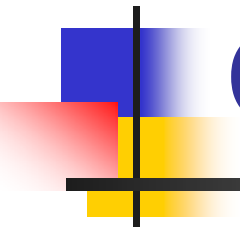
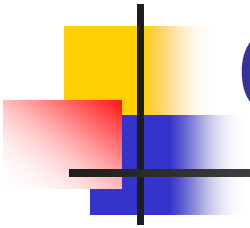


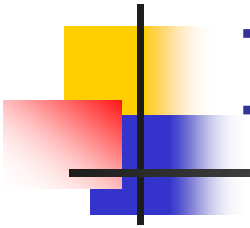
Chapter 1: Introduction

CMPS 105: Systems Programming
Prof. Scott Brandt
T Th 2-3:45
Soc Sci 2, Rm. 167



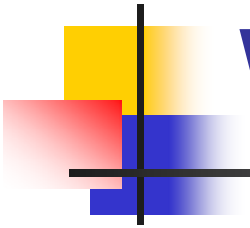
Class Outline

- Chapter 1: Introduction
- Chapter 2: Unix Standards and Implementations
- Chapter 3: File I/O
- Chapter 4: Files and Directories
- Chapter 5: Standard I/O Library
- Chapter 6: System Data Files and Information
- Chapter 7: The Environment of a Unix Process
- Chapter 8: Process Control
- Chapter 10: Signals
- Chapter 14: Interprocess Communication
- Efficient Programming



Introduction

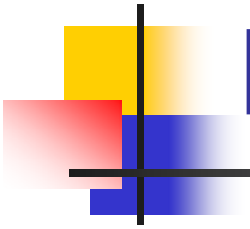
- Operating systems provide services for programs
  - Execute a program, open a file, read a file, allocate memory, get time of day, etc.
- Most programming languages provide very (too?) high-level abstractions for these services
  - It is almost impossible to write efficient programs at that level
- This class will focus on programming in the raw – right on top of the OS
- Goals: Efficient, powerful programs that leverage the power of the OS



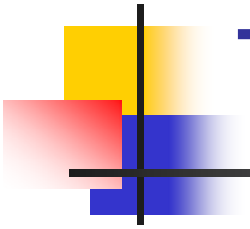
Why Unix?

- Unix is widely used and freely available
  - Linux, FreeBSD, System V, BSD4.4, MacOS, etc.
- Once you understand one system in detail, it is easy to learn others
  - At some level, Windows (the most widely used OS in the world) isn't all that different from Unix

Logging In to the Computer

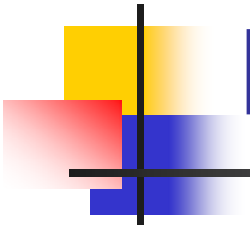


- Enter Username and Password
- Username identifies you to the computer
  - Why does it care who you are?
- Password proves you are who you say you are
  - Why does it need proof?
  - Is this adequate proof?



The Shell

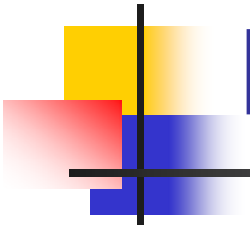
- A shell is a program that:
  - Accepts inputs from the user
  - Runs/manages programs for the user
  - Often supports limited programming
- Shells: sh, csh, ksh, bash, tcsh, zsh, …
- We will write a shell later in the quarter



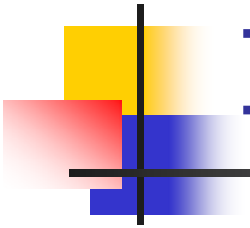
Files and Directories

- Files provide non-volatile data storage
  - What you write to a file stays there until you explicitly delete it
- Files have names
  - That's how you refer to them, find them, etc.
- Files are contained in directories (which also have names)
- Directories contain files and other directories
  - Directories form a hierarchy
  - There is a root directory called "/"
- Special files: "." and ".."

File and Directories (cont.)

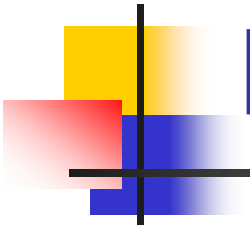


- Pathnames
  - Absolute pathname “/a/b/c”
  - Relative pathname “a/b/c” relative to the current directory
- Every process has a working directory
  - The current directory
- Home directory
  - The directory you start out in when you log in



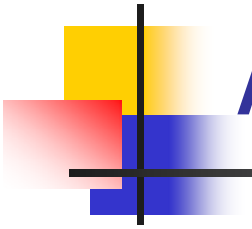
Input and Output

- File descriptors
  - Small integers the kernel uses to identify open files in a process
  - Actually an index into a table maintained by the kernel
- Standard Input, Output, and Error
  - Default file descriptors for scanf, printf, etc.
  - Can be redirected
- Unbuffered I/O
  - Default for open, read, write, lseek, and close



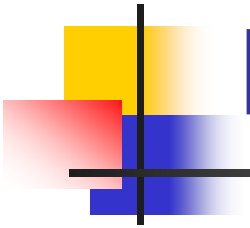
Programs and Processes

- Program
  - An executable file
- Process
  - A running program
- Process ID
  - An identifier for a running program
- Process control
  - Fork, exec, wait



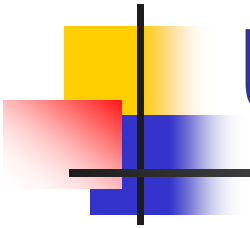
ANSI C Features

- Function prototypes
  - <unistd.h>
  - Probably in /usr/include/unistd.h
- Generic pointers (not important)
- Primitive System Data Types
  - End in _t (as in pid_t)
  - Defined in <sys/types.h>



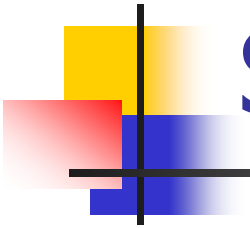
Error Handling

- Unix system functions return negative number to indicate an error (usually)
- errno contains addition information
- Defined in <errno.h>



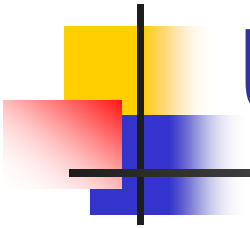
User Identification

- User ID
  - Unique identifier for each user that can use the computer
  - Why do we need this?
- Group ID
  - Unique identifier for the group the user is in
  - Useful for sharing information



Signals

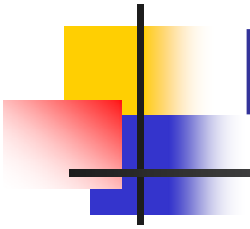
- Signals allow processes to communicate (but just barely)
- When a process receives a signal, it can
  - Ignore it
  - Let the default action occur
  - Handle it with a prespecified function (a signal handler)
- You must own a process to send it a signal



Unix Time Values

- Calendar time
  - Recorded as the number of seconds since January 1, 1970
  - Stored as time_t
  - May cause a second Y2K frenzy
- Process time
  - The amount of time used by a process
  - Clock time, user CPU time, system CPU time

System Calls and Library Functions



- System calls
  - Access points to call OS functions
  - Not direct function calls
  - Limited number of well-defined functions
  - Why?
- Library functions
  - A set of useful functions
  - May or may not invoke system calls

Library Functions and System Calls (cont.)