CSE 430  Spring 2011
Operating Systems

Georgios Varsamopoulos

Arizona State University

Lecture 2: Chapter 2
Technical writing
Academic/technical writing guides

• Basic writing
  – Strunk & White, “The Elements of Style”

• Academic writing
  – Harvey, “The nuts and bolts of college writing”

• Writing style guides
  – The Chicago Manual of Style
Some writing advice

- Organize your thoughts before you write. It is important that the reader can follow your writing. Read what you wrote to see if it would make sense to everybody else.
- Organize your writing into points or arguments. Stick to making one point or one argument at a time. Then move on to the next.
- Be clear and precise. Write what you want to say and nothing more. Get to the point.
- Be brief. Do not use verbose language.
- Do not use clichés. Do not write in a pompous manner.
- Never present a speculation as a fact. Always confirm and verify before you use it as a fact.
- Do not use adjectives of impression, e.g., nice, fantastic, amazing, awesome, exceptional, etc. Exception: use cool only when the object is actually of low temperature.
- Have your document proof-read by someone else.
- Use in-line references for facts that you directly use from other sources. Uncited end-of-document references are “further reading” suggestions, not references per se.
Today's lecture
Lecture objectives

- Sections
  - 2.1 – 2.8
  - User interface (shells/command interpreters)
  - System Calls
  - System Programs

- Learning objectives and concepts
  - Shells, macrokernels vs microkernels
  - First acquaintance with POSIX
  - Brief history of operating systems
  - Virtualization
Structure of a technical report

- Summary
- Introduction
- Problem Statement / Statement of Objective
- Short description of approach/plan
- Detailed description of your technical work
- Evaluation of your work
A brief history of operating system features
1950s: Batch Job Systems

- Input was done using punch-cards, output was done by printing

- First need
  - To queue-up several jobs and collect results the next day
    - Concept of batch job systems
  - **Monitor**: a program that starts a new job when another finishes.

1960s: SPOOLing

- CPU was doing no useful job while the deck of cards was being read

- Magnetic tape I/O was faster than cards and printing
  - Use smaller computers for card reading into tape and printing from tape
  - Use main computer for computation and tape IO
  - SPOOL
    - Simultaneous Peripheral Operation On Line
1960s: Multiprogramming and compatibility

- More memory started being available
- Load multiple jobs on memory
  - When one is doing I/O, let the other run
  - **Multi-programming**
    - **Foreground** and **background**
  - **Virtual memory**
  - **Basic protection** schemes
  - **System calls**
- IBM OS/360: first O/S to provide
  - common API/ABI (Application Binary Interface), ie. cross-compatibility and portability.
1970s: timesharing and filesystems

- Operating systems more like how we know them
- Technology allowed for use of dummy terminals
  - Each user had direct access to computer
    - Multi-user systems
  - Use of time-sharing (or time-slicing)
    - Concept of apparent exclusiveness to each user
- Concept of directory on disk
  - First hierarchical file systems
  - Early file systems only supported a flat structure
- Also, the first virtual machines
Virtual machines

processes

kernel

hardware

monitor or supervisor (hypervisor)

programming interface

guest OS

kernel

VM1

virtual-machine implementation

hardware

VM2

VM3

processes
Full virtualization vs paravirtualization

- **Full virtualization**
  - Virtualizes ISA and available hardware
  - E.g. Can run a PowerPC virtual machine over an x86 real machine

- **Paravirtualization**
  - Does not virtualize ISA
  - “Virtualizes” access to available hardware
    * Each guest has apparent exclusive access to devices

- **Why virtualization?**
  - Running older OS and other software on newer machines
  - Consolidation of hardware
Ok, now back to the present
## Operating system functions

<table>
<thead>
<tr>
<th>user and other system programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
</tr>
<tr>
<td>batch</td>
</tr>
<tr>
<td>command line</td>
</tr>
<tr>
<td>user interfaces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>system calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>program execution</td>
</tr>
<tr>
<td>I/O operations</td>
</tr>
<tr>
<td>file systems</td>
</tr>
<tr>
<td>communication</td>
</tr>
<tr>
<td>resource allocation</td>
</tr>
<tr>
<td>accounting</td>
</tr>
<tr>
<td>error detection</td>
</tr>
<tr>
<td>protection and security</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>services</td>
</tr>
</tbody>
</table>

| hardware                      |

- User interface: shell (text mode), GUI
- Program execution and process manipulation
- I/O operations (data transfer)
- File system operations
- Communication (inter-process and networking)
- Error (fault) detection
- Resource allocation
- Accounting
- Protection and security
Shell (command interpreter)

- There are various shells
  - Windows: “dos prompt”
  - UNIX: bourne shell (sh), C shell (csh), T c-shell (tcsh), Bourne-again shell (bash), Korn shell (ksh) etc.
- Commands have parameters and options/switches
System Calls

• Every operating system defines a set of system calls
  – The API to the kernel

• Depending on the programming language, the syntax may change
  – Each programming language that should provide an “interface” to the system calls
Example of system call usage

Example System Call Sequence
Acquire input file name
Write prompt to screen
Accept input
Acquire output file name
Write prompt to screen
Accept input
Open the input file
if file doesn't exist, abort
Create output file
if file exists, abort
Loop
Read from input file
Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
Example POSIX system calls

- **POSIX**
  - Portable Operating System Interface (for UNIX)
  - Defines the attributes of a process, threads, and a set of system calls and their functionality

- Examples
  - getpid, getppid
  - fork, exec, wait
  - read, write
Example with strace
shared libraries and linking

```c
#include <stdio.h>
int main ()
{
    ...
    printf ("Greetings");
    ...
    return 0;
}
```

printf(...) {
    ...
    Write(...) 
    ...
}
fopen(...) {
}

System programs (system utilities)

- Status information
  - active users, active processes, system utilization etc
- Process control
  - End, abort, get/set process attributes
- Configuration
  - e.g. registry access
- File management
  - Create, delete, copy, move files
- File system management
  - defrag, error checking
- Programming
  - C compiler, debugger etc
Design and Implementation
Design Goals

• Not all operating systems have some common design aspects
  – Widely vary in design and internal structure
  – Batch, time-shared, single-user or multiuser, centralized or distributed, real-time, general-purpose or specific-purpose.

• User goals
  – convenient to use, easy to learn, reliable, safe, and fast

• System goals
  – easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
Policy vs. Mechanism

- **Example:**
  - The system call API is a policy. It determines
    - A set of rules and conventions
    - What to be expected from each syscall
  - Kernel is the implementation of the policy
    - Exactly how API is implemented (ie. code)

- **In general**
  - **Policy:**
    - What rules to be enforced
  - **Mechanism**
    - How to enforce the rules

- **Example:** sort file
  - Policy: output file lines in alphabetic order
  - Mechanism: use bubblesort, quicksort etc
O/S Structure

Simple implementation (MS-DOS)

- application program
- resident system program
- MS-DOS device drivers
- ROM BIOS device drivers

Monolithic implementation (early UNIX)

- shells and commands
- compilers and interpreters
- system libraries
- system-call interface to the kernel
  - signals terminal handling
  - character I/O system terminal drivers
  - file system swapping block I/O system
  - disk and tape drivers
- CPU scheduling
- page replacement
- demand paging
- virtual memory
- kernel interface to the hardware
  - terminal controllers terminals
  - device controllers disks and tapes
  - memory controllers physical memory
Windows NT family architecture

microkernel
Macros, micro-kernels, nano-kernels

- **Macrokernel**
  - Lots of functionalities in one object (monolithic kernel)

- **Micro-kernel**
  - Removal of non-core functionality, which is implemented as either user-space or kernel-space processes
  - Modular

- **Nano-kernel**
  - Even smaller microkernel

- **Why and how to choose**
  - Hardware capabilities, design objectives
Modular O/S

- Usually implemented using a microkernel
- Example typical modules
  - Filesystem implementation
    - Why load NTFS support if no NTFS file system present?
  - Device support
- Example
  - `lsmod` in linux
Next class

- Processes (Chapter 3)