Homework

Task 1. `memoryleak` app (preparation)

We will use the same app as in the previous module, so here is the setup again:

- clone **`git clone https://github.com/sebastienros/memoryleak.git`** (great testing app by Sébastien Ros)
- run it (in **`.\src\MemoryLeak\MemoryLeak`**):

```
dotnet run -c Release
```

- **`https://localhost:5001/`** should present a nice introspective graph about memory usage (Working Set), allocations, CPU and current Request Per Second (RPS)
- MemoryLeak exposes some REST endpoints for testing various memory-related scenarios, we will use **`\bigstring`** which just allocates and returns 10KB string. You can test it at **`https://localhost:5001/api/bigstring`**
- you can hit it by F5 many times to observe some memory usage change
- we will make a simple load test agains **`bistring`** endpoint using **`https://github.com/aliostad/SuperBenchmarker`** command-line tool (just download single EXE file from the repository). Run the following command to confirm it is working correctly:

```
.\sb.exe -n 10 -c 1 -u http://localhost:5000/api/bigstring
```

As you see we use **`http`** endpoint to avoid unnecessary https handshake overhead.

Task 1. `memoryleak` app - Server GC

- by default it is running as Bacground Server GC, so let's observe its behaviour!
- start the app as usual:

```
dotnet run -c Release
```

- find the proper PID with the help of **`dotnet-trace ps`**:

```
❯ dotnet-trace ps
 ...
 32972 MemoryLeak ...\net5.0\MemoryLeak.exe
```

- ... or use **`-name MemoryLeak`** in the following commands
- start recording the event pipes session:

```
❯ dotnet-trace collect --profile gc-collect -o servergc_noloadtest.nettrace -n MemoryLeak
```

- hit F5 on **`https://localhost:5001/api/bigstring`** few (dozen) times

Task 1. `memoryleak` app - Server GC (cont.)

- don't be surprised if the recording trace will be **`0.00`** in size - if you have plenty of RAM the GC probably won't be triggered while making a few requests
- open such an "empty" trace in PerfView - you should see **no reports** prepared because the session is lacking of necessary events
- in *EventStats* you should see only small amount of not GC-related events
- in *Events* view you will see list of all the events. Double click **`Microsoft-Windows-DotNETRuntimeRundown/Runtime/Start`** event to see a single event recorded:
- in the **`Rest`** column you can find, for example, information about the runtime version (**`VMMajorVersion`** and **`VMMinorVersion`**) or the **`RuntimeDllPath`**
- and... that's it! That was your the very first contact with the PerfView tool (😍)

Task 1. `memoryleak` app - Server GC (cont.)

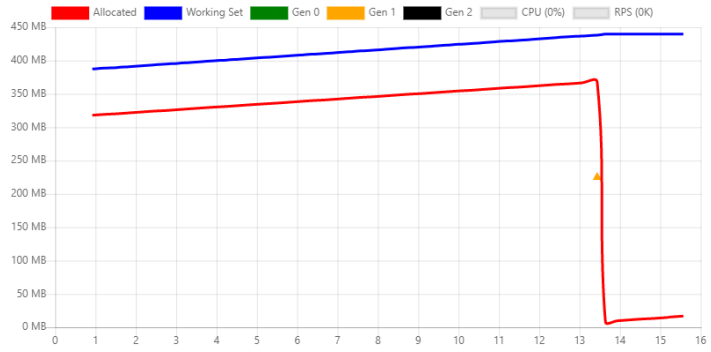
- let's make a load test like in previous module but now we will measure the app with the *GCStats*!
- start recording the event pipes session (notice file name change):

```
❯ dotnet-trace collect --profile gc-collect -o servergc_loadtest.nettrace -n MemoryLeak
```

- start the load test:

```
.\sb.exe -y 100 -n 10000000 -c 16 -u http://localhost:5000/api/bigstring
```

- wait until you will observe the very first GC (or a few GCs) in the app (as you may know it will be take longer or shorter depending on you RAM and cores) - **`https://localhost:5001/`** will show sth like:



- stop the dotnet-trace session!

Task 1. `memoryleak` app - Server GC (cont.)

- open the resulting **`servergc_loadtest.nettrace`** file in PerfView - you should see *GCStats* report under *Memory Group*. What's the *Total GC Pause*? What's the *Max GC Heap Size*?
- how does **`GC Rollup By Generation`** table look? Do you see **`Induced`** GCs?
- what were the reasons for the GCs happening? Look at **`GC Events by Time`** table and its **`Trigger Reason`** column.

Task 2. `memoryleak` app - Workstation GC

- let's make a load test again but now for the most "aggressive", single-threaded GC:

```
$Env:COMPlus_gcServer=0
$Env:COMPlus_gcConcurrent=0
dotnet run -c Release
```

- start recording the event pipes session (notice file name change):

```
❯ dotnet-trace collect --profile gc-collect -o wksgc_loadtest.nettrace -n MemoryLeak
```

- start the load test:

```
.\sb.exe -y 100 -n 10000000 -c 16 -u http://localhost:5000/api/bigstring
```

- wait a minute or two - you should see pretty a lot of GCs happening and illustrated at **`https://localhost:5001/`**
- stop the dotnet-trace session!

Task 2. `memoryleak` app - Workstation GC (cont.)

- open the resulting **`wksgc_loadtest.nettrace`** file in PerfView - you should again see *GCStats* report under *Memory Group*. What's the *Total GC Pause*? What's the *Max GC Heap Size*? How do they compare to the Server GC results?
- how does **`GC Rollup By Generation`** table look? Do you see **`Induced`** GCs?
- look at **`GC Events by Time`** table for super important information:
  - **`Peak MB`** - "observed" maximum memory usage before the GC
  - **`After MB`** - total memory usage after the GC
  - **`Ratio Peak/After`** - the ratio, saying simply how "productive" was the GC

Task 3. `memoryleak` app - Background Workstation GC

Use *Background Workstation GC* with the same app and load test, record a session and try to find *Background GCs* happening (hint: you may need to increase the load to observe them)! They will be listed in **`GC Events by time`** table with a letter **`B`** in **`Gen`** column. What are their pause times? 👀