МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 11
На тему: *"Організація взаємодії між процесами"*
З дисципліни: *"Операційні системи"*

Лектор:
Старший викладач
Грицай О.Д.

Виконав:
ст. гр. ПЗ-25
Шпак О.О.

Прийняв:
Старший викладач
Горечко О.М.

« ____ » _____ 2021 р.

Львів – 2021

Мета: ознайомитися зі способами міжпроцесної взаємодії. Ознайомитися з класичним прикладом взаємодії між процесами на прикладі задачі «виробник – споживач». Навчитися працювати із процесами з використанням способів міжпроцесної взаємодії, синхронізувати їхню роботу.

Теоретичні відомості

У багатьох ОС засоби, за допомогою яких забезпечується взаємна синхронізація процесів і потоків, називаються засобами міжпроцесної взаємодії – Inter Process Communications (IPC), що відображає історичну первинність поняття процес відносно поняття потік. Зазвичай до засобів IPC належать не тільки засоби міжпроцесної синхронізації, але й засоби обміну даними. Будь-яка взаємодія процесів або потоків залежить від їх синхронізації, яка полягає в узгодженні їх швидкостей через припинення потоку до настання деякої події й подальшої його активації під час настання цієї події. Синхронізація лежить в основі будь-якої взаємодії потоків, незалежно від того, чи пов'язана ця взаємодія з розподілом ресурсів або з обміном даними.

Ситуації, коли процесам доводиться взаємодіяти:

- Передавання інформації від одного процесу до іншого;
- Контроль над діяльністю процесів (наприклад, коли вони змагаються за один ресурс);
- Узгодження дій процесів (наприклад, коли один процес доставляє дані, а інший їх виводить на друк. Якщо узгодження не буде, то другий процес може почати друк раніше, ніж надійдуть дані).

Передавання інформації від одного процесу до іншого. Інформація може передаватися кількома способами:

- Колективна пам'ять;
- Канали (труби) – це псевдофайл, який один процес записує, а інший зчитує.
- Сокети – підтримувані ядром механізми, що приховують особливості середовища і дозволяють взаємодіяти процесам, як на одному комп'ютері, так і в мережі.
- Поштові скриньки (тільки у Windows) – однонапрямлені з можливістю широкомовної розсилки;
- Виклик віддаленої процедури – процес А може викликати процедуру в процесі В і отримувати назад дані.

У Windows підтримують наступні механізми міжпроцесної взаємодії:

- Clipboard
- COM

- Data Copy
- DDE
- File Mapping
- Mailslots
- Pipes
- RPC
- Windows Sockets

Pipes

Існує два типи труб для двостороннього зв'язку: анонімні труби та іменовані труби. Анонімні труби дозволяють пов'язаним процесам передавати інформацію один одному. Як правило, анонімна труба використовується для перенаправлення стандартного вводу або виводу дочірнього процесу, щоб він міг обмінюватися даними зі своїм батьківським процесом. Для обміну даними в обох напрямках (дуплексна робота) необхідно створити дві анонімні труби. Батьківський процес записує дані в одну трубку за допомогою своєї ручки запису, тоді як дочірній процес зчитує дані з цієї трубки за допомогою своєї ручки читання. Аналогічно, дочірній процес записує дані в іншу трубку і батьківський процес зчитує з нього. Анонімні труби не можуть використовуватися через мережу, а також не можуть використовуватися між спорідненими процесами. Названі труби використовуються для передачі даних між процесами, які не пов'язані між собою, і між процесами на різних комп'ютерах. Як правило, сервер з іменованим каналом створює іменовану трубу з добре відомим ім'ям або ім'ям, яке слід повідомити своїм клієнтам. Клієнтський процес з іменованою трубкою, який знає ім'я труби, може відкрити інший його кінець, за умови обмежень доступу, визначених сервером процес з іменованим каналом. Після того як і сервер, і клієнт підключилися до труби, вони можуть обмінюватися даними, виконуючи операції читання і запису на трубі. Анонімні канали забезпечують ефективний спосіб перенаправлення стандартного вводу або виводу на дочірні процеси на одному комп'ютері. Названі труби забезпечують простий інтерфейс програмування для передачі даних між двома процесами, незалежно від того, чи вони перебувають на одному комп'ютері чи по мережі.

Завдання

1. Реалізувати алгоритм моделювання заданої задачі за допомогою окремих процесів згідно індивідуального завдання.
2. Реалізувати синхронізацію роботи процесів.
3. Забезпечити зберігання результатів виконання завдання.
4. Результати виконання роботи відобразити у звіті.

Індивідуальне завдання (Варіант № 3)

Створити програму, що моделює наступну ситуацію: Процес-науковий керівник проекту пропонує виконавців проекту-дочірні процеси. Процес-керівник створює додаток-віртуальну дошку (файл), де можна генерувати ідеї для проекту. Процеси-виконавці генерують ідеї, записуючи їх на спільну дошку. На виконання даного завдання вони мають 3 хвилини, після чого процес-керівник призупиняє їхню роботу і виводить на екран усі згенеровані ідеї, нумеруючи кожну з них. Процеси-виконавці голосують за три найкращі ідеї. Після чого процес-керівник записує на дошку три найкращі ідеї і закриває роботу додатку-віртуальної дошки, зберігаючи її вміст. Реалізувати дану модуль, використовуючи пайпи (робота в межах однієї системи).

Послідовність виконання роботи:

Для генерації ідей від клієнтів використано файл з містами. Клієнт при запуску зчитує всі міста з файла у вектор. Потім після команди з сервера клієнт записує в стрічку всі свої варіанти і відправляє на сервер. Сервер повертає всі варіанти за, які отримав від клієнтів. Клієнти голосують за 3 варіанти, надсилають голоси на сервер. Потім сервер визначає 3 ідеї, які отримали найбільше голосів та виводить їх на екран.

Я займався роботою пайпів для сервера: створюю пайп для відправки повідомлень виконавцям, організовую все спілкування керівника з виконавцями, включаючи перше очікування на підключення з подальшою відправкою повідомлень виконавцям про початок роботи, дії зі списком ідей(приймання, опрацювання та відправлення виконавцям для голосування).

Рядки коду виділені жирним шрифтом.

Код програми:

```
// client.cpp
// Програма мовою C++ яка виступає клієнтом
// при комунікації процесів за допомогою
// іменованих каналів (named pipes) (FIFO)
// Created by Yana Luchyk, Oleksandr Shpak,
// Roman Chorniy, Matviy Suk on 08.12.2021
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <vector>
#include<fstream>
#include<chrono>
#include <iostream>
using namespace std;
#define MAXLINE 4096

int main()
{
```

```
    srand(time(NULL));

    int fd1;
    int fd2;

    // шлях до FIFO сервера
    char  server[] = "/tmp/fifo.srv";
    char str1[MAXLINE], str2[MAXLINE];

    int pid = getpid();

    char currentfifo[MAXLINE];
    snprintf(currentfifo, sizeof(currentfifo), "/tmp/fifo.%ld", (long) pid);
    cout << "[Client]: " << currentfifo << endl;
    mkfifo(currentfifo, 0666);

    // зчитати ідеї з файлу та покласти їх у вектор щоб потім рандомно обирати
    ifstream fin;
    fin.open("cities.txt");
    if (fin.is_open()) {
    //    cout << "Файл cities.txt успішно відкритий!" << endl;
    }
    else {
    //    cout << "Файл cities.txt не вдалося відкрити!" << endl;
       return -1;
    }

    vector<string> ideas;
    string in;
    // покласти всі ідеї у вектор
    while (!fin.eof()) {
       getline(fin, in);
       ideas.push_back(in);
    }
    fin.close();

    snprintf(str1, sizeof(str1), "%ld", (long) pid);
    // cout << str1 << endl;

    // Відправити на сервер свій pid щоб він міг сформувати адресу FIFO поточного
клієнта
    fd1 = open(server, O_WRONLY);
    write(fd1, str1, strlen(str1));
    close(fd1);

    cout << "Client is launched and waiting for command from server..." << endl;

    // Отримати з сервера команду про початок роботи і час, відведений для роботи
    fd1 = open(currentfifo,O_RDONLY);
    read(fd1, str1, MAXLINE);
    close(fd1);

    // Повідомити користувачу, що ми отримали з сервера
    printf("Received command from server: %s\n", str1);

    char * ptr;
    if ((ptr = strchr(str1, ' ')) == NULL)
     {
          fprintf(stderr, "Клієнт: неправильний запит: %s\n", str1);
     }

    *ptr++ = '\0'; /* нульовий байт вставлений в str1 на місце пробілу */
                   /* ptr після збільшення (++) вказує на час для роботи */

    // повідомити час, відведений на роботу
    int timeForWork = atoi(ptr);
```

```
    cout << "------\nTime for work: " << timeForWork << endl;

    sleep(1);

    auto start = chrono::high_resolution_clock::now();
    auto end = chrono::high_resolution_clock::now();
    auto elapsed = chrono::duration_cast<chrono::seconds>(end - start);

    string sent;    // стрічка для накопичення варіантів від клієнта
    while (1)
    {
        end = chrono::high_resolution_clock::now();
        elapsed = chrono::duration_cast<chrono::seconds>(end - start);
        int randint = rand() % (ideas.size() - 1);
        //  cout << "random number = " << randint << endl;
        snprintf(str2, sizeof(str2), "%s ",  ideas.at(randint).c_str());
        sent += str2;
        cout << "[time]: " << elapsed.count() << " " << str2 << endl;

        sleep(1);

        end = chrono::high_resolution_clock::now();
        elapsed = chrono::duration_cast<chrono::seconds>(end - start);

        if(elapsed.count() >= timeForWork) break;
    }

    // надіслати сформовну стрічку варіантів на сервер
    fd1 = open(server, O_WRONLY);
    write(fd1, sent.c_str(), strlen(sent.c_str()));
    close(fd1);

    // отримати від сервера усі можливі варіанти для голосування
    fd2 = open(currentfifo,O_RDONLY);
    read(fd2, str1, MAXLINE);
    close(fd2);

    // вивести можливі варіанти для голосування на еркан
    cout << str1 << endl;

    cout << "Vote for 3 variants you like the most: <var1> <var2> <var3>" << endl;
    cout << "Enter your votes: --> ";
    // отримати голоси
    int votes[3];
    for(int i = 0; i < 3; i++)
    {
        cin >> votes[i]; cin.get();
    }

    // сформувати стрічку з голосами
    snprintf(str1, sizeof(str1), "%d %d %d ", votes[0], votes[1], votes[2] );
    cout << str1 << endl;

    // відправити голоси клієнта на сервер
    fd2 = open(server, O_WRONLY);
    write(fd2, str1, strlen(str1));
    close(fd2);

    cout << "Votes has been sent to the server" << endl;
    cout << "End of work fo the client..." << endl;

    sleep(1);
    unlink(currentfifo);    // від'єднати пайп - клопоти клієнта, який його
створив

    return 0;
```

```
}
//------------------------------------------------------------------------------
```

// server.cpp

```
// Програма мовою C++ яка виступає сервером
// при комунікації процесів за допомогою
// іменованих каналів (named pipes) (FIFO)
// Created by Yana Luchyk, Oleksandr Shpak,
// Roman Chorniy, Matviy Suk on 08.12.2021
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include<chrono>
#include <vector>
#include <iostream>
#include <fstream>
#include <set>
#include <map>
#define TIME 10
#define MAXLINE 4096
using namespace std;
void savingVotes(map<int, int> &results, int voteValue);
void countVotesAndShowInConsole(map<int, int> &votes, const set<string>
&allDataSet);
int getTheSmallestElementIndex(map<int, int> &votes);

int main()
{
    int fd;
    // FIFO file path
    char server[] = "/tmp/fifo.srv";
    char  currentClient[MAXLINE];
    char arr1[MAXLINE], arr2[MAXLINE];

    mkfifo(server, 0666);   // створення FIFO сервера
    cout << "Server is launched..." << endl;
    cout << "Enter the number of needed clients: ";
    int count; cin >> count; cin.get();
    cout << "waiting for clients..." << endl;

    // отримати по каналу сервера pid клієнтів і занести їх у вектор клієнтів
    vector<string> clients;
    for(int i = 0; i < count; i++)
    {
        fd = open(server, O_RDONLY);
        read(fd, arr1, MAXLINE);
        //   cout << arr1 << endl;
        snprintf(currentClient, sizeof(currentClient), "/tmp/fifo.%s", arr1);
        cout << "[" << currentClient << "]: connected" << endl;
        clients.push_back(currentClient);
        close(fd);
    }
    // надіслати клієнтам команду початку роботи та час для роботи
    cout << "Message for clients --> ";
    cin >> arr2; cin. get();
    snprintf(arr1, sizeof(arr1), "%s %d", arr2, TIME);

    for(int i = 0; i < count; i++)
    {
        fd = open(clients[i].c_str(), O_WRONLY);
        write(fd, arr1, strlen(arr1)+1);
        close(fd);
```

```
    }

    // у циклі отримати від кожного клієнта стрічку
    // та додати цю стрічку до загальної стрічки, яку потім розбити на міста
    // міста додати у set а потім вивести пронумеровані міста на екран та
відправити клієнтам

    // отримати повідомлення від клієнтів
    string received;    // стрічка отриманих повідомлень від клієнтів

    fd = open(server, O_RDONLY);
    for(size_t i = 0; i < clients.size(); i++)
    {
        read(fd, arr1, MAXLINE);
        //  cout << arr1 << endl;
        received += arr1;
        memset(&arr1[0], 0, sizeof(arr1) );
    }
    close(fd);

   // розбити повідомлення на ідеї та занести їх у множину
   set<string>ideas;   // вектор отриманих ідей (без повторення)

   char * tmpArr = new char[received.length() + 1];
   strncpy(tmpArr, received.c_str(), received.length() + 1);
   //  cout << tmpArr << endl;

   // поділ за символом пробілу та занесення у вектор
   char * currentIdea = strtok(tmpArr, " ");

   do
   {
       ideas.insert(currentIdea);
       currentIdea = strtok(NULL, " ");
   }while(currentIdea != NULL);

   // сформувати стрічку для відправлення варіантів клієнтам та виведення
   // варіантів для голосуваня

   string toSend;  // накопичення номерів варіантів та їхніх значень
   toSend += "\nIdeas from clients:\n";
   set<string>::const_iterator it = ideas.begin();
   for(size_t i = 0; i < ideas.size(); i++, it++)
   {
       toSend += to_string(i + 1);
       toSend += "\t";
       toSend += *it;
       toSend += "\n";
   }

   cout << toSend << endl;

   // відправлення клієнтам списку для голосування
   for(size_t i = 0; i < clients.size(); i++)
   {
       fd = open(clients[i].c_str(), O_WRONLY);
       // Open FIFO for write only
       write(fd, toSend.c_str(), strlen(toSend.c_str()));
       close(fd);
   }
   cout << "Vote for 3 variants you like the most: <var1> <var2> <var3>" << endl;
   cout << "Waiting for votes from clients..." << endl;
   // отримати у стрічку голоси від клієнтів
   string votesStr;
   for(size_t i = 0; i < clients.size(); i++)
   {
```

```
        fd = open(server, O_RDONLY);
        read(fd, arr1, MAXLINE);
        cout << arr1 << endl;
        votesStr += arr1;
        memset(&arr1[0], 0, sizeof(arr1) );
        close(fd);
    }

    cout << "Received such votes:" << endl;
    cout << votesStr << endl;

    vector<int> votes;  // вектор голосів
    if(tmpArr) {delete [] tmpArr; tmpArr = nullptr; }

    tmpArr = new char[votesStr.length() + 1];
    strncpy(tmpArr, votesStr.c_str(), votesStr.length() + 1);

    // розбити стрічку по знаку пробілу на голоси та занести їх у вектор
    char * currentVote = strtok(tmpArr, " ");

    do
    {
        votes.push_back(atoi(currentVote)); // не забути перевести текст у число
        currentVote = strtok(NULL, " ");
    }
    while(currentVote != NULL);

    map<int, int> results;

    // занести індекси ідей у мапу і занулити 2-ге поле (к-сть голосів)
    for(size_t i = 0; i < ideas.size(); i++, it++)
    {
        results.insert({i, 0});
    }

    // цикл по кількості голосів, додати голос до потрібної ідеї
    for(size_t i = 0; i < votes.size(); i++)
    {
        savingVotes(results, votes[i]);
    }

    cout << "\n-------\nWe have counted your votes and happy to announce the
results" << endl;

    sleep(1);

    // ф-я визначає ідеї з найбільшими голосами та виводить їх на екран
    countVotesAndShowInConsole(results, ideas);

    unlink(server);  // від'єднати пайп - клопоти процесу, який його створив
    cout << "end of work of the server..." << endl;
    if(tmpArr) {delete [] tmpArr; tmpArr = nullptr; }
    sleep(1);

    return 0;
}
//--------------------------------------------------------------------------
void savingVotes(map<int, int> &results, int voteValue)
{
    map<int, int>::iterator itr;

    for(itr = results.begin(); itr != results.end(); ++itr)
    {
        if (itr->first == voteValue)
        {
            itr->second = itr->second + 1;
```

```
        }
    }
}
//--------------------------------------------------------------------------
int getTheSmallestElementIndex(map<int, int> &votes)
{
    int index = votes.begin()->first;
    int value = votes.begin()->second;
    map<int, int>::iterator itr;

    for(itr = votes.begin(); itr != votes.end(); ++itr)
    {
        if(itr->second < value)
        {
            value = itr->second;
            index = itr->first;
        }
    }
    return index;
}
//--------------------------------------------------------------------------
void countVotesAndShowInConsole(map<int, int> &votes, const set<string>
&allDataSet)
{
    map<int, int> top;
    vector<string> allDataVector;

    set<string>::const_iterator it = allDataSet.begin();
    while (it != allDataSet.end())
    {
        allDataVector.push_back(*it);
        it++;
    }

    if (votes.size() > 2)
    {
           for(int i = 0; i < 3; i++)
           {
               top.insert({ i+1, 0 });
               top.at(i+1) = votes.at(i+1);
           }

           for(size_t i = 4; i < votes.size() ; i++)
           {
               int index = getTheSmallestElementIndex(top);
               if(top.at(index) < votes.at(i))
               {
                   top.erase(index);
                   top.insert({ i, votes.at(i) });
               }
           }

           map<int, int>::iterator itr;
           cout << "Top 3:" << endl;
           for(itr = top.begin(); itr != top.end(); itr++)
           {
               cout << allDataVector[itr->first - 1] << endl;
           }
       }
    else
    {
        cout << "There is no Top 3 options!" << endl;
    }
}
//--------------------------------------------------------------------------
```

Протокол роботи програми

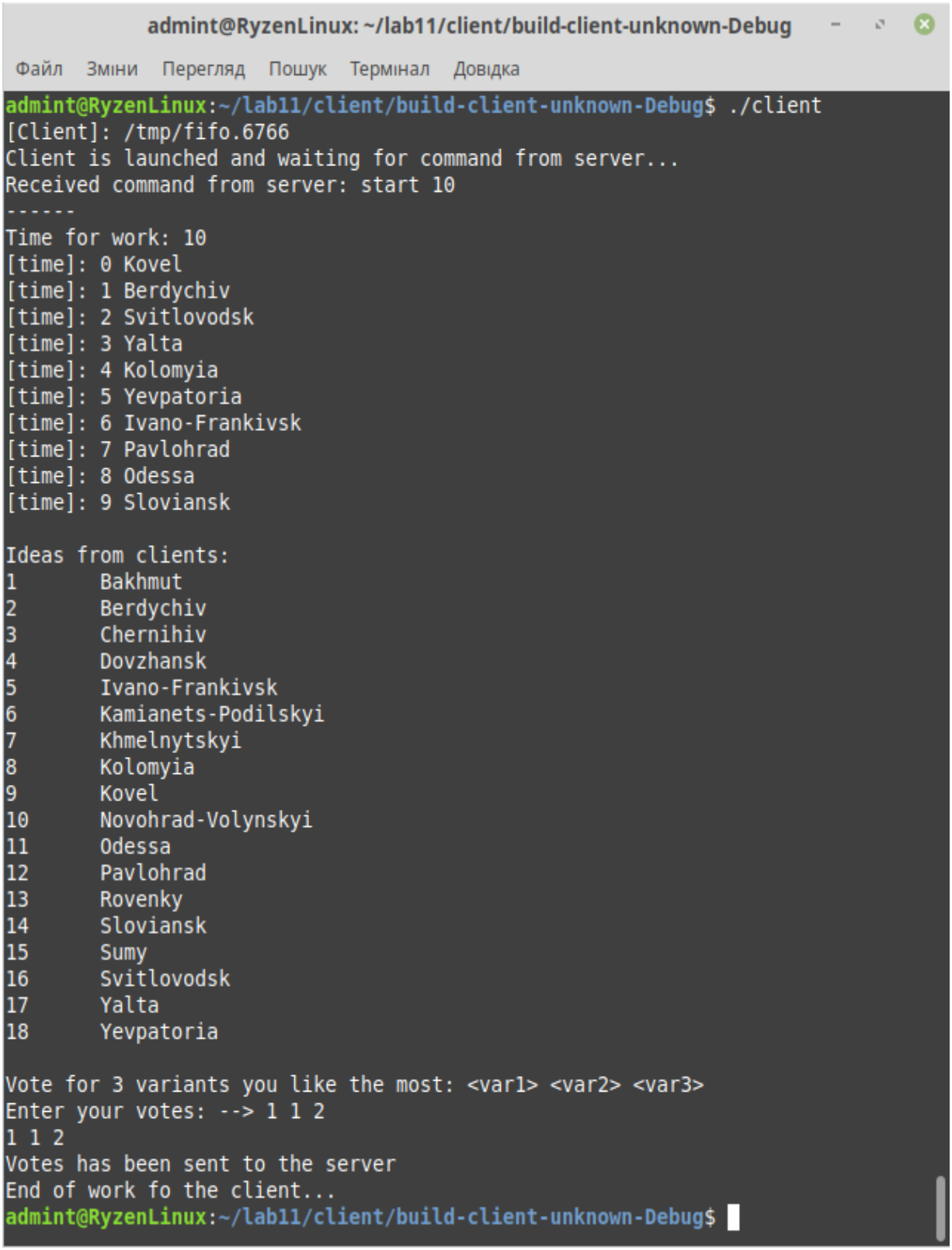

Рисунок 1. Протокол роботи клієнта

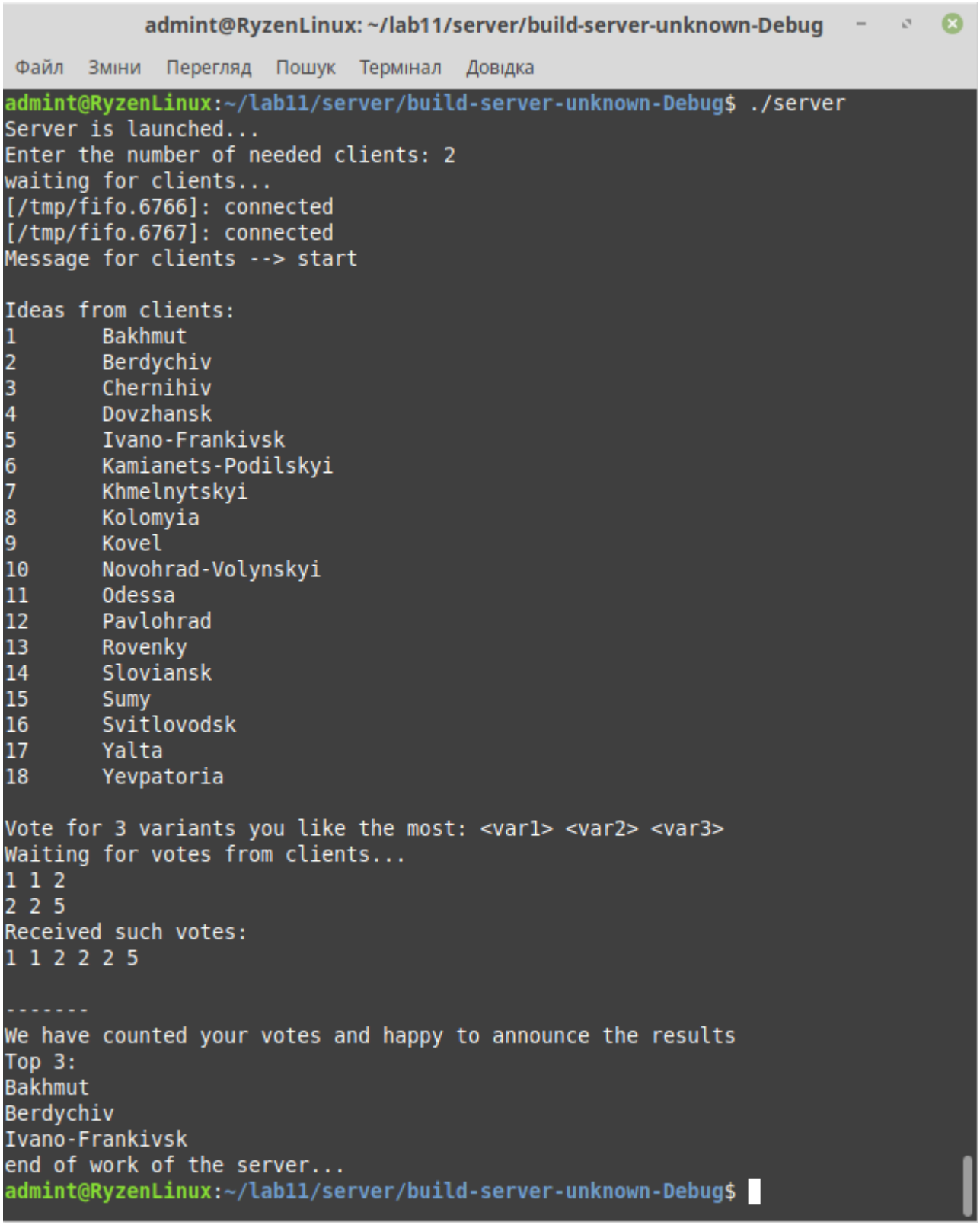

Рисунок 2. Протокол роботи сервера

Висновок

Під час виконання лабораторної роботи я ознайомився зі способами міжпроцесної взаємодії в ОС Windows та Linux. Реалізував разом з командою роботу клієнта та сервера з використанням іменованих каналів (named pipes, FIFO) під керуванням ОС Linux.