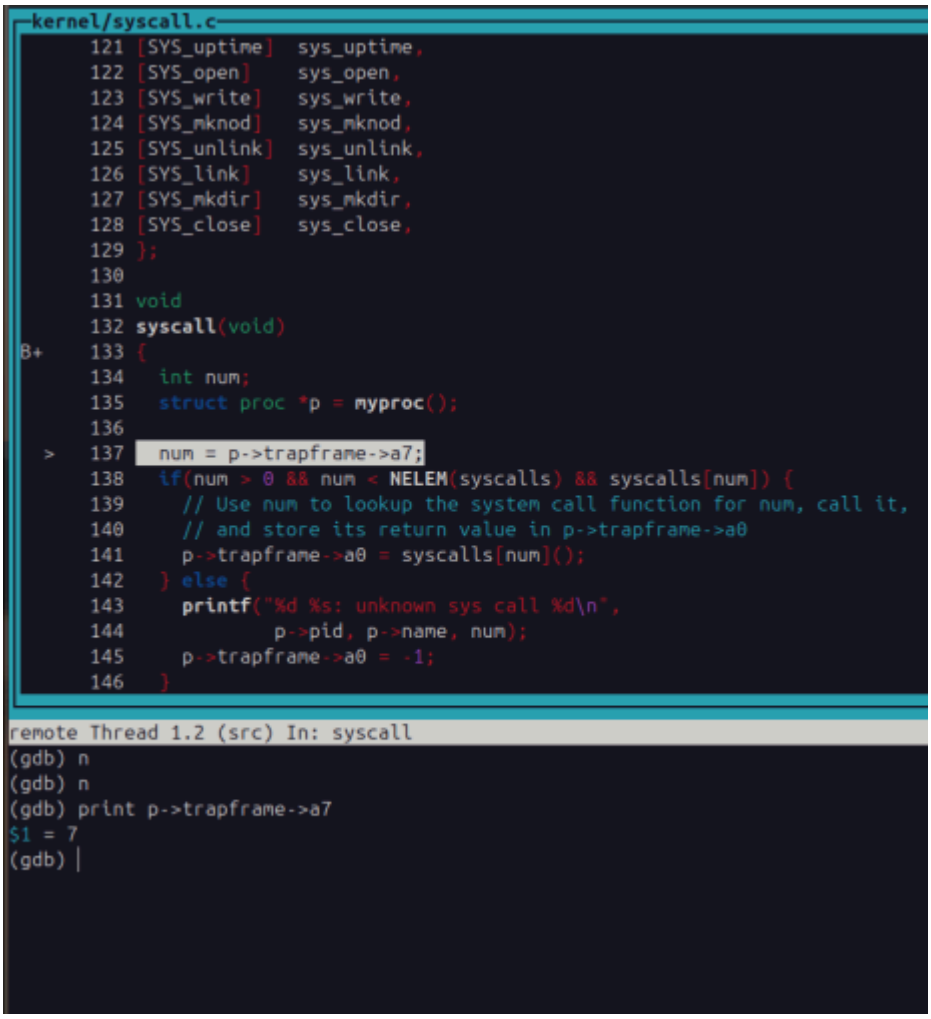
GDB

1. Looking at the backtrace output, which function called syscall?

```
(gdb) backtrace
#0  syscall () at kernel/syscall.c:133
#1  0x0000000080001a3e in usertrap () at kernel/trap.c:67
#2  0x0505050505050505 in ?? ()
(gdb)
```

The backtrace above indicates that the syscall is called by **usertrap()** in **kernel/trap.c**, line 67.

2. What is the value of p->trapframe->a7 and what does that value represent? (Hint: look user/initcode.S, the first user program xv6 starts.)



The value of **p->trapframe->a7** is **7**. Let's determine the reason behind this.

```
# exec(init, argv)
.globl start
start:
        la a0, init
        la a1, argv
        li a7, SYS_exec
        ecall
```

syscall.c | proc.c | initcode.S | syscall.h

kernel > syscall.h > ...

Frans Kaashoek, 13 years ago | 2 authors (F

```
1  // System call numbers                rsc, 18 ye
2  #define SYS_fork    1
3  #define SYS_exit    2
4  #define SYS_wait    3
5  #define SYS_pipe    4
6  #define SYS_read    5
7  #define SYS_kill    6
8  #define SYS_exec    7
9  #define SYS_fstat   8
10 #define SYS_chdir   9
11 #define SYS_dup     10
12 #define SYS_getpid  11
13 #define SYS_sbrk    12
14 #define SYS_sleep   13
15 #define SYS_uptime  14
16 #define SYS_open    15
17 #define SYS_write   16
18 #define SYS_mknod   17
19 #define SYS_unlink  18
20 #define SYS_link    19
21 #define SYS_mkdir   20
22 #define SYS_close   21
23
```

In **initcode.S**, the value of **SYS_exec** is loaded into register a7 before switching to kernel mode with the **ecall** instruction (as introduced in Chapter 2, Section 2.2).

Thus, the value of **p->trapframe->a7** represents the **SYS_exec** system call number.

3. What was the previous mode that the CPU was in?

```
(gdb) p /x $sstatus
$1 = 0x200000022
```

The eighth bit in **$sstatus** (which is 0), as defined in the book, indicates the previous mode. In this case, it is user mode.

4.1.1 Supervisor Status Register (sstatus)

The `sstatus` register is an SXLEN-bit read/write register formatted as shown in Figure 4.1 when SXLEN=32 and Figure 4.2 when SXLEN=64. The `sstatus` register keeps track of the processor's current operating state.

The SPP bit indicates the privilege level at which a hart was executing before entering supervisor mode. When a trap is taken, SPP is set to 0 if the trap originated from user mode, or 1 otherwise. When an SRET instruction (see Section 3.3.2) is executed to return from the trap handler, the

4. Write down the assembly instruction the kernel is panicing at. Which register corresponds to the variable num?

```
→  xv6-labs-2024 git:(util) ✗ make qemu-gdb
*** Now run 'gdb' in another window.
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographi
c -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -devic
e virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 -S -gdb tcp::26000

xv6 kernel is booting

hart 1 starting
hart 2 starting
scause=0xd sepc=0x80001c92 stval=0x0
panic: kerneltrap
```

Here, **spec** shows the address where the kernel is panicking. I searched in **kernel/kernel.asm** for the address **0x80001c92** and found the following:

```
4304    // num = p->trapframe->a7;
4305    num = *(int*)0;
4306      80001c92:  00002683          lw  a3,0(zero) # 0 <_entry-0x80000000>
4307    if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
```

As you can see, the corresponding assembly instruction for

num = p->trapframe->a7; is **lw a3, 0(zero)**.

This assembly code means: load a word (4 bytes) from address 0 in memory into the register a3. So **a3** corresponds to the variable **num**.

5. Why does the kernel crash? Hint: look at figure 3-3 in the text; is address 0 mapped in the kernel address space? Is that confirmed by the value in scause above? (See description of scause in RISC-V privileged instructions)

The kernel crashed because it attempted to load data from an unused memory address (0). Address 0 is not mapped into kernel space, which starts from 0x80000000.

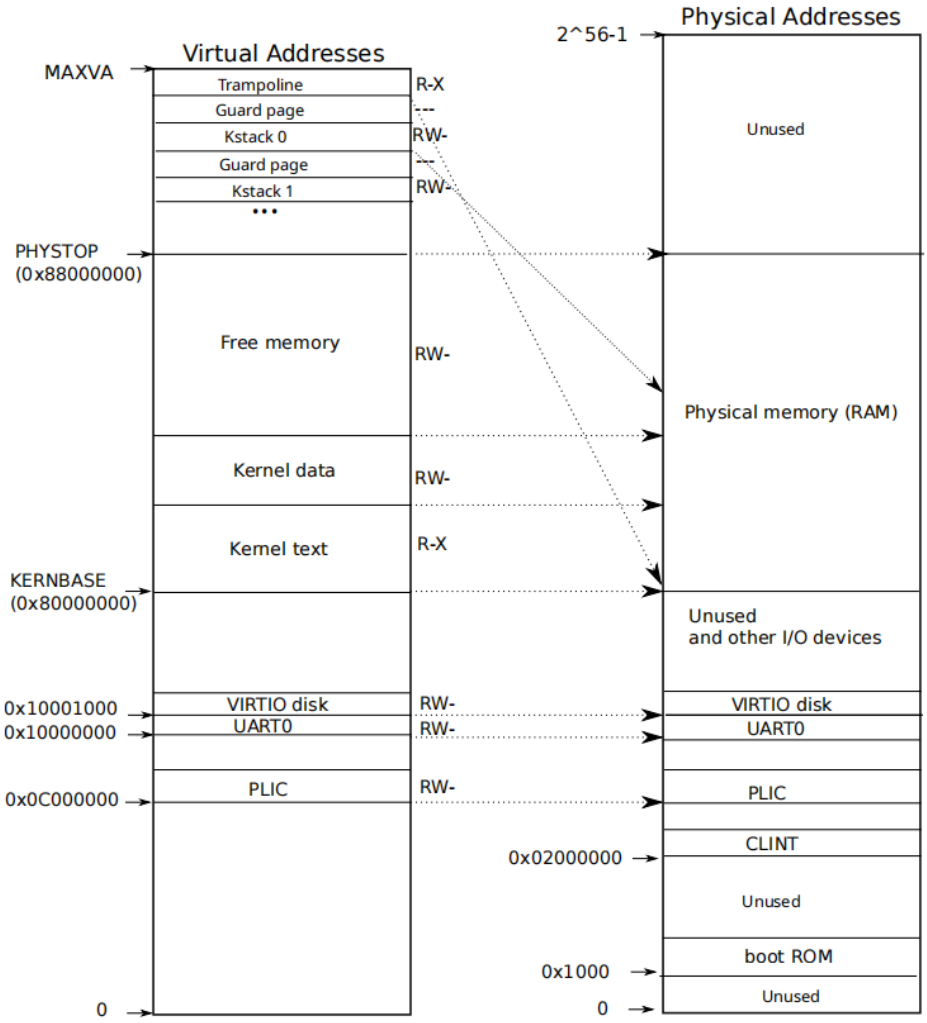


Figure 3.3: On the left, xv6's kernel address space. RWX refer to PTE read, write, and execute permissions. On the right, the RISC-V physical address space that xv6 expects to see.

To confirm this, I will investigate the **scause** register, which indicates the reason for the kernel panic.


```
(Ctrl + C to exit)
```

The value of scause is **13** (actually shown in the program output window as **0xd**).

Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2–4	*Reserved*
1	5	Supervisor timer interrupt
1	6–8	*Reserved*
1	9	Supervisor external interrupt
1	10–15	*Reserved*
1	≥16	*Designated for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10–11	*Reserved*
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	16–23	*Reserved*
0	24–31	*Designated for custom use*
0	32–47	*Reserved*
0	48–63	*Designated for custom use*
0	≥64	*Reserved*

Since 13 represents a **Load page fault**, it indicates that there was an error when attempting to load data from memory address 0 into a register.

Therefore, the initial statement is confirmed by the exception code in scause.

6. What is the name of the binary that was running when the kernel paniced? What is its process id (pid)?

Name of binary is "initcode\000\000\000\000\000\000\000". It's process id is 1.