

# INSTRUKCJA do ćwiczeń laboratoryjnych z

## PSI

G. Blinowski 30/10/2024 v.1.4  
(dotyczy wyłącznie grup projektowych GB)

**Formuła:** „Mini-projekt”, 3 osoby w zespole

**Założenia:** studenci rozwijają metodą inkrementalną program wg. podanych niżej ścieżek.

**Czas realizacji:** początek: 07.11.2024, koniec: 30.11.2024

**Punktacja:** 20 p. Uwaga: nie ma progu minimalnego niezbędnego do zaliczenia laboratorium; niezgłoszenie zadania w terminie powoduje uzyskanie z niego 0 p. – bez możliwości poprawy.

**Materiały dodatkowe:**

- na serwerze studia2 w zasobach przedmiotu

**Język i platforma:**

C/C++ oraz Python – wg zaleceń w danym zadaniu. Inne języki (jak np. Java, C#) nie są dozwolone. Platformą do realizacji zadań jest serwer „**bigubu**” i środowisko Dockera (instrukcje w osobnych plikach).

**Punktacja i kryteria oceny**

Zadanie	Punktacja	Realizacja do ...
1. 1	5	16.11.2024
1.2	8	23.11.2024
2.1a, 2.1b, 2.1c, 2.1d	7	30.11.2024

Zespoły otrzymają indywidualnie informację o zestawie zadań do realizacji.

**Sposób zaliczenia**

- W terminie podanym w tabelce wyżej (odpowiednio dla kolejnych zadań 1.1, 1.2 i 2.1) należy przesłać mailem kod źródłowy oraz pdf ze zrzutami z wynikami działania (zrzut z konsoli – widoczne polecenie i efekt działania)
- ... i oczekiwać na uwagi prowadzącego. Zespół może (ale nie musi) być poproszony o konsultacje zdalne przez Teams. Na tym spotkaniu zespół musi mieć techniczną możliwość zdalnego zademonstrowania działającego kodu.
- Po uwzględnieniu uwag od prowadzącego należy przesłać poprawiony kod źródłowy (o ile była konieczna poprawa i mini- sprawozdanie) – można to zrobić „hurtem” dla wszystkich zadań, po otrzymaniu kompletu uwag ze wszystkich zadań. Należy to zrobić w przeciągu 3 dni roboczych od otrzymania kompletu uwag.
- Proszę w temacie maila napisać: „[PSI] Zespół Znn”; dokumentacja jako pdf (**nie .doc, nie .odt, etc.**), pliki źródłowe (**bez binarnych!**) W archiwum **.zip** lub **.tar.gz**
- Na każdym etapie prac zainteresowanych zapraszam także na opcjonalne konsultacje.

## Uwaga:

- Jedyna dopuszczalna forma archiwum to .zip lub .tar.gz (.tgz).
- Jedyna dopuszczalna forma dokumentacji to .pdf.
- Pliki w innym formacie będą ignorowane, a maile z nimi kasowane bez czytania
- Nie są dopuszczalne linki do repozytoriów prywatnych lub publicznych! Maile zawierające linki zamiast załączników z kodem i/lub dokumentacją będą kasowane bez czytania.
- **Tworzenie kodu za pomocą mechanizmów AI jest zakazane. Złamanie tego warunku powoduje automatyczne wyzerowanie punktów z laboratorium i może mieć dalsze konsekwencje formalne.**

## Ocena wystawiana jest na podstawie:

- spełnienia wymagań,
- jakości kodu (patrz wskazówki na końcu w4.pdf) oraz zachowania ogólnej praktyki dobrego kodowania,
- jakości dokumentacji i raportu z testów.

## Co powinno zawierać mini-sprawozdanie do zadań 1 i 2:

- Nagłówek – nazwę przedmiotu, **autorów** (skład zespołu), datę sporządzenia, wersję i historię zmian (jeśli będzie poprawka).
- **Treść** zadania.
- Opis rozwiązania problemu. Może to być fragment kodu z wstawkami informacyjnymi. Przykład - jeśli zadanie dotyczy wykorzystania nieblokującego we/wy, należy skupić się na opisie rozwiązania tej właśnie komunikacji.
- Uwagi dot. problemów, na które natrafił zespół i jak zostały one rozwiązane.
- Opis konfiguracji testowej (adresy IP, porty, interfejsy, etc.).
- Opis testowania i wydruki z testów (mogą być skrócone z istotnymi danymi).
- Wnioski końcowe i uwagi własne.
- **NIE** należy podawać informacji ogólnie znanych, np. opisywać działania funkcji gniazd i systemowych, działania protokołów L3 i L4, etc. Jeśli jednak natrafi się na nietypowe, nieoczekiwane, słabo udokumentowane, etc. działanie funkcji, to należy o tym napisać.
- Projekt dotyczy technik programowania protokołów internetowych, inne aspekty np. związane z interfejsem użytkownika nie będą brane pod uwagę przy jego ocenie, jednak wymaga się minimum zdrowego rozsądku przy obsłudze trybu command-line.
- Wolno korzystać z istniejącego kodu (np. realizującego logowanie), ale trzeba wyraźnie o tym napisać w dokumentacji. Kluczowy kod sieciowy powinien być stworzony własnoręcznie i od podstaw.
- Dokumentacja powinna być napisana poprawną polszczyzną, bez wyrażen slangowych i niepotrzebnych anglicyzmów.

## Dodatkowe uwagi:

- Zrzuty z konsoli nie powinny być na ciemnym tle (nie powinno mieć to formatu zrzutu ekranu), powinny być czytelne i nie powinny zawierać nadmiaru informacji – komunikaty nadmiarowe, powtarzające się itp. należy usunąć, można je zastąpić symbolem [...]

## Zadania

### Z 1 Komunikacja UDP

Napisz zestaw dwóch programów – klienta i serwera wysyłające datagramy UDP. Wykonaj ćwiczenie w kolejnych inkrementalnych wariantach (rozszerzając kod z poprzedniej wersji). Klient oraz serwer powinien być napisany zarówno w C jak i Pythonie (4 programy).

Sprawdzić i przetestować działanie „między-platformowe”, tj. klient w C z serwerem Python i vice versa.

#### Z 1.1

Klient wysyła, a serwer odbiera datagramy o stałym rozmiarze (rzędu kilkuset bajtów). Datagramy powinny posiadać ustaloną formę danych. Przykładowo: pierwsze dwa bajty datagramu mogą zawierać informację o jego długości, a kolejne bajty kolejne litery A-Z powtarzające się wymaganą liczbę razy (ale można przyjąć inne rozwiązanie. Odbiorca powinien weryfikować odebrany datagram i odsyłać odpowiedź o ustalonym formacie. Klienta powinien wysyłać kolejne datagramy o przyrastającej wielkości np. 1, 100, 200, 1000, 2000... bajtów. Sprawdzić, jaki był maksymalny rozmiar wysłanego (przyjętego) datagramu. Ustalić z dokładnością do jednego bajta jak duży datagram jest obsługiwany. Wyjaśnić.

#### Z 1.2

Wychodzimy z kodu z zadania 1.1, tym razem pakiety datagramu mają stałą wielkość, można przyjąć np. 512B. Należy zaimplementować prosty protokół niezawodnej transmisji, uwzględniający możliwość gubienia datagramów. Rozszerzyć protokół i program tak, aby gubione pakiety były wykrywane i retransmitowane. Wskazówka – „Bit alternate protocol”. Należy uruchomić program w środowisku symulującym błędy gubienia pakietów. (Informacja o tym, jak to zrobić znajduje się w skrypcie opisującym środowisko Dockera).

To zadanie można wykonać, korzystając z kodu klienta i serwera napisanych w C lub w Pythonie (do wyboru). Nie trzeba tworzyć wersji w obydwu językach.

### Z 2 Komunikacja TCP

Napisz zestaw dwóch programów – klienta i serwera komunikujących się poprzez TCP. Transmitowany strumień danych powinien być stosunkowo duży, nie mniej niż 100 kB.

Jeden z wariantów do realizacji:

#### Z 2.1a

Klient TCP napisany w C wysyła złożoną strukturę danych. Przykładowo: tworzymy w pamięci listę jednokierunkową lub drzewo binarne struktur zawierających (oprócz danych organizacyjnych) pewne dane dodatkowe: np. liczbę całkowitą 16-o bitową, liczbę całkowitą 32-u bitową oraz napis zmiennej i ograniczonej długości. Serwer napisany w Pythonie/C powinien te dane odebrać, dokonać poprawnego „odpakowania” tej struktury i wydrukować jej pola (być może w skróconej postaci, aby uniknąć nadmiaru wyświetlanych danych).

Wskazówka: można wykorzystać moduły Python-a: struct i io.

#### Z 2.2b

Klient powinien wysyłać do serwera strumień danych w pętli (tzn. danych powinno być „dużo”, minimum rzędu kilkuset KB). Serwer powinien odbierać dane, ale między odczytami realizować sztuczne opóźnienie (np. przy pomocy funkcji sleep()). W ten sposób symulujemy zjawisko odbiorcy, który „nie nadąża” za szybkim nadawcą. Stos TCP będzie spowalniał nadawcę, aby uniknąć tracenia danych. Należy zidentyfikować objawy tego zjawiska po stronie klienta (dodając pomiar i logowanie czasu) i krótko przedstawić swoje wnioski poparte uzyskanymi statystykami czasowymi. Wskazane jest też przeprowadzenie eksperymentu z różnymi rozmiarami bufora nadawczego po stronie klienta (np. 100 B, 1 KB, 10 KB).

To zadanie należy wykonać, korzystając z kodu klienta i serwera napisanych w języku C.

#### Z 2.2c

Zmodyfikuj program klienta tak, aby jednorazowo wysyłane były małe porcje danych (mniejsze od pojedynczej struktury) i wprowadź dodatkowe sztuczne opóźnienie po stronie klienta (przy pomocy funkcji sleep()). W programie serwera zorganizuj kod tak, aby serwer kompletował dane i drukował je „na bieżąco”.

## **Z 2.2d**

Zmodyfikować serwer tak, aby miał konstrukcję współbieżną, tj. obsługiwał każdego klienta w osobnym procesie. Przy czym:

- Dla C należy posłużyć się funkcjami `fork()` oraz (obowiązkowo) `wait()`.
- Dla Pythona należy posłużyć się wątkami, do wyboru: wariant podstawowy lub skorzystanie z `ThreadPoolExecutor`

Przetestować dla kilku równoległe działających klientów.