Debugging

Computer systems are really REALLY complicated because it consisting of many moving parts working together.

If something goes wrong, how do we know what and where to look for the cause?

Debugging is the process of finding and resolving bugs within computer programs, software or systems.

- Wikipedia

Debugging is similar being your local GP doctor.

You are trying to ‘guess’ the problem using your own methods, tools, intuition, knowledge and skills before providing ‘treatment’.

(or referring to a ‘specialist’)

Types of errors

Compile-time Errors!

Compile-time errors are errors that occur when you build a program with a compiler.

Compiler-time errors are great because they help catch invalid lines before the program can be run or shipped

Not all languages have compilers!!

Run-time errors

Run-time errors are errors that happen while the program is running, usually in a form of a crash.

Run-time errors can be forced by:

- Developer (in some form of exceptions)
- OS (e.g. segmentation fault)

Crashing is always bad, of course, but be aware that sometimes it's better to crash than to let the program continue.

Crashing is always bad, of course, but be aware that sometimes it's better to crash than to let the program continue.

(i.e. crash bad but not crash maybe worse)

Logical errors

Logical errors are when the program does something that you do not expect it to do.

Logical errors are the hardest to debug because it depends on what the error is.

Usually, we will need to rely on some tool to help us find the bug.

User error prevention and catching

Some kind of error code

The simplest kind of run-time error checking is to simply output something from a function that indicates the type of error, so that the person using it can handle it accordingly.

```c
typedef enum Errors {
    Errors_None,
    Errors_CannotOpenFile,
    Errors_Die,
} Errors;

//...

static Errors FuncThatReturnError(...) {
    if (!fopen(...)) {
        return Errors_CannotOpenFile;
    }
    if (!CheckIfAlive(...)) {
        return Errors_Die;
    }
    return Errors_None;
}

int main() {
    Errors err = FuncThatReturnError(...);
    switch(err) {
        //....
    }
}
```

```c
// Some APIs do something like this...
// I'm looking at you WinAPI and OpenGL and STD

// Hidden from users
typedef enum GLenum {
    GL_NO_ERROR,
    GL_INVALID_ENUM,
    GL_INVALID_VALUE,
    // etc...
} GLenum;

static GLenum error = GL_NO_ERROR;

```

```c
// Visible to users
static int FuncThatReturnTrueFalse(...) {
    if (!something) {
        error = GL_INVALID_VALUE;
        return 0;
    }
    if (!something_else(...)) {
        error = GL_INVALID_ENUM;
        return 0;
    }
    return 1;
}

static GLenum GetLastError() {
    return error;
}

int main() {
    if(!FuncThatReturnTrueFalse(...)) {
        GLenum err = GetLastError();
        switch(err) {
            //...
        }
    }

}int main() {
```

Asserts

Asserts will crash the program when a line is encountered.

Some assert implementations return the line number and file in which the assert occurred, which is useful!

```c
typedef struct ArrayInt
{
    size_t count;
    int* data;
} ArrayInt;

static ArrayInt CreateArrayInt(size_t amount) {
    ArrayInt ret = {0};
    ret.data = malloc(amount);
    ret.count = amount;
}

static void FreeArrayInt(ArrayInt* arr) {
    assert(arr);
    assert(arr->data);
    free(arr->data);
    arr->count = 0;
}

static int* GetArrayInt(ArrayInt* arr, size_t index) {
    assert(index < arr->count);
    return arr->data + index;
}

int main() {
    ArrayInt arr = CreateArrayInt(5);

    for (size_t i = 0; i <= arr.count; ++i) {
        printf("accessing arr[%ld]\n", i);
        GetArrayInt(&arr, i);
    }

    FreeArrayInt(&arr);
}
```

```
momo@DESKTOP-N6DP5P1:/mnt/d/work/sandbox/c$ gcc main.c -o main.exe && ./main.exe
accessing arr[0]
accessing arr[1]
accessing arr[2]
accessing arr[3]
accessing arr[4]
accessing arr[5]
main.exe: main.c:25: GetArrayInt: Assertion `index < arr->count' failed.
Aborted (core dumped)
momo@DESKTOP-N6DP5P1:/mnt/d/work/sandbox/c$
```

Typically used in development builds to catch statements that should ‘never’ happen.

Usually, it’s disabled when shipping a production build.

Note that there is some overhead when using asserts. It's like spamming 'if' conditions everywhere.

But sometimes,
it's better to be safe than slow.
Keyword: sometimes.

Unit testing

Writing code that test code.

But what tests the code that test the code?

But what tests the code that tests the code?

Can all features be tested with code?

If all tests pass, does that mean we are safe?

In the end, it depends.

Always weigh the pros and cons.

Always consider the feasibility.

Always remember that test code is still code to be written and maintained.

Archiving

Archiving is to write data out to a file, so that when an error occurs, we can trace what happened at when.

We typically archive:

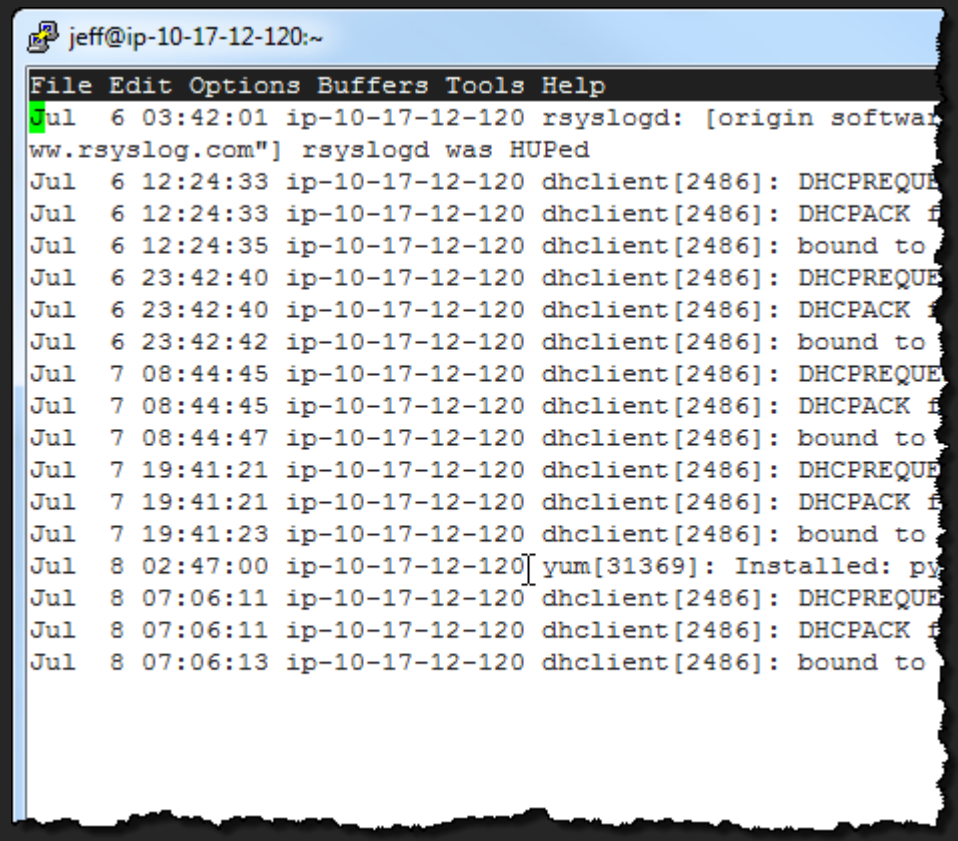
- Logs, text files written as program runs
- Dumps, snapshots of state (e.g. memory)

Logging

Logging is simply writing text to a file
‘just in case’

```c
int main() {

    if (...) {
        printf("There is a problem here!");
    }

    if (...) {
        printf("And here!!");
    }

    if (...) {
        printf("And also here!");
    }
}

```

Logging into human readable text is the simplest yet most flexible form of debugging.



You can log pretty much anything and everything! It is not just for debugging.

Just remember to adhere to a format so that it's easier to filter the texts with the tools you use.

```
2015-10-17 15:45:11,258 INFO [main] org.apache.hadoop.metrics2.impl.MetricsConfig: loaded properties from hadoop-metrics2.properties
2015-10-17 15:45:11,399 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: Scheduled snapshot period at 10 second(s).
2015-10-17 15:45:11,399 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: MapTask metrics system started
2015-10-17 15:45:11,430 INFO [main] org.apache.hadoop.mapred.YarnChild: Executing with tokens:
2015-10-17 15:45:11,430 INFO [main] org.apache.hadoop.mapred.YarnChild: Kind: mapreduce.job, Service: job_1445062781478_0015, Ident: (org.apache$
2015-10-17 15:45:11,602 INFO [main] org.apache.hadoop.mapred.YarnChild: Sleeping for 0ms before retrying again. Got null now.
2015-10-17 15:45:12,196 INFO [main] org.apache.hadoop.mapred.YarnChild: mapreduce.cluster.local.dir for child: /tmp/hadoop-msrabi/nm-local-dir/u$
2015-10-17 15:45:12,711 INFO [main] org.apache.hadoop.conf.Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session$
2015-10-17 15:45:13,602 INFO [main] org.apache.hadoop.yarn.util.ProcfsBasedProcessTree: ProcfsBasedProcessTree currently is supported only on Li$
2015-10-17 15:45:13,618 INFO [main] org.apache.hadoop.mapred.Task:  Using ResourceCalculatorProcessTree : org.apache.hadoop.yarn.util.WindowsBas$
2015-10-17 15:45:14,008 INFO [main] org.apache.hadoop.mapred.MapTask: Processing split: hdfs://msra-sa-41:9000/pageinput2.txt:402653184+134217728
2015-10-17 15:45:14,102 INFO [main] org.apache.hadoop.mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
2015-10-17 15:45:14,102 INFO [main] org.apache.hadoop.mapred.MapTask: mapreduce.task.io.sort.mb: 100
2015-10-17 15:45:14,102 INFO [main] org.apache.hadoop.mapred.MapTask: soft limit at 83886080
2015-10-17 15:45:14,102 INFO [main] org.apache.hadoop.mapred.MapTask: bufstart = 0; bufvoid = 104857600
2015-10-17 15:45:14,102 INFO [main] org.apache.hadoop.mapred.MapTask: kvstart = 26214396; length = 6553600
2015-10-17 15:45:14,118 INFO [main] org.apache.hadoop.mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuf$
2015-10-17 15:45:17,305 INFO [main] org.apache.hadoop.mapred.MapTask: Spilling map output
2015-10-17 15:45:17,305 INFO [main] org.apache.hadoop.mapred.MapTask: bufstart = 0; bufend = 48271024; bufvoid = 104857600
2015-10-17 15:45:17,305 INFO [main] org.apache.hadoop.mapred.MapTask: kvstart = 26214396(104857584); kvend = 17310640(69242560); length = 890375$
2015-10-17 15:45:17,305 INFO [main] org.apache.hadoop.mapred.MapTask: (EQUATOR) 57339776 kvi 14334940(57339760)
2015-10-17 15:45:26,696 INFO [SpillThread] org.apache.hadoop.mapred.MapTask: Finished spill 0
2015-10-17 15:45:26,696 INFO [main] org.apache.hadoop.mapred.MapTask: (RESET) equator 57339776 kv 14334940(57339760) kvi 12140764(48563056)
2015-10-17 15:45:30,603 INFO [main] org.apache.hadoop.mapred.MapTask: Spilling map output
2015-10-17 15:45:30,603 INFO [main] org.apache.hadoop.mapred.MapTask: bufstart = 57339776; bufend = 743078; bufvoid = 104857600
2015-10-17 15:45:30,603 INFO [main] org.apache.hadoop.mapred.MapTask: kvstart = 14334940(57339760); kvend = 5428644(21714576); length = 8906297/$
2015-10-17 15:45:30,603 INFO [main] org.apache.hadoop.mapred.MapTask: (EQUATOR) 9811814 kvi 2452948(9811792)
2015-10-17 15:45:39,525 INFO [SpillThread] org.apache.hadoop.mapred.MapTask: Finished spill 1
2015-10-17 15:45:39,525 INFO [main] org.apache.hadoop.mapred.MapTask: (RESET) equator 9811814 kv 2452948(9811792) kvi 244148(976592)
2015-10-17 15:45:43,307 INFO [main] org.apache.hadoop.mapred.MapTask: Spilling map output
2015-10-17 15:45:43,307 INFO [main] org.apache.hadoop.mapred.MapTask: bufstart = 9811814; bufend = 58036090; bufvoid = 104857600
2015-10-17 15:45:43,307 INFO [main] org.apache.hadoop.mapred.MapTask: kvstart = 2452948(9811792); kvend = 19751904(79007616); length = 8915445/6$
2015-10-17 15:45:43,307 INFO [main] org.apache.hadoop.mapred.MapTask: (EQUATOR) 67104842 kvi 16776204(67104816)
```

Dump

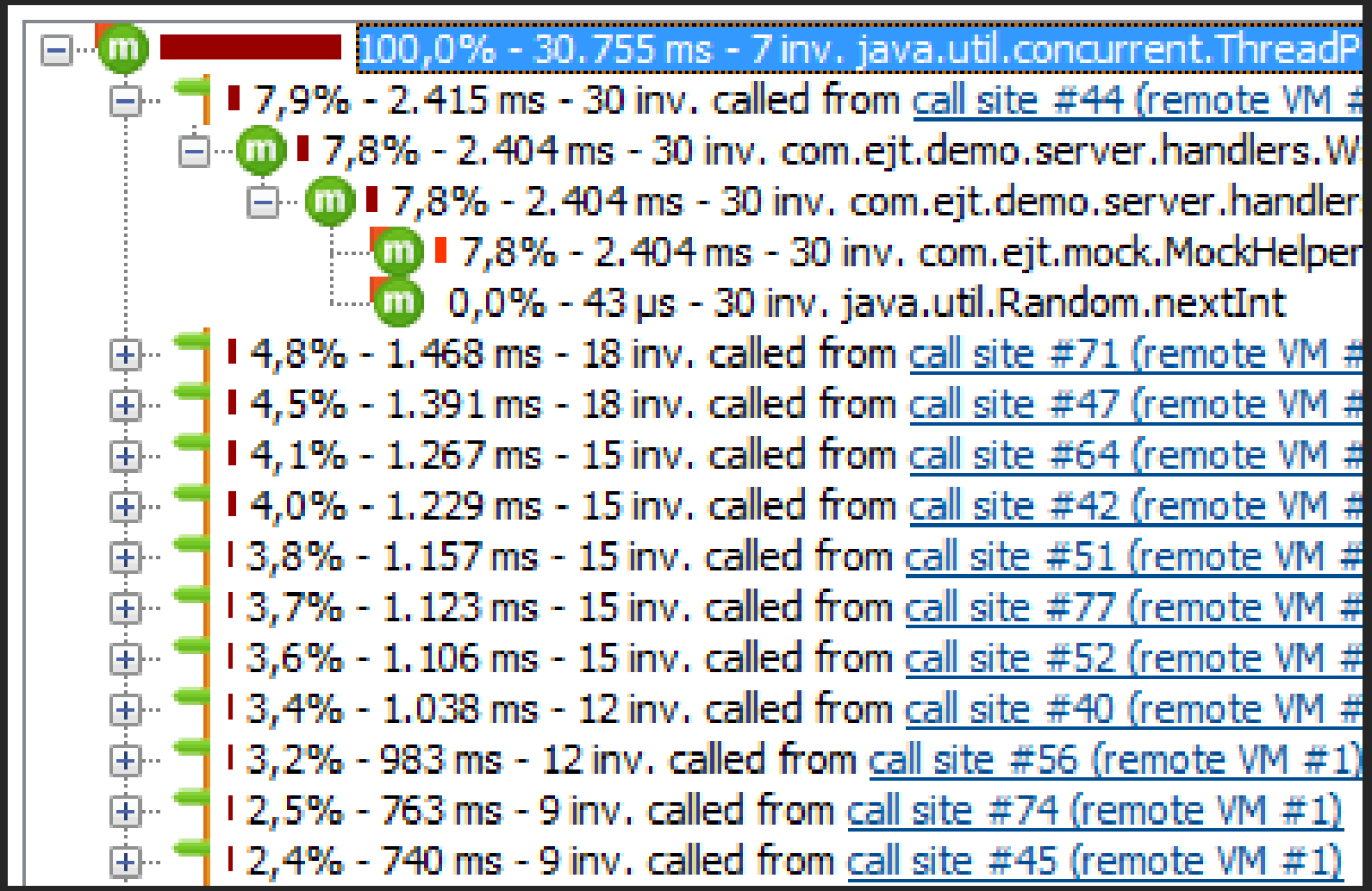
Dumping is the same as logging, but it is usually a once-in-a-while occurrence because the data to write is huge

Things to dump

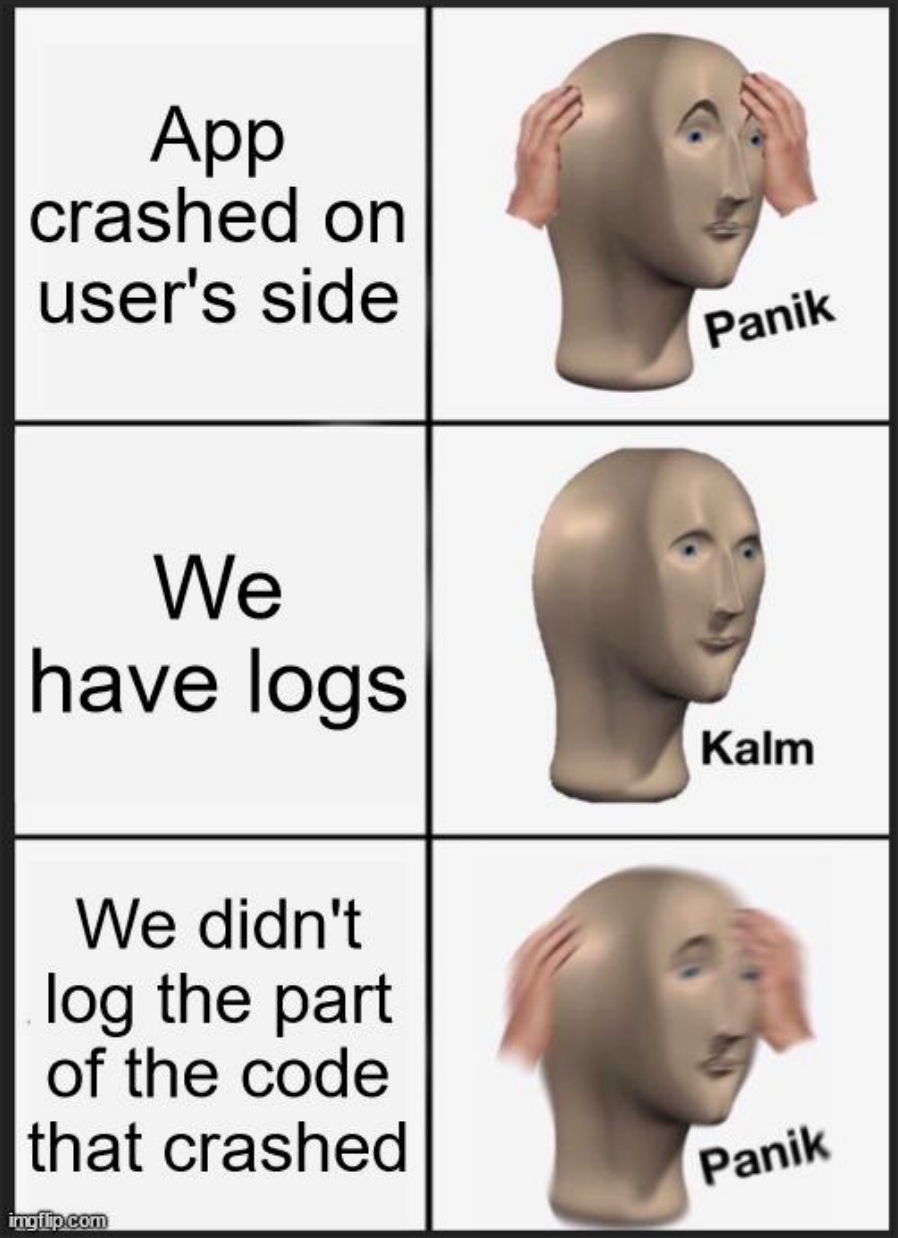
- Memory
- Databases
- Stack Traces

C:\Windows\System32\cmd.exe - testrom -b test.bin

```
> dump $2000 $2200
03-2000: 4083 130D 1311 1314 021F 2023 4C49 5354 @......... #LIST
03-2008: 204B 4559 A807 6C0F 0000 0828 47D1 443A  KEY..l....<G.D:
03-2010: 44E0 0360 F187 607C 2034 321D 07EC 0A1D D..`..`| 42.....
03-2018: 47D7 F082 44F6 6C05 6D2D 4526 E8C6 F081 G...D.l.m-E&....
03-2020: 4523 F081 F081 2060 4C49 5354 A805 6C2C E#.... `LIST..l,
03-2028: 0000 0A34 C12C 082A 47B3 441C 7171 834B ...4.,.*G.D.qq.K
03-2030: 2025 B800 7171 834B 2026 321D 0001 B21D  %..qq.K &2.....
03-2038: 834B 2025 8000 F020 2001 F403 47B8 6FBC .K %...  ...G.o.
03-2040: 0360 F187 607C 2038 321D 07BB 0A1D 47A8 .`..`| 82.....G.
03-2048: F082 7170 3360 7171 834B 2023 B800 7171 ..qp3`qq.K #..qq
03-2050: 2021 B800 F081 44BD 6C09 6D03 C103 E8C6  !....D.l.m.....
03-2058: 4504 6D01 C103 E8C6 F081 C103 6CFA 6FF9 E.m.........l.o.
03-2060: 2077 5255 4E20 A805 6C0F 0000 4406 F081  wRUN ..l...D...
03-2068: 0796 086C 4785 F082 0824 4772 7171 834B ...lG....$Grqq.K
03-2070: 2023 B800 F082 C0CE C103 6CE3 F081 208A  #........l... .
03-2078: 434F 4E54 A805 6C08 0000 47EF F081 077E CONT..l...G....~
03-2080: 086C 476E F082 C0CE C103 6C02 F081 0887 .lGn......l.....
03-2088: C0D3 F081 2097 5343 5241 5443 4820 4120 .... .SCRATCH A
03-2090: A808 E8CE 0000 C168 0000 0087 6C3E 20AA .......h....l> .
03-2098: 5343 5241 5443 4820 4B45 5920 A809 6C07 SCRATCH KEY ..l.
03-20A0: 0000 444D 0761 0834 474B F082 C0CE 449C ..DM.a.4GK....D.
03-20A8: F081 F081 20B5 5343 5241 5443 4820 5020 .... .SCRATCH P
03-20B0: A808 E8CE 0000 0022 6C20 20C0 5343 5241 ......."l  .SCRA

Press STOP to quit, B to step back or any other key to proceed_
```


100,0% - 30.755 ms - 7 inv. java.util.concurrent.ThreadP
7,9% - 2.415 ms - 30 inv. called from call site #44 (remote VM #
7,8% - 2.404 ms - 30 inv. com.ejt.demo.server.handlers.W
7,8% - 2.404 ms - 30 inv. com.ejt.demo.server.handler
7,8% - 2.404 ms - 30 inv. com.ejt.mock.MockHelper
0,0% - 43 µs - 30 inv. java.util.Random.nextInt
4,8% - 1.468 ms - 18 inv. called from call site #71 (remote VM #
4,5% - 1.391 ms - 18 inv. called from call site #47 (remote VM #
4,1% - 1.267 ms - 15 inv. called from call site #64 (remote VM #
4,0% - 1.229 ms - 15 inv. called from call site #42 (remote VM #
3,8% - 1.157 ms - 15 inv. called from call site #51 (remote VM #
3,7% - 1.123 ms - 15 inv. called from call site #77 (remote VM #
3,6% - 1.106 ms - 15 inv. called from call site #52 (remote VM #
3,4% - 1.038 ms - 12 inv. called from call site #40 (remote VM #
3,2% - 983 ms - 12 inv. called from call site #56 (remote VM #1)
2,5% - 763 ms - 9 inv. called from call site #74 (remote VM #1)
2,4% - 740 ms - 9 inv. called from call site #45 (remote VM #1)

As you can imagine, having archives is a BIG part of operations and maintenance teams to find and solve bugs caused by users that your team did not catch.


App crashed on user's side
Panik
We have logs
Kalm
We didn't log the part of the code that crashed
Panik
imgflip.com

Remember

Keeping an archive is not free.

Files accumulate and you need to manage them somehow.

Problems that you might eventually deal with

Logging to gather data

Log rotation for apps that run 24/7

Log archiving

Log merging (parallel programs)

Log filtering for different teams

Crash logs from users


We have too many logs. What should we do?
Zip them
Buy hard disk
delete them
imgflip.com

‘Debuggers’

Debuggers are programs that attaches itself to a running program and provide various information to you as the program runs.

Notable debuggers for C/C++

Visual Studios

GDB

Valgrind

WinDBG

RemebyBG

and many many more…

Notable debugger features

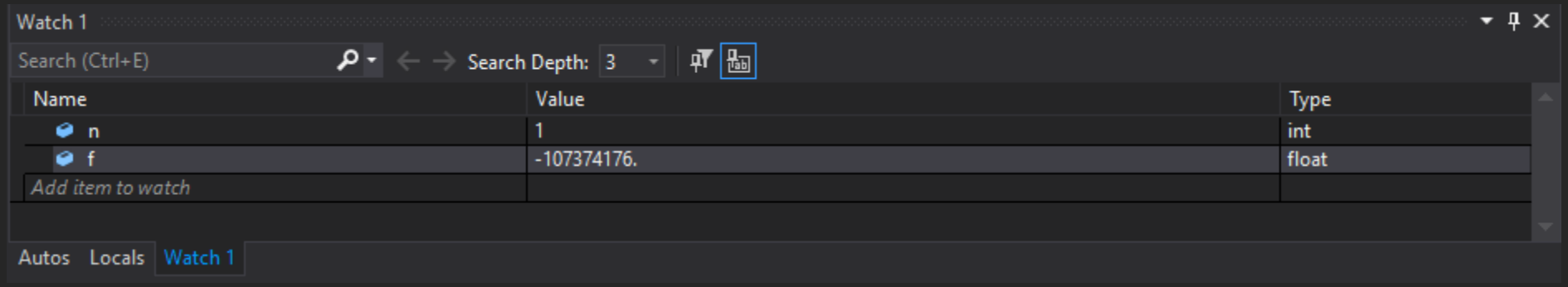
Most debuggers come with breakpoints.

They allow you to ‘pause’ your program while it’s running when a line is ‘hit’.

```
395         // ignore constructors
396         if(!string_match(containing_struct->string, name_str)) {
397
```

When a breakpoint is hit, there are a bunch of things you can inspect/edit including variables, memory, disassembly

Watches


Watch 1
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
n
1
int
f
-107374176.
float
Add item to watch
Autos
Locals
Watch 1

Call Stack

Call Stack

Name
Project1.exe!fib(int n) Line 6
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!fib(int n) Line 9
Project1.exe!main() Line 13
[External Code]
kernel32.dll![Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]

Memory

Memory 2

Address: 0x000000EED8FAFBB2

```
0x000000EED8FAFBB2  c3 ce f7 7f 00 00 31 00  ÃÎ÷...1.
0x000000EED8FAFBBA  5c 00 78 00 36 00 34 00  \.x.6.4.
0x000000EED8FAFBC2  5c 00 44 00 65 00 62 00  \.D.e.b.
0x000000EED8FAFBCA  75 00 67 00 5c 00 e6 cd  u.g.\.æÍ
0x000000EED8FAFBD2  8e 30 e3 8a 59 f7 65 00  Ž0ãŠY÷e.
0x000000EED8FAFBDA  63 00 74 00 31 00 2e 00  c.t.1...
0x000000EED8FAFBE2  65 00 78 00 65 00 96 55  e.x.e.–U
0x000000EED8FAFBEA  90 17 fb 7f 00 00 ff 00  ..û...ÿ.
0x000000EED8FAFBF2  00 40 31 65 59 f7 19 00  .@1eY÷..
0x000000EED8FAFBFA  00 00 fb 7f 00 00 ff 00  ..û...ÿ.
0x000000EED8FAFC02  00 40 31 65 59 f7 19 00  .@1eY÷..
0x000000EED8FAFC0A  00 00 fb 7f 00 00 68 48  ..û...hH
0x000000EED8FAFC12  a1 17 fb 7f 00 00 50 87  ¡.û...P.
0x000000EED8FAFC1A  90 17 fb 7f 00 00 e6 cd  ..û...æÍ
0x000000EED8FAFC22  8e 30 e3 8a 59 f7 50 87  Ž0ãŠY÷P.
0x000000EED8FAFC2A  90 17 fb 7f 00 00 00 10  ..û.....
0x000000EED8FAFC32  c3 ce f7 7f 00 00 00 00  ÃÎ÷.....
0x000000EED8FAFC3A  00 00 00 00 00 00 00 00  ........
0x000000EED8FAFC42  00 00 00 00 00 00 3b 18  ......;.
0x000000EED8FAFC4A  c2 ce f7 7f 00 00 00 10  ÂÎ÷.....
0x000000EED8FAFC52  c3 ce f7 7f 00 00 a0 98  ÃÎ÷... ˜
0x000000EED8FAFC5A  b2 ac fb 7f 00 00 4c 9c  .¬û...Lœ
0x000000EED8FAFC62  95 74 ca 02 00 00 a0 98  .tÊ... ˜
0x000000EED8FAFC6A  b2 ac fb 7f 00 00 02 00  .¬û.....
0x000000EED8FAFC72  00 00 ee 00 00 00 bb 98  ..î...»˜
```

Disassembly

```
91      current_easing_function_index = 0;
92  }
93  void GameUpdate(void)
94  {
95      timer += CP_System_GetDt();
96      if (timer >= duration) {
97          timer = 0.f;
98          SWAP(float, min_x, max_x);
99          SWAP(float, min_size, max_size);
100     }
101
102
103     if (CP_Input_KeyTriggered(KEY_RIGHT))
104     {
105         ++current_easing_function_index;
106         if (current_easing_function_index >= ArrayCount(ea
```

Viewing Options

```
00007FF7C4DE3E17  call        __CheckForDebuggerJustMyCode (07FF7C4DE139Dh)
    timer += CP_System_GetDt();
00007FF7C4DE3E1C  call        qword ptr [__imp_CP_System_GetDt (07FF7C4DF4000h)]
00007FF7C4DE3E22  movss       xmm1,dword ptr [timer (07FF7C4DF0B44h)]
00007FF7C4DE3E2A  addss       xmm1,xmm0
00007FF7C4DE3E2E  movaps      xmm0,xmm1
00007FF7C4DE3E31  movss       dword ptr [timer (07FF7C4DF0B44h)],xmm0
    if (timer >= duration) {
00007FF7C4DE3E39  movss       xmm0,dword ptr [timer (07FF7C4DF0B44h)]
00007FF7C4DE3E41  comiss      xmm0,dword ptr [duration (07FF7C4DF0B40h)]
00007FF7C4DE3E48  jb          __$EncStackInitStart+0AAh (07FF7C4DE3EA9h)
        timer = 0.f;
00007FF7C4DE3E4A  xorps       xmm0,xmm0
00007FF7C4DE3E4D  movss       dword ptr [timer (07FF7C4DF0B44h)],xmm0
        SWAP(float, min_x, max_x);
```

or you can also use https://godbolt.org/

Visual Studios Debugger Demo

Internal debugger

Instead of using an external debugger at hooks onto your program, you can write your own a debugger WITHIN your program.

The good thing is that the variables and functions you want to debug are immediately available to you and that it will be targeted to best fit your program.


Minecraft 1.17.1 (fabric-loader-0.11.6-1.17.1/fabric/Fabric)
56 fps T: inf vsyncfancy fancy-clouds B: 2
"vanilla" server, 2 tx, 1432 rx
C: 231/4624 (s) D: 8, pC: 000, pU: 00, aB: 08
E: 9/389, B: 0
P: 220, T: 389
Chunks[C] W: 625, 420 E: 389,139,420
minecraft:overworld FC: 0
295, 65, -17
XYZ: 295.512 / 65.00000 / -16.827
Block: 295 65 -17
Chunk: 7 1 15 in 18 4 -2
Facing: east (Towards positive X) (-99.3 / 4.8)
Client Light: 15 (15 sky, 10 block)
CH S: 64 M: 64
SH S: 64 O: -1 M: 64 ML: -1
Biome: minecraft:wooded_hills
Local Difficulty: 2.25 // 0.13 (Day 1774)
Sounds: 17/247 + 1/8 (Mood 0%)
[Fabric] Active renderer: IndigoRenderer
Debug: Pie [shift]: hidden FPS [alt]: hidden
For help: press F3 + Q
Java: 16.0.1 64bit
Mem: 38% 782/2048MB
Allocated: 78% 1600MB
CPU: 8x Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Display: 1920x1057 (NVIDIA Corporation)
GeForce GTX 1070/PCIe/SSE2
3.2.0 NVIDIA 456.71
Zombie dies
Footsteps >
Villager mumbles >
45
8
33
24
55
11
2


Memory
Arenas
Frames
Sizes
Debug
Call Sites
Game Mode|w:\handmade\code\handmade_world.cpp|221|120: 65624b


50%
x0
help commands
bind: Creates, deletes or lists key bindings associated to commands or shortcuts.
clear: Clears the in-game console.
help: Gets help on using commands.
Listed 3/3 commands.
bind
Listed 0/0 key bindings.
> _
CLUB
RAINBOW SIX

At this point, how you want to debug is up to your creativity!

At the end of the day, debugging is part of the engineering process. For every route you can take, you must think critically!

Also, don’t forget talking

Sometimes, it might be helpful to speak what you are thinking to another person.

(sounds ridiculous, but it might work for some)