Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Отчет

По Лабораторной работе N1

Автор:

Ли Евгений Владимирович

Санкт-Петербург
2022

Задание 1

DLL файл – это почти файл, содержащий машинный код а также таблицу с ее функциями и ресурсами. Т.е. это почти exe файл, только без точки входа. По запросу программы он загружается в память.

Главная проблема DLL – это то, что они абсолютно не могут в ООП. По факту DLL может только экспортировать имена функций и переменных. Если попытаться экспортировать классы и функции из C++, то компилятор начнет “манглить” (то есть искажать) имена классов и функций. И если при неявной загрузке (которая в свою очередь занимает место от запуска и до завершения программы) это не критично, так как компилятор сам справится, то при использовании явной загрузки приходится заново проводить все связи между классами и функциями, что естественно не удобно.

Более того, при внесении изменений в библиотеку, существующие программы перестанут работать.

Задание 2

Рассмотрим код после декомпиляции из F# в C#:

Рассмотрим метод следующий метод:

F#

```
  let pi = 3.14


  let area shape =
    match shape with
    | Circle r -> pi * r * r
    | Square s -> s * s
    | Rectangle(l, w) -> l * w
```

C#

```
    public static double area(Shape shape)
    {
      double radius;
      if (!(shape is Shape.Square))
      {
        if (!(shape is Shape.Rectangle))
        {
          Shape.Circle circle = (Shape.Circle)shape;
          radius = circle._radius;
          return 3.14 * radius * radius;
        }
        Shape.Rectangle rectangle = (Shape.Rectangle)shape;
```

```
            return rectangle._length * rectangle._width;
        }
        Shape.Square square = (Shape.Square)shape;
        radius = square._length;
        return radius * radius;
    }
```

Интересно, что все объявленные переменные были подставлены в функции, и их объявления в виде const после декомпиляции нету.
После декомпиляции метод превратился в последовательно if-ов, проверяющих является ли фигура одним из классов наследников.

Задание 3

Идея работы что у Nuget, что у Maven предельно проста: кто-то пишет крутую библиотеку -> публикует ее на какой-то сервер -> остальные разработчики, указывая ссылку на сервер скачивают ее и пользуются. Ссылки на библиотеки прописываются в файлах .csproj и pom.xml в блоках <ItemGroup> и <Dependencies> соответственно.

Задание 4

C# - Benchmarkdotnet

Method	Mean	Error	StdDev	Gen 0	Allocated
SortBubble	69,886.53 us	58.865 us	55.062 us	-	48 B
SortStd	69.03 us	0.211 us	0.187 us	-	-
MergeSort	310.83 us	2.273 us	2.015 us	377.9297	792,040 B

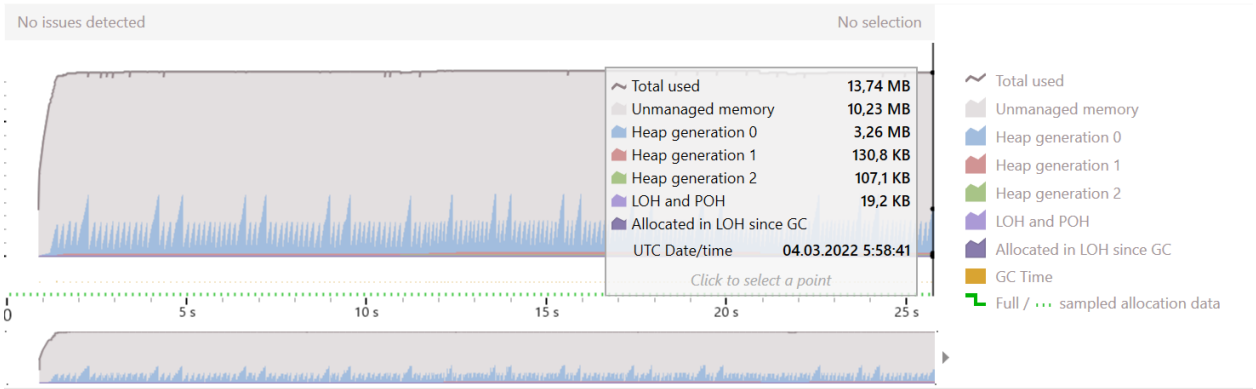
Java – JMH

Benchmark	Mode	Cnt	Score	Error	Units
HelloWorldBenchmark.MergesortBenchmark	thrpt	25	8649,004 ±	347,086	ops/ms
HelloWorldBenchmark.StdSortBenchmark	thrpt	25	45582,438 ±	1226,696	ops/ms

В общем то результаты по времени работы не удивляют – дольше всех работала сортировка пузырьком, быстрее всех – стандартная сортировка.

Задание 5

Работа бэкапов при файловом способе хранения.



Memory Snapshots ☑ Show unmanaged memory

Main thread

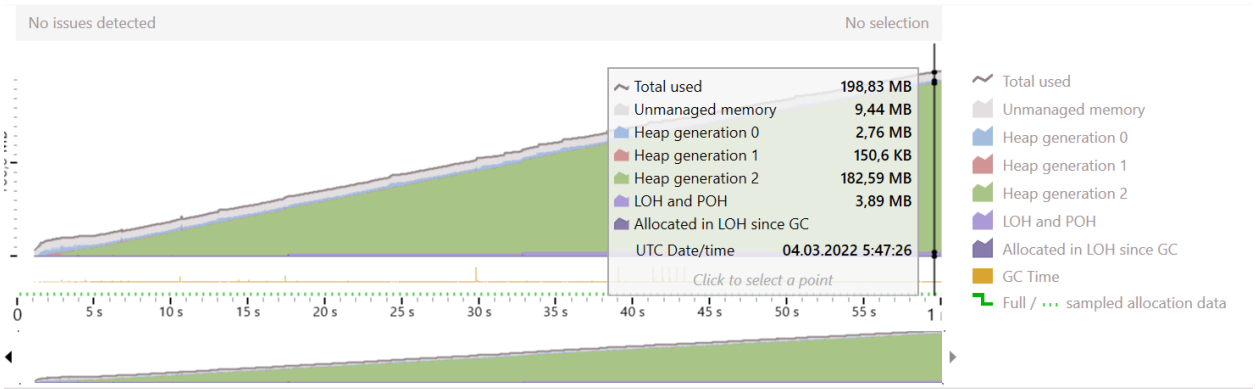
76,47%	File I/O
17,72%	System code
1,74%	Collections
1,36%	GC Wait
1,31%	String
0,77%	Special
0,65%	User code

Application

Options	Profile native, .NET Framework, .NET Core and Mono application Create console window Profile one process
Command line	C:\Users\Пользователь\Documents\GitHub\lievvl\BackupsExtra' \Debug\netcoreapp3.1\BackupsExtra.exe
Working directory	C:\Users\Пользователь\Documents\GitHub\lievvl\BackupsExtra' \Debug\netcoreapp3.1
Process Filters	Everything
Stats	14 Modules 246 Functions 6 Threads

Работа бэкапов при виртуальном способе хранения.

Profiling session has finished



Memory Snapshots ☑ Show unmanaged memory

Main thread

76,47%	File I/O
17,72%	System code
1,74%	Collections
1,36%	GC Wait
1,31%	String
0,77%	Special
0,65%	User code

Application

Options	Profile native, .NET Framework, .NET Core and Mono application Create console window Profile one process
Command line	C:\Users\Пользователь\Documents\GitHub\lievvl\BackupsExtra' \Debug\netcoreapp3.1\BackupsExtra.exe
Working directory	C:\Users\Пользователь\Documents\GitHub\lievvl\BackupsExtra' \Debug\netcoreapp3.1
Process Filters	Everything
Stats	14 Modules 246 Functions 6 Threads

Выводы:
В части быстродействия оптимизация бессмысленна, так как основное время взаимодействия приходится на файловое взаимодействие, что является зоной ответственности операционной системы.
Как видно по графику задействованной памяти при использовании виртуального репозитория, он написан, говоря мягко, не очень хорошо, скорее всего виртуальные файлы не удаляются/не вызывается dispose.