

**Kauno technologijos universitetas**

Informatikos fakultetas

**Pirmas laboratorinis darbas: komunikacijos realizacija naudojant soketus (angl. socket) ir RPC (angl. Remote Procedure Call)**

T120M337 Paskirstytosios sistemos ir algoritmai

**Gintautas Šedys**

Studentas

**Doc. Dalius Makackas**

Dėstytojas

Turinys

[Paveikslų sąrašas 3](#_Toc115187857)

[Komunikacijos realizacija naudojant soketus (angl. socket) 4](#_Toc115187858)

[Užduotis 4](#_Toc115187859)

[Realizacijos technologijos 4](#_Toc115187860)

[Realizacija 4](#_Toc115187861)

[Komunikacijos realizacija naudojant RPC (angl. Remote Procedure Call) 7](#_Toc115187862)

[Užduotis 7](#_Toc115187863)

[Šaltiniai 9](#_Toc115187864)

# Paveikslų sąrašas

[pav. 0.1 Klientų komunikacija su serveriu 7](#_Toc115187809)

# Komunikacijos realizacija naudojant soketus (angl. socket)

## Užduotis

Sukurti kliento serverio sistemą. kuri realizuotų:

1. Komunikacijos realizacija išpildoma soketų (angl. socket) pagalba, panaudojant variante nurodytą protokolą.
2. Serverio paslaugos funkcionalumas individualus (sugalvoti pačiam).
3. Sistemoje turi numatyta kliento užklausa, kurioje turi būti nemažiau, kaip du parametrai, sudedami į paketą panaudojant nurodytą variante formatavimą.
4. Turi būti numatyta, kad grąžinamas rezultatas gali būti sudarytas iš kelių paketų.

Variantas Nr. 0

* Protokolas: TCP
* Formatavimas: duomenų struktūra (binarinis)

Papildomi reikalavimai:

* Tarp klientų ir serverio ryšys turi būti daugialypis (angl. multi connection). [1]

**Serverio paslaugos funkcionalumas:**

* Serveris – atsitiktinių skaičių generatorius.
* Kliento užklausos formatas – trijų skaičių masyvas, kur:
  + Pirmas elementas nusako kiek skaičių generuoti, pvz. 11
  + Antras elementas nusako sugeneruotų skaičių dydį, pvz. 40 (Skaičiai bus tarp -40 ir 40)
  + Trečias elementas nusako sugeneruotų skaičių vidurkį, pvz. 14.54
* Serverio atsakymo formatas – n atsitiktinių skaičių masyvas

## Realizacijos technologijos

* Klientas ir serveris implementuoti su „Python“ programavimo kalba.
* Daugialypis ryšys buvo užtikrintas naudojantis „selectors“ paketu.
* Duomenų konvertavimui iš ir į binarini tipą buvo naudotas „pickle“ paketas.
* TCP komunikacijai sukurti buvo naudojamas „socket“ paketas. [2]

## Realizacija

Binarinės žinutės išskaidymui į fiksuoto dydžio gabalus buvo sukurta funkcija „divide\_chunks“.

def divide\_chunks(l, n):

    size = n - 1

    for i in range(0, len(l), size):

        res = l[i:i + size]

        if len(res) != size:

            res += bytes(size - len(res))

        res += bytes(1)

        yield bytearray(res)

Čia l – baitų masyvas, n – gabalo dydis.  
Prie kiekvieno gabalo yra pridedamas papildomas baitas informacijai apie žinutę saugoti.  
Vėliau į paskutinį žinutės gabalo baitą yra įrašoma, kad tai paskutinis žinutės gabalas.

**Serverio realizacija**

def accept\_wrapper(sock, sel):

    conn, addr = sock.accept()

    print(f"Accepted connection from {addr}")

    conn.setblocking(False)

    data = types.SimpleNamespace(addr=addr, my\_data\_bin = bytes(0))

    events = selectors.EVENT\_READ

    sel.register(conn, events, data=data)

def service\_connection(key, mask, sel, buff\_size):

    sock = key.fileobj

    data = key.data

    try:

        if mask & selectors.EVENT\_READ:

            recv\_data = sock.recv(buff\_size)

            if recv\_data:

                data.my\_data\_bin += recv\_data[:-1]

                if recv\_data[-1] == 1:

                    arr = pickle.loads(data.my\_data\_bin)

                    print(f"Just got {arr} from {data.addr}, ...!")

                    res = [random.uniform(-1, 1) \* arr[2] + arr[1] for i in range(arr[0])]

                    bin\_mess = pickle.dumps(res)

                    chunks = [i for i in divide\_chunks(bin\_mess, buff\_size)]

                    chunks[-1][-1] = 1

                    for i in chunks:

                        sock.send(i)

                    data.my\_data\_bin = bytes(0)

            else:

                sel.unregister(sock)

                sock.close()

    except ConnectionError:

        sel.unregister(sock)

def start\_server(host = "127.0.0.1", port = 65432, buff\_size = 8):

    # https://docs.python.org/3/library/selectors.html

    # Selector contains list of file objects

    # Function select returns object that is ready for action

    # For KeyboardInterrupt to work timeout was set to 1.

# If timeout wouldn't be set then select would block until

# some object is ready

    sel = selectors.DefaultSelector()

    # Init socket with:

    # AF\_INET - Internet address family for IPv4

    # SOCK\_STREAM - Socket type for TCP

    sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    sock.bind((host, port))

    sock.listen()

    sock.setblocking(False)

    # EVENT\_READ - object is available for read

    sel.register(sock, selectors.EVENT\_READ, data=None)

    try:

        while True:

            events = sel.select(timeout = 1)

            for key, mask in events:

                if key.data is None:

                    # If new connection

                    accept\_wrapper(key.fileobj, sel)

                else:

                    # If existing connection

                    service\_connection(key, mask, sel, buff\_size)

    except KeyboardInterrupt:

        print("Caught keyboard interrupt, exiting")

    finally:

        sel.close()

start\_server()

Serveris susideda iš trijų funkcijų:

* Serverio pradžios funkcijos „start\_server“  
  Ši funkcija sukuria serverį, ir jį paruošia klientų priėmimui, bei jų užklausų apdorojimui.
* Kliento priėmimo funkcijos „accept\_wrapper“  
  Ši funkcija apdoroja pradinį klientų prisijungimą ir jos užregistruoja į „slector“ objektą, per kurį vėliau bus priimamos jų užklausos.
* Kliento žinutės apdorojimo funkcijos „service\_connection“  
  Ši funkcija priima žinutę iš kliento, iš žinutės sugeneruoją skaičių masyvą ir jį išsiunčia klientui.   
  Jeigu gauną tuščią žinutę (TCP tuščia žinutė – klientas atsijungė), tai soketas su klientu yra uždaromas

**Kliento realizacija**

def read\_input():

    print("Įveskite skaičių kiekį: ", end="")

    n = int(input())

    print("Įveskite vidurkį: ", end="")

    mean = int(input())

    print("Įveskite standartinį nuokrypį: ", end="")

    std = int(input())

    message = [n, mean, std]

    return(message)

def start\_client(host = "127.0.0.1", port = 65432, buff\_size = 8):

    with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

        s.connect((host, port))

        while(True):

            message = read\_input()

            bin\_message = pickle.dumps(message)

            chunks = [i for i in divide\_chunks(bin\_message, buff\_size)]

            chunks[-1][-1] = 1

            for i in chunks:

                send\_bytes = s.send(i)

            res = bytes(0)

            end = False

            while not end:

                val = s.recv(buff\_size)

                res += val[:-1]

                if val[-1] == 1: end = True

            arr = pickle.loads(res)

            print(arr)

start\_client()

Klientas turi dvi funkcijas:

* Funkciją naudotojo įvestims nuskaityti – „read\_input“
* Kliento pradžios funkciją – „start\_client“  
  Ši funkcija užmezga ryšį su serveriu ir jam nusiunčią užklausą

**Veikimo pavyzdys**

Text

Description automatically generated

pav. . Klientų komunikacija su serveriu

Kaip matome iš pavyzdžio pav. 0.1 komunikacija tarp kelių klientų ir serverio vyksta sklandžiai.

[2] [3]

# Komunikacijos realizacija naudojant RPC (angl. Remote Procedure Call)

## Užduotis

Sukurti kliento serverio sistemą, kuri realizuotų:

1. Komunikacijos realizacija išpildoma RPC (.Net Remoting) pagalba, panaudojant variante nurodytą protokolą.
2. Serverio paslaugos funkcionalumas individualus (toks pat kaip ir pirmoje laboratorinio dalyje).
3. Sistemoje turi numatyta kliento užklausa, kurioje turi būti nemažiau, kaip du parametrai, panaudojant nurodytą variante formatavimą.
4. Turi būti numatyta, kad grąžinamas rezultatas gali būti sudarytas iš kelių laukų (sudėtinis tipas).

Variantas Nr. 0

* Protokolas: http
* Formatavimas: binarinis

Papildomi reikalavimai:

* Rezultatas gražinamas pagal nuorodą (grąžinamas rezultato objektas lieka serveryje).
* Serveris turi gražinti kiek kartų klientas į jį kreipiasi. [1]

# Išvados ir pastabėjimai

## Komunikacijos realizacija naudojant soketus išvados ir pastabėjimai

Komunikuojant svarbu užtikrinti vienodą išsiustų ir priimtų paketų dydį. Taip pat svarbu suteikti paketui meta informacijos, kad būtu galima pasakyti kada žinutė prasideda/baigęsi. Jeigu išsiustų ir priimtų paketų dydis nesutampa, tada gali iškilti problema kad pasimes meta informacijos baito vieta pakete (pvz. bitas bus ne paketo gale, o kažkur per vidurį). Dėl paketų išsiuntimo/priėmimo tvarkos galvoti nereikėjo, nes buvo naudojimas TCP protokolas.

## Komunikacijos realizacija naudojant RPC išvados ir pastabėjimai

## Bendros išvados ir pastabėjimai

# Šaltiniai

|  |  |
| --- | --- |
| [1] | KTU, „https://moodle.ktu.edu/,“ 09 2022. [Tinkle]. Available: https://moodle.ktu.edu/pluginfile.php/553409/mod\_resource/content/6/Uzduotys%202022.txt. [Kreiptasi 27 09 2022]. |
| [2] | N. Jennings, „Socket Programming in Python (Guide),“ 21 02 2022. [Tinkle]. Available: https://realpython.com/python-sockets/. [Kreiptasi 27 09 2022]. |
| [3] | The Python Software Foundation, „selectors — High-level I/O multiplexing,“ 27 09 2022. [Tinkle]. Available: https://docs.python.org/3/library/selectors.html. [Kreiptasi 27 09 2022]. |