

KIV/CPP – Programování v jazyce C++

11. Profilování kódu, hledání úniků paměti

Martin Úbl

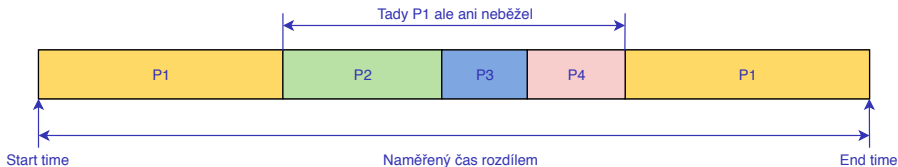
KIV ZČU

2023/2024

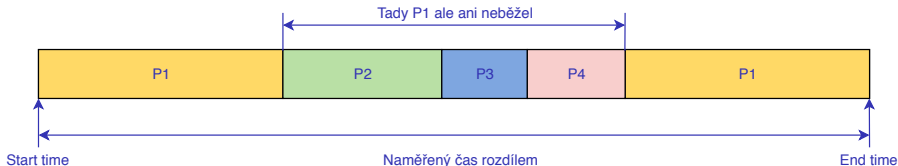
- (fyz.) množství práce za jednotku času
- počet instrukcí
 - nejednoznačné
 - každá instrukce může být provedena v rámci jiného počtu CPU cyklů
 - vektorové instrukce – početně klidně frakce CPU cyklů
- jiné měřítko?
- abstrahujeme na operace, potažmo čas
- ve výsledku nás stejně bude zajímat hlavně čas
- analýza výkonnosti programu: profilování

- čas provádění
- ovlivněn množstvím aspektů
 - vlastnosti CPU (frekvence, instr. sada, ...)
 - velikost vstupních dat
 - kvantum dat na úrovni cache
 - rychlost RAM
 - vlastnosti architektury
 - plánovač OS
 - dávkový, real-time?
 - preemptivní?
 - prioritizace
 - paralelizace
 - a mnoho dalších...
- měření by mělo být nezávislé na velké části z nich
 - např. preemptivním plánování

- čas provádění
- možné měřit na úrovni programu
- `std::chrono::steady_clock` + odečtení start a end času
- vrací hodnotu časovače s určitými vlastnostmi
- neodfiltruje dobu, kdy proces neběžel
 - pasivní čekání
 - preempce na jiný proces



- někdy je vhodné tuto dobu zahrnout
- např. čekání na I/O operace může být relevantní součástí běhu programu
- při čekání na I/O operaci preemptivní plánovače obvykle přepínají na jiné procesy



- více způsobů profilování
 - instrumentace
 - zakompilování diagnostických funkčních volání do programu
 - ruční vs. automatická
 - nutná modifikace programu, jisté zpomalení
 - vzorkování
 - periodické/událostní vzorkování zásobníku a CS:RIP
 - méně přesné, ale minimální dopad na výkon
 - interpretace
 - binární kód je pojat jako mezikód a je spuštěn ve virtuálním stroji
 - „jako Java a bytecode“
 - potenciálně velmi přesné, ale obrovský dopad na výkon
- kombinace

- instrumentace
- ruční
 - sběr časových značek, odečtení časů
 - otravné
- automatická
 - kompilátor při překladu zakompiluje dodatečná volání
 - počet volání funkcí, časy běhu
- často kombinace s vzorkováním

```
double transform(double a, double b)
{
    __mcount();
    return a*15.5+b;
}
```

- vzorkování
- prováděno buď na základě událostí nebo s pravidelnou periodou
- lze nastavit systémový časovač
 - softwarový
 - hardwarový
- obsluha časovače snímkuje stav programu
- systémové volání `profil()`

- někdy je tedy vhodné jiné měřítko
- *Hardware Performance Counters*
- speciální registry procesoru, které při události inkrementují svou hodnotu
- při přetečení generují přerušení (NMI)
- operační systém signalizuje sběrný program
 - ten zaznamená událost
 - přidá časová razítka
 - někdy snímek zásobníku

- *Hardware Performance Counters*
 - při změně programového čítače („čítač instrukcí“)
 - při výpadku rámce z L1/2/3 cache procesoru
 - při špatném predikování výsledku skoku
 - TLB miss
 - přerušení a systémová volání
 - a spousty dalších
- jsou součástí PCB a kontextu procesu
 - zaručují tedy izolovanost od ostatních procesů
 - až na sdílené prostředky, jako např. cache CPU a TLB

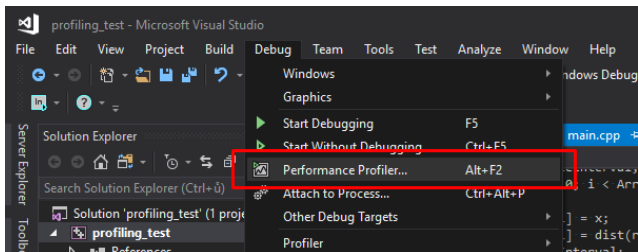
- program pro sběr profilovacích dat – profiler
- obvykle jen program, který využívá vlastností jádra OS
- „navěsí“ se na systémové události
 - časovač
 - přetečení HPC (NMI)
 - další volitelné
- od OS obdrží potřebné datové struktury
- uloží je v nějakém formátu
 - obvykle do paměti
 - jednou za čas „přesypání“ do souboru

- MS Visual Studio Performance Profiling
 - MS Windows, součást Visual Studio
 - instrumentace, vzorkování, HPC
- gprof
 - GNU/Linux, macOS, součástí gcc
 - instrumentace, vzorkování, (HPC)
- perf
 - GNU/Linux
 - vzorkování, HPC
- Intel VTune Amplifier
 - viz KIV/PPR

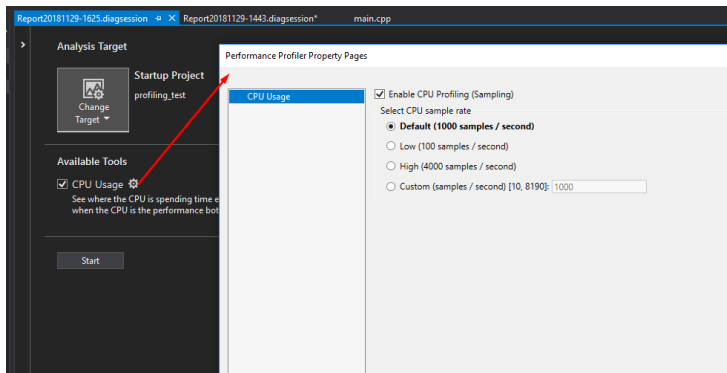
- občas může být problém s výstupy kvůli některým optimalizacím
 - implicitní inlinování funkcí
 - `-fno-inline-small-functions` (GCC)
 - `/Ob0` (MSVS)
 - nepoužití frame pointerů
 - `-fno-omit-frame-pointer` (GCC)
 - `/Oy-` (MSVS)

- čas/vzorky
 - inkluzivní
 - čas/vzorky ve funkci a všech, které volá
 - exkluzivní
 - čas/vzorky pouze v rámci instrukčního toku dané funkce

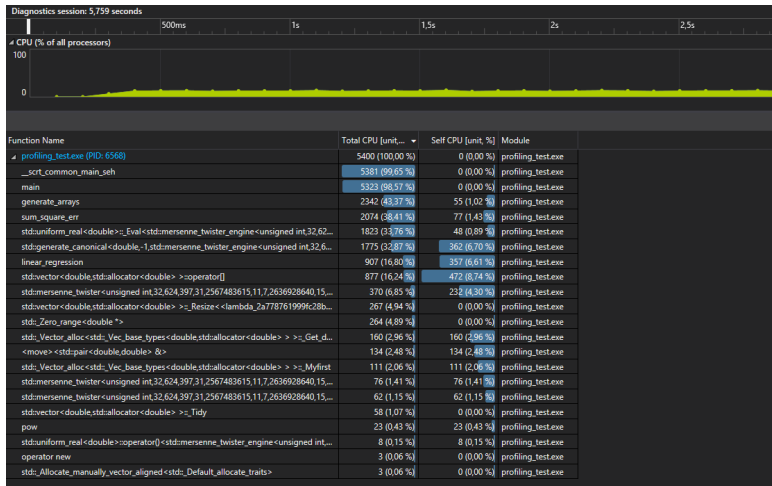
- MS Visual Studio Performance Profiling



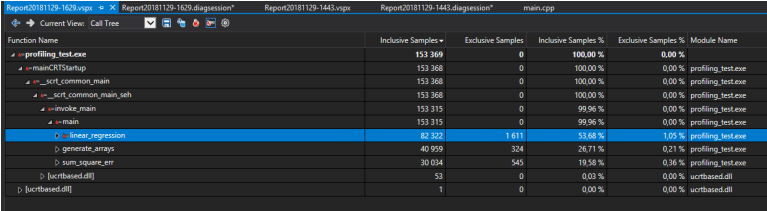
- MS Visual Studio Performance Profiling



- flat view



- call tree



Function Name	Inclusive Samples	Exclusive Samples	Inclusive Samples %	Exclusive Samples %	Module Name
profiling_test.exe	153 369	0	100,00 %	0,00 %	
mainCRTStartup	153 368	0	100,00 %	0,00 %	profiling_test.exe
__scrt_common_main	153 368	0	100,00 %	0,00 %	profiling_test.exe
__scrt_common_main_seh	153 368	0	100,00 %	0,00 %	profiling_test.exe
invoke_main	153 315	0	99,96 %	0,00 %	profiling_test.exe
main	153 315	0	99,96 %	0,00 %	profiling_test.exe
linear_regression	82 322	1 611	53,68 %	1,05 %	profiling_test.exe
generate_arrays	40 959	324	26,71 %	0,21 %	profiling_test.exe
sum_square_err	30 034	545	19,58 %	0,36 %	profiling_test.exe
[uCRTbased.dll]	53	0	0,03 %	0,00 %	uCRTbased.dll
[uCRTbased.dll]	1	0	0,00 %	0,00 %	uCRTbased.dll

- nutno kompilovat s přepínačem `-pg`
- pak klasicky spustit program
- vygeneruje se soubor `gmon.out`
- analýza příkazem `gprof`

```
g++ -pg main.cpp  
./a.out  
gprof
```

- nutno spouštět jako root
- spuštění příkazem `perf record`
 - jelikož budeme chtít zaznamenávat i strom volání, přidáme přepínač `-call-graph=fp`
- vygeneruje se soubor `perf.data`
- analýza příkazem `perf report`

```
g++ main.cpp
perf record --call-graph=fp ./a.out
perf report
```

- perf call tree

```

Samples: 864 of event 'cycles', Event count (approx.): 672421625
Children      Self Command Shared Object      Symbol
+ 97,51%    0,00% a.out [unknown] [k] 0x220e258d4c544155
+ 97,51%    0,00% a.out libc-2.27.so [.] __libc_start_main
+ 97,51%    0,00% a.out a.out [.] main
+ 88,34%    2,28% a.out a.out [.] generate_arrays
+ 39,52%   39,52% a.out a.out [.] std::generate_canonical<double, 53ul, std::mersenne_twister
+ 34,37%    4,86% a.out libc-2.27.so [.] __mcount
+ 30,44%   30,09% a.out libc-2.27.so [.] __mcount_internal
+ 12,45%    2,31% a.out a.out [.] std::vector<double, std::allocator<double> >::_M_default_app
+  8,23%    1,64% a.out [kernel.kallsyms] [k] __async_page_fault
+  6,59%    0,14% a.out [kernel.kallsyms] [k] __do_page_fault
+  6,45%    0,35% a.out [kernel.kallsyms] [k] handle_mm_fault
+  6,10%    1,06% a.out [kernel.kallsyms] [k] __handle_mm_fault
+  4,20%    4,20% a.out a.out [.] linear_regression
+  3,64%    3,53% a.out a.out [.] sum_square_err
+  3,18%    0,11% a.out [kernel.kallsyms] [k] alloc_pages_vma
+  3,15%    0,22% a.out [kernel.kallsyms] [k] get_page_from_freelist
+  3,15%    0,00% a.out [kernel.kallsyms] [k] __alloc_pages_nodemask
+  2,55%    2,55% a.out [kernel.kallsyms] [k] clear_page_erms
+  1,92%    0,00% a.out [kernel.kallsyms] [k] entry_SYSCALL_64_after_hwframe
+  1,92%    0,00% a.out [kernel.kallsyms] [k] do_syscall_64
+  1,77%    0,00% a.out [kernel.kallsyms] [k] do_munmap
+  1,77%    0,00% a.out [kernel.kallsyms] [k] unmap_region
+  1,73%    0,00% a.out libc-2.27.so [.] __munmap
+  1,73%    0,00% a.out [kernel.kallsyms] [k] __x64_sys_munmap
+  1,73%    0,00% a.out [kernel.kallsyms] [k] vm_munmap
+  1,40%    0,80% a.out [kernel.kallsyms] [k] error_entry
+  1,37%    0,68% a.out [kernel.kallsyms] [k] unmap_page_range
+  1,37%    0,00% a.out [kernel.kallsyms] [k] unmap_vmas
+  1,04%    0,23% a.out [kernel.kallsyms] [k] release_pages

```

- úniky paměti
- zapomenuté dealokace
- vynechané dealokace vlivem chybného řízení programu
- pro malý jednoúčelový program s minimem alokací *nemusí představovat problém*
 - stále bychom se ale měli snažit leaky nezavádět
- pro velký, třeba interaktivní systém je to velký problém

- možná řešení
 - nepoužívat dynamickou alokaci
 - používat RAII obalenou dynamickou alokaci
 - používat co chceme, ale odladit program s nástrojem pro detekci
- nástroje pro detekci
 - Address sanitizer (Clang, gcc, MSVS, ...)
 - CrtDebug (MSVS)
 - Visual Leak Detector (MSVS)
 - Dr. Memory (MS Windows)
 - valgrind memcheck (GNU/Linux, macOS)
 - nejen memory leaky

- princip fungování
- obalení alokací vlastní rutinou
- ta zaznamenává veškeré alokace a dealokace
- co v momentě ukončení sezení není uvolněno, reportuje

- třídy úniků paměti a problémů při používání paměti
 - new/delete mismatch
 - alokace větší, než maximální velikost heapu
 - calloc/alloca overflow
 - double free
 - use after free
 - global variable overflow
 - heap buffer overflow
 - invalid alignment
 - memcpy parameter overlap
 - stack buffer overflow (underflow)
 - use after scope
 - přístup do „otrávené“ paměti
 - a jiné

- Address Sanitizer (ASan)
- kompilován přímo do binárky
- standardní nástroj, podpora v LLVM, gcc, MSVS a jiných
- funkcemi podobný jako Valgrind/CrtDebug, ale potenciálně podstatně rychlejší a přesnější
- v MSVS lze zapnout pomocí `/fsanitize=address` (nutno doinstalovat)
- v LLVM/gcc obdobně pomocí `-fsanitize=address`

- CrtDebug – nastavení

```
_CRTDBG_DECLARE_ALLOC_MEM_DF  
_CRTDBG_DECLARE_LEAK_CHECK_DF  
_CRTDBG_DECLARE_CHECK_ALWAYS_DF);  
_CRTDBG_DECLARE_WARN_MODE_FILE);  
_CRTDBG_DECLARE_WARN_FILE_STDOUT);
```

- výpis neuvolněných bloků

```
_CRTDBG_DECLARE_DUMP_MEMORY_LEAKS());
```

- pozn. pro identifikaci místa alokace potřeba předefinovat new

```
#define MYDEBUG_NEW new( _NORMAL_BLOCK, \  
                        __FILE__, __LINE__ )  
#define new MYDEBUG_NEW
```

- vytvoření snímku heapu, a tím "checkpointu" v běhu programu

```
_CrtMemState state;  
_CrtMemCheckpoint(&state);
```

- výpis neuvolněných bloků od daného "checkpointu"

```
_CrtMemDumpAllObjectsSince(&state);
```

- stažení, instalace
 - <https://kinddragon.github.io/vld/>
- vložení do projektu

```
#include "<cesta k vld>\include\vld.h"  
#pragma comment(lib, "<cesta k vld>/vld_x64.lib")
```

- start

```
VLDEnable();
```

- výpis neuvolněných bloků

```
VLDReportLeaks();
```

- pozn. nástroj již není vyvíjen

- díky LD_PRELOAD překryje alokátory bez nutnosti zasahovat do kódu
- není třeba speciálního kódu nebo překladu
 - je nutné však přeložit s debug info (přepínač `-g` pro `g++`)
- zpomaluje ale výrazně běh programu (klidně 20x)
- spuštění

```
valgrind --tool=memcheck ./program param1
```

- někdy vhodné připojit příznaky pro zjištění původu a maximalizaci hledání

```
valgrind --tool=memcheck --leak-check=full  
--track-origins=yes ./program param1
```