Class

6

CSC 472 Software Security

System Calls

Dr. Si Chen (schen@wcupa.edu)

Tue / Thu · 2:00–3:15 PM · 25 University Ave., Room 158
Fall 2026 · Week 4 · Tuesday, September 15, 2026
WolfCTF + GDB / GEF · wcu.ninja/csc472

From a C function to the kernel

Recall: a normal function call

Arguments, return address, stack frame, and the C calling convention.

Today: a controlled kernel entry

Request an OS service using the Linux i386 system-call ABI.

By the end of class

Read the syscall registers, trace write(), and explain execve().

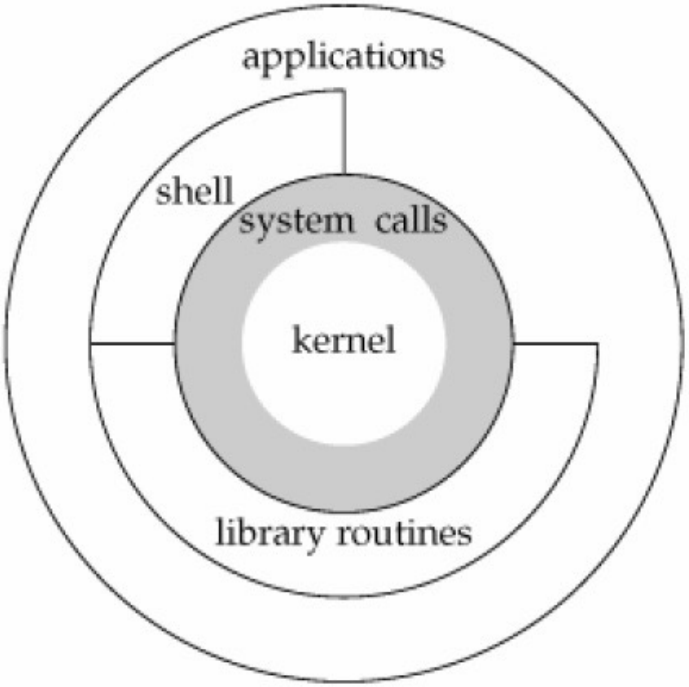
Continue Lab 1

Use GDB + GEF to connect registers, memory, and stack-frame evidence.

West Chester University

What is a system call?

A process requests a service from the operating-system kernel.



User space

Application code and libraries.

Kernel space

Protected OS code and resources.

The boundary

An approved entry point with arguments the kernel must validate.

Why a privilege boundary?

Applications can be buggy

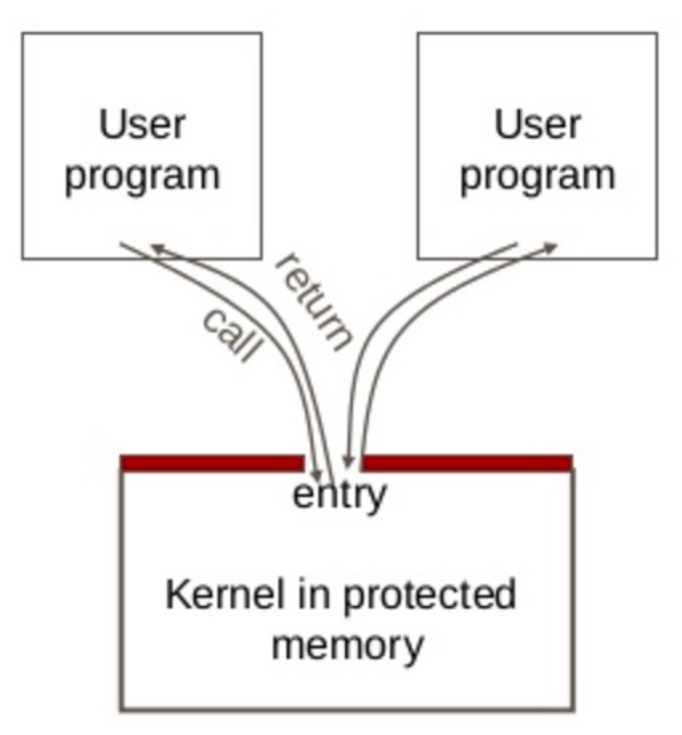
User code cannot freely read or modify protected kernel memory.

The kernel mediates access

Files, processes, networking, and memory permissions.

Controlled entry and return

The kernel validates requests, then resumes user execution.



Why ask the OS to wait?

A fragile delay

```
for (i = 0; i < N; ++i) {
    /* busy wait */
}
```

Loop time is not a clock

CPU speed, optimization, and other activity change the delay.

Use an OS timer

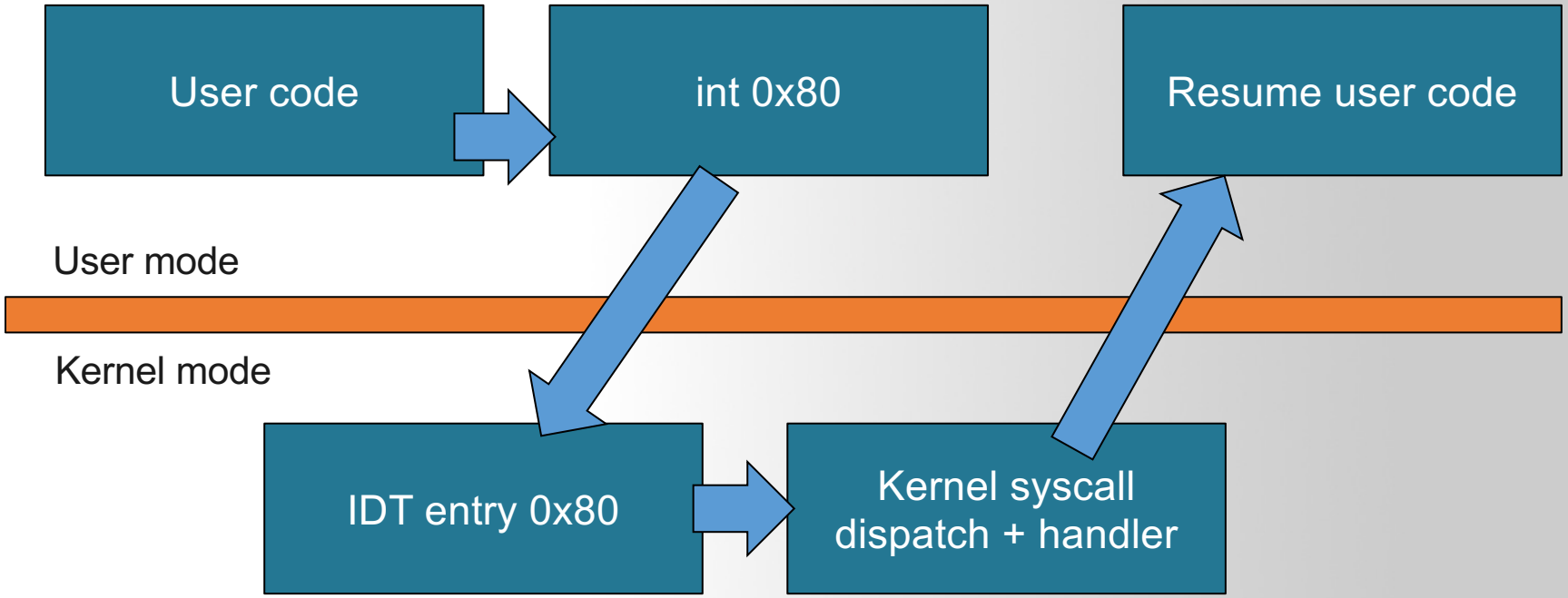
The thread can sleep while the CPU runs other work.

Still not exact timing

The scheduler may resume the thread later than requested.

West Chester University

Linux i386: crossing the boundary



Conceptual path for int 0x80. Other architectures use different entry mechanisms.

The Linux i386 syscall contract

Role	Linux i386 convention
Entry instruction	int 0x80
System-call number	EAX
Arguments 1–6	EBX, ECX, EDX, ESI, EDI, EBP
Result	EAX; raw errors are negative errno values

This is the syscall ABI. It is different from a C function's calling convention.

32-bit and 64-bit are different ABIs

	Linux i386 (our demos)	Linux x86-64
Entry	int 0x80	syscall
Number / result	EAX / EAX	RAX / RAX
First 3 arguments	EBX, ECX, EDX	RDI, RSI, RDX
write / exit / execve	4 / 1 / 11	1 / 60 / 59

Always identify the target ABI before choosing syscall numbers or registers.

West Chester University

Find the numbers for the target ABI

Inside WolfCTF

```
grep -E "__NR_(write|exit|execve) " \
  /usr/include/asm/unistd_32.h

#define __NR_exit 1
#define __NR_write 4
#define __NR_execve 11
```

The syscall table belongs to an ABI, not to the assembly syntax.

West Chester University

Get the classroom code on WolfCTF

Open portal.wolfctf.com/labs/lab1 → launch your terminal

Run these commands in the Linux terminal

```
cd ~
curl -fLO https://wcu.ninja/files/CSC472_Week04_code.tar.gz
tar -xzf CSC472_Week04_code.tar.gz
cd ~/csc472-week04/class06
make
file helloworld
```

Expected architecture: ELF 32-bit, Intel 80386. GDB loads GEF automatically.

West Chester University

Hello World through write()

helloworld.asm — Linux i386

```
section .text
global _start, write_call
_start:
    mov eax, 4          ; write
    mov ebx, 1          ; stdout
    mov ecx, msg        ; buffer address
    mov edx, msg_len    ; byte count
write_call:
    int 0x80
    mov eax, 1          ; exit
    xor ebx, ebx
    int 0x80
```

Data

```
section .data
msg: db 'Hello world!', 10
msg_len equ $ - msg
```

DB emits bytes.

12 visible characters
+ one newline
= 13 bytes.

Assemble, link, then observe

The commands behind make

```
nasm -f elf32 -g -F dwarf helloworld.asm -o helloworld.o
ld -m elf_i386 -z noexecstack helloworld.o -o helloworld
./helloworld

Hello world!
```

Object file → executable

The linker resolves the address of msg in the separate data section.

What does the kernel receive?

A file descriptor, a pointer, and a count — not a C string.

strace: see the kernel requests

Trace our 32-bit executable

```
strace -e trace=write,exit ./helloworld > /dev/null

write(1, "Hello world!\n", 13) = 13
exit(0)                        = ?
+++ exited with 0 +++
```

Read the line as a contract

write(fd = 1, buffer = "Hello world!\n", count = 13) returns 13.

A useful distinction

strace shows kernel calls; it does not list every C library call.

GEF: inspect the syscall registers

gdb -q ./helloworld

```
gef config context.layout 'regs code stack'
break write_call
run
info registers eax ebx ecx edx
x/13cb $ecx
si
print/d $eax
```

Before int 0x80: 4, 1, buffer address, 13. After this successful write: EAX = 13.

West Chester University

execve(): replace the current program

The syscall setup

```
mov eax, 11       ; execve
mov ebx, sh       ; pathname
mov ecx, argv     ; argument-vector pointer
mov edx, envp     ; environment-vector pointer
int 0x80
```

Data in shell.asm

```
argv: dd sh, 0
envp: dd 0
sh:   db '/bin/sh', 0
```

DD emits 32-bit values.
argv holds pointers.
DB emits the pathname bytes.

Running the shell example

In the WolfCTF terminal

```
./shell
id
exit
```

Success

/bin/sh becomes the current process image.

Failure

execve returns a negative error code in raw EAX.

Address dependence

This ELF uses absolute addresses for data — revisit on Thursday.

Type exit to return from the demonstration shell to your lab terminal.

Useful calls and memory permissions

Service	What the process asks the kernel to do
open / read / write	Open a file; transfer bytes through a file descriptor.
mmap	Create a mapping in the process address space.
mprotect	Change access permissions of a page-aligned range.
execve	Replace the current process image with a program.

Read, write, and execute are distinct permissions. Changing one mapping is local.

Library calls are another layer

fwrite() → buffering → write() as needed

Many library writes can become one kernel write. A flush may trigger I/O.

malloc() → allocator → mapping calls as needed

An allocation can reuse memory already owned by the process.

Error reporting

Raw syscall: negative error value. Typical libc wrapper: −1 and errno.

Interface choice

Libc provides a convenient API; raw syscalls expose the ABI directly.

Practice: predict before stepping

1 · Predict

What changes if the Hello World byte count is 5 instead of 13?

2 · Test

Change only EDX's count, rebuild, and inspect it at write_call.

3 · Explain

Why does write() not need a terminating NUL byte?

4 · Connect

Which registers belong to the syscall ABI, and which hold addresses?

Next: turn the program into bytes

Today's checkpoint

Identify the ABI → load registers → enter kernel → inspect the result.

Thursday, September 17

Shellcode, position-independent data, and GEF tracing of raw bytes.

Lab 1 continues on WolfCTF

Due Thursday, September 24, 2026 · 11:59 PM Eastern Time.

Submit the required PDF report

D2L → CSC 472 → Assignments → Lab 1
Report guide: wcu.ninja/csc472/lab-report

West Chester University

References and classroom files

Original course materials

cs.wcupa.edu/SCHEN/ss2024/ → Chapter 6

Linux interfaces

man7.org: syscall(2), execve(2), mmap(2), mprotect(2)

Assembly and object tools

nasm.us/doc/ · sourceware.org/binutils/docs/

This week's slides and code

wcu.ninja/csc472/
Code bundle: CSC472_Week04_code.tar.gz