Advanced Linux Game Programming

Leszek Godlewski
Programmer, Nordic Games

GAME DEVELOPERS CONFERENCE™ EUROPE
CONGRESS-CENTRUM OST KOELNMESSE · COLOGNE, GERMANY
AUGUST 11–13, 2014 · EXPO: AUGUST 11–12, 2014

Nordic Games GmbH

nordic games

- Started in 2011 as a sister company to Nordic Games Publishing (We Sing)
- Base IP acquired from JoWooD and DreamCatcher (SpellForce, The Guild, Aquanox, Painkiller)
- Initially focusing on smaller, niche games
- Acquired THQ IPs in 2013 (Darksiders, Titan Quest, Red Faction, MX vs. ATV)
- Now shifting towards being a production company with internal devs
- Since fall 2013: internal studio in Munich, Germany (Grimlore Games)

Leszek Godlewski
Programmer, Nordic Games

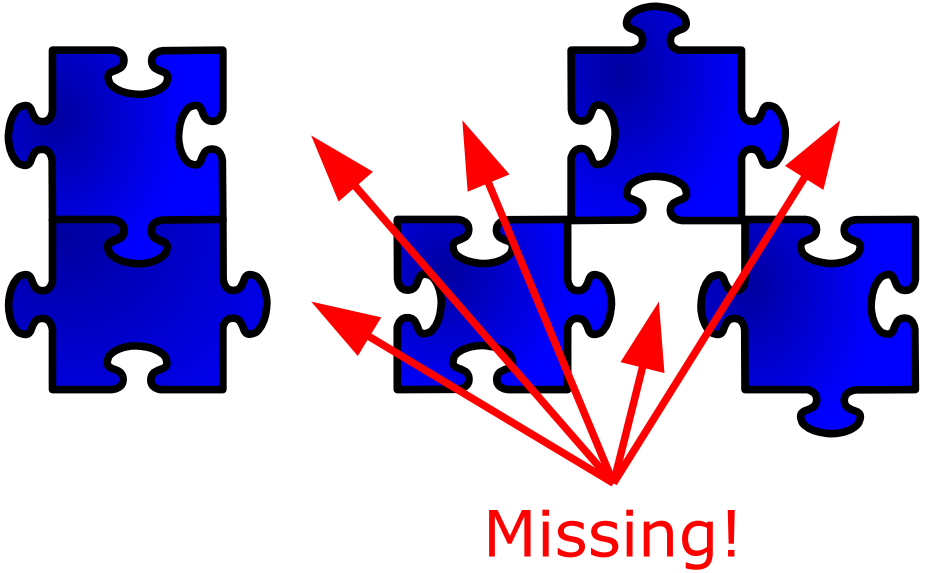
nordic games

- Ports
  - Painkiller Hell & Damnation (The Farm 51)
  - Deadfall Adventures (The Farm 51)
  - Darksiders (Nordic Games)
- Formerly generalist programmer on PKHD & DA at TF51

DARKSIDERS™

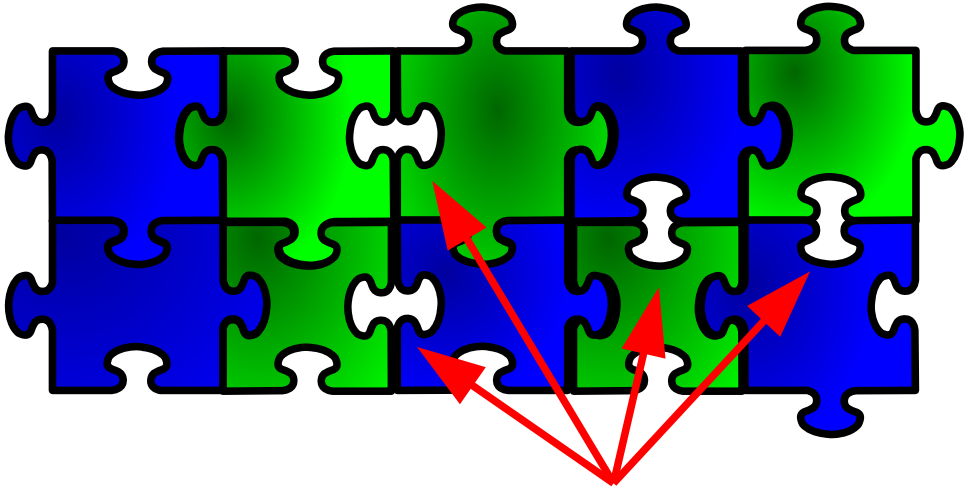
Objective of this talk

Your game engine on Linux, before porting:



Objective of this talk (cont.)

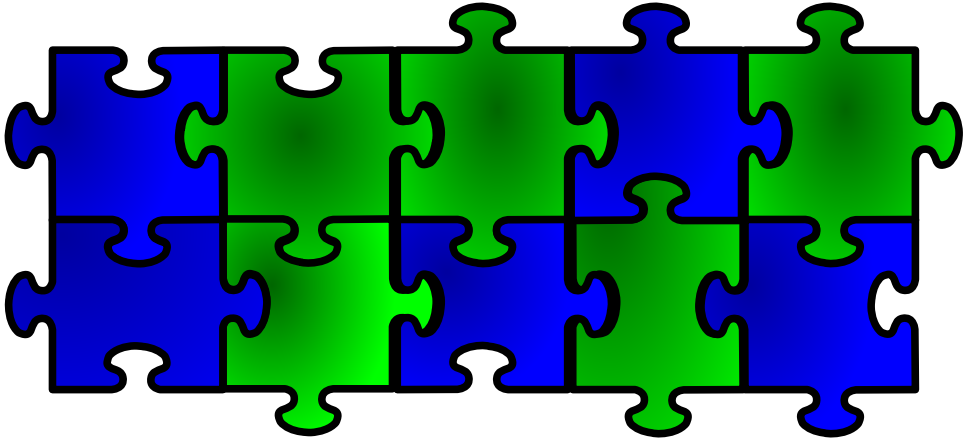
Your first “working” Linux port:



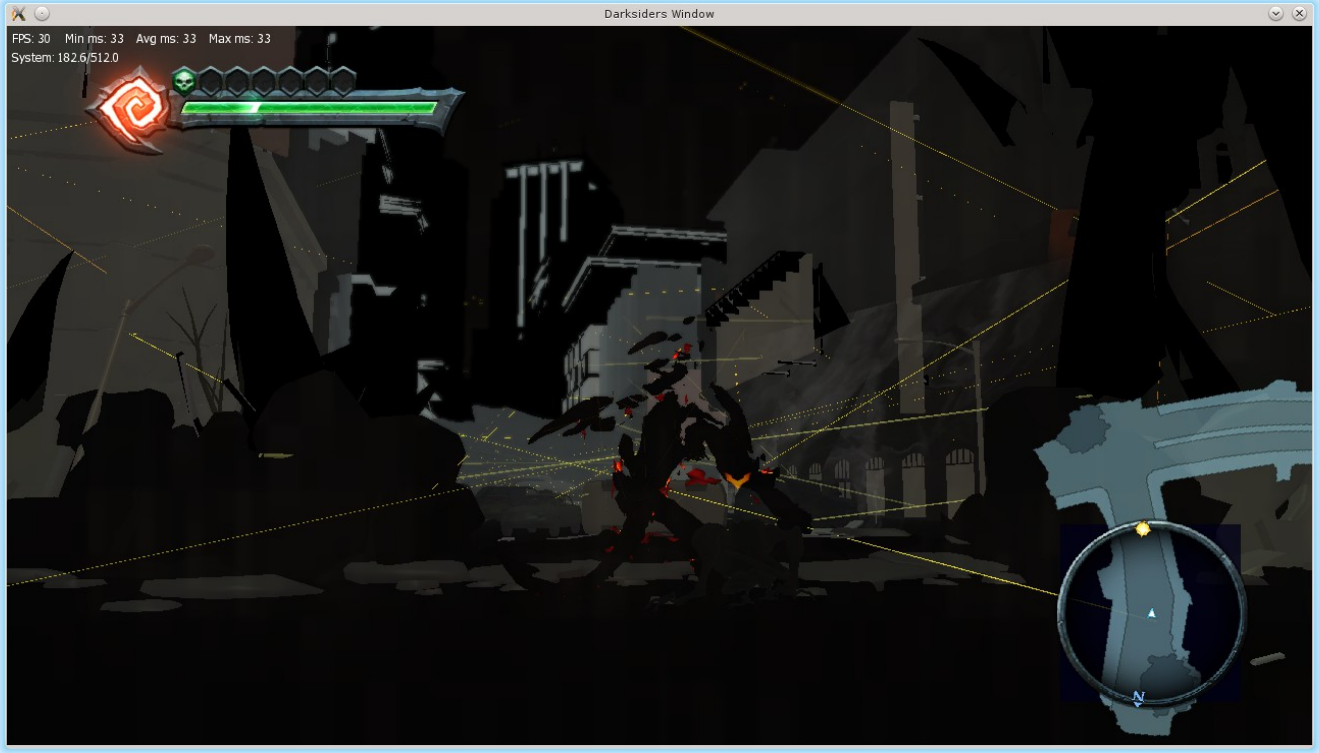
Oops. Bat-Signal!

Objective of this talk (cont.)

Where I want to try helping you get to:



In other words, from this:

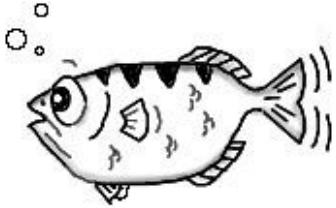


To this:


Darksiders Window
FPS: 12 Min ms: 81 Avg ms: 81 Max ms: 289029907
System: 186.3/512.0

And that's mostly debugging

All sorts of debuggers!



apitrace

ValveSoftware / **vogl**

NVIDIA® Nsight™

Demo code available

is.gd/GDCE14Linux

Intended takeaway

- Build system improvements
- Signal handlers
- Memory debugging with Valgrind
- OpenGL debugging techniques

~~Intended takeaway~~ Agenda

- **Build system improvements**
- Signal handlers
- Memory debugging with Valgrind
- OpenGL debugging techniques

Build systems

What I had initially with UE3:

- Copy/paste of the Mac OS X toolchain
- It worked, but...
  - **Slow**
  - **Huge** binaries because of debug symbols
  - Problematic linking of circular dependencies

Build systems (cont.)

- 32-bit binaries required for feature/hardware **parity** with Windows
- Original solution: a `chroot` jail with an entire 32-bit Ubuntu system **just for building**

Cross-compiling for 32/64-bit

- `gcc -m32/-m64` is **not enough**!
  - Only sets target code generation
  - **Not** headers & libraries (CRT, OpenMP, libgcc etc.)
- Fixed by installing `gcc-multilib`
  - Dependency package for non-default architectures (i.e. i386 on an amd64 system and vice versa)

Clang (ad nauseam)

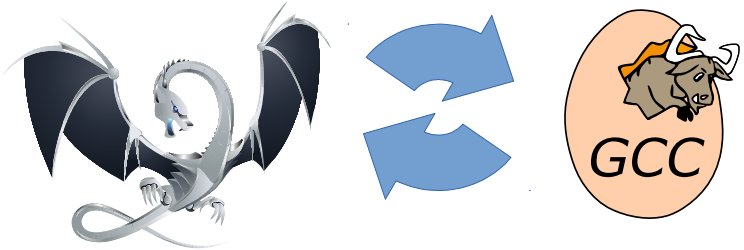
- Clang is **faster**
  - gcc: 3m47s
  - Clang: 3m05s
  - More benchmarks at Phoronix [LARABEL13]
- Clang has different diagnostics than gcc

```
test.cpp: In function 'int main(int, char**)':
test.cpp:8:2: error: 'integer' was not declared
  integer i = 0;
  ^
test.cpp:8:10: error: expected ';' before 'i'
  integer i = 0;
          ^
```

```
test.cpp:8:2: error: unknown type name 'integer'; did you mean 'Integer'?
        integer i = 0;
        ^~~~~~~
        Integer
test.cpp:4:13: note: 'Integer' declared here
typedef int Integer;
            ^
1 error generated.
```

Clang (cont.)

- Preprocessor macro compatibility
  - Declares __GNUC__ etc.
- Command line compatibility
  - Easily switch back & forth between Clang & gcc



Clang – caveats

- C++ object files may be incompatible with gcc & fail to link (need full rebuilds)
- Clang is not as mature as gcc
  - Occasionally has generated faulty code for me (YMMV)

Clang – caveats (cont.)

- Slight inconsistencies in C++ standard strictness
  - Templates
  - Anonymous structs/unions
  - May need to add `this->` in some places
  - May need to name some anonymous types

So: Clang or gcc?

Both:

- Clang – quick iterations during development
- gcc – final shipping binaries

Linking – GNU ld

- Default linker on Linux
- Ancient
- Single-threaded
- Requires specification of libraries in the order of reverse dependency...
- We are not doomed to use it!

Linking – GNU gold

- Multi-threaded linker for ELF binaries
  - ld: 18s
  - gold: 5s
- Developed at Google, now officially part of GNU binutils

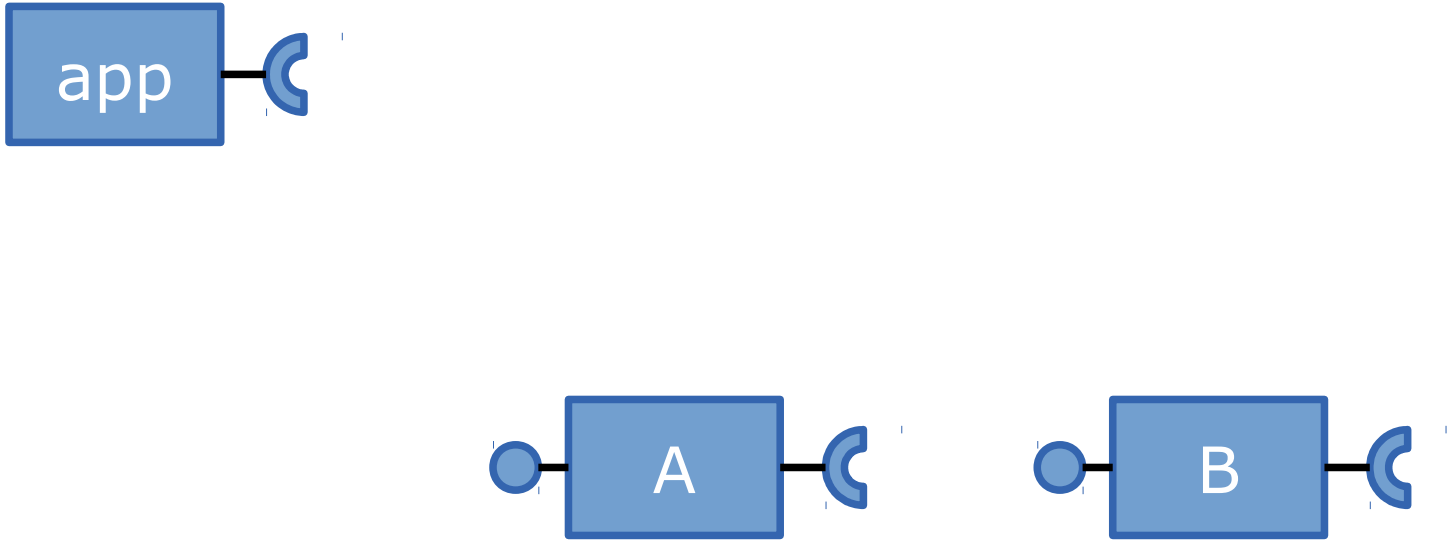
Linking – GNU gold (cont.)

- Drop-in replacement for ld
  - May need an additional parameter or toolchain setup
    - `clang++ -B/usr/lib/gold-ld ...`
    - `g++ -fuse-ld=gold ...`
- Still needs libs in the order of reverse dependency...

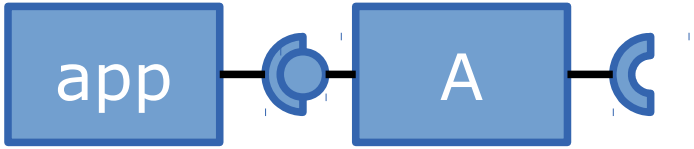
Linking – reverse dependency

- Major headache/game-breaker with circular dependencies
- ”Proper” fix: re-specify the same libraries over and over again
  - `gcc app.o -lA -lB -lA`

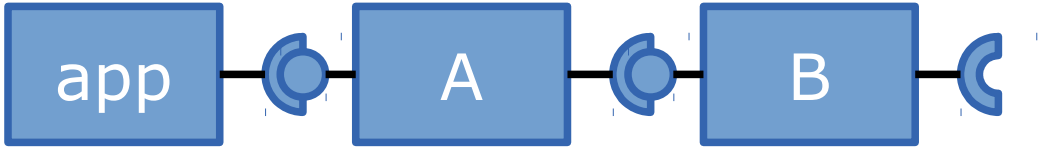
Linking – reverse dep. (cont.)



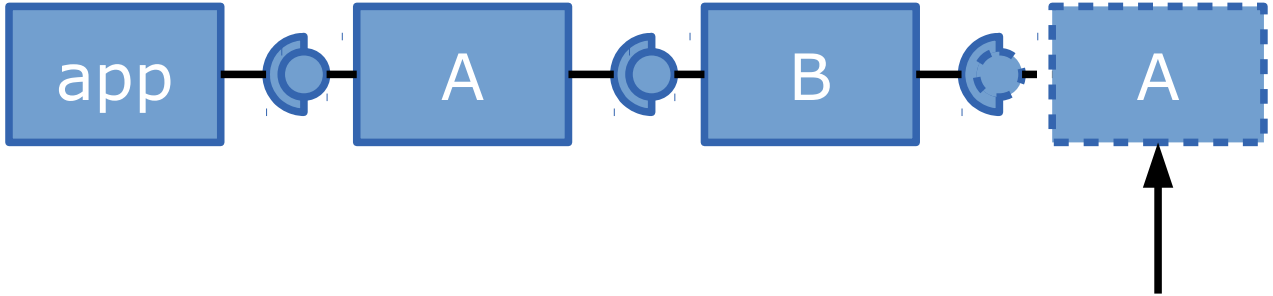
Linking – reverse dep. (cont.)



Linking – reverse dep. (cont.)



Linking – reverse dep. (cont.)



Just the missing symbols

Linking – library groups

- Declare library groups instead
  - Wrap library list with `--start-group`, `--end-group`
    - Shorthand: `-(`, `-)`
    - `g++ foo.obj -Wl,-\( -lA -lB -Wl,-\)`
- Results in exhaustive search for symbols

Linking – library groups (cont.)

- Actually used for non-library objects (TUs)
- Caveat: the exhaustive search!
  - Manual warns of possible performance hit
  - Not observed here, but keep that in mind!

Running the binary in debugger

```
inequation@spearhead:~/projects/largebinary$ gdb --silent largebinary
Reading symbols from /home/inequation/projects/largebinary/largebinary...
[zzz... several minutes later...]
done.
(gdb)
```

Caching the gdb-index

- Large codebases generate **heavy** debug symbols (**hundreds** of MBs)
- GDB does symbol indexing at **every single startup** ☹
  - **Massive** waste of time!

Caching the gdb-index (cont.)

- Solution: fold indexing into the build process
- Old linkers: as described in [GNU01]
- New linkers (i.e. gold): `--gdb-index`
  - May need to forward from compiler driver: `-Wl,--gdb-index`

Agenda

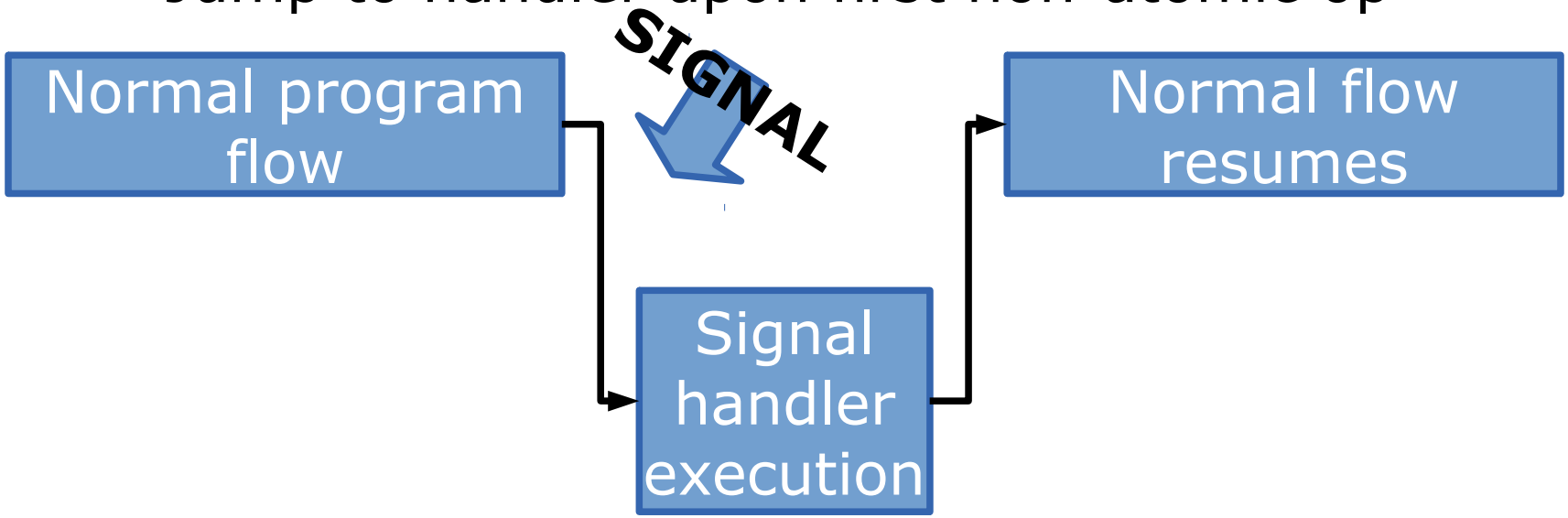
- Build system improvements
- **Signal handlers**
- Memory debugging with Valgrind
- OpenGL debugging techniques

Signal handlers

- Unix signals are async notifications
- Sources can be:
  - the process itself
  - another process
  - user
  - kernel

Signal handlers (cont.)

- A lot like interrupts
  - Jump to handler upon first non-atomic op



Signal handlers (cont.)

- System installs a default handler
  - Usually terminates and/or dumps core
    - Core ≈ minidump in Windows parlance, but entire mapped address range is dumped (truncated to `RLIMIT_CORE` bytes)
  - See `signal(7)` for default actions

Signal handlers (cont.)

- Can (should!) specify custom handlers
- Get/set handlers via `sigaction(2)`
  - `void handler(int, siginfo_t *, void *);`
  - Needs `SA_SIGINFO` flag in `sigaction()` call
- Extensively covered in [BENYOSSEF08]

Interesting `siginfo_t` fields

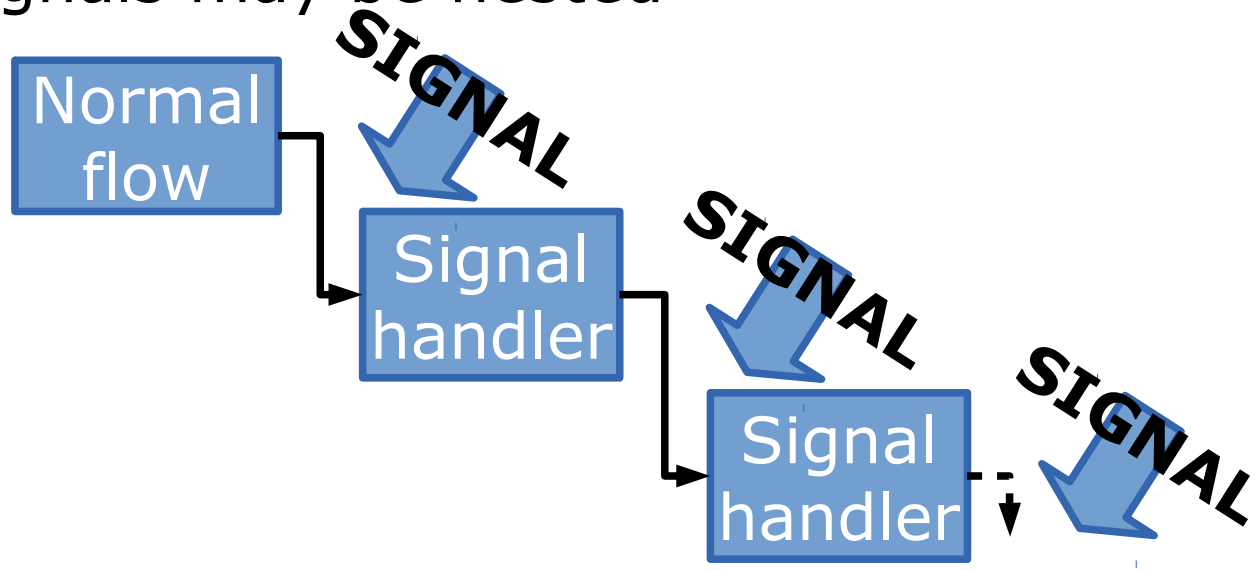
- `si_code` – reason for sending the signal
  - Examples: signal source, FP over/underflow, memory permissions, unmapped address
- `si_addr` – memory location (if relevant)
  - `SIGILL`, `SIGFPE`, `SIGSEGV`, `SIGBUS` and `SIGTRAP`

Interesting signals

- Worth catching
  - SIGSEGV, SIGILL, SIGHUP, SIGQUIT, SIGTRAP, SIGIOT, SIGBUS, SIGFPE, SIGTERM, SIGINT
- Worth ignoring
  - signal(signum, SIG_IGN);
  - SIGCHLD, SIGPIPE

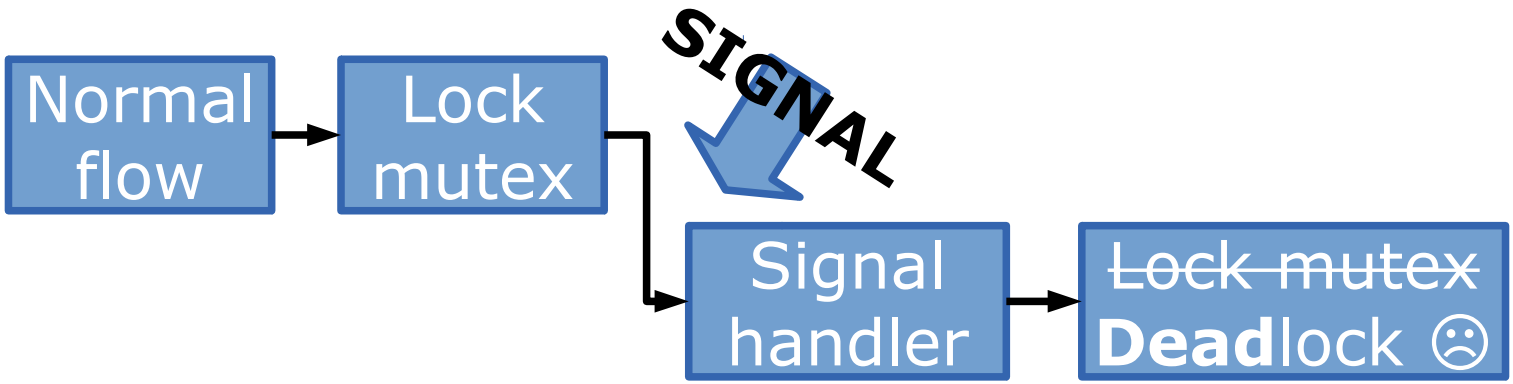
Signal handling caveats

- Prone to race conditions
  - Signals may be nested



Signal handling caveats (cont.)

- Prone to race conditions
  - Can't share locks with the main program

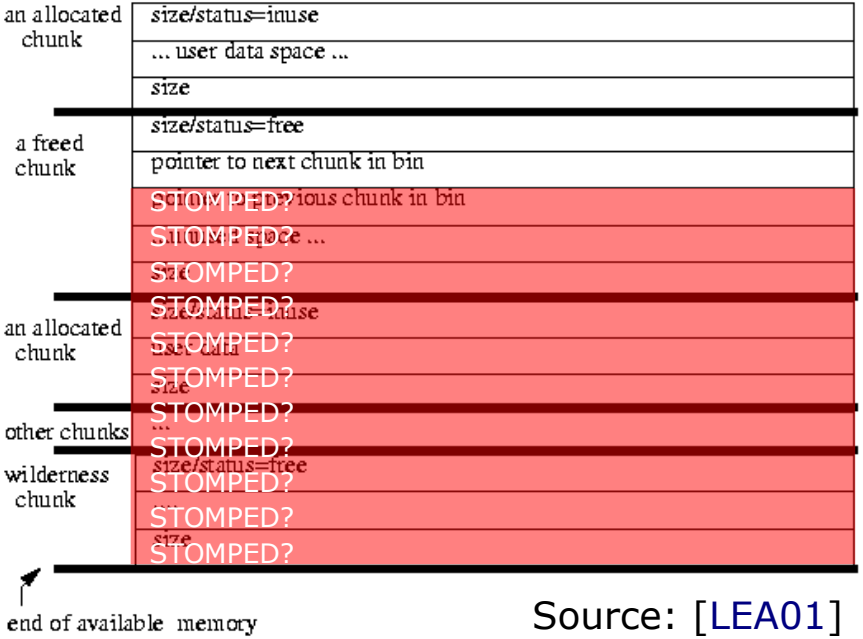


Signal handling caveats (cont.)

- Prone to race conditions
  - Can't call async-unsafe/non-reentrant functions
    - See `signal(7)` for a list of safe ones
    - Notable functions **not** on the list:
      - `printf()` and friends (formatted output)
      - `malloc()` and `free()`

Signal handling caveats (cont.)

- **Not safe** to allocate or free heap memory



Source: [LEA01]

Signal handling caveats (cont.)

- Custom handlers do not dump core
  - At handler installation time:
    - Raise `RLIMIT_CORE` to desired core size
  - Inside handler, after custom logging:
    - Restore default handler using `signal(2)` or `sigaction(2)`
    - `raise(signum);`

Safe stack walking

- glibc provides `backtrace(3)` and friends
- Symbols are read from the dynamic symbol table
  - Pass `-rdynamic` at compile-time to populate

Safe stack walking (cont.)

- `backtrace_symbols()` internally calls `malloc()`
  - **<u>Not</u>** safe... ☹
  - Still, can get away with it most of the time (YMMV)

Example “proper” solution

- Fork a watchdog process in `main()`
  - Communicate over a FIFO pipe
- In signal handler:
  - Collect & send information down the pipe
  - `backtrace_symbols_fd()` down the pipe
- Demo code: is.gd/GDCE14Linux

Agenda

- Build system improvements
- Signal handlers
- **Memory debugging with Valgrind**
- OpenGL debugging techniques

Is this even related to porting?

- Yes! Portability bugs easily overlooked
- Hardcoded struct sizes/offsets
- OpenGL buffers
- Incorrect binary packing/unpacking
- “How did we/they manage to ship that?!”

What is Valgrind?

- Framework for dynamic, runtime analysis
- Dynamic recompilation
  - machine code → IR → tool → machine code
  - Performance typically at 25-20% of unmodified code
    - Worse if heavily threaded – execution is serialized

What is Valgrind? (cont.)

- Many tools in it:
  - Memory error detectors (**Memcheck**, SGcheck)
  - Cache profilers (Cachegrind, Callgrind)
  - Thread error detectors (Helgrind, DRD)
  - Heap profilers (Massif, DHAT)

Memcheck basics

- Basic usage extremely simple
  - …as long as you use the vanilla libc `malloc()`
  - `valgrind ./app`
- Will probably report a ton of errors on the first run!
  - Again: “How did they manage to ship that?!”

Memcheck basics (cont.)

- Many false positives, esp. in 3rd parties
  - Xlib, NVIDIA driver
- Can suppress them via suppress files
  - Call Valgrind with `--gen-suppressions=yes` to generate suppression definitions
  - Be careful with that! Can let OpenGL bugs slip!

Contrived example

```
#include <stdlib.h>
int main(int argc, char *argv[]) {
    int foo, *ptr1 = &foo;
    int *ptr2 = malloc(sizeof(int));
    if (*ptr1)
        ptr2[1] = 0xabad1dea;
    else
        ptr2[1] = 0x15bad700;
    ptr2[0] = ptr2[2];
    return *ptr1;
}
```

Demo code:
is.gd/GDCE14Linux

Valgrind output for such

```
==8925== Conditional jump or move depends on
    uninitialised value(s)
==8925== Invalid write of size 4
==8925== Invalid read of size 4
==8925== Syscall param exit_group(status)
    contains uninitialised byte(s)
==8925== LEAK SUMMARY:
==8925==    definitely lost: 4 bytes in 1 blocks
```

What about custom allocators?

- Custom memory pool & allocation algo
- Valgrind only "sees" `mmap()/munmap()` of multiples of entire memory pages
- **All access** within those pages – now valid!
- How to track errors?

Client requests

- Allow annotation of custom allocators
- ~20 C macros defined in `valgrind.h`
  - Common and per-tool requests exist
- Can be cut out with `-DNVALGRIND`
- Detailed description in [VALGRIND01]

Example: Instrumenting dlmalloc

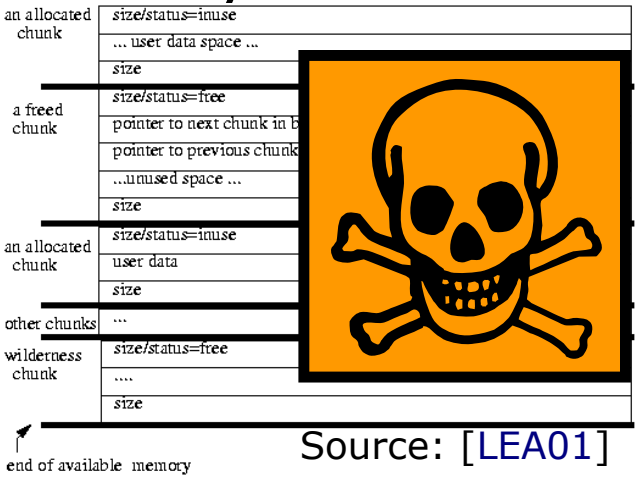
- 2.8.4 instrumentation from [CRYSTAL01]
- Demo code: is.gd/GDCE14Linux
  - Compile the sample with `-DDLMALLOC`
  - Similar results to libc `malloc()`

Other uses of client requests

- Pointer validation
  - Is address mapped? Is it defined?
- Mid-session leak checks
  - Level transitions

Other uses of client req. (cont.)

- Poisoning memory regions
  - Ensuring signal handlers don't touch the heap
  - Ensuring geometry buffers aren't read on CPU



Source: [LEA01]

Debugging inside Valgrind

- A gdbserver for “remote” debugging
- `SIGTRAP` (breakpoint) on every error
- **Unlimited** memory watchpoints!
  - Data breakpoints in Visual Studio parlance
  - Cf. 4 single-word hardware debug registers on x86

Debugging inside Valgrind (cont.)

- Terminal A:
  - `valgrind --vgdb=yes --vgdb-error=0 ./MyGame`
- Terminal B:
  - `gdb ./MyGame`
  - `target remote | vgdb`

Agenda

- Build system improvements
- Signal handlers
- Memory debugging with Valgrind
- **OpenGL debugging techniques**

Ye Olde Way

- Call `glGetError()` after each OpenGL call
- Get 1 of 8 (**sic!**) error codes
- Look up the call in the manual
- See what this particular error means in this particular context…

Ye Olde Way (cont.)

- …Then check what was **actually** the case
  - **6 possible reasons** for `GL_INVALID_VALUE` in `glTexImage*()` alone! See [OPENGL01]
  - Usually: attach a debugger, replay the scenario…
- This **sucks!**

Ye Olde Way (cont.)

- …Then check what was **actually** the case
  - **6 possible reasons** for `GL_INVALID_VALUE` in `glTexImage*()` alone! See [OPENGL01]
  - Usually: attach a debugger, replay the scenario…
- This ~~**sucks!**~~ used to suck ☺

Debug callback

- Never call `glGetError()` again!
- Much more detailed information
  - Incl. performance tips from the driver
  - Good to check what different drivers say
- May not work without a debug OpenGL context (`GLX_CONTEXT_DEBUG_BIT_ARB`)

Debug callback (cont.)

- Provided by either of (ABI-compatible): GL_KHR_debug [OPENGL02], GL_ARB_debug_output [OPENGL03]

	OpenGL	OpenGL ES	NVIDIA (official)	AMD (official)	Intel (Mesa)	AMD (Mesa)
ARB _debug _output	√	×	√	√	√	√
KHR _debug	√	√	√	√	×	×

Debug callback (cont.)

```
void callback(GLenum source,
              GLenum type,
              GLuint id,
              GLenum severity,
              GLsizei length,
              const GLchar* message,
              const void* userParam);
```

Filter by source, type, severity or individual messages

Debug callback (cont.)

- Verbosity can be controlled (filtering)
  - `glDebugMessageControl[ARB]()`
  - [OPENGL02][OPENGL03]
- Turn to 11 (GL_DONT_CARE) for valuable perf information!
  - Memory type for buffers, unused mip levels…

API call tracing

- Record a trace of the run of the application
- Replay and review the trace
  - Look up OpenGL state at a particular call
  - Inspect state variables, resources and objects: textures, shaders, buffers...
- apitrace or VOGL

Well, this is not helpful...

```
memcpy(0x1061f800, [binary data, size = 5.76562 kb], 5904)
glUnmapNamedBufferEXT(261) = GL_TRUE
memcpy(0x1061edc0, [binary data, size = 396 bytes], 396)
glUnmapBuffer(GL_ELEMENT_ARRAY_BUFFER) = GL_TRUE
glGenBuffers(1, [263])
glNamedBufferDataEXT(263, 1356, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(263, GL_WRITE_ONLY) = 0x10621340
memcpy(0x10621340, [binary data, size = 1.32422 kb], 1356)
glUnmapNamedBufferEXT(263) = GL_TRUE
glGenBuffers(1, [264])
glNamedBufferDataEXT(264, 5424, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(264, GL_WRITE_ONLY) = 0x10621f00
memcpy(0x10621f00, [binary data, size = 5.29688 kb], 5424)
glUnmapNamedBufferEXT(264) = GL_TRUE
glGenBuffers(1, [265])
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 265)
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 624, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(263, GL_READ_WRITE) = 0x10621340
glMapNamedBufferEXT(264, GL_READ_WRITE) = 0x10621f00
glMapBufferRange(GL_ELEMENT_ARRAY_BUFFER, 0, 624, GL_MAP_READ_BIT | GL_MAP_WRITE_BIT) = 0x10623480
memcpy(0x10621340, [binary data, size = 1.32422 kb], 1356)
glUnmapNamedBufferEXT(263) = GL_TRUE
memcpy(0x10621f00, [binary data, size = 5.29688 kb], 5424)
glUnmapNamedBufferEXT(264) = GL_TRUE
memcpy(0x10623480, [binary data, size = 624 bytes], 624)
glUnmapBuffer(GL_ELEMENT_ARRAY_BUFFER) = GL_TRUE
glGenBuffers(1, [266])
glNamedBufferDataEXT(266, 1608, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(266, GL_WRITE_ONLY) = 0x10623b40
memcpy(0x10623b40, [binary data, size = 1.57031 kb], 1608)
glUnmapNamedBufferEXT(266) = GL_TRUE
glGenBuffers(1, [267])
glNamedBufferDataEXT(267, 6432, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(267, GL_WRITE_ONLY) = 0x10624200
memcpy(0x10624200, [binary data, size = 6.28125 kb], 6432)
glUnmapNamedBufferEXT(267) = GL_TRUE
glGenBuffers(1, [268])
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 268)
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 624, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(266, GL_READ_WRITE) = 0x10623b40
glMapNamedBufferEXT(267, GL_READ_WRITE) = 0x10624200
glMapBufferRange(GL_ELEMENT_ARRAY_BUFFER, 0, 624, GL_MAP_READ_BIT | GL_MAP_WRITE_BIT) = 0x10623740
memcpy(0x10623b40, [binary data, size = 1.57031 kb], 1608)
glUnmapNamedBufferEXT(266) = GL_TRUE
memcpy(0x10624200, [binary data, size = 6.28125 kb], 6432)
glUnmapNamedBufferEXT(267) = GL_TRUE
memcpy(0x10623740, [binary data, size = 624 bytes], 624)
glUnmapBuffer(GL_ELEMENT_ARRAY_BUFFER) = GL_TRUE
glGenBuffers(1, [269])
glNamedBufferDataEXT(269, 4464, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(269, GL_WRITE_ONLY) = 0x10626080
memcpy(0x10626080, [binary data, size = 4.35938 kb], 4464)
glUnmapNamedBufferEXT(269) = GL_TRUE
glGenBuffers(1, [270])
glNamedBufferDataEXT(270, 17856, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(270, GL_WRITE_ONLY) = 0x10627840
memcpy(0x10627840, [binary data, size = 17.4375 kb], 17856)
glUnmapNamedBufferEXT(270) = GL_TRUE
glGenBuffers(1, [271])
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 271)
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 1872, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(269, GL_READ_WRITE) = 0x10626080
glMapNamedBufferEXT(270, GL_READ_WRITE) = 0x10627840
glMapBufferRange(GL_ELEMENT_ARRAY_BUFFER, 0, 1872, GL_MAP_READ_BIT | GL_MAP_WRITE_BIT) = 0x1062be40
memcpy(0x10626080, [binary data, size = 4.35938 kb], 4464)
glUnmapNamedBufferEXT(269) = GL_TRUE
memcpy(0x10627840, [binary data, size = 17.4375 kb], 17856)
glUnmapNamedBufferEXT(270) = GL_TRUE
memcpy(0x1062be40, [binary data, size = 1.82812 kb], 1872)
glUnmapBuffer(GL_ELEMENT_ARRAY_BUFFER) = GL_TRUE
glGenBuffers(1, [272])
glNamedBufferDataEXT(272, 7620, NULL, GL_STATIC_DRAW)
glMapNamedBufferEXT(272, GL_WRITE_ONLY) = 0x1062ca40
memcpy(0x1062ca40, [binary data, size = 7.44141 kb], 7620)
glUnmapNamedBufferEXT(272) = GL_TRUE
glGenBuffers(1, [273])
```

Much better!

```
glBindSampler(3, 4)
glBindSampler(2, 3)
glBindSampler(1, 2)
glBindSampler(0, 1)
> glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]update)
glDebugMessageInsert(GL_DEBUG_SOURCE_APPLICATION, GL_DEBUG_TYPE_OTHER, 0, GL_DEBUG_SEVERITY_NOTIFICATION, -1, "FBO cache hit: 1 = { c0 = 1,  d = 2 }")
glBindFramebuffer(GL_FRAMEBUFFER, 1)
glClearColor(0, 0, 0, 1)
glClear(GL_COLOR_BUFFER_BIT)
glUseProgram(190)
glBindMultiTextureEXT(GL_TEXTURE0, GL_TEXTURE_2D, 281)
glBindMultiTextureEXT(GL_TEXTURE1, GL_TEXTURE_2D, 282)
glBindMultiTextureEXT(GL_TEXTURE2, GL_TEXTURE_2D, 283)
glDisable(GL_BLEND)
v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, "virtual void [redacted]( ...)")
    glDebugMessageInsert(GL_DEBUG_SOURCE_APPLICATION, GL_DEBUG_TYPE_OTHER, 0, GL_DEBUG_SEVERITY_NOTIFICATION, -1, "FBO up to date")
    glViewport(0, 0, 1280, 720)
    glScissor(0, 0, 1280, 720)
  v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, "virtual void [redacted] ...)")
    v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 0, -1, "void [redacted]...)")
      v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 0, -1, MOJOSHADER_glProgramReady)
            glDisableVertexAttribArray(2)
            glUniform4fv(3, 1, [1, 1, 1, 1])
            glPopDebugGroup()
          glDebugMessageInsert(GL_DEBUG_SOURCE_APPLICATION, GL_DEBUG_TYPE_OTHER, 0, GL_DEBUG_SEVERITY_NOTIFICATION, -1, "FBO up to date")
          glPopDebugGroup()
        glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 16, [binary data, size = 56 bytes])
        glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 16, [binary data, size = 56 bytes])
        glDrawArrays(GL_QUADS, 0, 4)
        glPopDebugGroup()
    glDebugMessageInsert(GL_DEBUG_SOURCE_APPLICATION, GL_DEBUG_TYPE_OTHER, 0, GL_DEBUG_SEVERITY_NOTIFICATION, -1, "FBO up to date")
    glViewport(0, 0, 1280, 720)
    glScissor(0, 0, 1280, 720)
    glPopDebugGroup()
v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]drawUI)
    glDebugMessageInsert(GL_DEBUG_SOURCE_APPLICATION, GL_DEBUG_TYPE_OTHER, 0, GL_DEBUG_SEVERITY_NOTIFICATION, -1, "FBO up to date")
    glViewport(0, 0, 1280, 720)
    glScissor(0, 0, 1280, 720)
  v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted])
        glUseProgram(160)
        glEnable(GL_BLEND)
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
        glUseProgram(163)
        glBindMultiTextureEXT(GL_TEXTURE0, GL_TEXTURE_2D, 22)
        glSamplerParameteri(1, GL_TEXTURE_WRAP_S, GL_REPEAT)
        glSamplerParameteri(1, GL_TEXTURE_WRAP_T, GL_REPEAT)
        glSamplerParameteri(1, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)
        glSamplerParameteri(1, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
      v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, "virtual void [redacted]...)")
        v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 0, -1, "void [redacted]...)")
          v glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 0, -1, MOJOSHADER_glProgramReady)
                glEnableVertexAttribArray(2)
                glUniform4fv(1, 4, [0.0015625, 0, 0, -1, 0, -0.00277778, 0, ...1])
                glPopDebugGroup()
              glDebugMessageInsert(GL_DEBUG_SOURCE_APPLICATION, GL_DEBUG_TYPE_OTHER, 0, GL_DEBUG_SEVERITY_NOTIFICATION, -1, "FBO up to date")
              glPopDebugGroup()
            glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 48, [binary data, size = 252 bytes])
            glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 48, [binary data, size = 256 bytes])
            glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 48, [binary data, size = 252 bytes])
            glDrawArrays(GL_TRIANGLES, 0, 6)
            glPopDebugGroup()
          glPopDebugGroup()
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
    > glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION, 4294967295, -1, [redacted]fillRect)
```

Annotating the call stream

	KHR_debug	EXT_debug_marker	EXT_debug_label	GREMEDY_string_marker	GREMEDY_frame_terminator
One-off messages	√	√	×	√	×
Call grouping	√	√	×	×	×
Object labels	√	×	√	×	×
Frame terminators	×	×	×	×	√
Support	Good	Limited	Limited	Limited	Limited

Annotating the call stream (cont.)

- All aforementioned extensions supported by apitrace regardless of driver
- Recommended: GL_KHR_debug

Annotating the call stream (cont.)

- Call grouping
  - `glPushDebugGroup()/glPopDebugGroup()`
- One-off messages
  - `glDebugMessageInsert[ARB]()`
  - `glStringMarkerGREMEDY()`

Object labelling

- `glObjectLabel()`, `glGetObjectLabel()`
  - Buffer, shader, program, vertex array, query, program pipeline, transform feedback, sampler, texture, render buffer, frame buffer, display list
- `glObjectPtrLabel()`, `glGetObjectPtrLabel()`
  - Sync objects

Annotation caveats

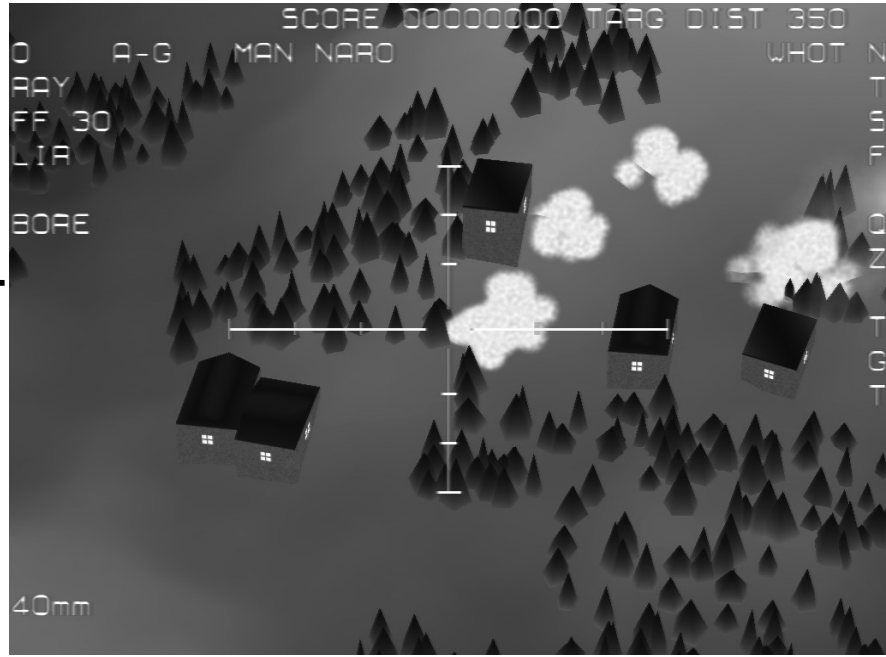
- Multi-threaded grouping may break hierarchy
- `glDebugMessageInsert()` calls the debug callback, polluting error streams
  - Workaround: drop if `source == GL_DEBUG_SOURCE_APPLICATION`

Example 1: PIX events emulation

```
#define D3DPERF_BeginEvent(colour, name)                    \
    if (GLEW_KHR_debug && threadOwnsDevice())               \
        glPushDebugGroup(GL_DEBUG_SOURCE_APPLICATION,\
                         (GLuint)colour, -1, name)
#define D3DPERF_EndEvent()                                  \
    if (GLEW_KHR_debug && threadOwnsDevice())               \
        glPopDebugGroup()
```

Example 2: Game tech demo

- University assignment from 2009 ☺
- Annotated OpenGL 1.4
- Demo code: is.gd/GDCE14Linux



Takeaway

- gcc-multilib is **the** prerequisite for 32/64-bit cross-compilation
- Switching back and forth between Clang and gcc is easy and useful
- Link times can be greatly improved by using gold

Takeaway (cont.)

- Caching the gdb-index improves debugging experience
- Crash handling is easy to do, tricky to get right

Takeaway (cont.)

- Valgrind is an enormous aid in memory debugging
- Even when employing custom allocators
- OpenGL debugging experience can be vastly improved using some extensions

Questions?

lgodlewski@nordicgames.at
@TheIneQuation
inequation.org

Thank you!

Further Nordic Games information:

🔗 www.nordicgames.at

Development information:

🔗 www.grimloregames.com

References

- **LARABEL13** – Larabel, M. *"Clang 3.4 Performance Very Strong Against GCC 4.9"* [link]
- **GNU01** – *"Index Files Speed Up GDB"* [link]
- **GNU02** – *"Options for Debugging Your Program or GCC"* [link]
- **BENYOSSEF08** – Ben-Yossef, G. *"Crash N' Burn: Writing Linux application fault handlers"* [link]
- **LEA01** – Lea, D. *"A Memory Allocator"* [link]
- **VALGRIND01** – *"The Client Request mechanism"* [link]
- **CRYSTAL01** – *"Crystal Space 3D SDK"* [link]
- **OPENGL01** – *"glTexImage2D"* [link]
- **OPENGL02** – *"ARB_debug_output"* [link]
- **OPENGL03** – *"KHR_debug"* [link]
- **XDG01** – *"XDG Base Directory Specification"* [link]
- **`<page>(<section>)`**, e.g. `sigaction(2)` – *"Linux Programmer's Manual"*; to view, type `man <section> <page>` into a terminal or a web search engine

Special thanks

Fabian Giesen
Katarzyna Griksa
Damian Kobiałka
Jetro Lauha
Eric Lengyel
Krzysztof Narkowicz
Reinhard Pollice
Bartłomiej Wroński
Kacper Ząber

Bonus slides!

- OpenGL resource leak checking
- Intel i965 driver vs stack
- Locating user data according to FreeDesktop.org guidelines
- Thread priorities in Linux
- Additional/new debug features

OpenGL resource leak checking

Courtesy of Eric Lengyel & Fabian Giesen

```
static void check_for_leaks()
{
    GLuint max_id = 10000; // better idea would be to keep track of assigned names.
    GLuint id;
    // if brute force doesn't work, you're not applying it hard enough
    for ( id = 1 ; id <= max_id ; id++ )
    {
#define CHECK( type ) if ( glIs##type( id ) ) fprintf( stderr, "GLX: leaked " #type " handle 0x%x\n", (unsigned int) id )
        CHECK( Texture );
        CHECK( Buffer );
        CHECK( Framebuffer );
        CHECK( Renderbuffer );
        CHECK( VertexArray );
        CHECK( Shader );
        CHECK( Program );
        CHECK( ProgramPipeline );
#undef CHECK
    }
}
```

Intel i965 vs stack

- Been chasing a segfault on a `call` instruction down `_mesa_Clear()` (`glClear()`)
- Region of code copy/pasted from D3D renderer
- Address mapped, so not an invalid jump...
- Only 16 function frames – surely this can't be a stack overflow?

Intel i965 vs stack (cont.)

- Oh no, wait:
  - Check ESP against `/proc/[pid]/maps`
  - Yup, encroaching on unmapped address space
- Moral: cut your render some stack slack (160+ kB), or Mesa will blow it up with locals (e.g. in clear shader generation)

Locating user data

- There is a spec for that – see [XDG01]
- Savegames, screenshots, options etc.:
  - $XDG_CONFIG_HOME or ~/.config/<app>
- Caches of all kinds:
  - $XDG_CACHE_HOME or ~/.cache/<app>
- Per-user persistent data (e.g. DLC):
  - $XDG_DATA_HOME or ~/.local/share/<app>

Locating user data (cont.)

- <app> subdirectory currently unregulated
  - De-facto standard: simplified or "Unix name"
    - Lowercase, "safe" ASCII characters, e.g. `blender`
- When asked, XDG people suggest rev-DNS
  - `com.company.appname`

Thread priorities in Linux

- Priority elevation **requires** root permissions 😱
  - No user will ever grant you root (scary!)
  - Reason: DoS protection in servers (probably)
- Priority can be tweaked with `nice()`
  - Think "how nice the process is to others"
  - Being nice to everyone will starve your process
  - Niceness can be negative (but only with root)

Thread priorities in Linux (cont.)

- Why not `setpriority(2)`?
  - Also sets scheduling algorithm → here be dragons
  - Priority values have different meaning per scheduler
  - Still needs root
- What about `capabilities(7)`?
  - This might actually work if your users trust you
  - Demo code: is.gd/GDCE14Linux

Thread priorities in Linux (cont.)

- Don't all threads in a process share niceness?
  - They should, according to POSIX, but they don't!
  - One of the few cases where Linux is non-compliant

Additional/new debug features

- Additional debug info: `-g3`
  - Including #defines (macros)
- Better debugger performance [GNU02]:
  - `-fdebug-types-section`: improved layout
  - `-gpubnames`: new format for index