COMP304
Operating Systems (OS)

Operating System Structure

Didem Unat

Lecture 2

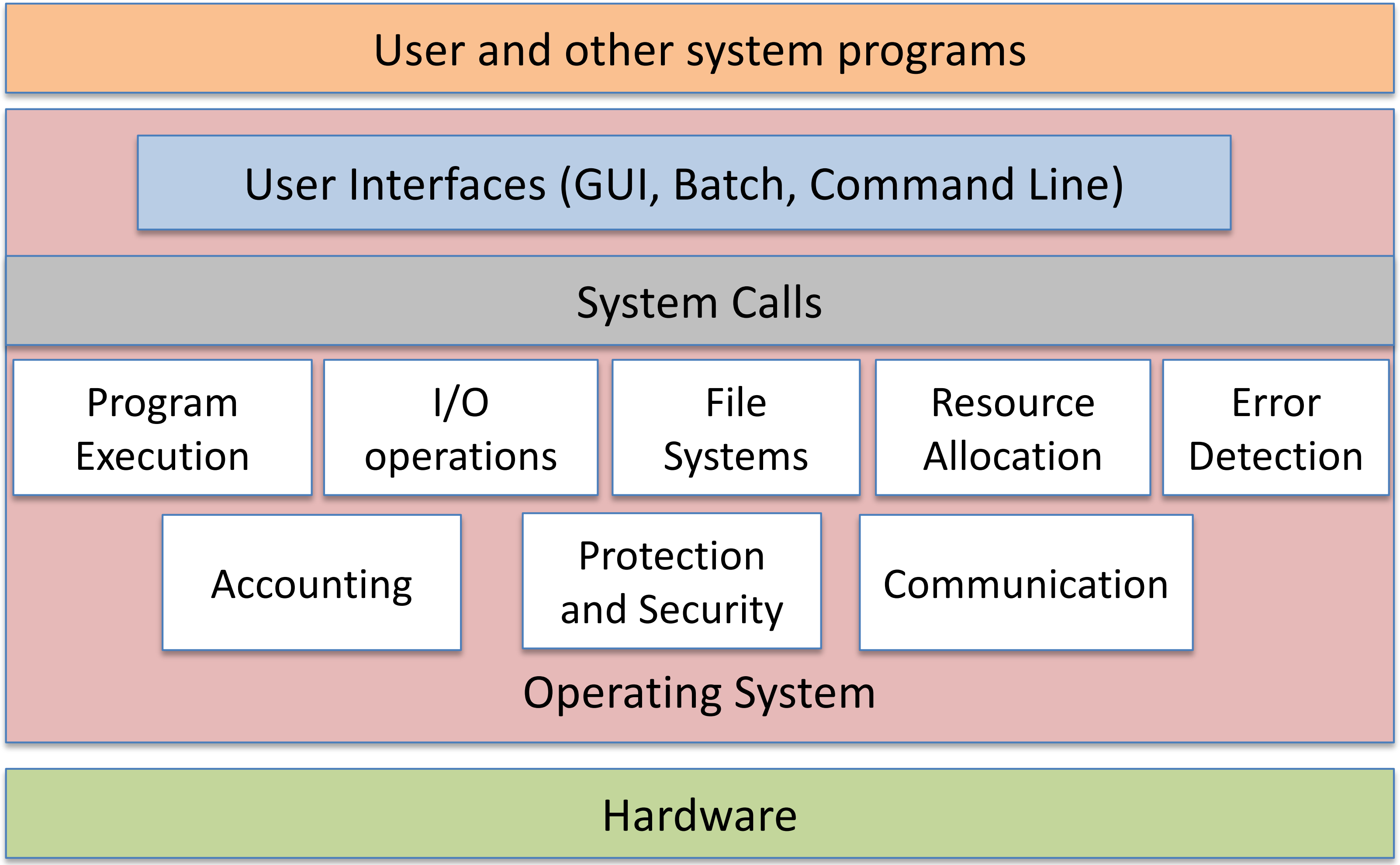
Outline

- Operating System Services
- Command Interpreter
- Dual Mode Operation
- System Calls and Types
- I/O, Memory and CPU Protection
- Operating System Design Structure

Computer Startup

- **Bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM, generally known as **firmware**
  - Initializes all aspects of a system
  - Loads operating system **kernel** into main memory and starts execution
    - The first system process is ‘**init**’ in Linux
  - When the system is fully booted, it waits for some event to occur
- Kernel
  - The ``one” program running at all times (the core of OS)
    - Everything else is an application program
- Process
  - An executing program (active program)

Operating System Services



Operating System Services (1/3)

- **User interface** - Almost all operating systems have a user interface (**UI**).
  - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **or Batch**
- **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
- **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
- **File-system manipulation** - Programs need to read and write files and directories, create and delete them, search them, list file information, manage permissions.

Operating System Services (2/3)

- **Communications** – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)

- **Error detection** – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing

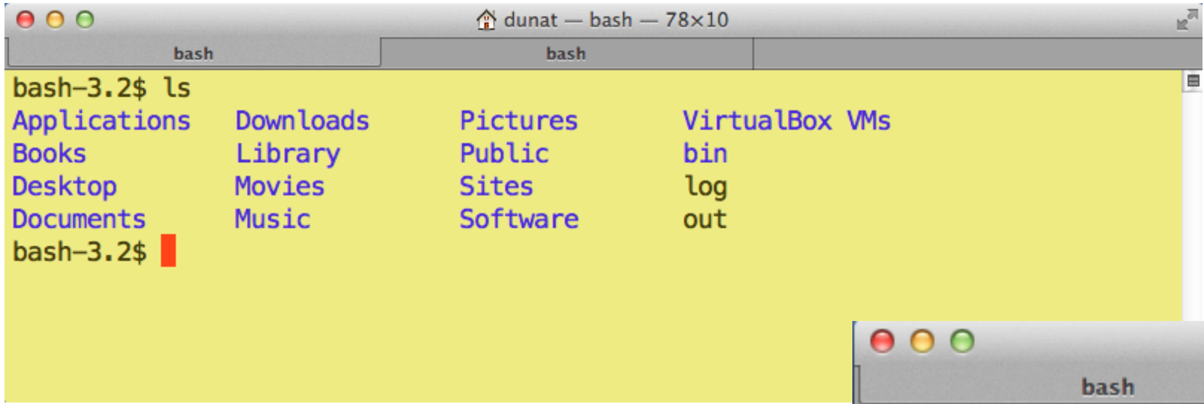
Operating System Services (3/3)

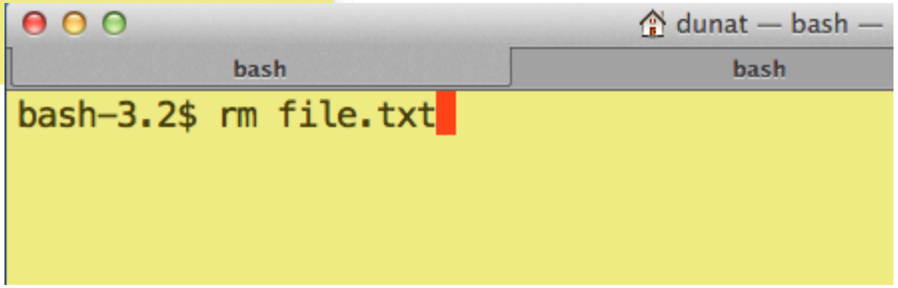
- **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
  - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code

- **Accounting -** To keep track of which users use how much and what kind of computer resources, improve response time to users

- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

Command Interpreters





Command Interpreters (Shells)

- In UNIX everything is a file
  - Command interpreter does not understand the command (e.g. “rm”)
  - It merely uses the command to identify a file to be loaded into memory and executed.
    - For example, shell would search for a file called ‘rm’, load the file into memory and execute it with the parameter file.txt
  - Thus, programmer can add new commands to the system easily by creating new files

Src code of Linux Commands

- All these basic commands are part of the **coreutils** package.
  - http://www.gnu.org/software/coreutils/
  - commands such as rm, ls, chmod, cp …
    - https://github.com/coreutils/coreutils/tree/master/src

- For example, "ls" command:
  - https://github.com/coreutils/coreutils/blob/master/src/ls.c
  - Only 5000+ code lines for a command 'easy enough'

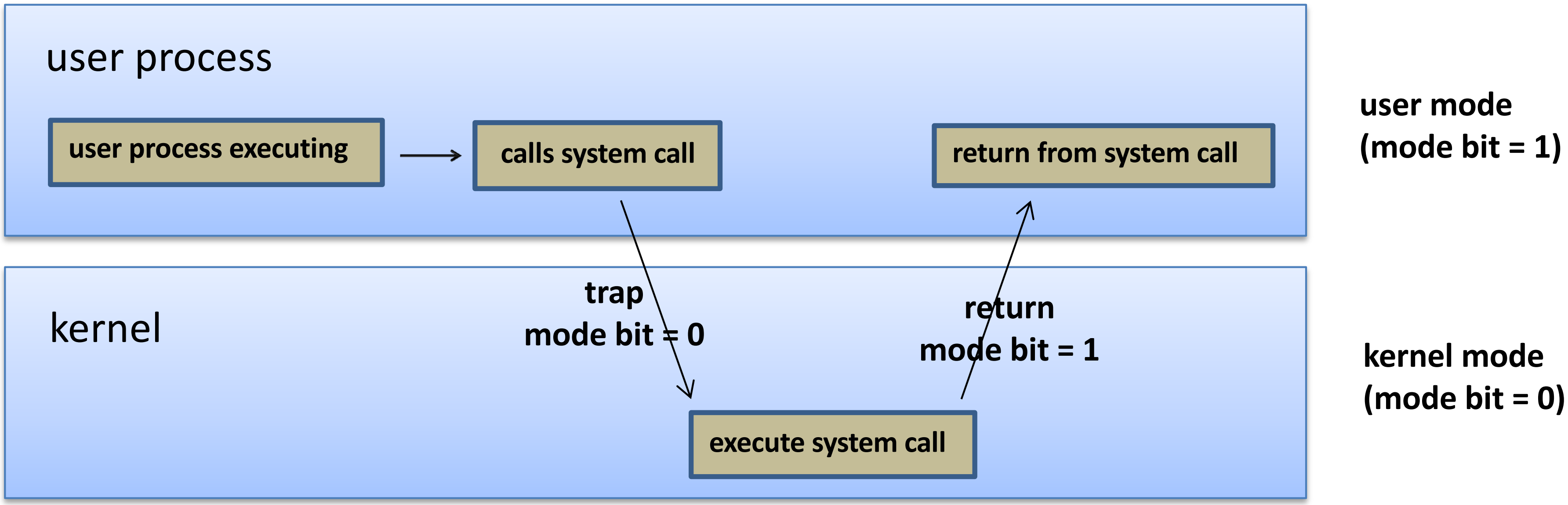
OS Protection: Dual-Mode Operation

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as **privileged**, only executable in kernel mode
    - For example, I/O related instructions are privileged

- Ensures that an incorrect program cannot cause other programs to execute incorrectly.

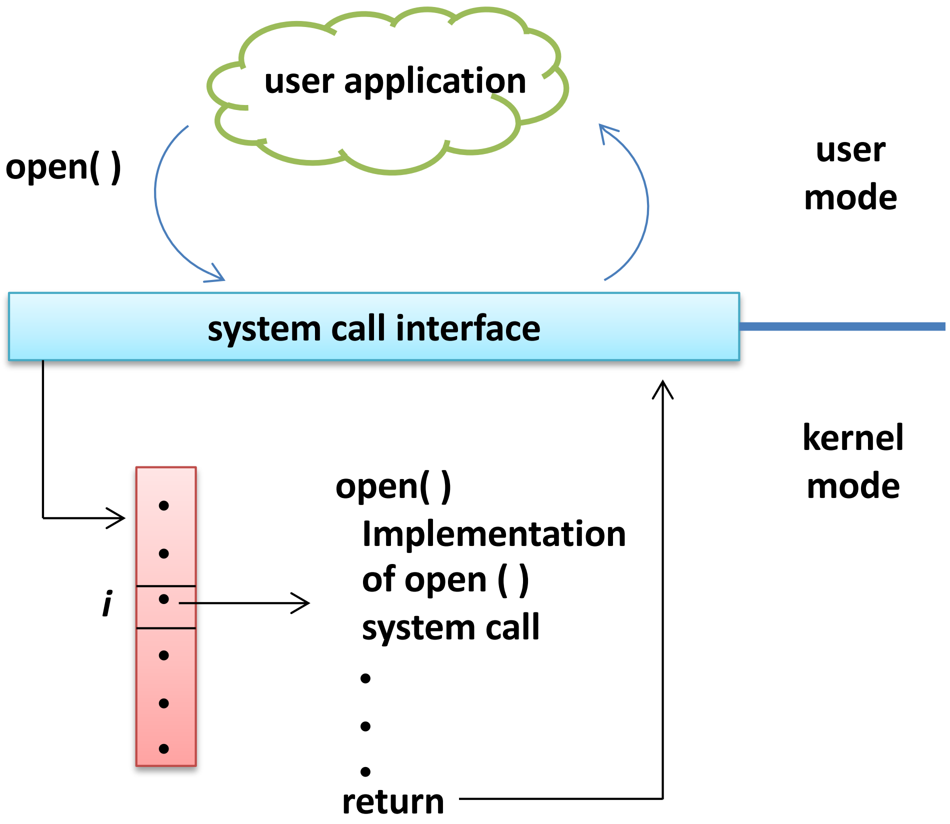
Transition from User to Kernel Mode

- **System Call**
  - Results in a transition from user to kernel mode
  - Return from call resets it to user mode
- Software error or a user request creates an exception or trap



Example: Linux System Calls

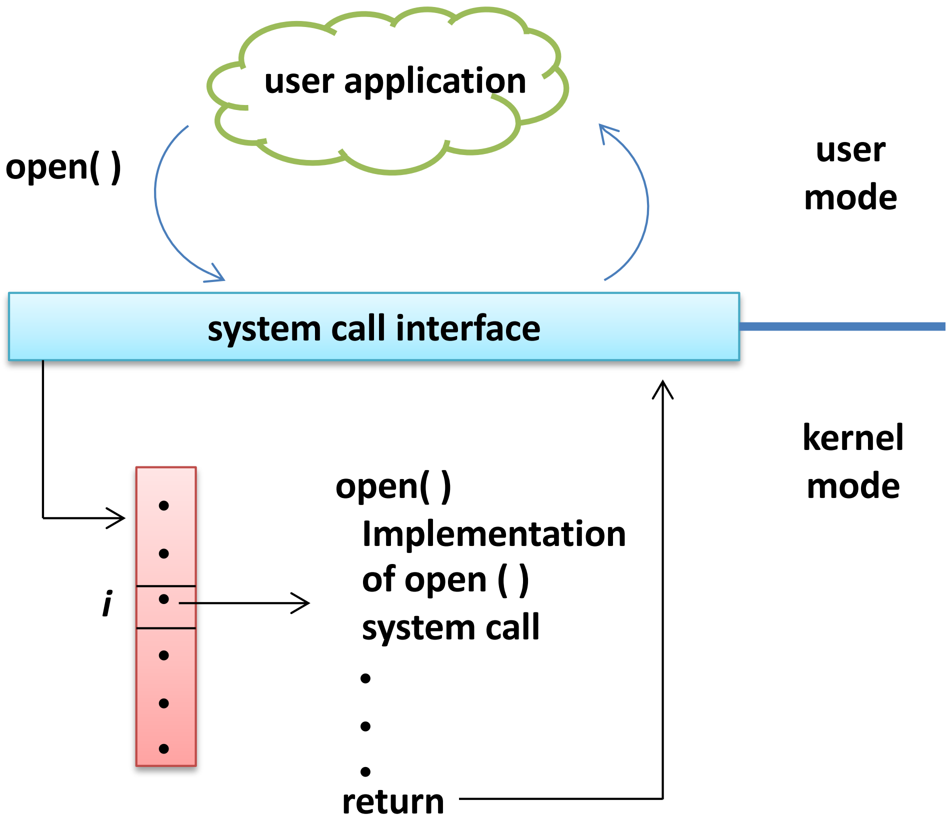
- A system call number is a unique integer in Unix-based OSs
  - There are about >300 system calls in Linux
  - A list of all registered system calls is maintained in the *system call table*
    - *Those numbers cannot be changed or recycled*
  - See the list of system calls with a command
    - (Location might differ depending on the Unix distribution)




```
cat /usr/include/asm/unistd.h | less
```

Example: Linux System Calls

- One can also call a service by directly using its number
  - syscall(system_call_number, arguments)
- Actual Implementation of a system call is in different files in the kernel src
  - https://elixir.bootlin.com/linux/latest/source
- An example, open system call
  - https://0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-5.html



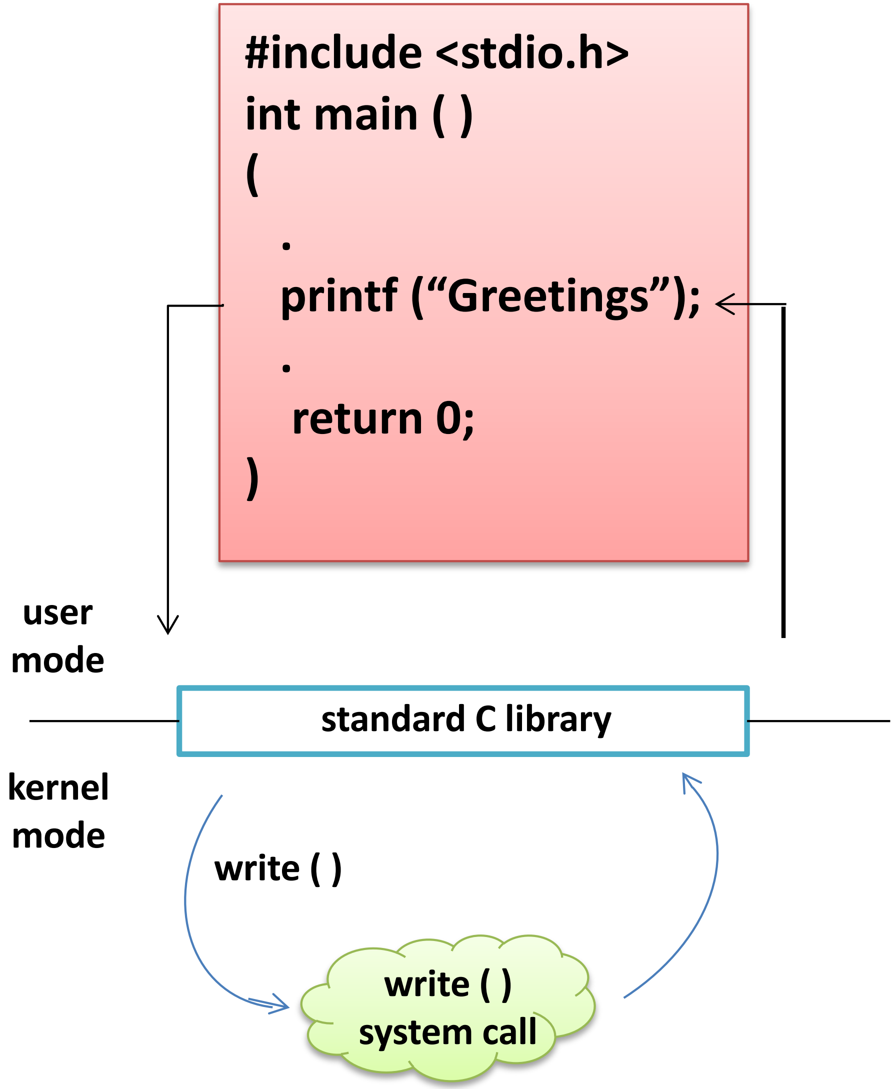
System Calls

- Programming interface to the services provided by the OS
  - Well-defined and safe implementation for service requests
  - Typically written in a high-level language (C or C++)
- A typical OS executes 1000s of system calls per second

- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
  - Wrapper functions for the system calls

- Three most common APIs are
  - Windows API for Windows,
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X),
  - Java API for the Java virtual machine (JVM)

Standard C Library Example



- C program invoking printf() library call, intercepts function call in the API and invokes the necessary system calls within the operating system
  - Calls *write()* system call

- Caller needs to know nothing about
  - how the system call is implemented
  - what it does during execution

Why use APIs instead of using system calls directly?

Examples for Types of System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

- Types of system calls classified under 6 categories. Table gives an example for Windows and Unix syscalls.

Privileged Instructions

- The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.
- We accomplish this protection by designating some of the machine instructions that may cause harm as **privileged instructions**.
  - The hardware allows privileged instructions to be executed only in kernel mode.
  - If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system

I/O Protection

- All I/O instructions are *privileged instructions*.
  - Must ensure that a user program could never gain control of the computer in **kernel** mode (i.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

1. "normal" instructions, e.g., add, sub, etc.
2. "privileged" instructions, e.g., initiate I/O switch state vectors or contexts load/save from protected memory etc.

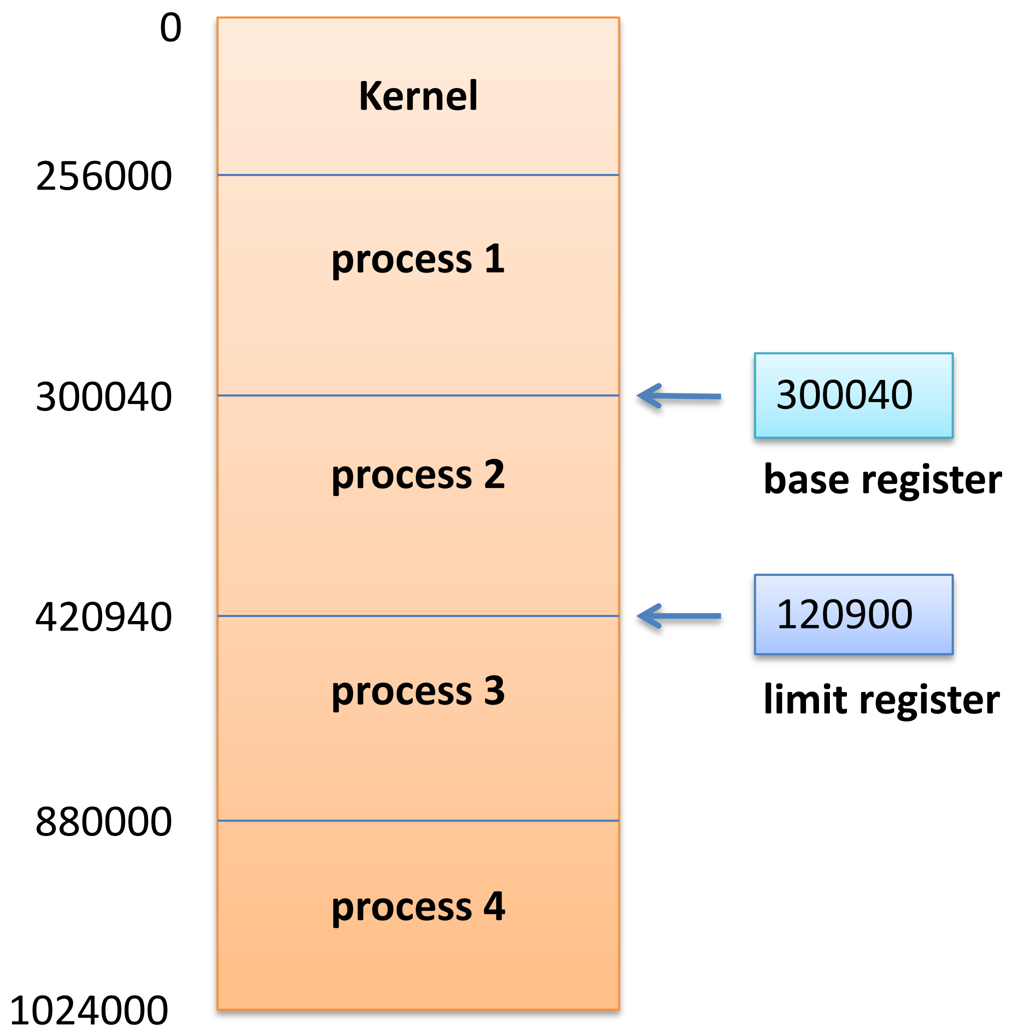
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.

- In order to have memory protection, add two registers that determine the range of legal addresses a process may access:
  - **Base register** – holds the smallest legal physical memory address.
  - **Limit register** – contains the size of the range

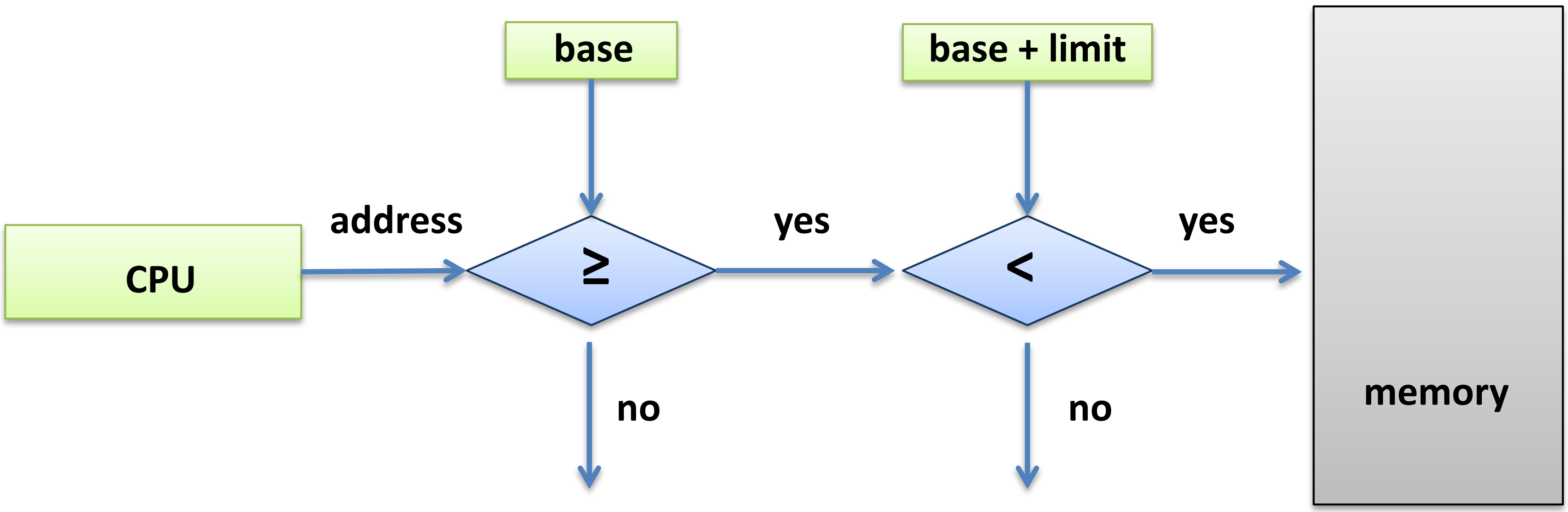
- Memory outside the defined range is protected.

Use of a Base and Limit Registers



Hardware Protection

- When executing in kernel mode, the operating system has unrestricted access to both kernel and user's memory.
- The load instructions for the *base* and *limit* registers are **privileged instructions.**



A fault raised by hardware, notifying the operating system about an addressing error

CPU Protection

- *Timer* – interrupts computer after specified period to ensure operating system maintains control.
  – Timer is decremented every clock tick.
  – When timer reaches the value 0, an interrupt occurs.

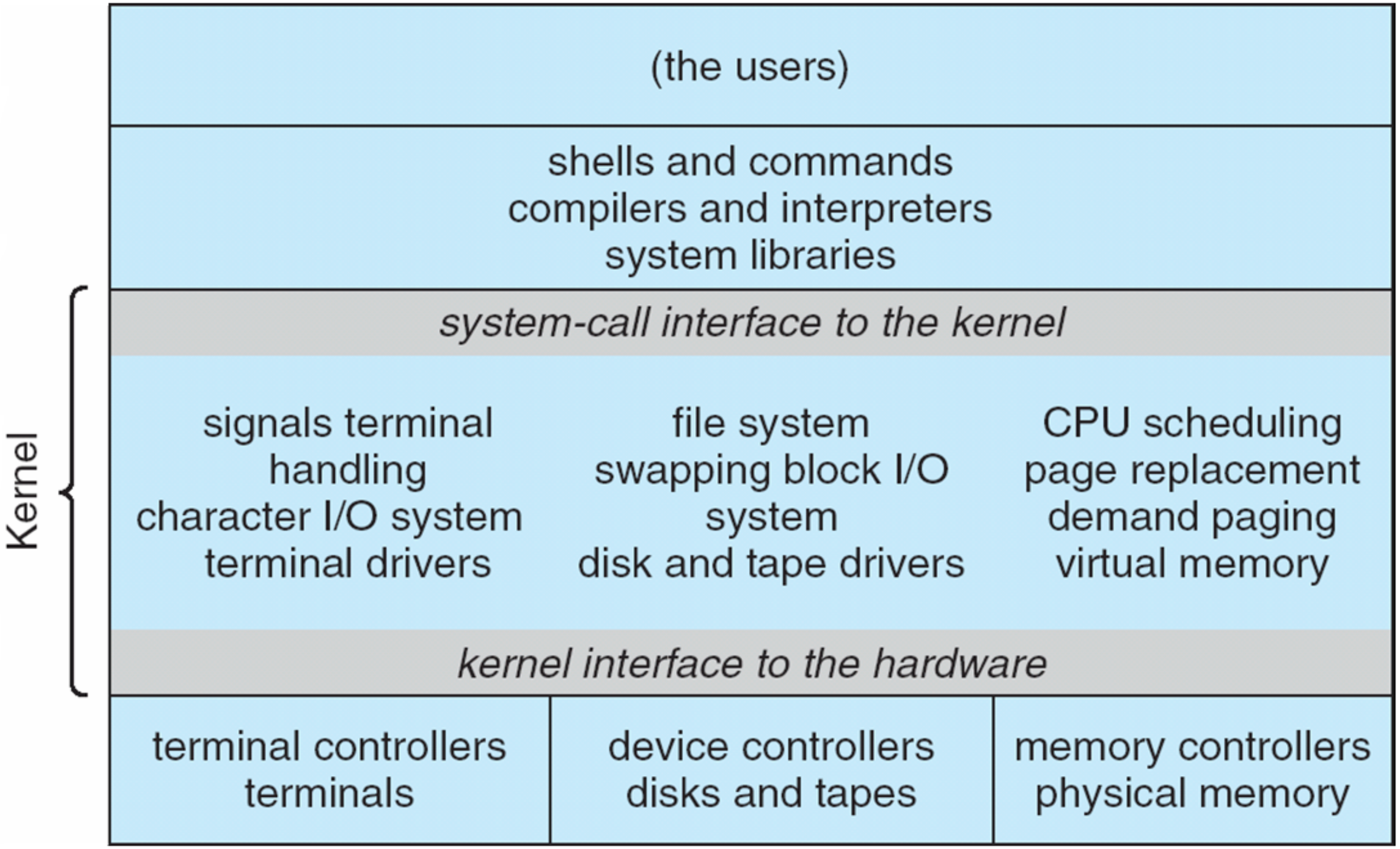
- Timer commonly used to implement time sharing systems.
- Clearly, instructions that modify the content of the timer are privileged.

Operating System Structure

- General-purpose OS is very large program
  - Typically written in assembly, C/C++, some scripts in Perl or Python
- Various ways to structure it
  - **Monolithic Kernel**: All the OS services are implemented in the kernel. Fast OS but hard to extend
    - Ex: MS-DOS, Unix
  - **Microkernel:** Moves all the nonessential components from the kernel to user level. Smaller kernel, uses messages with system and user-level programs
    - Ex: Mach
  - **Modular Approach**: Loadable kernel modules, load additional services if needed at boot or run time
    - Ex: Solaris
- Most current OS combines all three approaches nowadays
  - Ex: Windows, Mac OS X, Linux

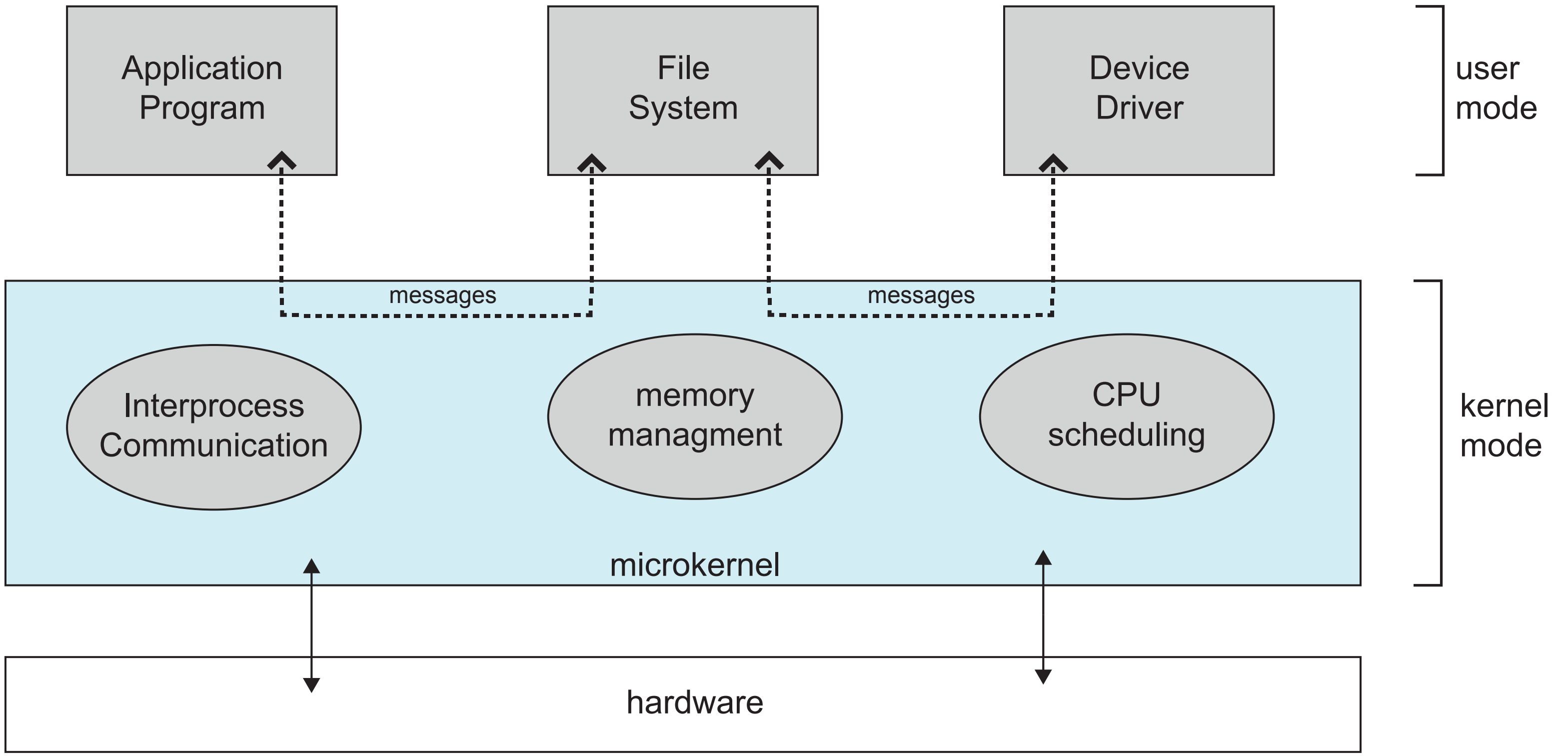
Monolithic Kernel

All the OS services are implemented in the kernel. Fast OS but hard to extend



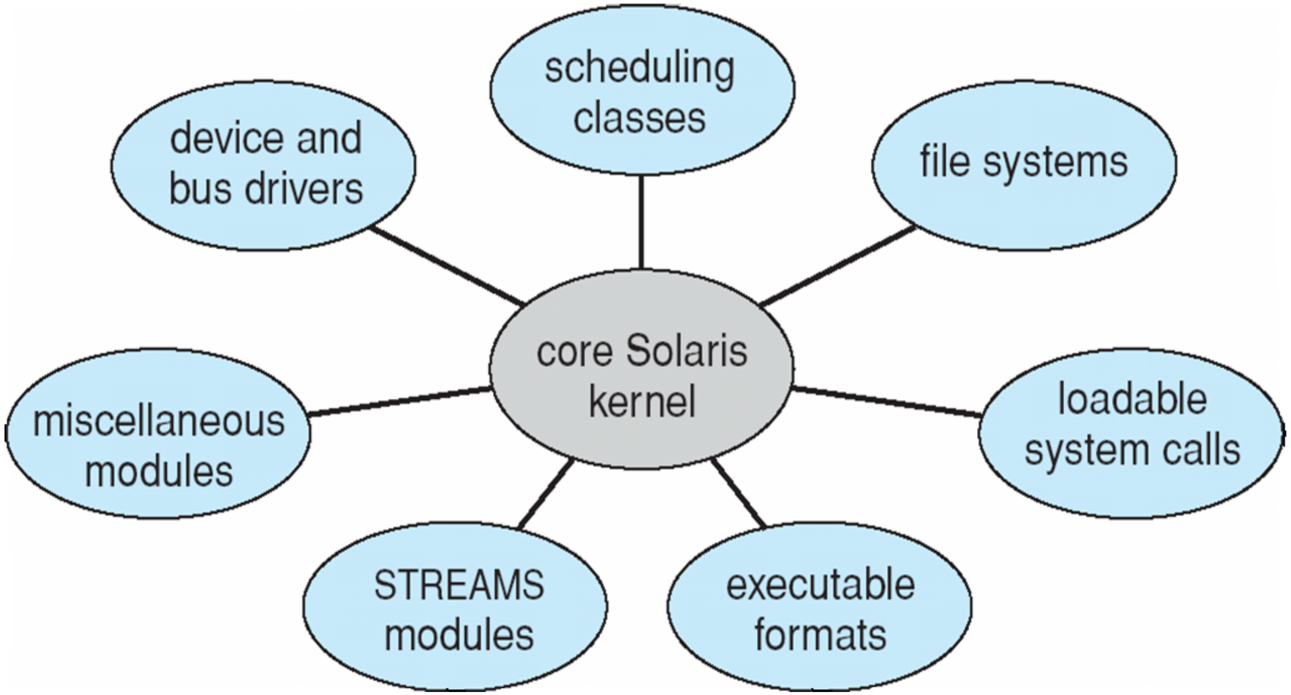
Microkernel

- Moves all the nonessential components from the kernel to user level. Smaller kernel, uses messages with system and user-level programs

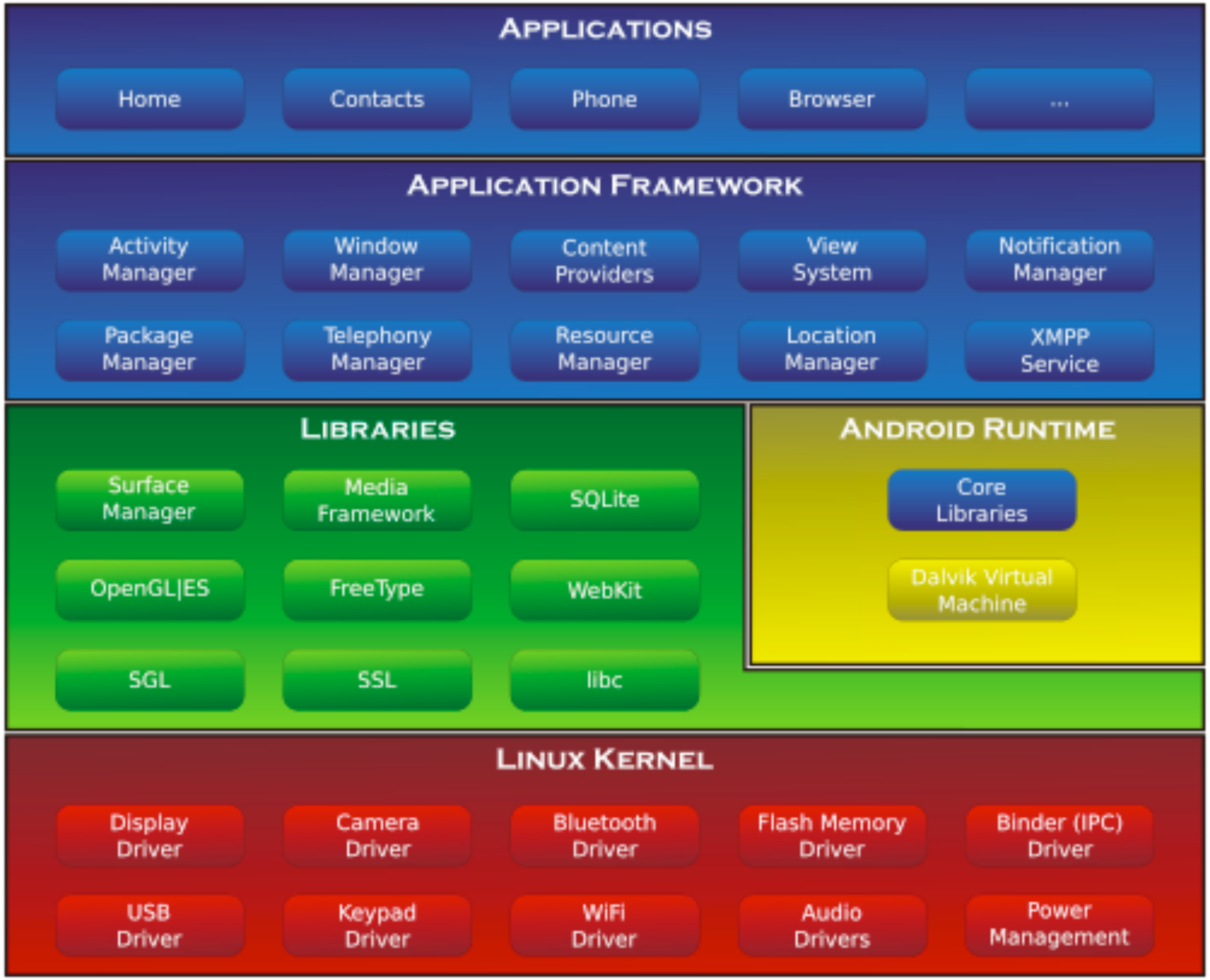


Modular Kernel

- **Modular Approach**: Loadable kernel modules, load additional services if needed at boot or run time



Android



Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service
Libraries
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Question

- Which of the following instructions should be privileged?

  a. Set value of timer.

  b. Read the clock.

  c. Clear memory.

  d. Issue a trap instruction.

  e. Turn off interrupts.

  f. Modify entries in device-status table.

  g. Access I/O device.

- a, c, e, f, g

Question

- A _____ can be used to prevent a user program from never returning control to the operating system.
  A) Dual mode
  B) Program counter
  C) Kernel module
  D) Timer

  D

Reading

- From text book
  - Read Chapter 2: Section 2.1-2.4, 2.10
- Linux System Call References
  - https://man7.org/linux/man-pages/man2/syscalls.2.html
- Kernel source code
  - https://elixir.bootlin.com/linux/latest/source

- Acknowledgments
  - These slides are adapted from
    - Öznur Özkasap (Koç University)
    - Operating System and Concepts (9th edition) Wiley