CS194-24
Advanced Operating Systems Structures and Implementation
Lecture 3

Hardware/Software Interface
OS Structure

January 29th, 2014
Prof. John Kubiatowicz
http://inst.eecs.berkeley.edu/~cs194-24

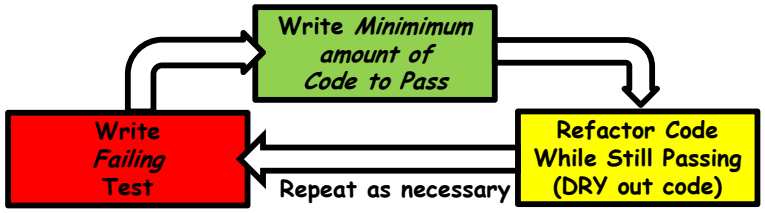
Goals for Today

- Finish discussion of TDD/BDD
- Operating Systems Resources
- API access to hardware resources
- OS Design

Interactive is important!
Ask Questions!

Note: Some slides and/or pictures in the following are adapted from slides ©2013

Review: Test-Driven Development (TDD)



- Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle:
  - First the developer writes an (initially failing) automated test case that defines a desired improvement or new function,
  - Then produces the minimum amount of code to pass that test, and
  - Finally refactors the new code to acceptable standards.
- Key thing – *Tests* come before *Code*

Review: A Ubiquitous Language for Analysis

- Need a framework for analyzing the process:
  - As a [X]
  - I want [Y]
  - so that [Z]
- Then, need a way of expressing the *acceptance criteria* in terms of *scenarios:*
  - Given some initial context (the givens),
  - When an event occurs,
  - Then ensure some outcomes
- Example in cucumber (called, say "valid_card_withdrawal.feature")

```
Feature: The Customer tries to withdraw cache using valid ATM card
     As a customer,
     I want to withdraw cache from an ATM
     so that I don't have to wait in line at the bank
  scenario: Successful Cache Withdrawal
     Given I have an ATM card that is owned by me
     When I request $40
     and my account has enough money
     Then I will receive $40
  scenario: Unsuccessful Cache Withdrawal
     Given I have an ATM card that is owned by me
     When I request $40
     And my account does not have enough money
     Then I will receive an error
```

Review: What do Step definitions look like?

- What do these steps translate into?

```
Given I have an ATM card that is owned by me
When I request $40
and my account has enough money
Then I will receive $40
```

- Answer: Regular expressions in a step file:

```
Given /^I have an ATM card that is owned by me$/ do
    # Set up  machine with card and valid PIN
      @my_account ||= Account.new
end

When /^I request \$(\d+)$/ do |amount|
      @my_request = amount
end

And /^my account has enough money$/
      @my_account.balance.should <= @my_request
end

Then /^I will receive \$(\d+)$/ do |amount|
      @my_account.request_money(@my_request).should = amount
end
```

- Steps interact with actual implementation
  - Reference code you "wish you had", not "code you already have"
- I put up a pointer to "Rubular" of the *Resources* page
  - It lets you enter regular expressions and experiment

Review: Amusing example: Verify Apple-II

- Start with Apple-II Emulator, then add BDD testing with Cucumber (Thanks to Armando Fox):
  https://github.com/armandofox/cucumber-virtualii

```
Feature: enter and run a short BASIC program

  As a beginning programmer in the late 1970's
  So that I can get excited about CS and
  become a professor someday
  I want to learn BASIC by entering and running
  simple programs

Background: The Apple II is booted and the
      BASIC interpreter is activated
   Given there is no current BASIC program

Scenario: enter and run Hello World
  When I enter the following program:
    | lines                  |
    | 10 HOME                |
    | 20 PRINT "HELLO WORLD!"|
  And I clear the screen
  And I type "RUN"
  Then I should see "HELLO WORLD!"

Scenario: enter and run a Fibonacci program
  When I enter the following program:
    | lines                                      |
    | 10 INPUT "COMPUTE FIBONACCI NUMBER "; F    |
    | 20 N1 = 1 : N2 = 1                         |
    | 30 FOR I = F TO 3 STEP -1                  |
    | 40 T = N2                                  |
    | 50 N2 = N2 + N1                            |
    | 60 N1 = T                                  |
    | 70 NEXT I                                  |
    | 80 PRINT "RESULT IS "; N2                  |
  And I type "RUN"
  Then I should see "COMPUTE FIBONACCI NUMBER"
  When I type "6"
  Then I should see "RESULT IS 8"
```

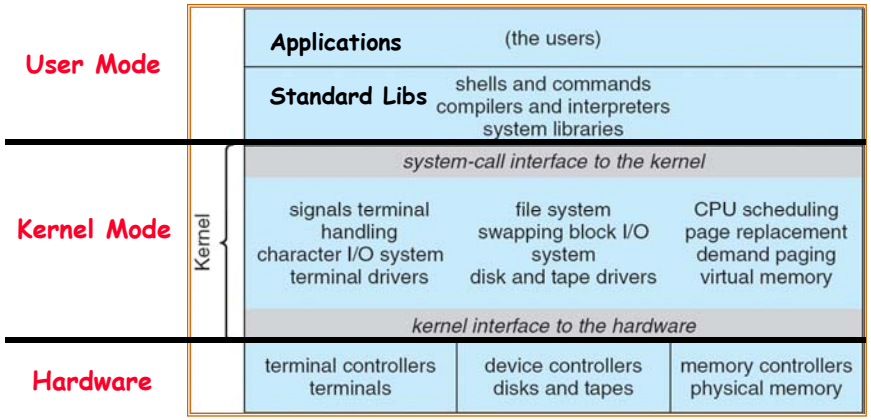

Verification Methodology

- Need for both User Stories (Behaviors) and Component Tests (Unit testing)
  - Behavioral Tests represent desired behavior from standpoint of stakeholders and involve whole code base
    - » Executable documentation!
    - » Slower, whole-system acceptance testing
    - » Run after every change
  - Unit testing frameworks (Like Rspec, CUnit, CPPSpec, etc) thoroughly test modules
    - » Fast execution
    - » Only run tests when change actual module
- Behavioral tests
  - High-level description independent of implementation
  - Test files named for behaviors being tested
    - » When failures happen, know where to start looking
  - Always in sync with code: tests run after every change
  - JBehave, Cucumber, etc
- Unit tests
  - Express individual details of implementation
  - Consider writing one or more unit test for every module
  - Can use CPPSpec, Cunit, etc.
  - Can be systematic, catch corner cases, etc

How Agile Methods Address Project Risks

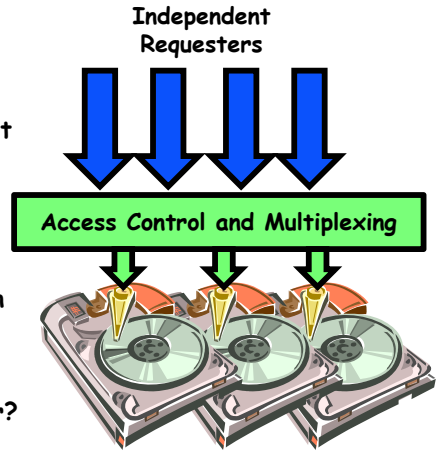
- No longer Delivering Late or Over Budget
  - Deliver system in tiny, one- or two-week iterations (or mini-projects)
  - Always have a working release
  - Know exactly how much it costs
- No Longer Delivering the Wrong Thing
  - Can demonstrate new features to stakeholders and make any tweaks or correct any misunderstandings while work fresh in developer's minds
- No Longer Unstable in Production
  - Deliver something on every iteration
  - Must get good at building and deploying the application
    - » Releasing to production or testing hardware just another build to just another environment
    - » Rely on software automation to manage this
  - Application servers automatically configured, database schemas automatically updated, code automatically built, assembled, and deployed
  - All types of tests automatically executed to ensure system working
- No Longer Costly to Maintain
  - With first iteration –team is effectively in maintenance mode!
  - Adding code to a working system, so they have to be very careful

Recall: UNIX System Structure



OS Resources – at the center of it all!

- What do modern OSs do?
  - Control access to resources!
- Control of Resources
  - Access/No Access/ Partial Access
    - » Check every access to see if it is allowed
  - Resource Multiplexing
    - » When multiple valid requests occur at same time – how to multiplex access?
    - » What fraction of resource can requester get?
  - Performance Isolation
    - » Can requests from one entity prevent requests from another?
- What or Who is a requester???
  - Process? User? Public Key?
  - Think of this as a "Principle"



What is a Resource?

- Processor, Memory, Cache
  - Multiplex through: scheduling, Virtual memory
  - Abstraction: Process, Thread
  - Need Kernel Level to Multiplex?
    - » Need to Sandbox somehow
    - » Kernel control of memory, prevent certain instructions
- Network
  - Multiplex through: Queues, Input Filters
  - Abstraction: Sockets API
  - Need Kernel Level to Multiplex?
    - » Not necessarily – New hardware has on-chip filters
    - » Setup Cost, but not necessarily a per-packet cost
    - » Is network really secure anyway? (Need Crypto!)
- Disk
  - Multiplex through: Buffer Cache
  - Abstraction: File System API
  - Need Kernel Level to Multiplex?
    - » Traditionally all access control through kernel
    - » What about assigning unlimited access to partitions?

More Complex Resources: Operating System Services

- System Services are really complex resources
  - File system (Uses Disk Drive)
    - » File API: Create, Read, Write, Delete
    - » Access Control: User, Group, World, Read/Write/Execute
  - Windows System (Uses Graphics Card)
    - » Windowing API: Write Text, Draw/Fill in figures
    - » Access Control: Per Window (User created)
  - Data Base (Uses Disk Drive or Memory or Network)
    - » DB API: SQL Queries and Transactions
    - » Access Control: Per user, Group, others
  - Lock Service (Memory)
    - » Lock API: Acquire (Read, Write), Release
    - » Access Control: By group/per user
- Access controlled through syscall interface (Kernel Level)
  - Funnel all access through trusted (verified) API
    - » Kernel controls access to API, verifies identity
    - » Service controls access to resources using identity
  - Service decides multiplexing/isolation policies
    - » Often based on first-come-first-serve!

The Protection vs Management Split

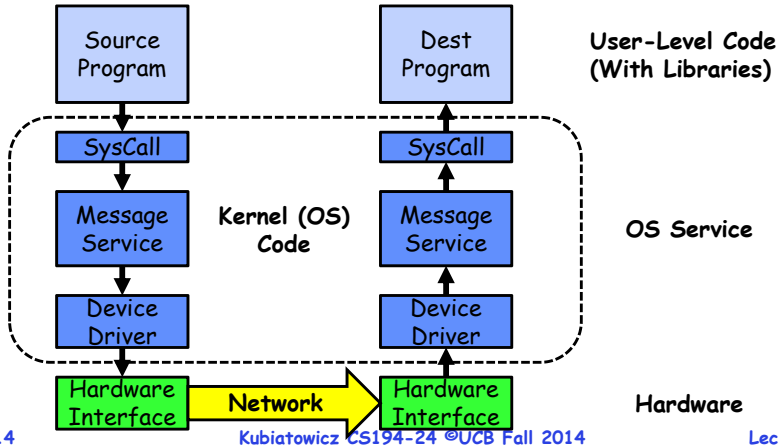
- Kernels Mix Protection, Performance Isolation, and Management
  - Protection: Should a principle have access to a given resource?
    - » Yes or No?
    - » Based on local password file? Thumbprint? Cryptographic Key?
  - Performance Isolation
    - » How Much of Bandwidth-Limited resources should the principle have access to?
    - » Examples:
      - 50% CPU
      - As much Network as Desired
      - Fraction of Paging Disk for Virtual memory?
  - Management: How should the principle use this resource?
    - » Scheduling, Policies
    - » Examples:
      - Use my CPU resources to meet realtime deadlines vs highest throughput scheduling
      - Keep certain pages in memory
- Problem with putting all three of these together is that APIs limited, complex, or insufficient…

Administrivia

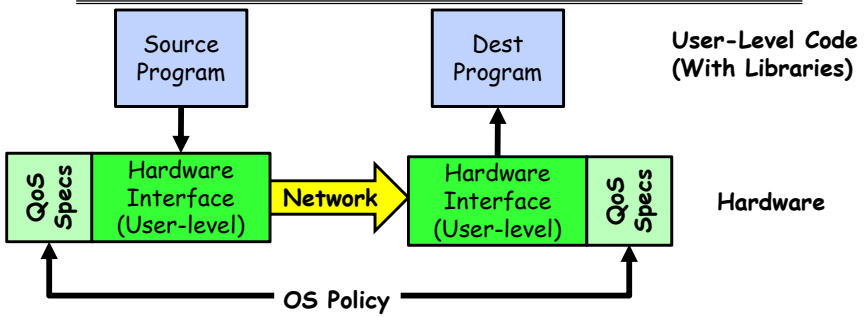
- Putting up useful resources off the *Resources* page
  - Pointers to Ruby, Git
  - Will put up pointers to C, other languages
  - Will continue to add resources over time
- Issues with Redmine site: Sorry about that
  - Seems like a plugin I used has some issues
    - » Put in temporary fix
    - » Key management issues should now fix themselves every 15 minutes (at max) if they occur
  - Also, possible that will get a 500 error first time you try to look at Repository link
    - » This is because of the size of the repository!
- Group Signup will be due by Monday
  - Remember – groups are 4 to 5 members
  - Signup link will be operational soon (don't try until Friday)

Traditional Approach to Handling Resource

- Example: Send message from one processor to another
  - Check Permissions, Format Message
  - Enforce forward progress, Handle interrupts
  - Prevent Denial Of Service (DOS) and/or Deadlock
- Traditional Approach: Use a system call+OS Service



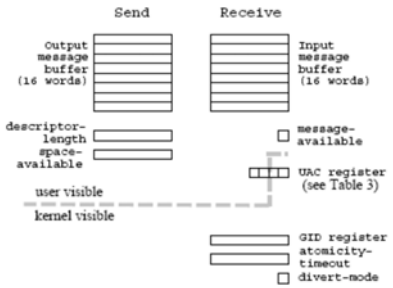
Alternative: Mechanisms (or even Policy!?) in HW

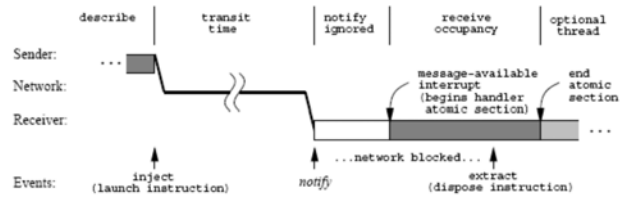


- Permit *user-level* code to send messages
  - Have hardware check permission and/or rate
  - Have hardware enforce format/consistency
  - Have hardware guarantee forward progress
  - Have Hardware deliver messages/interrupts to usercode
- OS sets registers to control behavior based on policy

Push Access Control to Hardware: User-Level Alewife Messaging

- Send message
  - write words to special network interface registers
  - Execute atomic launch instruction
- Receive
  - Generate interrupt/launch user-leve thread context
  - Examine message by reading from special network interface registers
  - Execute dispose message
  - Exit atomic section



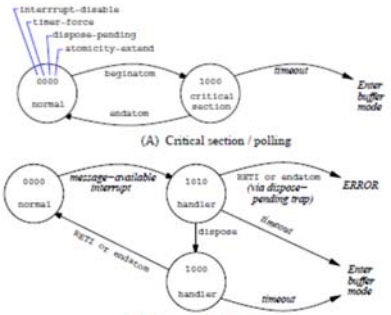


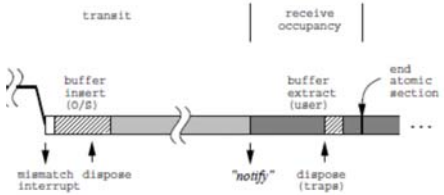
Sharing of Network Interface

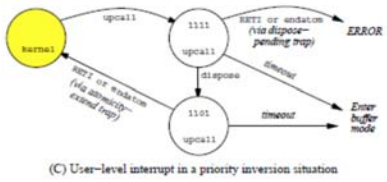
- What if user in middle of constructing message and must context switch???
  - Need *Atomic Send operation!*
    - » Message either completely in network or not at all
    - » Can save/restore user's work if necessary (think about single set of network interface registers
  - J-Machine mistake: after start sending message must let sender finish
    - » Flits start entering network with first *SEND* instruction
    - » Only a SENDE instruction constructs tail of message
- Receive Atomicity
  - If want to allow user-level interrupts or polling, must give user control over network reception
    - » Closer user is to network, easier it is for him/her to screw it up: Refuse to empty network, etc
    - » However, must allow *atomicity:* way for good user to select when their message handlers get interrupted
  - Polling: ultimate receive atomicity – never interrupted
    - » Fine as long as user keeps absorbing messages

Alewife User-level event mechanism

- Disable during polling:
  - Allowed as long as user code properly removing messages
- Disable as atomicity for user-level interrupt
  - Allowed as long as user removes message quickly
- Emulation of hardware delivery in software:







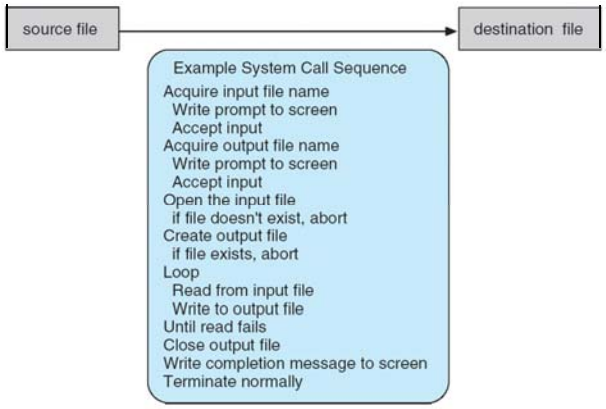
System Calls: Details

- Challenge: Interaction Despite Isolation
  - How to isolate processes and their resources…
    - » While still permitting them to request help from the kernel
    - » Letting them interact with resources while maintaining usage policies such as security, QoS, etc
  - Letting processes interact with one another in a controlled way
    - » Through messages, shared memory, etc
- Enter the System Call interface
  - Layer between the hardware and user-space processes
  - Programming interface to the services provided by the OS
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than directly
  - Get at system calls by linking with libraries in glibc

- 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)

Example of System Call usage

- System call sequence to copy the contents of one file to another file:



- Many crossings of the User/Kernel boundary!
  - The cost of traversing this boundary can be high

Example: Use strace to trace syscalls

- prompt% strace wc production.log

```
execve("/usr/bin/wc", ["wc", "production.log"], [/* 52 vars */]) = 0
brk(0)                                  = 0x1987000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8f7000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=137151, ...}) = 0
mmap(NULL, 137151, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff24b8d5000
close(3)                                = 0
open("/lib64/libc.so.6", O_RDONLY)      = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360\355\241,0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1922112, ...}) = 0
mmap(0x302ca00000, 3745960, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x302ca00000
mprotect(0x302cb89000, 2097152, PROT_NONE) = 0
mmap(0x302cd89000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x189000) = 0x302cd89000
mmap(0x302cd8e000, 18600, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x302cd8e000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8d4000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8d3000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8d2000
arch_prctl(ARCH_SET_FS, 0x7ff24b8d3700) = 0
mprotect(0x302cd89000, 16384, PROT_READ) = 0
mprotect(0x302c81f000, 4096, PROT_READ) = 0
munmap(0x7ff24b8d5000, 137151)          = 0
brk(0)                                  = 0x1987000
brk(0x19a8000)                          = 0x19a8000
open("/usr/lib/locale/locale-archive", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=99158576, ...}) = 0
mmap(NULL, 99158576, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff245a41000
close(3)                                = 0
stat("production.log", {st_mode=S_IFREG|0644, st_size=526550, ...}) = 0
open("production.log", O_RDONLY)        = 3
read(3, "## Logfile created on Fri Dec 28 "..., 16384) = 16384
open("/usr/lib64/gconv/gconv-modules.cache", O_RDONLY) = 4
fstat(4, {st_mode=S_IFREG|0644, st_size=26060, ...}) = 0
mmap(NULL, 26060, PROT_READ, MAP_SHARED, 4, 0) = 0x7ff24b8f0000
close(4)                                = 0
read(3, "m: cannot remove `/tmp/fixrepo/g"..., 16384) = 16384
read(3, "a36de93203e0b4972c1a3c81904e': P"..., 16384) = 16384
read(3, "xrepo/git-tess/gitolite-admin/.g"..., 16384) = 16384
        Many repetitions of these reads
read(3, "ixrepo/git-tess/gitolite-admin\n "..., 16384) = 16384
read(3, "ite/readmine/vendor/plugins/redmi"..., 16384) = 16384
read(3, "ited with positive recursionChec"..., 16384) = 16384
read(3, "ting changes to gitolite-admin r"..., 16384) = 2262
read(3, "", 16384)                      = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff24b8ef000
write(1, "  4704  28993 526550 production."..., 36  4704  28993 526550 production.log) = 36
close(3)                                = 0
close(1)                                = 0
munmap(0x7ff24b8ef000, 4096)            = 0
close(2)                                = 0
exit_group(0)
```


Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value | function name | parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

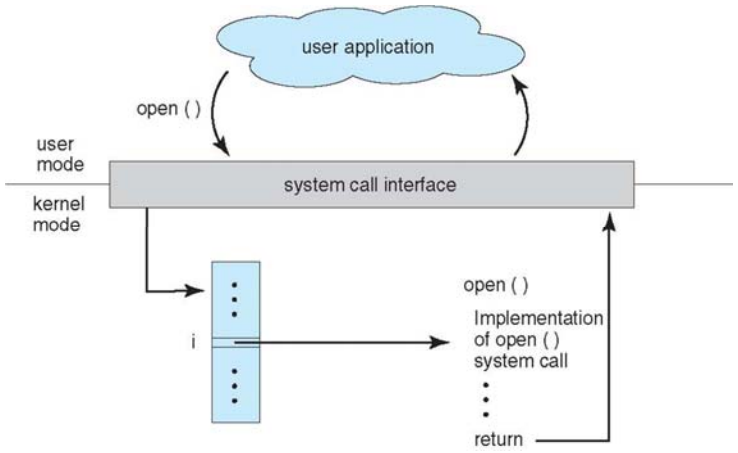
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
  - The fact that the call is by "number", is essential for security reasons!
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
  - Return value: often a long (integer)
    - » Return of zero is usually a sign of success, but not always
    - » Return of -1 is almost always reflects an error
  - On error – return code placed into global "errno" variable
    - » Can translate into human-readable errors with the "perror()" call
- 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)

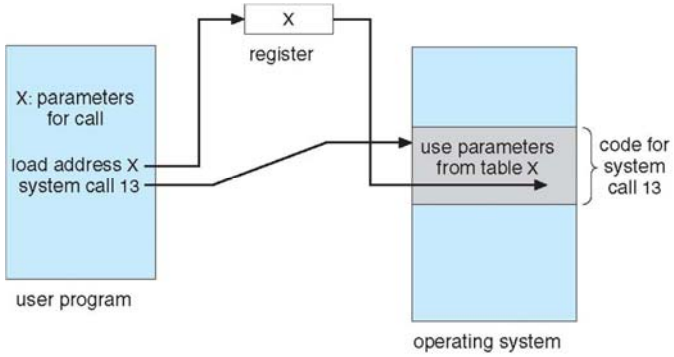
API – System Call – OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- 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

Parameter Passing via Table



- Kernel must always verify parameters passed to it by the user
  - Are parameters in a reasonable range?
  - Are memory addresses actually owned by the calling user (rather than bogus addresses)

Types of System Calls

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - 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

Types of System Calls (Con't)

- 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

Types of System Calls (Cont.)

- 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

POSIX standard

- Portable Operating System Interface for UNIX
  - » An attempt to standardize a "UNIXy" interface
- Conformance: IEEE POSIX 1003.1 and ISO/IEC 9945
  - Latest version from 2008
  - Originally one document consisting of a core programming inteface – now 19 separate docs
  - Many OSes provide "partial conformance" (including Linux)
- What does POSIX define?
  - POSIX.1: Core Services
    - » Process Creation and Control
    - » Signals
    - » Floating Point Exceptions, Segmentation/memory violations, illegal instructions, Bus Erors
    - » Timers
    - » File and Directory Operations
    - » Pipes
    - » C Library (Standard C)
    - » I/O Port Interface and Control
    - » Process Triggers
  - POSIX.1b: Realtime Extensions
  - POSIX.2: Shell and Utilities

POSIX (cont)

- Process Primitives:
  - fork, execl, execlp, execv, execve, execvp, wit, waitpid
  - _exit, kill, sigxxx, alarm, pause, sleep….
- Example file access primitives:
  - opendir, readdir, rewinddir, closedir, chdir, getcwd, open, creat, umask, link, mkdir, unlink, rmdir, rename, stat, fstat, access, fchmod, chown, utime, ftruncate,pathconf,fpathconf
- I/O primitives:
  - pipe, dup, dup2, close, read, write, fcntl, lseek, fsync
- C-Language primitives:
  - abort, exit, fclose, fdopen, fflush, fgetc, fgets, fileno, fopen, fprintf, fputc, fputs, fread, freopen, fscanf, fseek, ftell, fwrite, getc, getchar, gets, perror, printf, putc, putchar, puts, remove, rewind, scanf, setlocale, siglongjmp, sigsetjmp, tmpfile, tmpnam, tzset
- Synchronization:
  - sem_init, sem_destroy, sem_wait, sem_trywait, sem_post, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock
- Memory Management
  - mmap, mprotect, msync, munmap
- How to get information on a system call?
  - Type "man callname", i.e. "man open"
  - System calls are in section "2" of the man pages

Portability

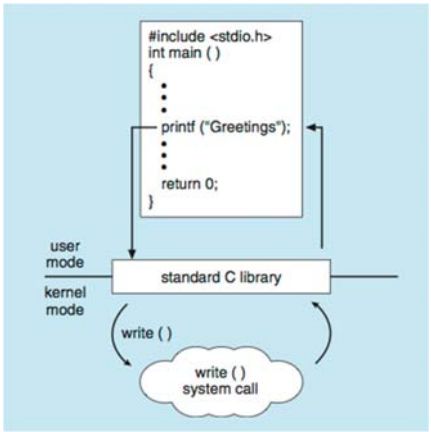
- POSIX does provide some portability
  - But is still pretty high level
  - Does not specify file systems, network interfaces, power management, other important things
  - Many variations in compilers, user programs, libraries, other build environment aspects
- UNIX Portability:
  - C-preprocessor conditional compilation
  - Conditional and multi-target Makefile Rules
  - GNU configure scripts to generate Makefiles
  - Shell environment variables (LD_LIBRARY_PATH, LD_PRELOAD, others)

Examples of Windows and Unix 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()

Standard C Library Example

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



Operating Systems Structure (What is the organizational Principle?)

- Simple
  - Only one or two levels of code
- Layered
  - Lower levels independent of upper levels
- Microkernel
  - OS built from many user-level processes
- Modular
  - Core kernel with Dynamically loadable modules
- ExoKernel
- Cell-based OS (Space-Time Partitioning)

Implementation Issues (How is the OS implemented?)

- Policy vs. Mechanism
  - Policy: What do you want to do?
  - Mechanism: How are you going to do it?
  - Should be separated, since both change
- Algorithms used
  - Linear, Tree-based, Log Structured, etc…
- Event models used
  - threads vs event loops
- Backward compatability issues
  - Very important for Windows 2000/XP
- System generation/configuration
  - How to make generic OS fit on specific hardware

Conclusion

- Test-Driven Development (TDD)
  - Write tests first, then write code, then refactor
- Behavior-Driven Development (BDD)
  - Executable Behavior specifications
  - Cucumber for Integration Behaviors, Unit tests for implementation.
- Resource Control: In HW or SW!
  - Access/No Access/Partial Access
  - Resource Multiplexing
  - Performance Isolation
- System-Call interface
  - This is the I/O for the process "virtual machine"
  - Accomplished with special trap instructions which vector off a table of system calls
  - Usually programmers use the standard API provided by the C library rather than direct system-call interface
- POSIX interface
  - An attempt to standardize "unixy" Oses
- OS Structure: Many approaches
  - But it is all about control of Resource!