Lecture 06: Process Creation

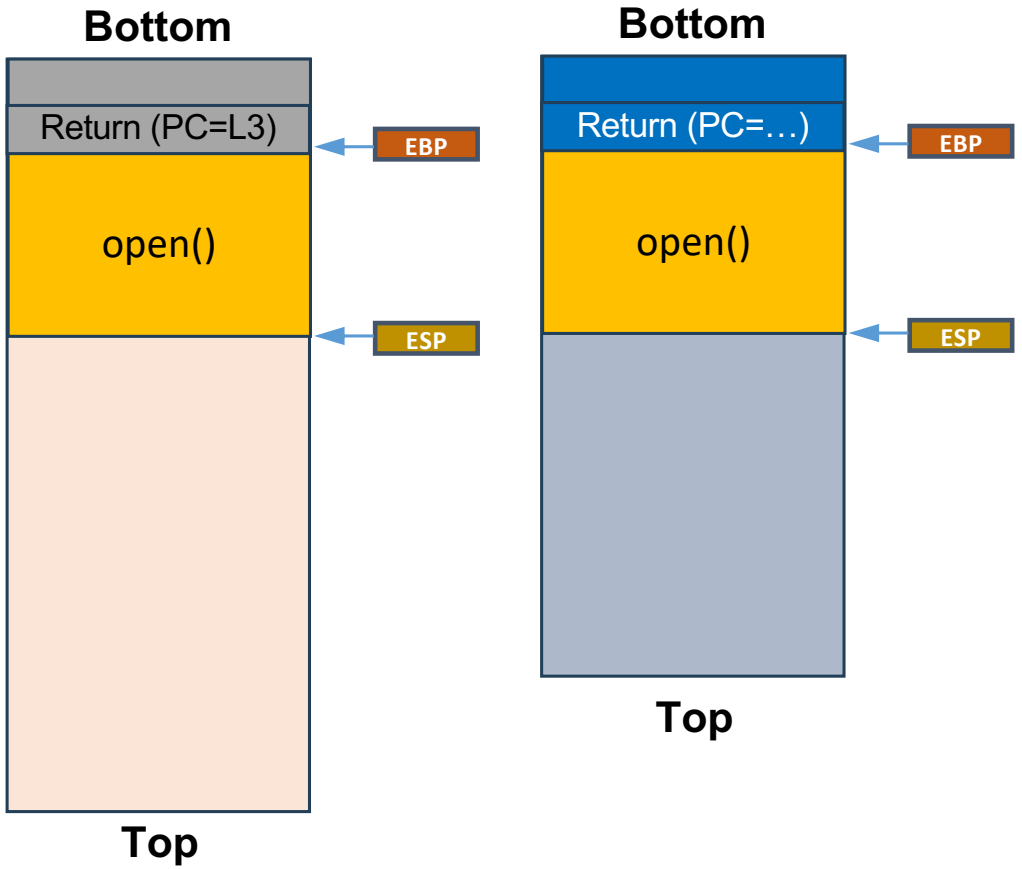
Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Today's Class

- Unix architecture
- System calls
- Process creation and termination

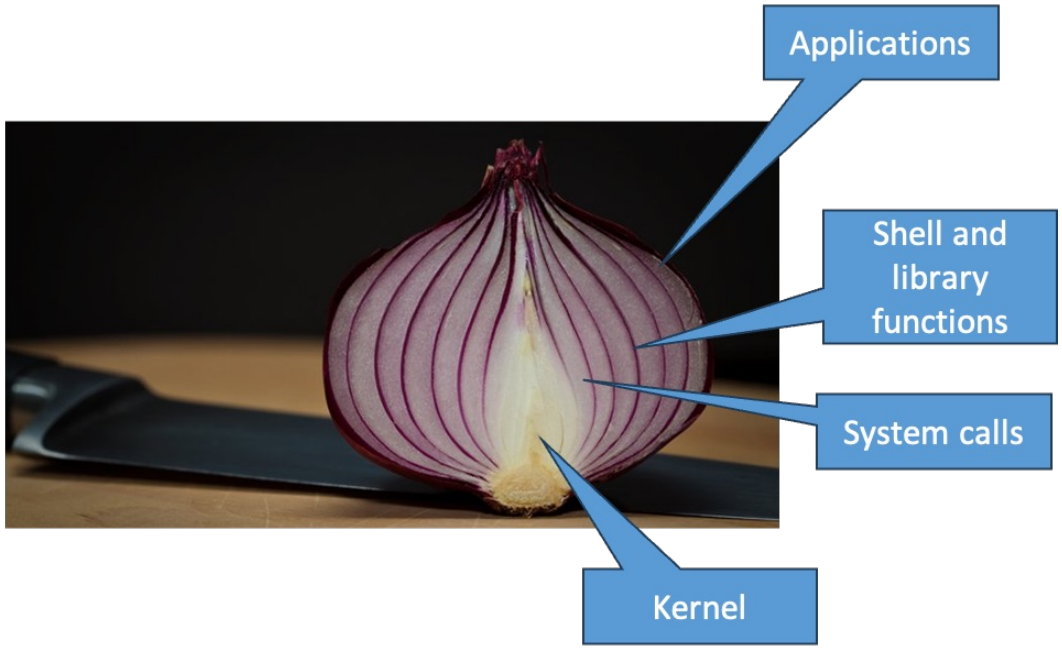
Return Address on Stack can be Modified

```
L1: main () {
      <save_regs>
      <save(PC)>
L2:   open();
      <restore(PC)>
      <restore_regs>
L3: }
```



- Issues
  1. If **open**() is a regular method call then it's machine code won't be able to have a foolproof check if the calling process has the permission to access this file
     - The loader can simply patch the machine code with the call to open() at the runtime
  2. The main() called **open**(), but then **open** could be **forced** to return to some malicious process call stack

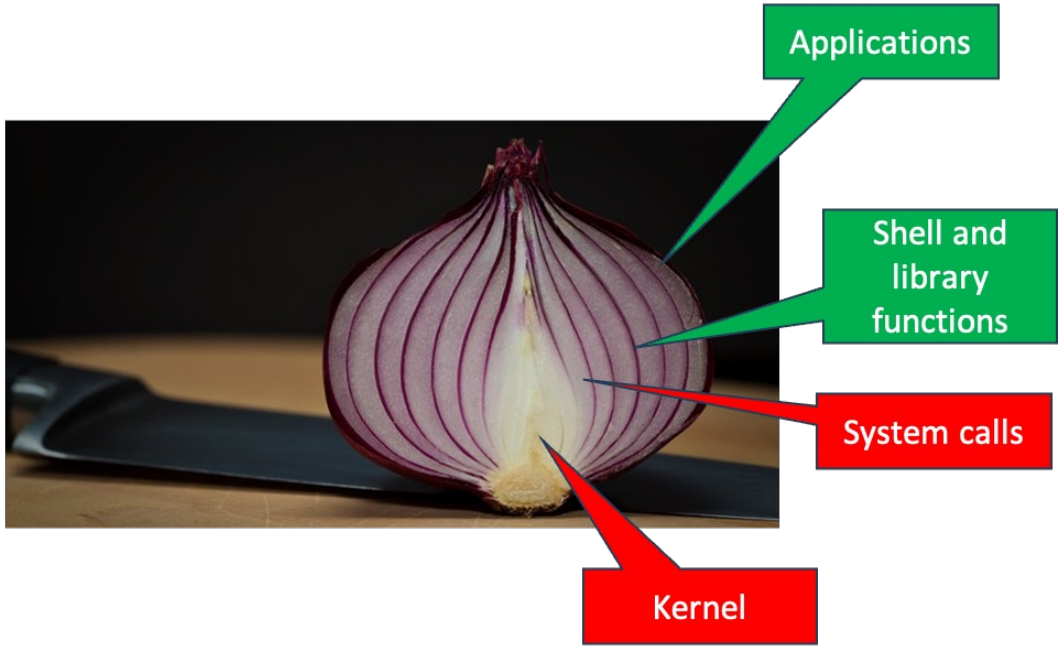
Unix Architecture



Recall the analogy discussed in Lecture 01. Who interacts with the hardware in that analogy?

- A major goal of OS is to support portability across various architecture
- A layered approach helps in achieving this goal as the innermost layer is only one that has to interact with the hardware
  - This innermost layer is called as OS **kernel**
  - It is relatively small, controls the hardware resources, and provides an environment under which programs can run
- System call is the interface to the kernel (e.g., read, open, etc.)
- Library functions are are built on top of system calls that applications can use
  - E.g., C-library function malloc uses sbrk system call, open(), etc.
- Shell is a command line interpreter that reads user input and execute commands
  - E.g., bash, csh, etc.

Protection Rings



- Protection rings are hardware supported mechanisms used by the OS for protection of data and functionality
  - E.g., x86 supports four protection levels (0-3)
  - Level-0 is called as **kernel mode** (supervisor mode) and Level-3 is **user mode**
- Unix-like OS execute user applications, shell, and library functions in the user mode whereas kernel/syscalls in the kernel mode

Today's Class

- Unix architecture
- System calls
- Process creation and termination

Interrupts: Gateway to the Kernel Mode

- Interrupts are signals to the CPU that something special has to happen
  - INT instruction on x86 to generate an interrupt
  - Interrupts are handled only by the kernel
- Three kinds of interrupts
  - Exceptions
    - Attempt to access invalid memory address, divide by zero, etc.
      - Covered in next lecture
  - **System calls (software interrupt)** – today's focus
  - Hardware interrupt
    - Signal generated by some hardware device. E.g., disk can generate an interrupt when a block of memory has been read and is ready to be retrieved
- As per Unix terminology, "interrupts" refer to the hardware interrupts, whereas "trap" refers to software interrupts (e.g., some special type of exceptions and all **system calls**)

libc Syscall Wrapper v/s Direct Syscall

- Code portability across different OS versions
- Easy to use APIs simplify coding experience as compared to using raw system call APIs
- Error handling support

Steps for Making a System Call (1/5)

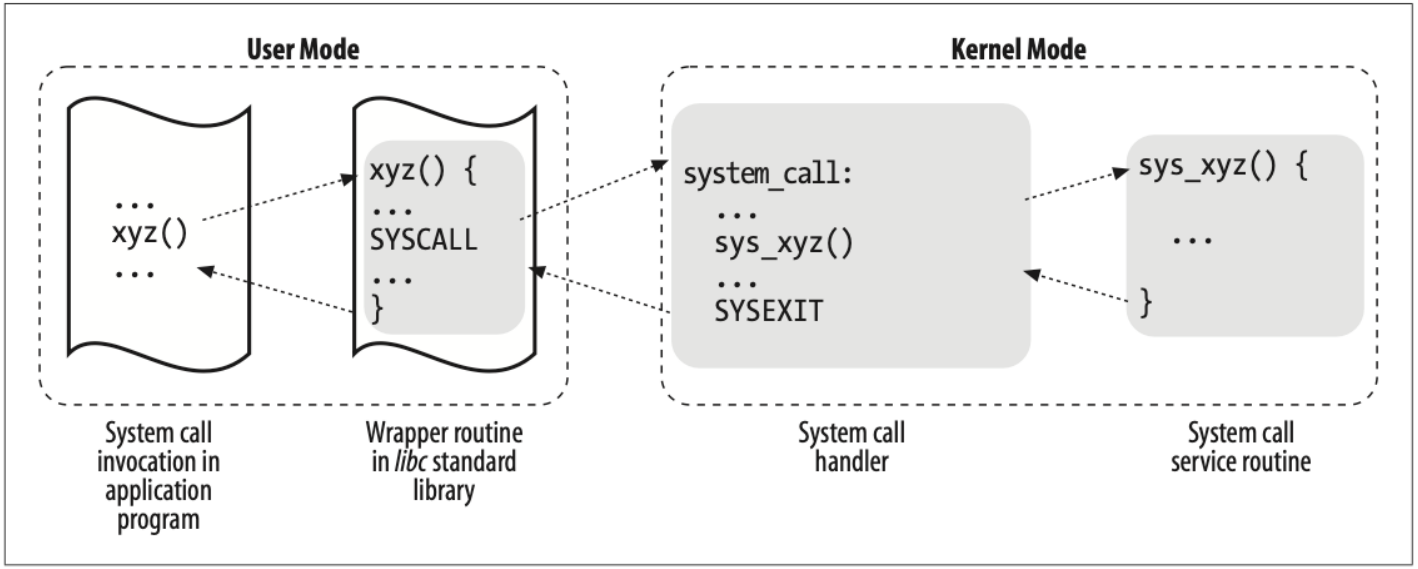
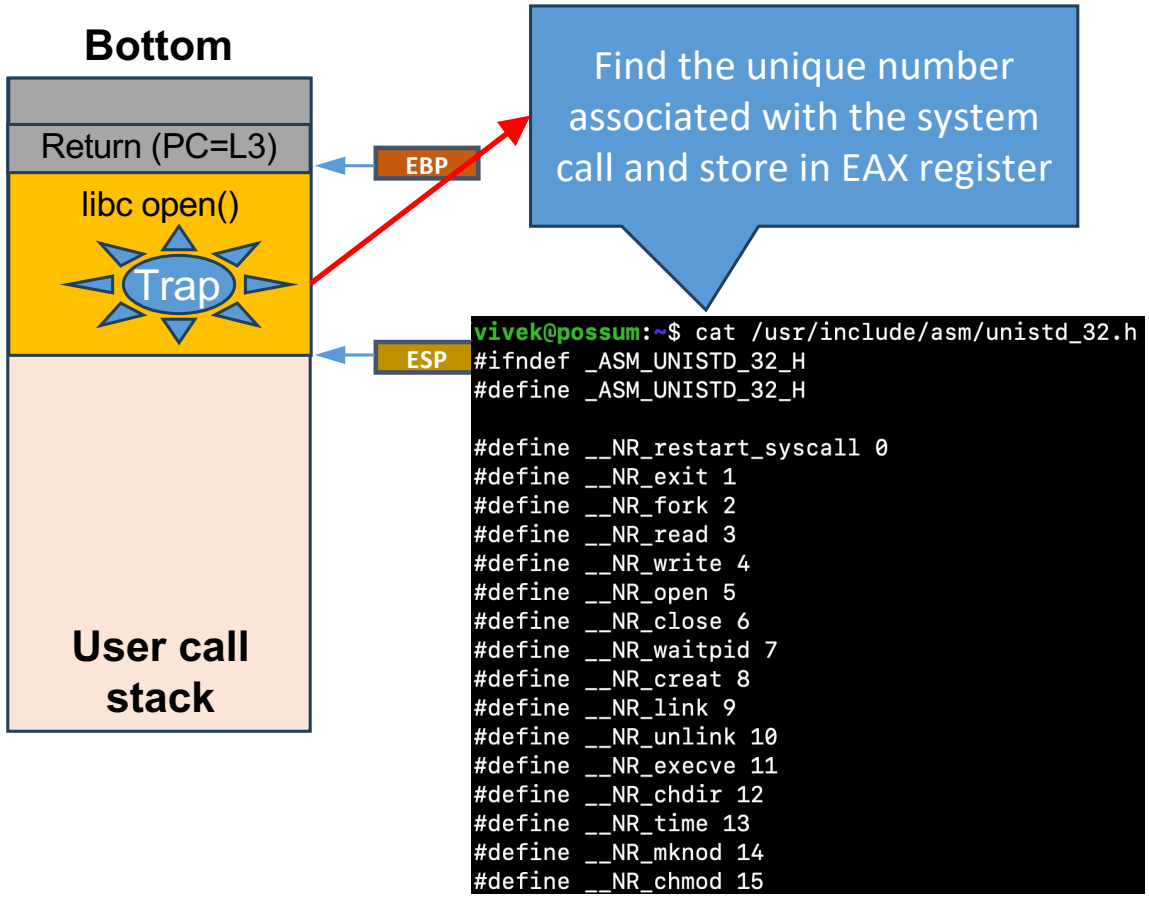


Figure 10-1. Invoking a system call

- **At a high level** the steps are as follows:
  1. Save registers in kernel mode stack
  2. Handle the system call by invoking a corresponding system call service routine
  3. Restore the registers and switch back into user mode

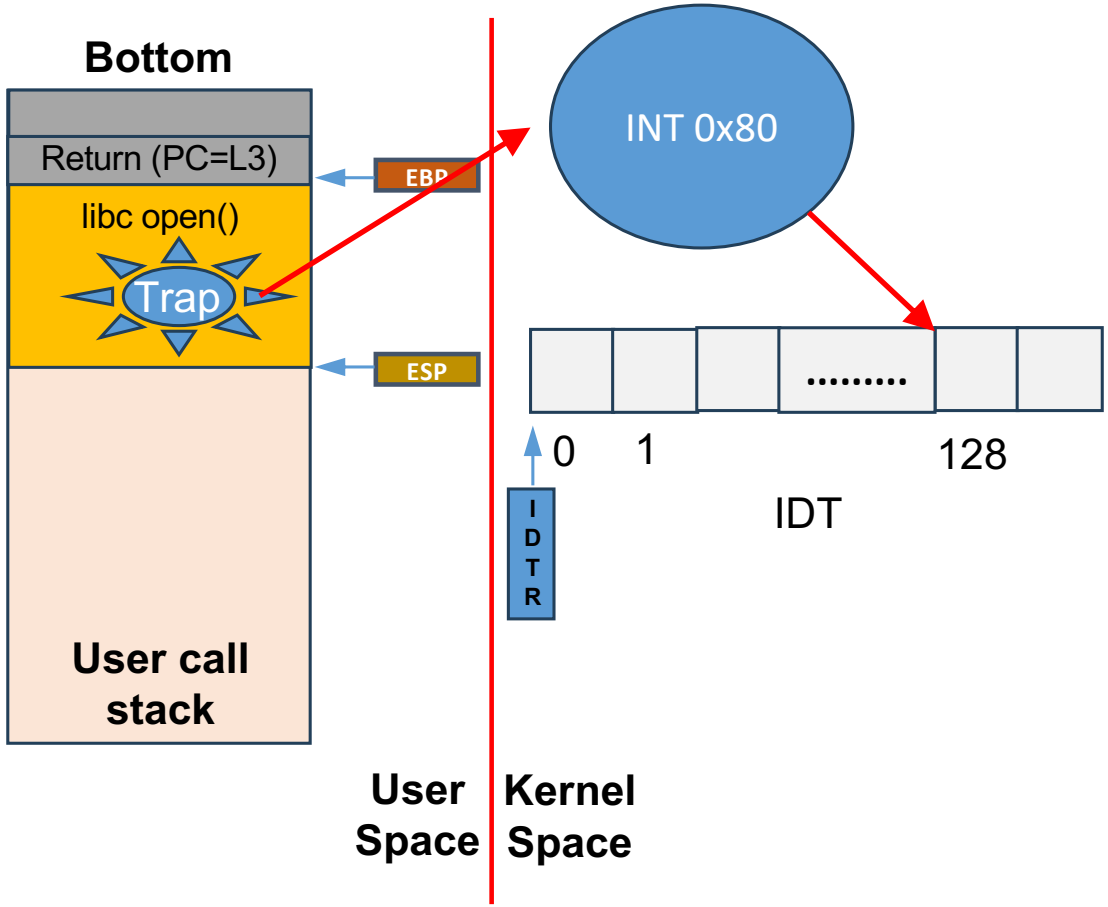
Figure source: from the book "Understanding the Linux Kernel"

Steps for Making a System Call (2/5)



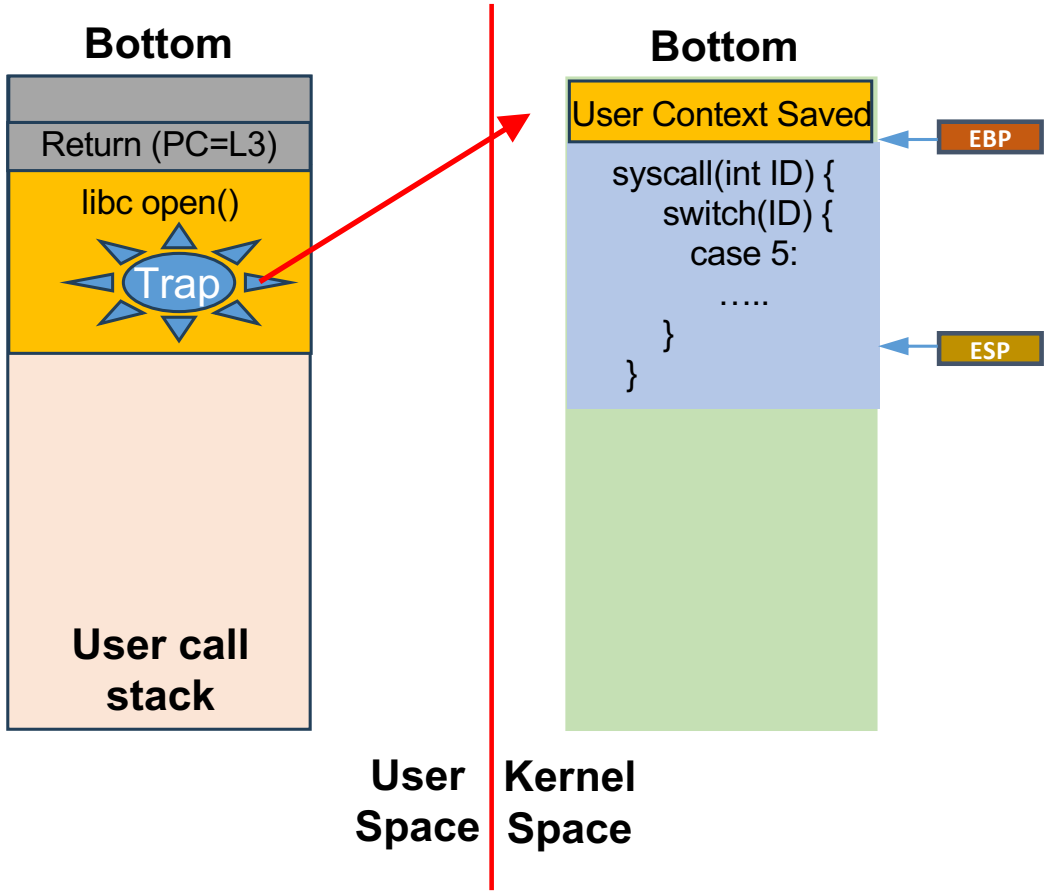
- Each system call has a unique number associated that is declared inside the file unistd.h
- Store this number in the EAX register
- Process is still executing inside user space

Steps for Making a System Call (3/5)



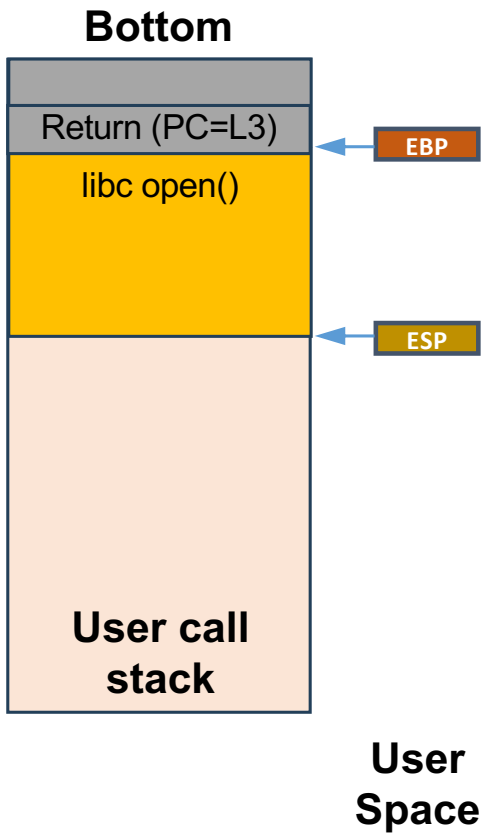
- Kernel sets up Interrupt Descriptor Table (IDT) at boot time whose address is stored in IDT register (IDTR). Total 256 entries in IDT on x86
- Each IDT index has the address of some interrupt handler
- 128th entry associated with the system call handler

Steps for Making a System Call (4/5)



- Switch to kernel stack
  - What should be the first step?
- Save registers on kernel stack. These registers are currently holding details to return back to user stack
- Look up the syscall number from EAX and process the syscall

Steps for Making a System Call (5/5)



- For returning to user call stack, first pop the user registers from kernel stack and store them back into respective CPU registers
  - Return back to user call stack in the user mode
    - Protection level-3

Today's Class

- Unix architecture
- System calls
- Process creation and termination

Next Lecture

- Process life lessons (contd.) and inter process communication