Memory leaks kommentar

Denne kommentar er skrevet af: Daniel F. Hauge (cph-dh136), denne kommentar er til bloggen: Memory leaks

Kommentar

Hej, jeg har læst jeres blog indlæg. Jeg finder netop dette emne utrolig interessant. Jeg har derfor fundet et par artikler som jeg syntes var rigtig spændende og relevant for lige netop dette emne.

- 1 Pointers and memory leaks in C
- 2 Garbage Collector Basics and Performance Hints
- 3 Functional Programming

Jeg vil tage udgangspunkt i disse artikler fremadrettet. Jeg vil helt klart tale for jeres problemformulering og resultater. Det ses netop igennem tiden, at memory leaks har været et meget stort problem for mange udviklere. Dem som stadig den dag idag programmere i et low level language som C, har stadig mange problemer med at styre den midlertidige lagring af data. I C skal man selv programmere hvornår der skal allokeres plads til data, og hvornår denne plads skal frigives til brug igen. Her falder rigtig mange i den fælde at glemme at frigive pladsen igen, som ender med en masse memory leaks som typisk resultere i netop denne fejl som der skrives i resultaterne: "Out of memory" error. Dette resultere i at softwaren ikke længere fungere eller kan fortsætte.1

Heldigvis er det blevet nemmere med tiden, da mere abstrakte high level programmeringssprog er udviklet. Som java eller C# som har indbygget garbage collection. En funktionalitet som gør at man ikke længere er påkrævet, til at fortælle hvornår og hvor meget plads som skal allokeres eller frigives. Dog kan man stadig snuble over en "out of memory" fejl i eksempelvis java eller C#, hvis man specifikt beder om at indsætte en for stor mængde data ind i en variable. Eksempelvis, hvis man kun har omkring 500 mb ram til rådighed, men beder om at flytte en stor(+500mb) csv fil ind i hukommelse, opstår fejlen også. Men vil typisk ikke opstå ved at glemme at tømme de små variabler man håndtere i en funktion eller lign.

I C#, er .NET garbage collector indbygget. Som gør netop ovenstående. Deres resume af hvad den funktion gør er således:2

> The .NET garbage collector provides a high-speed allocation service with good use of memory and no long-term fragmentation problems.

Essensen af lige netop dette problem opstår også fordi at den instruktion man gerne vil have computeren til at udføre typisk indeholder en form for tilstand som der skal gemmes. også kendt som "Stateful". Her er et eksempel på et tilstands fuldt program.

```
static int Tal = 10;

        static void Main(string[] args)
        {
            TælNed(Tal);
            TælNed(Tal);
            TælNed(Tal);
            TælNed(Tal);
            TælNed(Tal);
        }

        private static void TælNed(int tal)
        {
            Console.WriteLine(tal);
            Tal--;
        }
```

Her er tilstanden Tal, som ændres. Dette er en tilstand som kræver en persistent hukommelses tilstand af int tal. Men der er måder hvorpå man kan undgå tilstande. Med funktionelle programmeringssprog kan man undgå det helt. Her er et citat fra wikipedia's side omkring funktionel programmering

> In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions or declarations instead of statements. In functional code, the output value of a function depends only on the arguments that are passed to the function, so calling a function f twice with the same value for an argument x produces the same result f(x) each time; this is in contrast to procedures depending on a local or global state, which may produce different results at different times when called with the same arguments but a different program state. Eliminating side effects, i.e., changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Dette mindsker alle disse states eller data som kunne spænde ben for den midlertidig data lagring (ram).3 Dette eliminere dog ikke en mulig "Out of memory" fejl 100%, da man sagtens kan skubbe for stort et argument ind i en funktion skrevet i et funktionelt sprog. Men det mindsker ihvertfald mulighederne for at den forekommer.

For at afslutte, jeg taler for at dette er et problem. Som stadig kan plage nogle applikationer med crashes og lign. Refactoring af kode kan give et pusterum og i mange tilfælde løse problemet helt, hvis det er opstået ved over konsumering. Man kan også gå til funktionel programmering, for at mindske chancen for en "Out of memory" fejl endnu mere.