ARM 프로세서 메모리 사용량 확인

https://youtu.be/ef6lvQ23G2k

정보컴퓨터공학과 송경주

한성대학교
HANSUNG UNIVERSITY

CryptoCraft LAB

메모리 사용량 측정 – Valgrind

- Valgrind를 사용하여 메모리 사용량을 측정할 수 있음
- 측정 방법
  1. 메모리 사용량을 측정할 소스코드 컴파일
     - 이때, 반복 횟수는 1회로 고정, Valgrind는 동적할당이 발생할 때마다 이를 메모리 소모로 기록함
  2. Valgrind 명령어를 통해 바이너리 파일 실행
     - valgrind --tool=massif --stacks=yes ./(측정파일이름)
  3. 출력된 로그를 확인하여 heap_tree=peak 일 때, Stack 크기와 Haep 크기 합산
- MAC OS ARM processer에서 동작하지 않았음

메모리 사용량 측정 – Valgrind

Supported Platforms

Current

Valgrind supports the following platforms:

- **X86/Linux:** up to and including SSSE3, but not higher -- no SSE4, AVX, AVX2. This target is in maintenance mode now..
- **AMD64/Linux:** up to and including AVX2. This is the primary development target and tends to be well supported.
- **PPC32/Linux, PPC64/Linux, PPC64LE/Linux:** up to and including Power8.
- **S390X/Linux:** supported.
- **ARM/Linux:** supported since ARMv7.
- **ARM64/Linux:** supported for ARMv8.
- **MIPS32/Linux, MIPS64/Linux:** supported.
- **X86/FreeBSD, AMD64/FreeBSD**: supported since FreeBSD 11.3.
- **ARM64/FreeBSD**: supported since FreeBSD 14.
- **X86/Solaris, AMD64/Solaris, X86/illumos, AMD64/illumos**: supported since Solaris 11.
- **X86/Darwin (10.5 to 10.13), AMD64/Darwin (10.5 to 10.13):** supported.
- **ARM/Android, ARM64/Android, MIPS32/Android, X86/Android:** supported.

On Linux, you must be running kernel 3.0 or later, and glibc 2.5.X or later. On Mac OS X you must be running 10.9.x or later.

For details of which distributions the current release (valgrind-3.23.0) builds and runs its regression tests on, see the release notes.

메모리 사용량 측정 – Valgrind

Valgrind for arm64 based macos

hey guys I use MacBook with m1 chip and apparently I can't use valgrind because it's not supported on this architecture. I'm new to the field. I tried leaks command with the my program's pid. I don't trust its results because it doesn't show leaks when I created a an integer array, malloced it and initialized random data. Printfed them and deliberately didn't free. I added sleep(60). During the whole run of the program, leaks command didn't show any leaks. Nor did the address sanitizer. What else can I try?

5 3 Share

Add a Comment

Sort by: Best Search Comments

spank12monkeys • 7mo ago

-fsanitizer=leaks does not seem to be available on Apple's clang 15.0.0 that I have on my arm mac running Ventura. It does have a working sanitizer=address though. Maybe this is because Apple has a whole bunch of malloc debugging tools? Not sure.

To use the leaks tool, if you have a file like this

valgrind가 찾을 수 있는 오류 중 일부를 잡을 수 있지만 메모리 누수는 찾을 수 없음
→ 즉, valgrind를 완전히 대체할 수 없음

Valgrind Alternatives for Macos?

Question

After finally getting gdb setup on Macos, I thought it was time to try to spend time getting used to other C development tools. From what I can tell Valgrind seems to be a key tool in C development. I tried getting it set up but it turns out it is no longer supported on Macos.

I looked into some preprocessor options in gcc https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html But it looks like some options are not supported on mac such as `-fsanitize=leak`.

On a positive note `-fsanitize=undefined` seemed to work, so I will count that as win.

What tools should I be looking at for debugging memory issues without resorting to duel boots, virtual machines or docker?

3 6 Share

How to get Valgrind on MacOS?

IDE

I am on Lecture 4 -Memory And I am taking the course on my own computer (running MacOS Big Sur) and I could install all the CS50 tools and a debugger on VSCode. But it seems that Valgrind isn't available on brew. How can I install it? Thanks.

2 9 Share

Add a Comment

Sort by: Best Search Comments

Fuelled_By_Coffee • 3년 전

Valgrind is sadly not compatible with MacOS. You'll have to rely on check50 to check for memory leaks.

You can compile your code with address sanitizer (I recommend it), just add `-fsanitize=address` to your CFLAGS environment variable. That will catch some of the errors that valgrind would find (as well as some that valgrind would ignore), but it can't find memory leaks.

2 Reply Award Share ...

메모리 사용량 측정 – LeakSanitizer

README GPL-3.0 license

Welcome to the LeakSanitizer!

The LeakSanitizer is a tool designed to debug memory leaks.

It can be used with almost any programming language that compiles to native machine code.
Officially supported are currently: **C**, **C++**, **Objective-C**, **Swift**.

> Important
>
> **Objective-C** and **Swift** objects are never considered to become memory leaks - even in the case of strong reference cycles.

This sanitizer has been optimized for both **macOS** and **Linux** - all memory leaks are detected on both platforms.

Quickstart

Use the LeakSanitizer to check for memory leaks as follows:

1. Clone the repository: `git clone --recursive https://github.com/mhahnFr/LeakSanitizer.git`
2. Build it: `cd LeakSanitizer && make`
3. Link your code with: `-L<path/to/library> -llsan`

Installation

Get started by either downloading a prebuilt version of this sanitizer here. Alternatively you can also build it from source:

1. Clone the repository: `git clone --recursive https://github.com/mhahnFr/LeakSanitizer.git`
2. go into the cloned repository: `cd LeakSanitizer`
3. and build the library: `make`

Or in one step:

```
git clone --recursive https://github.com/mhahnFr/LeakSanitizer.git && cd LeakSanitizer && make
```

Once you have a copy of this sanitizer you can install it using the following command:

```
make INSTALL_PATH=/usr/local install
```

If you downloaded a release you can simply move the headers and the library anywhere you like.

Uninstallation

Uninstall the sanitizer by simply removing its library and its header files from the installation directory.
This can be done using the following command:

```
make INSTALL_PATH=/usr/local uninstall
```

https://github.com/mhahnFr/LeakSanitizer

메모리 사용량 측정 – LeakSanitizer

```
⊗ song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % cc main.c -g -L</usr/local/lib/liblsan.dylib> -llsan
  zsh: no such file or directory: /usr/local/lib/liblsan.dylib
⊗ song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % gcc -g -fsanitize=address -fsanitize=leak aes.c
  clang: error: unsupported option '-fsanitize=leak' for target 'arm64-apple-darwin22.3.0'
```

fsanitize=leak 명령어가 arm을 지원하지 않음

```
● song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % gcc -g -fsanitize=address aes.c
● song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % ./a.out
```

가능한 명령어만을 사용하여 메모리 측정

```
● song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % export ASAN_OPTIONS="verbosity=2:log_path=asan_log"
● song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % export LSAN_OPTIONS="verbosity=2:report_objects=1"
● song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % gcc -fsanitize=address -g aes.c
aes.c:9:3: warning: ignoring return value of function declared with 'warn_unused_result' attribute [-Wunused-result]
  malloc(7);
  ^~~~~~ ~
aes.c:9:3: warning: ignoring return value of function declared with 'warn_unused_result' attribute [-Wunused-result]
  malloc(5);
  ^~~~~~ ~
2 warnings generated.
● song-gyeongju@song-gyeongjuui-MacBookPro-2 AIMer-l1 % DYLD_INSERT_LIBRARIES=/Users/song-gyeongju/LeakSanitizer/liblsan.dylib ./a.out

Exiting

No leaks possible.

Report by LeakSanitizer v1.9

Copyright (C) 2022 – 2024  mhahnFr and contributors
Licensed under the terms of the GNU GPL version 3 or later.

For more information, visit github.com/mhahnFr/LeakSanitizer
```

옵션을 바꿔가며 측정

spank12monkeys • 7mo ago

-fsanitizer=leaks does not seem to be available on Apple's clang 15.0.0 that I have on my arm mac running Ventura. It does have a working sanitizer=address though. Maybe this is because Apple has a whole bunch of malloc debugging tools? Not sure.

To use the leaks tool, if you have a file like this

EXPLORER

AIMER-L1

- a.out.dSYM
- -llsan
- a.out
- aes.c
- asan_log.81602
- asan_log.81670
- asan_log.81777
- asan_log.81879
- asan_log.81942
- asan_log.81974
- asan_log.82081
- asan_log.82155
- supper.txt

aes.c | asan_log.81602 | supper.txt

asan_log.81602

```
==81602==AddressSanitizer: libc interceptors initialized
==81602==FindDynamicShadowStart, space_size = 0x100000003fff
|| `[0x107000020000, 0x7fffffffffff]` || HighMem    ||
|| `[0x027e00024000, 0x10700001ffff]` || HighShadow ||
|| `[0x007e00024000, 0x027e00023fff]` || ShadowGap  ||
|| `[0x007000020000, 0x007e00023fff]` || LowShadow  ||
|| `[0x000000000000, 0x00700001ffff]` || LowMem     ||
MemToShadow(shadow): 0x007e00024000 0x007fc00247ff 0x00bfc0024800 0x027e00023fff
redzone=16
max_redzone=2048
quarantine_size_mb=256M
thread_local_quarantine_size_kb=1024K
malloc_context_size=30
SHADOW_SCALE: 3
SHADOW_GRANULARITY: 8
SHADOW_OFFSET: 0x7000020000
==81602==Installed the sigaction for signal 11
==81602==Installed the sigaction for signal 10
==81602==Installed the sigaction for signal 8
==81602==SetCurrentThread: 0x000104284000 for thread 0x000200078140
==81602==T0: stack [0x00016b444000,0x00016bc40000) size 0x7fc000; local=0x00016bc3da48
AddressSanitizer: parsing ''
==81602==Checking file existence is not allowed under sandbox.
==81602==Checking file existence is not allowed under sandbox.
==81602==Checking file existence is not allowed under sandbox.
==81602==Checking file existence is not allowed under sandbox.
==81602==Using atos found at: /usr/bin/atos
==81602==Using dladdr symbolizer.
==81602==AddressSanitizer Init done
```

메모리 사용량 측정 – LeakSanitizer

EXPLORER

AIMER-L1
- a.out.dSYM
- -llsan
- a.out
- aes.c
- asan_log.81602
- asan_log.81670
- asan_log.81777
- asan_log.81879
- asan_log.81942
- asan_log.81974
- asan_log.82081
- asan_log.82155
- supper.txt

aes.c | asan_log.81602 | supper.txt

asan_log.81602

```
==81602==AddressSanitizer: libc interceptors initialized
==81602==FindDynamicShadowStart, space_size = 0x100000003fff
|| `[0x107000020000, 0x7fffffffffff]` || HighMem    ||
|| `[0x027e00024000, 0x10700001ffff]` || HighShadow ||
|| `[0x007e00024000, 0x027e00023fff]` || ShadowGap  ||
|| `[0x007000020000, 0x007e00023fff]` || LowShadow  ||
|| `[0x000000000000, 0x00700001ffff]` || LowMem     ||
MemToShadow(shadow): 0x007e00024000 0x007fc00247ff 0x00bfc0024800 0x027e00023fff
redzone=16
max_redzone=2048
quarantine_size_mb=256M
thread_local_quarantine_size_kb=1024K
malloc_context_size=30
SHADOW_SCALE: 3
SHADOW_GRANULARITY: 8
SHADOW_OFFSET: 0x7000020000
==81602==Installed the sigaction for signal 11
==81602==Installed the sigaction for signal 10
==81602==Installed the sigaction for signal 8
==81602==SetCurrentThread: 0x000104284000 for thread 0x000200078140
==81602==T0: stack [0x00016b444000,0x00016bc40000) size 0x7fc000; local=0x00016bc3da48
AddressSanitizer: parsing ''
==81602==Checking file existence is not allowed under sandbox.
==81602==Checking file existence is not allowed under sandbox.
==81602==Checking file existence is not allowed under sandbox.
==81602==Checking file existence is not allowed under sandbox.
==81602==Using atos found at: /usr/bin/atos
==81602==Using dladdr symbolizer.
==81602==AddressSanitizer Init done
```

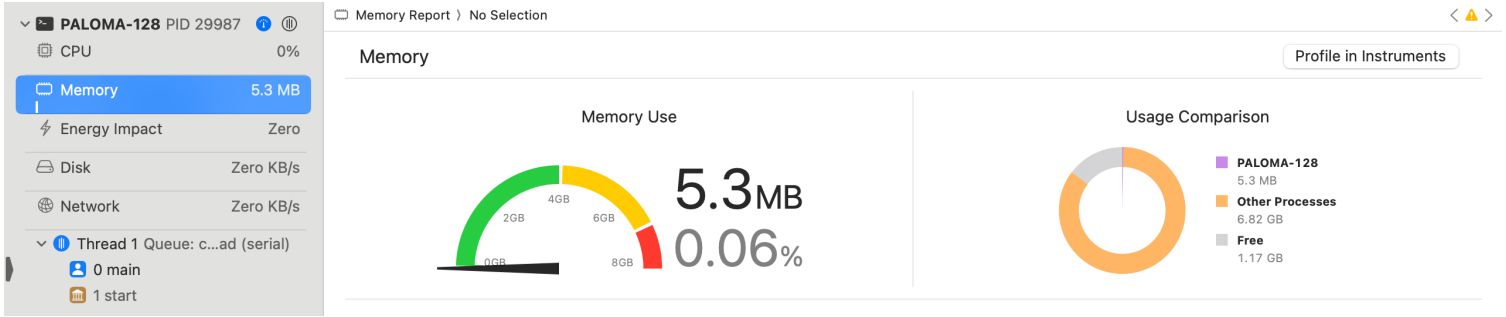
Valgrind와 동일 선상으로 확인할 수 있을 만한 결과값을 얻기 어려움

메모리 사용량 측정 – Xcode Memory Report

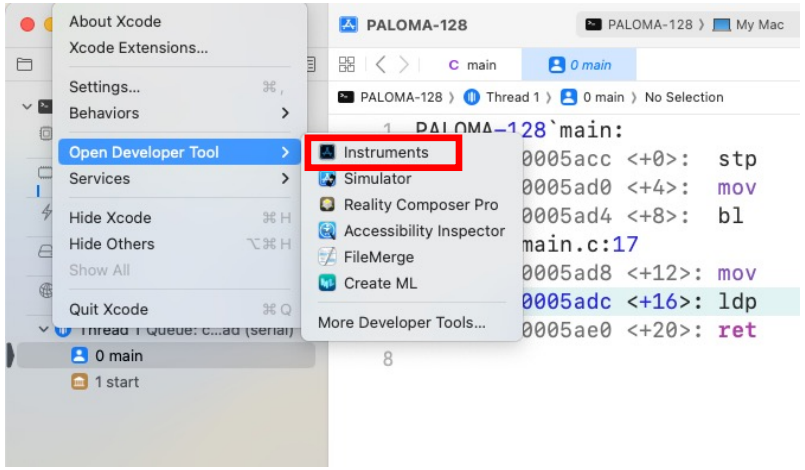
- 앱의 전체 메모리 사이즈에 대한 그래프를 보여줌
  - 따라서 대략적인 코드의 메모리 사용량을 확인하기 위해 코드 부분만 주석한 것과 원본코드의 메모리 사용량 차를 사용할 수 있음

```
/* Step 1: Generate Precomputation Table */
gf2m_tab gf2m_tables;
gen_precomputation_tab(&gf2m_tables);

for (int i = 0; i < TEST_LOOP; i++)
{
    crypto_kem_keypair(pk, sk, &gf2m_tables);
    crypto_kem_enc(ct, ss, pk);
    crypto_kem_dec(dss, ct, sk, &gf2m_tables);

}
```

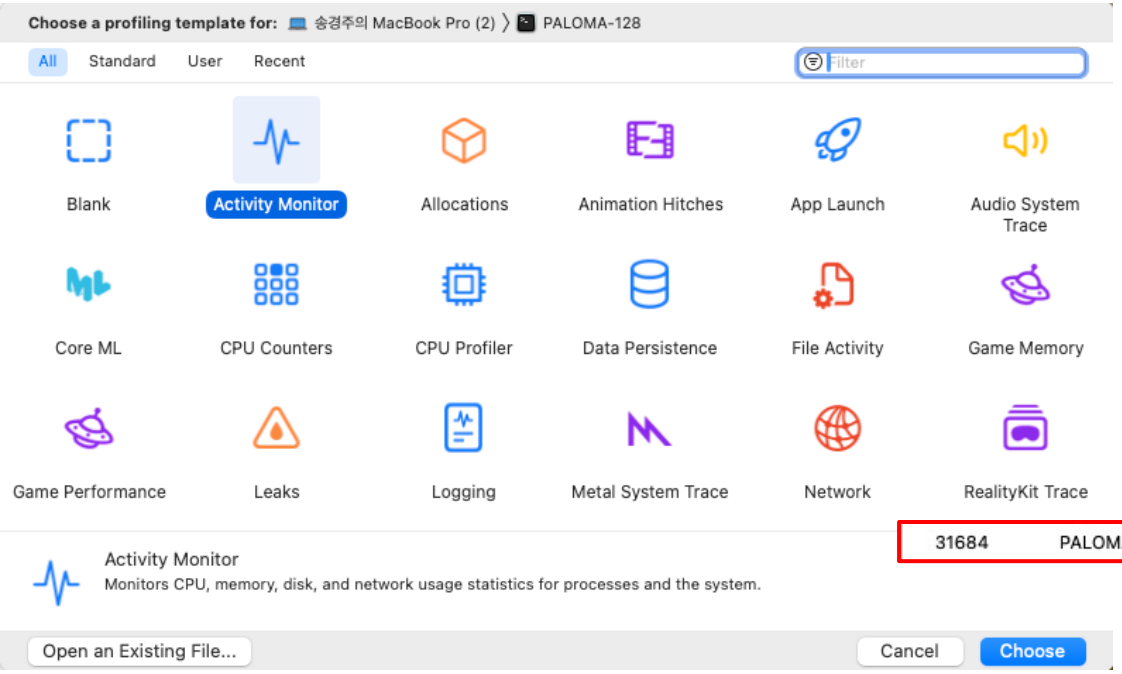
- 하지만 소수점 첫째자리의 MB에 대해서만 보여주므로 작은 메모리 사용량까지 추정하기 어려움
  - 즉, 코드의 메모리 사용량이 적은 경우 차가 발생하지 않아 추정 어려움

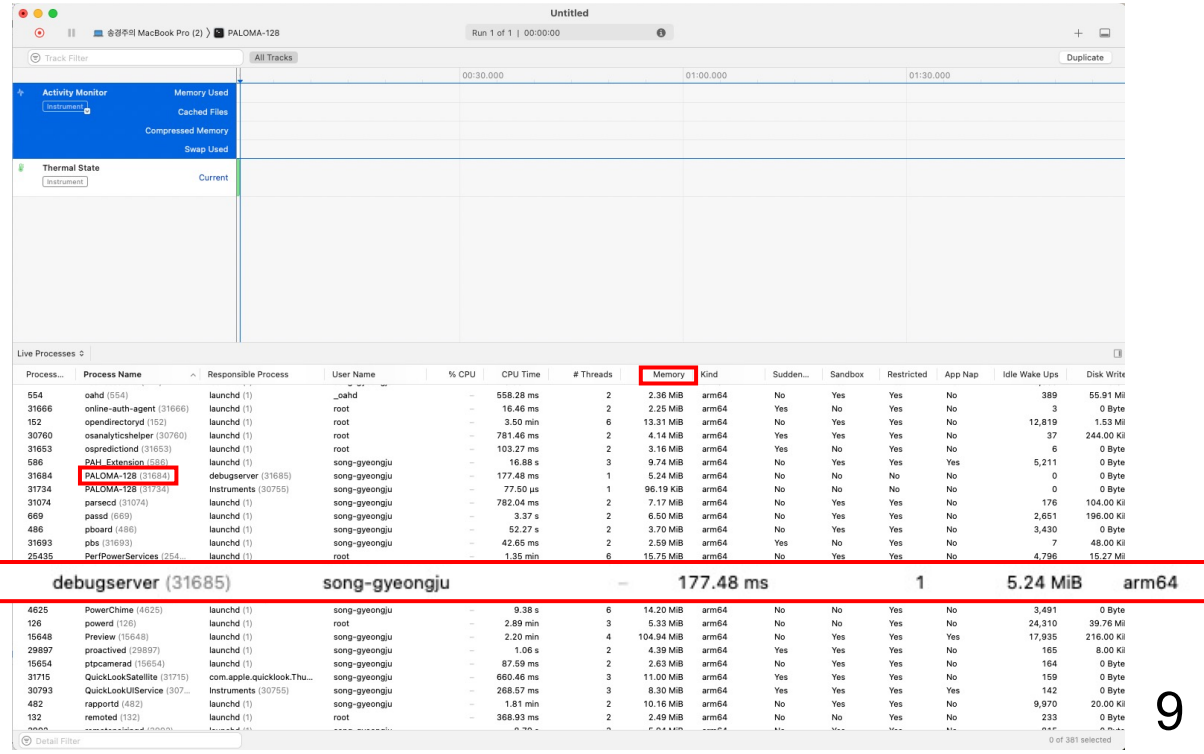


메모리 사용량 측정 – Xcode Memory Profiler



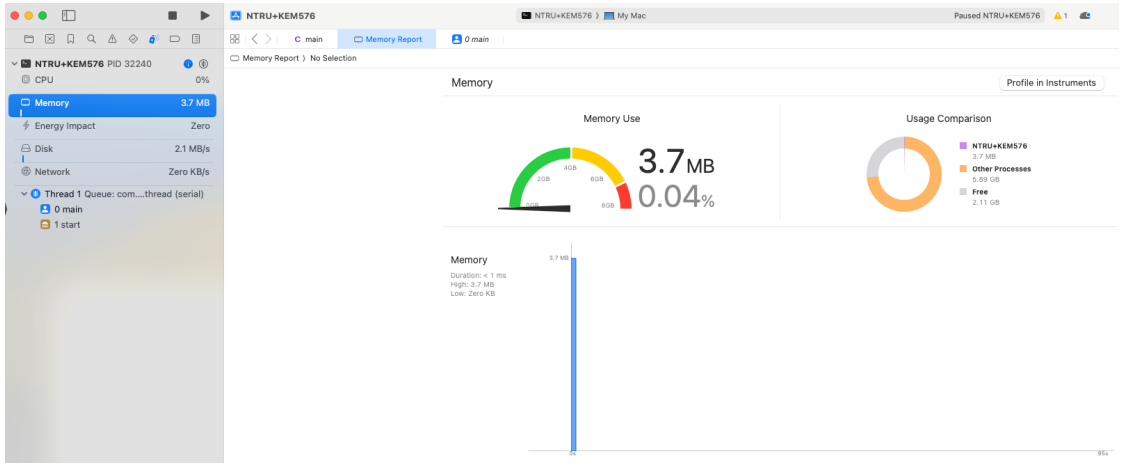

```
/* Step 1: Generate Precomputation Table */
gf2m_tab gf2m_tables;
gen_precomputation_tab(&gf2m_tables);

for (int i = 0; i < TEST_LOOP; i++)
{
    crypto_kem_keypair(pk, sk, &gf2m_tables);
    crypto_kem_enc(ct, ss, pk);
    crypto_kem_dec(dss, ct, sk, &gf2m_tables);

}
```

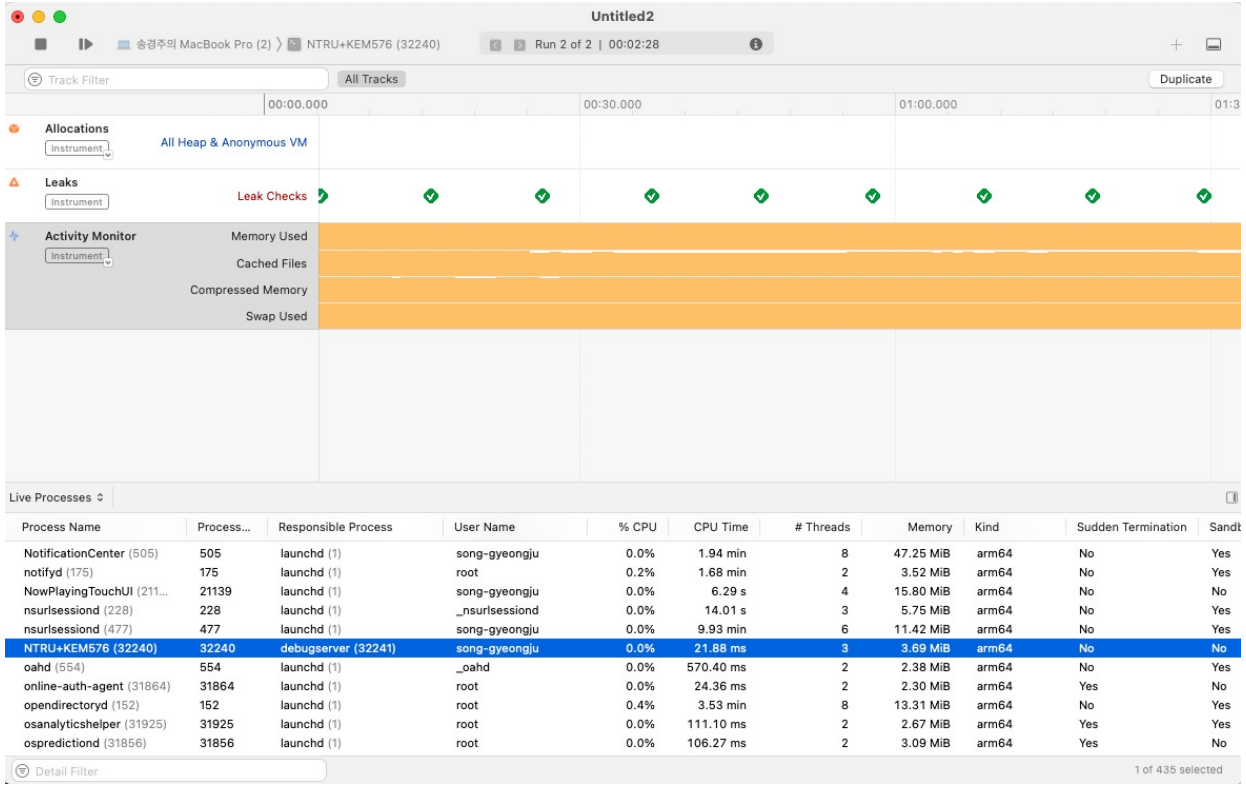




메모리 사용량 측정 – Xcode Memory Report + Profiler

- Profiler에서 소수점 둘째자리의 MB 단위까지 확인 가능
- 하지만 해당 방법도 Valgrind와 같이 heap, stack에 대한 정보를 확인하긴 어려움
- 메모리 전체 사용에 대해서만 대략적으로 확인가능
- 코드 분석용이 아닌, 컴퓨터의 현재 메모리 사용량 확인용으로 더 적합





Q & A