, a pierwszego nieprzesłanego. Jeśli testowany datagram przekracza dozwoloną wielkość w bajtach zgłaszany jest komunikat “Communication MTP_ERROR occurred” - co oznacza że należy szukać datagramu o mniejszej wielkości - w ten sposób z dokładnością do 1 bajta znajdywana jest maksymalna dozwolona wielkość datagramu.

- **maksymalna wielkość datagramu w sieci testowej =1472**
    - domyślny bridge Docker ma MTU=1500 B
    - z tego 20 B to header IPv4, a 8 B to header UDP, więc zostaje 1472 B

Wykonane testy widoczne są w zapisanej wersji sesji terminala (demo.cast), gdzie przeysłane były ciągi znaków a b”a…a”, rosnące wraz z kolejnymi wysłanymi datagramami