Developing

Tilck

A Tiny Linux-Compatible Kernel

Vladislav K. Valtchev (2022)

What Tilck is

- A project consisting on:
  - A monolithic kernel written in C and assembly
  - A bootloader working both on UEFI and legacy BIOS systems
  - Several test suites and a powerful CMake-based build system
  - Buildroot-like scripts for downloading & building 3rd party software
- Partially compatible with Linux at binary level
- Uniprocessor, but fully preemptable
- Educational, with potential to be more than that (see testing etc.)
- Runs only on i686, at the moment (will be ported to ARM, RISC-V etc.)
- Open source, distributed under the BSD 2-clause license

What Tilck is **NOT**

- An attempt to replace Linux
- An attempt to be yet another desktop operating system
- An attempt to be a large-scale server operating system
- A real-time OS, but it might become one in the future
- A OS running on NOMMU machines, but (probably) will in the future
- Ready for production use: it still lacks features as storage, networking etc.

Why the binary compatibility with Linux?

- It’s cool being able to test the same “bits” both on Linux and Tilck
- Robustness: Tilck can empirically show robustness and correctness by running 3rd party software never written for it
- Didn’t want to design a whole new syscall interface from scratch
- Didn’t want to implement a whole libc too
- Didn’t want to build a custom GCC toolchain. I wanted to use the pre-built toolchains from: https://toolchains.bootlin.com/
- Increase the likelihood the project to get more interest from the community?
- Porting pre-existing software to Tilck will require a little or no effort at all.

Core values & goals

- Minimal memory footprint
- Ultra low-latency
- Deterministic behavior
- Extra robustness
- Portability
- Simplicity
- Partial compatibility with Linux
- Must work on real (modern) hardware
- Exceptional developer experience: building & testing the project should be as easy as technologically possible

Live demo

Because a demo is worth more than a thousand words

Funny stories & interesting challenges

Vladislav K. Valtchev (2022)

My latest bug [1/6]

(and its 2-char fix)

- I have a test (fork_oom) that:
  1. Estimates the amount of *committed* memory that can be used
  2. Allocates and commits more than half of that
  3. Calls fork()
  4. In the child, tries to commit *all* of that memory
  5. Expects the child to be killed by the kernel

My latest bug [1/6]

(and its 2-char fix)

- I have a test (fork_oom) that:
  1. Estimates the amount of *committed* memory that can be used
  2. Allocates and commits more than half of that
  3. Calls fork()
  4. In the child, tries to commit *all* of that memory
  5. Expects the child to be killed by the kernel
- I just found that it fails on real HW machines

My latest bug [1/6]

(and its 2-char fix)

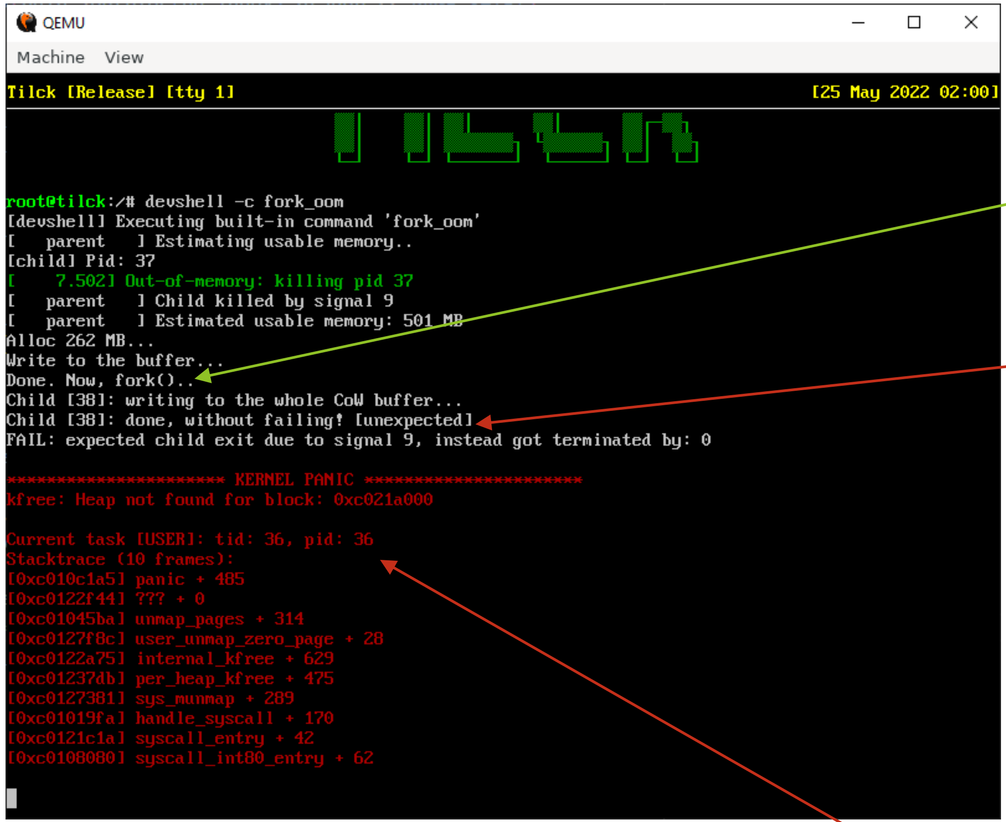
- I have a test (fork_oom) that:
  1. Estimates the amount of *committed* memory that can be used
  2. Allocates and commits more than half of that
  3. Calls fork()
  4. In the child, tries to commit *all* of that memory
  5. Expects the child to be killed by the kernel
- I just found that it fails on real HW machines
- Quickly, I discovered that it fails on VMs too but only when they have significantly more RAM. That’s weird. Mmm…

My latest bug [1/6]

(and its 2-char fix)

- I have a test (fork_oom) that:
    1. Estimates the amount of *committed* memory that can be used
    2. Allocates and commits more than half of that
    3. Calls fork()
    4. In the child, tries to commit *all* of that memory
    5. Expects the child to be killed by the kernel
- I just found that it fails on real HW machines
- Quickly, I discovered that it fails on VMs too but only when they have significantly more RAM. That’s weird. Mmm…
- I had to debug that.

My latest bug [2/6]



```
Tilck [Release] [tty 1]                                       [25 May 2022 02:00]

root@tilck:/# devshell -c fork_oom
[devshell] Executing built-in command 'fork_oom'
[   parent   ] Estimating usable memory..
[child] Pid: 37
[    7.502] Out-of-memory: killing pid 37
[   parent   ] Child killed by signal 9
[   parent   ] Estimated usable memory: 501 MB
Alloc 262 MB...
Write to the buffer...
Done. Now, fork()..
Child [38]: writing to the whole CoW buffer...
Child [38]: done, without failing! [unexpected]
FAIL: expected child exit due to signal 9, instead got terminated by: 0

********************** KERNEL PANIC **********************
kfree: Heap not found for block: 0xc021a000

Current task [USER]: tid: 36, pid: 36
Stacktrace (10 frames):
[0xc010c1a5] panic + 485
[0xc0122f44] ??? + 0
[0xc01045ba] unmap_pages + 314
[0xc0127f8c] user_unmap_zero_page + 28
[0xc0122a75] internal_kfree + 629
[0xc01237db] per_heap_kfree + 475
[0xc0127381] sys_munmap + 289
[0xc01019fa] handle_syscall + 170
[0xc0121c1a] syscall_entry + 42
[0xc0108080] syscall_int80_entry + 62
```

That's fine…

How could we commit so much memory?
262 MB x 2 = 524 MB > 501 MB [usable]
(ehm, we don't have swap)

That means trying to free a page not allocated in the heap, during munmap().

My latest bug [3/6]

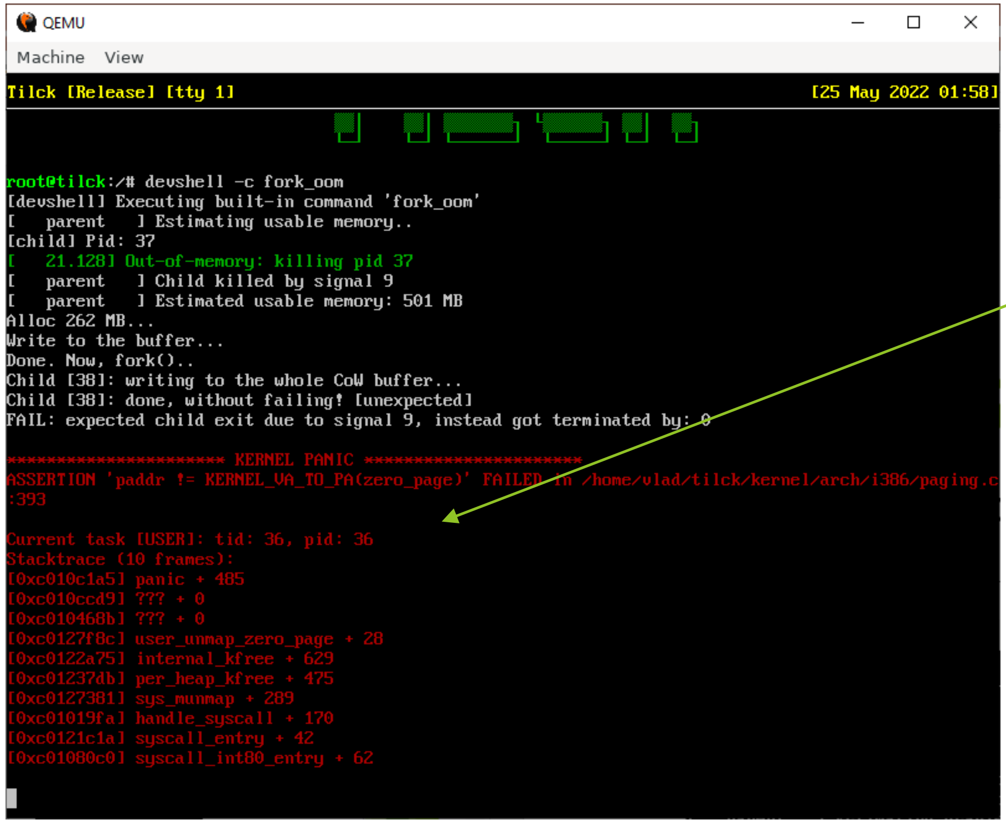
So, I started debugging the CoW page-fault logic…

```c
bool handle_potential_cow(void *context)
{
   /* ... */

   if (!(pt->pages[pt_index].avail & PAGE_COW_ORIG_RW))
      return false; /* Not a COW page */

   const u32 orig_page_paddr = (u32)
      pt->pages[pt_index].pageAddr << PAGE_SHIFT;

   if (pf_ref_count_get(orig_page_paddr) == 1) {

      /* This page is not shared anymore. No need for copying it. */
      pt->pages[pt_index].rw = true;
      pt->pages[pt_index].avail = 0;
      invalidate_page_hw(vaddr);
      return true;
   }

   /* ... */
}
```

After committing a few MBs in the child, we end up here!

My latest bug [4/6]

I realized I had ASSERTs disabled in that build! So, after turning them on…

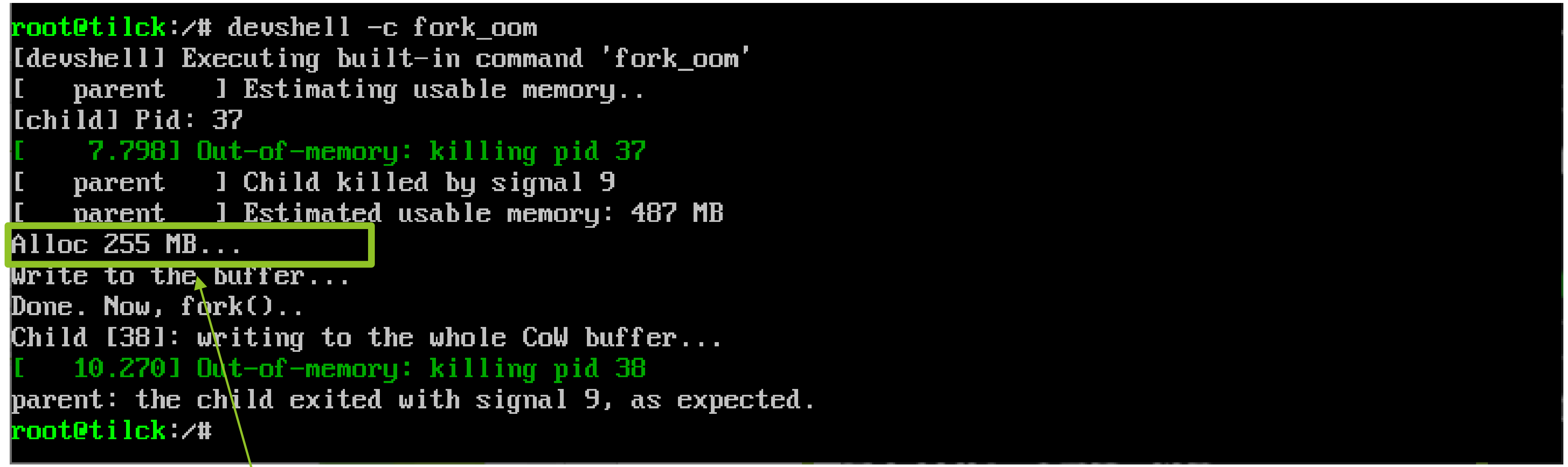


```
Tilck [Release] [tty 1]                                        [25 May 2022 01:58]

root@tilck:/# devshell -c fork_oom
[devshell] Executing built-in command 'fork_oom'
[    parent    ] Estimating usable memory..
[child] Pid: 37
[    21.128] Out-of-memory: killing pid 37
[    parent    ] Child killed by signal 9
[    parent    ] Estimated usable memory: 501 MB
Alloc 262 MB...
Write to the buffer...
Done. Now, fork()..
Child [38]: writing to the whole CoW buffer...
Child [38]: done, without failing! [unexpected]
FAIL: expected child exit due to signal 9, instead got terminated by: 0

********************** KERNEL PANIC **********************
ASSERTION 'paddr != KERNEL_VA_TO_PA(zero_page)' FAILED in /home/vlad/tilck/kernel/arch/i386/paging.c
:393

Current task [USER]: tid: 36, pid: 36
Stacktrace (10 frames):
[0xc010c1a5] panic + 485
[0xc010ccd9] ??? + 0
[0xc010468b] ??? + 0
[0xc0127f8c] user_unmap_zero_page + 28
[0xc0122a75] internal_kfree + 629
[0xc01237db] per_heap_kfree + 475
[0xc0127381] sys_munmap + 289
[0xc01019fa] handle_syscall + 170
[0xc0121c1a] syscall_entry + 42
[0xc01080c0] syscall_int80_entry + 62
```

Aha, gotcha! You're really trying to free the **zero page**!

My latest bug [5/6]

Let’s look at this limit case…



```
root@tilck:/# devshell -c fork_oom
[devshell] Executing built-in command 'fork_oom'
[   parent   ] Estimating usable memory..
[child] Pid: 37
[    7.798] Out-of-memory: killing pid 37
[   parent   ] Child killed by signal 9
[   parent   ] Estimated usable memory: 487 MB
Alloc 255 MB...
Write to the buffer...
Done. Now, fork()..
Child [38]: writing to the whole CoW buffer...
[   10.270] Out-of-memory: killing pid 38
parent: the child exited with signal 9, as expected.
root@tilck:/#
```

Allocating 255 MB works…

My latest bug [6/6]

That means only one thing…

```
static u16 *pageframes_refcount;
static ulong phys_mem_lim;

static ALWAYS_INLINE u32 pf_ref_count_get(u32 paddr)
{
    if (UNLIKELY(paddr >= phys_mem_lim))
        return 0;

    return pageframes_refcount[paddr >> PAGE_SHIFT];
}
```

My latest bug [6/6]

That means only one thing…

That's the problem: a 16-bit ref-count

```c
static u16 *pageframes_refcount;
static ulong phys_mem_lim;

static ALWAYS_INLINE u32 pf_ref_count_get(u32 paddr)
{
    if (UNLIKELY(paddr >= phys_mem_lim))
        return 0;

    return pageframes_refcount[paddr >> PAGE_SHIFT];
}
```

My latest bug [6/6]

That means only one thing…

That's the problem: a 16-bit ref-count

```
static u16 *pageframes_refcount;
static ulong phys_mem_lim;

static ALWAYS_INLINE u32 pf_ref_count_get(u32 paddr)
{
    if (UNLIKELY(paddr >= phys_mem_lim))
        return 0;

    return pageframes_refcount[paddr >> PAGE_SHIFT];
}
```

It wraps around after 65,535 pages, meaning that the kernel cannot support 256 MB or more of *uncommited* memory!

Making the framebuffer console *fast*

Vladislav K. Valtchev (2022)

Making the framebuffer console *fast*

- Premise: why implement a framebuffer console?
  - Text mode was almost completely dead even 5 years ago
  - Pure-UEFI machines don’t support text mode
  - Text mode is a x86 thing: Raspberry PI and other machines don’t support it

Making the framebuffer console *fast*

- Premise: why implement a framebuffer console?
  - Text mode was almost completely dead even 5 years ago
  - Pure-UEFI machines don’t support text mode
  - Text mode is a x86 thing: Raspberry PI and other machines don’t support it
- Why speed matters so much? Just mark the pages as WC and it will be reasonably fast.

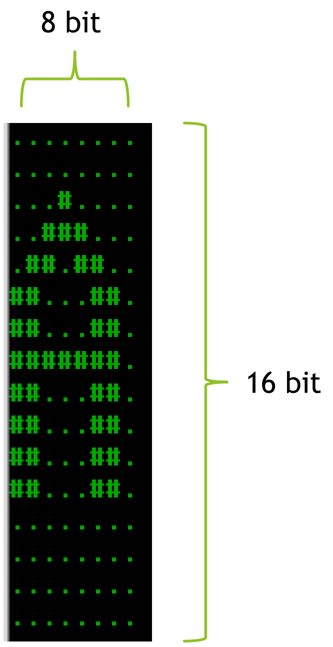
Making the framebuffer console *fast*

- Premise: why implement a framebuffer console?
  - Text mode was almost completely dead even 5 years ago
  - Pure-UEFI machines don't support text mode
  - Text mode is a x86 thing: Raspberry PI and other machines don't support it
- Why speed matters so much? Just mark the pages as WC and it will be reasonably fast.
  - I didn't know about WC (write-combining) at the time

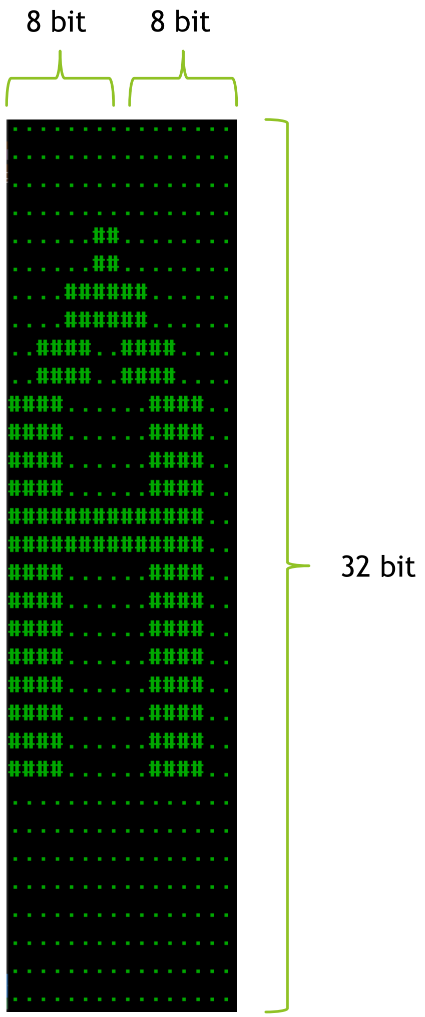
Making the framebuffer console *fast*

- Premise: why implement a framebuffer console?
  - Text mode was almost completely dead even 5 years ago
  - Pure-UEFI machines don't support text mode
  - Text mode is a x86 thing: Raspberry PI and other machines don't support it
- Why speed matters so much? Just mark the pages as WC and it will be reasonably fast.
  - I didn't know about WC (write-combining) at the time
  - Therefore, I implemented a series of optimizations before discovering WC

PSF fonts: a bitfield per each glyph



```
........
........
...#....
..###...
.##.##..
##...##.
##...##.
#######.
##...##.
##...##.
##...##.
##...##.
........
........
........
........
```



```
................
................
................
................
......##........
......##........
....######......
....######......
..####..####....
..####..####....
####......####..
####......####..
####......####..
####......####..
##############..
##############..
####......####..
####......####..
####......####..
####......####..
####......####..
####......####..
####......####..
####......####..
................
................
................
................
................
................
................
................
```

The simplest draw function (failsafe)

```c
static inline void fb_draw_pixel(u32 x, u32 y, u32 color)
{
   if (fb_bpp == 32)
      *(volatile u32 *) (fb_vaddr + (fb_pitch * y) + (x << 2)) = color;
   else
      // Assumption: bpp is 24
      memcpy((void *) (fb_vaddr + (fb_pitch * y) + (x * 3)), &color, 3);
}

void fb_draw_char(u32 x, u32 y, u16 e)
{
   u8 *data = font_glyph_data + font_bytes_per_glyph * vgaentry_get_char(e);
   u32 arr[] = { vga_rgb_colors[vgaentry_get_fg(e)], vga_rgb_colors[vgaentry_get_bg(e)] };

   for (u32 row = y; row < (y + font_h); row++, data += font_width_bytes) {
      for (u32 b = 0; b < font_width_bytes; b++) {
         for (u32 i = 0; i < 8; i++)
            fb_draw_pixel(x + (b << 3) + (8 - i - 1),   /* x */
                          row,                          /* y */
                          arr[!(data[b] & (1 << i))]);  /* color */
      }
   }
}
```

Performance? Too slow, in particular on the modern machine (left)

16x8 font, 800x600

Intel Core i7-7500U Kaby Lake

- 1,124,773 RDTSC cycles / char (avg.) [~385.7 μs]

Intel Atom N270 Diamondville (32-bit)

- 297,287 RDTSC cycles / char (avg.) [~186.3 μs]

32x16 font, 3200x1800

- 7,416,012 RDTSC cycles / char (avg.) [~2543.2 μs]

Scrolling the whole screen takes several seconds!!

A naïve optimization: loop unrolling

```c
#define draw_char_partial(b)                                                \
   do {                                                                     \
      fb_draw_pixel(x + (b << 3) + 7, row, arr[!(data[b] & (1 << 0))]);    \
      fb_draw_pixel(x + (b << 3) + 6, row, arr[!(data[b] & (1 << 1))]);    \
      fb_draw_pixel(x + (b << 3) + 5, row, arr[!(data[b] & (1 << 2))]);    \
      fb_draw_pixel(x + (b << 3) + 4, row, arr[!(data[b] & (1 << 3))]);    \
      fb_draw_pixel(x + (b << 3) + 3, row, arr[!(data[b] & (1 << 4))]);    \
      fb_draw_pixel(x + (b << 3) + 2, row, arr[!(data[b] & (1 << 5))]);    \
      fb_draw_pixel(x + (b << 3) + 1, row, arr[!(data[b] & (1 << 6))]);    \
      fb_draw_pixel(x + (b << 3) + 0, row, arr[!(data[b] & (1 << 7))]);    \
   } while (0)

void fb_draw_char(u32 x, u32 y, u16 e)
{
   u8 *data = font_glyph_data + font_bytes_per_glyph * vgaentry_get_char(e);
   u32 arr[] = { vga_rgb_colors[vgaentry_get_fg(e)], vga_rgb_colors[vgaentry_get_bg(e)] };

   if (LIKELY(font_width_bytes == 1))
      for (u32 row = y; row < (y+font_h); row++, data += font_width_bytes)
         draw_char_partial(0);
   else if (font_width_bytes == 2)
      for (u32 row = y; row < (y+font_h); row++, data += font_width_bytes) {
         draw_char_partial(0);
         draw_char_partial(1);
      }
   else
      for (u32 row = y; row < (y+font_h); row++, data += font_width_bytes)
         for (u32 b = 0; b < font_width_bytes; b++)
            draw_char_partial(b);
}
```

Benefits? Nah.

Intel Core i7-7500U Kaby Lake

Before (avg.)	385.72 µs / char
After (avg.)	384.44 µs / char
Speed up	0.3% faster

Intel Atom N270 Diamondville (32-bit)

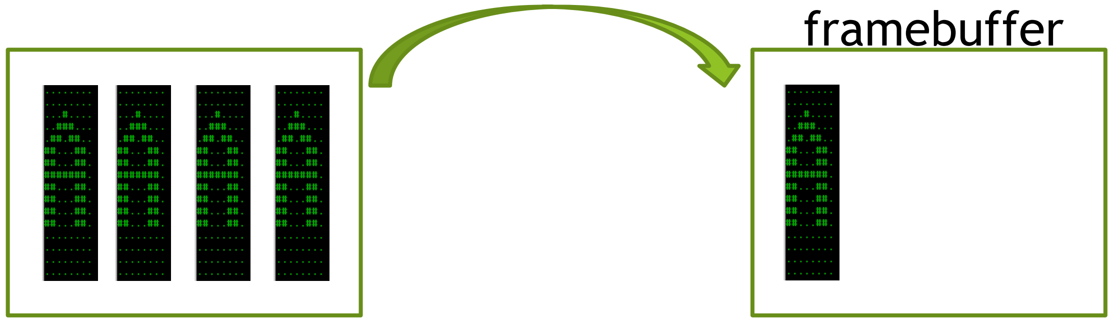
Before (avg.)	186.27 µs / char
After (avg.)	175.30 µs / char
Speed up	6.2% faster

Old school optimizations work better on old school machines!

Intuition 1: rendering glyphs pixel by pixel is too slow

Solution 1: pre-rendering!

- But… is pre-rendering *every* glyph in the font even feasible?



Pre-rendering! (font 16x8)

$$\underbrace{16 \text{ x } 8}_{\text{Height x Width}} \text{ x } \underbrace{4}_{\text{Bytes per pixel}} \text{ x } \underbrace{256}_{\text{\# glyphs}} \text{ x } \underbrace{16}_{\text{FG colors}} \text{ x } \underbrace{16}_{\text{BG colors}} =$$

32 MB: unfeasible!

Pre-rendering! (font 32x16)

$$\underbrace{32 \times 16}_{\text{Height x Width}} \times \underbrace{4}_{\text{Bytes per pixel}} \times \underbrace{256}_{\text{\# glyphs}} \times \underbrace{16}_{\text{FG colors}} \times \underbrace{16}_{\text{BG colors}} =$$

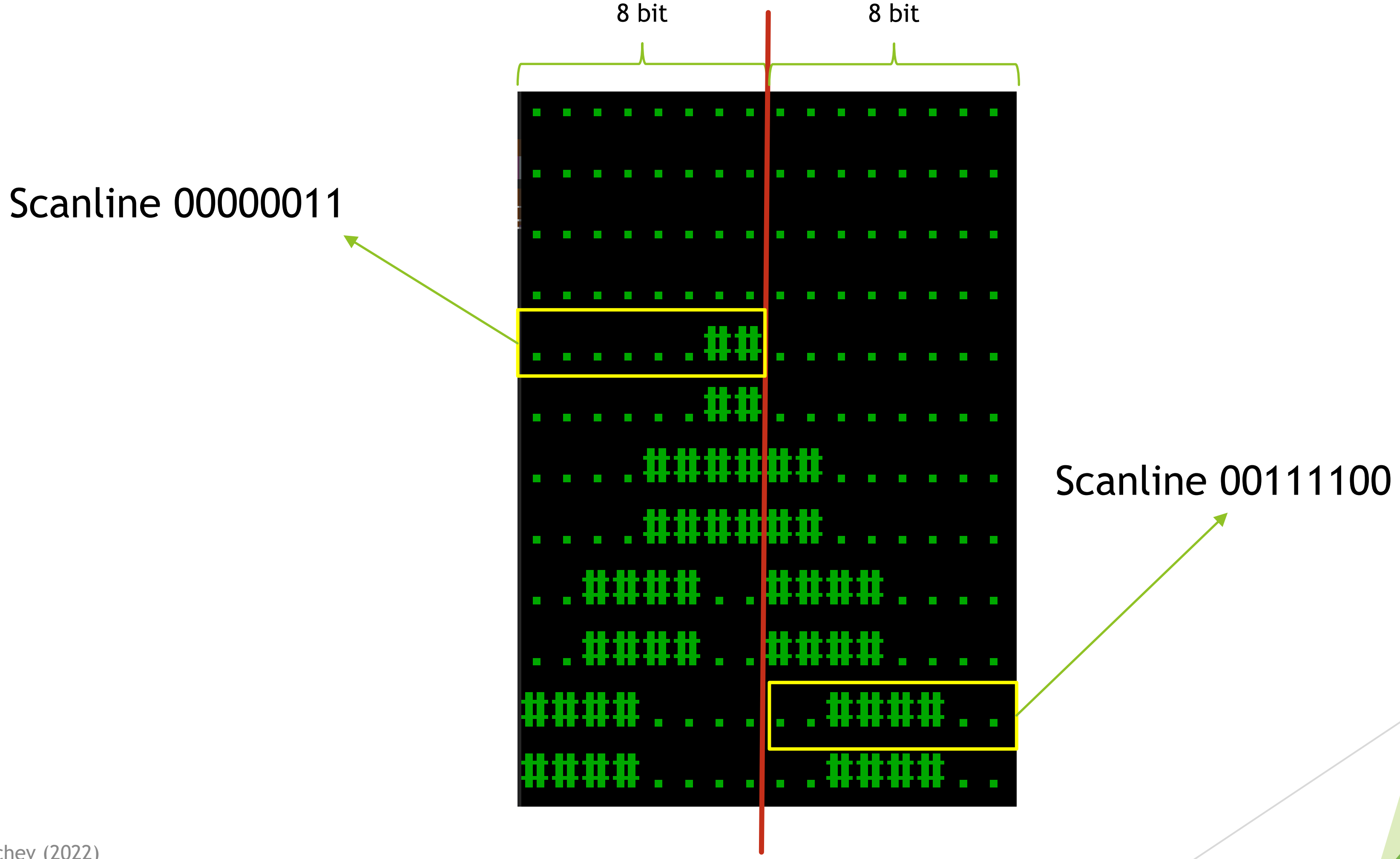
128 MB: pure madness!

A better idea: pre-render all the possible 8-bit “scanlines” (= glyph rows)

$$\underbrace{2^8}_{\text{All scanlines}} \times \underbrace{4}_{\text{Bytes per pixel}} \times \underbrace{8}_{\text{Scanline length}} \times \underbrace{16}_{\text{FG colors}} \times \underbrace{16}_{\text{BG colors}} = 2\ \text{MB}$$

Still expensive, but affordable!

It works on 32x16 fonts too!



```
................
................
................
................
......##........
......##........
....######......
....######......
..####..####....
..####..####....
####......####..
####......####..
```

The pre-render code

```c
#define PSZ          4     /* pixel size = 32 bpp / 8 = 4 bytes */
#define SL_COUNT   256     /* all possible 8-pixel scanlines */
#define SL_SIZE      8     /* scanline size: 8 pixels */
#define FG_COLORS   16     /* #fg colors */
#define BG_COLORS   16     /* #bg colors */
#define TOT_CHAR_SCANLINES_SIZE (PSZ*SL_COUNT*FG_COLORS*BG_COLORS*SL_SIZE)

bool fb_pre_render_char_scanlines(void)
{
   fb_w8_char_scanlines = kmalloc(TOT_CHAR_SCANLINES_SIZE);

   if (!fb_w8_char_scanlines)
      return false;

   for (u32 fg = 0; fg < FG_COLORS; fg++) {
      for (u32 bg = 0; bg < BG_COLORS; bg++) {
         for (u32 sl = 0; sl < SL_COUNT; sl++) {
            for (u32 pix = 0; pix < SL_SIZE; pix++) {
               fb_w8_char_scanlines[
                  fg * (BG_COLORS * SL_COUNT * SL_SIZE) +
                  bg * (SL_COUNT * SL_SIZE) +
                  sl * SL_SIZE +
                  (SL_SIZE - pix - 1)
               ] = (sl & (1 << pix)) ? vga_rgb_colors[fg] : vga_rgb_colors[bg];
            }
         }
      }
   }
   return true;
}
```

Intuition 2: copying 4 bytes at a time is too slow!

- Pre-rendering the glyphs or the just the “scanlines” is **not enough**
- The x86 **`rep movsl`** instruction copies just 4 bytes (= 1 pixel) at a time

Solution 2: use the FPU

- Introduce something like **fpu_memcpy()**
- Write a whole row at a time during scrolling
- Only this way, we could offset the cost of saving/restoring the FPU registers

```c
void fb_draw_row(u32 y, u16 *entries, u32 count, bool fpu)
{
   static const void *ops[] = {
      &&width_1_nofpu, &&width_1_fpu, &&width_2_nofpu, &&width_2_fpu
   };

   const u32 bpg_shift = 4 + (font_bytes_per_glyph == 64) * 2; // 4 or 6
   const u32 w4_shift  = 5 + (font_w == 16);                   // 5 or 6
   const void *const op = ops[(font_w == 16) * 2 + fpu];       // ops[0..3]
   const ulong vaddr_base = fb_vaddr + (fb_pitch * y);

   for (u32 ei = 0; ei < count; ei++) {

      const u16 e = entries[ei];
      const u32 c_off = (u32) ((vgaentry_get_fg(e) << 15) + (vgaentry_get_bg(e) << 11));
      void *vaddr = (void *)vaddr_base + (ei << w4_shift);
      const u8 *d = &font_glyph_data[vgaentry_get_char(e) << bpg_shift];
      u32 *scanlines = &fb_w8_char_scanlines[c_off];
      goto *op;

      width_1_fpu:
         for (u32 r = 0; r < font_h; r++, d++, vaddr += fb_pitch)
            fpu_cpy_single_256_nt(vaddr, &scanlines[d[0] << 3]);
         continue;

      width_1_nofpu:
         for (u32 r = 0; r < font_h; r++, d++, vaddr += fb_pitch)
            memcpy32(vaddr, &scanlines[d[0] << 3], SL_SIZE);
         continue;

      width_2_fpu:
         for (u32 r = 0; r < font_h; r++, d+=2, vaddr += fb_pitch) {
            fpu_cpy_single_256_nt(vaddr,      &scanlines[d[0] << 3]);
            fpu_cpy_single_256_nt(vaddr + 32, &scanlines[d[1] << 3]);
         }
         continue;

      width_2_nofpu:
         for (u32 r = 0; r < font_h; r++, d+=2, vaddr += fb_pitch) {
            memcpy32(vaddr,      &scanlines[d[0] << 3], SL_SIZE);
            memcpy32(vaddr + 32, &scanlines[d[1] << 3], SL_SIZE);
         }
         continue;
   }
}
```

Flag: during IRQ, we cannot use the FPU

Scanlines for the given FG/BG colors

Jump to the same address during the whole loop

Copy 256 bit (32 bytes) the fastest way possible

The FPU code [1/2]

```c
static void *get_fpu_cpy_single_256_nt_func(void)
{
    if (!kopt_no_fpu_memcpy) {

        if (x86_cpu_features.can_use_avx2)
            return &fpu_cpy_single_256_nt_avx2;

        if (x86_cpu_features.can_use_sse2)
            return &fpu_cpy_single_256_nt_sse2;

        if (x86_cpu_features.can_use_sse)
            return &fpu_cpy_single_256_nt_sse;
    }

    return NULL;
}

void init_fpu_memcpy(void)
{
    void *func;

    if ((func = get_fpu_cpy_single_256_nt_func())) {
        simple_hot_patch(&__asm_fpu_cpy_single_256_nt, func, 128);
    }
}
```

The FPU code [2/2]

```c
ALWAYS_INLINE FASTCALL void
fpu_cpy_single_256_nt_avx2(void *dest, const void *src)
{
    asmVolatile("vmovdqa   (%0), %%ymm0\n\t"
                "vmovntdq %%ymm0,   (%1)\n\t"
                : /* no output */
                : "r" (src), "r" (dest)
                : "memory");
}


ALWAYS_INLINE FASTCALL void
fpu_cpy_single_256_nt_sse2(void *dest, const void *src)
{
    asmVolatile("movdqa   (%0), %%xmm0\n\t"
                "movdqa 16(%0), %%xmm1\n\t"
                "movntdq %%xmm0,   (%1)\n\t"
                "movntdq %%xmm1, 16(%1)\n\t"
                : /* no output */
                : "r" (src), "r" (dest)
                : "memory");
}
```

```c
ALWAYS_INLINE FASTCALL void
fpu_cpy_single_256_nt_sse(void *dest, const void *src)
{
    asmVolatile("movq (%0), %%mm0\n\t"
                "movq 8(%0), %%mm1\n\t"
                "movq 16(%0), %%mm2\n\t"
                "movq 24(%0), %%mm3\n\t"
                "movntq %%mm0, (%1)\n\t"
                "movntq %%mm1, 8(%1)\n\t"
                "movntq %%mm2, 16(%1)\n\t"
                "movntq %%mm3, 24(%1)\n\t"
                : /* no output */
                : "r" (src), "r" (dest)
                : "memory");
}
```

The moment of truth

Font 16x8, resolution 800x600, default memory type, not WC*

** Typically that means UC (uncacheable) set through MTRRs*

Core i7-7500U Kaby Lake, AVX 2, 256-bit regs

Before (avg.)	385.72 µs / char
After (avg.)	67.42 µs / char
Speed up	5.72x faster

Not bad at all!

Atom N270 Diamondville (32-bit), SSSE 3, 128-bit regs

Before (avg.)	186.27 µs / char
After (avg.)	94.82 µs / char
Speed up	1.96x faster

Smaller impact, but smaller regs here

The moment of truth

Font 16x8, resolution 800x600, default memory type, not WC*

** Typically that means UC (uncacheable) set through MTRRs*

Core i7-7500U Kaby Lake, AVX 2, 256-bit regs

Before (avg.)	385.72 µs / char
After (avg.)	67.42 µs / char
Speed up	5.72x faster

Atom N270 Diamondville (32-bit), SSSE 3, 128-bit regs

Before (avg.)	186.27 µs / char
After (avg.)	94.82 µs / char
Speed up	1.96x faster

Font 32x16, resolution 3200x1800, default memory type, not WC*

Before (avg.)	2543.21 µs / char
After (avg.)	371.54 µs / char
Speed up	6.84x faster

Still not fast enough, though

Wow, that's close to the max 8x improvement!
(From 32 bit/write to 256 bit/write)

The writing combining memory type (WC)

- Allows data to combined, temporarily stored in a buffer (WCB) and then released in burst mode
- Cannot be used most of the time because offers just **weak ordering**
- Can be set using **PAT** or **MTRRs**
- It’s perfect for frame buffers

Performance: the full picture [modern machine]

Font 16x8, resolution 800x600, 32 bbp

Intel Core i7-7500U Kaby Lake (AVX 2, 256-bit fpu regs)

Mode	Opt	Wc	FPU	Cycles/char	usec/char
Failsafe slow				1,124,773	385.72
Failsafe opt				1,121,034	384.44
Opt + fpu	✓		✓	196,584	67.42
Opt (no fpu)	✓			1,177,902	403.94
Wc		✓		34,055	11.68
Opt + Wc (no fpu)	✓	✓		44,294	15.19
Opt + Wc + fpu	✓	✓	✓	30,271	10.38

32.9x faster!

Just 12.5% faster

Performance: the full picture [older machine]

Font 16x8, resolution 800x600, 32 bbp

Intel Atom N270 Diamondville (32-bit, SSSE 3, 128 bit fpu regs)

Mode	Opt	Wc	FPU	Cycles/char	usec/char
Failsafe slow				297,287	186.27
Failsafe opt				279,781	175.30
Opt + fpu	✓		✓	151,337	94.82
Opt (no fpu)	✓			279,453	175.10
Wc		✓		136,925	85.79
Opt + Wc (no fpu)	✓	✓		136,914	85.79
Opt + Wc + fpu	✓	✓	✓	136,906	85.78

2.04x faster

No difference at all!

Performance on native res [modern machine]

Font 32x16, resolution 3200x1800, 32 bbp

Intel Core i7-7500U Kaby Lake (AVX 2, 256-bit fpu regs)

Mode	Opt	Wc	FPU	Cycles/char	usec/char
Failsafe slow				7,416,012	2543.21
Failsafe opt				7,408,637	2540.68
Opt + fpu	✓		✓	1,083,408	371.54
Opt (no fpu)	✓			7,815,696	2680.28
Wc		✓		73,165	25.09
Opt + Wc (no fpu)	✓	✓		159,067	54.55
Opt + Wc + fpu	✓	✓	✓	27,841	9.55

6.84x faster

101.26x faster!

2.63x faster

Not bad!

Performance vs Linux [modern machine]

Font 32x16, resolution 3200x1800, 32 bbp

Commit a858f229, release build

Kernel 5.4.0 (Ubuntu 20.04.4 LTS)

Performance vs Linux [modern machine]

Font 32x16, resolution 3200x1800, 32 bbp

Tilck
A Tiny Linux-Compatible Kernel

Commit a858f229, release build

Kernel 5.4.0 (Ubuntu 20.04.4 LTS)

- 9.55 µs / char

Performance vs Linux [modern machine]

Font 32x16, resolution 3200x1800, 32 bbp

Commit a858f229, release build

- 9.55 µs / char

Kernel 5.4.0 (Ubuntu 20.04.4 LTS)

- 56.40 µs / char

Performance vs Linux [modern machine]

Font 32x16, resolution 3200x1800, 32 bbp

Tilck
A Tiny Linux-Compatible Kernel

Commit a858f229, release build

- 9.55 µs / char

Kernel 5.4.0 (Ubuntu 20.04 LTS)

- 56.40 µs / char

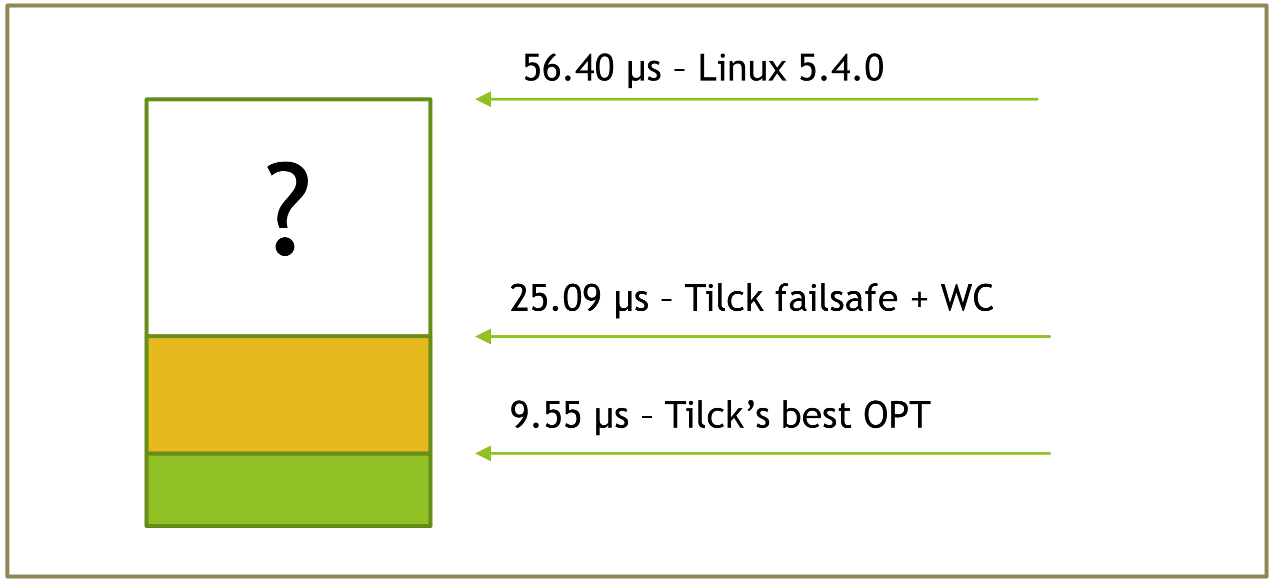
5.9x faster!

Performance vs Linux [modern machine]

Font 32x16, resolution 3200x1800, 32 bbp

Commit a858f229, release build

Kernel 5.4.0 (Ubuntu 20.04 LTS)



The benchmark code

```c
void console_perf_test(void)
{
   static const char letters[] =
      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      "abcdefghijklmnopqrstuvwxyz0123456789";

   int iters = 3;
   struct winsize w;
   char *buf, tot_time_s[32], c_time_s[32];
   ssize_t r, tot, written;
   struct timespec ts_before, ts_after;
   uint64_t start, end, c;
   double tot_time_real, tot_time, time_c, cycles_per_sec;

   if (ioctl(1, TIOCGWINSZ, &w) != 0) {
      perror("ioctl() failed");
      return;
   }

   tot = w.ws_row * w.ws_col;

   if (!(buf = malloc(tot))) {
      perror("malloc() failed\n");
      return;
   }

   for (int i = 0; i < tot; i++) {
      buf[i] = letters[i % (sizeof(letters) - 1)];
   }

   printf("%s", CSI_ERASE_DISPLAY CSI_MOVE_CURSOR_TOP_LEFT);

retry:
   clock_gettime(CLOCK_REALTIME, &ts_before);
   start = RDTSC();

   for (int i = 0; i < iters; i++) {
      for (r = 0, written = 0; written < tot; written += r) {

         if ((r = write(1, buf + written, tot - written)) < 0) {
            perror("write() failed");
            free(buf);
            return;
         }
      }
   }

   end = RDTSC();
   clock_gettime(CLOCK_REALTIME, &ts_after);

   c = (end - start) / iters;
   tot_time_real = timespec_diff(&ts_after, &ts_before);
   tot_time = tot_time_real / iters;
   time_c = tot_time / (double)tot;
   cycles_per_sec = (end - start) / tot_time_real;

   if (tot_time_real <= 0.1) {
      iters *= 10;
      goto retry;
   }

   timespec_to_human_str(tot_time_s, sizeof(tot_time_s), tot_time);
   timespec_to_human_str(c_time_s, sizeof(c_time_s), time_c);
   printf("Term size: %d rows x %d cols\n", w.ws_row, w.ws_col);
   printf("Tot iterations: %d\n\n", iters);
   printf("Screen redraw:        %12llu cycles (%s)\n", c, tot_time_s);
   printf("Avg. character cost: %12llu cycles (%s)\n", c / tot, c_time_s);
   printf("Cycles per sec:       %12.0f cycles/sec\n", cycles_per_sec);
   free(buf);
}
```

Making libmusl applications to work

Why libmusl?

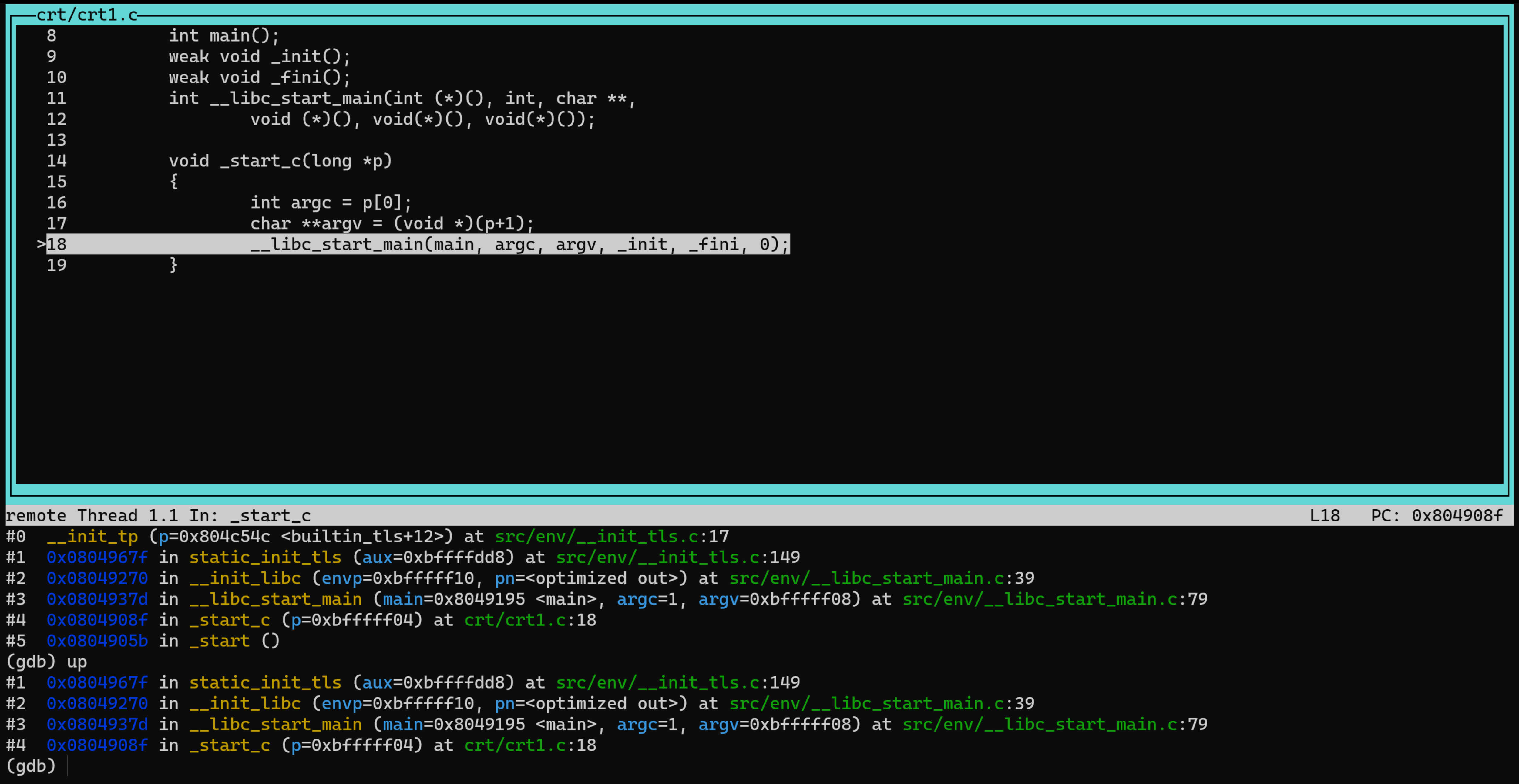
- It made no sense to write a custom libc.
- Libmusl produces the smallest binaries (~13 KB for “hello world”)
- It’s actively maintained and widely used in the Embedded Linux world
- It’s supported by https://toolchains.bootlin.com/
- **Uclibc-ng** is more customizable but:
  - Typically produces larger binaries
  - Using a pre-built toolchain means no customization anyway
- **Dietlibc** is not well-maintained and has no pre-built toolchains

Libmusl requires TLS support

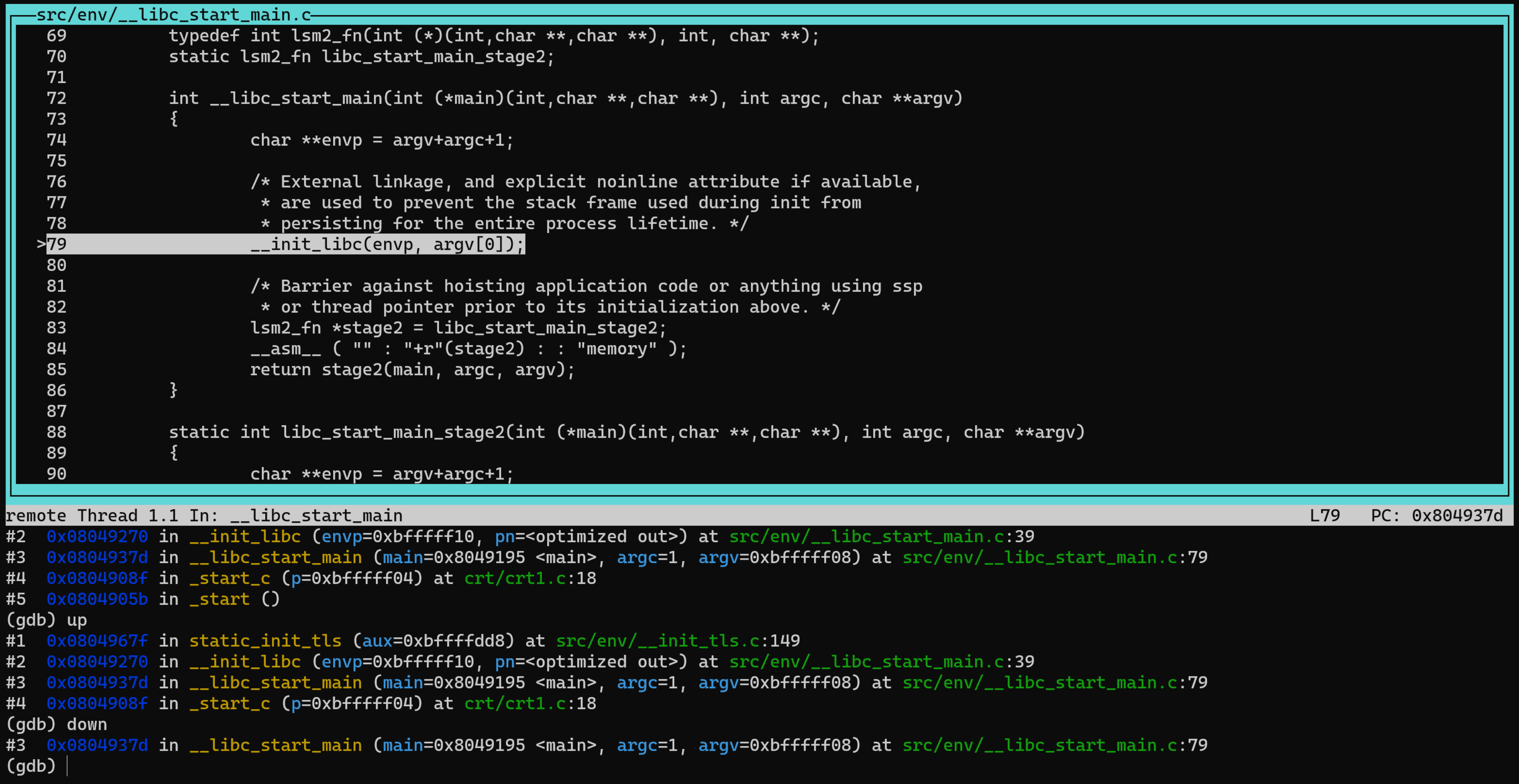
- TLS requires set_thread_area()

Libmusl requires TLS support

- TLS requires set_thread_area()
- Can we cheat by returning –ENOSYS ? ☺

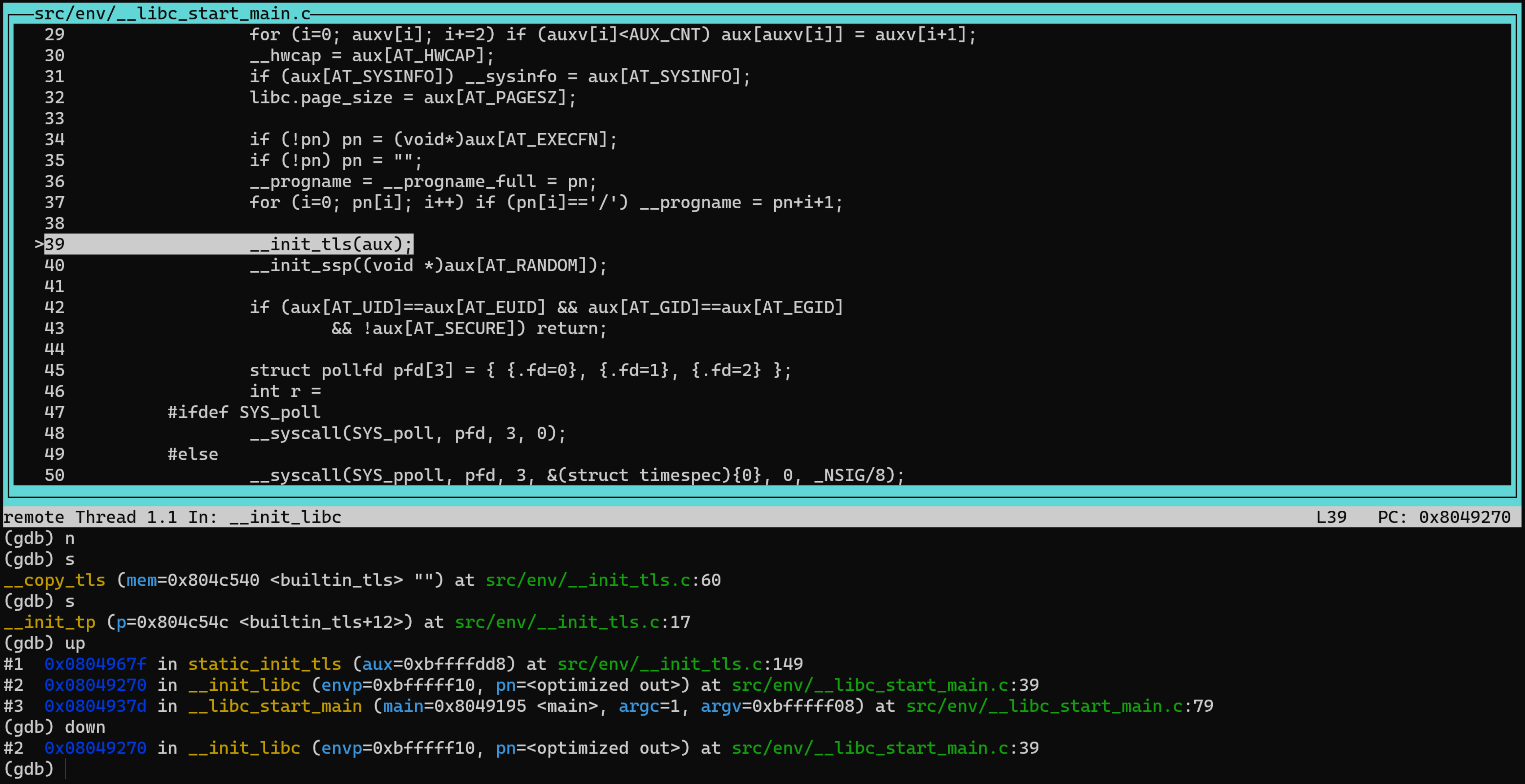


```
crt/crt1.c
   8            int main();
   9            weak void _init();
  10            weak void _fini();
  11            int __libc_start_main(int (*)(), int, char **,
  12                    void (*)(), void(*)(), void(*)());
  13
  14            void _start_c(long *p)
  15            {
  16                    int argc = p[0];
  17                    char **argv = (void *)(p+1);
 >18                    __libc_start_main(main, argc, argv, _init, _fini, 0);
  19            }

remote Thread 1.1 In: _start_c                                                                         L18   PC: 0x804908f
#0  __init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:17
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:79
#4  0x0804908f in _start_c (p=0xbfffff04) at crt/crt1.c:18
#5  0x0804905b in _start ()
(gdb) up
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:79
#4  0x0804908f in _start_c (p=0xbfffff04) at crt/crt1.c:18
(gdb)
```

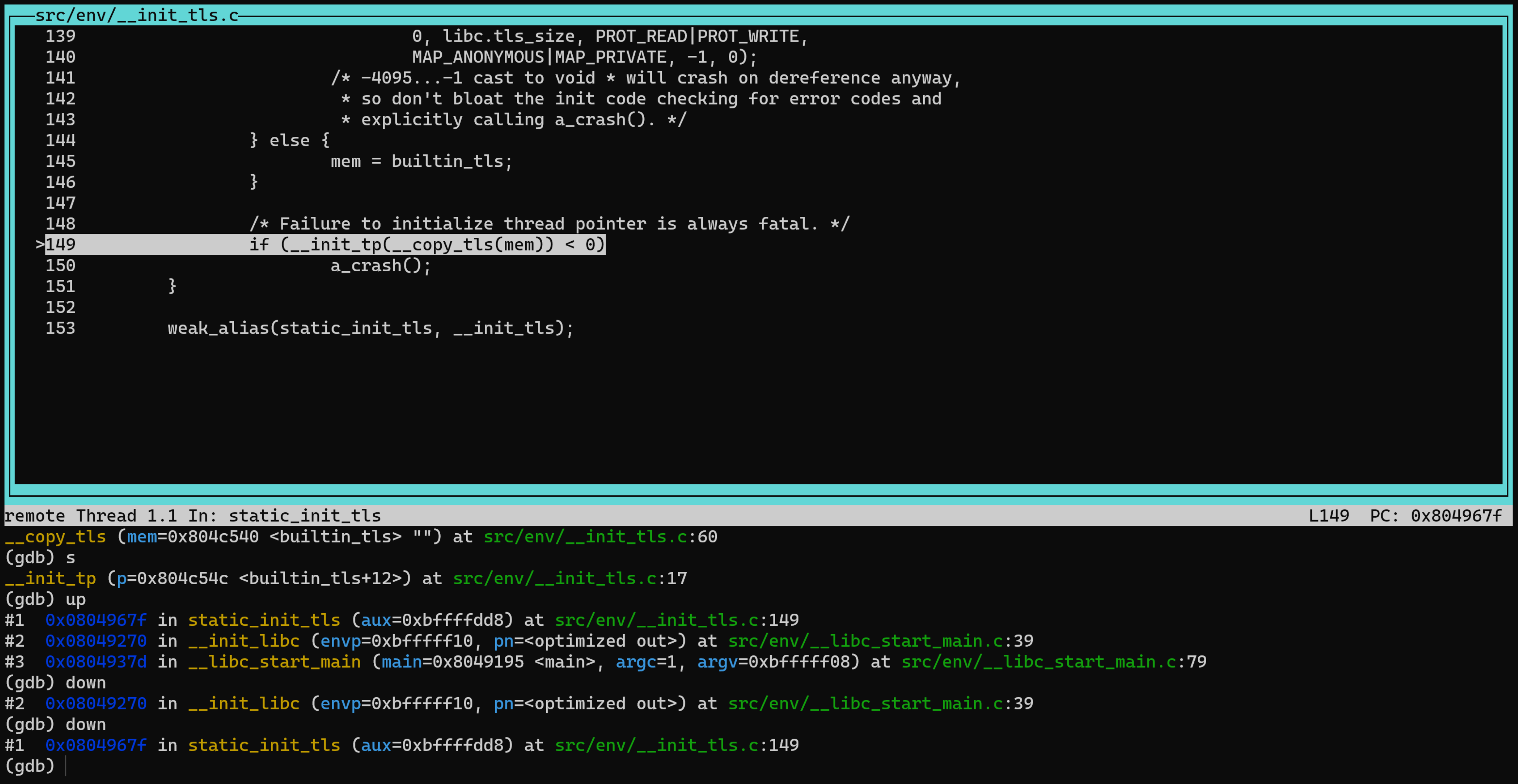


```
src/env/__libc_start_main.c
 69         typedef int lsm2_fn(int (*)(int,char **,char **), int, char **);
 70         static lsm2_fn libc_start_main_stage2;
 71
 72         int __libc_start_main(int (*main)(int,char **,char **), int argc, char **argv)
 73         {
 74                 char **envp = argv+argc+1;
 75
 76                 /* External linkage, and explicit noinline attribute if available,
 77                  * are used to prevent the stack frame used during init from
 78                  * persisting for the entire process lifetime. */
>79                 __init_libc(envp, argv[0]);
 80
 81                 /* Barrier against hoisting application code or anything using ssp
 82                  * or thread pointer prior to its initialization above. */
 83                 lsm2_fn *stage2 = libc_start_main_stage2;
 84                 __asm__ ( "" : "+r"(stage2) : : "memory" );
 85                 return stage2(main, argc, argv);
 86         }
 87
 88         static int libc_start_main_stage2(int (*main)(int,char **,char **), int argc, char **argv)
 89         {
 90                 char **envp = argv+argc+1;
```

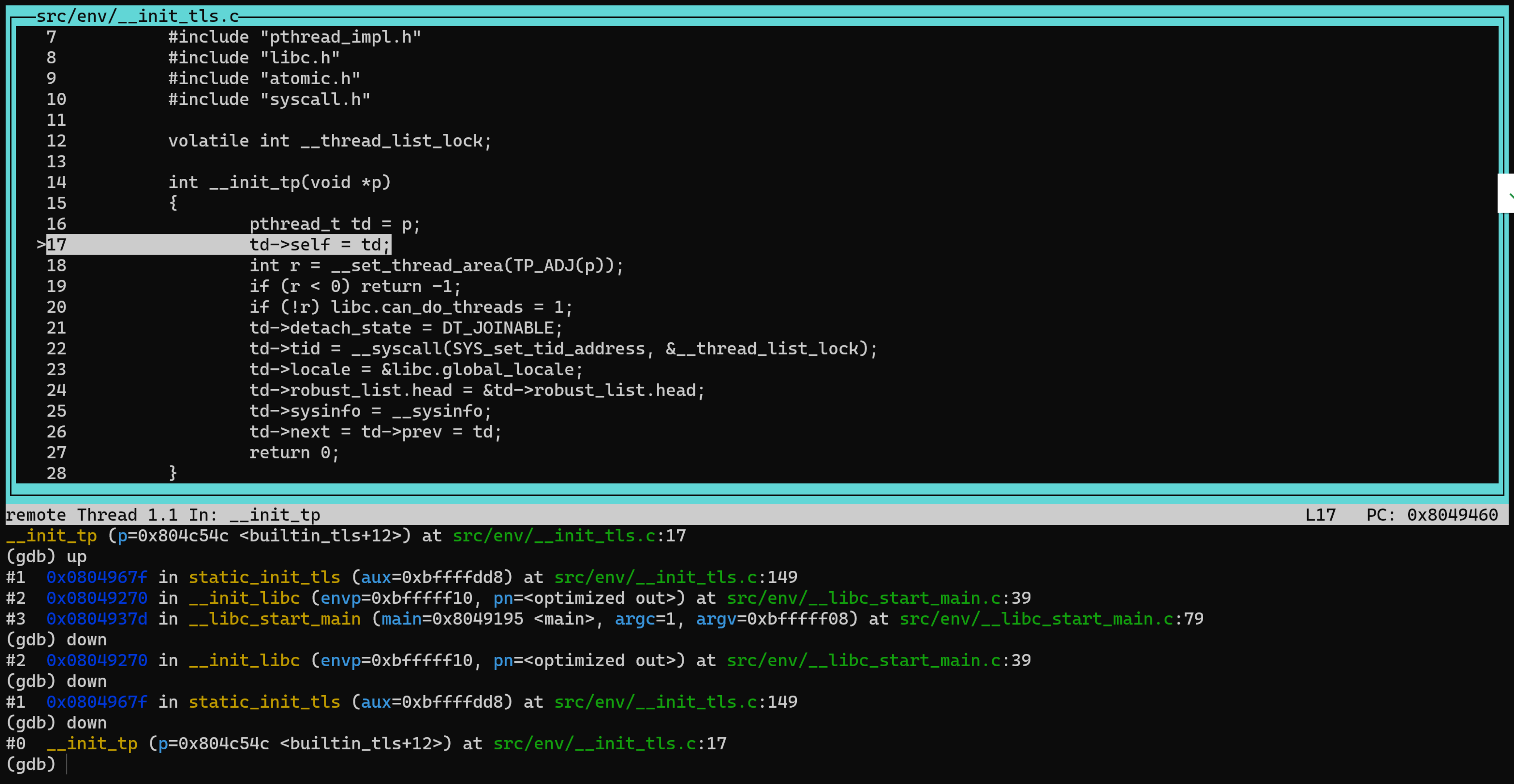
```
remote Thread 1.1 In: __libc_start_main                                                          L79   PC: 0x804937d
#2  0x08049270 in __init_libc (envp=0xbffff f10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbffff f08) at src/env/__libc_start_main.c:79
#4  0x0804908f in _start_c (p=0xbffff f04) at crt/crt1.c:18
#5  0x0804905b in _start ()
(gdb) up
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
#2  0x08049270 in __init_libc (envp=0xbffff f10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbffff f08) at src/env/__libc_start_main.c:79
#4  0x0804908f in _start_c (p=0xbffff f04) at crt/crt1.c:18
(gdb) down
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbffff f08) at src/env/__libc_start_main.c:79
(gdb)
```



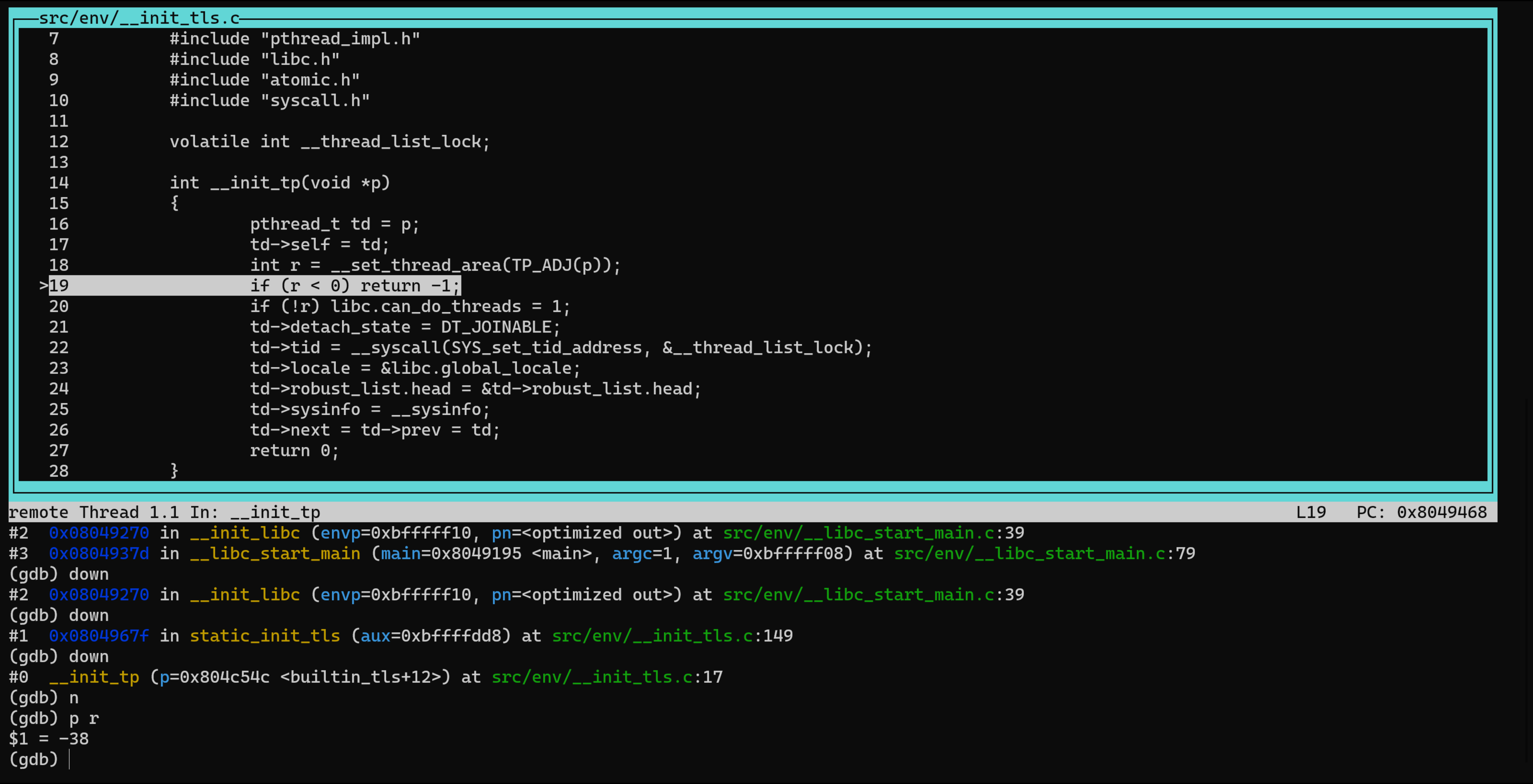
```
src/env/__libc_start_main.c
                for (i=0; auxv[i]; i+=2) if (auxv[i]<AUX_CNT) aux[auxv[i]] = auxv[i+1];
                __hwcap = aux[AT_HWCAP];
                if (aux[AT_SYSINFO]) __sysinfo = aux[AT_SYSINFO];
                libc.page_size = aux[AT_PAGESZ];

                if (!pn) pn = (void*)aux[AT_EXECFN];
                if (!pn) pn = "";
                __progname = __progname_full = pn;
                for (i=0; pn[i]; i++) if (pn[i]=='/') __progname = pn+i+1;

>               __init_tls(aux);
                __init_ssp((void *)aux[AT_RANDOM]);

                if (aux[AT_UID]==aux[AT_EUID] && aux[AT_GID]==aux[AT_EGID]
                        && !aux[AT_SECURE]) return;

                struct pollfd pfd[3] = { {.fd=0}, {.fd=1}, {.fd=2} };
                int r =
        #ifdef SYS_poll
                __syscall(SYS_poll, pfd, 3, 0);
        #else
                __syscall(SYS_ppoll, pfd, 3, &(struct timespec){0}, 0, _NSIG/8);
remote Thread 1.1 In: __init_libc                                        L39   PC: 0x8049270
(gdb) n
(gdb) s
__copy_tls (mem=0x804c540 <builtin_tls> "") at src/env/__init_tls.c:60
(gdb) s
__init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:17
(gdb) up
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:79
(gdb) down
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
(gdb) 
```



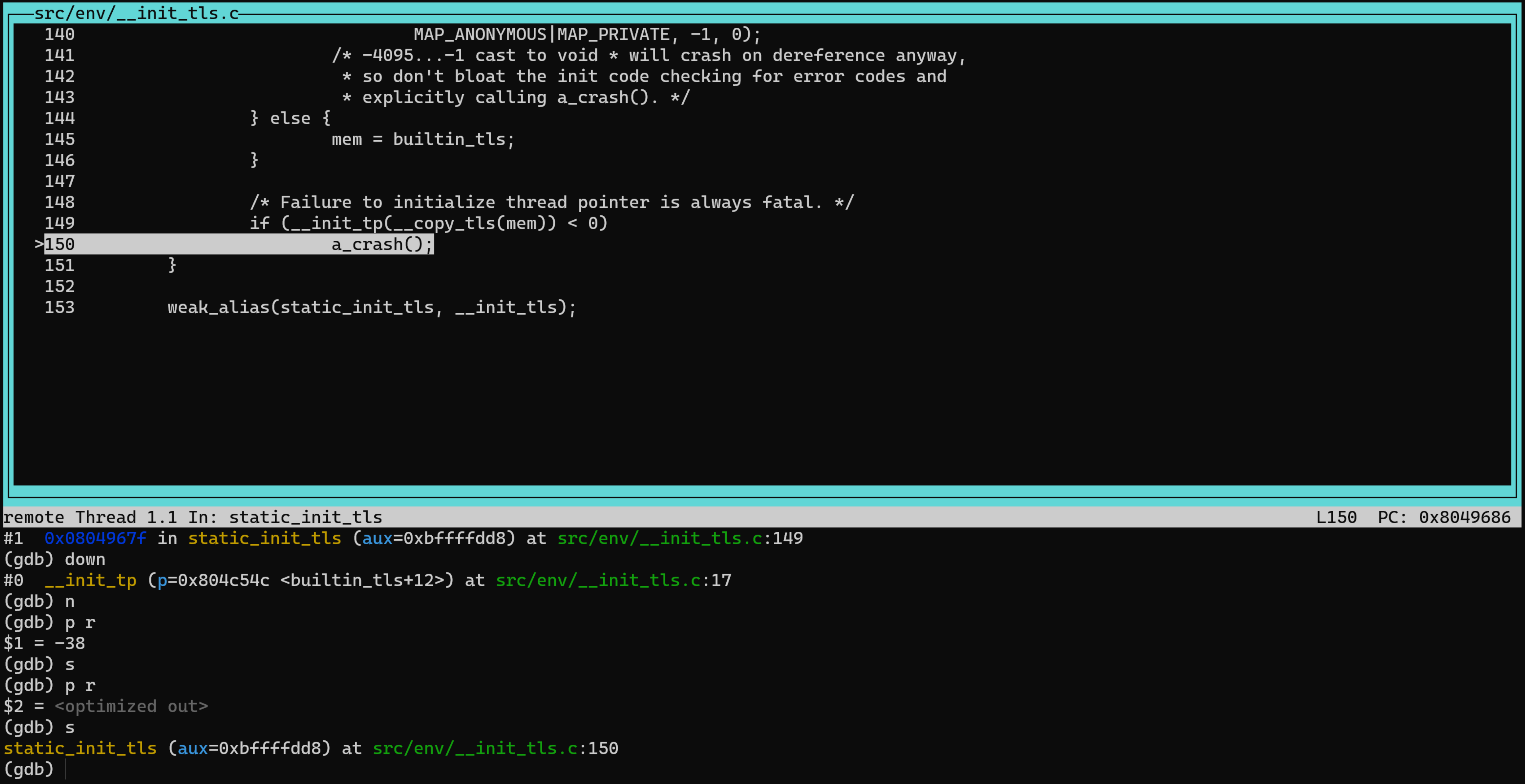
```
src/env/__init_tls.c
 139                                 0, libc.tls_size, PROT_READ|PROT_WRITE,
 140                                 MAP_ANONYMOUS|MAP_PRIVATE, -1, 0);
 141                         /* -4095...-1 cast to void * will crash on dereference anyway,
 142                          * so don't bloat the init code checking for error codes and
 143                          * explicitly calling a_crash(). */
 144                 } else {
 145                         mem = builtin_tls;
 146                 }
 147
 148                 /* Failure to initialize thread pointer is always fatal. */
>149                 if (__init_tp(__copy_tls(mem)) < 0)
 150                         a_crash();
 151         }
 152
 153         weak_alias(static_init_tls, __init_tls);

remote Thread 1.1 In: static_init_tls                                                                L149  PC: 0x804967f
__copy_tls (mem=0x804c540 <builtin_tls> "") at src/env/__init_tls.c:60
(gdb) s
__init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:17
(gdb) up
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:79
(gdb) down
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
(gdb) down
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
(gdb)
```



```
src/env/__init_tls.c
 7         #include "pthread_impl.h"
 8         #include "libc.h"
 9         #include "atomic.h"
 10        #include "syscall.h"
 11
 12        volatile int __thread_list_lock;
 13
 14        int __init_tp(void *p)
 15        {
 16                pthread_t td = p;
>17                td->self = td;
 18                int r = __set_thread_area(TP_ADJ(p));
 19                if (r < 0) return -1;
 20                if (!r) libc.can_do_threads = 1;
 21                td->detach_state = DT_JOINABLE;
 22                td->tid = __syscall(SYS_set_tid_address, &__thread_list_lock);
 23                td->locale = &libc.global_locale;
 24                td->robust_list.head = &td->robust_list.head;
 25                td->sysinfo = __sysinfo;
 26                td->next = td->prev = td;
 27                return 0;
 28        }
remote Thread 1.1 In: __init_tp                                                          L17   PC: 0x8049460
__init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:17
(gdb) up
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:79
(gdb) down
#2  0x08049270 in __init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
(gdb) down
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
(gdb) down
#0  __init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:17
(gdb) 
```



```
┌─src/env/__init_tls.c─────────────────────────────────────────────────────────┐
│   7          #include "pthread_impl.h"                                        │
│   8          #include "libc.h"                                                │
│   9          #include "atomic.h"                                              │
│   10         #include "syscall.h"                                             │
│   11                                                                          │
│   12         volatile int __thread_list_lock;                                 │
│   13                                                                          │
│   14         int __init_tp(void *p)                                           │
│   15         {                                                                │
│   16                 pthread_t td = p;                                        │
│   17                 td->self = td;                                           │
│   18                 int r = __set_thread_area(TP_ADJ(p));                    │
│  >19                 if (r < 0) return -1;                                    │
│   20                 if (!r) libc.can_do_threads = 1;                         │
│   21                 td->detach_state = DT_JOINABLE;                          │
│   22                 td->tid = __syscall(SYS_set_tid_address, &__thread_list_lock);
│   23                 td->locale = &libc.global_locale;                        │
│   24                 td->robust_list.head = &td->robust_list.head;            │
│   25                 td->sysinfo = __sysinfo;                                 │
│   26                 td->next = td->prev = td;                                │
│   27                 return 0;                                                │
│   28         }                                                                │
└──────────────────────────────────────────────────────────────────────────────┘
remote Thread 1.1 In: __init_tp                                L19   PC: 0x8049468
#2  0x08049270 in __init_libc (envp=0xbffffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
#3  0x0804937d in __libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:79
(gdb) down
#2  0x08049270 in __init_libc (envp=0xbffffff10, pn=<optimized out>) at src/env/__libc_start_main.c:39
(gdb) down
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
(gdb) down
#0  __init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:17
(gdb) n
(gdb) p r
$1 = -38
(gdb) 
```



```
src/env/__init_tls.c
   140                                 MAP_ANONYMOUS|MAP_PRIVATE, -1, 0);
   141                         /* -4095...-1 cast to void * will crash on dereference anyway,
   142                          * so don't bloat the init code checking for error codes and
   143                          * explicitly calling a_crash(). */
   144                 } else {
   145                         mem = builtin_tls;
   146                 }
   147
   148                 /* Failure to initialize thread pointer is always fatal. */
   149                 if (__init_tp(__copy_tls(mem)) < 0)
  >150                         a_crash();
   151         }
   152
   153         weak_alias(static_init_tls, __init_tls);

remote Thread 1.1 In: static_init_tls                                                     L150  PC: 0x8049686
#1  0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
(gdb) down
#0  __init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:17
(gdb) n
(gdb) p r
$1 = -38
(gdb) s
(gdb) p r
$2 = <optimized out>
(gdb) s
static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:150
(gdb) 
```

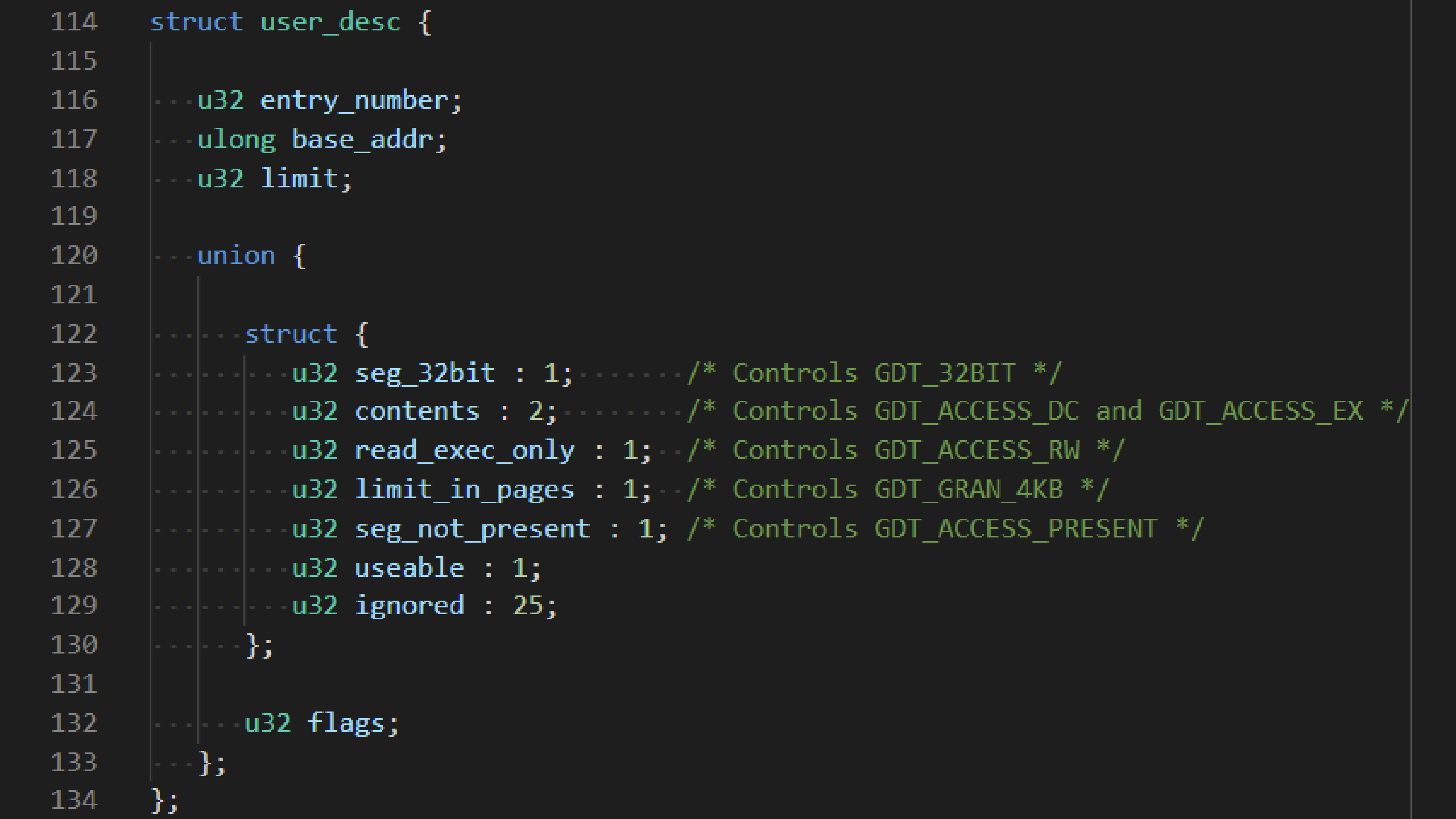
Sometimes cheating works

Sometimes cheating works

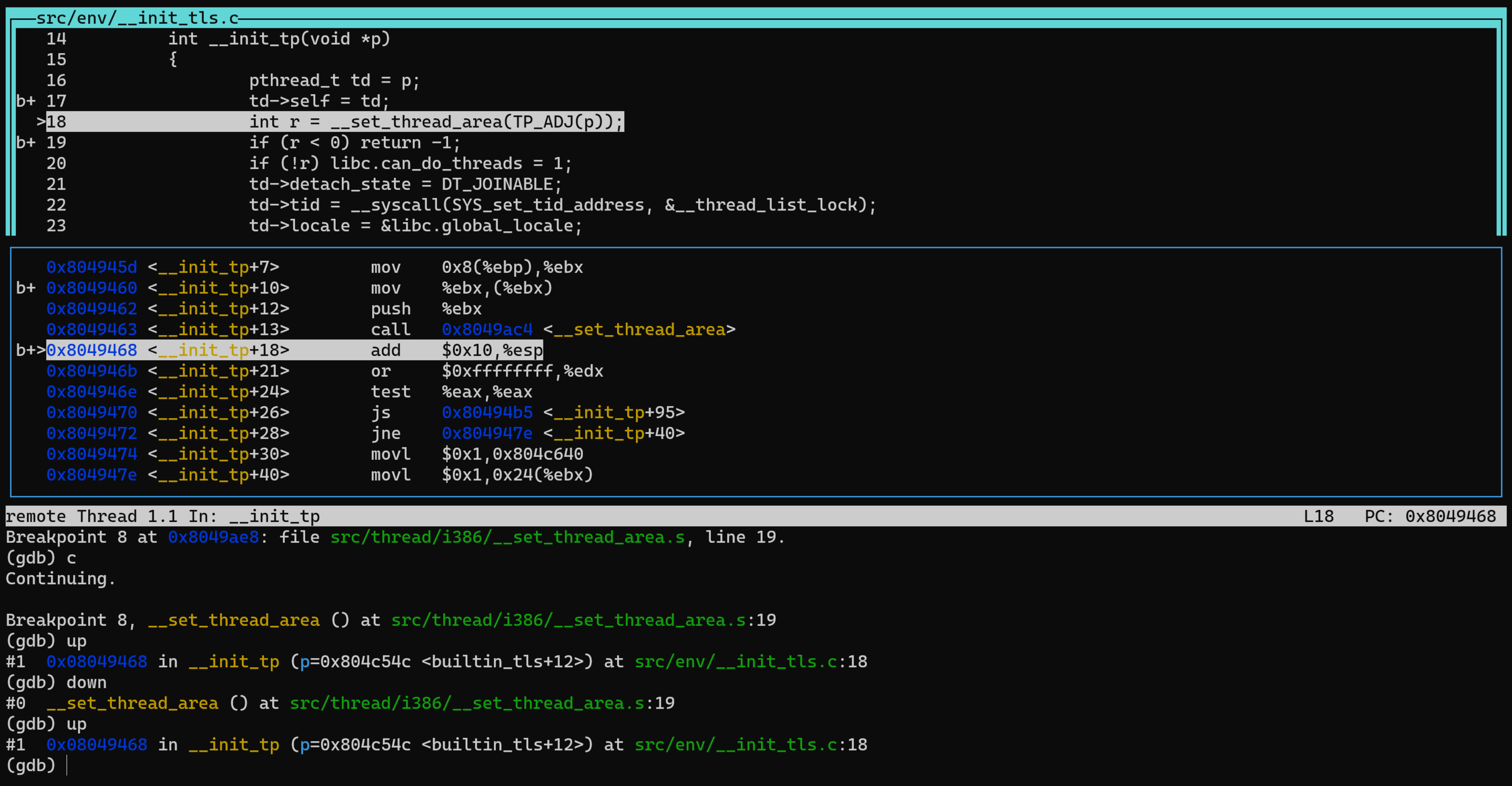
- Sometimes it **doesn’t.**

Sometimes cheating works

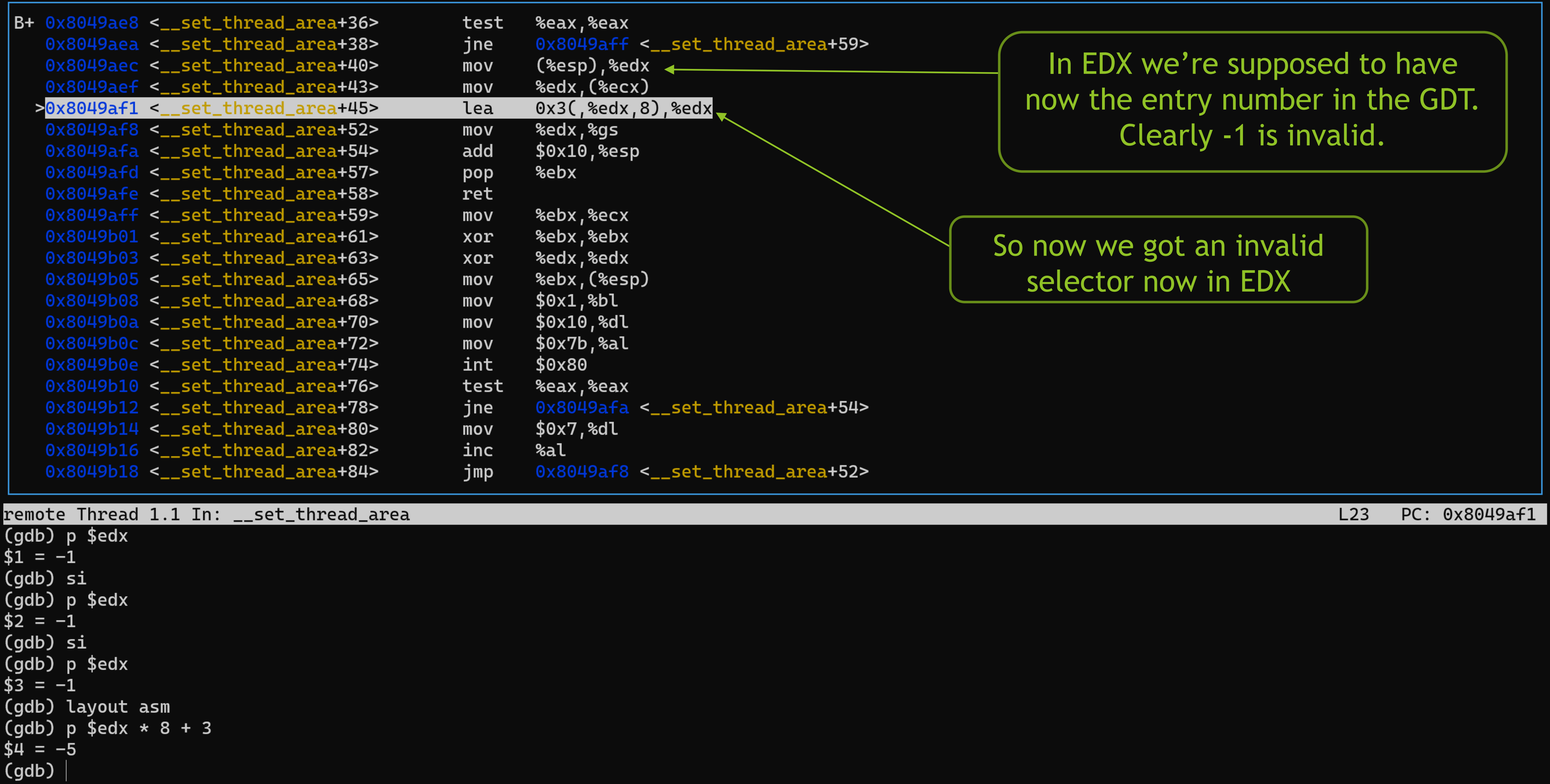
- Sometimes it **doesn’t.**
- Can we try returning 0 instead and see what happens?



```c
struct user_desc {

    u32 entry_number;
    ulong base_addr;
    u32 limit;

    union {

        struct {
            u32 seg_32bit : 1;       /* Controls GDT_32BIT */
            u32 contents : 2;        /* Controls GDT_ACCESS_DC and GDT_ACCESS_EX */
            u32 read_exec_only : 1;  /* Controls GDT_ACCESS_RW */
            u32 limit_in_pages : 1;  /* Controls GDT_GRAN_4KB */
            u32 seg_not_present : 1; /* Controls GDT_ACCESS_PRESENT */
            u32 useable : 1;
            u32 ignored : 25;
        };

        u32 flags;
    };
};
```



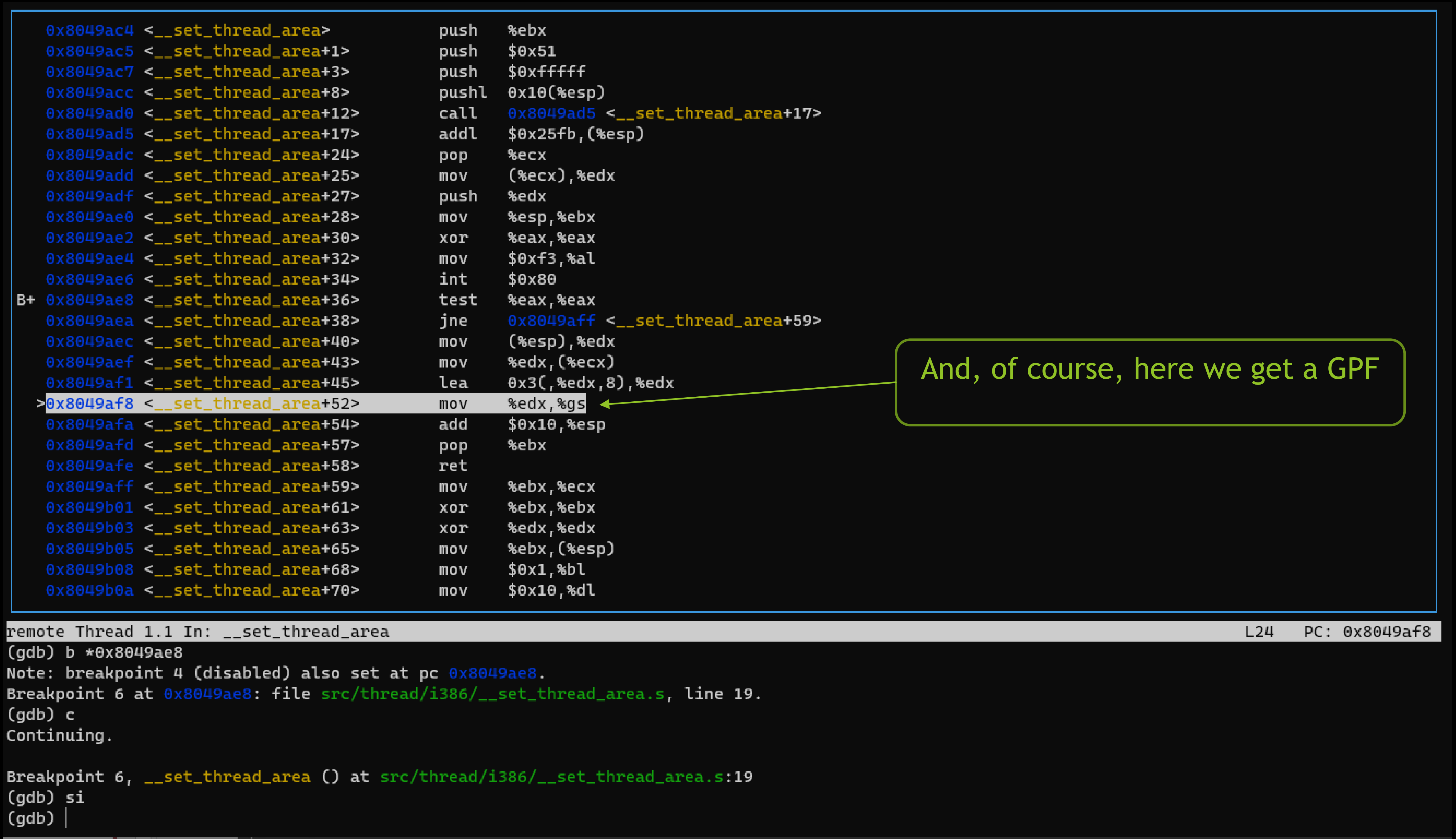
```
src/env/__init_tls.c
    14          int __init_tp(void *p)
    15          {
    16                  pthread_t td = p;
b+  17                  td->self = td;
  > 18                  int r = __set_thread_area(TP_ADJ(p));
b+  19                  if (r < 0) return -1;
    20                  if (!r) libc.can_do_threads = 1;
    21                  td->detach_state = DT_JOINABLE;
    22                  td->tid = __syscall(SYS_set_tid_address, &__thread_list_lock);
    23                  td->locale = &libc.global_locale;

    0x804945d <__init_tp+7>         mov     0x8(%ebp),%ebx
b+  0x8049460 <__init_tp+10>        mov     %ebx,(%ebx)
    0x8049462 <__init_tp+12>        push    %ebx
    0x8049463 <__init_tp+13>        call    0x8049ac4 <__set_thread_area>
b+> 0x8049468 <__init_tp+18>        add     $0x10,%esp
    0x804946b <__init_tp+21>        or      $0xffffffff,%edx
    0x804946e <__init_tp+24>        test    %eax,%eax
    0x8049470 <__init_tp+26>        js      0x80494b5 <__init_tp+95>
    0x8049472 <__init_tp+28>        jne     0x804947e <__init_tp+40>
    0x8049474 <__init_tp+30>        movl    $0x1,0x804c640
    0x804947e <__init_tp+40>        movl    $0x1,0x24(%ebx)

remote Thread 1.1 In: __init_tp                                                                  L18   PC: 0x8049468
Breakpoint 8 at 0x8049ae8: file src/thread/i386/__set_thread_area.s, line 19.
(gdb) c
Continuing.

Breakpoint 8, __set_thread_area () at src/thread/i386/__set_thread_area.s:19
(gdb) up
#1  0x08049468 in __init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:18
(gdb) down
#0  __set_thread_area () at src/thread/i386/__set_thread_area.s:19
(gdb) up
#1  0x08049468 in __init_tp (p=0x804c54c <builtin_tls+12>) at src/env/__init_tls.c:18
(gdb) 
```



```
B+ 0x8049ae8 <__set_thread_area+36>        test   %eax,%eax
   0x8049aea <__set_thread_area+38>        jne    0x8049aff <__set_thread_area+59>
   0x8049aec <__set_thread_area+40>        mov    (%esp),%edx
   0x8049aef <__set_thread_area+43>        mov    %edx,(%ecx)
  >0x8049af1 <__set_thread_area+45>        lea    0x3(,%edx,8),%edx
   0x8049af8 <__set_thread_area+52>        mov    %edx,%gs
   0x8049afa <__set_thread_area+54>        add    $0x10,%esp
   0x8049afd <__set_thread_area+57>        pop    %ebx
   0x8049afe <__set_thread_area+58>        ret
   0x8049aff <__set_thread_area+59>        mov    %ebx,%ecx
   0x8049b01 <__set_thread_area+61>        xor    %ebx,%ebx
   0x8049b03 <__set_thread_area+63>        xor    %edx,%edx
   0x8049b05 <__set_thread_area+65>        mov    %ebx,(%esp)
   0x8049b08 <__set_thread_area+68>        mov    $0x1,%bl
   0x8049b0a <__set_thread_area+70>        mov    $0x10,%dl
   0x8049b0c <__set_thread_area+72>        mov    $0x7b,%al
   0x8049b0e <__set_thread_area+74>        int    $0x80
   0x8049b10 <__set_thread_area+76>        test   %eax,%eax
   0x8049b12 <__set_thread_area+78>        jne    0x8049afa <__set_thread_area+54>
   0x8049b14 <__set_thread_area+80>        mov    $0x7,%dl
   0x8049b16 <__set_thread_area+82>        inc    %al
   0x8049b18 <__set_thread_area+84>        jmp    0x8049af8 <__set_thread_area+52>

remote Thread 1.1 In: __set_thread_area                                          L23   PC: 0x8049af1
(gdb) p $edx
$1 = -1
(gdb) si
(gdb) p $edx
$2 = -1
(gdb) si
(gdb) p $edx
$3 = -1
(gdb) layout asm
(gdb) p $edx * 8 + 3
$4 = -5
(gdb)
```

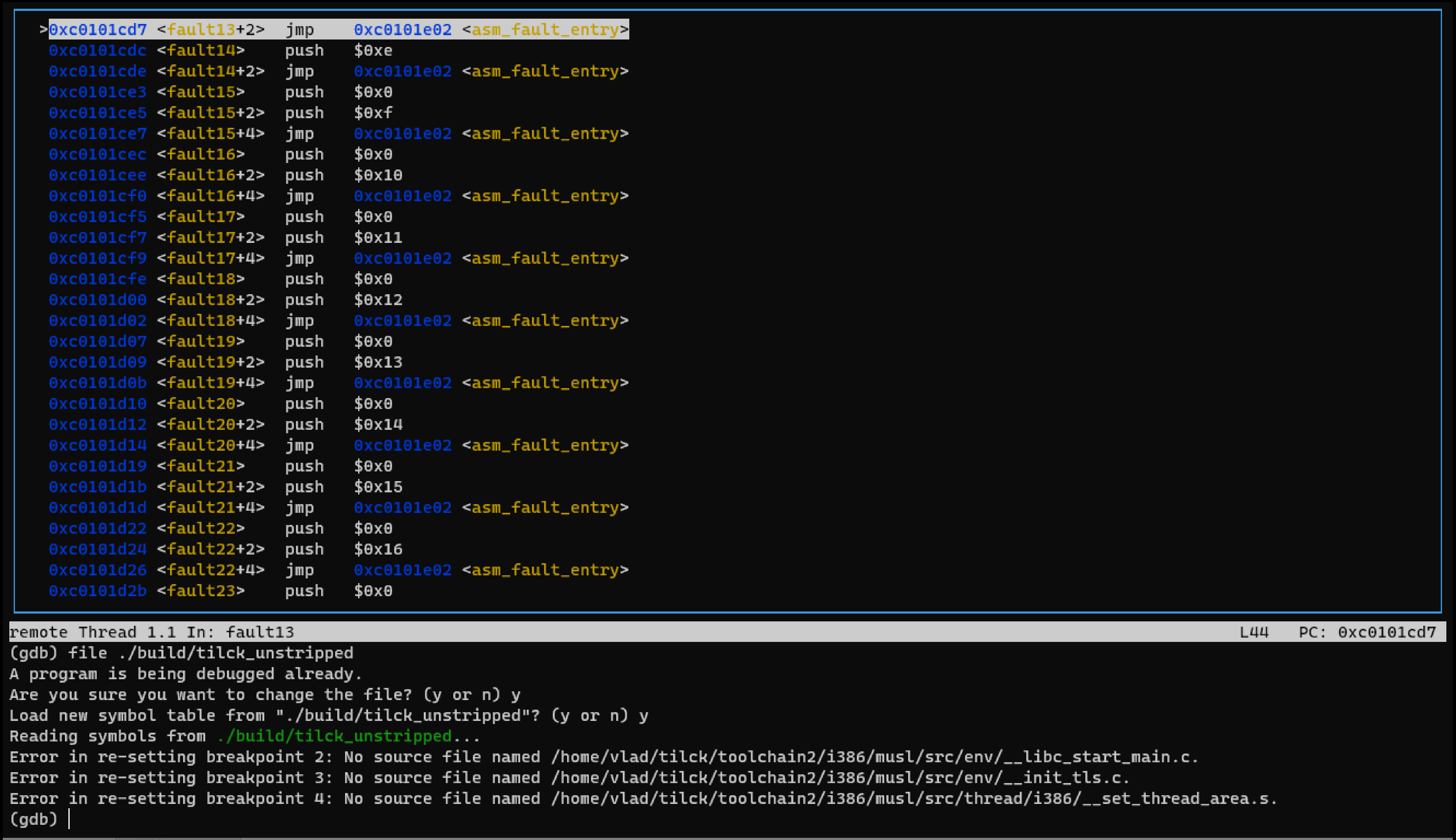
In EDX we're supposed to have now the entry number in the GDT. Clearly -1 is invalid.

So now we got an invalid selector now in EDX



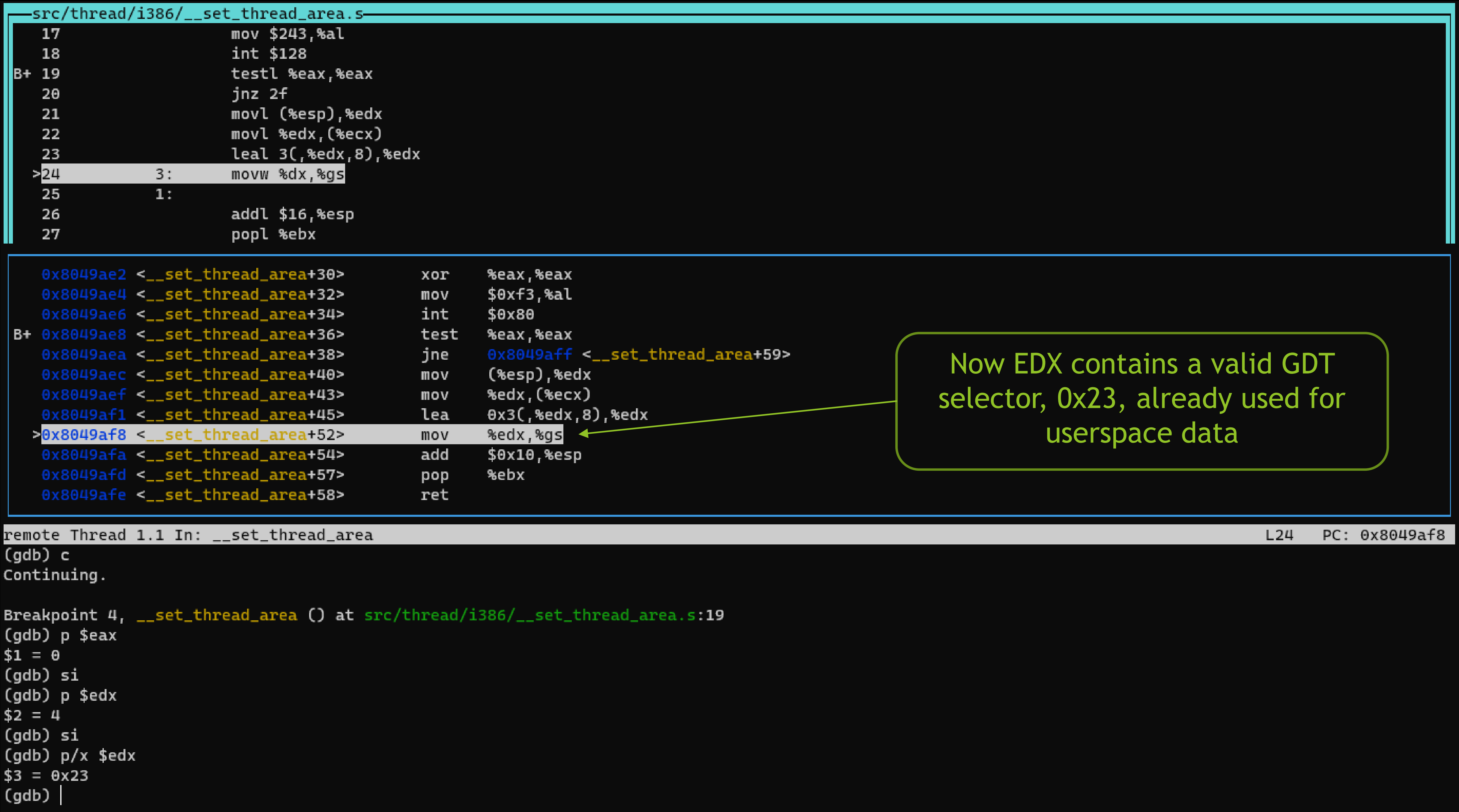
```
   0x8049ac4 <__set_thread_area>          push   %ebx
   0x8049ac5 <__set_thread_area+1>        push   $0x51
   0x8049ac7 <__set_thread_area+3>        push   $0xfffff
   0x8049acc <__set_thread_area+8>        pushl  0x10(%esp)
   0x8049ad0 <__set_thread_area+12>       call   0x8049ad5 <__set_thread_area+17>
   0x8049ad5 <__set_thread_area+17>       addl   $0x25fb,(%esp)
   0x8049adc <__set_thread_area+24>       pop    %ecx
   0x8049add <__set_thread_area+25>       mov    (%ecx),%edx
   0x8049adf <__set_thread_area+27>       push   %edx
   0x8049ae0 <__set_thread_area+28>       mov    %esp,%ebx
   0x8049ae2 <__set_thread_area+30>       xor    %eax,%eax
   0x8049ae4 <__set_thread_area+32>       mov    $0xf3,%al
   0x8049ae6 <__set_thread_area+34>       int    $0x80
B+ 0x8049ae8 <__set_thread_area+36>       test   %eax,%eax
   0x8049aea <__set_thread_area+38>       jne    0x8049aff <__set_thread_area+59>
   0x8049aec <__set_thread_area+40>       mov    (%esp),%edx
   0x8049aef <__set_thread_area+43>       mov    %edx,(%ecx)
   0x8049af1 <__set_thread_area+45>       lea    0x3(,%edx,8),%edx
  >0x8049af8 <__set_thread_area+52>       mov    %edx,%gs
   0x8049afa <__set_thread_area+54>       add    $0x10,%esp
   0x8049afd <__set_thread_area+57>       pop    %ebx
   0x8049afe <__set_thread_area+58>       ret
   0x8049aff <__set_thread_area+59>       mov    %ebx,%ecx
   0x8049b01 <__set_thread_area+61>       xor    %ebx,%ebx
   0x8049b03 <__set_thread_area+63>       xor    %edx,%edx
   0x8049b05 <__set_thread_area+65>       mov    %ebx,(%esp)
   0x8049b08 <__set_thread_area+68>       mov    $0x1,%bl
   0x8049b0a <__set_thread_area+70>       mov    $0x10,%dl
remote Thread 1.1 In: __set_thread_area                                     L24   PC: 0x8049af8
(gdb) b *0x8049ae8
Note: breakpoint 4 (disabled) also set at pc 0x8049ae8.
Breakpoint 6 at 0x8049ae8: file src/thread/i386/__set_thread_area.s, line 19.
(gdb) c
Continuing.

Breakpoint 6, __set_thread_area () at src/thread/i386/__set_thread_area.s:19
(gdb) si
(gdb)
```

And, of course, here we get a GPF



```
  >0xc0101cd7 <fault13+2>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101cdc <fault14>     push   $0xe
   0xc0101cde <fault14+2>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101ce3 <fault15>     push   $0x0
   0xc0101ce5 <fault15+2>   push   $0xf
   0xc0101ce7 <fault15+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101cec <fault16>     push   $0x0
   0xc0101cee <fault16+2>   push   $0x10
   0xc0101cf0 <fault16+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101cf5 <fault17>     push   $0x0
   0xc0101cf7 <fault17+2>   push   $0x11
   0xc0101cf9 <fault17+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101cfe <fault18>     push   $0x0
   0xc0101d00 <fault18+2>   push   $0x12
   0xc0101d02 <fault18+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101d07 <fault19>     push   $0x0
   0xc0101d09 <fault19+2>   push   $0x13
   0xc0101d0b <fault19+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101d10 <fault20>     push   $0x0
   0xc0101d12 <fault20+2>   push   $0x14
   0xc0101d14 <fault20+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101d19 <fault21>     push   $0x0
   0xc0101d1b <fault21+2>   push   $0x15
   0xc0101d1d <fault21+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101d22 <fault22>     push   $0x0
   0xc0101d24 <fault22+2>   push   $0x16
   0xc0101d26 <fault22+4>   jmp    0xc0101e02 <asm_fault_entry>
   0xc0101d2b <fault23>     push   $0x0

remote Thread 1.1 In: fault13                                                           L44   PC: 0xc0101cd7
(gdb) file ./build/tilck_unstripped
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "./build/tilck_unstripped"? (y or n) y
Reading symbols from ./build/tilck_unstripped...
Error in re-setting breakpoint 2: No source file named /home/vlad/tilck/toolchain2/i386/musl/src/env/__libc_start_main.c.
Error in re-setting breakpoint 3: No source file named /home/vlad/tilck/toolchain2/i386/musl/src/env/__init_tls.c.
Error in re-setting breakpoint 4: No source file named /home/vlad/tilck/toolchain2/i386/musl/src/thread/i386/__set_thread_area.s.
(gdb) 
```

What if we returned 0 and set a valid GDT entry number in **user_desc**, without doing anything else?

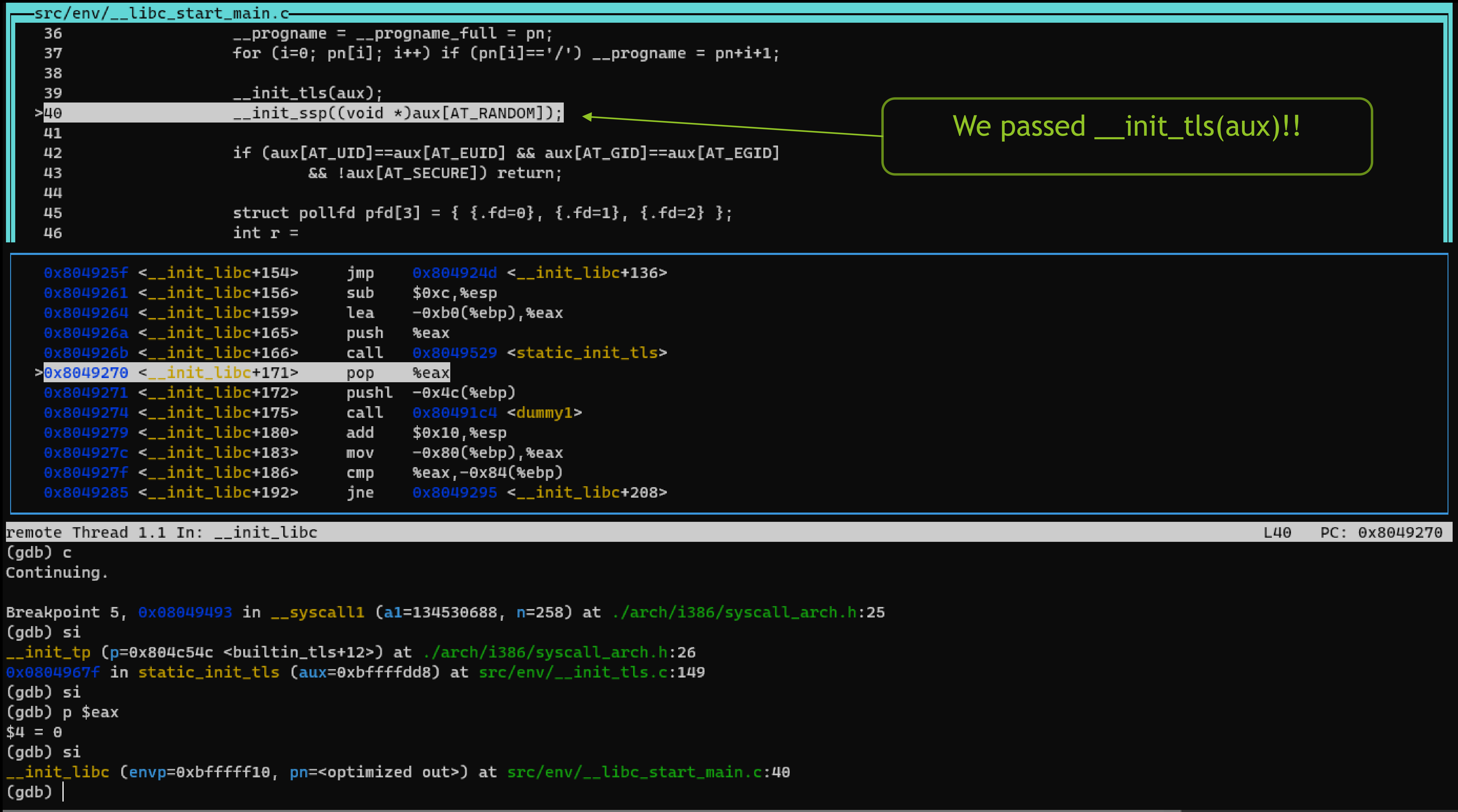


```
src/thread/i386/__set_thread_area.s
                           mov $243,%al
                           int $128
B+                         testl %eax,%eax
                           jnz 2f
                           movl (%esp),%edx
                           movl %edx,(%ecx)
                           leal 3(,%edx,8),%edx
  >             3:         movw %dx,%gs
                1:
                           addl $16,%esp
                           popl %ebx
```

```
   0x8049ae2 <__set_thread_area+30>        xor    %eax,%eax
   0x8049ae4 <__set_thread_area+32>        mov    $0xf3,%al
   0x8049ae6 <__set_thread_area+34>        int    $0x80
B+ 0x8049ae8 <__set_thread_area+36>        test   %eax,%eax
   0x8049aea <__set_thread_area+38>        jne    0x8049aff <__set_thread_area+59>
   0x8049aec <__set_thread_area+40>        mov    (%esp),%edx
   0x8049aef <__set_thread_area+43>        mov    %edx,(%ecx)
   0x8049af1 <__set_thread_area+45>        lea    0x3(,%edx,8),%edx
  >0x8049af8 <__set_thread_area+52>        mov    %edx,%gs
   0x8049afa <__set_thread_area+54>        add    $0x10,%esp
   0x8049afd <__set_thread_area+57>        pop    %ebx
   0x8049afe <__set_thread_area+58>        ret
```

Now EDX contains a valid GDT selector, 0x23, already used for userspace data

```
remote Thread 1.1 In: __set_thread_area                                          L24   PC: 0x8049af8
(gdb) c
Continuing.

Breakpoint 4, __set_thread_area () at src/thread/i386/__set_thread_area.s:19
(gdb) p $eax
$1 = 0
(gdb) si
(gdb) p $edx
$2 = 4
(gdb) si
(gdb) p/x $edx
$3 = 0x23
(gdb)
```

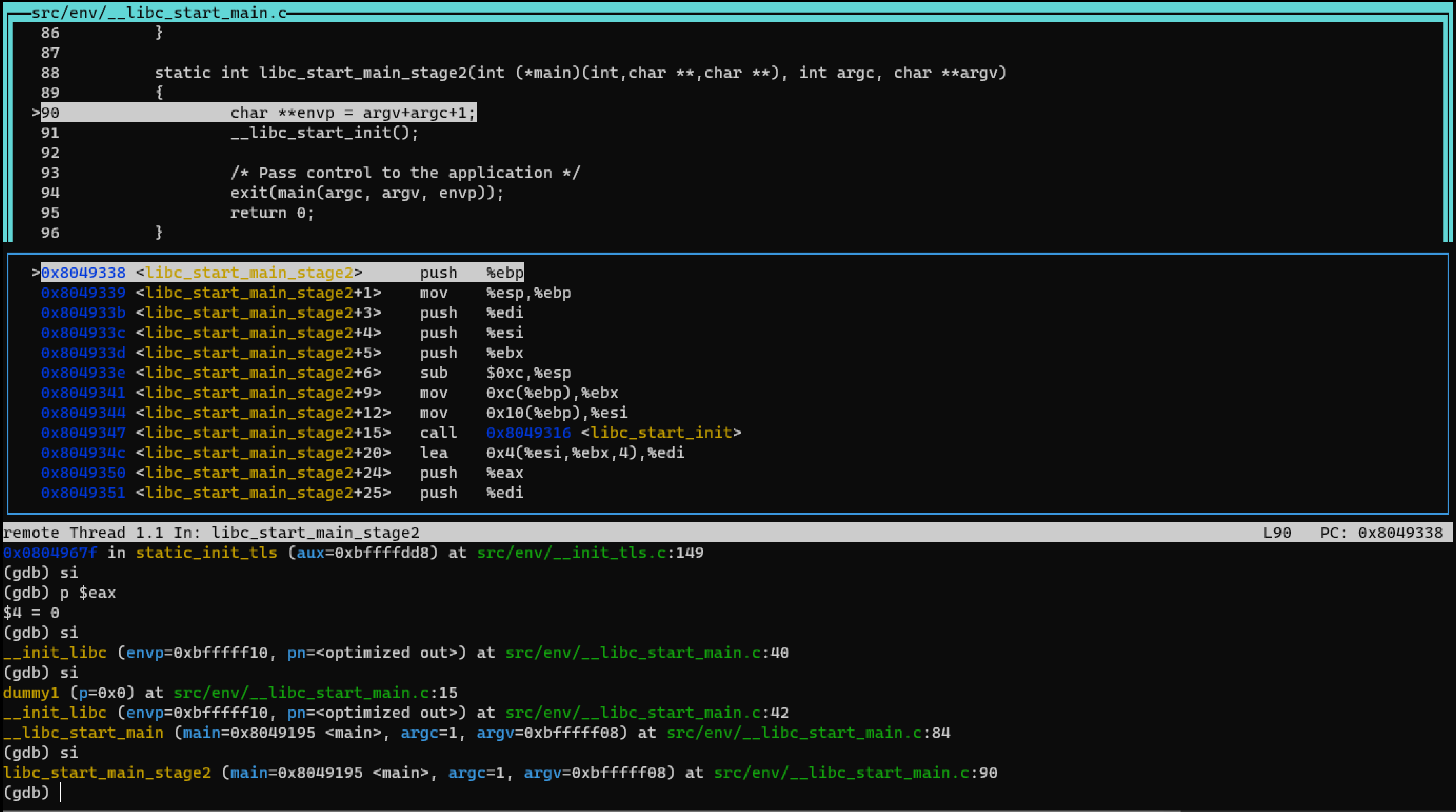


```
src/env/__libc_start_main.c
 36                     __progname = __progname_full = pn;
 37                     for (i=0; pn[i]; i++) if (pn[i]=='/') __progname = pn+i+1;
 38
 39                     __init_tls(aux);
>40                     __init_ssp((void *)aux[AT_RANDOM]);
 41
 42                     if (aux[AT_UID]==aux[AT_EUID] && aux[AT_GID]==aux[AT_EGID]
 43                             && !aux[AT_SECURE]) return;
 44
 45                     struct pollfd pfd[3] = { {.fd=0}, {.fd=1}, {.fd=2} };
 46                     int r =
```

We passed __init_tls(aux)!!

```
 0x804925f <__init_libc+154>     jmp    0x804924d <__init_libc+136>
 0x8049261 <__init_libc+156>     sub    $0xc,%esp
 0x8049264 <__init_libc+159>     lea    -0xb0(%ebp),%eax
 0x804926a <__init_libc+165>     push   %eax
 0x804926b <__init_libc+166>     call   0x8049529 <static_init_tls>
>0x8049270 <__init_libc+171>     pop    %eax
 0x8049271 <__init_libc+172>     pushl  -0x4c(%ebp)
 0x8049274 <__init_libc+175>     call   0x80491c4 <dummy1>
 0x8049279 <__init_libc+180>     add    $0x10,%esp
 0x804927c <__init_libc+183>     mov    -0x80(%ebp),%eax
 0x804927f <__init_libc+186>     cmp    %eax,-0x84(%ebp)
 0x8049285 <__init_libc+192>     jne    0x8049295 <__init_libc+208>
```

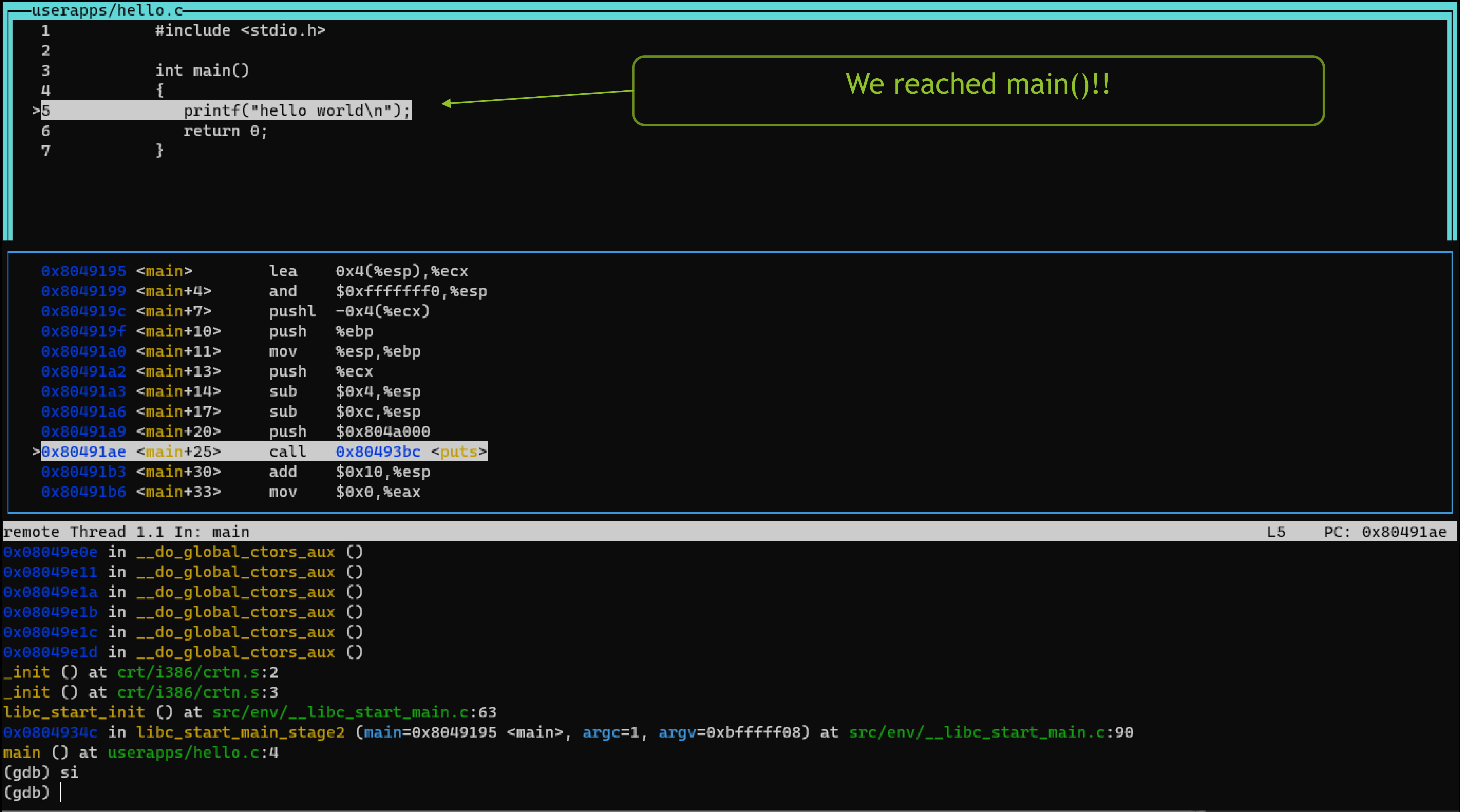
```
remote Thread 1.1 In: __init_libc                                                                    L40   PC: 0x8049270
(gdb) c
Continuing.

Breakpoint 5, 0x08049493 in __syscall1 (a1=134530688, n=258) at ./arch/i386/syscall_arch.h:25
(gdb) si
__init_tp (p=0x804c54c <builtin_tls+12>) at ./arch/i386/syscall_arch.h:26
0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
(gdb) si
(gdb) p $eax
$4 = 0
(gdb) si
__init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:40
(gdb) |
```



```
src/env/__libc_start_main.c
 86         }
 87
 88         static int libc_start_main_stage2(int (*main)(int,char **,char **), int argc, char **argv)
 89         {
>90                 char **envp = argv+argc+1;
 91                 __libc_start_init();
 92
 93                 /* Pass control to the application */
 94                 exit(main(argc, argv, envp));
 95                 return 0;
 96         }
```

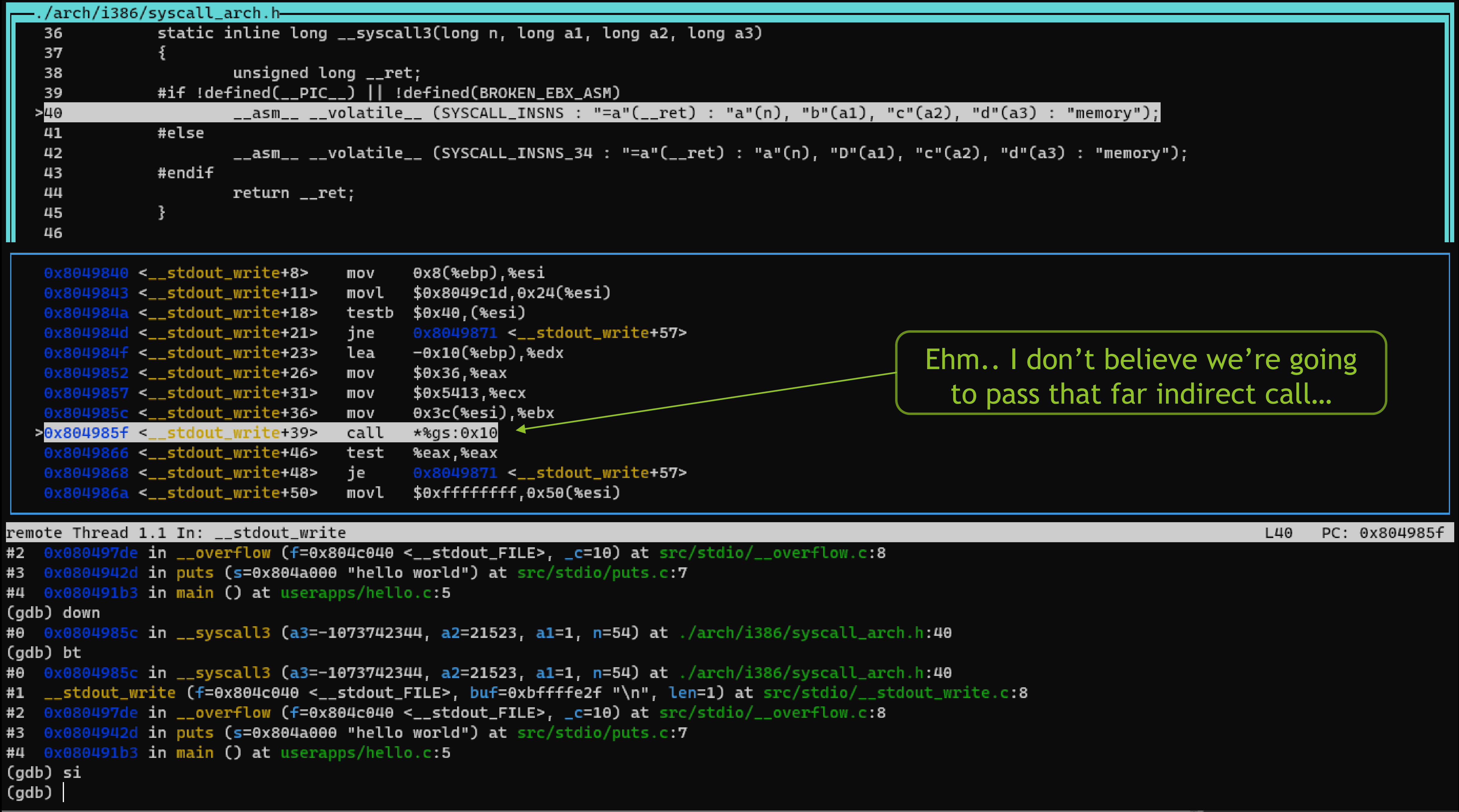
```
>0x8049338 <libc_start_main_stage2>       push   %ebp
 0x8049339 <libc_start_main_stage2+1>     mov    %esp,%ebp
 0x804933b <libc_start_main_stage2+3>     push   %edi
 0x804933c <libc_start_main_stage2+4>     push   %esi
 0x804933d <libc_start_main_stage2+5>     push   %ebx
 0x804933e <libc_start_main_stage2+6>     sub    $0xc,%esp
 0x8049341 <libc_start_main_stage2+9>     mov    0xc(%ebp),%ebx
 0x8049344 <libc_start_main_stage2+12>    mov    0x10(%ebp),%esi
 0x8049347 <libc_start_main_stage2+15>    call   0x8049316 <libc_start_init>
 0x804934c <libc_start_main_stage2+20>    lea    0x4(%esi,%ebx,4),%edi
 0x8049350 <libc_start_main_stage2+24>    push   %eax
 0x8049351 <libc_start_main_stage2+25>    push   %edi
```

```
remote Thread 1.1 In: libc_start_main_stage2                                        L90   PC: 0x8049338
0x0804967f in static_init_tls (aux=0xbffffdd8) at src/env/__init_tls.c:149
(gdb) si
(gdb) p $eax
$4 = 0
(gdb) si
__init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:40
(gdb) si
dummy1 (p=0x0) at src/env/__libc_start_main.c:15
__init_libc (envp=0xbfffff10, pn=<optimized out>) at src/env/__libc_start_main.c:42
__libc_start_main (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:84
(gdb) si
libc_start_main_stage2 (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:90
(gdb) 
```



We reached main()!!

```
userapps/hello.c
  1         #include <stdio.h>
  2
  3         int main()
  4         {
 >5             printf("hello world\n");
  6             return 0;
  7         }

   0x8049195 <main>       lea    0x4(%esp),%ecx
   0x8049199 <main+4>     and    $0xfffffff0,%esp
   0x804919c <main+7>     pushl  -0x4(%ecx)
   0x804919f <main+10>    push   %ebp
   0x80491a0 <main+11>    mov    %esp,%ebp
   0x80491a2 <main+13>    push   %ecx
   0x80491a3 <main+14>    sub    $0x4,%esp
   0x80491a6 <main+17>    sub    $0xc,%esp
   0x80491a9 <main+20>    push   $0x804a000
  >0x80491ae <main+25>    call   0x80493bc <puts>
   0x80491b3 <main+30>    add    $0x10,%esp
   0x80491b6 <main+33>    mov    $0x0,%eax

remote Thread 1.1 In: main                                                          L5    PC: 0x80491ae
0x08049e0e in __do_global_ctors_aux ()
0x08049e11 in __do_global_ctors_aux ()
0x08049e1a in __do_global_ctors_aux ()
0x08049e1b in __do_global_ctors_aux ()
0x08049e1c in __do_global_ctors_aux ()
0x08049e1d in __do_global_ctors_aux ()
_init () at crt/i386/crtn.s:2
_init () at crt/i386/crtn.s:3
libc_start_init () at src/env/__libc_start_main.c:63
0x0804934c in libc_start_main_stage2 (main=0x8049195 <main>, argc=1, argv=0xbfffff08) at src/env/__libc_start_main.c:90
main () at userapps/hello.c:4
(gdb) si
(gdb)
```

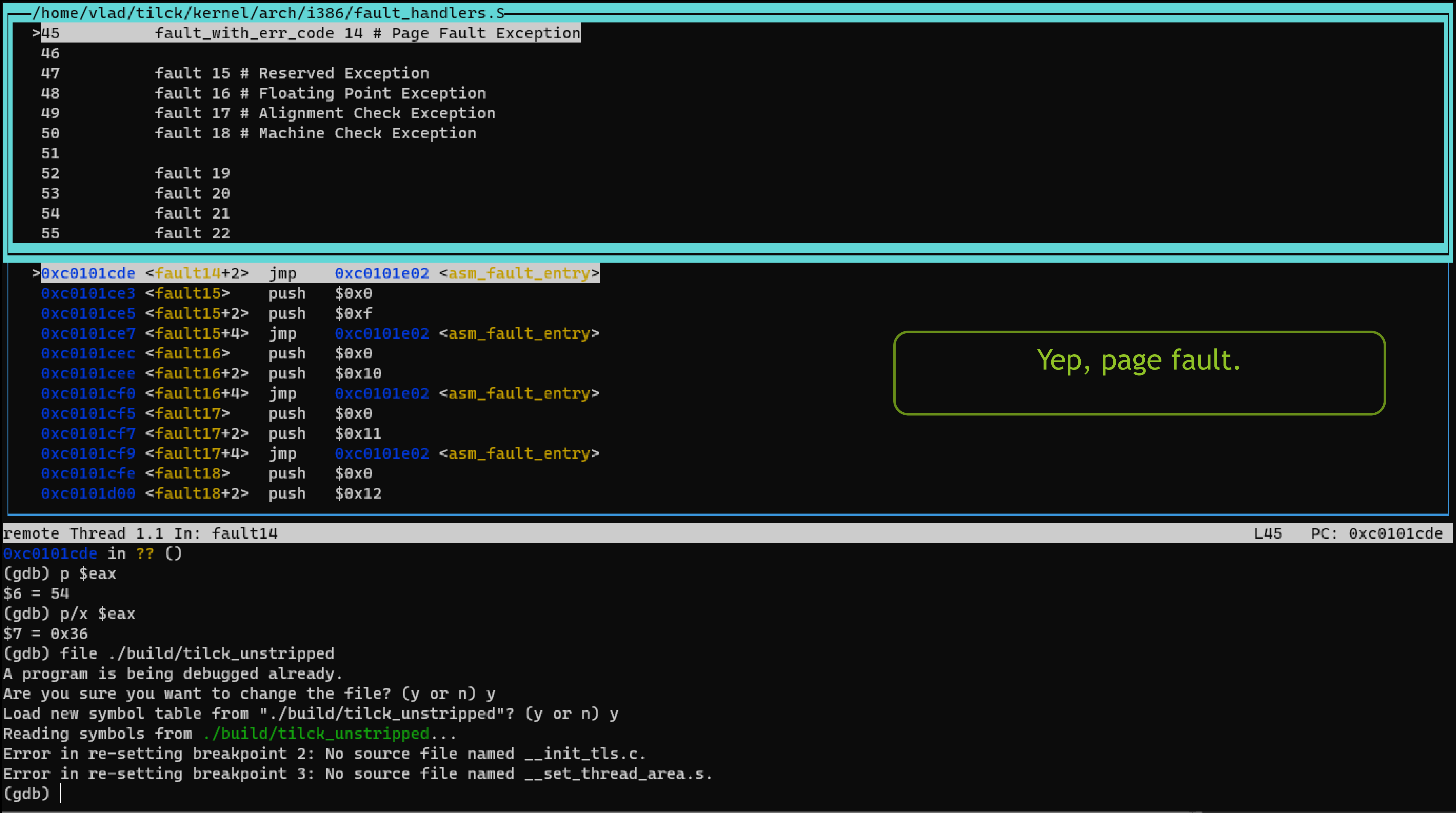


```
./arch/i386/syscall_arch.h
 36         static inline long __syscall3(long n, long a1, long a2, long a3)
 37         {
 38                 unsigned long __ret;
 39         #if !defined(__PIC__) || !defined(BROKEN_EBX_ASM)
>40                 __asm__ __volatile__ (SYSCALL_INSNS : "=a"(__ret) : "a"(n), "b"(a1), "c"(a2), "d"(a3) : "memory");
 41         #else
 42                 __asm__ __volatile__ (SYSCALL_INSNS_34 : "=a"(__ret) : "a"(n), "D"(a1), "c"(a2), "d"(a3) : "memory");
 43         #endif
 44                 return __ret;
 45         }
 46
```

```
 0x8049840 <__stdout_write+8>    mov    0x8(%ebp),%esi
 0x8049843 <__stdout_write+11>   movl   $0x8049c1d,0x24(%esi)
 0x804984a <__stdout_write+18>   testb  $0x40,(%esi)
 0x804984d <__stdout_write+21>   jne    0x8049871 <__stdout_write+57>
 0x804984f <__stdout_write+23>   lea    -0x10(%ebp),%edx
 0x8049852 <__stdout_write+26>   mov    $0x36,%eax
 0x8049857 <__stdout_write+31>   mov    $0x5413,%ecx
 0x804985c <__stdout_write+36>   mov    0x3c(%esi),%ebx
>0x804985f <__stdout_write+39>   call   *%gs:0x10
 0x8049866 <__stdout_write+46>   test   %eax,%eax
 0x8049868 <__stdout_write+48>   je     0x8049871 <__stdout_write+57>
 0x804986a <__stdout_write+50>   movl   $0xffffffff,0x50(%esi)
```

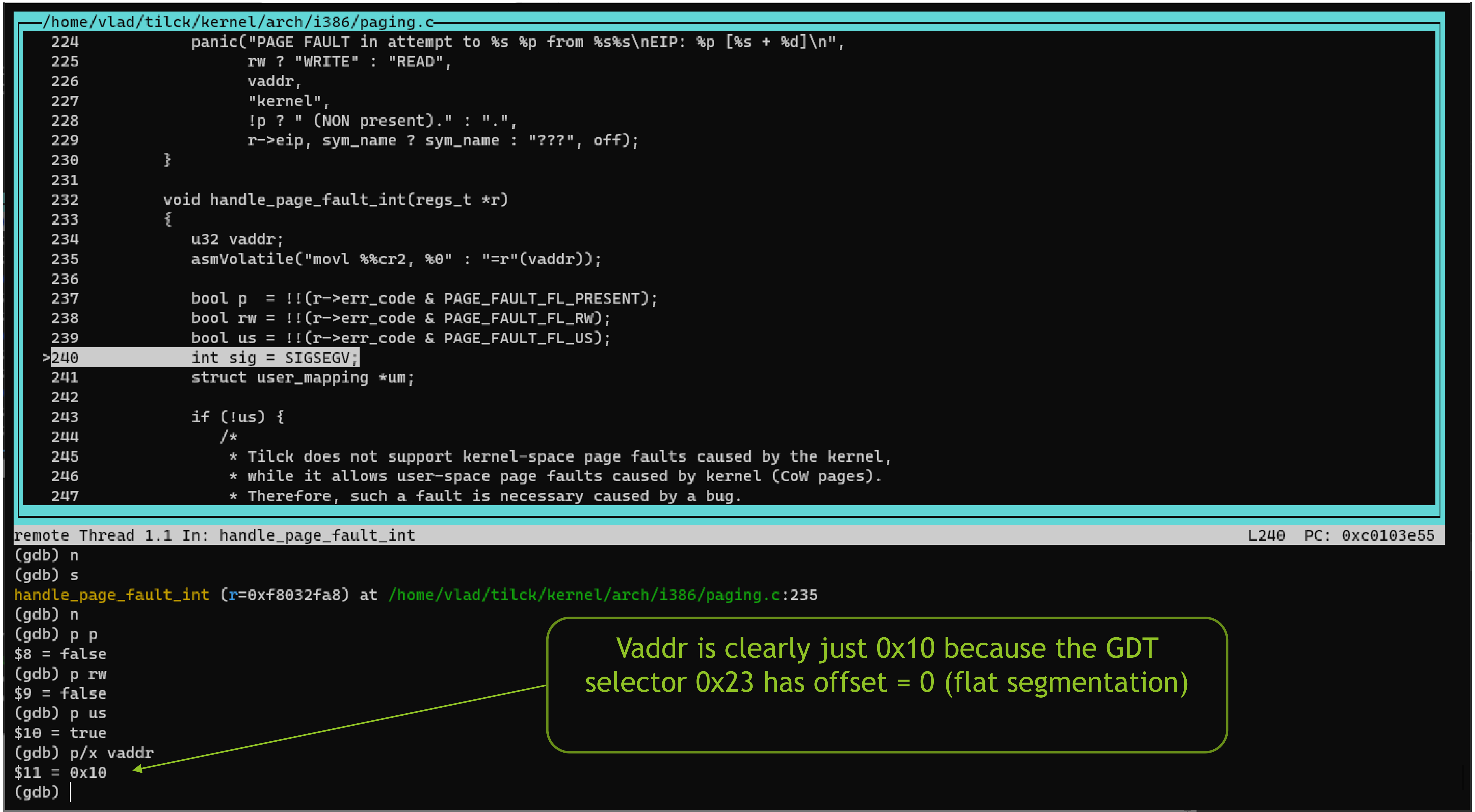
Ehm.. I don't believe we're going to pass that far indirect call...

```
remote Thread 1.1 In: __stdout_write                                                    L40   PC: 0x804985f
#2  0x080497de in __overflow (f=0x804c040 <__stdout_FILE>, _c=10) at src/stdio/__overflow.c:8
#3  0x0804942d in puts (s=0x804a000 "hello world") at src/stdio/puts.c:7
#4  0x080491b3 in main () at userapps/hello.c:5
(gdb) down
#0  0x0804985c in __syscall3 (a3=-1073742344, a2=21523, a1=1, n=54) at ./arch/i386/syscall_arch.h:40
(gdb) bt
#0  0x0804985c in __syscall3 (a3=-1073742344, a2=21523, a1=1, n=54) at ./arch/i386/syscall_arch.h:40
#1  __stdout_write (f=0x804c040 <__stdout_FILE>, buf=0xbffffe2f "\n", len=1) at src/stdio/__stdout_write.c:8
#2  0x080497de in __overflow (f=0x804c040 <__stdout_FILE>, _c=10) at src/stdio/__overflow.c:8
#3  0x0804942d in puts (s=0x804a000 "hello world") at src/stdio/puts.c:7
#4  0x080491b3 in main () at userapps/hello.c:5
(gdb) si
(gdb) 
```



```
/home/vlad/tilck/kernel/arch/i386/fault_handlers.S
>45             fault_with_err_code 14 # Page Fault Exception
 46
 47             fault 15 # Reserved Exception
 48             fault 16 # Floating Point Exception
 49             fault 17 # Alignment Check Exception
 50             fault 18 # Machine Check Exception
 51
 52             fault 19
 53             fault 20
 54             fault 21
 55             fault 22

>0xc0101cde <fault14+2>  jmp    0xc0101e02 <asm_fault_entry>
 0xc0101ce3 <fault15>    push   $0x0
 0xc0101ce5 <fault15+2>  push   $0xf
 0xc0101ce7 <fault15+4>  jmp    0xc0101e02 <asm_fault_entry>
 0xc0101cec <fault16>    push   $0x0
 0xc0101cee <fault16+2>  push   $0x10
 0xc0101cf0 <fault16+4>  jmp    0xc0101e02 <asm_fault_entry>
 0xc0101cf5 <fault17>    push   $0x0
 0xc0101cf7 <fault17+2>  push   $0x11
 0xc0101cf9 <fault17+4>  jmp    0xc0101e02 <asm_fault_entry>
 0xc0101cfe <fault18>    push   $0x0
 0xc0101d00 <fault18+2>  push   $0x12

remote Thread 1.1 In: fault14                                        L45   PC: 0xc0101cde
0xc0101cde in ?? ()
(gdb) p $eax
$6 = 54
(gdb) p/x $eax
$7 = 0x36
(gdb) file ./build/tilck_unstripped
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "./build/tilck_unstripped"? (y or n) y
Reading symbols from ./build/tilck_unstripped...
Error in re-setting breakpoint 2: No source file named __init_tls.c.
Error in re-setting breakpoint 3: No source file named __set_thread_area.s.
(gdb)
```

Yep, page fault.



```
/home/vlad/tilck/kernel/arch/i386/paging.c
                panic("PAGE FAULT in attempt to %s %p from %s%s\nEIP: %p [%s + %d]\n",
                      rw ? "WRITE" : "READ",
                      vaddr,
                      "kernel",
                      !p ? " (NON present)." : ".",
                      r->eip, sym_name ? sym_name : "???", off);
          }

          void handle_page_fault_int(regs_t *r)
          {
                u32 vaddr;
                asmVolatile("movl %%cr2, %0" : "=r"(vaddr));

                bool p  = !!(r->err_code & PAGE_FAULT_FL_PRESENT);
                bool rw = !!(r->err_code & PAGE_FAULT_FL_RW);
                bool us = !!(r->err_code & PAGE_FAULT_FL_US);
>               int sig = SIGSEGV;
                struct user_mapping *um;

                if (!us) {
                   /*
                    * Tilck does not support kernel-space page faults caused by the kernel,
                    * while it allows user-space page faults caused by kernel (CoW pages).
                    * Therefore, such a fault is necessary caused by a bug.
remote Thread 1.1 In: handle_page_fault_int                                        L240   PC: 0xc0103e55
(gdb) n
(gdb) s
handle_page_fault_int (r=0xf8032fa8) at /home/vlad/tilck/kernel/arch/i386/paging.c:235
(gdb) n
(gdb) p p
$8 = false
(gdb) p rw
$9 = false
(gdb) p us
$10 = true
(gdb) p/x vaddr
$11 = 0x10
(gdb)
```

Vaddr is clearly just 0x10 because the GDT selector 0x23 has offset = 0 (flat segmentation)

Lesson learned

- Often, we cannot cheat.

Lesson learned

- Often, we cannot cheat.
- Even basic I/O functions use TLS variables.

Lesson learned

- Often, we cannot cheat.
- Even basic I/O functions use TLS variables.
- Had to provide a fully-functional implementation for **set_thread_area()**, in order run even single-threaded libmusl applications.

```c
int sys_set_thread_area(void *arg)
{
    int rc = 0;
    struct gdt_entry e = {0};
    struct user_desc dc;
    struct user_desc *ud = arg;

    rc = copy_from_user(&dc, ud, sizeof(struct user_desc));

    if (rc != 0)
        return -EFAULT;

    disable_preemption();

    if (!(dc.flags == USER_DESC_FLAGS_EMPTY && !dc.base_addr && !dc.limit)) {
        gdt_set_entry(&e, dc.base_addr, dc.limit, 0, 0);
        e.s = 1;
        e.dpl = 3;
        e.d = dc.seg_32bit;
        e.type |= (dc.contents << 2);
        e.type |= !dc.read_exec_only ? GDT_ACCESS_RW : 0;
        e.g = dc.limit_in_pages;
        e.avl = dc.useable;
        e.p = !dc.seg_not_present;
    } else {
        /* The user passed an empty descriptor: entry_number cannot be -1 */
        if (dc.entry_number == INVALID_ENTRY_NUM) {
            rc = -EINVAL;
            goto out;
        }
    }

    if (dc.entry_number == INVALID_ENTRY_NUM) {

        int slot = find_available_slot_in_user_task();
        if (slot < 0) {
            rc = -ESRCH;
            goto out;
        }

        dc.entry_number = (u32)gdt_add_entry(&e);
        if (dc.entry_number == INVALID_ENTRY_NUM) {

            rc = gdt_expand();

            if (rc < 0) {
                rc = -ESRCH;
                goto out;
            }
            dc.entry_number = (u32)gdt_add_entry(&e);
            ASSERT(dc.entry_number != INVALID_ENTRY_NUM);
        }
        gdt_set_slot(get_curr_proc(), (u16)slot, (u16)dc.entry_number);
        goto out;
    }

    /* Handling the case where the user specified a GDT entry number */

    int slot = get_user_task_slot_for_gdt_entry(dc.entry_number);

    if (slot < 0) {
        /* A GDT entry with that index has never been allocated by this task */

        if (dc.entry_number >= gdt_size || gdt[dc.entry_number].access) {
            /* The entry is out-of-bounds or it's used by another task */
            rc = -EINVAL;
            goto out;
        }

        /* The entry is available, now find a slot */
        slot = find_available_slot_in_user_task();

        if (slot < 0) {
            /* Unable to find a free slot in this struct task struct */
            rc = -ESRCH;
            goto out;
        }

        gdt_set_slot(get_curr_proc(), (u16)slot, (u16)dc.entry_number);
    }

    ASSERT(dc.entry_number < gdt_size);

    set_entry_num(dc.entry_number, &e);

    /*
     * We're here because either we found a slot already containing this index
     * (therefore it must be valid) or the index in-bounds and it is free.
     */

out:
    enable_preemption();

    if (!rc) {

        /*
         * Positive case: we get here with rc = SUCCESS, now flush back the
         * the struct user_desc (we might have changed its entry_number).
         */
        rc = copy_to_user(ud, &dc, sizeof(struct user_desc));

        if (rc < 0)
            rc = -EFAULT;
    }

    return rc;
}
```

That was quite some code, but it’s not enough. We need a **ref-count** for GDT entries as well.

Why? Think about **fork().** What happens if the parent dies before the child and we free the GDT slots?

```c
void
arch_specific_new_proc_setup(struct process *pi, struct process *parent)
{
    arch_proc_members_t *arch = get_proc_arch_fields(pi);

    if (!parent)
        return;       /* we're done */

    memcpy(&pi->pi_arch, &parent->pi_arch, sizeof(pi->pi_arch));

    if (arch->ldt)
        gdt_entry_inc_ref_count(arch->ldt_index_in_gdt);

    for (int i = 0; i < ARRAY_SIZE(arch->gdt_entries); i++)
        if (arch->gdt_entries[i])
            gdt_entry_inc_ref_count(arch->gdt_entries[i]);

    pi->set_child_tid = NULL;
}
```

```c
void
arch_specific_free_proc(struct process *pi)
{
    arch_proc_members_t *arch = get_proc_arch_fields(pi);

    if (arch->ldt) {
        gdt_clear_entry(arch->ldt_index_in_gdt);
        arch->ldt = NULL;
    }

    for (int i = 0; i < ARRAY_SIZE(arch->gdt_entries); i++) {
        if (arch->gdt_entries[i]) {
            gdt_clear_entry(arch->gdt_entries[i]);
            arch->gdt_entries[i] = 0;
        }
    }
}
```

ACPICA & AcpiOsWaitSemaphore()

- ACPICA requires the OSL to provide a counting semaphore implementation capable of waiting and signaling N units.
- That is weird requirement.
- It could be trivially implemented on the top of a regular counting semaphore, but that would be extremely inefficient.
- I implemented such a semaphore in Tilck.

Classic semaphore

```c
void ksem_wait(struct ksem *s)
{
   struct task *curr = get_curr_task();
   disable_preemption();

   if (--s->counter < 0) {

      task_set_wait_obj(curr, WOBJ_SEM, s, NO_EXTRA, &s->wait_list);
      enable_preemption_nosched();
      kernel_yield();
      return;
   }

   enable_preemption();
}
```

New semaphore [1/2]

```c
int ksem_wait(struct ksem *s, int units, int timeout_ticks)
{
   int rc = -ETIME;
   ASSERT(units > 0);

   if (s->max != KSEM_NO_MAX && units > s->max)
      return -EINVAL;

   disable_preemption();
   {
      if (timeout_ticks != KSEM_NO_WAIT) {
         u64 start_ticks, end_ticks;

         if (timeout_ticks > 0) {

            start_ticks = get_ticks();
            end_ticks = start_ticks + (u32)timeout_ticks;

            if (s->counter < units)
               task_set_wakeup_timer(get_curr_task(), (u32)timeout_ticks);
         }

         while (s->counter < units) {

            if (timeout_ticks > 0 && get_ticks() >= end_ticks)
               break;

            prepare_to_wait_on(WOBJ_SEM, s, (u32)units, &s->wait_list);
            enter_sleep_wait_state(); /* after that, preemption will be enabled */
            disable_preemption();
         }

         if (timeout_ticks > 0)
            task_cancel_wakeup_timer(get_curr_task());
      }

      if (s->counter >= units) {
         s->counter -= units;
         rc = 0;
      }
   }
   enable_preemption();
   return rc;
}
```

Classic semaphore

```c
void ksem_signal(struct ksem *s)
{
   disable_preemption();

   if (s->counter++ < 0) {

      ASSERT(!list_is_empty(&s->wait_list));

      struct wait_obj *task_wo =
         list_first_obj(&s->wait_list, struct wait_obj, wait_list_node);

      struct task *ti = CONTAINER_OF(task_wo, struct task, wobj);
      task_reset_wait_obj(ti);
   }

   enable_preemption();
}
```

New semaphore [2/2]

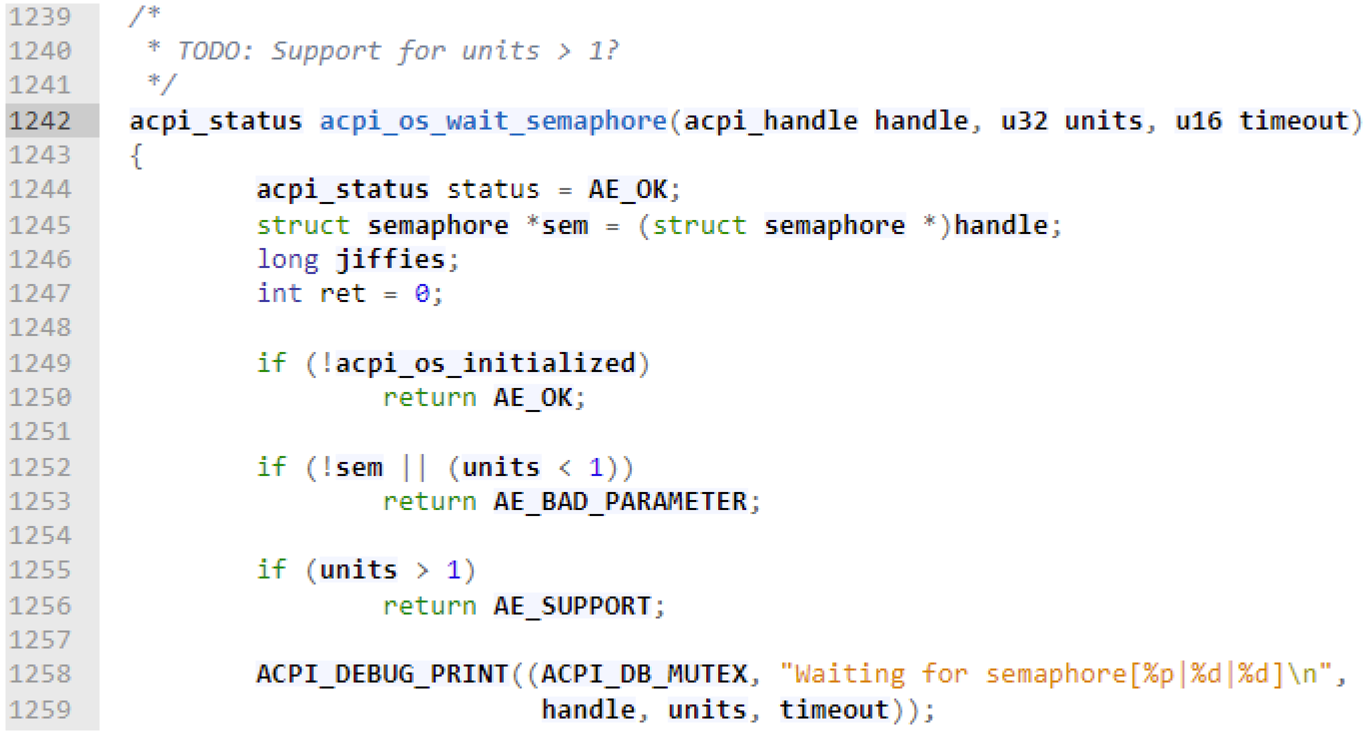
```c
int ksem_signal(struct ksem *s, int units)
{
   struct wait_obj *wo, *tmp;
   int rem_counter, rc = 0;
   ASSERT(units > 0);

   disable_preemption();

   if (s->max != KSEM_NO_MAX) {

      if (units > s->max) {
         rc = -EINVAL;
         goto out;
      }

      if (s->counter > s->max - units) {
         rc = -EDQUOT;
         goto out;
      }
   }

   s->counter += units;
   rem_counter = s->counter;

   list_for_each(wo, tmp, &s->wait_list, wait_list_node) {

      if (rem_counter <= 0)
         break; /* not enough units to unblock anybody */

      int wait_units = (int)wo->extra;

      if (wait_units <= rem_counter) {
         struct task *ti = CONTAINER_OF(wo, struct task, wobj);
         rem_counter -= wait_units;
         wake_up(ti);
      }
   }

out:
   enable_preemption();
   return rc;
}
```

But.. how Linux did implement the counting semaphore to make ACPICA happy?

But.. how Linux did implement the counting semaphore to make ACPICA happy?

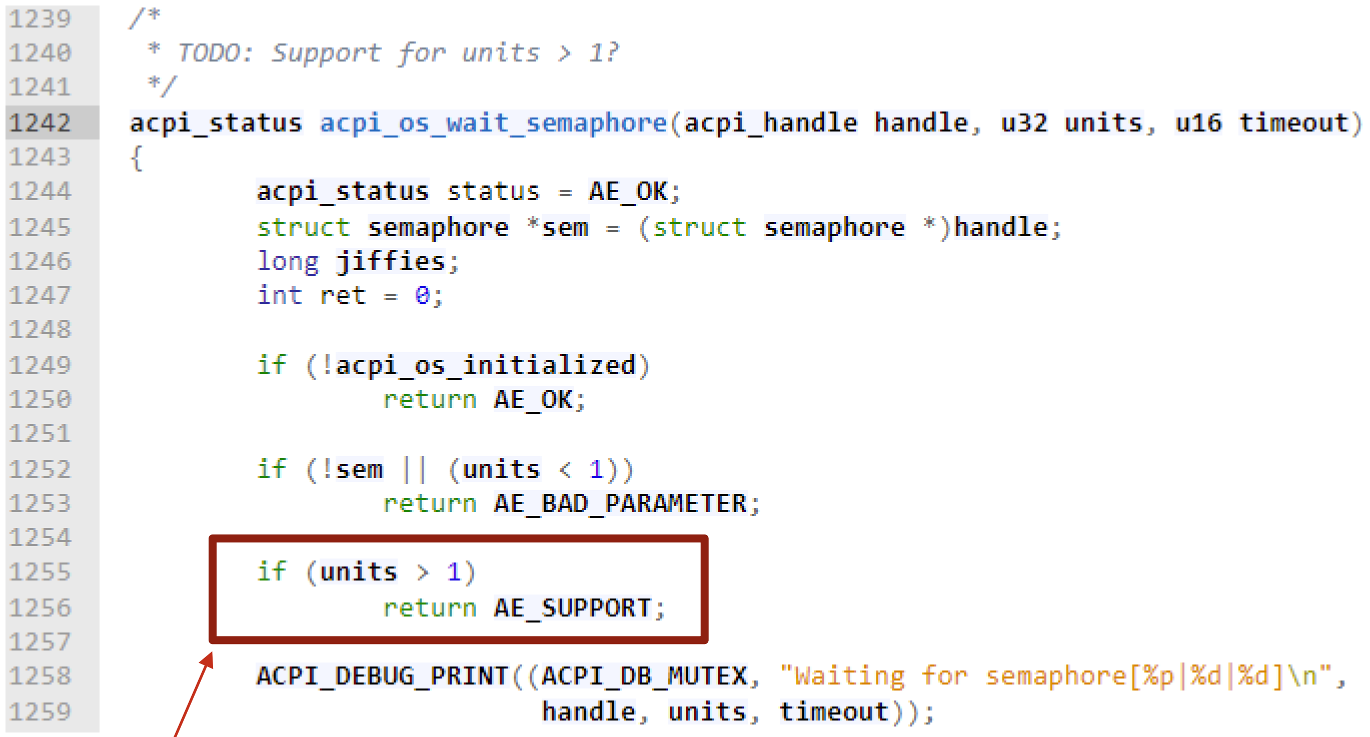
It didn’t ☺

But.. how Linux did implement the counting semaphore to make ACPICA happy?



```
/*
 * TODO: Support for units > 1?
 */
acpi_status acpi_os_wait_semaphore(acpi_handle handle, u32 units, u16 timeout)
{
        acpi_status status = AE_OK;
        struct semaphore *sem = (struct semaphore *)handle;
        long jiffies;
        int ret = 0;

        if (!acpi_os_initialized)
                return AE_OK;

        if (!sem || (units < 1))
                return AE_BAD_PARAMETER;

        if (units > 1)
                return AE_SUPPORT;

        ACPI_DEBUG_PRINT((ACPI_DB_MUTEX, "Waiting for semaphore[%p|%d|%d]\n",
                          handle, units, timeout));
```

But.. how Linux did implement the counting semaphore to make ACPICA happy?



```
/*
 * TODO: Support for units > 1?
 */
acpi_status acpi_os_wait_semaphore(acpi_handle handle, u32 units, u16 timeout)
{
        acpi_status status = AE_OK;
        struct semaphore *sem = (struct semaphore *)handle;
        long jiffies;
        int ret = 0;

        if (!acpi_os_initialized)
                return AE_OK;

        if (!sem || (units < 1))
                return AE_BAD_PARAMETER;

        if (units > 1)
                return AE_SUPPORT;

        ACPI_DEBUG_PRINT((ACPI_DB_MUTEX, "Waiting for semaphore[%p|%d|%d]\n",
                          handle, units, timeout));
```

Sometimes cheating works.

Thank you!

https://github.com/vvaltchev/tilck

Tilck
A Tiny Linux-Compatible Kernel

Vladislav K. Valtchev (2022)