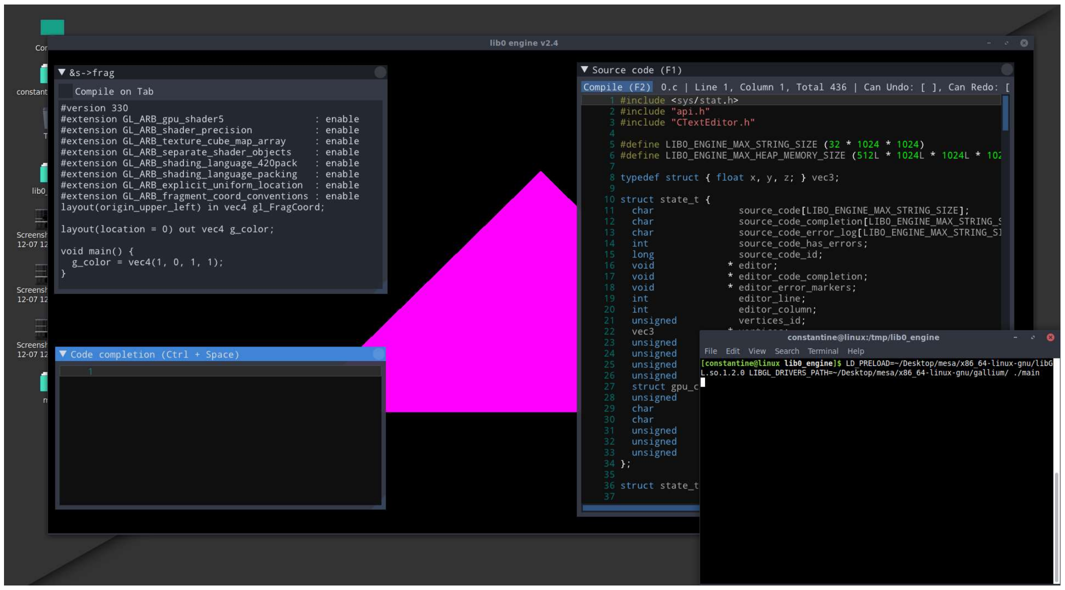
Procedural

Arch Linux is not supported due to confirmed Mesa bug.

#13690

36 posts / 1 project

Here's a screenshot of lib0 engine running with a custom Mesa build (and now Arch Linux's stock Mesa 17.2.6) on Manjaro MATE 17.0.4:



```
#version 330
#extension GL_ARB_gpu_shader5                : enable
#extension GL_ARB_shader_precision           : enable
#extension GL_ARB_texture_cube_map_array     : enable
#extension GL_ARB_separate_shader_objects    : enable
#extension GL_ARB_shading_language_420pack   : enable
#extension GL_ARB_shading_language_packing   : enable
#extension GL_ARB_explicit_uniform_location  : enable
#extension GL_ARB_fragment_coord_conventions : enable
layout(origin_upper_left) in vec4 gl_FragCoord;

layout(location = 0) out vec4 g_color;

void main() {
  g_color = vec4(1, 0, 1, 1);
}
```

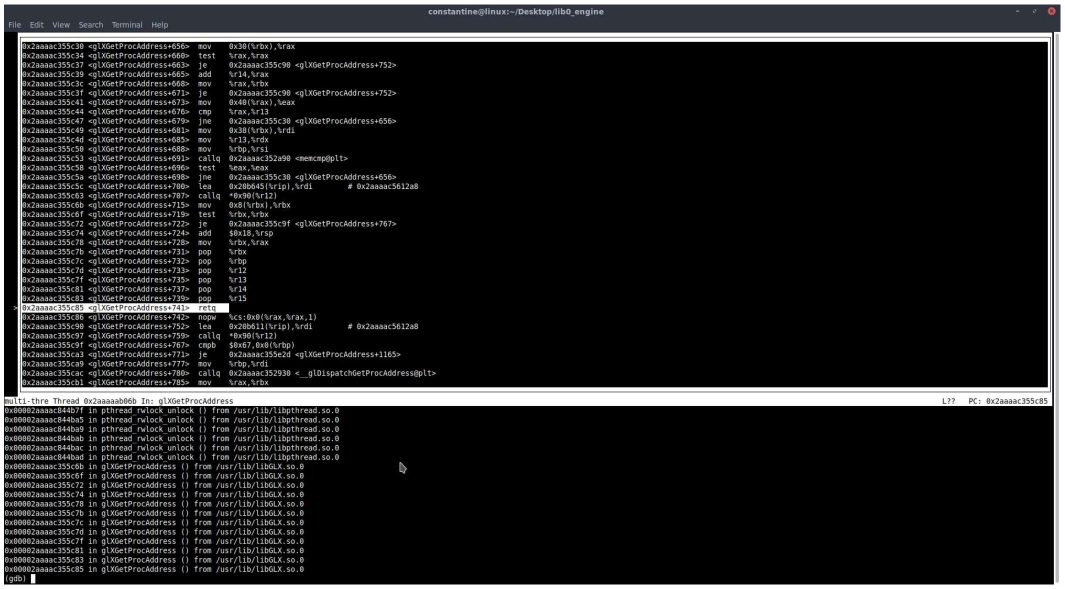
```
0.c | Line 1, Column 1, Total 436 | Can Undo: [ ], Can Redo: [
#include <sys/stat.h>
#include "api.h"
#include "CTextEditor.h"

#define LIB0_ENGINE_MAX_STRING_SIZE (32 * 1024 * 1024)
#define LIB0_ENGINE_MAX_HEAP_MEMORY_SIZE (512L * 1024L * 1024L * 102

typedef struct { float x, y, z; } vec3;

struct state_t {
  char                  source_code[LIB0_ENGINE_MAX_STRING_SIZE];
  char                  source_code_completion[LIB0_ENGINE_MAX_STRING_S
  char                  source_code_error_log[LIB0_ENGINE_MAX_STRING_SI
  int                   source_code_has_errors;
  long                  source_code_id;
  void                * editor;
  void                * editor_code_completion;
  void                * editor_error_markers;
  int                   editor_line;
  int                   editor_column;
  unsigned              vertices_id;
  vec3
  unsigned
  unsigned
  unsigned
  unsigned
  struct gpu_c
  unsigned
  char
  char
  unsigned
  unsigned
  unsigned
};

struct state_t
```

```
constantine@linux:/tmp/lib0_engine
[constantine@linux lib0_engine]$ LD_PRELOAD=~/Desktop/mesa/x86_64-linux-gnu/libG
L.so.1.2.0 LIBGL_DRIVERS_PATH=~/Desktop/mesa/x86_64-linux-gnu/gallium/ ./main
```

It looks normal, but it's actually broken because when I try to switch focus from a terminal to lib0 engine window **the whole desktop screen turns black**.

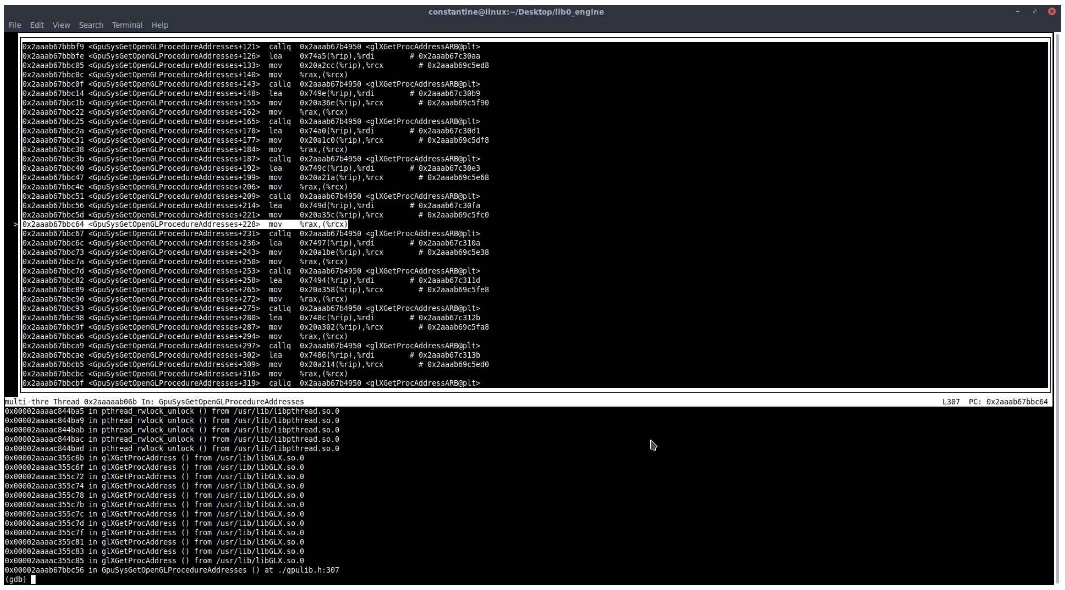
Also, Arch Linux **crashes** the same binary that worked a few minutes before on Ubuntu 16.04 like this:

First it returns from glXGetProcAddress just fine:


constantine@linux:~/Desktop/lib0_engine
File Edit View Search Terminal Help

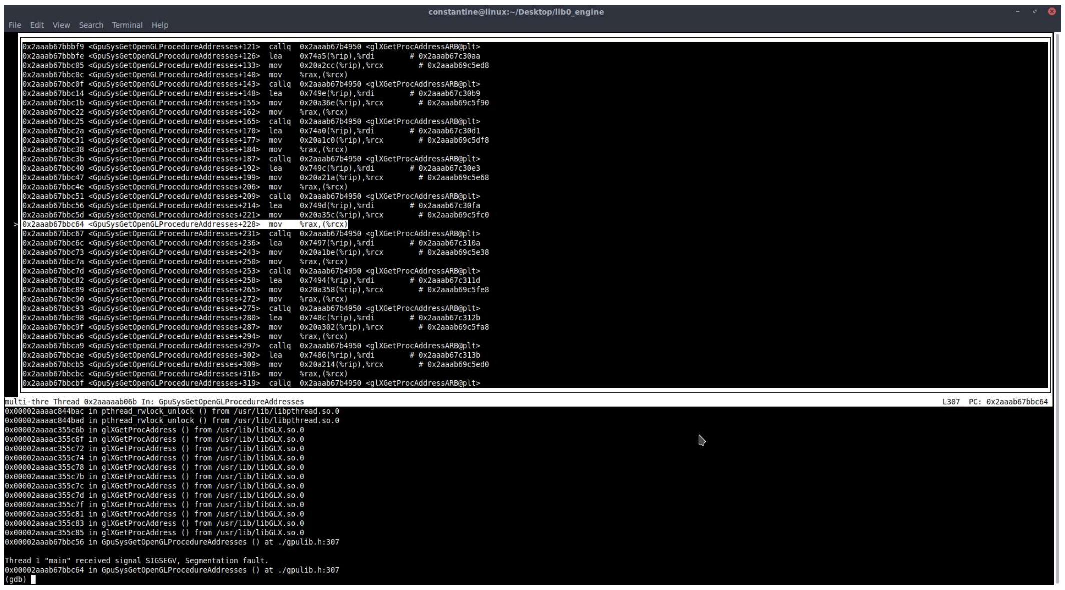
```
 0x2aaaac355c30 <glXGetProcAddress+656>  mov    0x30(%rbx),%rax
 0x2aaaac355c34 <glXGetProcAddress+660>  test   %rax,%rax
 0x2aaaac355c37 <glXGetProcAddress+663>  je     0x2aaaac355c90 <glXGetProcAddress+752>
 0x2aaaac355c39 <glXGetProcAddress+665>  add    %r14,%rax
 0x2aaaac355c3c <glXGetProcAddress+668>  mov    %rax,%rbx
 0x2aaaac355c3f <glXGetProcAddress+671>  je     0x2aaaac355c90 <glXGetProcAddress+752>
 0x2aaaac355c41 <glXGetProcAddress+673>  mov    0x40(%rax),%eax
 0x2aaaac355c44 <glXGetProcAddress+676>  cmp    %rax,%r13
 0x2aaaac355c47 <glXGetProcAddress+679>  jne    0x2aaaac355c30 <glXGetProcAddress+656>
 0x2aaaac355c49 <glXGetProcAddress+681>  mov    0x38(%rbx),%rdi
 0x2aaaac355c4d <glXGetProcAddress+685>  mov    %r13,%rdx
 0x2aaaac355c50 <glXGetProcAddress+688>  mov    %rbp,%rsi
 0x2aaaac355c53 <glXGetProcAddress+691>  callq  0x2aaaac352a90 <memcmp@plt>
 0x2aaaac355c58 <glXGetProcAddress+696>  test   %eax,%eax
 0x2aaaac355c5a <glXGetProcAddress+698>  jne    0x2aaaac355c30 <glXGetProcAddress+656>
 0x2aaaac355c5c <glXGetProcAddress+700>  lea    0x20b645(%rip),%rdi        # 0x2aaaac5612a8
 0x2aaaac355c63 <glXGetProcAddress+707>  callq  *0x90(%r12)
 0x2aaaac355c6b <glXGetProcAddress+715>  mov    0x8(%rbx),%rbx
 0x2aaaac355c6f <glXGetProcAddress+719>  test   %rbx,%rbx
 0x2aaaac355c72 <glXGetProcAddress+722>  je     0x2aaaac355c9f <glXGetProcAddress+767>
 0x2aaaac355c74 <glXGetProcAddress+724>  add    $0x18,%rsp
 0x2aaaac355c78 <glXGetProcAddress+728>  mov    %rbx,%rax
 0x2aaaac355c7b <glXGetProcAddress+731>  pop    %rbx
 0x2aaaac355c7c <glXGetProcAddress+732>  pop    %rbp
 0x2aaaac355c7d <glXGetProcAddress+733>  pop    %r12
 0x2aaaac355c7f <glXGetProcAddress+735>  pop    %r13
 0x2aaaac355c81 <glXGetProcAddress+737>  pop    %r14
 0x2aaaac355c83 <glXGetProcAddress+739>  pop    %r15
>0x2aaaac355c85 <glXGetProcAddress+741>  retq
 0x2aaaac355c86 <glXGetProcAddress+742>  nopw   %cs:0x0(%rax,%rax,1)
 0x2aaaac355c90 <glXGetProcAddress+752>  lea    0x20b611(%rip),%rdi        # 0x2aaaac5612a8
 0x2aaaac355c97 <glXGetProcAddress+759>  callq  *0x90(%r12)
 0x2aaaac355c9f <glXGetProcAddress+767>  cmpb   $0x67,0x0(%rbp)
 0x2aaaac355ca3 <glXGetProcAddress+771>  je     0x2aaaac355e2d <glXGetProcAddress+1165>
 0x2aaaac355ca9 <glXGetProcAddress+777>  mov    %rbp,%rdi
 0x2aaaac355cac <glXGetProcAddress+780>  callq  0x2aaaac352930 <__glDispatchGetProcAddress@plt>
 0x2aaaac355cb1 <glXGetProcAddress+785>  mov    %rax,%rbx
multi-thre Thread 0x2aaaaab06b In: glXGetProcAddress                             L??   PC: 0x2aaaac355c85
0x00002aaaac844b7f in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844ba5 in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844ba9 in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844bab in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844bac in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844bad in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac355c6b in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c6f in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c72 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c74 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c78 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7b in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7c in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7d in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7f in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c81 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c83 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c85 in glXGetProcAddress () from /usr/lib/libGLX.so.0
(gdb)
```

Then it moves returned pointer with no issues too:



```
   0x2aaab67bbbf9 <GpuSysGetOpenGLProcedureAddresses+121>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbbfe <GpuSysGetOpenGLProcedureAddresses+126>  lea    0x74a5(%rip),%rdi        # 0x2aaab67c30aa
   0x2aaab67bbc05 <GpuSysGetOpenGLProcedureAddresses+133>  mov    0x20a2cc(%rip),%rcx        # 0x2aaab69c5ed8
   0x2aaab67bbc0c <GpuSysGetOpenGLProcedureAddresses+140>  mov    %rax,(%rcx)
   0x2aaab67bbc0f <GpuSysGetOpenGLProcedureAddresses+143>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc14 <GpuSysGetOpenGLProcedureAddresses+148>  lea    0x749e(%rip),%rdi        # 0x2aaab67c30b9
   0x2aaab67bbc1b <GpuSysGetOpenGLProcedureAddresses+155>  mov    0x20a36e(%rip),%rcx        # 0x2aaab69c5f90
   0x2aaab67bbc22 <GpuSysGetOpenGLProcedureAddresses+162>  mov    %rax,(%rcx)
   0x2aaab67bbc25 <GpuSysGetOpenGLProcedureAddresses+165>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc2a <GpuSysGetOpenGLProcedureAddresses+170>  lea    0x74a0(%rip),%rdi        # 0x2aaab67c30d1
   0x2aaab67bbc31 <GpuSysGetOpenGLProcedureAddresses+177>  mov    0x20a1c0(%rip),%rcx        # 0x2aaab69c5df8
   0x2aaab67bbc38 <GpuSysGetOpenGLProcedureAddresses+184>  mov    %rax,(%rcx)
   0x2aaab67bbc3b <GpuSysGetOpenGLProcedureAddresses+187>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc40 <GpuSysGetOpenGLProcedureAddresses+192>  lea    0x749c(%rip),%rdi        # 0x2aaab67c30e3
   0x2aaab67bbc47 <GpuSysGetOpenGLProcedureAddresses+199>  mov    0x20a21a(%rip),%rcx        # 0x2aaab69c5e68
   0x2aaab67bbc4e <GpuSysGetOpenGLProcedureAddresses+206>  mov    %rax,(%rcx)
   0x2aaab67bbc51 <GpuSysGetOpenGLProcedureAddresses+209>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc56 <GpuSysGetOpenGLProcedureAddresses+214>  lea    0x749d(%rip),%rdi        # 0x2aaab67c30fa
   0x2aaab67bbc5d <GpuSysGetOpenGLProcedureAddresses+221>  mov    0x20a35c(%rip),%rcx        # 0x2aaab69c5fc0
 > 0x2aaab67bbc64 <GpuSysGetOpenGLProcedureAddresses+228>  mov    %rax,(%rcx)
   0x2aaab67bbc67 <GpuSysGetOpenGLProcedureAddresses+231>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc6c <GpuSysGetOpenGLProcedureAddresses+236>  lea    0x7497(%rip),%rdi        # 0x2aaab67c310a
   0x2aaab67bbc73 <GpuSysGetOpenGLProcedureAddresses+243>  mov    0x20a1be(%rip),%rcx        # 0x2aaab69c5e38
   0x2aaab67bbc7a <GpuSysGetOpenGLProcedureAddresses+250>  mov    %rax,(%rcx)
   0x2aaab67bbc7d <GpuSysGetOpenGLProcedureAddresses+253>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc82 <GpuSysGetOpenGLProcedureAddresses+258>  lea    0x7494(%rip),%rdi        # 0x2aaab67c311d
   0x2aaab67bbc89 <GpuSysGetOpenGLProcedureAddresses+265>  mov    0x20a358(%rip),%rcx        # 0x2aaab69c5fe8
   0x2aaab67bbc90 <GpuSysGetOpenGLProcedureAddresses+272>  mov    %rax,(%rcx)
   0x2aaab67bbc93 <GpuSysGetOpenGLProcedureAddresses+275>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc98 <GpuSysGetOpenGLProcedureAddresses+280>  lea    0x748c(%rip),%rdi        # 0x2aaab67c312b
   0x2aaab67bbc9f <GpuSysGetOpenGLProcedureAddresses+287>  mov    0x20a302(%rip),%rcx        # 0x2aaab69c5fa8
   0x2aaab67bbca6 <GpuSysGetOpenGLProcedureAddresses+294>  mov    %rax,(%rcx)
   0x2aaab67bbca9 <GpuSysGetOpenGLProcedureAddresses+297>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbcae <GpuSysGetOpenGLProcedureAddresses+302>  lea    0x7486(%rip),%rdi        # 0x2aaab67c313b
   0x2aaab67bbcb5 <GpuSysGetOpenGLProcedureAddresses+309>  mov    0x20a214(%rip),%rcx        # 0x2aaab69c5ed0
   0x2aaab67bbcbc <GpuSysGetOpenGLProcedureAddresses+316>  mov    %rax,(%rcx)
   0x2aaab67bbcbf <GpuSysGetOpenGLProcedureAddresses+319>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
multi-thre Thread 0x2aaaaab06b In: GpuSysGetOpenGLProcedureAddresses                          L307  PC: 0x2aaab67bbc64
0x00002aaaac844ba5 in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844ba9 in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844bab in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844bac in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844bad in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac355c6b in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c6f in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c72 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c74 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c78 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7b in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7c in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7d in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7f in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c81 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c83 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c85 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaab67bbc56 in GpuSysGetOpenGLProcedureAddresses () at ./gpulib.h:307
(gdb) 
```

But watch this: if I step one assembly instruction further -- it crashes. For no reason.



```
   0x2aaab67bbbf9 <GpuSysGetOpenGLProcedureAddresses+121>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbbfe <GpuSysGetOpenGLProcedureAddresses+126>  lea    0x74a5(%rip),%rdi        # 0x2aaab67c30aa
   0x2aaab67bbc05 <GpuSysGetOpenGLProcedureAddresses+133>  mov    0x20a2cc(%rip),%rcx        # 0x2aaab69c5ed8
   0x2aaab67bbc0c <GpuSysGetOpenGLProcedureAddresses+140>  mov    %rax,(%rcx)
   0x2aaab67bbc0f <GpuSysGetOpenGLProcedureAddresses+143>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc14 <GpuSysGetOpenGLProcedureAddresses+148>  lea    0x749e(%rip),%rdi        # 0x2aaab67c30b9
   0x2aaab67bbc1b <GpuSysGetOpenGLProcedureAddresses+155>  mov    0x20a36e(%rip),%rcx        # 0x2aaab69c5f90
   0x2aaab67bbc22 <GpuSysGetOpenGLProcedureAddresses+162>  mov    %rax,(%rcx)
   0x2aaab67bbc25 <GpuSysGetOpenGLProcedureAddresses+165>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc2a <GpuSysGetOpenGLProcedureAddresses+170>  lea    0x74a0(%rip),%rdi        # 0x2aaab67c30d1
   0x2aaab67bbc31 <GpuSysGetOpenGLProcedureAddresses+177>  mov    0x20a1c0(%rip),%rcx        # 0x2aaab69c5df8
   0x2aaab67bbc38 <GpuSysGetOpenGLProcedureAddresses+184>  mov    %rax,(%rcx)
   0x2aaab67bbc3b <GpuSysGetOpenGLProcedureAddresses+187>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc40 <GpuSysGetOpenGLProcedureAddresses+192>  lea    0x749c(%rip),%rdi        # 0x2aaab67c30e3
   0x2aaab67bbc47 <GpuSysGetOpenGLProcedureAddresses+199>  mov    0x20a21a(%rip),%rcx        # 0x2aaab69c5e68
   0x2aaab67bbc4e <GpuSysGetOpenGLProcedureAddresses+206>  mov    %rax,(%rcx)
   0x2aaab67bbc51 <GpuSysGetOpenGLProcedureAddresses+209>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc56 <GpuSysGetOpenGLProcedureAddresses+214>  lea    0x749d(%rip),%rdi        # 0x2aaab67c30fa
   0x2aaab67bbc5d <GpuSysGetOpenGLProcedureAddresses+221>  mov    0x20a35c(%rip),%rcx        # 0x2aaab69c5fc0
 > 0x2aaab67bbc64 <GpuSysGetOpenGLProcedureAddresses+228>  mov    %rax,(%rcx)
   0x2aaab67bbc67 <GpuSysGetOpenGLProcedureAddresses+231>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc6c <GpuSysGetOpenGLProcedureAddresses+236>  lea    0x7497(%rip),%rdi        # 0x2aaab67c310a
   0x2aaab67bbc73 <GpuSysGetOpenGLProcedureAddresses+243>  mov    0x20a1be(%rip),%rcx        # 0x2aaab69c5e38
   0x2aaab67bbc7a <GpuSysGetOpenGLProcedureAddresses+250>  mov    %rax,(%rcx)
   0x2aaab67bbc7d <GpuSysGetOpenGLProcedureAddresses+253>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc82 <GpuSysGetOpenGLProcedureAddresses+258>  lea    0x7494(%rip),%rdi        # 0x2aaab67c311d
   0x2aaab67bbc89 <GpuSysGetOpenGLProcedureAddresses+265>  mov    0x20a358(%rip),%rcx        # 0x2aaab69c5fe8
   0x2aaab67bbc90 <GpuSysGetOpenGLProcedureAddresses+272>  mov    %rax,(%rcx)
   0x2aaab67bbc93 <GpuSysGetOpenGLProcedureAddresses+275>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbc98 <GpuSysGetOpenGLProcedureAddresses+280>  lea    0x748c(%rip),%rdi        # 0x2aaab67c312b
   0x2aaab67bbc9f <GpuSysGetOpenGLProcedureAddresses+287>  mov    0x20a302(%rip),%rcx        # 0x2aaab69c5fa8
   0x2aaab67bbca6 <GpuSysGetOpenGLProcedureAddresses+294>  mov    %rax,(%rcx)
   0x2aaab67bbca9 <GpuSysGetOpenGLProcedureAddresses+297>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
   0x2aaab67bbcae <GpuSysGetOpenGLProcedureAddresses+302>  lea    0x7486(%rip),%rdi        # 0x2aaab67c313b
   0x2aaab67bbcb5 <GpuSysGetOpenGLProcedureAddresses+309>  mov    0x20a214(%rip),%rcx        # 0x2aaab69c5ed0
   0x2aaab67bbcbc <GpuSysGetOpenGLProcedureAddresses+316>  mov    %rax,(%rcx)
   0x2aaab67bbcbf <GpuSysGetOpenGLProcedureAddresses+319>  callq  0x2aaab67b4950 <glXGetProcAddressARB@plt>
multi-thre Thread 0x2aaaaab06b In: GpuSysGetOpenGLProcedureAddresses                          L307  PC: 0x2aaab67bbc64
0x00002aaaac844bac in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac844bad in pthread_rwlock_unlock () from /usr/lib/libpthread.so.0
0x00002aaaac355c6b in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c6f in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c72 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c74 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c78 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7b in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7c in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7d in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c7f in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c81 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c83 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaaac355c85 in glXGetProcAddress () from /usr/lib/libGLX.so.0
0x00002aaab67bbc56 in GpuSysGetOpenGLProcedureAddresses () at ./gpulib.h:307

Thread 1 "main" received signal SIGSEGV, Segmentation fault.
0x00002aaab67bbc64 in GpuSysGetOpenGLProcedureAddresses () at ./gpulib.h:307
(gdb) 
```

It executed glXGetProcAddress several times before that point, what's the problem on this particular line? I don't know. No one knows.

This forces me to ban any other distro that can't execute code that works just fine on Ubuntu 16.04 and Mesa 17.0.7 until a person who understands the inner workings of graphics drivers will explain this random behavior. At no point I, as a software developer, should care about any of this given the same binary runs perfectly fine under another environment.

lib0 engine requirements are updated to exclude any other distro except Ubuntu 16.04 which so far at least works reliably.

mmozeiko
Mārtiņš
Možeiko

Arch Linux is not supported due to confirmed Mesa bug. #13693

That's not what "confirmed bug" means. Confirmed means the reason is clear and it can be explained why particular place crashes.

I am 100% sure this is not a bug in mesa, but bug in your code. Disassembly clearly shows that.

1802 posts / 1 project

First of all - have you tried running code without any hacks? libc replacements/_start/syscalls? Because bug can be in one of those - it messes up something that compiler does not expect (for example, stack alignment), and thus you see crash later. And it was not happening before because you were lucky, the compiler produced code that did not trigger the bug.

Crash doesn't crash "without reason". There is a reason. The instruction you highlighted can crash if %rcx register contains invalid address. Check the address in debugger and why it is wrong.

I could not compile your code on my ArchLinux:

1) I needed to add "-fno-stack-protector" to compiler arguments in build_main.sh. Otherwise it failed with missing __stack_chk_fail symbol.

2) I needed to change clang to gcc in build_lib0.sh. Clang takes forever to compile, not sure if it finishes at all. Waited like for 1 minute. gcc finishes instantly.

3) I needed to change order of scripts in build.sh - build_texteditor.sh should come before build_lib0.sh

After I could run ./main. Application launched. No crashes (on ArchLinux with mesa 17.2.6). But I don't see anything in window. Its completely empty and not refreshing.

Btw your stdlib_gettimeofday implementation is super inefficient. Real gettimeofday implementation doesn't do syscalls (which are very slow). It simply reads shared memory region which is super fast operation: https://blog.packagecloud.io/eng/...ystem-calls/#virtual-system-calls

mmozeiko
Mārtiņš Možeiko

Arch Linux is not supported due to confirmed Mesa bug. #13695

Ok, I realized I need to put all the files in /tmp/lib0_engine. Then it runs. And then I see the crash.

The bug is in your code. I can explain why. The problem is that you are linking to libGL.so. This means there will be glAttachShader and other global symbols loaded into your process. These symbols are in read-only section. That's why following line will crash:

```
glAttachShader = (void *)glXGetProcAddressARB((unsigned char *)"glAttachShader");
```

1802 posts / 1 project

Because it will try to write to read-only location (that's why I asked to check %rcx value above, you would see that it belongs to libGL.so memory map).

Why this does not happen in main.c? ./main is not relocatable that's why it will use local glAttachShader functions (why? that's a long topic, read up how dynamic linking is implemented in Linux).

The fix is to not name global symbols (functions or variables) with same name as imported symbols from shared libraries. Name them differently if you want to keep them global. This is what glew, glad or others are doing for GL functions - renaming them to not conflict with original GL symbols.

In 9999 cases of 10000 the bug is in your code, not in libraries/compiler/hardware.

Procedural

36 posts / 1 project

Arch Linux is not supported due to confirmed Mesa bug. #13697

> I can explain why. The problem is that you are linking to libGL.so.

...

THANK YOU!

Dependency on libGL is removed!

Any non-deterministic garbage that alters behavior of my program will be annihilated out of my code. Although at first I simply tried to dlload libGL.so, but turns out even this way global gl* procedures get defined in read only locations somehow, because I kept getting crashes even though a binary wasn't linked to libGL.so directly anymore. So I decided to encapsulate all the procedures I use in a struct, like I did for Dear ImGui state already. This way they do not change somehow still globally located alien procedures from outer space.

> Btw your stdlib_gettimeofday implementation is super inefficient. Real gettimeofday implementation doesn't do syscalls (which are very slow). It simply reads shared memory

> region which is super fast operation: https://blog.packagecloud.io/eng/...ystem-calls/#virtual-system-calls

Yeah, I know, stdlib is a work of 20 minutes, I just copied and pasted gettimeofday from musl, they use syscalls, I didn't bother to change it because I just wanted to get rid of garbage GNU glibc. At first I wanted to link against static musl library, but it turns out you can't mix glibc and musl, they (hey, maybe because of global stuff too) interfere with each other and X11 calls just crash. The official repsonse of musl developers on forums is "build your whole distro with musl only".

> Why this does not happen in main.c? ./main is not relocatable that's why it will use local glAttachShader functions (why? that's a long topic, read up how dynamic linking is implemented in Linux).

And why this crash does not happen on Ubuntu 16.04?

mmozeiko
Mārtiņš Možeiko

Arch Linux is not supported due to confirmed Mesa bug. #13698

Yeah, shared libraries work differently on Linux than Windows. Symbol table & dynamic symbols are more tricker on Linux. On Windows dll files are pretty isolated.

I guess on Ubuntu they build libraries differently. Different linker hardening flags? Import relocations (or whatever they are called) are not read-only and you are overwriting them? Maybe.

1802 posts / 1 project