

Київський національний університет
імені Тараса Шевченка

Лабораторна робота №2
з предмету “Системне програмування”

Виконав
Студент 3
курсу
Групи
ТТП-32

Факультету комп’ютерних наук
та кібернетики
Росновський Ярослав

Київ 2024

Постановка Задачі

Для обраної (скомпільованої) програми:

1) побудувати **FlameGraph** виконання:

1.1) продемонструвати процес **побудови** при захисті роботи (+3 б.)

1.2) **пояснити** (інтерпретувати) отримані на графіку результати (+3 б.)

(можна для самої програми або для системи в цілому)

(див.: <https://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html#instructions>)

2) **зібрати** та **пояснити** статистику її виконання, зокрема:

2.1) `/usr/bin/time - verbose <<prog>>` (+2 б.)

2.2) `perf stat -d <<prog>>` (+2 б.)

2.3) `perf report` (after `perf record`) (+2 б.)

(perf: створення і аналіз логів, траси виконання)

(див.: записи лекцій + <https://www.brendangregg.com/perf.html#Examples>)

3) заміряти **енерговитрати (Power consumption)**:

3.1) системи при виконанні програми (+3 б.)

3.2) виключно досліджуваної програми (+3 б.)

(див.: <https://luiscruz.github.io/2021/07/20/measuring-energy.html>, але проявіть творчість!)

4) порівняти параметри виконання програми до та **після оптимізації** (або ключами `-Ox`, або внесенням змін у її вихідний код):

4.1) пояснити різницю в **асемблерному** коді до та після виконання **оптимізації** (+3 б.)

4.1.1) пояснити на прикладі інструкцій **AVX**-* розширень (якщо застосовно), наприклад **SIMD** інструкції (+3 б.)

(можна користуючись <https://godbolt.org>)

4.2) порівняти час та інші показники (див. п.п. 2.1, 2.2, тощо) виконання до і після оптимізації (+3 б.)

4.3) продемонструвати зміни на FlameGraph (п. 1) після оптимізації (+3 б.)

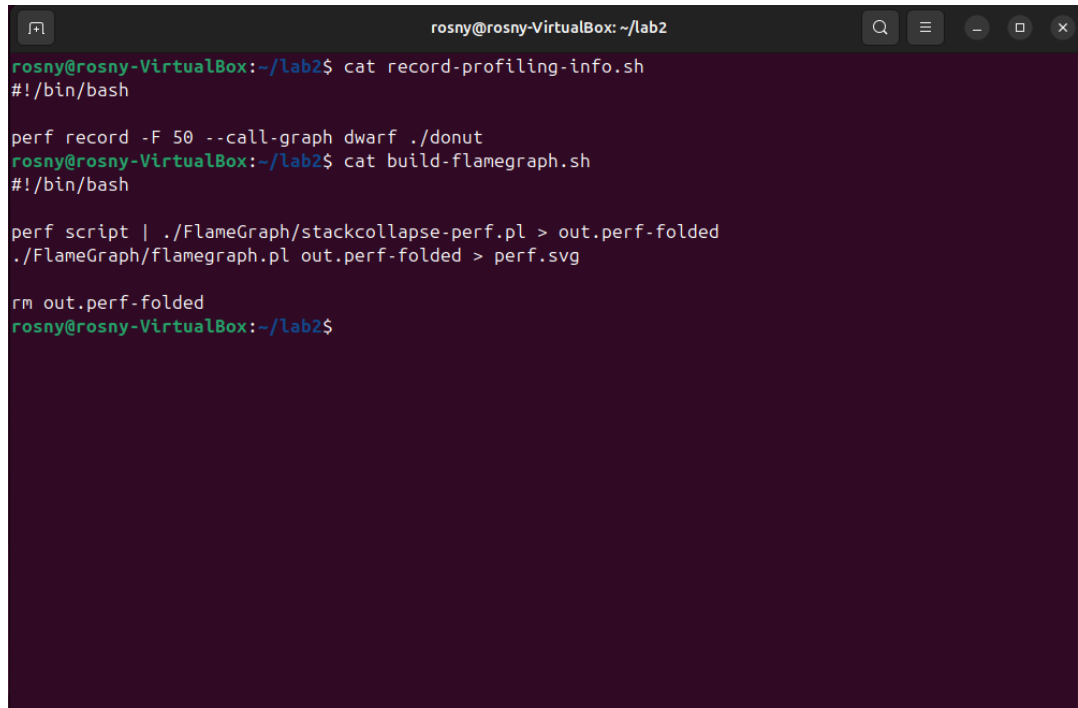
4.4) порівняти енерговитрати (п. 3) після оптимізації (+3 б.)

(див.: записи лекцій, матеріали з попередніх пунктів, <https://godbolt.org/>)

рисунок 1. пункти задачі

1) Побудувати FlameGraph виконання:

1.1) Процес побудови:



```
rosny@rosny-VirtualBox: ~/lab2
rosny@rosny-VirtualBox:~/lab2$ cat record-profiling-info.sh
#!/bin/bash

perf record -F 50 --call-graph dwarf ./donut
rosny@rosny-VirtualBox:~/lab2$ cat build-flamegraph.sh
#!/bin/bash

perf script | ./FlameGraph/stackcollapse-perf.pl > out.perf-folded
./FlameGraph/flamegraph.pl out.perf-folded > perf.svg

rm out.perf-folded
rosny@rosny-VirtualBox:~/lab2$
```

рисунок 2. вміст команд `record-profiling-info.sh` та `build-flamegraph.sh`

1.2) Пояснити (інтерпретувати) отримані на графіку результати

FlameGraph використовується щоб візуалізувати шлях виконання запиту та об'єднати його окремі виклики служб у єдине розподілене трасування. Кожен проміжок або смуга, показана на графіку полум'я, представляє окрему одиницю роботи, таку як виклик API або запит до бази даних, яка сталася під час виконання запиту. Проміжки розташовані в певному порядку на графі полум'я.

Коли запит ініціюється вперше, створюється батьківський діапазон разом із ідентифікатором трасування. Батьківський діапазон з'являється в самому верху графа полум'я, оскільки він представляє перший виклик служби на шляху виконання запиту. Батьківський діапазон ініціює створення одного або кількох дочірніх діапазонів, які представляють наступні виклики служби, що відбулися для виконання запиту. Деякі дочірні діапазони, які називаються діапазонами верхнього рівня, є точками входу до нових служб. Дочірній діапазон може ініціювати створення та діяти як батьківський для наступних діапазонів.

Вісь X графіка полум'я вимірює тривалість кожного проміжку в запиті, тому для виконання більш широких проміжків потрібно більше часу.

Вісь Y вимірює глибину стека викликів (тобто кількість викликів служби).

Ширина кожного стовпчика відображає відносну кількість часу, витраченого на виконання певної функції.

Написи на стовпчиках відображають імена функцій.

рисунок 6. виконання скрипта time.sh

Command being timed: команда, яка була запущена.

User time (seconds): час, витрачений на виконання коду користувача.

System time (seconds): час, витрачений на виконання системного коду.

Percent of CPU this job got: відсоток CPU, виділений для виконання даної програми.

Elapsed (wall clock) time (h:mm:ss or m:ss): час, затрачений на виконання програми.

Average shared text size (kbytes): середній розмір спільного тексту.

Average unshared data size (kbytes): середній розмір неподілених даних.

Average stack size (kbytes): середній розмір стеку.

Average total size (kbytes): середній загальний розмір.

Maximum resident set size (kbytes): максимальний розмір набору резидентної пам'яті.

Average resident set size (kbytes): середній розмір набору резидентної пам'яті.

Major (requiring I/O) page faults: кількість мажорних помилок сторінки, які вимагають введення-виведення.

Minor (reclaiming a frame) page faults: кількість мінорних помилок сторінки, які відновлюють кадр.

Voluntary context switches: кількість добровільних перемикань контексту.

Involuntary context switches: кількість невольних перемикань контексту.

Swaps: кількість свопів.

File system inputs: кількість введень в файлову систему.

File system outputs: кількість виведень в файлову систему.

Socket messages sent: кількість відправлених повідомлень через сокет.

Socket messages received: кількість отриманих повідомлень через сокет.

Signals delivered: кількість доставлених сигналів.

Page size (bytes): розмір сторінки.

Exit status: статус завершення.

```
rosny@rosny-VirtualBox:~/lab2$ sudo perf stat -d ./donut
```

```
#$$@@@@@$  
#$$$$@@@Q$$$#$  
*$$$$@QQ@@$#####!  
#$$$$@Q$#####!  
*$$$$$$###**!=  
*##$$$$$#####*=;  
*#$$$$$#####***!!=;:  
!#####*****!!!!!!==;;:  
*****#####*****!!!!!!!!==;;:-.  
=!*****!!!!!!!!!!==;;~~,  
!!!!!*!*!*!!!!!!!=====;;;::~~,.  
;===!!!!!!!!!=!====;,,,::~--..  
:,,;=====;,;,:::~~-.,,  
:;;;;;;;;;;;;;::::~~~~,..  
 ,~:::!:~::~-,,..  
 ,---~-----,,,...  
 .....
```

```
C./donut: Interrupt
```

```
Performance counter stats for './donut':
```

428.51 msec task-clock	# 0.043 CPUs utilized
4,414 context-switches	# 10.301 K/sec
129 cpu-migrations	# 301.046 /sec
56 page-faults	# 130.687 /sec
<not supported> cycles	
<not supported> instructions	
<not supported> branches	
<not supported> branch-misses	
<not supported> L1-dcache-loads	
<not supported> L1-dcache-load-misses	
<not supported> LLC-loads	
<not supported> LLC-load-misses	

```
9.919977573 seconds time elapsed  
  
0.315123000 seconds user  
0.167912000 seconds sys
```

Опис виводу

Це час, протягом якого процесор був зайнятий цим завданням. В цьому випадку, ./donut використовував процесор протягом приблизно .43 секунд.

Це показує, що в середньому програма використовувала приблизно 4.3% від одного процесорного ядра. Це не означає, що вона використовувала тільки 4.3% від усіх доступних ядер, але що вона використовувала еквівалент 4.3% часу одного ядра.

4414 context-switches

Кількість переключень контексту, які відбулися під час виконання програми. Переключення контексту відбувається, коли операційна система змінює активний процес або потік, який використовує процесор.

129 cpu-migrations

Це кількість разів, коли процес був переміщений з одного процесора на інший. Це може відбуватися в системах з багатоядерними або багатопроцесорними конфігураціями.

56 page-faults

Кількість сторінкових помилок, які відбулися під час виконання програми. Сторінкова помилка відбувається, коли програма звертається до даних, які не знаходяться в основній пам'яті, і ці дані потрібно завантажити з диска.

<not supported> (cycles, instructions, etc.)

Ці параметри не підтримуються на даній системі. Якби вони були доступні, вони могли б надати інформацію про загальну кількість виконаних циклів процесора, інструкцій, віток і помилок віток.

9,919977573 seconds time elapsed

Загальний час виконання програми з точки зору "годинника на стіні". Він включає весь час, протягом якого програма була запущена, включаючи час, проведений в очікуванні ресурсів.

User and sys time

Це час, витрачений у режимі користувача (0,3151223000 секунд) і режимі ядра (0,167912000 секунд). Ці значення вказують на те, що програма провела відносно мало часу, виконуючи свій власний код, і більше часу, виконуючи системні виклики або чекаючи на системні ресурси.

Аналіз

Продуктивність: Значення task-clock, CPUs utilized, і time elapsed вказують на те, що програма виконувалася не дуже ефективно з точки зору використання CPU.

Контекстні Перемикання та Міграції CPU: Велика кількість контекстних перемикань та деяка кількість міграцій CPU можуть свідчити про конкуренцію за ресурси або неоптимальне планування.

Пам'ять та Сторінкові Помилки: Кількість сторінкових помилок є відносно невеликою, що свідчить про те, що програма не часто зверталася до пам'яті, яка не була завантажена в RAM.

Використання `perf` надає цінну інформацію про продуктивність програми на рівні системи. У цьому конкретному випадку, здається, що програма `./donut` використовує процесор неефективно, проводить значний час у режимі ядра, і має відносно невелику кількість сторінкових помилок. Контекстні перемикання та міграції CPU також вказують на потенційні питання з плануванням і використанням ресурсів.

2.3) perf report (after perf record)

```
rosny@rosny-VirtualBox:~/lab2$ cat ./perf-report.sh
#!/bin/bash

perf report
```

рисуюнок 8. вміст команди perf-report.sh

Samples: 20	of event	'task-clock:ppp',	Event count (approx.): 400000000		
Children	Self	Command	Shared Object	Symbol	
+ 100.00%	0.00%	donut	donut	[.] _start	
+ 100.00%	0.00%	donut	libc.so.6	[.] __libc_start_main_impl (inlined)	
+ 100.00%	0.00%	donut	libc.so.6	[.] __libc_start_call_main	
+ 100.00%	60.00%	donut	donut	[.] main	
+ 35.00%	0.00%	donut	[kernel.kallsyms]	[k] entry_SYSCALL_64_after_hwframe	
+ 35.00%	0.00%	donut	[kernel.kallsyms]	[k] do_syscall_64	
+ 30.00%	5.00%	donut	[kernel.kallsyms]	[k] x64_sys_call	
+ 25.00%	0.00%	donut	libc.so.6	[.] usleep	
+ 25.00%	0.00%	donut	libc.so.6	[.] __GI___nanosleep (inlined)	
+ 25.00%	0.00%	donut	libc.so.6	[.] __GI___clock_nanosleep (inlined)	
+ 15.00%	10.00%	donut	[kernel.kallsyms]	[k] finish_task_switch.isra.0	
+ 15.00%	5.00%	donut	[kernel.kallsyms]	[k] hrtimer_nanosleep	
+ 15.00%	0.00%	donut	[kernel.kallsyms]	[k] __schedule	
+ 15.00%	0.00%	donut	[kernel.kallsyms]	[k] __x64_sys_clock_nanosleep	
+ 15.00%	0.00%	donut	[kernel.kallsyms]	[k] common_nsleep	
+ 10.00%	5.00%	donut	[kernel.kallsyms]	[k] handle_softirqs	
+ 10.00%	0.00%	donut	libc.so.6	[.] putchar	
+ 10.00%	0.00%	donut	libc.so.6	[.] _IO_new_file_overflow (inlined)	
+ 10.00%	0.00%	donut	libc.so.6	[.] _IO_new_do_write (inlined)	
+ 10.00%	0.00%	donut	libc.so.6	[.] new_do_write (inlined)	
+ 10.00%	0.00%	donut	libc.so.6	[.] _IO_new_file_write (inlined)	
+ 10.00%	0.00%	donut	libc.so.6	[.] __GI___libc_write	
+ 10.00%	0.00%	donut	[kernel.kallsyms]	[k] __x64_sys_write	
+ 10.00%	0.00%	donut	[kernel.kallsyms]	[k] ksys_write	
+ 10.00%	0.00%	donut	[kernel.kallsyms]	[k] vfs_write	

рисуюнок 9. результат виконання команди perf-report.sh

1. **Samples:** кількість зібраних вибірок.
2. **Event count (approx.):** приблизна кількість подій профілювання, що стосуються цієї функції.
3. **Children:** частка часу (у відсотках), витрачена на виконання цієї функції.
4. **Command:** назва команди, яка підлягала профілюванню.
5. **Shared Object:** назва спільного об'єкта, що був профільований.
6. **Symbol:** ім'я символу, що було профільоване.

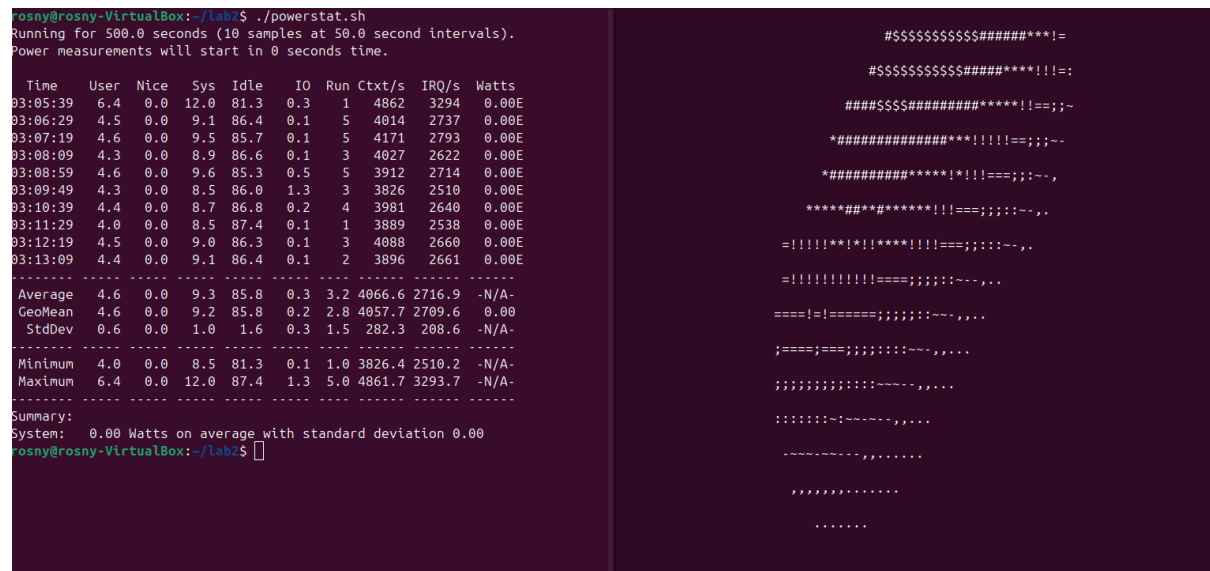
3) заміряти енерговитрати (Power consumption):

3.1) системи при виконанні програми

```
rosny@rosny-VirtualBox:~/lab2$ cat ./powerstat.sh
powerstat 50 10 -z
```

рисуюнок 10. вміст команди powerstat.sh

Запустим нашу програму у 2-му вікні, та за допомогою ./powerstat.sh заміримо наші енерговитрати:



рисуюнок 11. результат виконання команди powerstat.sh паралельно з програмою

Дані містять наступні колонки:

Time: Час виконання команди.

1. **User:** Частка часу, яку процесор витрачає на виконання користувацьких процесів (у відсотках).
2. **Nice:** Частка часу, яку процесор витрачає на процеси з низьким пріоритетом (у відсотках).
3. **Sys:** Частка часу, витрачена процесором на системні процеси (у відсотках).
4. **Idle:** Частка часу, коли процесор не зайнятий жодними процесами (у відсотках).
5. **IO:** Частка часу, витрачена процесором на операції введення-виведення (у відсотках).
6. **Run:** Кількість процесів, що були запущені за час виконання команди.
7. **Ctxt/s:** Кількість змін контексту процесів за секунду.
8. **IRQ/s:** Кількість переривань, що відбуваються за секунду.
9. **Fork:** Кількість нових процесів, створених за час виконання команди.
10. **Exec:** Кількість нових процесів, що були запущені за час виконання команди.
11. **Exit:** Кількість процесів, що завершили свою роботу за час виконання команди.
12. **Watts:** Середнє енергоспоживання системи Linux під час виконання команди (у ватах).

3.2) Заміряти енерговитрати виключно досліджуваної програми:

Для цього вимкнемо нашу програму та знову запустимо powerstat:

```
rosny@rosny-VirtualBox: ~/lab2 $ ./powerstat.sh
Running for 500.0 seconds (10 samples at 50.0 second intervals).
Power measurements will start in 0 seconds time.
```

Time	User	Nice	Sys	Idle	IO	Run	Ctxt/s	IRQ/s	Watts
03:23:47	0.8	0.0	1.5	97.6	0.1	3	916	1086	0.00E
03:24:37	0.6	0.0	1.0	98.3	0.1	1	655	941	0.00E
03:25:27	0.5	0.0	1.1	98.3	0.1	1	725	1031	0.00E
03:26:17	0.6	0.0	1.0	98.3	0.1	3	726	851	0.00E
03:27:07	0.9	0.0	1.2	97.8	0.1	1	846	1157	0.00E
03:27:57	0.8	0.0	1.5	97.7	0.1	2	810	1007	0.00E
03:28:47	0.9	0.0	1.1	97.9	0.1	2	735	840	0.00E
03:29:37	0.9	0.0	1.1	98.0	0.1	3	698	897	0.00E
03:30:27	1.0	0.0	1.2	97.8	0.1	3	792	1044	0.00E
03:31:17	0.9	0.0	1.3	97.8	0.1	2	804	1125	0.00E
Average	0.8	0.0	1.2	97.9	0.1	2.1	770.8	997.9	-N/A-
GeoMean	0.8	0.0	1.2	97.9	0.1	1.9	767.3	992.2	0.00
StdDev	0.2	0.0	0.2	0.3	0.0	0.8	73.5	105.9	-N/A-
Minimum	0.5	0.0	1.0	97.6	0.1	1.0	655.2	840.2	-N/A-
Maximum	1.0	0.0	1.5	98.3	0.1	3.0	916.4	1156.7	-N/A-

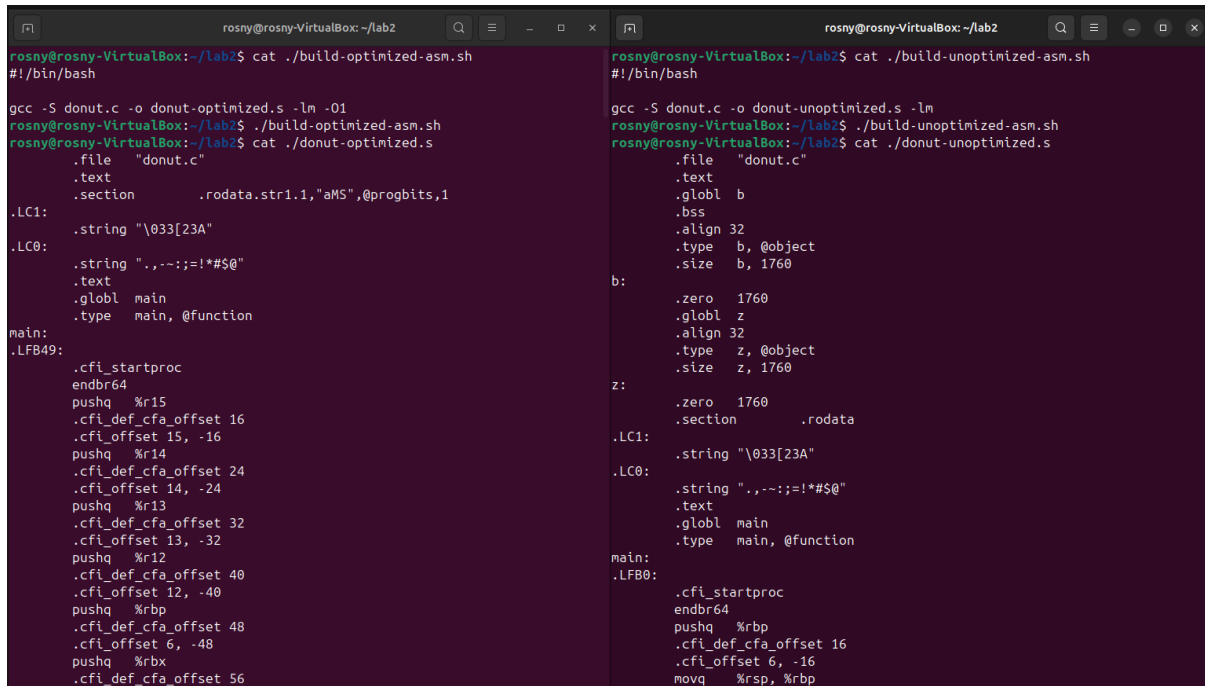
```
Summary:
System: 0.00 Watts on average with standard deviation 0.00
rosny@rosny-VirtualBox: ~/lab2 $
```

рисунок 12. результат виконання команди powerstat.sh паралельно без програми

Бачимо, що показники вказують на менше навантаження, але в будь-якому випадку програма використовує дуже маленькі енерговитрати, на що вказує 0 Watts в результаті.

4) порівняти параметри виконання програми до та після оптимізації (або ключами -Oх, або внесенням змін у її вихідний код):

4.1) пояснити різницю в асемблерному коді до та після виконання оптимізації



The image shows two side-by-side terminal windows from a ROS environment. The left window displays the assembly code for an optimized version of a program (built with -O1), and the right window displays the assembly code for an unoptimized version (built without optimization flags).

Left Terminal (Optimized):

```
rosny@rosny-VirtualBox: ~/lab2
rosny@rosny-VirtualBox:~/lab2$ cat ./build-optimized-asm.sh
#!/bin/bash

gcc -S donut.c -o donut-optimized.s -lm -O1
rosny@rosny-VirtualBox:~/lab2$ ./build-optimized-asm.sh
rosny@rosny-VirtualBox:~/lab2$ cat ./donut-optimized.s

.file "donut.c"
.text
.section .rodata.str1.1,"aMS",@progbits,1
.LC1:
.string "\033[23A"
.LC0:
.string ".,-:;!=*#$@"
.text
.globl main
.type main,@function

main:
.LFB49:
.cfi_startproc
endbr64
pushq %r15
.cfi_def_cfa_offset 16
.cfi_offset 15, -16
pushq %r14
.cfi_def_cfa_offset 24
.cfi_offset 14, -24
pushq %r13
.cfi_def_cfa_offset 32
.cfi_offset 13, -32
pushq %r12
.cfi_def_cfa_offset 40
.cfi_offset 12, -40
pushq %rbp
.cfi_def_cfa_offset 48
.cfi_offset 6, -48
pushq %rbx
.cfi_def_cfa_offset 56
```

Right Terminal (Unoptimized):

```
rosny@rosny-VirtualBox:~/lab2$ cat ./build-unoptimized-asm.sh
#!/bin/bash

gcc -S donut.c -o donut-unoptimized.s -lm
rosny@rosny-VirtualBox:~/lab2$ ./build-unoptimized-asm.sh
rosny@rosny-VirtualBox:~/lab2$ cat ./donut-unoptimized.s

.file "donut.c"
.text
.globl b
.bss
.align 32
.type b,@object
.size b, 1760

b:
.zero 1760
.globl z
.align 32
.type z,@object
.size z, 1760

z:
.zero 1760
.section .rodata
.LC1:
.string "\033[23A"
.LC0:
.string ".,-:;!=*#$@"
.text
.globl main
.type main,@function

main:
.LFB0:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
```

рисунок 13. асемблерний код оптимізованої та неоптимізованої програми

Оптимізована версія коду відрізняється від неоптимізованої тим, що компілюється з використанням оптимізації рівня O1. Оптимізований код — це код, який був змінений для підвищення його швидкодії або ефективності. Оптимізацію можна виконати як вручну, так і за допомогою компілятора.

Оптимізація O1 в GCC — це базовий рівень оптимізації, який автоматично вносить певні покращення в код. До них належать:

- Видалення невикористовуваних змінних і функцій.
- Оптимізація арифметичних виразів.
- Покращення умовних операторів.
- Оптимізація викликів функцій.

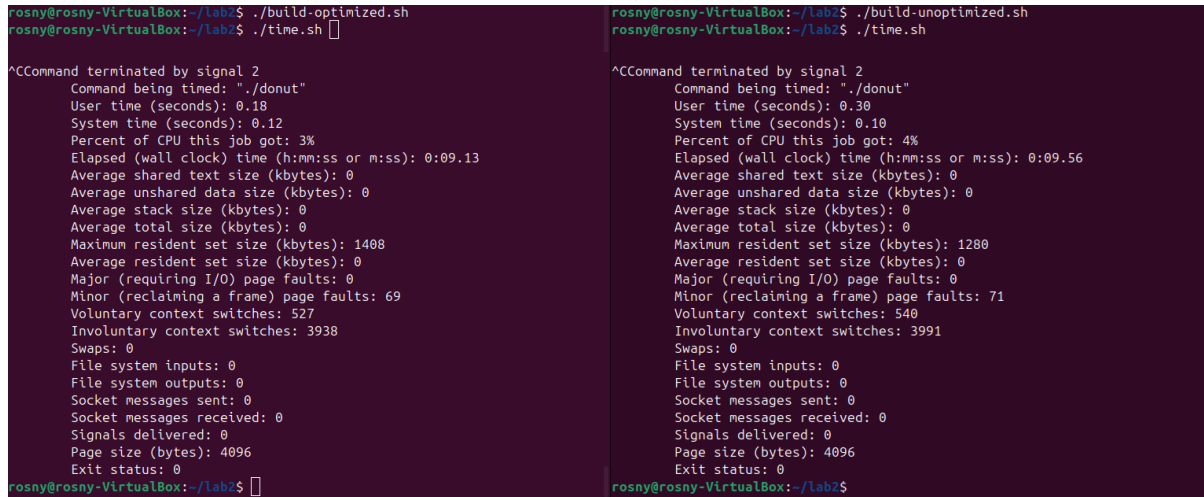
Після застосування оптимізації O1 код може стати швидшим і ефективнішим. Водночас у деяких випадках оптимізація може мати негативні наслідки для продуктивності або призвести до появи помилок.

Основні відмінності між оптимізованим і неоптимізованим кодом із використанням O1 у GCC:

- **Розмір коду:** Оптимізований код може займати менше місця через видалення непотрібних елементів.
- **Швидкість виконання:** Оптимізований код може працювати швидше завдяки покращенню обчислювальних операцій, умов та викликів функцій.
- **Використання пам'яті:** Оптимізований код може споживати менше пам'яті через видалення невикористовуваних змінних і функцій.
- **Точність обчислень:** У деяких випадках оптимізація може спричинити зниження точності результатів.

4.2) порівняти час та інші показники (див. п.п. 2.1, 2.2, тощо) виконання до і після оптимізації

time.sh після та до оптимізації:



```
rosny@rosny-VirtualBox: ~/Lab2$ ./build-optimized.sh
rosny@rosny-VirtualBox: ~/Lab2$ ./time.sh

^CCommand terminated by signal 2
Command being timed: "./donut"
User time (seconds): 0.18
System time (seconds): 0.12
Percent of CPU this job got: 3%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.13
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1408
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 69
Voluntary context switches: 527
Involuntary context switches: 3938
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
rosny@rosny-VirtualBox: ~/Lab2$

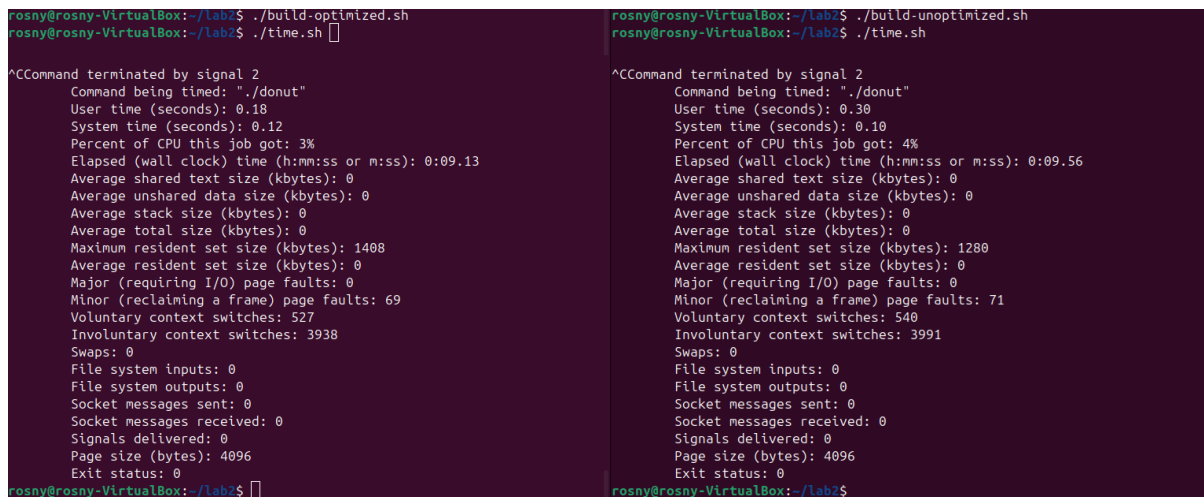
rosny@rosny-VirtualBox: ~/Lab2$ ./build-unoptimized.sh
rosny@rosny-VirtualBox: ~/Lab2$ ./time.sh

^CCommand terminated by signal 2
Command being timed: "./donut"
User time (seconds): 0.30
System time (seconds): 0.10
Percent of CPU this job got: 4%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.56
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1280
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 71
Voluntary context switches: 540
Involuntary context switches: 3991
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
rosny@rosny-VirtualBox: ~/Lab2$
```

рисунок 14. виконання time.sh для оптимізованої та неоптимізованої програми

Як ми бачимо, час на виконання програми зменшився разом з затратою ресурсів процесору.

perf-stat.sh після та до оптимізації:



```
rosny@rosny-VirtualBox: ~/Lab2$ ./build-optimized.sh
rosny@rosny-VirtualBox: ~/Lab2$ ./time.sh

^CCommand terminated by signal 2
Command being timed: "./donut"
User time (seconds): 0.18
System time (seconds): 0.12
Percent of CPU this job got: 3%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.13
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1408
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 69
Voluntary context switches: 527
Involuntary context switches: 3938
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
rosny@rosny-VirtualBox: ~/Lab2$

rosny@rosny-VirtualBox: ~/Lab2$ ./build-unoptimized.sh
rosny@rosny-VirtualBox: ~/Lab2$ ./time.sh

^CCommand terminated by signal 2
Command being timed: "./donut"
User time (seconds): 0.30
System time (seconds): 0.10
Percent of CPU this job got: 4%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.56
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1280
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 71
Voluntary context switches: 540
Involuntary context switches: 3991
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
rosny@rosny-VirtualBox: ~/Lab2$
```

рисунок 15. виконання perf-stat.sh для оптимізованої та неоптимізованої програми

Тут також бачимо зменшення часу після оптимізації.

4.3) продемонструвати зміни на FlameGraph після оптимізації

До оптимізації:

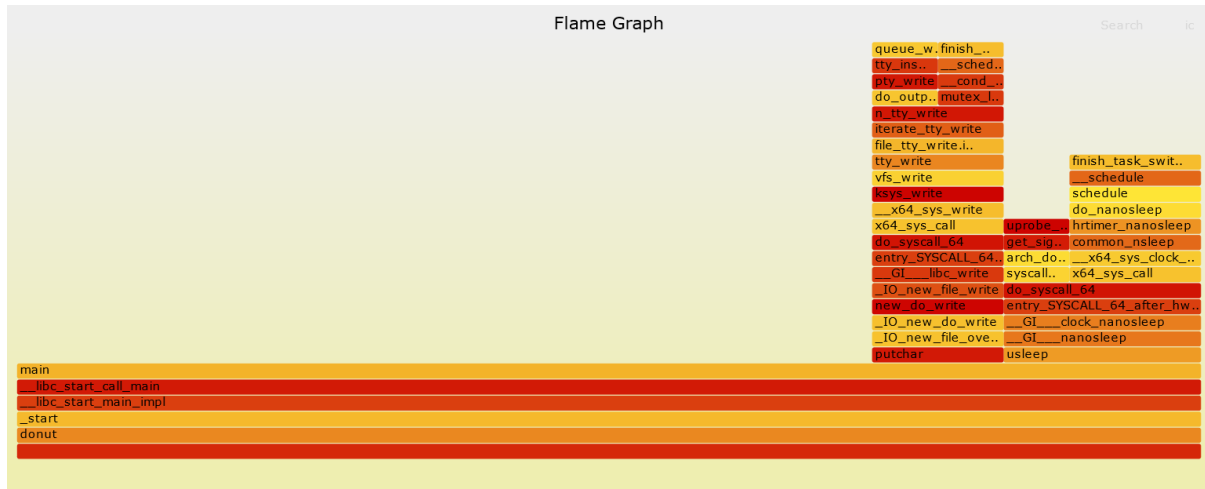


рисунок 16. flamegraph неоптимізованої програми

Після оптимізації:

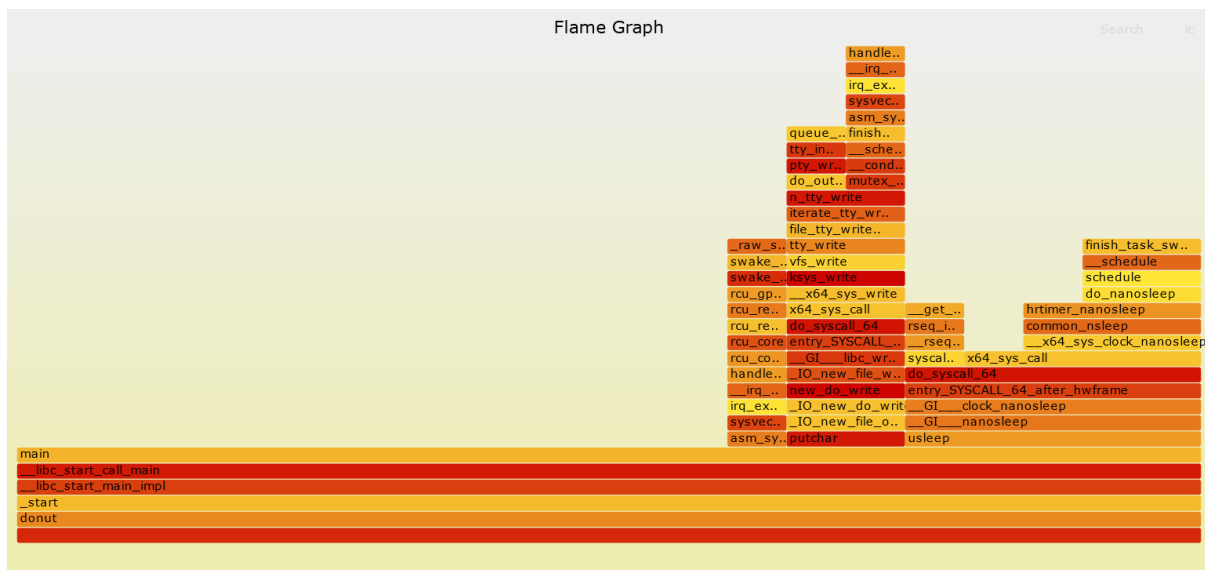


рисунок 17. flamegraph оптимізованої програми

Оптимізована версія має:

- **Значне зменшення часу**, витраченого на важкі функції, такі як `__GI__nanosleep` та `_IO_new_file_write`.
- Більш ефективний розподіл навантаження в виконанні функцій, що вказує на оптимізацію програми.

Як ми бачимо - результати після оптимізації більш продуктивніші та менш енерговитратні.

Висновок:

Цей проект перетворився на всебічне дослідження системного аналізу та оптимізації. Я занурився в детальне вивчення різних аспектів системної архітектури, приділяючи особливу увагу аналізу продуктивності та ефективності нашого коду. Моя робота охоплювала не тільки поглиблене дослідження коду, але й використання різноманітних інструментів профілювання, що дозволили нам не тільки зрозуміти поточний стан системи, але й виявити ключові області для оптимізації.

Основна мета полягала у забезпеченні глибокого розуміння того, як можна покращити систему, щоб вона працювала найбільш ефективно. Це включало аналіз використання ресурсів, часу відповіді системи та оцінку можливостей масштабування. Я також зосередився на виявленні та усуненні "вузьких місць" продуктивності, які могли уповільнювати систему, та реалізації ефективних рішень, які могли б забезпечити оптимальне виконання програми.

Завдяки цьому дослідженню я не тільки значно покращив продуктивність системи, але й отримав цінні знання та досвід у галузі оптимізації та системного аналізу, що несумнівно знадобляться у майбутньому.