Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное

учреждение высшего образования

Московский технический университет связи и информатики

Кафедра «Системного программирования»

Лабораторная работа №4. Часть 2.

по дисциплине

«Операционные системы»

Выполнила: студентка группы БСТ2104

Первухина А. А.

Проверила: Алексанян Диана Ашотовна

Москва 2023

**Содержание**

[**Задание 2. Реализация обмена данными между процессами** 3](#_Toc151654783)

[**Задание 2.1. Разделяемая память** 4](#_Toc151654784)

[**Задание 2.2. Сокеты** 7](#_Toc151654785)

[**Задание 2.3. Каналы** 10](#_Toc151654786)

[**Вывод** 12](#_Toc151654787)

# **Задание 2. Реализация обмена данными между процессами**

**Цели работы:**

* изучение набора средств коммуникации процессов в ОС GNU/LINUX;
* получение практических навыков по использованию механизмов межпроцессорного взаимодействия.

**Задание.** Написать программы, выполняющие с помощью всех перечисленных механизмов межпроцессорного взаимодействия (разделяемая память, сокеты, каналы) задачу в соответствии с № по журналу, приведённую в Таблице 1.

Для решения задачи создавать минимально 2 ведомых процесса, выполняющих переданные ведущим процессом подзадачи и возвращающие результаты ведущему процессу. Финальную обработку результатов по необходимости, осуществлять ведущим процессом.

Таблица 1 – Задача

|  |  |
| --- | --- |
| № варианта | Задача |
| 13 | Реализовать обмен текстовыми сообщениями между несколькими процессами. Обеспечить возможность отправки сообщения сразу нескольким адресатам. Реализовать подтверждение приёма сообщения адресатом или, в случае потери сообщения, повторную его передачу. |

# **Задание 2.1. Разделяемая память**

Результаты выполнения представлены на рисунке 1:

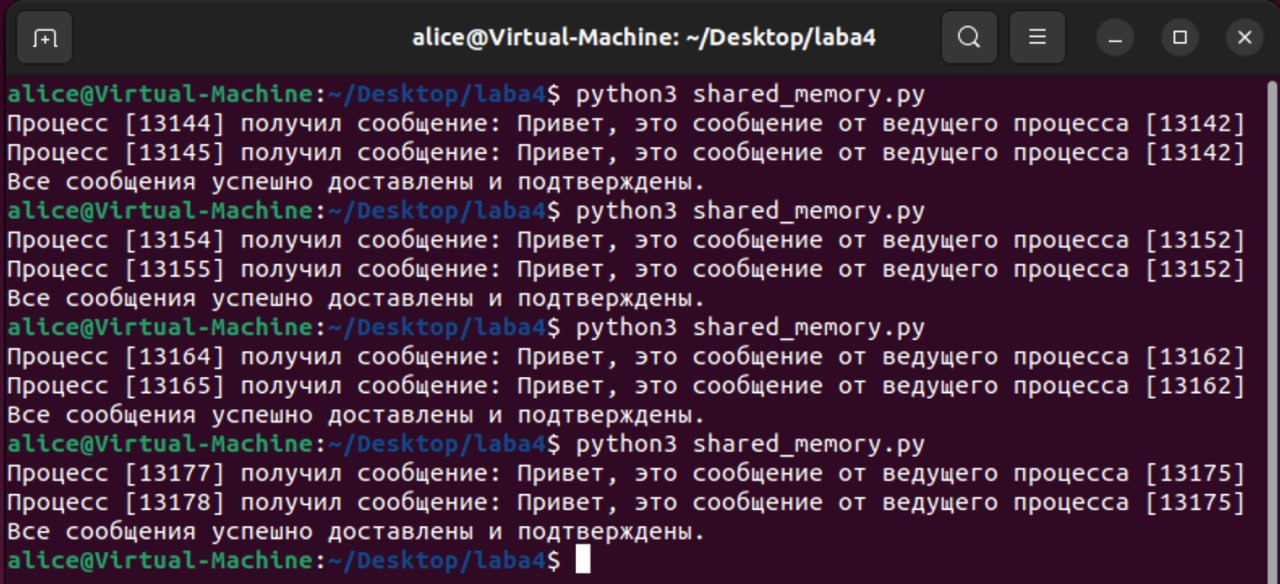


Рисунок 1 – Результаты работы программы “shared\_memory.py”

Текст программы “shared\_memory.py”

from multiprocessing import shared\_memory, Process

import os

import time

message\_size = 256  # Размер сообщения в байтах

# Функция для отправки сообщений

def send\_message(shared\_data, recipients, message):

    message\_bytes = f"{message} [{os.getpid()}]".encode() # Формируем сообщение

    for recipient in recipients: # Для каждого адресата

        offset = recipient \* message\_size # Вычисляем смещение в разделяемой памяти на основе размера сообщения и номера адресата

        shared\_data.buf[offset:offset+len(message\_bytes)] = message\_bytes # Записываем сообщение в разделяемую память

        # Добавляем нулевые байты, чтобы очистить оставшееся место в разделяемой памяти

        shared\_data.buf[offset+len(message\_bytes):offset+message\_size] = b'\0' \* (message\_size - len(message\_bytes))

# Функция для приёма сообщений

def receive\_message(shared\_data, confirmation\_data, recipient):

    offset = recipient \* message\_size # Вычисляем смещение для данного адресата в разделяемой памяти

    message\_bytes = shared\_data.buf[offset:offset+message\_size].tobytes() # Извлекаем байты сообщения из разделяемой памяти на основе смещения и размера сообщения

    message = message\_bytes.split(b'\0', 1)[0].decode() # Декодируем байты сообщения, разделяемые символов '\0', чтобы получить текст сообщения

    if message: # Если сообщение непустое

        print(f"Процесс [{os.getpid()}] получил сообщение: {message}") # Выводим сообщение с PID

        shared\_data.buf[offset:offset+message\_size] = b'\0' \* message\_size # Очищаем разделяемую память

        confirmation\_data.buf[recipient] = 1 # Устанавливаем подтверждение в разделяемой памяти для данного адресата

if \_\_name\_\_ == '\_\_main\_\_':

    num\_processes = 2 # Количество процессов

    recipients = [0, 1] # Список адресатов

    # Создаем разделяемую память для сообщений

    shared\_data = shared\_memory.SharedMemory(create=True, size=message\_size \* num\_processes)

    # Создаем разделяемую память для подтверждений

    confirmation\_data = shared\_memory.SharedMemory(create=True, size=num\_processes)

    # Инициализируем процессы-ведомые

    processes = []

    for recipient in recipients: # Для каждого адресата

        # Создаём процесс с целью receive\_message (каждый процесс получает доступ к разделяемой памяти)

        process = Process(target=receive\_message, args=(shared\_data, confirmation\_data, recipient))

        process.start()

        processes.append(process)

    # Отправляем сообщение сразу нескольким адресатам

    send\_message(shared\_data, recipients, f"Привет, это сообщение от ведущего процесса")

    # Ожидаем подтверждения приема сообщений

    while not all(confirmation\_data.buf):

        time.sleep(1)

    print("Все сообщения успешно доставлены и подтверждены.")

    # Останавливаем процессы-ведомые

    for process in processes:

        process.terminate()

    # Закрываем разделяемую память

    shared\_data.close() # Закрытие объекта разделяемой памяти

    shared\_data.unlink() # Удаление объекта разделяемой памяти

    confirmation\_data.close() # Закрытие объекта разделяемой памяти

    confirmation\_data.unlink() # Удаление объекта разделяемой памяти

# **Задание 2.2. Сокеты**

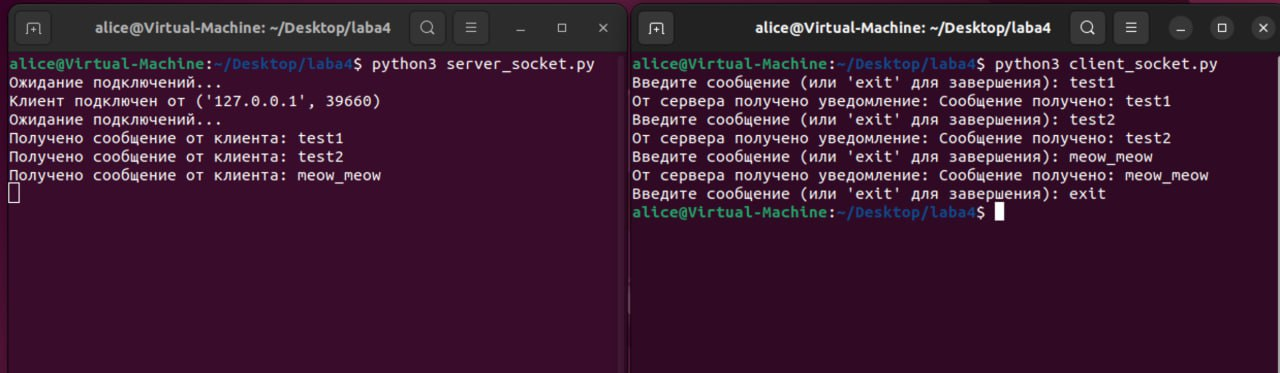
Результаты выполнения представлены на рисунке 2:

Рисунок 2 – Результаты работы программы “server\_socket.py” и “client\_socket.py”

Текст программы “ server\_socket.py ”

# import socket

# # Создаем сокет для сервера

# server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# server\_socket.bind(("localhost", 12345)) # 127.0.0.1 = localhost

# server\_socket.listen(1)  # Ожидание только одного клиента

# print("Ожидание подключения клиента...")

# client\_socket, addr = server\_socket.accept()

# print("Клиент подключен от", addr)

# while True:

#     data = client\_socket.recv(1024) # Приём сообщения от клиента

#     if not data:

#         break

#     message = data.decode('utf-8') # Декодинг строки из байтовой в текстовую

#     print("Получено сообщение от клиента:", message)

# server\_socket.close()

import socket

import multiprocessing

# Функция обработки входящих соединений

def handle\_client(client\_socket):

    while True:

        try:

            data = client\_socket.recv(1024)

            if not data:

                break

            # Получение сообщения

            message = data.decode('utf-8')

            print("Получено сообщение от клиента:", message)

            # Подтверждение получения клиента

            confirmation\_message = "Сообщение получено: " + message

            client\_socket.send(confirmation\_message.encode('utf-8'))

        except ConnectionResetError:

            break

    client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

    # Создаем сокет для сервера

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server\_socket.bind(("localhost", 12345)) # 127.0.0.1 = localhost

    server\_socket.listen(2)  # Ожидание двух клиентов

    while True:

        print("Ожидание подключений...")

        client\_socket, addr = server\_socket.accept()

        print("Клиент подключен от", addr)

        # Создаем новый процесс для обработки клиента

        client\_process = multiprocessing.Process(target=handle\_client, args=(client\_socket,))

        client\_process.start()

Текст программы “ client\_socket.py ”

# import socket

# # Создаем сокет для клиента

# client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# # Подключаемся к серверу

# client\_socket.connect(("localhost", 12345)) # 127.0.0.1 = localhost

# while True:

#     message = input("Введите сообщение: ")

#     if message == 'exit':

#             break

#     client\_socket.send(message.encode('utf-8'))

import socket

if \_\_name\_\_ == "\_\_main\_\_":

    # Создаем сокет для клиента

    client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    client\_socket.connect(("localhost", 12345)) # 127.0.0.1 = localhost

    while True:

        message = input("Введите сообщение (или 'exit' для завершения): ")

        if message == 'exit':

            break

        client\_socket.send(message.encode('utf-8'))

        # Подтверждение получения сообщения сервером

        confirmation\_message = client\_socket.recv(1024)

        response = confirmation\_message.decode('utf-8')

        print("От сервера получено уведомление:", response)

# **Задание 2.3. Каналы**

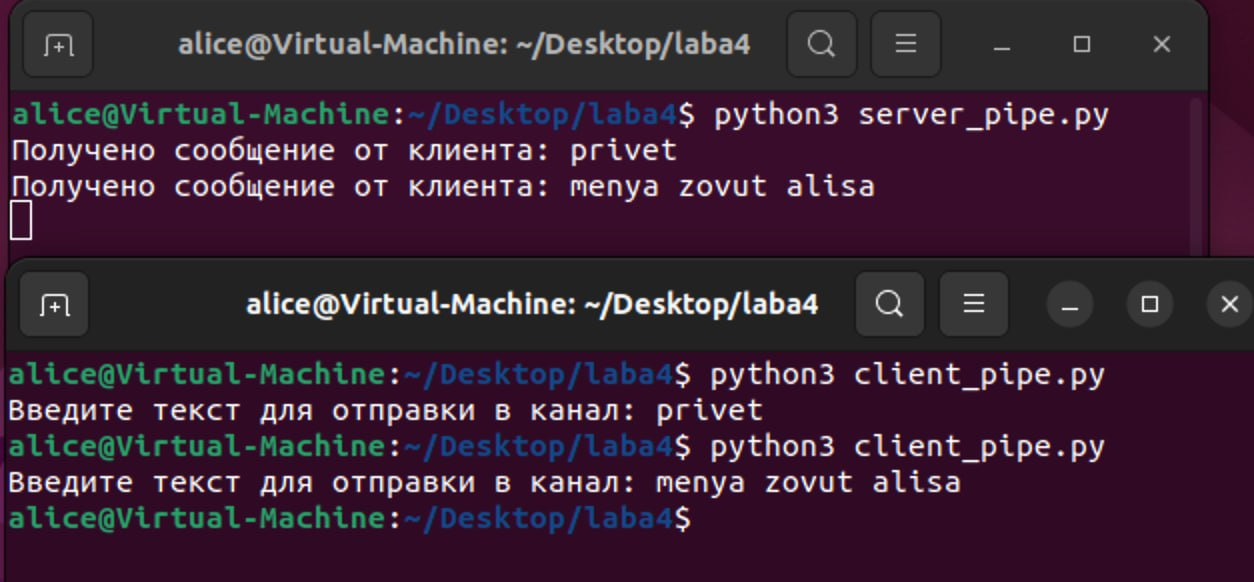
Результаты выполнения представлены на рисунке 3:

Рисунок 2 – Результаты работы программы “server\_pipe.py” и “client\_ pipe.py”

Текст программы “ server\_pipe.py ”

import os

def main():

    # Проверяем, существует ли канал

    if not os.path.exists("my\_pipe"):

        os.mkfifo("my\_pipe") # Создаём именованный канал, если он не существует

    while True:

        # Открываем канал для чтения

        read\_pipe = os.open("my\_pipe", os.O\_RDONLY)

        # Читаем данные из канала

        message = os.read(read\_pipe, 1024).decode('utf-8')

        print(f"Получено сообщение от клиента: {message}")

        # Закрываем канал после чтения сообщения

        os.close(read\_pipe)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Текст программы “ client\_pipe.py ”

import os

def main():

    # Проверяем, существует ли канал

    if not os.path.exists("my\_pipe"):

        os.mkfifo("my\_pipe") # Создаём именованный канал, если он не существует

    # Открываем канал для записи

    write\_pipe = os.open("my\_pipe", os.O\_WRONLY)

    # Считываем данные из stdin (ввод с клавиатуры) и отправляем в канал

    message = input("Введите текст для отправки в канал: ")

    os.write(write\_pipe, message.encode('utf-8'))

    # Закрываем канал

    os.close(write\_pipe)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# **Вывод**

В результате проделанной работы были изучены основные команды и инструменты для управления процессами, приобретены практически навыки работы со средствами мониторинга производительности в Linux и автоматизации запуска процессов, а также были изучены средства коммуникации процессов в ОС GNU/LINUX и получены практически навыки по использованию механизмов межпроцессорного взаимодействия.