ELEC 377 – Operating Systems

Week 1 – Class 2

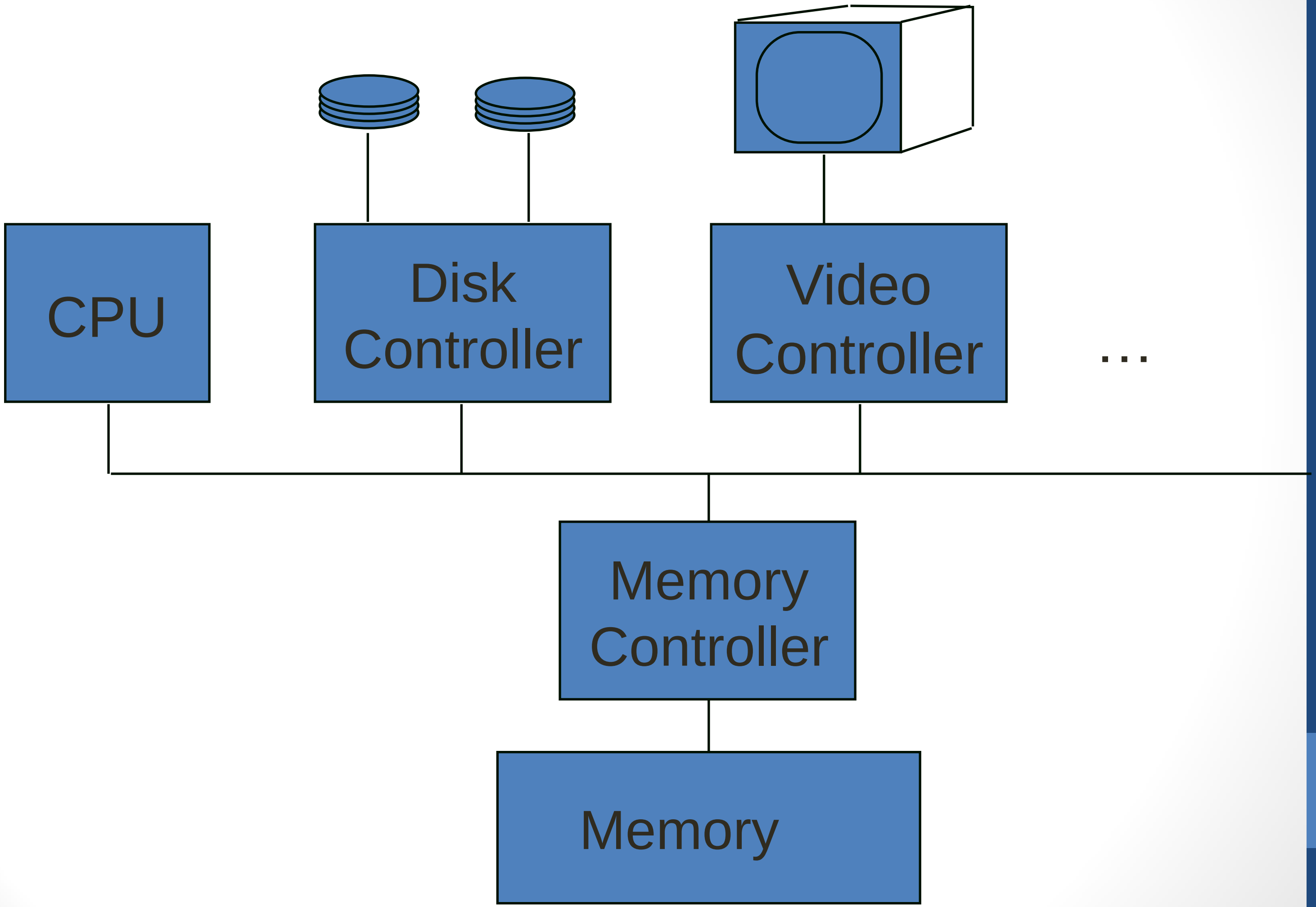
Labs vs. Assignments

- The only work to turn in are the labs. In some of the handouts I refer to the labs as assignments. There are no assignments separate from the labs.
- With the exception of the first introductory lab, the labs are due the Thursday after the last lab period for the lab. This is to allow you to write up the documentation for the lab.

Announcements

- Various announcements are also posted to the class website, so you should check there on a regular basis.

Computer System Structures



Computer System Structure

- CPU and devices execute concurrently
- Each controller is in charge of a one or more devices of a similar type
- Device controller has some sort of local buffer that is used to communicate data to and from the device
- Either CPU or device controller moves data to and from main memory
- Interrupt used to indicate the operation is complete

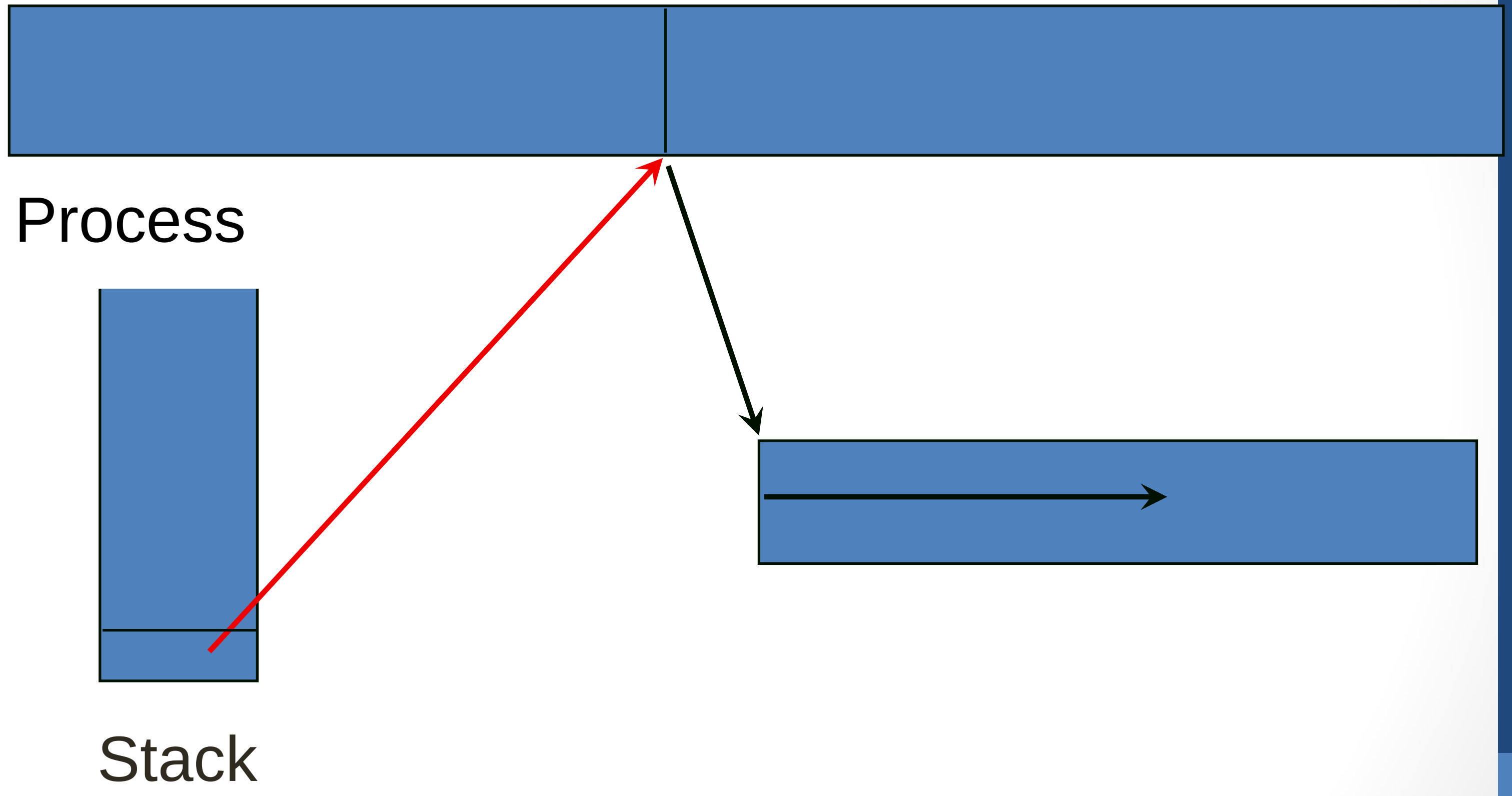
Interrupts

- Suspends execution of the current program (saves current execution location)
- Transfers control to the interrupt routine
- Determines which controller generated the interrupt
  - ◊ Polling
  - ◊ Vectored Interrupts
- Separate routines for each type of interrupt

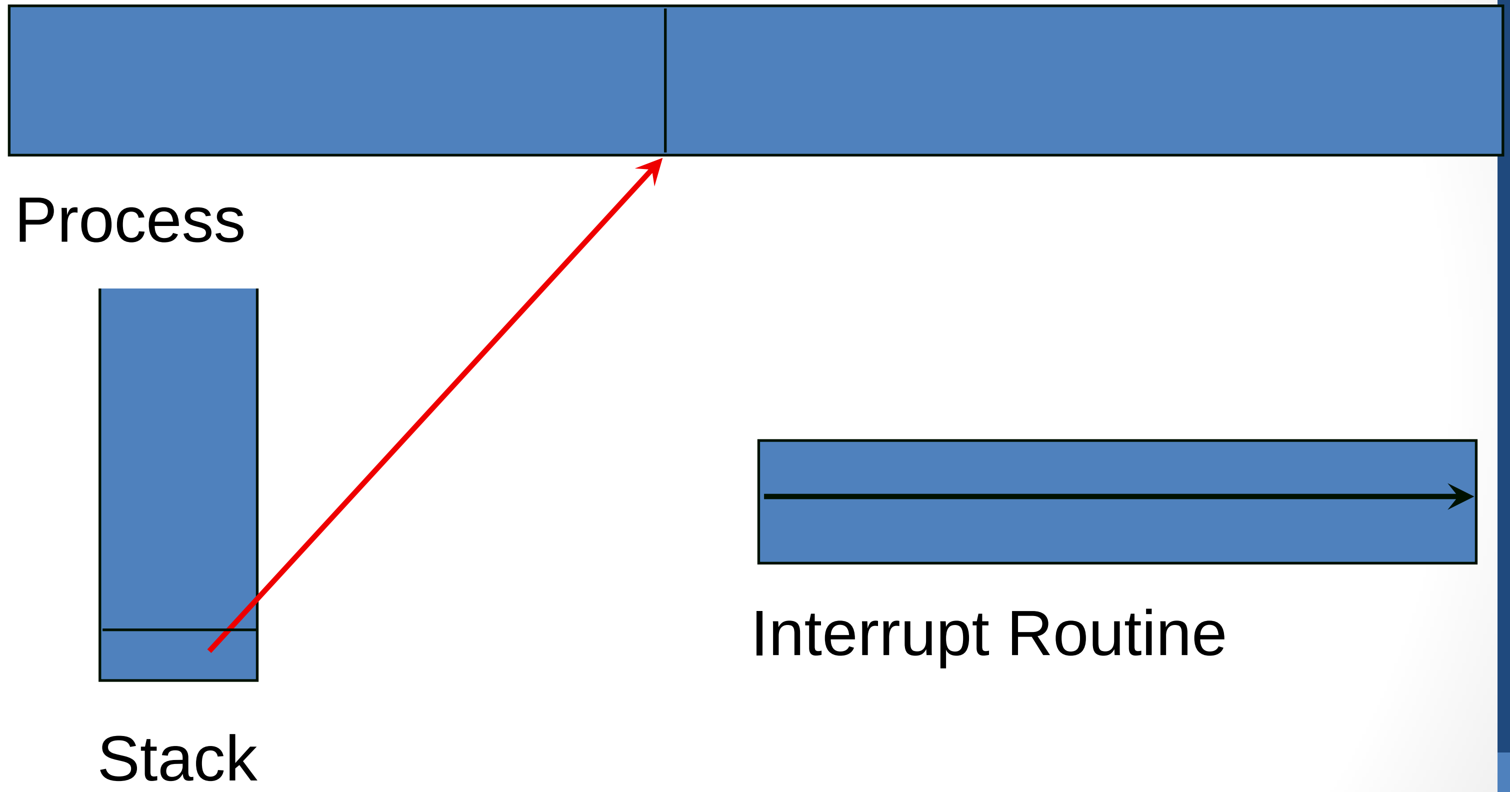
Interrupts

Process

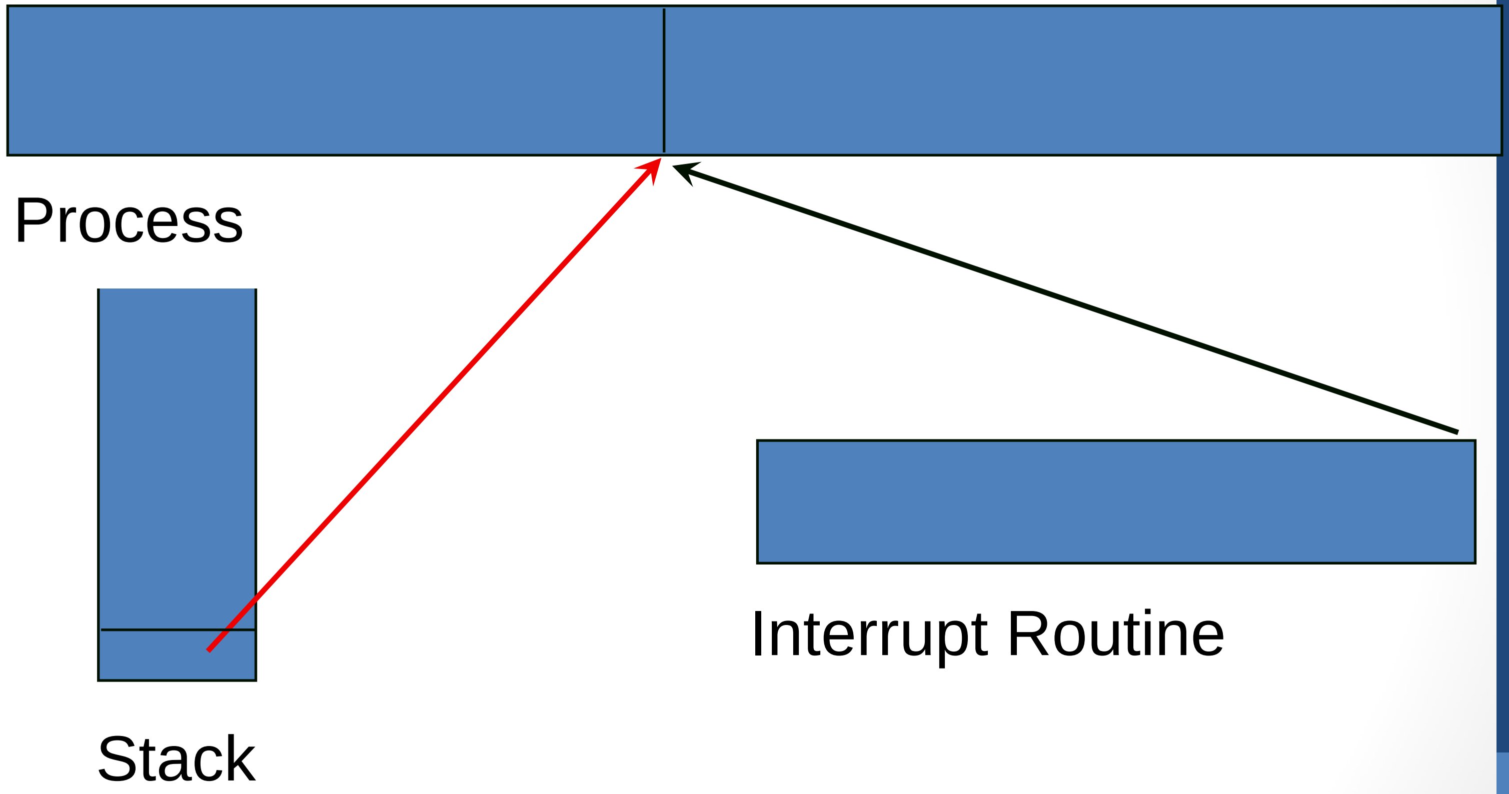
Interrupts



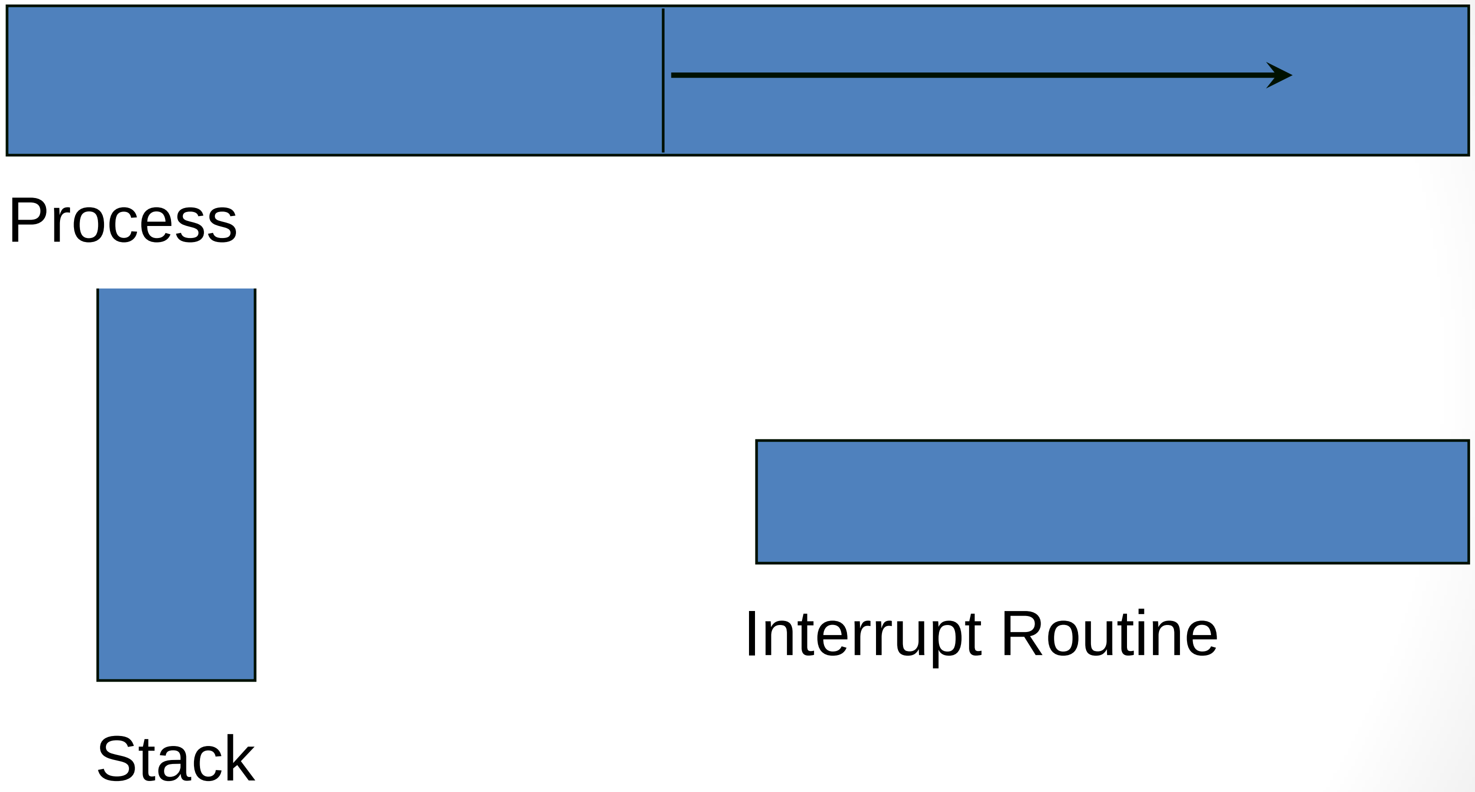
Interrupts



Interrupts



Interrupts



Interrupts & Traps

- Interrupts are asynchronous
  ◊ They can happen at any time
- *Traps* are like interrupts, but are generated by the program executing.
  ◊ Division by Zero, other exceptions
  ◊ Trap instruction to request a service from the operating system
- Traps typically use the same mechanism as interrupts for management.
- Interrupts and Traps drive the operating system

Low level I/O Paradigms

- Synchronous I/O
  - control returns to program only after I/O is complete
  - wait instruction or polling
  - more efficient to give the CPU to another process
- Asynchronous
  - return directly to program (explicit wait call or callback)
  - OS gives the CPU to another program
    - process is synchronous, but OS is asynchronous
  - If nothing else ready, whole system waits

I/O Queues

- More than one process may request I/O from a controller (e.g. more than one program reading a file)
- If the device is busy, then the I/O request is queued. When the interrupt for that device occurs, the next request is removed from the queue and given to the device controller.
- In the example, when one block has been read for one program, the next read request is passed to the disk controller

Storage Structure

- Main Memory
  - CPU can access directly
- Secondary Storage
  - large nonvolatile storage
  - different types have different speeds/costs
  - Minidisk/drum/electronic disk fastest
  - Tapes slowest
  - Hard disks in the middle
- Cache
  - High speed memory between CPU and main memory. Offsets speed difference between memory and CPU
  - May be more than one level

Caching

- Cache is a buffer between the CPU and main memory.
- Programs and data loaded from disk to memory and then data stored back to the disk, so memory can be viewed as a cache of disk space.
- AS/400 (IBM midrange computer) Persistent Object Store. System comprised entirely of objects on disk. Memory is purely a cache for CPU execution. Data files are just memory objects. Ram is invisible to the user.

Hardware Protection

- In multi-user systems, protection between programs is imperative. Important in single user systems too.
- Memory Protection - your e-mail is in memory while you read it. Someone else on the same machine should not be able to peek at it. Same for marks in QCARD.
- I/O Protection. Programs should not be able to examine or change I/O devices. Your e-mail may also be on disk.
- CPU Protection – CPU has privilege resources that must be protected as well.

Hardware Protection

- Solution is multiple levels of privilege. A minimum of two levels is required. Some systems have more. The hardware used by MULTICS had 8 levels of protection, all of which were used by the operating system. We will look at two levels
- Two levels

  ◊ User Mode

  – User programs doing user computation

  ◊ Kernel Mode (monitor mode, system mode, supervisor mode)

  – System operations (i.e. task switching), including user requests (i.e. open a file)

Dual-Mode Operation

- Interrupt or Trap causes the system to suspend the user program and move to kernel mode.
- Returning from the interrupt or trap causes the system to go back into user mode and resume the user program
- Remember — Operating System is interrupt driven
- Some instructions are privileged. They can only be executed in kernel mode. User mode causes an illegal instruction trap. Some examples are:
  - I/O instructions
  - Memory management instructions
- Lack of mode leads to problems - MS-DOS, Amiga, early Macintoshes.

I/O Protection

- All I/O instructions are privileged instructions
  - User programs must request the operating system to perform the I/O operation, even if the program has exclusive access to the device (e.g. tape drive).

- Ensure that the user never gets control of the CPU in supervisor mode, or might as well not have modes.

- Security holes, malicious attacks, viruses, worms.

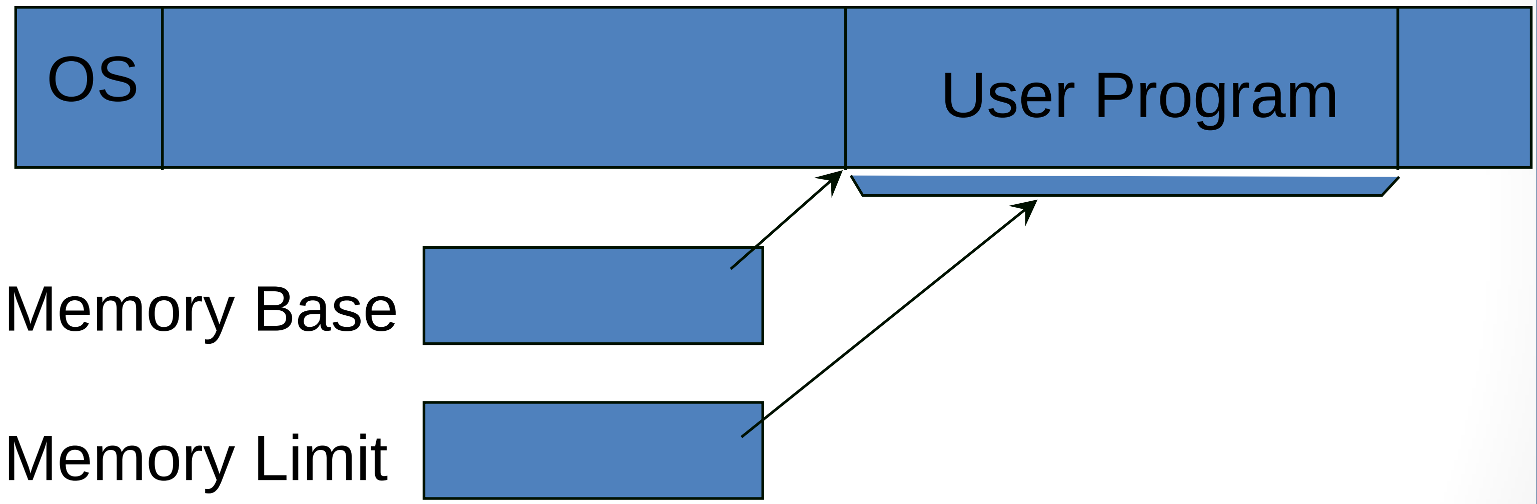
Memory Protection

- Limit users access to memory of his program only.
  ◊ For memory mapped I/O, this accomplishes I/O protection
  ◊ Protects the Interrupt Vector Table and the Operating system itself

- Early version (used on DEC PDP-11) is a memory base register and a memory limit register. Defines a region of memory that can be accessed. Access outside of this region causes an error.

- Enforced by hardware, changing memory access registers are privileged instructions

Memory Protection



CPU Protection

- Cooperative multitasking (early windows, macintosh). Programs yield the CPU, usually by calling a system routine.
- Poorly written programs can take over the machine (infinite loop, not yielding machine)
- Timers interrupt process, allow O/S to take away CPU.
- Calculate time at the same time
- Allows programs to be written without having to explicitly share the CPU.
- Timer interaction privileged instructions

Operating System Components

- Process Management
- Memory Management
- File Management
- I/O Management
- Secondary Management
- Networking
- Protection System
- Command-Interpreter

Process Management

- What is a process?
  ◊ more than program. A program is a file containing executing code.
  ◊ Two instances of the program are each separate processes.
  ◊ A process is a collection of resources: Memory, CPU time, open files, I/O, current program counter, etc.
- OS Activities
  ◊ Process creation and deletion
  ◊ Suspension and resumption
  ◊ Synchronization and Communication

Memory Management

- Memory is a continuous sequence of bytes (or words) each of which has its own address
- Volatile

- OS Activities
  ◊ Allocating, deallocating and keeping track of memory
  – don’t let one process free another processes memory.
  ◊ Deciding which process to load

File Management

- One of the most visible parts of an operating system
- File is a collection of related information
- May be structured (MacOS, IBM Mainframes, VMS) or unstructured (UNIX, DOS, Windows)
- OS Activities
  - ◊ File creation and deletion
  - ◊ Directory creation and deletion
  - ◊ Manipulating files, directories
  - ◊ Storage of files on secondary media
  - ◊ Backup and Restore

I/O Management

- Hide specifics of devices behind device drivers
- Typically contains:
  - ◊ buffer and caching management
  - ◊ generalized device driver interface (ioctl)
  - ◊ specific drivers for specific devices

Secondary Storage Management

- Most common device on a computer
- OS Handles

  ◊ storage allocation and free space management

  ◊ scheduling of requests for reading and writing disk blocks

- Can get complex

  ◊ logical vs physical volumes

  ◊ RAID (Redundant array of inexpensive disks)

Protection

- Some way to specify protection to the system
- authorized users (system operators)
- enforce restrictions to files, devices
  - ◊ Unix users/groups
  - ◊ Access Control Lists

Command Interpreter

- User Interface
- accept commands and execute them
- scripting languages
- Graphical Interface
- Sometimes built into the system
  - ◊ Windows/Macintosh
  - ◊ Old Basic Systems (Pet, Apple 2)
- Sometimes separate process
  - ◊ Unix shell
  - ◊ TSO Interpreter
  - ◊ MS-DOS COMMAND.COM

System Services

- All of the systems we have discussed have to be accessible to user and user programs
- Have to have mechanisms to start programs, request I/O, communicate with other processes and with other computers
- Error detection is required (hardware, user programs)

- Other services needed are:
  - ◊ Resource Allocation
  - ◊ Accounting (time, disk space, pages, etc.)
  - ◊ Protection

System Calls

- Interface between a running process and the operating system
  ◊ Generally implemented as an assembly language instruction (INT on INTEL, TRAP on PDP-11, ALine instruction on 68000)
  ◊ Sometimes a function call (Amiga, NCR32000)
  ◊ Programming languages provide libraries that wrap these instructions for easy access.

- Like most function calls, they take arguments and return a result, it’s just the mechanism of the call that is different.

System Calls

- The system call is identified in one of two ways
  ◊ One of the parameters (Dos)
  ◊ Different instruction (Macintosh)

- Three general ways are used to pass parameters
  ◊ Use the registers of the CPU (Amiga, MS-DOS)
  ◊ Store them in memory and pass the address of the memory in a register (Linux)
  ◊ Pass them on the stack

- Results are passed back in one of two ways:
  ◊ On the stack
  ◊ In a register

Types of System Calls

- Process Management
  - ◊ exit, abort, TSR,load, execute
  - ◊ manage child processes (attributes, wait, trace)
  - ◊ memory allocation, signals
- File System Management
  - ◊ create,open,close,read,write,etc.
- Device Management (e.g. serial speed)
  - ◊ request, release, read,write,attributes
- Information Maintenance
  - ◊ time, date, file and device attributes
- Communication
  - ◊ open,close,send,receive, status

Communication Model

- Interprocess Communication

  ◊ Command interpreter -> child process

  ◊ other processes need to communicate (Mac publish and subscribe)
- Message Model

  ◊ Operating system provides message passing facility.

  ◊ In some cases, works over network (sockets)
- Shared Memory Model

  ◊ both processes have access to the same memory

  ◊ small segments or all of processes memory

Systems Program

- System calls provide capability, but user must be able to invoke them.
- Users view of system defined by system programs, not system calls. No system call to copy a file, but there is a system program.
- Programs must be written that the user can invoke that make the system calls
  ◊ Command interpreter
  ◊ File System Management (copy, cp, mkdir, rmdir)
  ◊ Info (date, time, disk space, list directory)
  ◊ Editors
  ◊ compilers, project support
  ◊ communication (web browsers, ftp clients)

System Structure

- Simple Structure
  ◊ Limited by hardware
  ◊ MS-DOS - interfaces and functionality not well separated
  ◊ Unix is two separate parts
  – kernel system call interface (some modularity)
  – systems programs
- Layered (OS/2)
  ◊ Operating system divided into layers
  ◊ Each layer only uses lower layers (no sibling calls)
- Microkernel (Mach)
  ◊ Minimal kernel (no consensus on what minimal is)
  ◊ Move as much as possible into system programs

Virtualization

- Multiprocessing creates the illusion that there is more than one CPU
  ◊ What if the OS provided an interface that looked exactly like a computer system (virtual CPU, devices, etc.)
  ◊ IBM VM Operating system (each can run it's own operating system such as CMS or Linux)
  ◊ Requires more hardware support
  ◊ Available on PCs (vmware/xen/kvm)
  ◊ Limits sharing of resources
  ◊ Perfect for OS research and development
  ◊ Emulation (Sheepshaver/BOCHS)

Virtual Machines

- Portability
  - ◊ virtual machine provides same interface on all hardware
- Security
  - ◊ sandbox
- Like an OS
  - ◊ class loader (loads bytecode)
  - ◊ threads (sort of like virtual cpus)
- Performance
  - ◊ Just in time compilation
- Android
  - ◊ Linux Kernel
  - ◊ Dalvik virtual machine

Design and Implementation

- Mechanisms and Policies
  ◊ Mechanisms are how
  – Use timer to protect CPU
  ◊ Policy is what will be done
  – timer value (max time process can have CPU)
  ◊ more flexibility
- Implementation
  ◊ language
  ◊ efficiency
  ◊ portability (Linux runs on many architectures)
  ◊ monitoring capability
  ◊ Sysgen (customization of operating system)