INTERNALS SESSIONS 01: Syscalls Journey in Linux

Abolfazl Kazemi

Mahsan

Your Support In Digital World

mahsan.co
info@mahsan.co

Agenda

1 Setting up the Lab

2 Tracing system calls in User mode

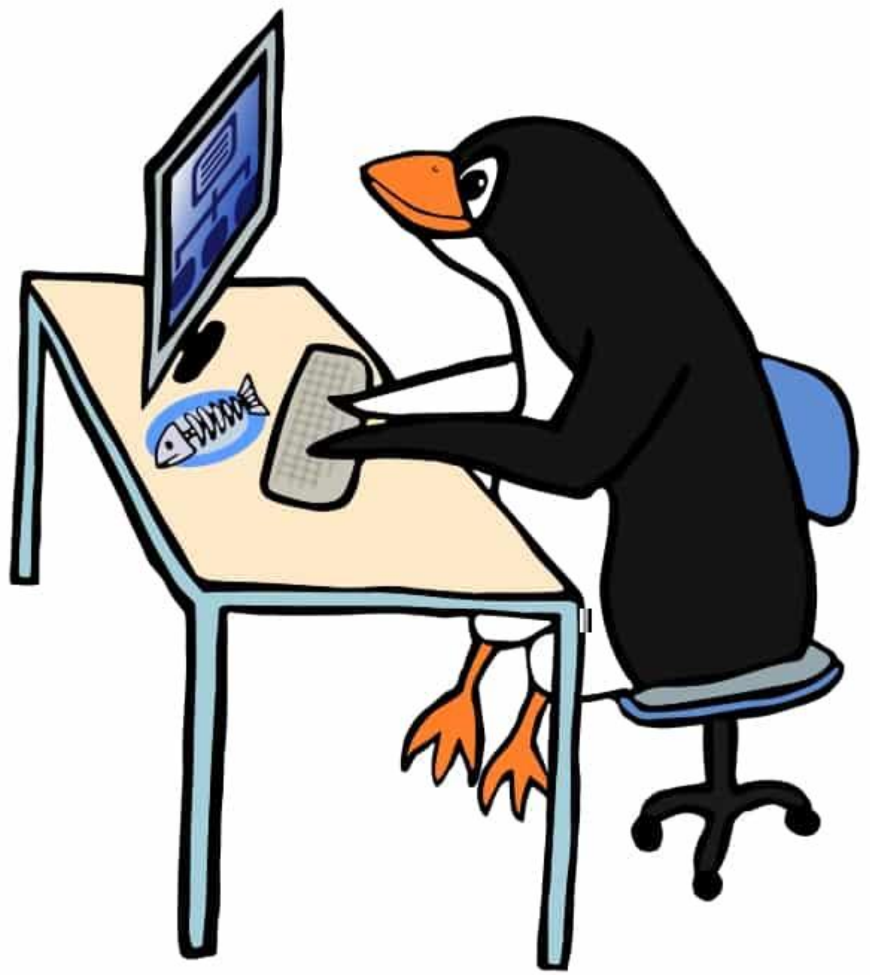
3 Intel’s syscall instruction

4 System Call Table and how to add a new syscall

5 Tracing a system call inside Kernel

مهسان
تکیه گاه شما
در دنیای هوشمند

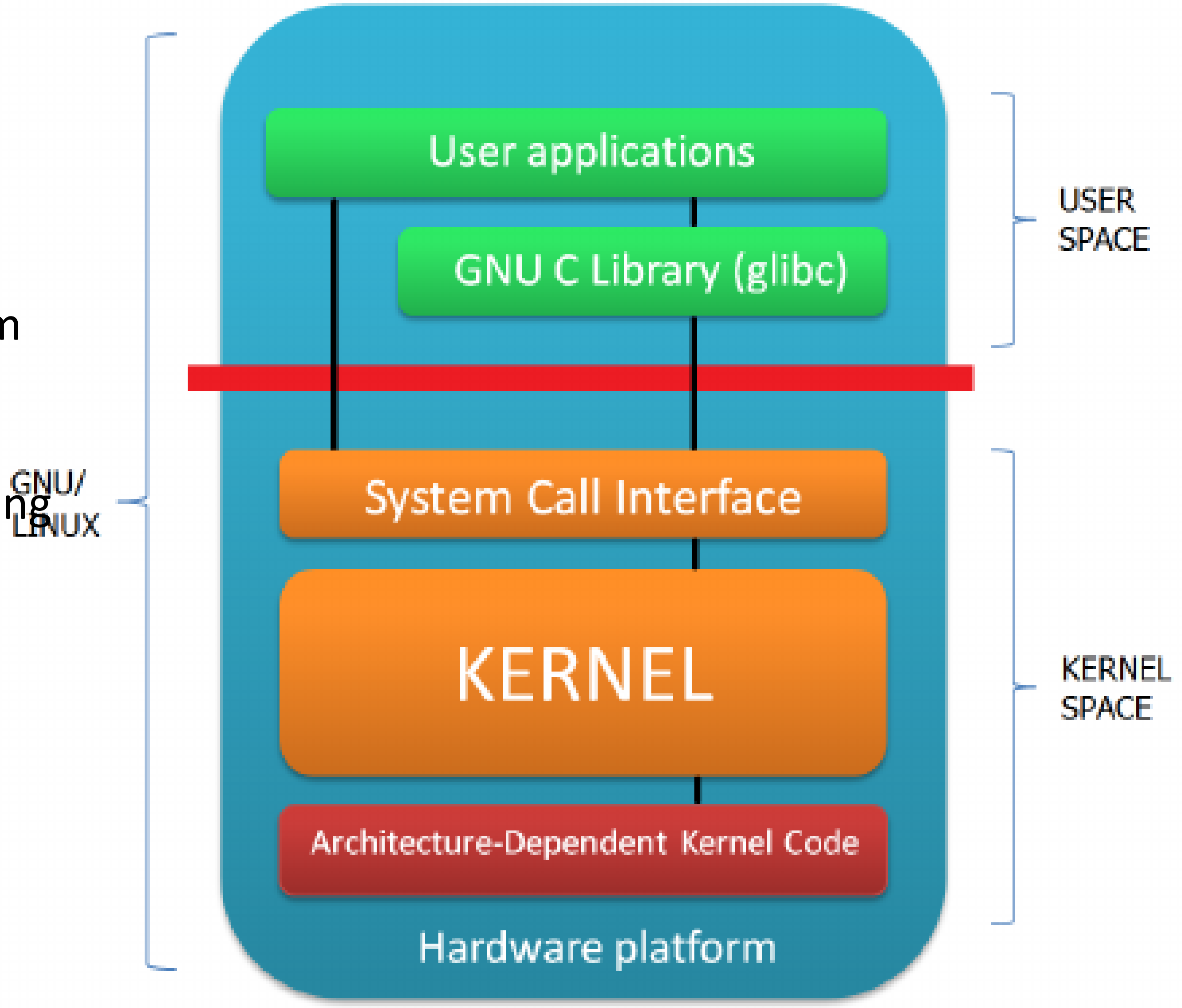
Back to the Basics: What is Linux?



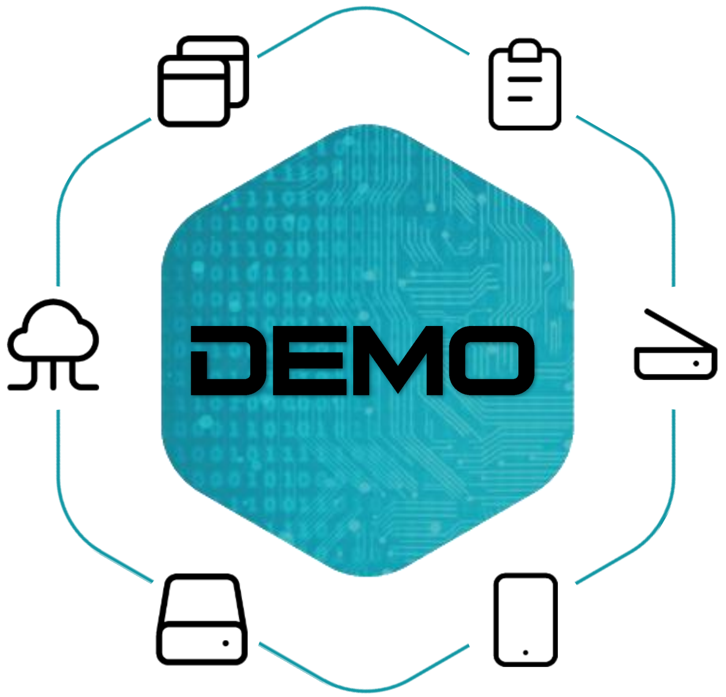
User vs Kernel Space

- ❑ User mode
  - Allows access to non-operating system code & data only
  - No access to the hardware
  - Protects user applications from crashing the system

- ❑ Kernel mode
  - Privileged mode for use by the kernel and device drivers only
  - Allows access to all system resources
  - Can potentially crash the system



Source



Tracing System Calls in user space. (ltrace/strace)

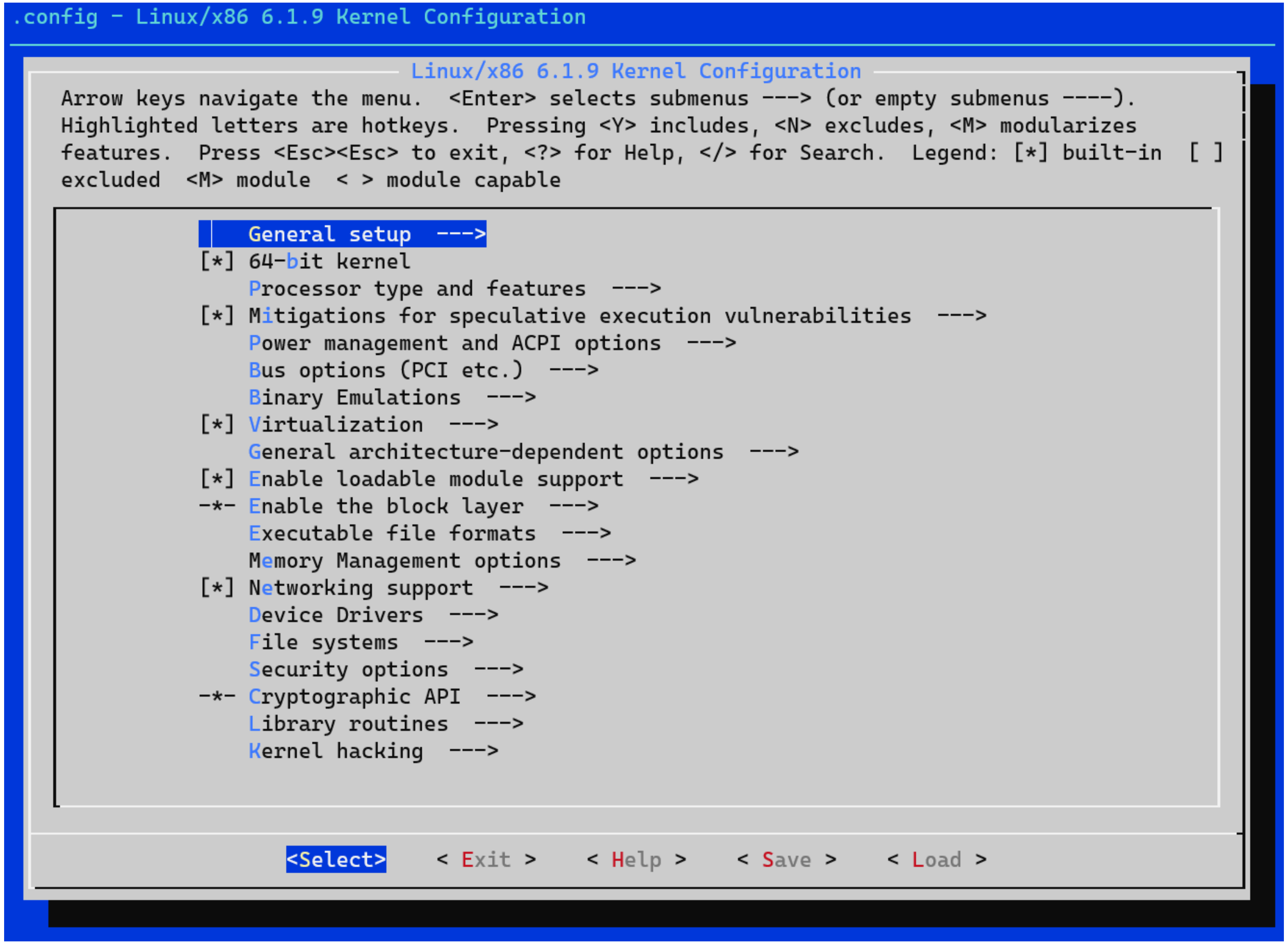
Linux Compilation

Grab linux from kernel.org:

make defconfig
make menuconfig

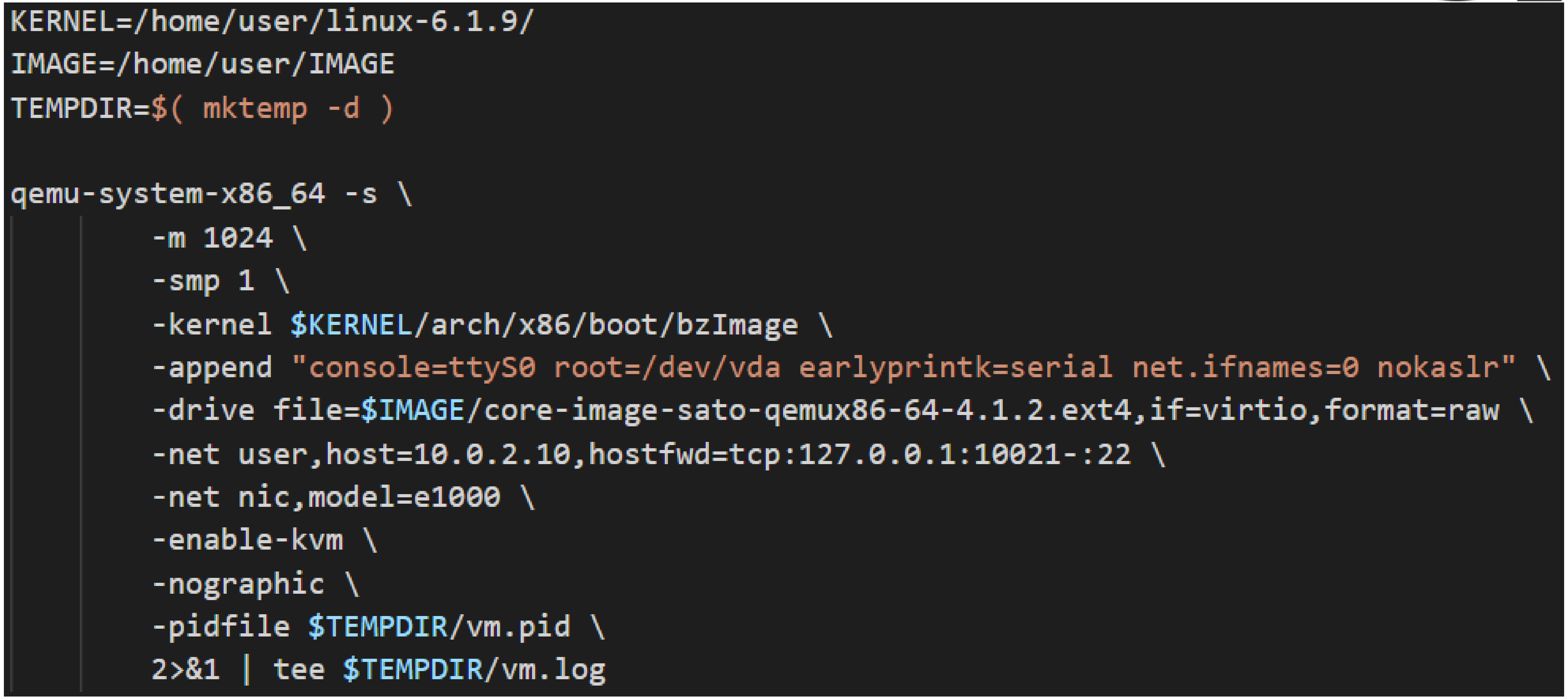
make –jN

make install
make modules_install
update-grub

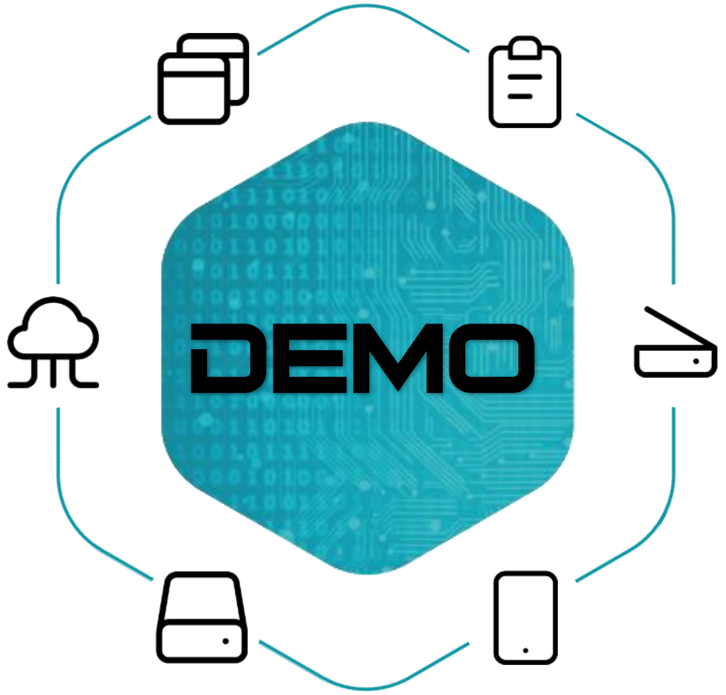


```
.config - Linux/x86 6.1.9 Kernel Configuration
                         Linux/x86 6.1.9 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

                 General setup  --->
             [*] 64-bit kernel
                 Processor type and features  --->
             [*] Mitigations for speculative execution vulnerabilities  --->
                 Power management and ACPI options  --->
                 Bus options (PCI etc.)  --->
                 Binary Emulations  --->
             [*] Virtualization  --->
                 General architecture-dependent options  --->
             [*] Enable loadable module support  --->
             -*- Enable the block layer  --->
                 Executable file formats  --->
                 Memory Management options  --->
             [*] Networking support  --->
                 Device Drivers  --->
                 File systems  --->
                 Security options  --->
             -*- Cryptographic API  --->
                 Library routines  --->
                 Kernel hacking  --->

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Running Compiled Linux (QEMU)

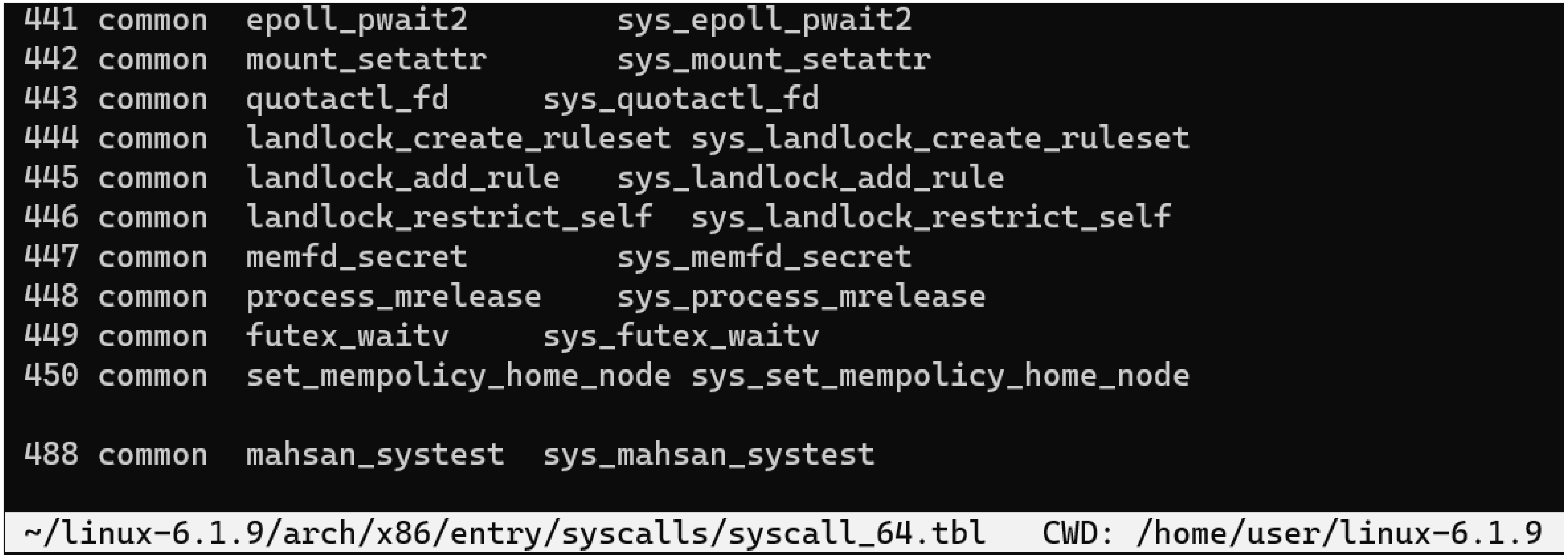


```
KERNEL=/home/user/linux-6.1.9/
IMAGE=/home/user/IMAGE
TEMPDIR=$( mktemp -d )

qemu-system-x86_64 -s \
        -m 1024 \
        -smp 1 \
        -kernel $KERNEL/arch/x86/boot/bzImage \
        -append "console=ttyS0 root=/dev/vda earlyprintk=serial net.ifnames=0 nokaslr" \
        -drive file=$IMAGE/core-image-sato-qemux86-64-4.1.2.ext4,if=virtio,format=raw \
        -net user,host=10.0.2.10,hostfwd=tcp:127.0.0.1:10021-:22 \
        -net nic,model=e1000 \
        -enable-kvm \
        -nographic \
        -pidfile $TEMPDIR/vm.pid \
        2>&1 | tee $TEMPDIR/vm.log
```



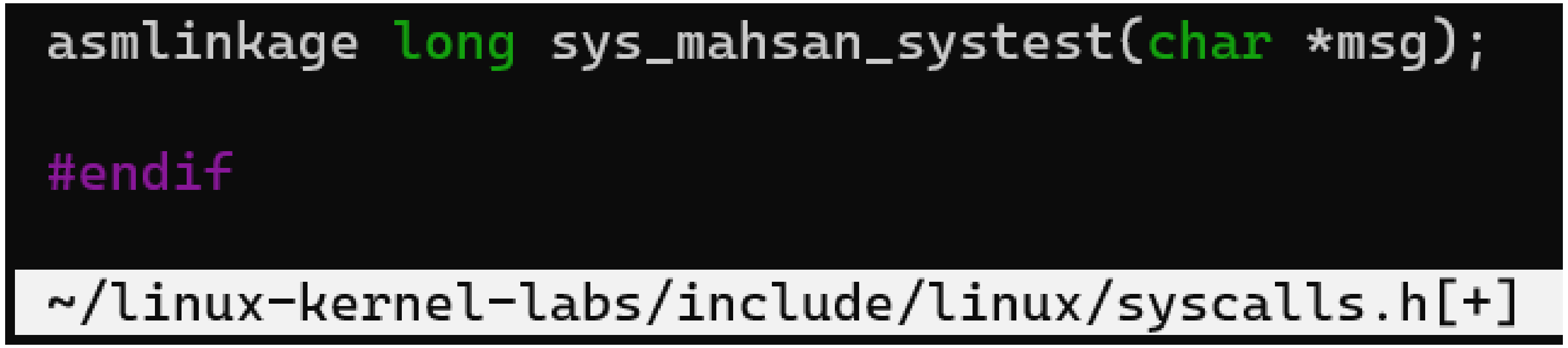
Running Linux using QEMU.

System Call Table



```
441 common  epoll_pwait2            sys_epoll_pwait2
442 common  mount_setattr           sys_mount_setattr
443 common  quotactl_fd     sys_quotactl_fd
444 common  landlock_create_ruleset sys_landlock_create_ruleset
445 common  landlock_add_rule       sys_landlock_add_rule
446 common  landlock_restrict_self  sys_landlock_restrict_self
447 common  memfd_secret            sys_memfd_secret
448 common  process_mrelease        sys_process_mrelease
449 common  futex_waitv     sys_futex_waitv
450 common  set_mempolicy_home_node sys_set_mempolicy_home_node

488 common  mahsan_systest  sys_mahsan_systest

~/linux-6.1.9/arch/x86/entry/syscalls/syscall_64.tbl   CWD: /home/user/linux-6.1.9
```

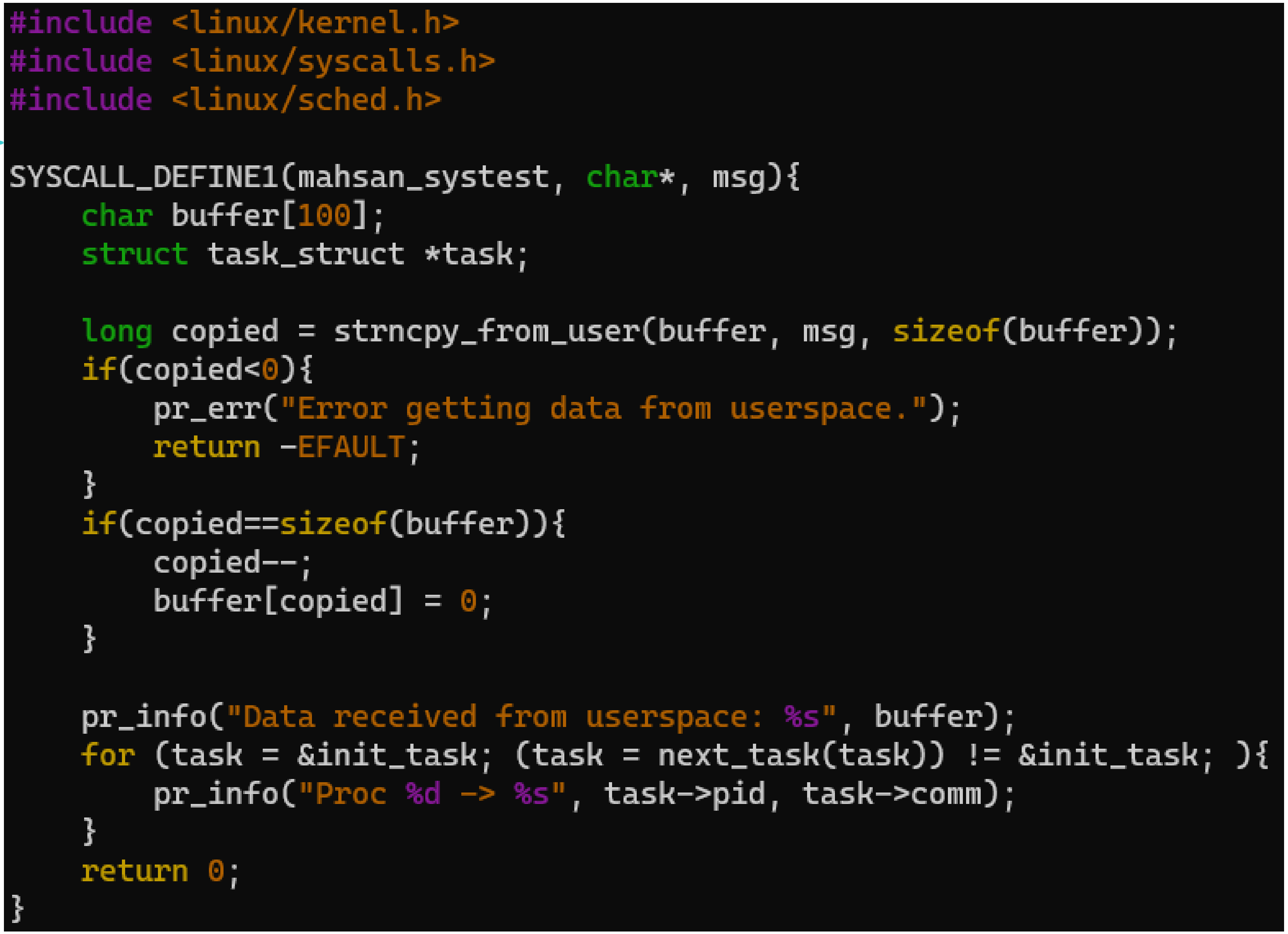
System Call Signature



```
asmlinkage long sys_mahsan_systest(char *msg);

#endif

~/linux-kernel-labs/include/linux/syscalls.h[+]
```

asmlinkage: This is a #define for some gcc magic that tells the compiler that the function should not expect to find any of its arguments in registers (a common optimization), but only on the CPU's stack. (link)

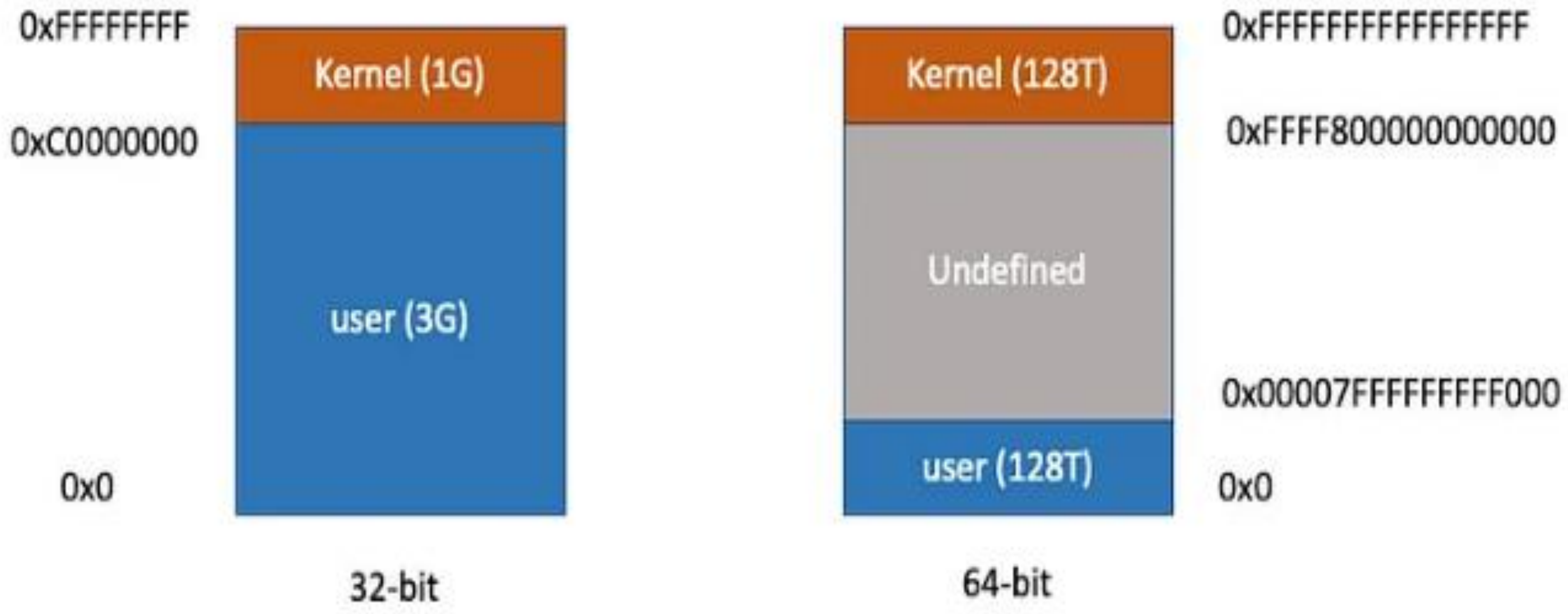
System Call Implementation



```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/sched.h>

SYSCALL_DEFINE1(mahsan_systest, char*, msg){
    char buffer[100];
    struct task_struct *task;

    long copied = strncpy_from_user(buffer, msg, sizeof(buffer));
    if(copied<0){
        pr_err("Error getting data from userspace.");
        return -EFAULT;
    }
    if(copied==sizeof(buffer)){
        copied--;
        buffer[copied] = 0;
    }

    pr_info("Data received from userspace: %s", buffer);
    for (task = &init_task; (task = next_task(task)) != &init_task; ){
        pr_info("Proc %d -> %s", task->pid, task->comm);
    }
    return 0;
}
```

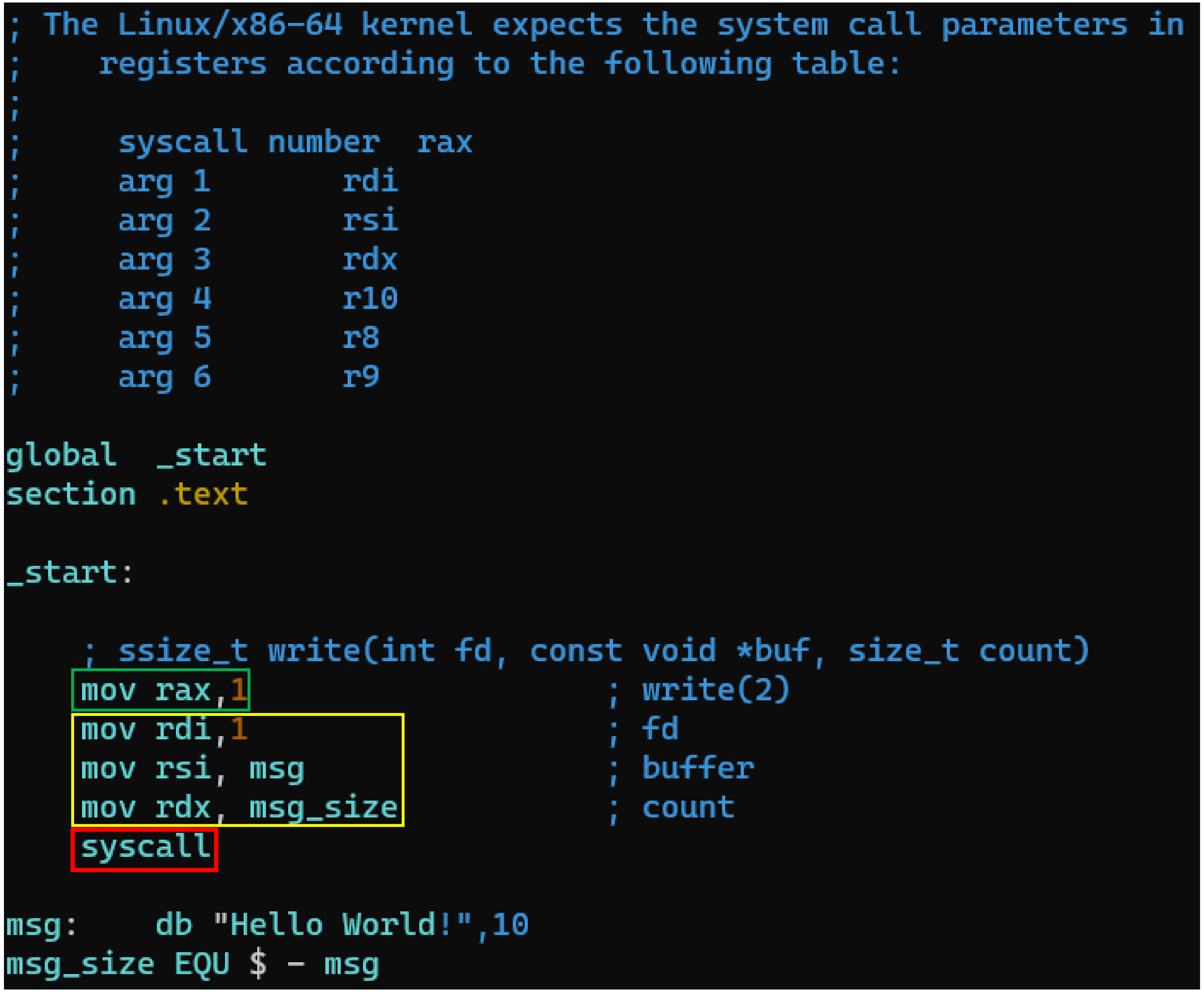
Virtual Memory

copy_from_user
copy_to_user

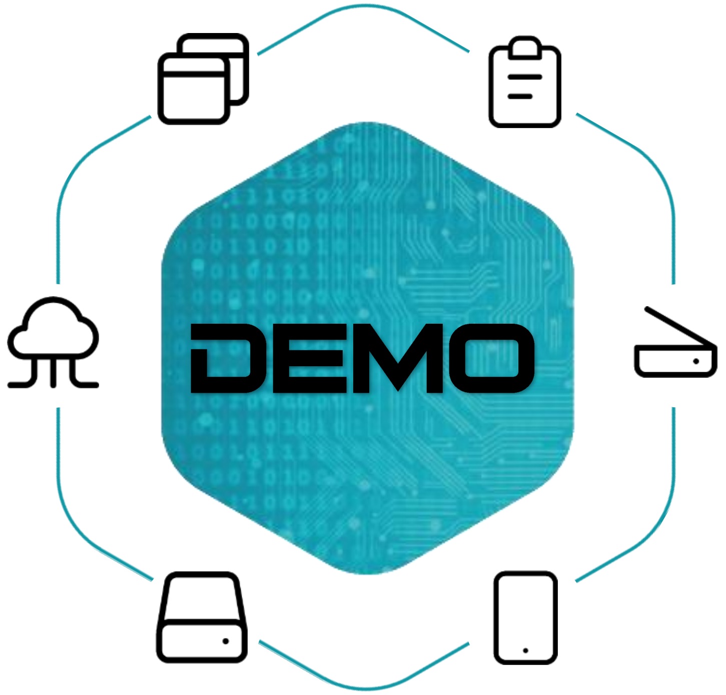


Source

Low-level Sys-calling

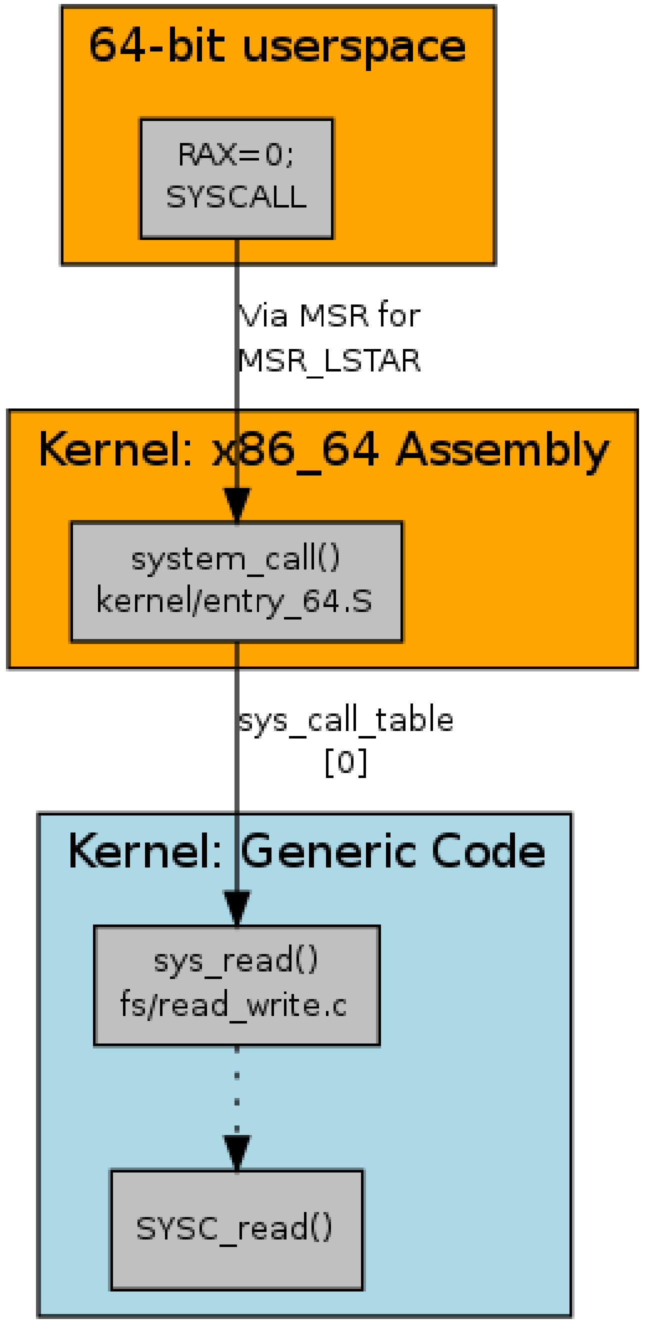


```
; The Linux/x86-64 kernel expects the system call parameters in
;    registers according to the following table:
;
;
;      syscall number  rax
;      arg 1           rdi
;      arg 2           rsi
;      arg 3           rdx
;      arg 4           r10
;      arg 5           r8
;      arg 6           r9

global  _start
section .text

_start:

    ; ssize_t write(int fd, const void *buf, size_t count)
    mov rax,1                             ; write(2)
    mov rdi,1                             ; fd
    mov rsi, msg                          ; buffer
    mov rdx, msg_size                     ; count
    syscall

msg:    db "Hello World!",10
msg_size EQU $ - msg
```



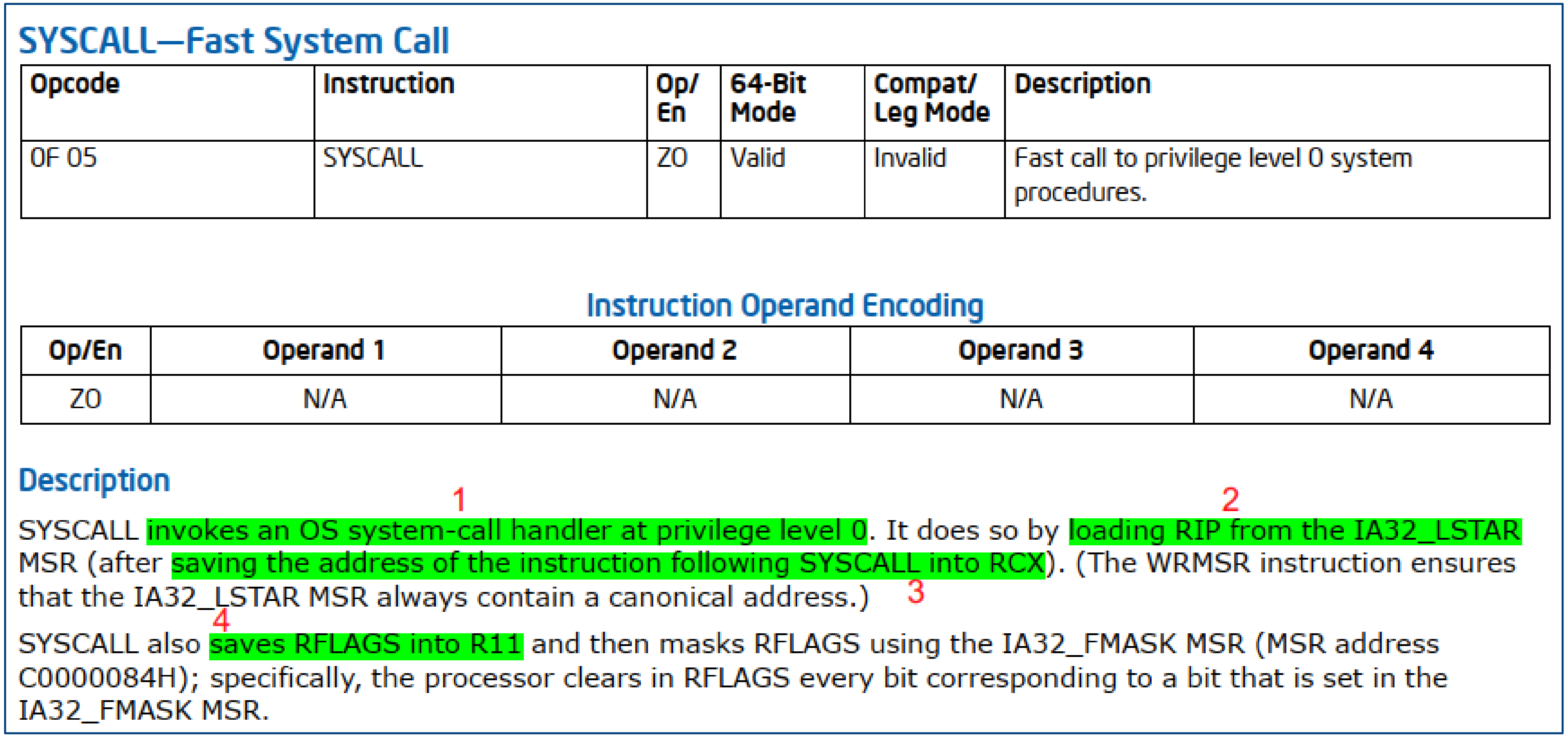
Executing our custom system call and tracing kernel functions using ftrace.

x64 Syscalls Invocation



Source

syscall instruction



SYSCALL—Fast System Call

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 05	SYSCALL	ZO	Valid	Invalid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	N/A	N/A	N/A	N/A

Description

1 2

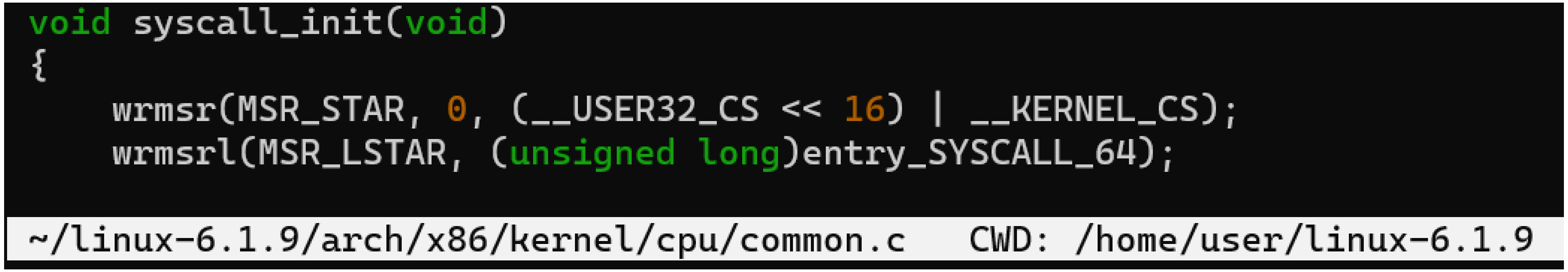
SYSCALL invokes an OS system-call handler at privilege level 0. It does so by loading RIP from the IA32_LSTAR MSR (after saving the address of the instruction following SYSCALL into RCX). (The WRMSR instruction ensures that the IA32_LSTAR MSR always contain a canonical address.) 3

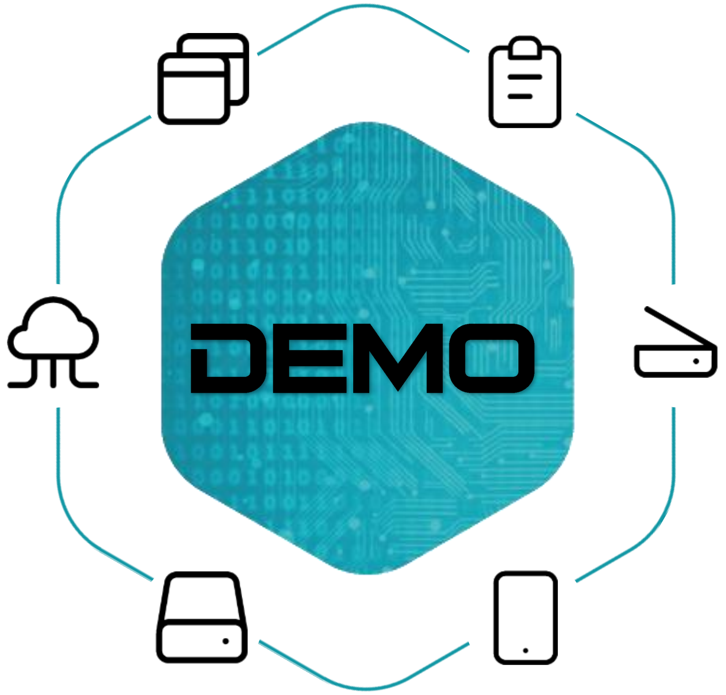
4

SYSCALL also saves RFLAGS into R11 and then masks RFLAGS using the IA32_FMASK MSR (MSR address C0000084H); specifically, the processor clears in RFLAGS every bit corresponding to a bit that is set in the IA32_FMASK MSR.

Source: Intel Software Developer's Manual

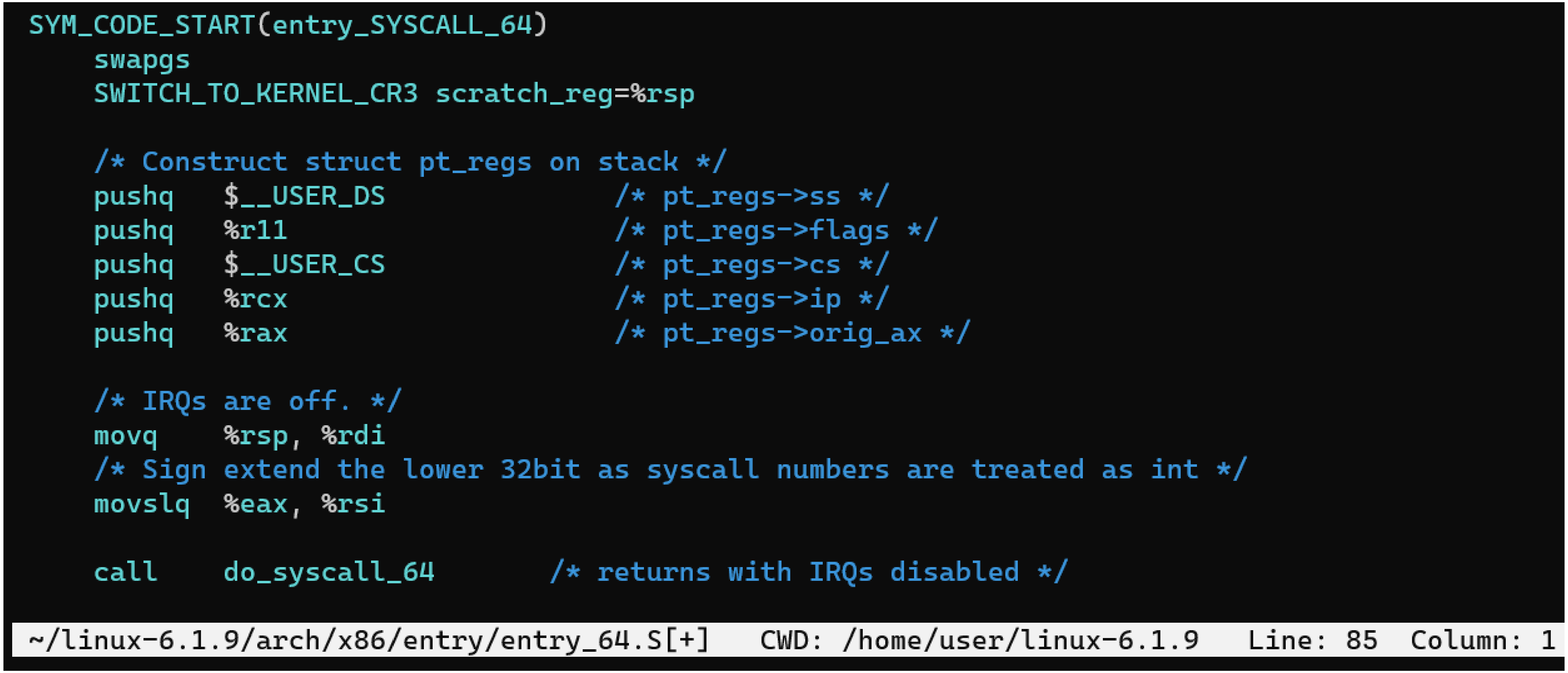
Syscalls Initialization


```
void syscall_init(void)
{
    wrmsr(MSR_STAR, 0, (__USER32_CS << 16) | __KERNEL_CS);
    wrmsrl(MSR_LSTAR, (unsigned long)entry_SYSCALL_64);

~/linux-6.1.9/arch/x86/kernel/cpu/common.c   CWD: /home/user/linux-6.1.9
```



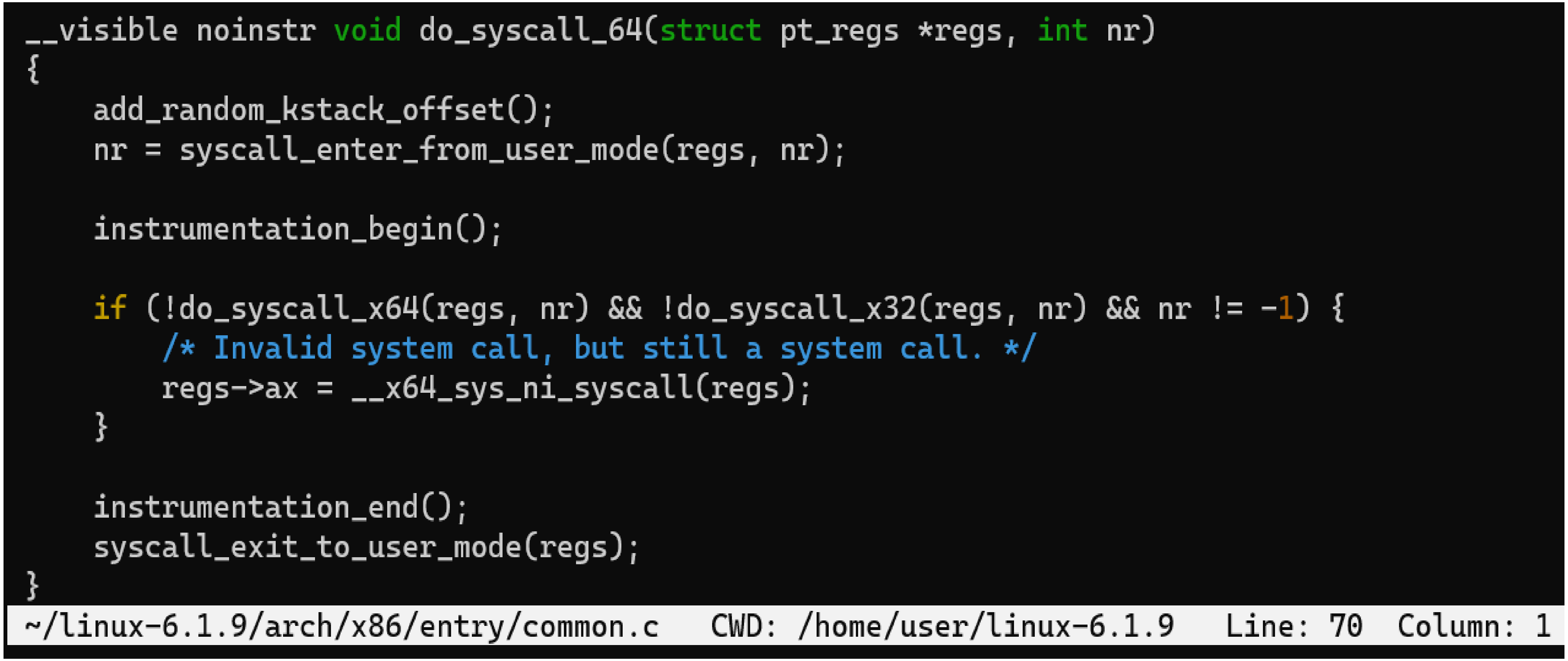
System Call initialization in Linux and the value of MSR_LSTAR register.

Syscalls Handler 1



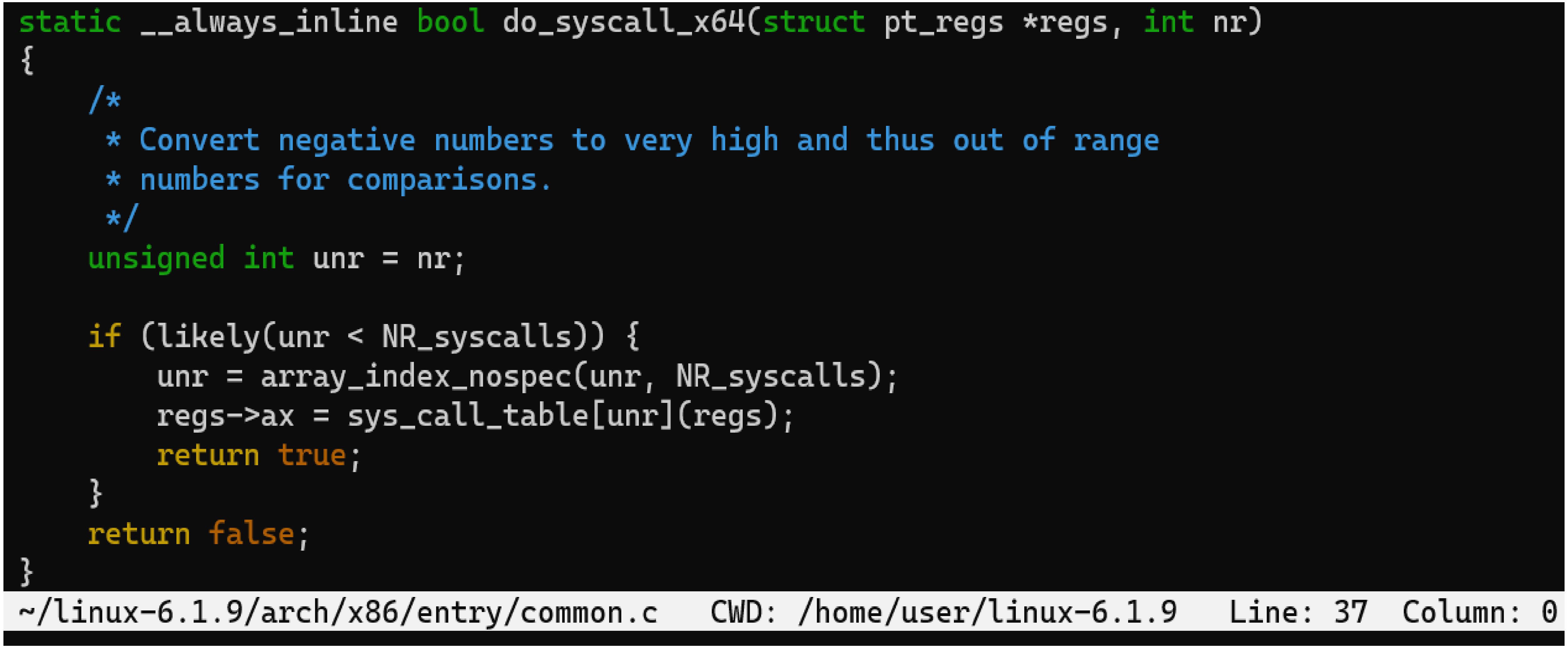
```
SYM_CODE_START(entry_SYSCALL_64)
    swapgs
    SWITCH_TO_KERNEL_CR3 scratch_reg=%rsp

    /* Construct struct pt_regs on stack */
    pushq   $__USER_DS                  /* pt_regs->ss */
    pushq   %r11                        /* pt_regs->flags */
    pushq   $__USER_CS                  /* pt_regs->cs */
    pushq   %rcx                        /* pt_regs->ip */
    pushq   %rax                        /* pt_regs->orig_ax */

    /* IRQs are off. */
    movq    %rsp, %rdi
    /* Sign extend the lower 32bit as syscall numbers are treated as int */
    movslq  %eax, %rsi

    call    do_syscall_64       /* returns with IRQs disabled */

~/linux-6.1.9/arch/x86/entry/entry_64.S[+]   CWD: /home/user/linux-6.1.9   Line: 85  Column: 1
```

Syscalls Handler 2



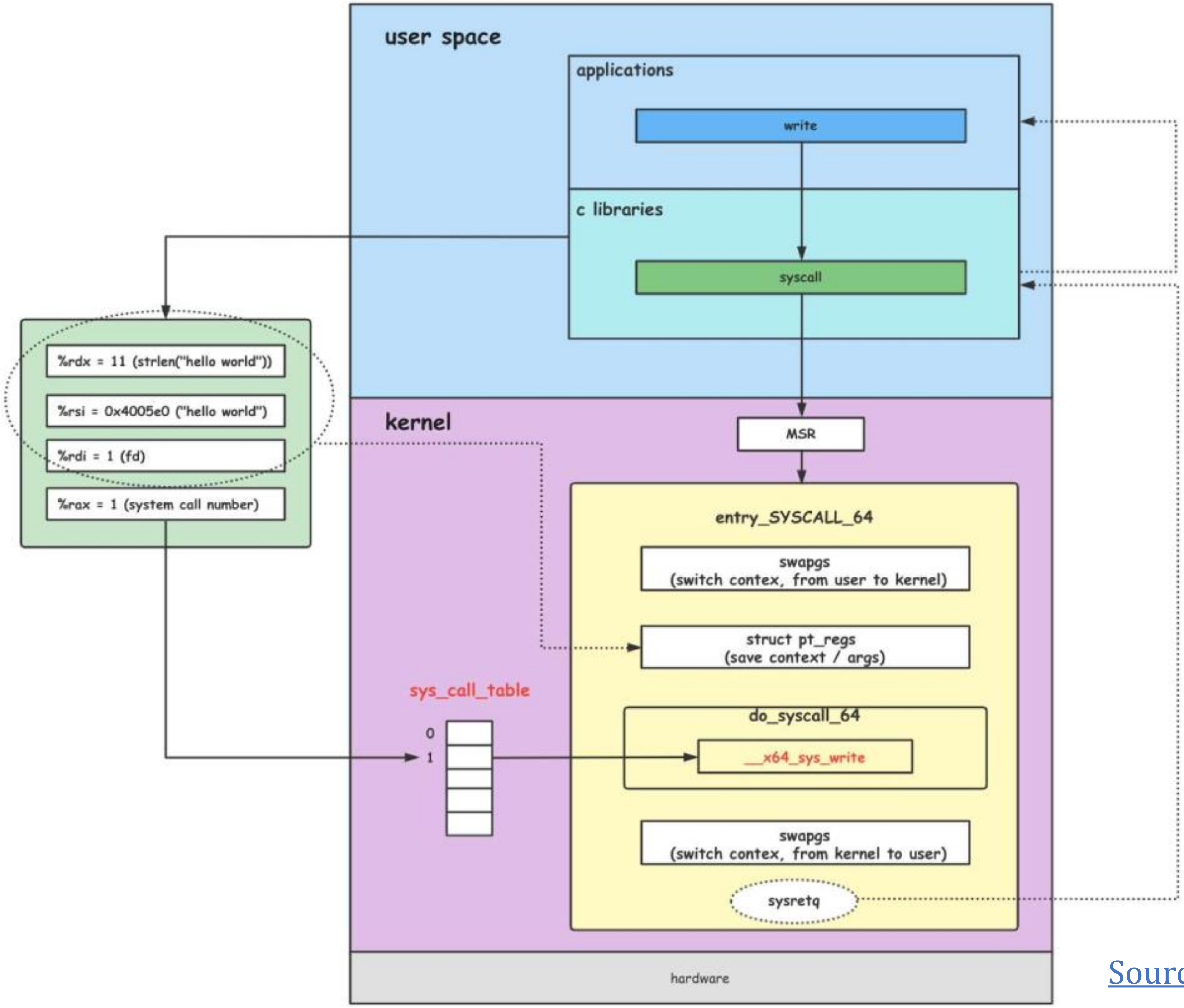
```
__visible noinstr void do_syscall_64(struct pt_regs *regs, int nr)
{
	add_random_kstack_offset();
	nr = syscall_enter_from_user_mode(regs, nr);

	instrumentation_begin();

	if (!do_syscall_x64(regs, nr) && !do_syscall_x32(regs, nr) && nr != -1) {
		/* Invalid system call, but still a system call. */
		regs->ax = __x64_sys_ni_syscall(regs);
	}

	instrumentation_end();
	syscall_exit_to_user_mode(regs);
}
~/linux-6.1.9/arch/x86/entry/common.c   CWD: /home/user/linux-6.1.9   Line: 70   Column: 1
```

Syscalls Handler (Finally)

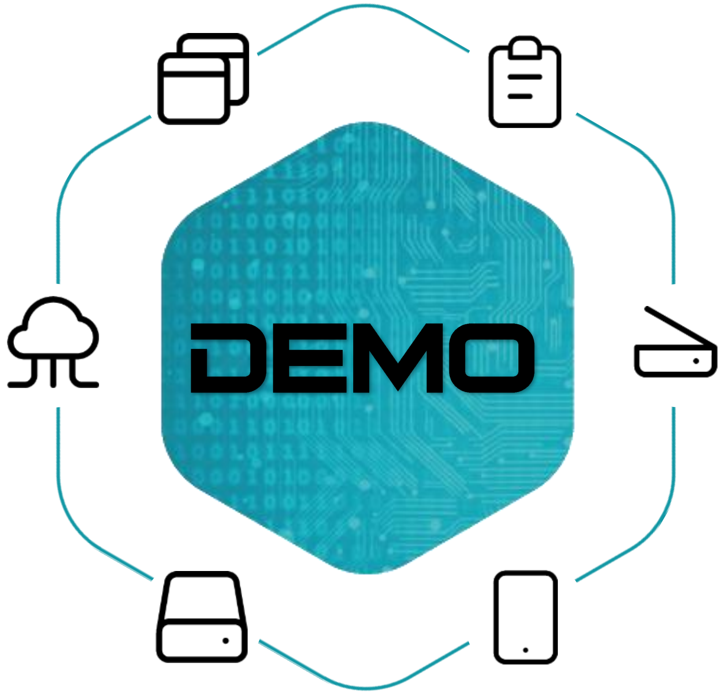


```
static __always_inline bool do_syscall_x64(struct pt_regs *regs, int nr)
{
	/*
	 * Convert negative numbers to very high and thus out of range
	 * numbers for comparisons.
	 */
	unsigned int unr = nr;

	if (likely(unr < NR_syscalls)) {
		unr = array_index_nospec(unr, NR_syscalls);
		regs->ax = sys_call_table[unr](regs);
		return true;
	}
	return false;
}
~/linux-6.1.9/arch/x86/entry/common.c   CWD: /home/user/linux-6.1.9   Line: 37   Column: 0
```

Syscalls Flow



Source



Examining entries of sys_call_table using gdb and breaking upon system call invocation.

Thanks

Mahsan

Your Support In Digital World