UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

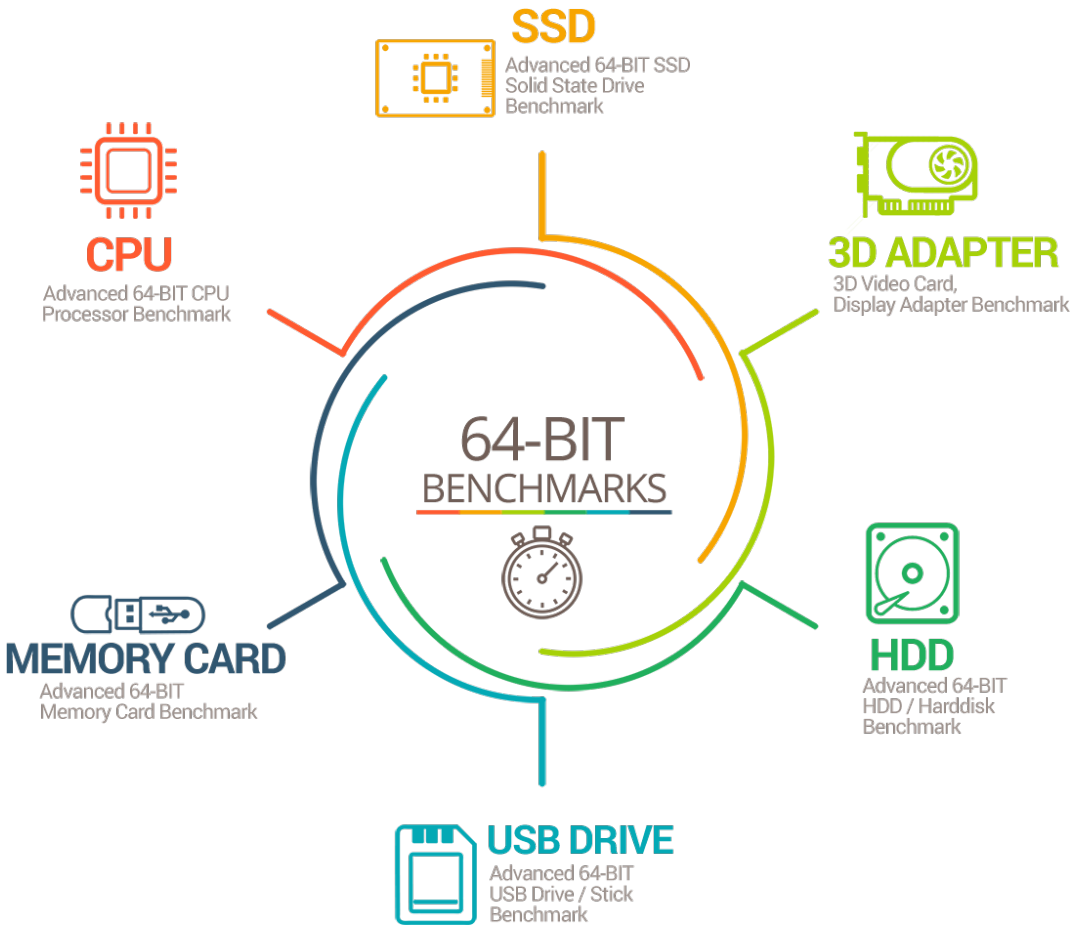
Program de testare a parametrilor de performanță ai unui PC

DOCUMENTAȚIE

Student: Tecuci Dragoș

Grupa: 30232

Cuprins



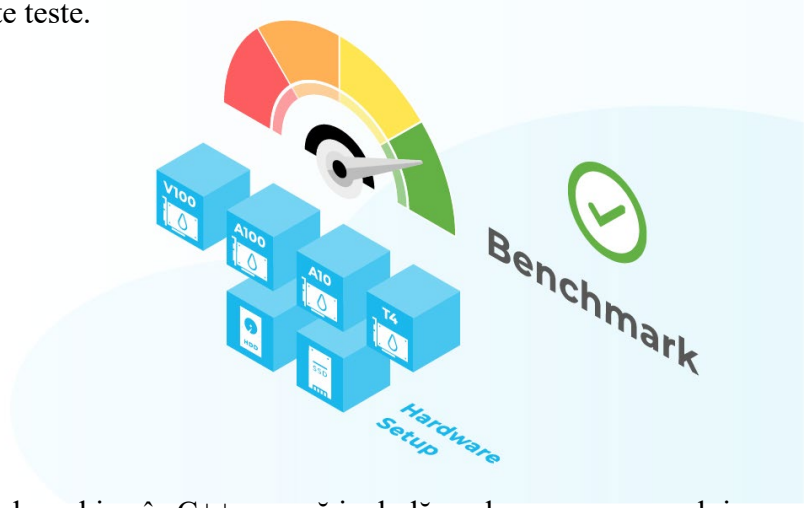
1. Introducere

În era tehnologiei moderne, calculatoarele personale reprezintă o parte indispensabilă a vieții noastre de zi cu zi. Fie că utilizăm computere pentru sarcini simple, cum ar fi navigarea pe internet, sau pentru aplicații complexe, cum ar fi jocuri video sau dezvoltarea software, performanța PC-ului nostru este un factor crucial. Pentru a asigura funcționarea optimă a unui computer, este necesar să înțelegem și să măsurăm parametrii săi de performanță. Acest lucru este posibil prin utilizarea unui instrument de testare numit "benchmark".

Titlul proiectului este "Program de testare a parametrilor de performanță ai unui PC (benchmark)". Scopul acestui proiect este de a dezvolta un software specializat care să permită utilizatorilor să evalueze și să compare performanța diferitelor componente ale unui PC, cum ar fi procesorul, placa video, memoria RAM și unitatea de stocare. Această evaluare se va baza pe măsurători obiective și indicatori relevanți care vor oferi o imagine clară a capacității PC-ului de a gestiona sarcinile și aplicațiile dorite.

Programele de benchmarking sunt instrumente esențiale pentru pasionații de tehnologie, dezvoltatori, dar și pentru utilizatorii obișnuiți, care doresc să-și optimizeze sistemul sau să-și evalueze investiția în hardware. Un program de benchmarking eficient va oferi date precise și utile privind performanța PC-ului într-un mod ușor de înțeles.

Acest proiect nu numai că va dezvolta un program de benchmarking eficient, dar va explora și diferitele aspecte ale testării de performanță, inclusiv conceptele de bază, cum ar fi viteza de procesare, capacitatea de calcul, performanța grafică, latenta memoriei și altele. De asemenea, va oferi o perspectivă asupra utilizării și interpretării datelor rezultate din aceste teste.



2. Studiu bibliografic

Pentru dezvoltarea unui program de benchmarking în C++ care să includă evaluarea procesorului, frecvenței, dimensiunii memoriei, vitezei de transfer a datelor, execuției operațiilor aritmetice și logice, precum și benchmarking pentru GPU și rețea, vor fi necesare diverse biblioteci și metode. Iată o listă detaliată a bibliotecilor potențial relevante pentru fiecare subpunct:

1. **Evaluarea tipului și a frecvenței procesorului**:
   - Pentru identificarea tipului de procesor, voi utiliza biblioteca "cpuid" din C++, care permite citirea informațiilor despre procesor.
   - Pentru măsurarea frecvenței procesorului, voi folosi biblioteca "chrono" din C++ pentru a măsura timpul.
2. **Evaluarea dimensiunii memoriei**:
   - Pentru a determina dimensiunea memoriei RAM, voi utiliza funcții din biblioteca "Windows.h" (pe platforma Windows) sau funcții din "unistd.h" (pe platforme Unix/Linux).
3. **Măsurarea vitezei de transfer a unui bloc de date**:
   - Pentru măsurarea vitezei de transfer a datelor, voi utiliza biblioteca "chrono" din C++ pentru a măsura timpul necesar pentru citirea și scrierea datelor.

4. **Evaluarea vitezei de execuție a operațiilor aritmetice și logice**:
    - Nu este nevoie de biblioteci externe pentru a efectua operații aritmetice și logice în C++. Voi folosi operatorii și funcțiile C++ standard pentru aceste operații.
5. **Benchmarking pentru GPU**:
    - Pentru benchmarking-ul GPU-ului, voi utiliza biblioteci specifice ca să acceseze și să măsoare performanța GPU-ului. Mai exact, OpenGL pentru compatibilitate.
6. **Benchmarking pentru rețea**:
    - Pentru benchmarking-ul rețelei, voi utiliza biblioteci precum "libcurl" sau "boost.asio" pentru a efectua operațiuni de rețea, cum ar fi transferul de fișiere sau măsurarea latenței.

3. Analiză și design

Pentru a descrie analiza și designul unei aplicații de benchmarking în C++ vom împărți această secțiune în câteva subsecțiuni pentru o mai bună înțelegere.

Funcționalități și mecanisme posibile/algoritmi:

1. **Evaluarea procesorului și frecvenței**:
    - **Funcționalitate**: Detectarea tipului și a caracteristicilor procesorului și măsurarea frecvenței procesorului.
    - **Mecanisme posibile**:
        - Măsurarea frecvenței procesorului se poate face măsurând timpul necesar pentru executarea unor operații CPU intensive, cum ar fi calculul unui număr fibonacci.

          Numerele mari calculate cum ar fi fib(10 000) depășesc limita pentru double, de aceea vor fi calculate și salvate cu ajutorul string-uri.

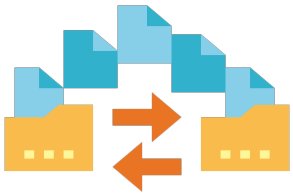
```
void fib(int a) {
    int a;
    std::cout << "Find how many numbers? \n";
    std::cin >> a;

    std::string x = "0";
    std::string y = "1";
    std::string z;

    for (int b = 0, c = 0; b < a; ++b) {
        if (b % 3 == 0) {
            ++c;
            std::cout << "\n" << c << ") ";
        }

        std::cout << x << "  ";
        z = y;
        y = x;
        x = addTwoStrings(x, z);

        if (x.length() < z.length()) {
            std::cout << "\nMaxed out at " << b + 1;
            break;
        }
    }
    std::cout << x << "  ";

}
```

2. **Evaluarea dimensiunii memoriei**:
   - **Funcționalitate**: Determinarea dimensiunii memoriei RAM disponibile.
   - **Mecanisme posibile**:
     - Se poate utiliza funcția "GetPhysicallyInstalledSystemMemory" (pe Windows) sau "sysinfo" (pe Linux) pentru a obține informații despre memorie.

```cpp
#include <windows.h>

int main() {
    MEMORYSTATUSEX memoryStatus;
    memoryStatus.dwLength = sizeof(MEMORYSTATUSEX);

    if (GlobalMemoryStatusEx(&memoryStatus)) {
        // Dimensiunea totală a memoriei RAM
        long long totalMemory = memoryStatus.ullTotalPhys;
        // Afișați dimensiunea totală a memoriei
        std::cout << "Dimensiunea totală a memoriei RAM: " << totalMemory << " bytes\n";
    } else {
        std::cerr << "Eroare la accesarea informațiilor despre memorie\n";
    }

    return 0;
}
```

3. **Măsurarea vitezei de transfer a datelor**:
   - **Funcționalitate**: Testarea vitezei de citire și scriere a datelor.
   - **Mecanisme posibile**:

     - Crearea datelor de test : Generarea datelor care urmează să fie transferate. Aceste date sunt generate aleator. Fișier cu dimensiunea - 1MB
     - Pentru începerea testului de citire: Încep măsurarea timpului și încep procesul de citire a datelor din memoria de stocare în memoria RAM a sistemului. Acest lucru se poate face folosind funcții precum `fread()` sau `ifstream` în C++ pentru citire din fișiere, sau utilizând funcții de copiere, cum ar fi `memcpy()` pentru copierea directă în memorie.
     - Calcularea vitezei de citire/scriere: Pentru a calcula viteza de citire, se împarte dimensiunea datelor la timpul înregistrat pentru citire. De exemplu, dacă s-a citit 1 MB de date în 0,1 secunde, viteza de citire ar fi de 10 MB/sec.



4. **Benchmarking pentru rețea**:
   - **Funcționalitate**: Efectuarea de teste de transfer de date prin rețea și măsurarea latenței rețelei.
   - **Mecanisme posibile**:
     - Utilizarea bibliotecii `#include <winsock2.h>` pentru măsurarea timpului pentru pachetele de date primite.

```cpp
double performNetworkTest(const char* host, const char* path)
{
    // Create a socket
    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == INVALID_SOCKET)
    {
        std::cerr << "Failed to create socket." << std::endl;
        return -1;
    }
```

```
14.
15.     // Resolve server address
16.     addrinfo hints{};
17.     hints.ai_family = AF_INET;
18.     hints.ai_socktype = SOCK_STREAM;
19.     hints.ai_protocol = IPPROTO_TCP;
20.
21.     addrinfo* result = nullptr;
22.     if (getaddrinfo(host, "http", &hints, &result) != 0)
23.     {
24.         std::cerr << "Failed to resolve server address." << std::endl;
25.         closesocket(clientSocket);
26.         return -1;
27.     }
28.
29.     // Connect to the server
30.     if (connect(clientSocket, result->ai_addr, static_cast<int>(result->ai_addrlen)) ==
    SOCKET_ERROR)
31.     {
32.         std::cerr << "Failed to connect to the server." << std::endl;
33.         freeaddrinfo(result);
34.         closesocket(clientSocket);
35.         return -1;
36.     }
37.
38.     // Free the result structure
39.     freeaddrinfo(result);
40.
41.     // HTTP GET request
42.     std::string httpRequest = "GET " + std::string(path) + " HTTP/1.1\r\n"
43.         "Host: " + std::string(host) + "\r\n"
44.         "Cache-Control: no-cache\r\n"
45.         "Connection: close\r\n\r\n";
46.
47.
48.     // Send the HTTP request
49.     if (send(clientSocket, httpRequest.c_str(), static_cast<int>(httpRequest.length()), 0)
    == SOCKET_ERROR)
50.     {
51.         std::cerr << "Failed to send HTTP request." << std::endl;
52.         closesocket(clientSocket);
53.         return -1;
54.     }
55.
56.     // Receive the server's response
57.     char buffer[4096];
58.     std::ostringstream response;
59.
60.     while (true)
61.     {
62.         int bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);
63.         if (bytesRead > 0)
64.         {
65.             response.write(buffer, bytesRead);
66.         }
67.         else if (bytesRead == 0)
68.         {
69.             // Connection closed by the server
70.             break;
71.         }
72.         else
73.         {
74.             std::cerr << "Error while receiving data." << std::endl;
75.             closesocket(clientSocket);
76.             return -1;
77.         }
78.     }
79.
80.     closesocket(clientSocket);
81.     return transferTime;
```

Exemplu de utilizare:

```
WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        std::cerr << "Failed to initialize WinSock." << std::endl;
        return 0;

    }

performNetworkTest("www.google.com", "/search?q=tesla");

WSACleanup();
```

Interacțiunea Utilizatorului cu Aplicația:

Utilizatorul va interacționa cu aplicația printr-o interfață de utilizator (UI) simplificată, care va oferi opțiuni clare pentru configurarea și rularea testelor de benchmarking. Interfața de utilizator include următoarele elemente:

1. **Selectarea Funcționalității de Benchmarking:**

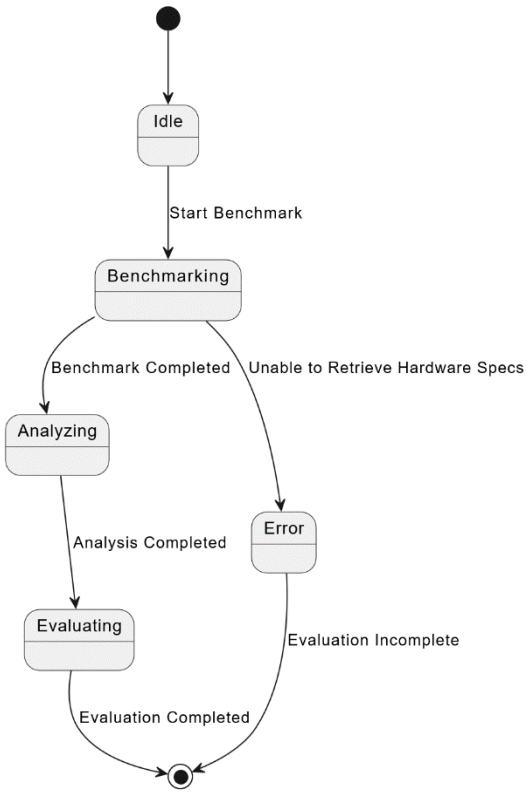
- Utilizatorul va putea alege funcționalitatea de benchmarking pe care dorește să o efectueze, precum evaluarea procesorului, a memoriei, a vitezei de scriere/citire date.

2. **Inițierea Benchmarking-ului:**

- După configurarea parametrilor, utilizatorul va putea iniția testul de benchmarking. Aplicația va efectua testul în conformitate cu parametrii specificați.

3. **Monitorizarea Progresului Testului:**

- În timpul benchmarking-ului, utilizatorul are acces la o bară de progress/ informații în timp real despre stadiul testului. Acest lucru poate oferi o perspectivă asupra duratei rămase.



Un mod in care m-am gândit să implementez monitorizarea progresului în proiect:

Inițierea Progresului:

- Atunci când încep testul, setez valoarea inițială a barei de progres la 0%.

Actualizarea Progresului:

- În timp ce testul rulează, are loc actualizarea la valoarea barei de progres pe măsură ce progresul este înregistrat.

Afisarea Progresului:

- Afișarea valorii curente a barei de progres utilizatorului, astfel încât acesta să poată vedea cum avansează testul.

Oprirea Progresului:

- Când testul se încheie, setez bara de progres la 100% pentru a indica că testul a fost finalizat.

Feedback vizual sau textual:

- Pe lângă bara de progres, ofer feedback textual pentru a descrie stadiul testului sau detalii suplimentare utilizatorului. De exemplu, se va afișa un text care spune "Testul în curs de desfășurare..." sau "Test finalizat cu succes! și scorul calculat".

4. **Posibilitatea de Salvare sau Export a Rezultatelor:**

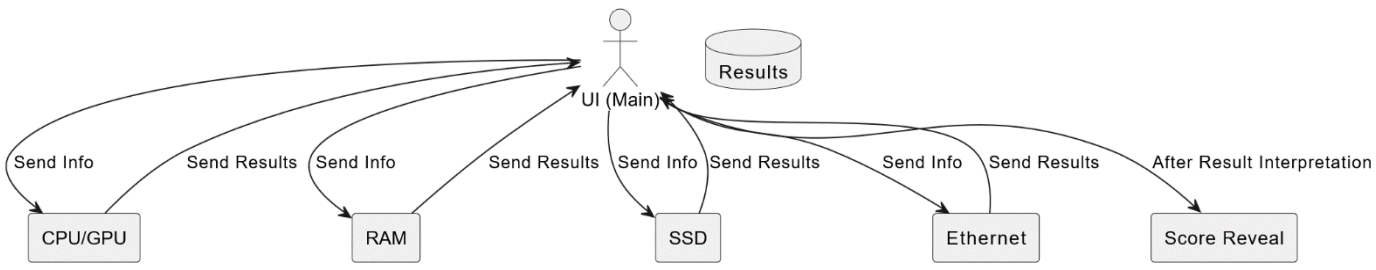
- Utilizatorul poate dori să salveze rezultatele testelor într-un fișier pentru referință ulterioară. Această opțiune este oferită pentru a asigura persistența datelor.

5. **Gestionarea Erorilor și Afișarea de Mesaje de Informare:**

- Aplicația ar trebui să gestioneze erorile și să afișeze mesaje clare de informare utilizatorului în cazul în care apar probleme sau dacă utilizatorul face selecții incorecte.

Interacțiunea între Componente:

- Utilizatorul selectează tipul de test și configurează parametrii în interfața de utilizator.
- Interfața de utilizator inițiază modulele de benchmarking pentru testul selectat.
- Modulele de benchmarking rulează testul și colectează datele de performanță.
- Datele colectate sunt trimise către motorul de măsurare a performanței.
- Motorul de măsurare procesează datele și le afișează în interfața de utilizator.
- Utilizatorul primește rezultatele testului în interfața de utilizator.



4. Implementare

Fișiere și librării

- **ProgressForm.h**: Fișierul antet pentru clasa ProgressForm.
- **iostream, chrono, string, etc.**: Biblioteci standard C++ utilizate în aplicație.
- **winsock2.h, ws2tcpip.h**: API-ul Windows Socket pentru operațiile de rețea.

Funcții

- **getCPUInfo()**: Obține informații despre CPU, inclusiv tipul și numărul de nuclee.
- **ProgressForm::info()**: Furnizează informații despre RAM.
- **performNetworkTest(host, path)**: Măsoară timpul de transfer al rețelei către un host și un traseu specificat.
- **ProgressForm::networkMeasurement()**: Realizează teste de rețea pe URL-uri predefinite.
- **generateRandomString(length)**: Generează un șir aleator de o anumită lungime.
- **generateRandomText()**: Generează un text cu conținut aleator.
- **writeToFile(filename)**: Scrie conținutul generat într-un fișier și măsoară timpul necesar.
- **readFromFile(filename)**: Citește conținutul dintr-un fișier, șterge fișierul și măsoară timpul necesar.
- **ProgressForm::dataMeasurement()**: Măsoară timpul de scriere și citire a fișierelor.
- **addTwoStrings(str1, str2)**: Adaugă două șiruri care reprezintă numere.
- **fib(a)**: Generează secvența Fibonacci până la o limită dată.

- **calculateFibInThread(n)**: Calculează secvența Fibonacci într-un fir de execuție separat.
- **ProgressForm::cpuMeasurement()**: Măsoară timpul total al procesorului, executând calcule Fibonacci în mai multe fire de execuție.

Detalii de Implementare ale funcțiilor principale

Informații despre CPU

- Funcția `getCPUInfo()` utilizează instrucțiunea CPUID pentru a obține informații despre CPU, inclusiv șirul de marcă și numărul de nuclee.

Măsurare vitezei Rețelei

- Funcția `performNetworkTest()` stabilește o conexiune la un server și măsoară timpul necesar pentru transferul datelor. Repetă testul de 4 ori.

Măsurarea vitezei de citire/scriere a datelor

- Funcția `dataMeasurement()` folosește `writeToFile()` și `readFromFile()` pentru a măsura timpul necesar pentru a scrie și citi date dintr-un fișier, respectiv.

- Funcția `calculateFibInThread()` calculează secvența Fibonacci într-un fir de execuție separat, iar funcția `cpuMeasurement()` măsoară timpul total al procesorului.

```
std::vector<std::thread> threads;
 unsigned int numThreads = std::thread::hardware_concurrency();

 for (int i = 0; i < numThreads; ++i) {

     threads.emplace_back(calculateFibInThread, i);

  }
```

Explicație de ce am ales hardware concurrency și cum am gestionat thread-urile:

Diferența în numărul de thread-uri între `std::thread::hardware_concurrency()` și crearea manuală a thread-urilor cu biblioteca `pthreads` poate avea mai multe explicații:

Implementarea Platformă-Specifică:

- `std::thread::hardware_concurrency()` furnizează numărul de nuclee de procesare (cores) disponibile raportate de către sistem. Aceasta nu reprezintă un număr maxim de thread-uri pe care le-am putut crea. Sistemul de operare poate utiliza tehnici precum planificarea la nivel de nucleu și hiper-threading pentru a gestiona mai multe thread-uri pe fiecare nucleu fizic.
- Biblioteca `pthreads` poate permite crearea a un număr mare de thread-uri indiferent de numărul de nuclee fizice, dar performanța reală va depinde de capacitatea sistemului de a gestiona aceste thread-uri și de resursele hardware disponibile.

Limitări ale Sistemului de Operare:

Sistemele de operare pot impune limite asupra numărului maxim de thread-uri care pot fi create de o aplicație. Aceste limite pot varia în funcție de configurarea sistemului și de politica de administrare a resurselor. De aceea am considerat să creez atâtea thread-uri cât raportează sistemul că le poate gestiona eficient.

5. Concluzii

Când am creat programul de testare a parametrilor de performanță a unui PC, am experimentat o serie de beneficii care mi-au îmbunătățit în mod semnificativ cunoștințele și abilitățile în domeniul informaticii. Iată câteva concluzii personale despre această experiență:

Am Experimentat cu Programarea la Nivel de Sistem:

- Crearea programului m-a determinat să înțeleg în mod profund arhitectura hardware-ului PC-ului, explorând și interacționând direct cu parametri precum CPU, RAM și rețeaua.

Am Învățat Cum să Optimizez Performanța:

- Identificarea bottleneck-urilor și optimizarea performanței sistemului au reprezentat provocări pe care le-am abordat, dezvoltându-mi competențe valoroase în îmbunătățirea eficienței codului și a sistemului.

M-am Inițiat în Măsurarea Performanței:

- Procesul de măsurare a performanței sistemului și interpretarea rezultatelor m-a familiarizat cu abilități esențiale pentru dezvoltarea de software eficient și eficace.

Am Dezvoltat Competențe în Comunicarea Tehnică:

- Scrierea documentației și comunicarea rezultatelor au consolidat competențele mele de comunicare tehnică, pregătindu-mă pentru colaborarea în medii profesionale.

Am Adăugat un Proiect Valoros în Portofoliul Meu:

- Programul de testare a devenit o componentă semnificativă în portofoliul meu :)

6. Bibliografie

- https://www.linkedin.com/advice/0/how-do-you-compare-benchmark-your-hardware-performance
- https://www.lucidchart.com/blog/8-steps-of-the-benchmarking-process
- https://www.userbenchmark.com/page/guide
- https://www.researchgate.net/publication/273133047_How_to_Build_a_Benchmark
- https://www.simplilearn.com/tutorials/cpp-tutorial/cpp-gui
- https://stackoverflow.com/questions/14679105/c-whats-the-proper-way-to-perform-a-speed-test
- https://inside.caratlane.com/measure-app-performance-by-using-macrobenchmark-fe5e174a9a67
- https://stackoverflow.com/questions/850774/how-to-determine-the-hardware-cpu-and-ram-on-a-machine
- https://www.quora.com/With-overflow-in-C-and-C-how-do-I-output-Fibonacci-numbers-without-an-eventual-overflow-I-could-do-it-with-two-arrays-digits-in-each-element-Is-there-a-better-way
- https://stackoverflow.com/questions/65389235/how-to-find-fibonacci-sums-of-huge-numbers
- `https://inversepalindrome.com/blog/how-to-create-a-random-string-in-cpp`