FACULTATEA: Automatică si Calculatoare
SPECIALIZAREA: Calculatoare si Tehnologia Informației
DISCIPLINA: Structura sistemelor de calcul
PROIECT: Măsurarea timpului de execuție

Îndrumător:
Neagu Mădălin

Realizator:
Mihalache Rareș

1. Introducere:

Scopul acestui proiect este de a observa si de a compara timpul de executie a mai multor procese scrise in diferite limbaje de programare, printre care: C, C++, Java si C#. Se urmareste realizarea mai multor programe, pentru a masura diferite cazuri de utilizare: alocarea memoriei (mod static si dinamic), accesul la memorie, crearea unui thread, thread context switch si thread migration.

2. Obiective:

- Identificarea si utilizarea functiilor destinate pentru a masura timpul de executie/ numarul de cicluri parcurse a unei secvente de cod dintr-un program.

- Implementarea mai multor programe (in diferite limbaje de programare) pentru cateva din operatiile uzuale folosite, cu scopul de a face o comparatie intre timpii de executie necesari fiecarui limbaj de programare folosit. Operatiile pe care le vom testa sunt:
    - Alocarea dinamica a memoriei
    - Alocarea statica a memoriei
    - Accesul la memorie
    - Crearea unui thread
    - Thread context switch
    - Thread migration

Deci, vom avea 6 programe, scrise in mai multe limbaje de programare.

3. Studiu bibliografic

Timpul de rulare al unei secvente de cod dintr-un program se poate calcula in mai multe metode, in functie de limbajul folosit. Acest timp poate varia in functie de mai multi factori, cum ar fi: compilatorul folosit (daca exista), folosirea diferitelor “build configurations” (Release sau Debug mode), felul in care programatorul a implementat codul si functiile/ API-urile pe care acesta le-a folosit in masurarea timpului. (unele pot avea o precizie/rezolutie mai mare decat altele).

Cel mai important criteriu pentru determinarea timpului de executie este felul in care programatorul implementeaza codul.

Imediat dupa, probabil cel mai important rol il are compilatorul. Acesta poate sa optimizeze codul prin diferite transformari, astfel incat din programul scris sa reiasa un program cu un output echivalent pe toate cauzrile (acelasi rezultat), dar care sa foloseasca mai putine resurse sau care sa se execute mai rapid.

Proiectul a fost facut folosind: C, C++, Java si C#. Atat C, cat si C++ sunt limbaje compilate in intregime. Asta inseamna ca sunt convertite in cod masina, cod pe care procesorul il poate executa. Java si C# sunt semi-compilate, adica sunt compilate si optimizate pana intr-un stadiu intermediar, de unde se vor interpreta. La compilare, Java genereaza un "bytecode" care este interpretat la runtime de un JVM (Java Virtual Machine). Aceasta masina virtuala transforma "bytecode-ul" in cod masina, care mai apoi se poate executa de catre procesor. Similar, C# este compilat in CIL (Common Intermediate Language), un limbaj intermediar care va fi rulat de CLR (Common Language Runtime), parte a .NET framework.

Teoretic, limbajele compilate sunt mai rapide ca timp de executie fata de cele interpretate, deci C si C++ ar trebui sa aiba timpi mai buni.

Ca si build configuration vom folosi varianta "Release" pentru toate limbajele, pentru ca este mai rapid.

Fiecare limbaj vine cu un set de functii/ API-uri care pot fi folosite pentru a masura timpul de executie.

4. Design si implementare

Pentru C, am folosit functia **clock()**, care tine cont de numarul de cicluri din momentul in care programul a fost rulat. Pentru a obtine numarul de secunde folosite de CPU, este nevoie sa impart numarul de cicluri la **CLOCKS_PER_SEC (= 1000).**

Pentru C++, am folosit biblioteca **<chrono>** pentru a masura timpul de executie. Aceasta biblioteca contine o clasa **steady_clock**, cu ajutorul careia, prin metoda **now()**, putem masura timpul de executie de la rularea programului si pana la punctul in care am ajuns cu executia. Deci, putem masura mai multe intervale de timp, pentru a gasi timpul unei anumite secvente de cod.

In Java, folosim metoda **nanoTime()** din clasa **System**, pentru a masura la nivel de nanosecunda timpul de executie. Apoi, convertim valoarea la secunda inmultind cu 10e-9.

Nu in ultimul rand, in C# vom folosi 2 obiecte: un obiect de tip **Stopwatch**, care cronometreaza timpul parcurs dintr-un anumit punct in altul si un obiect **TimeSpan**, care are rolul de a lua valoarea masurata si de a o converti in secunde.

5. Testare

Mai jos se gasesc timpii de executie pentru programele testate.

Acces la memorie:

C (sec.)	**C++** (sec.)	**Java** (sec.)	**C#** (sec.)
0.018	0.021	0.021	0.017
0.018	0.018	0.034	0.019
0.017	0.018	0.022	0.017
0.019	0.018	0.038	0.017
0.020	0.017	0.037	0.017
0.018	0.042	0.030	0.018
0.017	0.026	0.030	0.019
0.01814	**0.02285**	**0.03028**	**0.01771**

Alocare de memorie (mod static):

C (sec.)	**C++** (sec.)	**Java** (sec.)	**C#** (sec.)
0.000	0.000	0.014	0.000
0.000	0.000	0.016	0.000
0.000	0.000	0.015	0.000
0.000	0.000	0.015	0.000
0.000	0.000	0.014	0.000
0.000	0.000	0.015	0.000
0.000	0.000	0.015	0.000
0.00000	**0.00000**	**0.01485**	**0.00000**

Alocare de memorie (mod dinamic):

C (sec.)	**C++** (sec.)	**Java** (sec.)	**C#** (sec.)
0.004	1.910	0.013	0.005
0.005	1.670	0.014	0.005
0.004	1.932	0.013	0.004
0.006	1.529	0.015	0.004
0.004	1.507	0.014	0.005
0.006	1.534	0.013	0.005
0.005	1.543	0.014	0.006
0.00485	**1.66071**	**0.01371**	**0.00485**

Crearea a 10000 threaduri:

C (sec.)	**C++** (sec.)	**Java** (sec.)	**C#** (sec.)
0.367	0.357	0.021	0.033
0.361	0.373	0.021	0.032
0.357	0.355	0.023	0.031
0.360	0.403	0.021	0.028
0.357	0.371	0.023	0.029
0.364	0.422	0.023	0.028
0.360	0.357	0.023	0.029
0.36085	**0.37685**	**0.02214**	**0.03000**

Thread context switch:

C (sec.)	**C++** (sec.)	**Java** (sec.)	**C#** (sec.)
0.001	0.025	0.924	0.000
0.000	0.000	0.918	0.000
0.001	0.037	0.935	0.000
0.000	0.000	0.920	0.000
0.000	0.000	0.918	0.000
0.000	0.000	0.920	0.000
0.000	0.021	0.920	0.000
0.00028	**0.01185**	**0.92214**	**0.00000**

Thread migration:

C (sec.)	**C++** (sec.)	**Java** (sec.)	**C#** (sec.)
0.0002	0.0002	---	0.0400
0.0001	0.0001	---	0.0390
0.0003	0.0003	---	0.0460
0.0002	0.0002	---	0.0400
0.0001	0.0001	---	0.0400
0.0001	0.0001	---	0.0380
0.0002	0.0002	---	0.0400
0.000171	**0.000171**	**---**	**0.040428**

6. Concluzii

Prin acest proiect am invatat diferite metode de a masura timpul de executie in diferite limbaje de programare, folosindu-ma de diferite biblioteci si API-uri si am exersat programarea concurenta, utilizand mai multe threaduri.

In unele cazuri, exista diferente destul de mari la comparatii (pentru acelasi caz de utilizare), aceste diferente fiind cauzate de cele mai multe ori de aspect legate de limbajul de programare: de cum lucreaza acesta cu memoria, de implementarile anumitor API-uri sau a anumitor biblioteci care nu se regasesc in celelalte limbaje, de compilator, etc.

In general, am incercat sa implementez o forma standard a codului (sa fie aproximativ la fel) pentru toate limbajele folosite, pentru a se evidentia cat mai bine diferentele de timpi, insa uneori nu s-a reusit. (exemplu: Static memory allocation -> in C# am folosit un array de 10^5 elemente, in timp ce in celelalte limbaje de programare am folosit un array de 10^8 elemente. Motiv: nu puteam declara mai multe elemente in C#, pentru ca riscam sa primesc stack overflow, alocarea statica in C#, realizandu-se doar pe stiva).

7. Bibliografie

- https://en.cppreference.com/w/cpp/chrono/duration/duration_cast
- https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch.elapsed?view=net-5.0
- https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-9-82
- https://www.thoughtco.com/about-compilers-and-interpreters-958276
- https://stackify.com/heres-how-to-calculate-elapsed-time-in-java/
- https://www.baeldung.com/java-stack-heap
- https://stackoverflow.com/questions/19361888/static-and-dynamic-memory-in-java/19361954
- http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
- https://www.cplusplus.com/reference/condition_variable/condition_variable/
- https://eli.thegreenplace.net/2016/c11-threads-affinity-and-hyperthreading/
- https://www.linkedin.com/pulse/c-threading-tasks-async-code-synchronization-part-1-meikopoulos/