Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

Operating Systems

Schedule

- **6 more classes left (after today)!**
- **Quiz 6** – Tuesday, Nov 22nd
  - Input / Output (HW #15)
  - Operating Systems (HW #16)
  - Compilers & Assemblers (HW #16)
  - **Review the lecture notes before the quiz (not just the homework!)**
  - **Bring a Calculator**

Solid State Disks (SSDs)

- See Slides from Thursday

Operating Systems

Introduction

- **Beginning Chapter 8**
- System software – in the form of **operating systems** and **middleware** – is the glue that binds user applications and hardware together

Operating Systems

- **What software first runs when you turn on your machine?**
- Not the OS! The **BIOS** (basic input-output system)
  - Stored on flash memory chip at known location
  - Examines system configuration
    - How many CPUs are installed?
    - How much memory is installed?
    - Where is the video card / keyboard / mouse / hard drive?
  - Assigns devices memory addresses and initializes them
  - Locates OS on disk, loads it into memory, and executes it
- A BIOS permits a single operating system to function on different computers (with different peripherals)

Operating Systems

- The evolution of operating systems has paralleled the evolution of computer hardware
  - As hardware became more powerful, operating systems allowed people to more easily manage the power of the machine
- In the days when main memory was measured in kilobytes, and tape drives were the only form of magnetic storage, operating systems were simple **resident monitor** programs
  - The resident monitor could only **load**, **execute**, and **terminate programs** (command-line only!)

Operating Systems

- In the 1960s, hardware has become powerful enough to accommodate **multiprogramming**, the concurrent execution of more than one task.
  - Multiprogramming is achieved by allocating each process a given portion of CPU time (a *timeslice*)
  - Systems were still *batch* oriented – submit your job, and wait hours to see the results
- Interactive multiprogramming systems were called **timesharing** systems
  - You now get a *terminal* to interact with computer directly!
  - When a process is taken from the CPU and replaced by another, we say that a **context switch** has occurred

Operating Systems

- Multiprogrammed and timesharing systems require a more complex operating system
- How to handle a context switch?
  - Save all data from current running process
    - Data includes CPU registers, page table, etc…
  - Load all data from new running process
  - … let it run for a while …
  - Save all data from current running process
  - Restore data from the previous running process

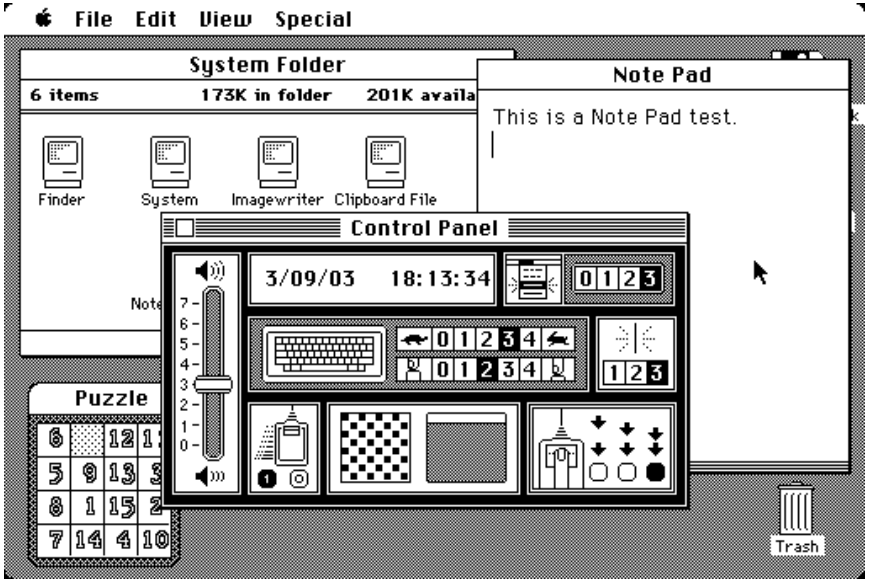
Operating System Tasks

Operating System Tasks

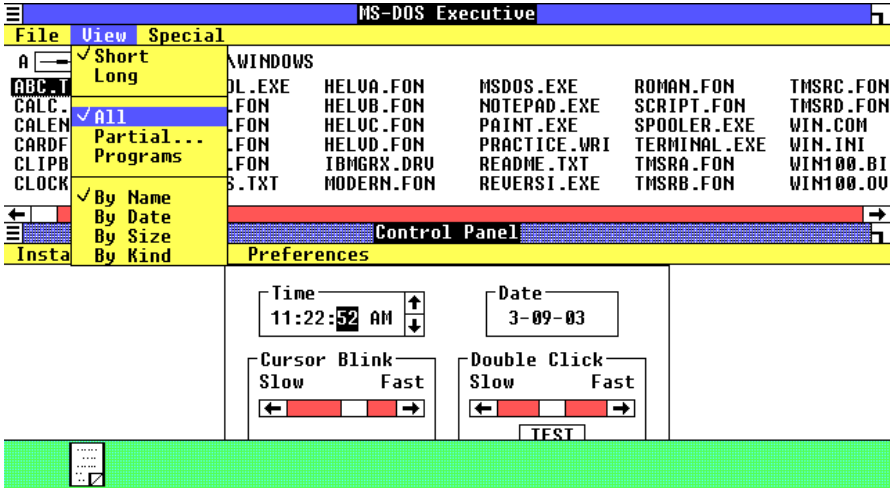
- **What does the OS need to do?**
  - Schedule processes to run
  - Memory management
  - Interrupt handling (manage hardware in general)
  - Security (between processes)
  - Network access
  - Storage management (filesystem)
  - Graphical user interface
    - May be a **middleware** layer on top of the OS

Operating Systems – GUI

- Operating systems with graphical user interfaces (GUI) were first brought to market in the 1980s



Apple Mac OS 1.0 (released 1984)

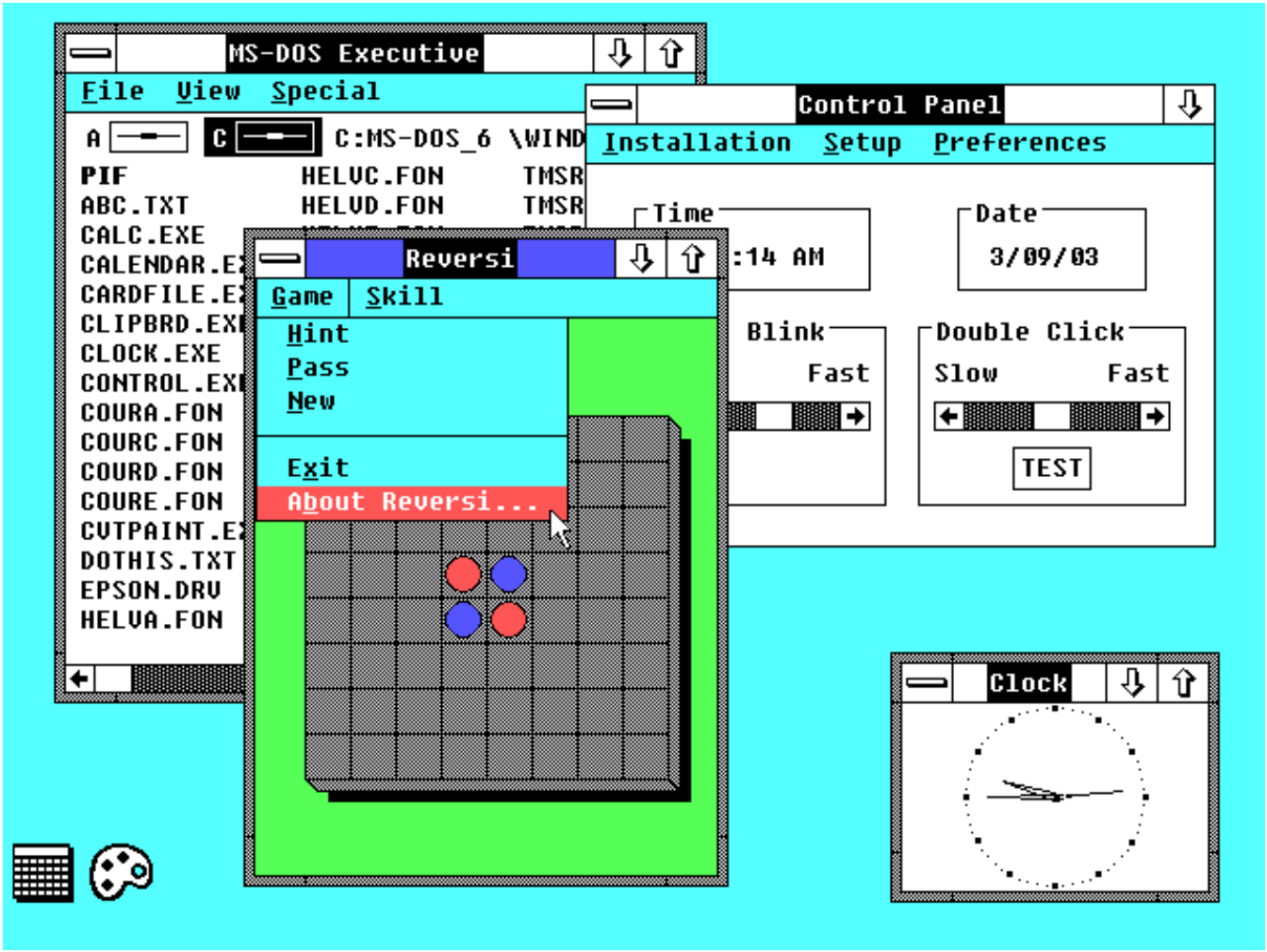


Microsoft Windows 1.0 (released 1986)

Captures from http://www.guidebookgallery.org/screenshots

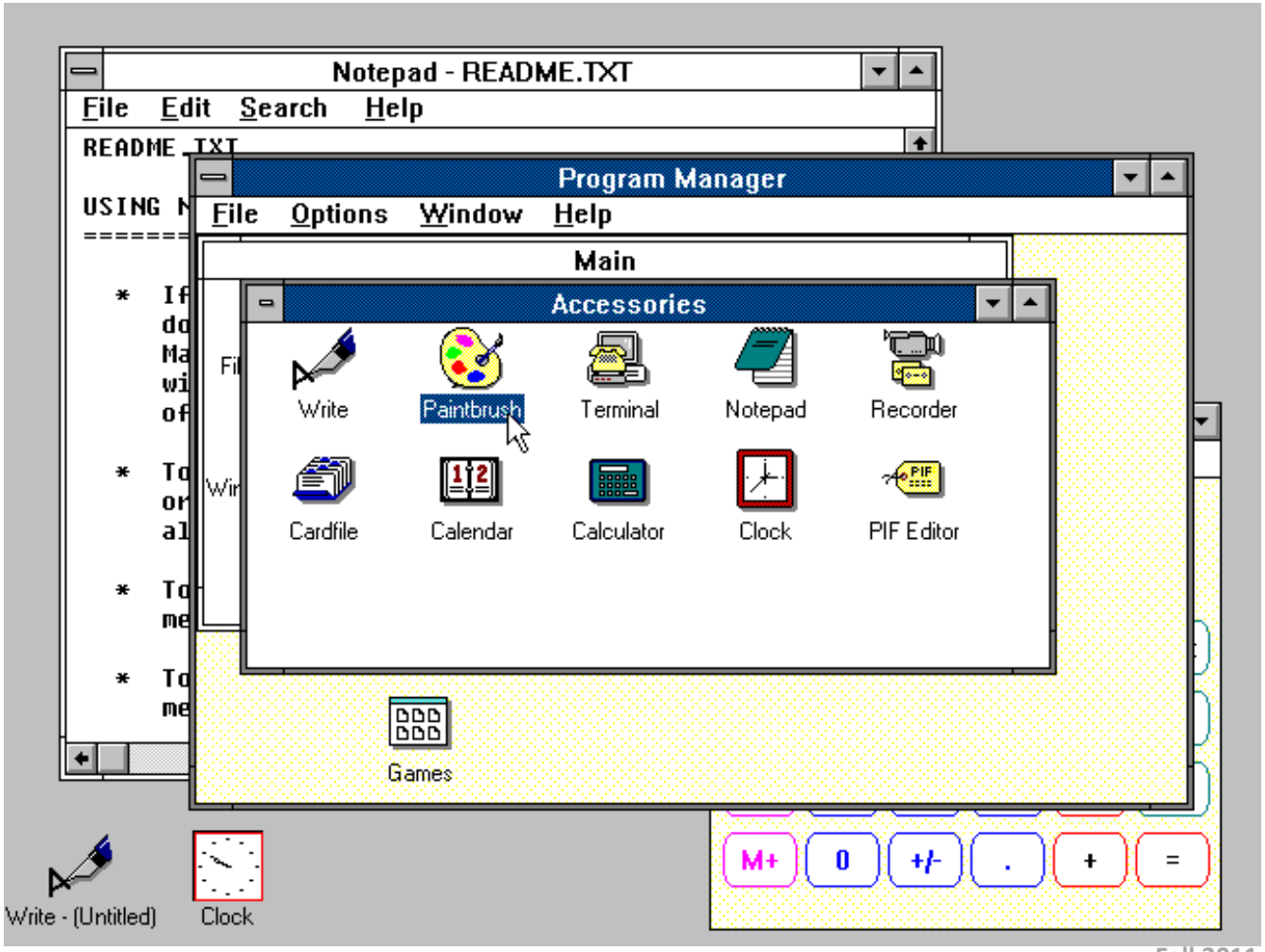
Operating Systems – GUI

- Windows 2.0.3
- Released 1987



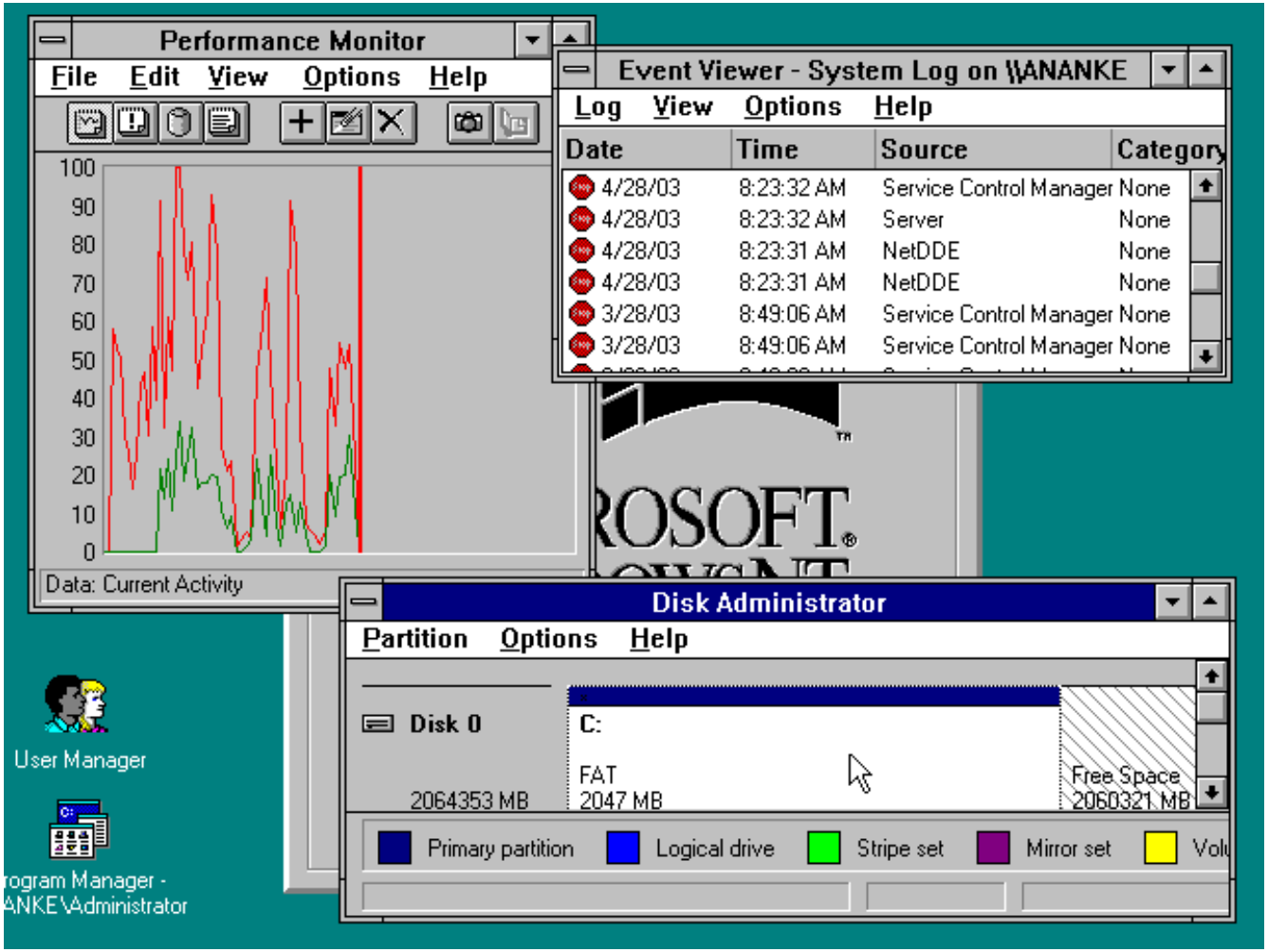
Operating Systems – GUI

- Windows 3.0
- Released 1990



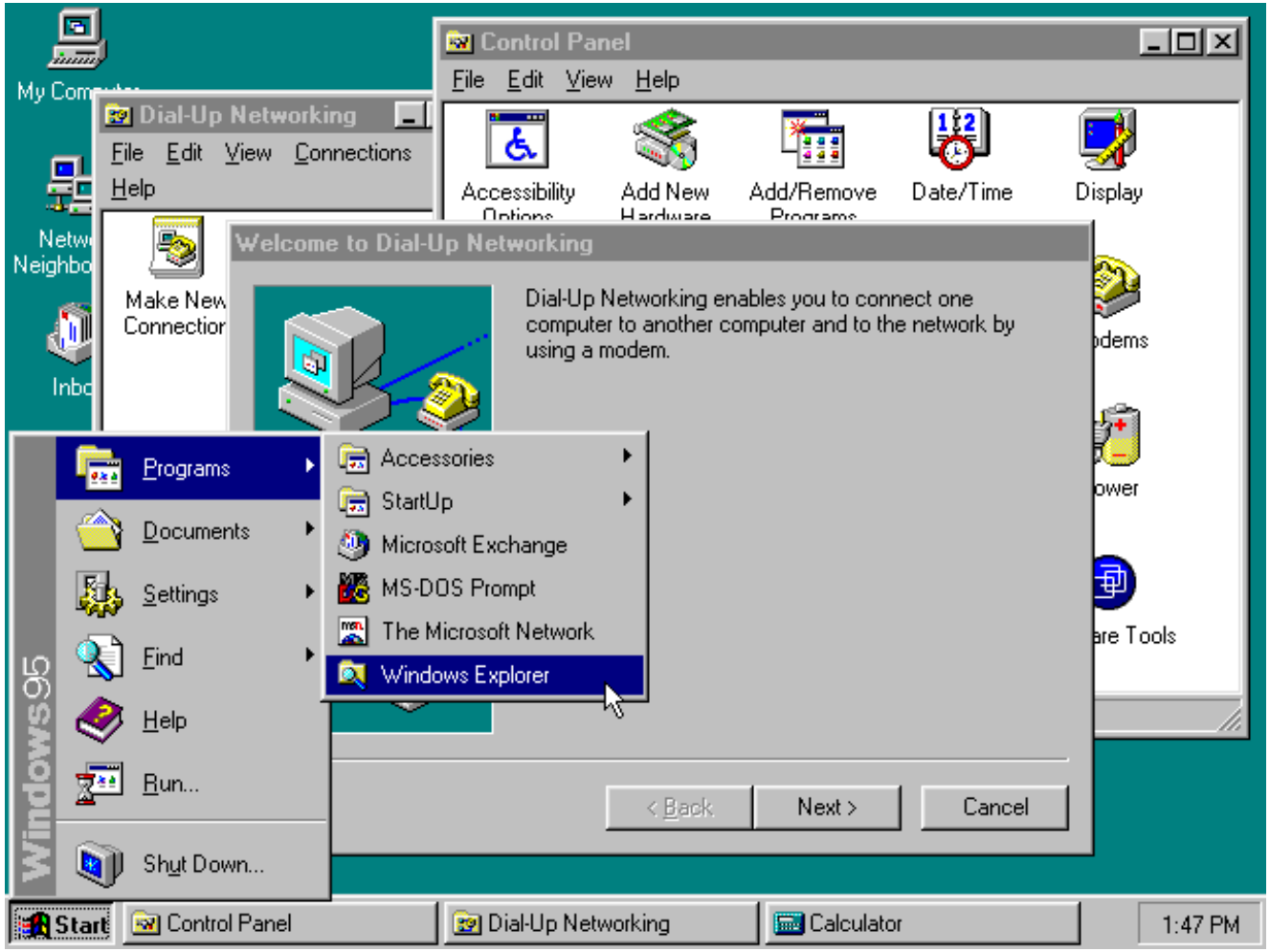
Operating Systems – GUI

- Windows NT 3.1
- Released 1993



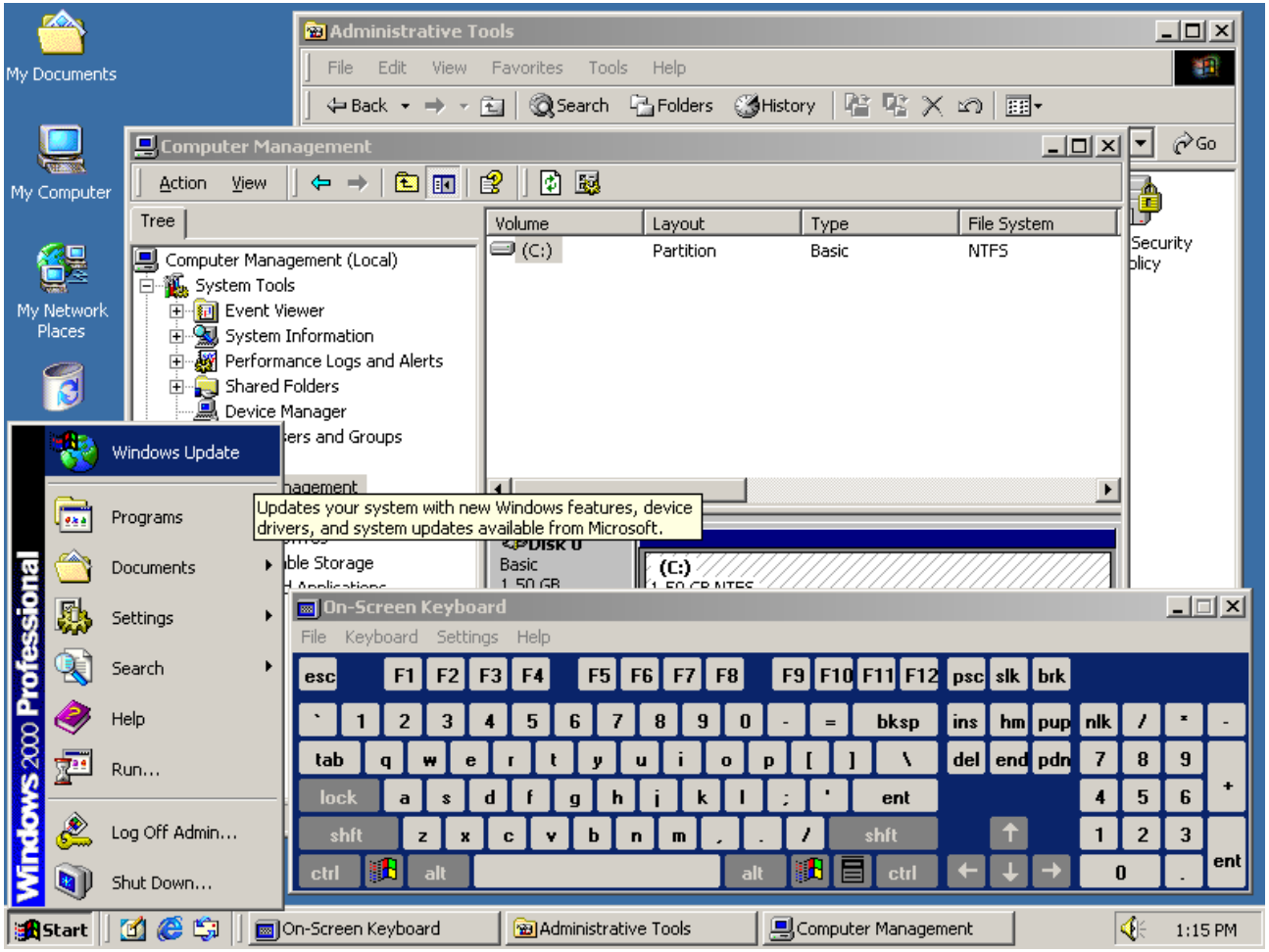
Operating Systems – GUI

- Windows 95
- Released 1995



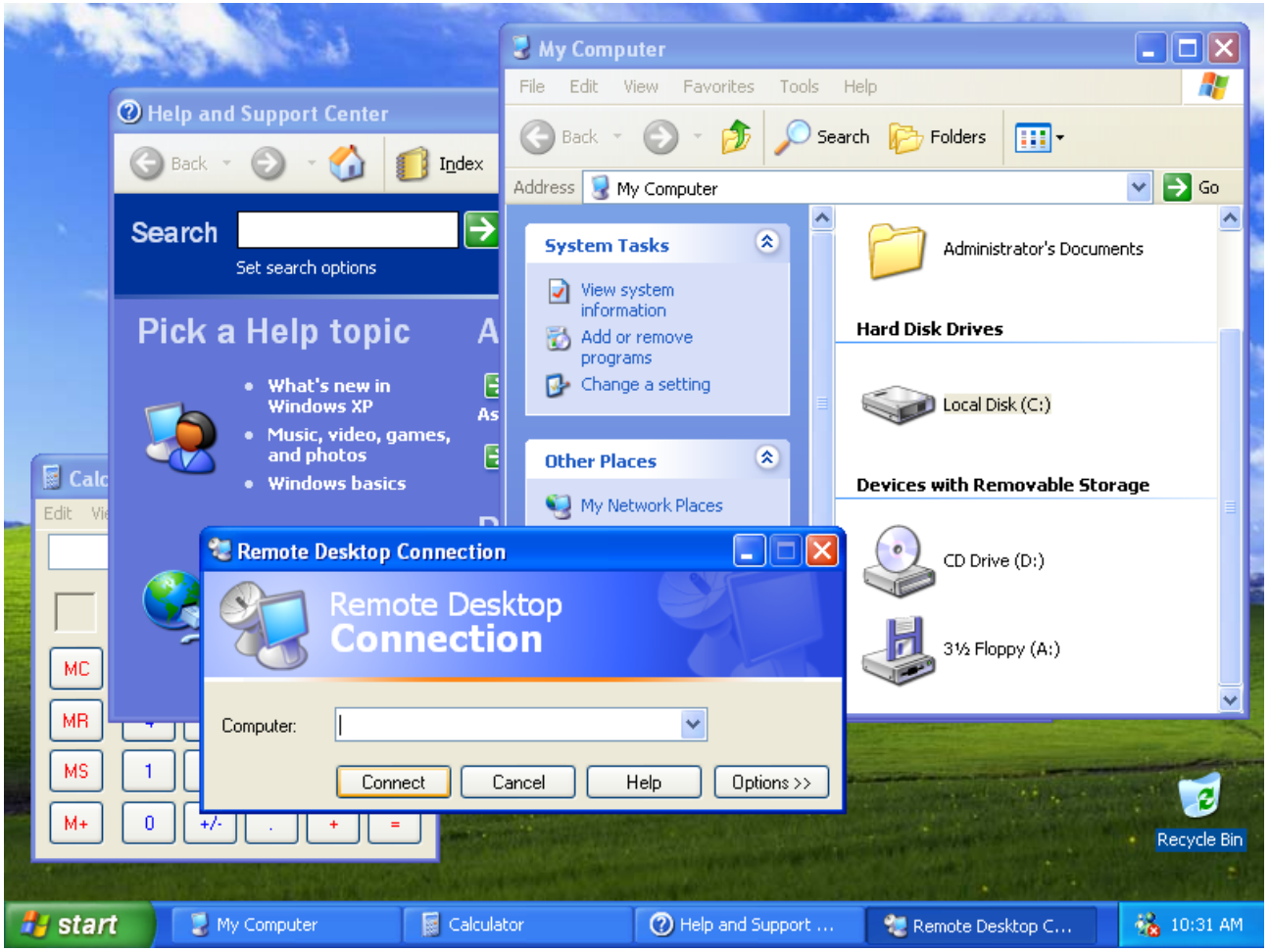
Operating Systems – GUI

- Windows 2000
- Released 2000



Operating Systems – GUI

- Windows XP
- Released 2001



Operating Systems – GUI

- From a technical perspective, the GUI is one of the least important parts of the operating system!
  - *But to the users, it's the most important part*
- A GUI does not even have to be part of the *true* OS at all
  - Windows 1.0 was just a program that ran on top of DOS, the *true* operating system (of that era)

Operating Systems – Processes

- **Process management** is a key operating system task
- OS must initially **create processes**
  - When you run your program!
- OS can allow processes to **access resources**
  - If resources are shared (e.g. CPU), the OS must *schedule* access to them
- OS can allow processes to **communicate** with each other
  - OS provides the mechanisms to do this
- OS must **clean up** after process finishes
  - Deallocate resources (e.g. memory, network sockets, file descriptors, etc…) that were created during process execution

Operating Systems – Scheduling

- The operating system schedules process execution
- First, the operating system determines which process shall be granted access to the CPU
  - This is **long-term scheduling**
- After a number of processes have been admitted, the operating system determines which one will have access to the CPU at any particular moment
  - This is **short-term scheduling**
- **Context switches** occur when a process is taken from the CPU and replaced by another process
  - CPU *state* (registers, current PC, etc…) is preserved during a context switch

Operating Systems – Scheduling

- Short-term scheduling can be **non-preemptive** or **preemptive**
  - **Non-preemptive** scheduling – a process has use of the CPU until either it terminates, or must wait for resources that are temporarily unavailable
  - **Preemptive** scheduling – each process is allocated a timeslice. When the timeslice expires, a context switch occurs
    - A context switch can also occur when a higher-priority process needs the CPU
- **Which method do you think is better? (consider complexity to implement -vs- reliability)**

Operating Systems – Scheduling

- Four approaches to CPU scheduling:
  - **First-come, first-served** – Jobs are serviced in arrival sequence and run to completion if they have all of the resources they need
  - **Shortest job first –** Smallest jobs get scheduled first. (The trouble is in knowing which jobs are shortest!)
  - **Round robin** - Each job is allotted a certain amount of CPU time. A context switch occurs when the time expires
  - **Priority** scheduling preempts a job with a lower priority when a higher-priority job needs the CPU
- **Which method(s) do modern operating systems use?**

Operating Systems – Security

- Process A is forbidden from reading/modifying/writing the memory of Process B
  - **Virtual memory** is a huge help here!
  - OS must set up virtual memory (establish page table) for each process and handle page faults when the occur
  - **Wait – what if I want my two programs to share memory?**
- Process A has other limits besides which pages it can access
  - **Ideas of other limits?**
  - Amount of memory consumed
  - Number of open files on disk
  - Which files on disk can be read/written

Operating Systems – Filesystem

- OS is responsible for managing data on persistent storage
- Job of the **filesystem!**
  - What files exist? (i.e. names)
  - How are they organized? (i.e. paths/folders)
  - Who owns and can access them? (i.e. usernames, permissions)
  - Where are individual file blocks stored on the disk?
    - i.e. filename “database.dat” is really composed of 15823 blocks, where block 1 is located at position …

Operating Systems – Device Management

- Manage devices
  - How do we send data to the NIC for transmission?
  - How do we render an image for display on screen?
  - How do we read a block of data from our RAID disk controller?
- Operating systems can be extended through **device drivers** to manage new hardware
  - Hardware vendors write software to manage their devices
  - OS provides a fixed interface (API) that driver must follow
- Common task for a device driver is **responding to interrupts** (from that device)

Operating Systems – The Kernel

- Who does all this essential work in the operating system? (besides the GUI)
  - The **kernel** (i.e. the heart or core of the OS)
- Kernel performs scheduling, synchronization, memory management, interrupt handling, security and protection, etc…
- Two extremes on the design space
  - **Microkernel** systems – Kernel provides minimal functionality and most services are carried out by external programs
  - **Monolithic** systems – Single kernel (program) provides most of their services directly

Operating Systems – The Kernel

- **Microkernel** systems provide better security, easier maintenance, and portability at the expense of execution speed
  - Examples are Windows 2000, Mach, and QNX.
- **Monolithic** systems give faster execution speed, but are more difficult to port from one architecture to another
  - Examples are Linux, MacOS, and DOS
- Modern systems (Mac OS X, Windows XP/Vista/7) are hybrids that blend both approaches

Operating System Varients

- Several sub-categories of operating systems for special purpose applications
- **Real-time Operating System**
- **Mobile Operating System**

Real-Time Operating Systems

- Some I/O devices need to be controlled in real-time, where their correctness depends on
  - Output being logically correct, **and**
  - Output occurring at correct time
- Car processor must read sensors and decide whether to deploy airbags in 15-30 milliseconds
  - Similar story for traction control, ABS, etc…
  - Bad time to be running garbage collection algorithm over the memory!
  - Close enough is not good enough
- **Real time operating systems (RTOS)** are necessary for these types of systems

Real-Time Operating Systems

- Operating system can provide time guarantees
  - **Hard real time** systems have **tight** timing constraints
  - **Soft real time systems** have weaker guarantees (but are easier to create!)
- **Do we want virtual memory on a RTOS?**
- **Do we want long-running interrupts on a RTOS? (or any interrupts?)**
- **Do we want garbage collection (memory management) on a RTOS?**

Mobile Operating Systems

- Compare the mobile operating system on your phone (e.g. iOS, Android) versus the operating system on your laptop (e.g. Windows 7, Mac OS X)
- **What does a mobile OS do that is similar to a desktop OS?**
- **What does a mobile OS do that is different to a desktop OS?**