1.04.2021

Na dzisiejszych zajęciach implementujemy strumieniową (używającą protokołu TCP) wersję serwera sumującego liczby według poniższej specyfikacji:

> Komunikacja pomiędzy klientem a serwerem odbywa się przy pomocy połączenia strumieniowego. Klient wysyła jedną lub więcej linii zawierających liczby. Dla każdej odebranej linii serwer zwraca linię zawierającą pojedynczą liczbę (obliczoną sumę) bądź komunikat o błędzie.
>
> Ogólna definicja linii jest taka jak w wielu innych protokołach: ciąg drukowalnych znaków ASCII (być może pusty) zakończony dwuznakiem `\r\n`.
>
> Linia z liczbami zawierać może tylko cyfry i spacje. Ciągi cyfr należy interpretować jako liczby dziesiętne. Spacje służą jako separatory liczb; każda spacja musi znajdować się pomiędzy dwiema cyframi. Linia nie może być pusta, musi zawierać przynajmniej jedną liczbę.
>
> Linia sygnalizująca niemożność poprawnego obliczenia sumy zawiera pięć liter składających się na słowo `ERROR` (po tych literach oczywiście jest jeszcze terminator linii, czyli `\r\n`).
>
> Serwer może, ale nie musi, zamykać połączenie w reakcji na nienaturalne zachowanie klienta. Obejmuje to wysyłanie danych binarnych zamiast znaków ASCII, wysyłanie linii o długości przekraczającej przyjęty w kodzie źródłowym serwera limit, długi okres nieaktywności klienta, itd. Jeśli serwer narzuca maksymalną długość linii, to limit ten powinien wynosić co najmniej 1024 bajty (1022 drukowalne znaki i dwubajtowy terminator linii).
>
> Serwer nie powinien zamykać połączenia jeśli udało mu się odebrać poprawną linię w sensie ogólnej definicji, ale dane w niej zawarte są niepoprawne (np. oprócz cyfr i spacji są przecinki). Powinien wtedy zwracać komunikat o błędzie i przechodzić do przetwarzania następnej linii przesłanej przez klienta.
>
> Serwer powinien zwracać komunikat błędu również wtedy, gdy przesłane przez klienta liczby bądź ich suma przekraczają zakres typu całkowitoliczbowego wykorzystywanego przez serwer do prowadzenia obliczeń.

Proszę zwrócić uwagę, że konsekwencją fragmentu „spacja musi znajdować się pomiędzy dwiema cyframi” jest to, że nie można mieć spacji na początku ani też na końcu linii, nie można też oddzielać liczb więcej niż jedną spacją.

Zastanów się nad sposobem działania serwera. Jego algorytm będzie musiał być bardziej złożony niż w przypadku sumatora UDP. Tam pojedyncza operacja odczytu zawsze zwracała jeden datagram, czyli jeden kompletny ciąg liczb do zsumowania. W przypadku połączeń TCP niestety tak łatwo nie jest.

Po pierwsze, jeśli klient od razu po nawiązaniu połączenia wysłał kilka zapytań jedno za drugim, to serwer może je odebrać sklejone ze sobą. Pojedyncza operacja odczytu ze strumienia może np. zwrócić 15 bajtów odpowiadających znakom `2 2\r\n10 22 34\r\n` — jak widać są to dwa ciągi liczb. Serwer w odpowiedzi powinien zwrócić ciąg 7 bajtów `4\r\n66\r\n`.

Po drugie, operacja odczytu może zwrócić tylko początkową część zapytania. Kod serwera musi wtedy ponownie wywołać read(). Takie ponawianie odczytów i odbieranie kolejnych fragmentów ciągu liczb musi trwać aż do chwili odebrania \r\n — dopiero wtedy wiemy, że w buforze mamy kompletny ciąg liczb do zsumowania.

Takie ponawianie odczytów nie wymaga użycia dodatkowego bufora. Rozważcie Państwo: jeśli wywołanie `read(s, buf, 1024)` zwróciło nam np. 30, i w tych 30 bajtach nie ma `\r\n`, to znaczy że następne bajty odczytane ze strumienia będą przedłużeniem tych, które już są w buforze. Należy więc teraz wywołać

`read(s, buf + 30, 1024 - 30)`, i w ten sposób druga porcja bajtów zostanie zapisana jako ciąg dalszy pierwszej porcji.

Po trzecie, mogą się zdarzyć oba powyższe przypadki równocześnie. Serwer może np. odczytać ze strumienia 9 bajtów odpowiadających znakom `2 2\r\n10 2`.

Spróbuj rozpisać w formie pseudokodu algorytm serwera obsługujący powyższe komplikacje i jeszcze raz się zastanów, czy na pewno poradzi sobie nawet przy założeniu maksymalnie złej woli ze strony klienta.

To zadanie jest ważne — zaimplementowane rozwiązanie (archiwum zawierające plik źródłowy i np. makefile) trzeba przesłać poprzez formularz zadania w MS Teams najpóźniej w czwartek 14 kwietnia.

(na koniec) Proszę przeanalizować przedstawioną na wykładzie koncepcję automatu przetwarzającego kolejne bajty z wejścia. Automat nie potrzebuje bufora z kompletnym ciągiem liczb, on po prostu konsumuje bajty w miarę tego jak nadchodzą. Zawarte w poprzednim punkcie rozważania o tym, jak sprawdzać czy w buforze mamy kompletną linię, są więc zupełnie zbędne. Użycie automatu powinno wielce ułatwić zaprojektowanie poprawnie działającego algorytmu (który dodatkowo będzie w stanie przetwarzać dowolnie długie linie).

(nieobowiązkowe) Zaimplementuj automat z wykładu i zastosuj go w serwerze sumującym liczby według powyższej specyfikacji. W tym zadaniu można użyć dowolnego języka programowania.