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”*

Лекторка:
стр. викл. каф. ПЗ
Грицай О. Д.

Виконав:
стд. гр. ПЗ-26
Гладкий В. В.

Прийняв:
доцент каф. ПЗ
Горечко О. М.

« ____ » _____ 2021 р.

∑= ____ _____.

Львів – 2021

Тема: Організація взаємодії між процесами.

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

Існує досить великий клас засобів ОС, за допомогою яких забезпечується взаємна синхронізація процесів і потоків. Потреба в синхронізації потоків виникає тільки в мультипрограмній ОС і залежить від спільного використання апаратних та інформаційних ресурсів обчислювальної системи. Синхронізація потрібна для запобігання перегонам та безвиході під час обміну даними між потоками,поділу даних, доступу до процесора і пристроїв введення-виведення.У багатьох ОС ці засоби називаються засобами міжпроцесної взаємодії – Inter Process Communications (IPC), що відображає історичну первинність поняття процес відносно поняття потік.

Синхронізація лежить в основі будь-якої взаємодії потоків, незалежно від того, чи пов’язана ця взаємодія з розподілом ресурсів або з обміном даними. Наприклад, потік-одержувач повинен звертатися за даними тільки після того, як вони поміщені в буфер потоком-відправником. Якщо ж потік-одержувач звернувся до даних до моменту їх надходження в буфер, то він має бути припинений. Синхронізація також потрібна у разі спільного використання апаратних ресурсів.

Існує дві основні моделі міжпроцесорної комунікації: спільна пам'ять та передача повідомлень. У моделі спільної пам’яті встановлюється область пам’яті, яка поділяється процесами співпраці. Потім процеси можуть обмінюватися інформацією, читаючи та записуючи дані в спільний регіон. У моделі передачі повідомлень, спілкування відбувається за допомогою повідомлень, що обмінюються між взаємодіючими процесами.

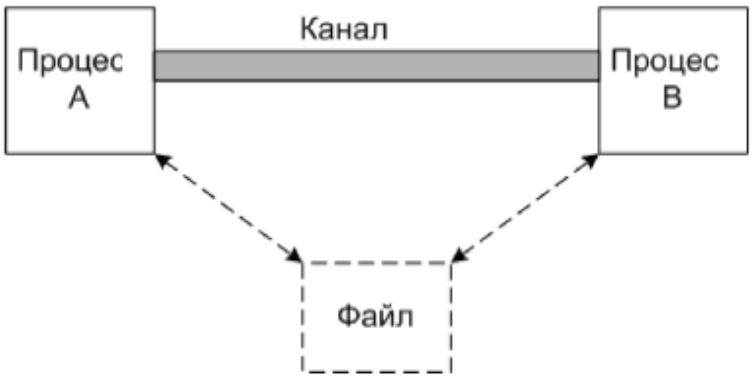
Ситуації, коли процесам доводиться взаємодіяти;

- 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.

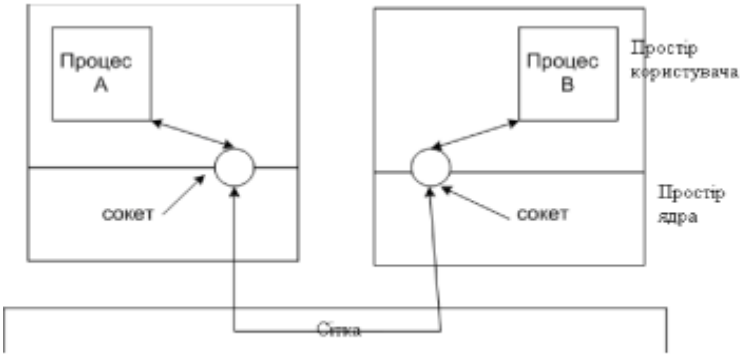
Якщо узгодженості не буде, то другий процес може почати друк раніше, ніж надійдуть дані).

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.

Міжпроцесовий зв’язок із використанням спільної пам'яті вимагає комунікаційних процесів для встановлення області спільної пам'яті. Зазвичай область спільної пам’яті міститься в адресному просторі процесу, створюючи сегмент спільної пам’яті. Як правило, операційна система намагається запобігти доступу одного процесу до пам’яті іншого процесу. Загальна пам’ять вимагає, щоб два більше обробляли процес,

щоб зменшити це обмеження. Потім вони можуть обмінюватися інформацією, читаючи та записуючи спільні ділянки.

У WINDOWS підтримуються 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 названі 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. 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. 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.

Windows Sockets

Windows Sockets - це незалежний від протоколу інтерфейс. Він використовує переваги комунікаційних можливостей базових протоколів. У Windows Sockets 2 дескриптор сокету може додатково використовуватися як файловий дескриптор зі стандартними функціями вводу / виводу файлів. Сокети Windows засновані на сокетах, вперше популяризованих Berkeley Software Distribution (BSD). Додаток, що використовує Windows Sockets, може спілкуватися з іншою реалізацією сокета в інших типах систем. Однак не всі постачальники транспортних послуг підтримують усі доступні варіанти.

File Mapping

Відображення файлів (File mapping)- це об'єднання вмісту файлу з частиною віртуального адресного простору процесу. Відображення файлів дозволяє процесу використовувати як випадковий вхід і вихід (I/O) , так і послідовні (I/O) . Це також дозволяє ефективно працювати з великим файлом даних, таким як база даних, без

необхідності зіставляти весь файл у пам'ять. Кілька процесів також можуть використовувати файли, зібрані в пам'яті, для обміну даними. Процеси зчитують та записують у file view використовуючи вказівники так само, як і з динамічно виділеною пам'яттю. Використання відображення файлів підвищує ефективність, оскільки файл знаходиться на диску, але file view знаходиться в пам'яті.

Лабораторне 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 - 1

Реалізувати міжпроцесну взаємодію одним із відомих вам методів. Один із процесів має бути сервером, який дозволяє процесам-клієнтам підписатись/відписатись на один із сервісів розсилки (щогодинний прогноз погоди, щохвилинний курс акцій, щоденний курс валют). Для збереження інформації на сервері можна використати бази даних. Реалізувати дану модель, використовуючи, що проектуються у пам’ять або пайпи (робота в межах однієї системи).

Склад команди

Андрущак Святослав – реалізація серверу для стягування курсу криптовалют та відправленню клієнтам цього сервера

Берещак Ігор – створення процесу-клієнта який приймає дані з сервера та виводить і зберігає їх

Гладкий Владислав – створення головної програми для запуску сервера та його клієнтів для їхньої взаємодії

Емірова Аліє – реалізація інтерфейсу для серверу та клієнта , створення та редагування звіту

Посилання на GitHub проекту - https://github.com/senoron/Lab_11_OS

Посилання на сайт з акціями - https://coinmarketcap.com

Код програми

Код клієнта:

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <fstream>
#include <iostream>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>

#include "..//..//constants.h"

#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>

#define CountMsg 5
using namespace std;

struct json{
    double BTC;
    double ETH;
    double SOL;
    double ADA;
    double DOGE;
    double MATIC;
    double BCH;
    double LTC;

    QString update;
};


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

struct mystruct{
    Ui::MainWindow *ui;
    QList <json*> listJson;
};

class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private slots:

    void on_updates_currentIndexChanged(int index);

public:
    Ui::MainWindow *ui;
    mystruct *clientStruct;
};


#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>

void* client( void* Mystruct);

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    pthread_t HANDLE;
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    ui->setupUi(this);

    clientStruct=new mystruct();
    clientStruct->ui=ui;

    pthread_create(&HANDLE,& attr,client,(void*)clientStruct);

}

MainWindow::~MainWindow()
{
    delete ui;
}


void* client( void* Mystruct)
{
    qDebug("enter");
    mystruct *st= (mystruct*)(Mystruct);

    Ui::MainWindow *ui=st->ui;

    int handleFile;
    void* pBuf;
    sem_t* semptr = sem_open(semName, O_CREAT ,  0666 , 0);
    int temp = -2;

    sem_getvalue(semptr , &temp);
    printf("value sem : %d\n" , temp);

    handleFile = shm_open(memoryName , O_RDWR , 0666);
    pBuf = mmap(NULL , sharedSize , PROT_WRITE | PROT_READ , MAP_SHARED , handleFile , 0);
```

```
    int i = 0;
    while(1)
    {
        json *lastUpdate=new json();
        cout<<("I wait\n");

        sem_wait(semptr);

        sem_getvalue(semptr , &temp);
        printf("value sem : %d\n" , temp);

        QString str=QString::fromUtf8((char*)pBuf);
        QJsonDocument jsondocument = QJsonDocument::fromJson(str.toUtf8());
        lastUpdate->update=QString::fromStdString(
jsondocument.object().value("lastUpdated").toString().toStdString());
        ui->label_update->setText(lastUpdate->update);;

        //ui->label->setText(QString::fromStdString(
jsondocument.object().value("lastUpdated").toString().toStdString()));
        QJsonObject details= jsondocument.object().value("detail").toObject();

        lastUpdate->ADA=details.value("ADA").toDouble();
        lastUpdate->BTC=details.value("BTC").toDouble();
        lastUpdate->DOGE=details.value("DOGE").toDouble();
        lastUpdate->ETH=details.value("ETH").toDouble();
        lastUpdate->LTC=details.value("LTC").toDouble();
        lastUpdate->MATIC=details.value("MATIC").toDouble();
        lastUpdate->BCH=details.value("BCH").toDouble();
        lastUpdate->SOL=details.value("SOL").toDouble();

        st->listJson.push_back(lastUpdate);
        sem_post(semptr);
        sem_getvalue(semptr , &temp);
        printf("value sem : %d\n" , temp);

        ui->lineEdit_ada->setText(QString::number(lastUpdate->ADA));
        ui->lineEdit_btc->setText(QString::number(lastUpdate->BTC));
        ui->lineEdit_doge->setText(QString::number(lastUpdate->DOGE));
        ui->lineEdit_eth->setText(QString::number(lastUpdate->ETH));
        ui->lineEdit_ltc->setText(QString::number(lastUpdate->LTC));
        ui->lineEdit_matic->setText(QString::number(lastUpdate->MATIC));
        ui->lineEdit_bch->setText(QString::number(lastUpdate->BCH));
        ui->lineEdit_sol->setText(QString::number(lastUpdate->SOL));

        ui->updates->addItem(lastUpdate->update.split(" ")[4]);

        sleep(8);

        if(i == 5)
            break;
    }

    close(handleFile);
    sem_close(semptr);
    shm_unlink(memoryName);

    qDebug("enter");

    return 0;
}



void MainWindow::on_updates_currentIndexChanged(int index)
{
    ui->lineEdit_ada->setText(QString::number(clientStruct->listJson[index]->ADA));
    ui->lineEdit_btc->setText(QString::number(clientStruct->listJson[index]->BTC));
    ui->lineEdit_doge->setText(QString::number(clientStruct->listJson[index]->DOGE));
    ui->lineEdit_eth->setText(QString::number(clientStruct->listJson[index]->ETH));
    ui->lineEdit_ltc->setText(QString::number(clientStruct->listJson[index]->LTC));
    ui->lineEdit_matic->setText(QString::number(clientStruct->listJson[index]->MATIC));
```

```
    ui->lineEdit_bch->setText(QString::number(clientStruct->listJson[index]->BCH));
    ui->lineEdit_sol->setText(QString::number(clientStruct->listJson[index]->SOL));

    ui->label_update->setText(clientStruct->listJson[index]->update);
}
```

Код сервера:

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <fstream>
#include <iostream>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>

#include "..//..//constants.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fstream>

using namespace std;

string pricesFileName = "Prices.json";
pid_t runner;
sem_t* semptr;
int handleFile;
```

```
void* server( void* Mystruct);

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    runner = fork(); // creating process for auto updating crypto currencies
    if(runner == 0){
        char* args[2];
        args[0] = "..//..//server//crypto-runner";
        args[1] = NULL;

        execvp(args[0], args);
    }

    pthread_t HANDLE;
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    pthread_create(&HANDLE,& attr, server, NULL);
}

MainWindow::~MainWindow()
{
    kill(runner, 200);
    sem_close(semptr);
    shm_unlink(memoryName);
    delete ui;
}

void* server( void* Mystruct)
{
        void* pBuf;

        semptr = sem_open(semName, O_CREAT ,  0666 , 1); // creating semaphore
        int temp = -2;

        sem_getvalue(semptr , &temp);
        printf("value sem : %d\n" , temp);

        if(semptr == SEM_FAILED) { // failed to create the semaphore
            printf("Error with pBuf");
        }


        handleFile = shm_open(memoryName , O_CREAT | O_RDWR , 0666); // creating sharing
memory
        ftruncate(handleFile, sharedSize); // cutting file size

        pBuf = mmap(NULL , sharedSize , PROT_WRITE | PROT_READ , MAP_SHARED , handleFile , 0);
// getting pointer to shring mem

        if(!pBuf) { // error with pointer
            printf("Error with pBuf");
        }

        int i = 0;

        int longSleep = 15; // waiting for crypto to be updated
        int shortSleep = 5; // waiting for clients to read the mem

        while(1)
        {
            sem_wait(semptr);

            sem_getvalue(semptr , &temp);
            printf("value sem : %d\n" , temp);
```

```
            sleep(longSleep);

            ifstream file; // reating from file with updated crypto
            file.open(pricesFileName);
            string str = "";
            getline(file, str);
            file.close();

            printf("Writing info to clients");

            const char * mess = str.c_str(); // string to char*

            sprintf((char*)pBuf , "%s" , mess); // writing to memory

            sem_post(semptr);

            sem_getvalue(semptr , &temp);
            printf("value sem : %d\n" , temp);

            sleep(shortSleep); // time for clients to read the memory
        }


        close(handleFile);
}
```

Цю частину коду реалізував Гладкий Владислав

Основна програма(ранер):

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <stdlib.h>
#include<unistd.h>
#include<errno.h>
#include <sys/types.h>
#include <sys/times.h>
#include <time.h>
#include<sys/wait.h>
#include<signal.h>
#include<sys/resource.h>
#include "vector"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

```
private slots:
    void on_pushButton_stop_clicked();
    void on_pushButton_run_serveer_clicked();
    void on_pushButton_clicked();
    void on_pushButton_filltable_clicked();

private:
    Ui::MainWindow *ui;

    int CountClients = 0;
    pid_t idServer = -1;
    std::vector<pid_t> idClients;

    QString StatusServer = "Running";

    char* argvServer[2] = {"..//..//server-qt//build-server-qt-Desktop-Debug//server-
qt" , NULL};
    char* argvClient[2] = {"..//..//client//build-client-Desktop-Debug//client" ,
NULL};
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->lineEdit_state->setText("Not running");
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_run_serveer_clicked()
{
    pid_t pid = fork();

    if(pid == 0)  //Call server`s process
    {
        qDebug() << "I am child-server";
        execvp(argvServer[0] , argvServer);
    }

    if(pid > 0)  //Saving server`s pid
    {
        idServer = pid;
        ui->lineEdit_state->setText(StatusServer);
    }
}
```

```
void MainWindow::on_pushButton_clicked()
{
    CountClients = ui->spinBox_clients->value();
    idClients.clear();  //Clearing array with proceses` id

    for (int i = 0 ; i < CountClients ; i++ )
    {
        pid_t pid = fork();

        if(pid == 0)  //Call client`s process
        {
            qDebug() << "I am child-client";
            execvp(argvClient[0] , argvServer);
        }

        if(pid > 0)  //Saving client`s pid
        {
            idClients.push_back(pid);
        }
    }
}


void MainWindow::on_pushButton_filltable_clicked()
{
    ui->list_client->clear();
    for (int i = 0 ; i < CountClients ; i++ )  //Filling of table
    {
        ui->list_client->addItem("Client " + QString :: number(i + 1) + " , id : " +
QString :: number(idClients[i]));
    }
}


void MainWindow::on_pushButton_stop_clicked() //Completion of all processes
{
    for (int i = 0 ; i < CountClients ; i++ )
    {
        kill(idClients[i] , SIGKILL);
    }

    if(idServer != -1)
    {
        kill(idServer , SIGKILL);
    }

    CountClients = 0;
    idClients.clear();
    StatusServer = "Killed";

    ui->lineEdit_state->setText(StatusServer);
}
```

Фотозвіт

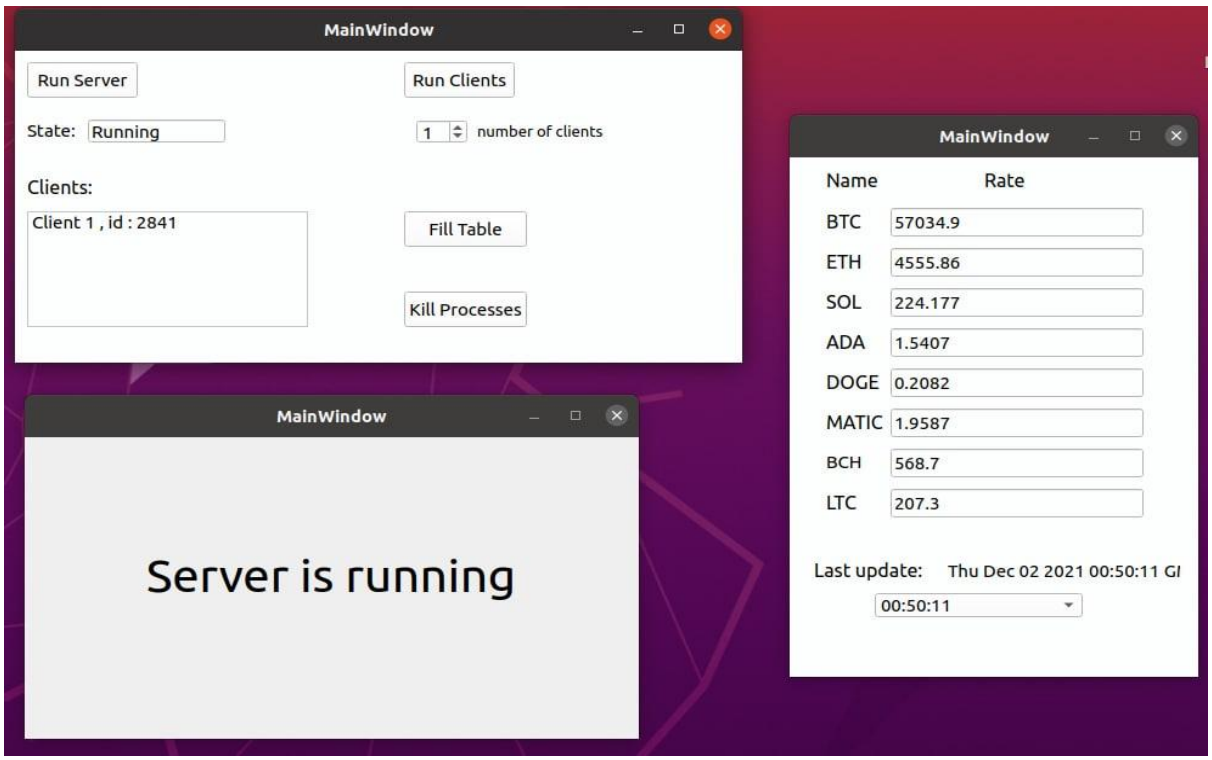


Рис.1 Результат роботи програми з одним процесом-клієнтом.

Рис.2 Знищення процесів сервера та клієнтів.

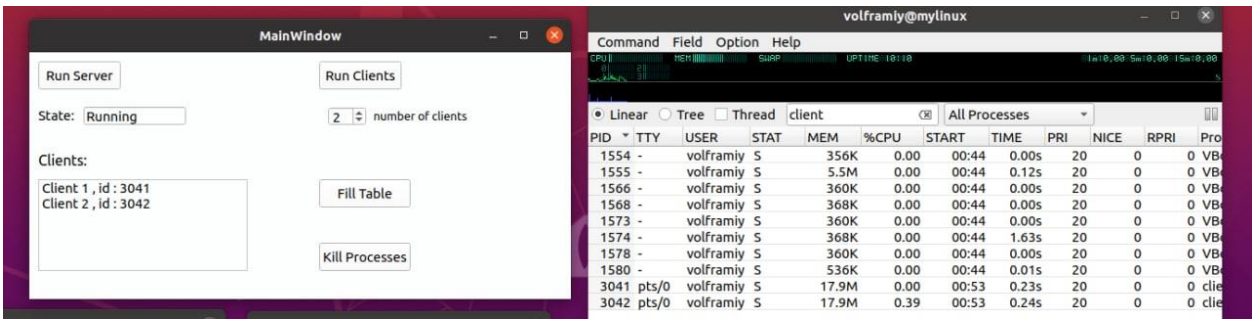


Рис.3 Створення процесів-клієнтів.

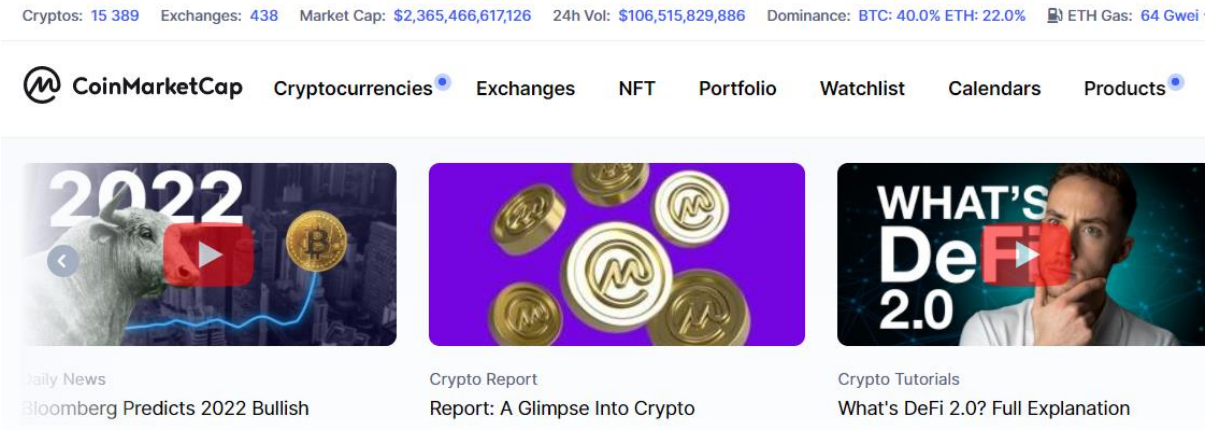


Рис.4 Сайт з якого стягувалась інформація для сервера

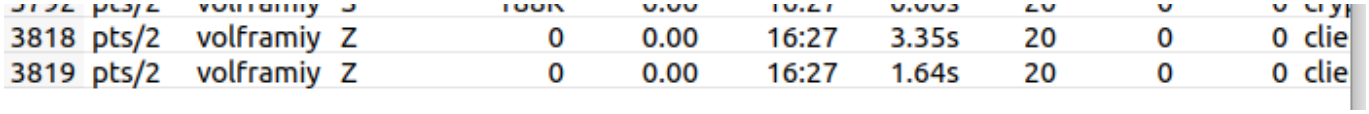


Рис.5 Клієнти після від'єднання від сервера(стан зомбі)

Висновки

На даній лабораторній роботі я ознайомився зі способами міжпроцесної взаємодії, а саме з класичним прикладом взаємодії між процесами на прикладі задачі «виробник – споживач». Навчився працювати із процесами з використанням способів міжпроцесної взаємодії, синхронізувати їхню роботу. Зокрема розробив свою частину лабораторної , а саме реалізував процес для запуску сервера та клієнтів.