**Operating System Services**

* **User interface-** Command-Line (CLI), Graphics User Interface (GUI), Batch
* **Program execution-** 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, permission management.
* **Communications**- Processes may exchange information, on the same computer or between computers over a network

-via shared memory or

-through message passing (packets moved by the OS)

* **Error detection -** the CPU and memory hardware, in I/O devices, in user program

-Debugging facilities

-Each type of error. OS should take the appropriate action to ensure correct and consistent computing

* **Resource allocation** - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
* **Accounting** - To keep track of which users use how much and what kinds of computer resources
* **Protection and security**

-**Protection** involves ensuring that all access to system resources is controlled

-**Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

**User Operating System Interface – CLI**

* CLI or **command interpreter** allows direct command entry
* Sometimes implemented in kernel, sometimes by systems program
* Sometimes multiple flavors implemented – **shells**
* Primarily fetches a command from user and executes it
* Sometimes commands built-in, sometimes just names of programs
  + - -If the latter, adding new features doesn’t require shell modification

**User Operating System Interface – GUI**

* User-friendly **desktop** metaphor interface
* Usually mouse, keyboard, and monitor
* **Icons** represent files, programs, actions, etc
* various actions (provide information, options, execute function, open directory (known as a **folder**)
* **Invented at Xerox PARC**
* Many systems now include both CLI and GUI interfaces
* **Microsoft Windows** is GUI with CLI “command” shell
* **Apple Mac OS X** is “Aqua” GUI interface with UNIX kernel underneath and shells available
* **Unix and Linux** have CLI with optional GUI interfaces (CDE, KDE, GNOME)

**Touchscreen Interfaces**

* Actions and selection based on gestures
* Virtual keyboard for text entry
* Voice commands.

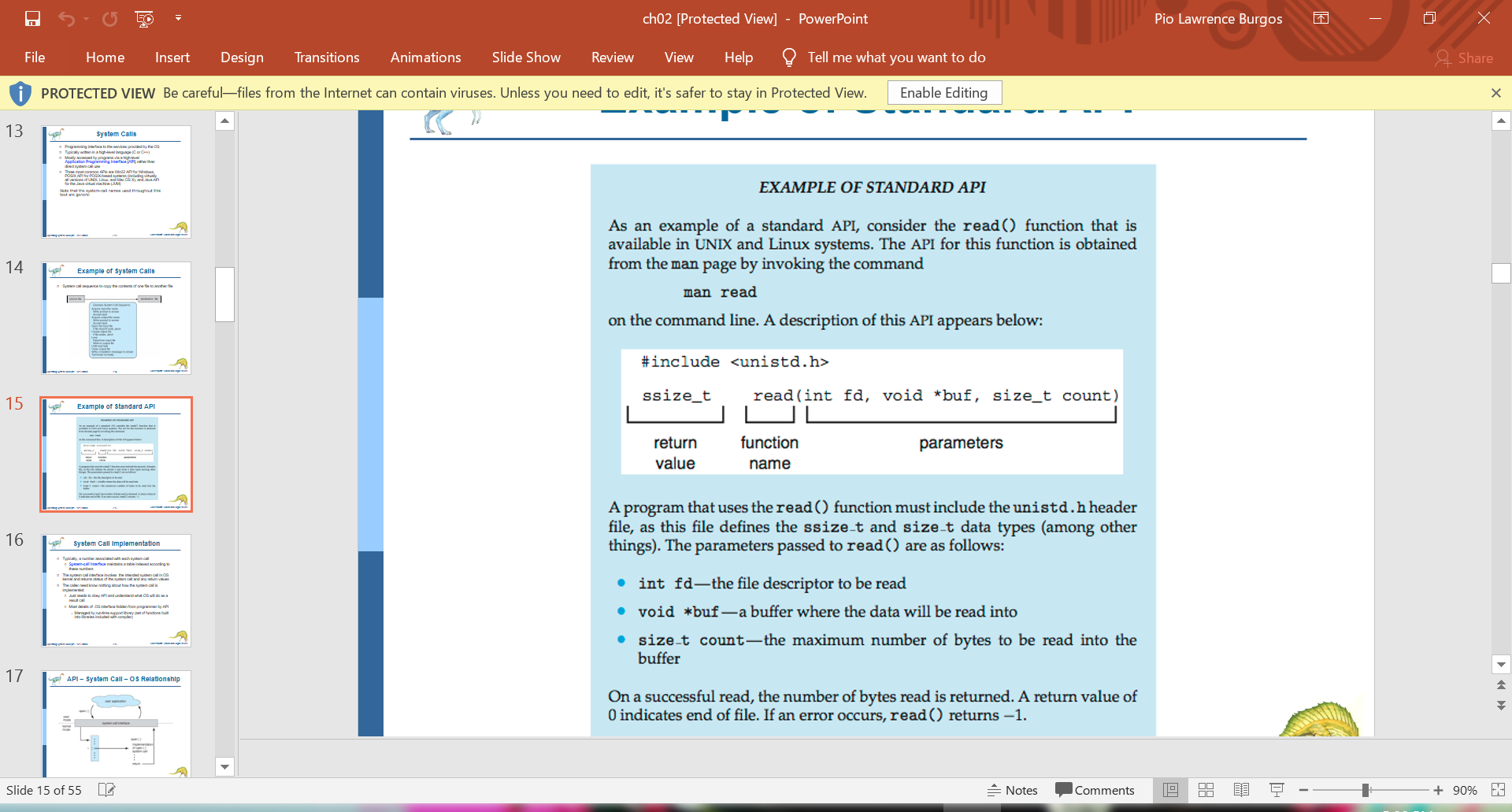
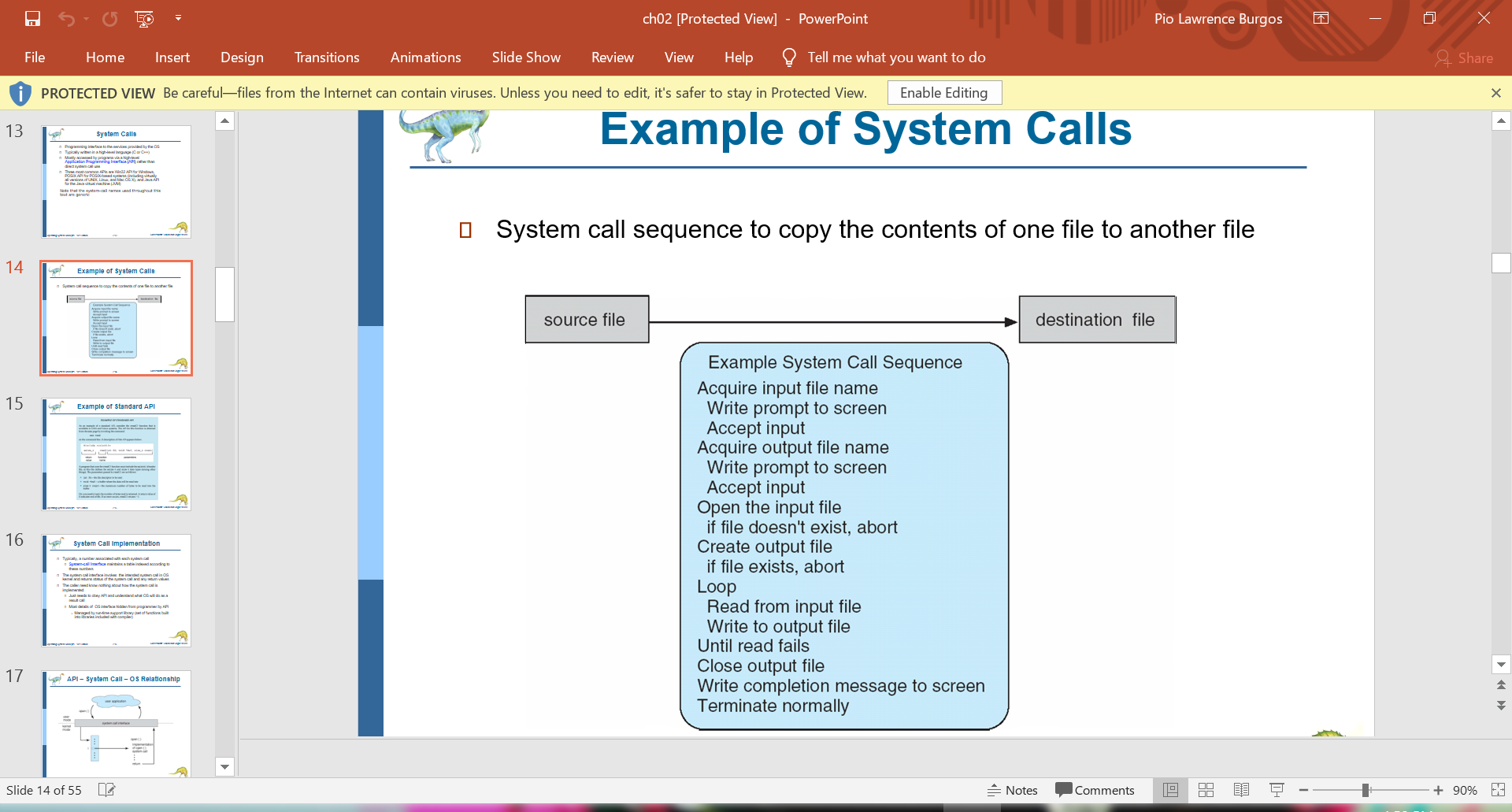
**System Calls**

* In computer science, the mechanism used by an application program to request service from the operating system or another application program
* Typically written in a high-level language (C or C++)
* Mostly accessed by programs via a high-level **Application Programming Interface (API)**
* Three most common APIs are:

- Win32 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)



**System Call Implementation**

* Typically, a number associated with each system call
* **System-call interface** maintains a table indexed according to these numbers
* The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
* The caller need know nothing about how the system call is implemented
  + Just needs to obey API and understand what OS will do as a result call
  + Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

**System Call Parameter Passing**

Three general methods used to pass parameters to the OS

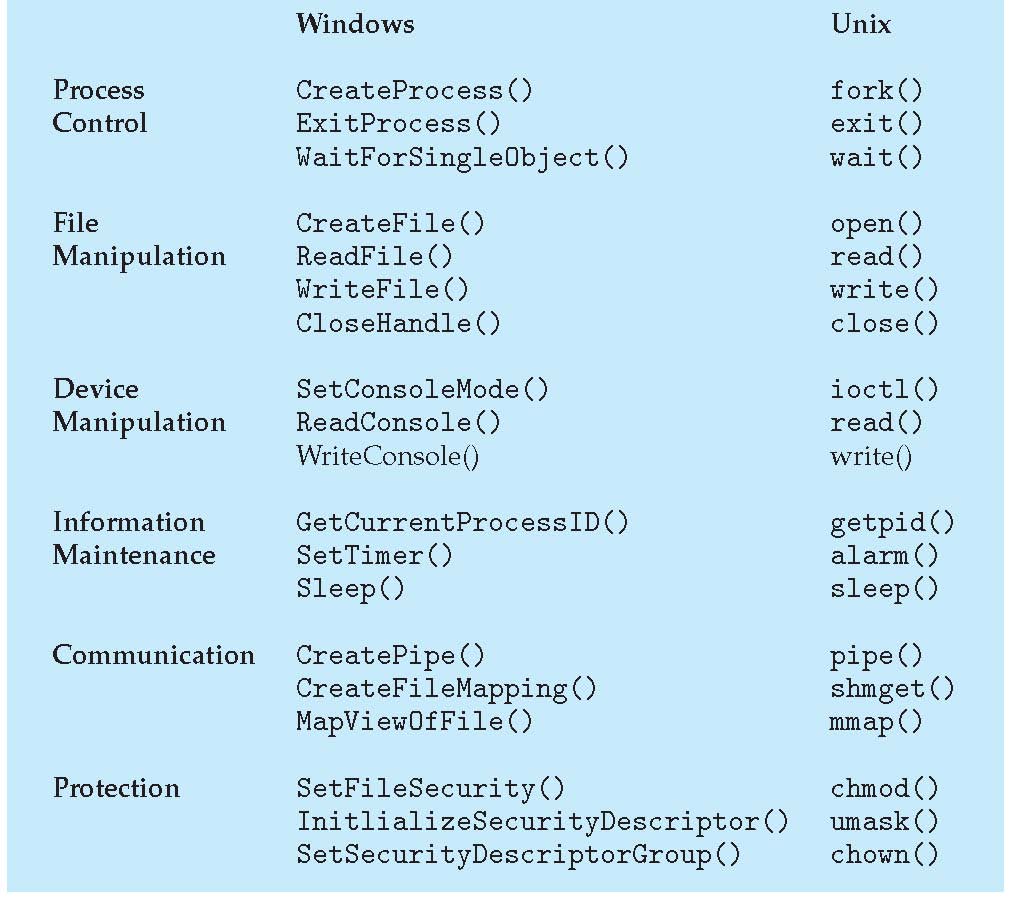
* **Simplest:** pass the parameters in registers
* In some cases, may be more parameters than registers
* **Parameters stored in a block*,* or table**, in memory, and address of block passed as a parameter in a register
* This approach taken by Linux and Solaris
* Parameters placed, or **pushed***,* onto the **stack**by the program and **popped**off the stack by the operating system
* Block and stack methods do not limit the number or length of parameters being passed

**Types of System Calls**

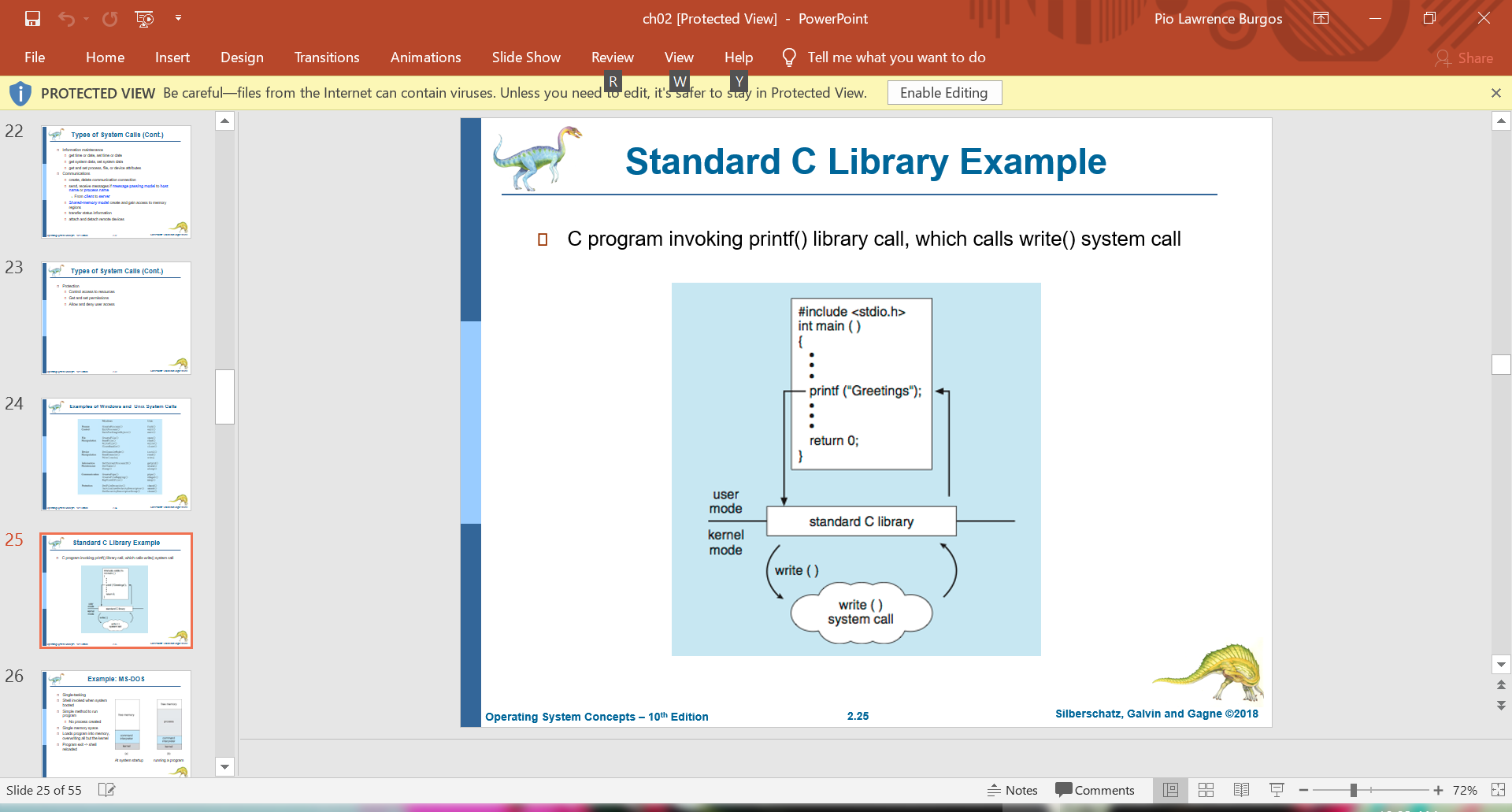
* **Process control**
* create process, terminate process
* end, abort
* load, execute
* get process attributes, set process attributes
* wait for time
* wait event, signal event
* allocate and free memory
* Dump memory if error
* **Debugger** for determining bugs, single step execution
* **Locks** for managing access to shared data between processes
* **File management**
* create file, delete file
* open, close file
* read, write, reposition
* get and set file attributes
* **Device management**
* request device, release device
* read, write, reposition
* get device attributes, set device attributes
* logically attach or detach devices
* **Information maintenance**
* get time or date, set time or date
* get system data, set system data
* get and set process, file, or device attributes
* **Communications**
* create, delete communication connection
* send, receive messages if **message passing model** to **host name** or **process name**

From client to server

* **Shared-memory model** create and gain access to memory regions
* transfer status information
* attach and detach remote devices
* **Protection**
* Control access to resources
* Get and set permissions
* Allow and deny user access

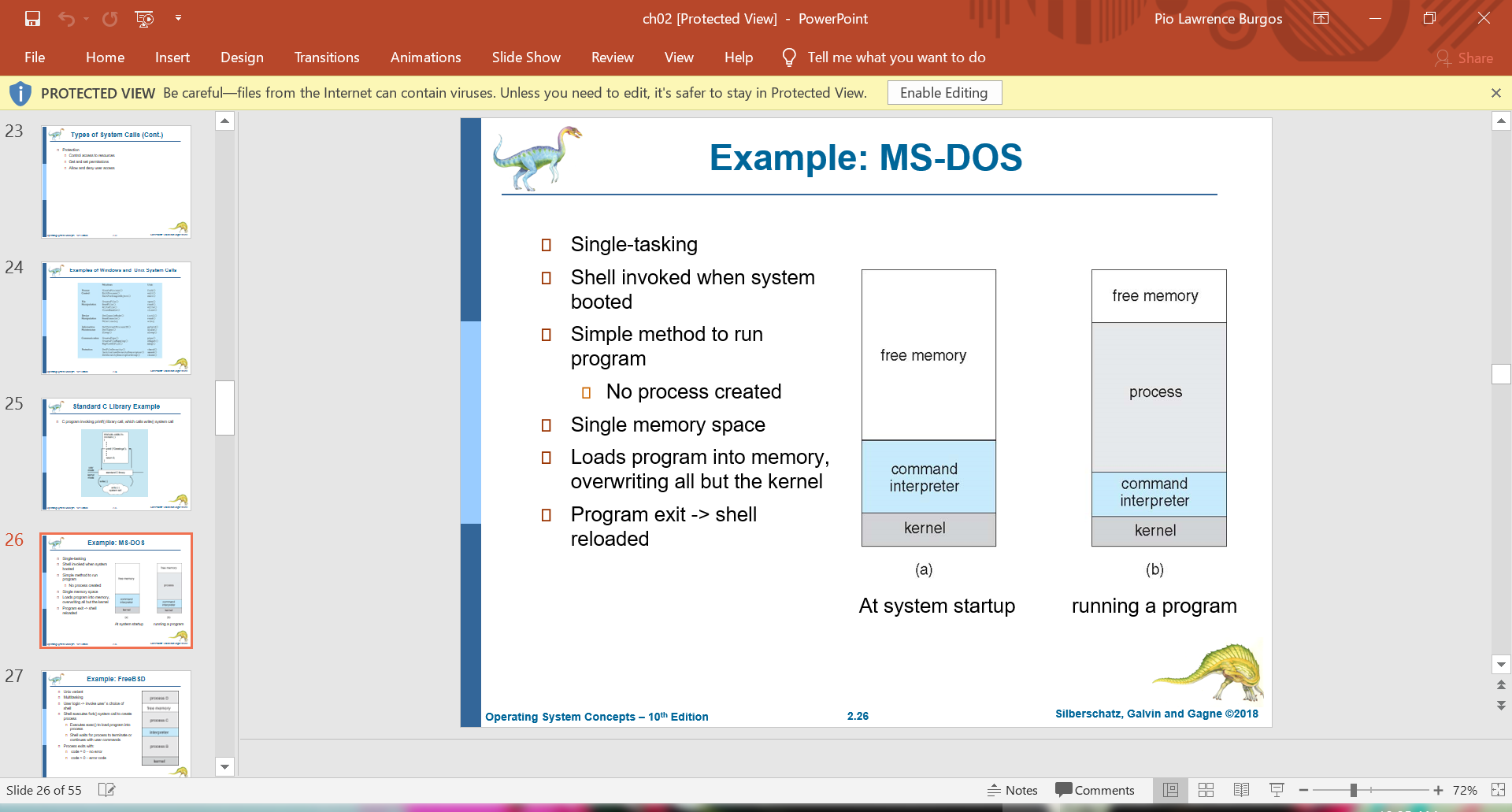


**Standard C Library Example:**

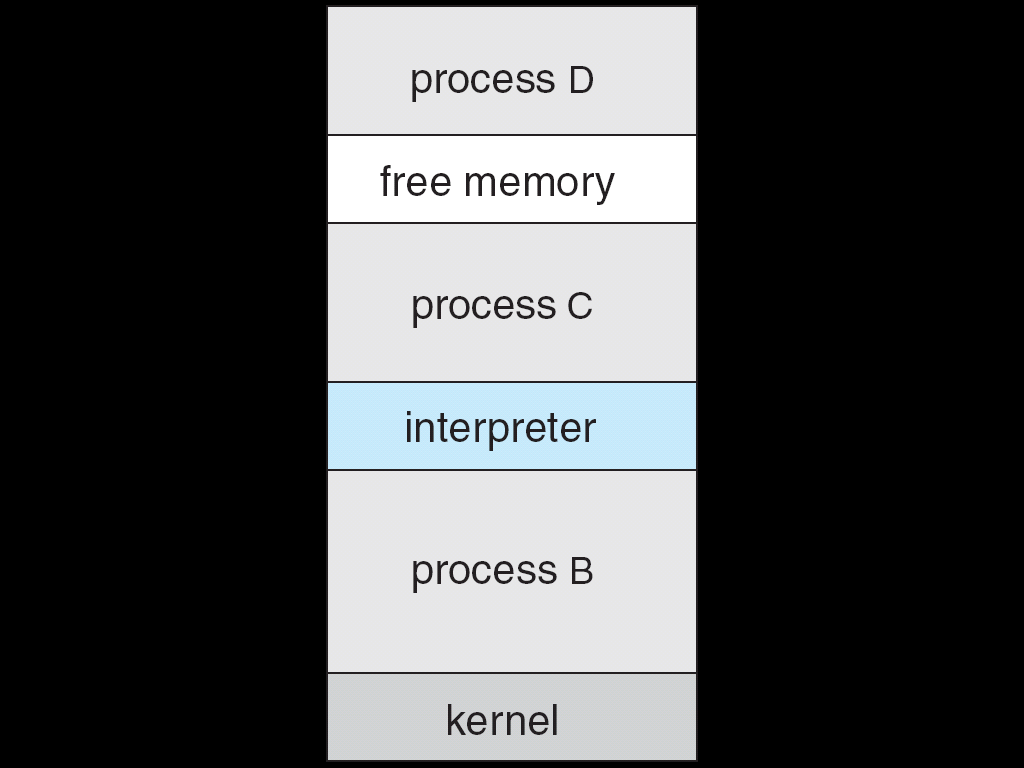
C program invoking printf() library call, which calls write() system call

**Example: MS-DOS**

* **Single-tasking**



**Example: FreeBSD**

* **Multitasking**

**System Programs**

* File manipulation
* Status information sometimes stored in a File modification
* Programming language support
* Program loading and execution
* Communications
* Background services
* Application programs
* Most users’ view of the operation system is defined by system programs, not the actual system calls
* Provide a convenient environment for program development and execution
* Some of them are simply user interfaces to system calls; others are considerably more complex
* **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
* **Status information**
* **Some ask the system for info** - date, time, amount of available memory, disk space, number of users
* Others provide detailed performance, logging, and debugging information
* Typically, these programs format and print the output to the terminal or other output devices
* Some systems implement a **registry** - used to store and retrieve configuration information
* **File modification**
* Text editors to create and modify files
* Special commands to search contents of files or perform transformations of the text
* **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
* **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
* **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
* Allow users to send messages to one another’s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another
* **Background Services**
  + Launch at boot time

Some for system startup, then terminate

Some from system boot to shutdown

* + Provide facilities like **disk checking, process scheduling, error logging**, **printing**
  + Run in user context not kernel context
  + Known as **services**, **subsystems**, **daemons**
* **Application programs**
  + Don’t pertain to system
  + Run by users
  + **Not typically considered part of OS**
  + Launched by command line, mouse click, finger poke