Proiect Nr. 9

Studenti: Dulea Mihai & Lefter Madalin
Îndrumător: Prof. Dr. Ing. Adrian Florea

Tema Proiectului **3**
Functionalitati **3**
Descrierea simulatorului **3**
Client Side Vs Server Side **4**
Clasa SimData **5**
Clasa Consumer **7**
Main Window **8**

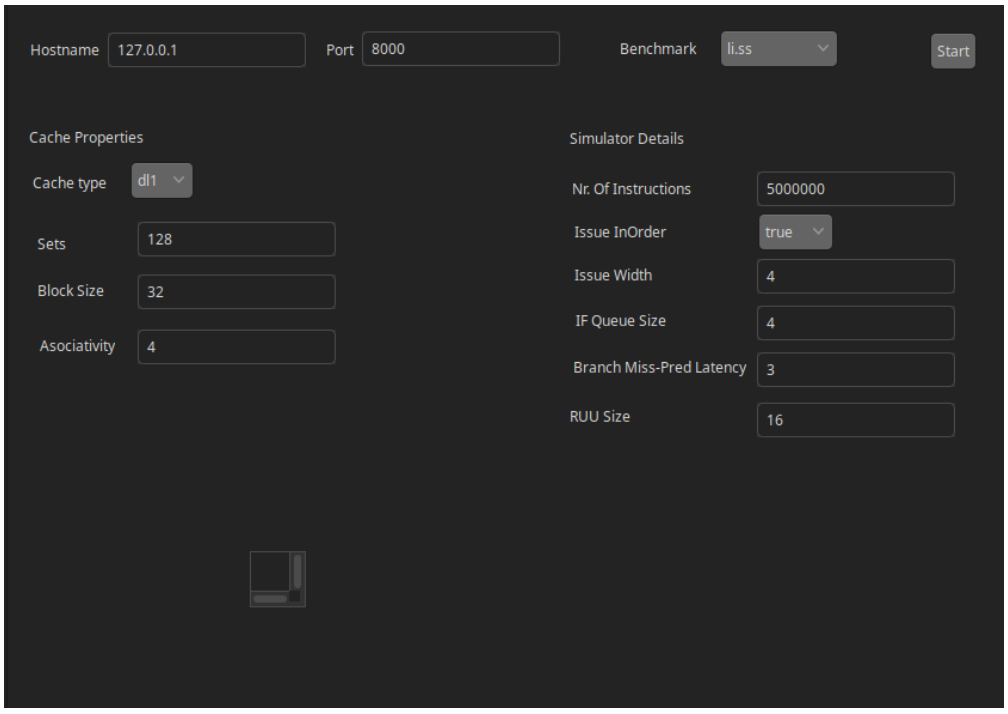
Tema Proiectului:

Implementarea unei interfeţe vizuale aferente simulatorului sim-outorder care simulează execuţia out-of-order, interfaţa procesor-cache, predictor avansat de salturi. Se va permite introducerea parametrilor de editare, simularea în reţea în arhitectură client / sever, implementarea unui help ataşat, prezentarea grafica a rezultatelor simulării IR(fetch rate), IR(dimensiunea cache-ului)

Functionalitati:

- Interfata grafica simpla
- Alegerea parametrilor doriti
- Rularea benchmark-urilor in retea
- Afisarea rezultatului benchmark-ului in program

Descrierea simulatorului:



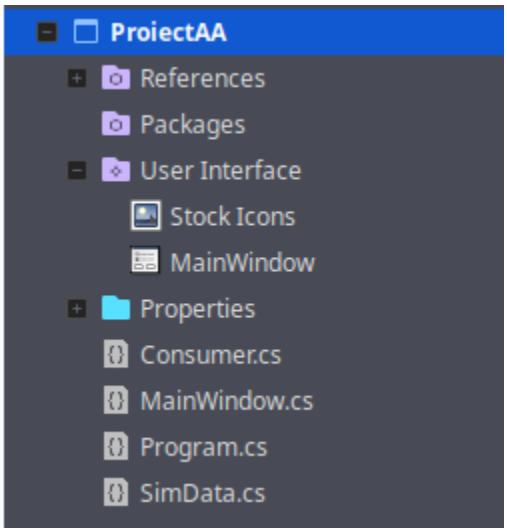
- In partea de sus putem alege Hostname-ul si Port-ul server-ului unde ar trebui sa ruleze benchmark-urile dorite
- Tot in partea de sus putem alege ce benchmark sa rulam dupa care sa dam start daca consideram ca am ales toate detaliile legate de simulator si cache
- In partea din stanga si dreapta putem selecta proprietatile Cache-ului dar si ale simulatorului dupa bunul plac
- Dupa ce se va apasa pe butonul start in textbox-ul de jos vor aparea rezultatele benchmark-ului

Client Side Vs Server Side

Pentru a putea realiza o rulare a benchmark-urilor in rete, a trebuit sa realizam 2 proiecte, unul pentru partea de server unde server-ul o sa ruleze si o sa capteze inputul de la client, iar un proiect pentru client, unde se va afla UI-ul si unde vor fi afisate rezultatele.

Toate calculele computationale vor fi realizate pe partea de server pentru a nu pune host-ul la munca grea.

Client Side



Partea de client este construita pe baza a 2 clase: **SimData** si **Consumer.**

SimData contine detaliile legate de simulator. Cand utilizatorul apasa pe butonul de start **SimData** ar trebui sa capteze toate rezultatele introduse de utilizator din UI.

Consumer contine detaliile legate de Client Side, mai exact procesul de trimitere a liniei de comanda pentru rularea benchmark-ului, respectiv preluarea output-ului.

Main Form-ul contine event-urile si partea de conectare la server

Clasa SimData

- aceasta in principiu ar trebui sa contine Getters&Setters pentru fiecare field din UI, care ulterior vor fi folosite la construirea liniei de comanda

```
//BENCHMARK
public string Benchmark { get; set; }

//SIM GENERAL
public string NrInstructions { get; set; }
public string IssueInOrder { get; set; }
public string IssueWidth { get; set; }
public string IFQsize { get; set; }
public string Mplat { get; set; }
public string RUU { get; set; }
public string LSQ { get; set; }

//CACHE
public string CacheType { get; set; }
public string CacheSets { get; set; }
public string CacheBlockSize { get; set; }
public string CacheAsoc { get; set; }
```

- contine o metoda pentru returnarea detaliilor legate de cache in formatul dorit de simulator

```
private string getCacheDetails()
{
    return $"{CacheType}:{CacheSets}:{CacheBlockSize}:{CacheAsoc}:1";
}
```

- contine si metoda care returneaza linia de comanda care ulterior va fi trimisa catre server pentru rulare

```
public string getCommand() {
    string command = "-redir:sim {OutputFile}.res -max:inst {NrInstructions} -cache:{CacheType}

    command = command.Replace("{NrInstructions}", NrInstructions)
        .Replace("{CacheType}", CacheType)
        .Replace("{CacheDetails}", getCacheDetails())
        .Replace("{IFQsize}", IFQsize)
        .Replace("{Mplat}", Mplat)
        .Replace("{IssueInOrder}", IssueInOrder)
        .Replace("{IssueWidth}", IssueWidth)
        .Replace("{RUU}", RUU)
        .Replace("{benchmark}", Benchmark);

    return command;
}
```

- linia de comanda are formatul urmator (valorile dintre acolade fiind inlocuite cu valorile necesare):

```
string command = "-redir:sim {OutputFile}.res -max:inst
{NrInstructions} -cache:{CacheType} {CacheDetails} -fetch:ifqsize
{IFQsize} -fetch:mplat {Mplat} -issue:inorder {IssueInOrder}
-issue:width {IssueWidth} -ruu:size {RUU}
{benchmark_path}/{benchmark}";
```

Clasa Consumer

- aceasta clasa ar trebui sa contina logica legata de comunicatia client catre server, adica mai exact pasii de conectare, deconectare, trimitere de comenzi si preluare de rezultat dupa ce a fost trimisa o comanda

- metoda **start_connction** creeaza conxiunea dintre client si server bazate pe hostname-ul si port-ul introduse de utilizator in UI.

```
public static void close_connection()
{
    client.Close();
    stream.Close();
}
```

- metoda **close_connection** inchide conxiunea si stream-ul de citire de date dintre client si server

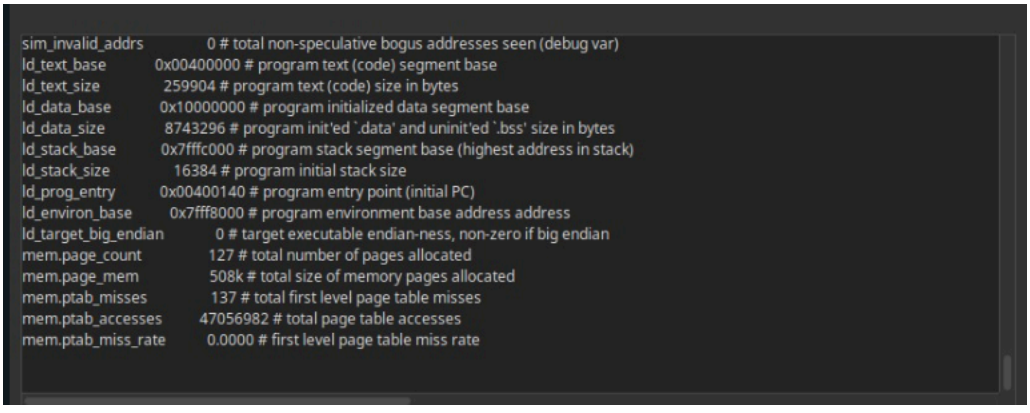
```
public static void send_command_to_server(string command)
{
    streamWriter.WriteLine(command);
    streamWriter.Flush();
}
```

- metoda **send_command_to_server** are ca parametru un string care ar trebui sa reprezinte linia de comanda trimisa catre server pentru a rula un anume benchmark

```
public static string get_output_from_server()
{
    string output = "";
    output = streamReader.ReadToEnd();

    return output;
}
```

- **metoda get_output_from_server** ia output-ul trimis de catre server si il afiseaza in textbox-ul principal din view



Main Window

- contine 2 event-uri foarte importante : eventul pentru start button si eventul pentru afisarea datelor default pentru fiecare tip de cache

Eventul de start button este putin mai complex asa ca il vom lua pe bucati

- in momentul in care utilizatorul apasa pe buton de start, acest event va fi triggaruit si va rula urmatoarele seturi de comenzi:

```
//Start Conncetion with the details from UI
serverOutput.Buffer.Clear();
sim_data = new SimData();

string hostname = hostname_textbox.Text;
int port = Convert.ToInt32(port_textbox.Text);
Consumer.start_connection(hostname, port);
```

- in primul rand va goli buffer-ul textbox-ului pentru a ne asigura ca este gol inainte de a pune informatii in el
- dupa care ne vom crea o noua instanta de SImData unde vom plasa detaliile legate de simulator
- vom prelua hostname si port din textbox-uri si vom crea conexiunea cu serverul

```
//Get Details from text boxes
//Benchmark
sim_data.Benchmark = benchmark_combobox.ActiveText;


//Get Cache Details
sim_data.CacheType = cache_type_combobox.ActiveText;
sim_data.CacheSets = sets_textbox.Text;
sim_data.CacheBlockSize = blocksize_textbox.Text;
sim_data.CacheAsoc = asoc_textbox.Text;


//Get Sim Details
sim_data.NrInstructions = nr_instructions_textbox.Text;
sim_data.IssueInOrder = issue_inorder_option.ActiveText;
sim_data.IssueWidth = issue_width_textbox.Text;
sim_data.IFQsize = if_queuesize_textbox.Text;
sim_data.Mplat = branch_latency_textbox.Text;
sim_data.RUU = ruu_size_textbox.Text;
```

- mai departe vom lua fiecare detaliu din fiecare field din View si le vom pune in SImData

```
Consumer.send_command_to_server(sim_data.getCommand());
serverOutput.Buffer.Text = Consumer.get_output_from_server();
Consumer.close_connection();
```

- mai departe vom crea comanda din sim_data si o vom trimite catre server.
- vom prelua dupa output-ul din server si il vom pune in locul continutului curent al textbox-ului
- in final vom inchide conexiunea cu serverul

Eventul pentru modificarea datelor din UI

```
protected void CacheType_onChange(object sender, EventArgs e)
{
    string cache_type = cache_type_combobox.ActiveText;

    if (cache_type.Equals("dl1")) {
        sets_textbox.Text = "128";
        blocksize_textbox.Text = "32";
        asoc_textbox.Text = "4";
    }
    else if (cache_type.Equals("ul2")){
        sets_textbox.Text = "1024";
        blocksize_textbox.Text = "64";
        asoc_textbox.Text = "4";
    }
    else if (cache_type.Equals("il1")){
        sets_textbox.Text = "512";
        blocksize_textbox.Text = "32";
        asoc_textbox.Text = "1";
    }
```

- Aceasta bucata de cod este rulata de fiecare data cand tipul cache-ului este schimbat pentru a updata valorile default de la fiecare tip

Server Side

```
public static void Main(string[] args)
{
    IPAddress IP = IPAddress.Parse("0.0.0.0");
    int PORT = 8000;

    Thread serverThread = new Thread(() => Run(IP, PORT));
    serverThread.Start();
    Console.WriteLine($"Server started. Listening for requests at {IP}:{PORT}");
}
```

- Serverul se lanseaza in execuție intr-un thread separat prin intermediul metodei Run, care se apeleaza cu 2 parametri: adresa IP si port.

Metoda Run()

```
public static void Run(IPAddress IP, int PORT)
{
    TcpListener listener = null;

    try
    {
        listener = new TcpListener(IP, PORT);
        listener.Start();

        while (true)
        {
            TcpClient client = listener.AcceptTcpClient();
            Console.WriteLine("\n================== [ NEW REQUEST ] ==================");
            Console.WriteLine("\n[+] Request confirmed. ");

            Thread handleClient = new Thread(new ParameterizedThreadStart(handleClientRequest));
            handleClient.Start(client);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Eroare: " + e.Message);
    }
    finally
    {
        listener.Stop();
    }
}
```

- Primeste parametru IP-ul si portul pentru care sa se initializeze listenerul.
- Odata ce o conexiune este realizata catre IP:port, se printeaza un mesaj care indica acest lucru si se apeleaza metoda **handleClientRequest** care se executa intr-un nou thread.

Metoda **handleClientRequest**

- Componenta cea mai esentiala a serverului.

- Este componenta responsabila pentru primire si parsarea parametrilor de la client, dar si de lansarea in executie a simularii, de scrierea rezultatelor si de trimiterea acestora inapoi catre client.

- Algoritmul implementat de acest segment de cod este urmatorul
    - Primeste parametrii necesari executiei sim-outorder de la client
    - Parseaza parametrii si identifica benchmark-ul pentru care se doreste simularea.
    - In functie de benchmark, se atribuie input-ul aferent (.in sau .lsp, dupa caz).
    - Se executa simularea si datele sunt salvate intr-un fisier cu numele **output_DATA_ora:minute:secunde**
    - Se asteapta finalizarea simularii si se deschide **ultimul** fisier (deci cel generat in urma ultimei executii).
    - Se trimite continutul acestuia inapoi catre client.

- Aceasta functionalitate este ilustrata/vizualizata mai usor cu ajutorul mesajelor de DEBUG

```
Server started. Listening for requests at 127.0.0.1:8000

================== [ NEW REQUEST ] ==================

[+] Request confirmed.

[+] Received args: -redir:sim {OutputFile}.res -max:inst 5000000 -cache:dl1 dl1:128:32:4:l -fetch:ifqsize 4 -fetch:mplat 3 -issue:
inorder true -issue:width 4 -ruu:size 16 {benchmark_path}/wave5.ss

[+] Will execute: sim-outorder -redir:sim outputs/output_23042024_02:10:24.res -max:inst 5000000 -cache:dl1 dl1:128:32:4:l -fetch:
ifqsize 4 -fetch:mplat 3 -issue:inorder true -issue:width 4 -ruu:size 16 benchs/ss/wave5.ss < benchs/in/wave5.in

[+] Simulation completed successfully.
[+] Results written to outputs/output_23042024_02:10:24.res.

[+] Simulation results were sent back to the requester.

================== [ END REQUEST ] ==================


Waiting for new requests ...
```

- Dupa ce simularea este finalizata, serverul tranziteaza inapoi spre starea de listening pentru preluarea altor request uri.