Operating Systems Overview

Chapter 2
An operating System:
The interface between hardware and the user

From the user’s perspective: OS is a program that controls the execution of application programs.
Hardware/Software Interfaces

- Instruction set architecture (ISA):
  - defines the instruction set available on a given piece of computer hardware
Hardware/Software Interfaces

- Application binary interface (ABI):
  - ABI defines the structures and methods used to access external, already compiled libraries/code at the level of machine code.
  - OS provides an ABI for system calls that is independent of the underlying ISA.
Hardware/Software Interfaces

- Application programming interface (API):
  - programming interfaces provided for high-level languages (C++, Java, Ruby, etc...) that abstract away the low-level details
  - productive programmers spend their time at this level if they have any choice
High-level OS Goals

- Convenience
  - provide the user easy access to the functionality offered by the hardware

- Efficiency
  - ensure that programs make efficient use of the hardware

- Abstraction
  - make memory management, multiprogramming, etc. transparent to the user
  - enable programmers to interact with the hardware via high-level APIs
  - enable users to interact with the hardware via high-level applications
Facilitate program execution

Several steps need to be performed to begin executing a program:

- loading instructions and data into main memory
- initializing I/O devices and files, as well as any other resources the program needs
- saving and restoring program state on interrupts and process changes
OS Responsibilities (2)

- Provide access to I/O devices

- Each I/O device requires its own set of control signals
  - the OS abstracts away these details
  - it provides a single consistent interface so that programmers can access devices using simple reads and writes
OS Responsibilities (3)

- Provide and control access to files

- To serve files, the OS must understand
  - the nature of the I/O devices hosting the file system
  - the format of the data contained in the files

- It should also validate that a given user has access
  - e.g.: rw-r--r--

<table>
<thead>
<tr>
<th>Three permission triads</th>
</tr>
</thead>
<tbody>
<tr>
<td>first triad</td>
</tr>
<tr>
<td>second triad</td>
</tr>
<tr>
<td>third triad</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Each triad</th>
</tr>
</thead>
<tbody>
<tr>
<td>first character</td>
</tr>
<tr>
<td>second character</td>
</tr>
<tr>
<td>third character</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
OS Responsibilities (4)

- **System access**

- The OS administers the entire computer
  - the OS controls access to the system as a whole and to specific system resources (System resource allocation)
  - the OS has privileged access to certain instructions, allowing it to administer the hardware (Data and user management)
A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: atikmdag.sys

SYSTEM_THREAD_EXCEPTION_NOT_HANDLED

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use safe mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:

*** STOP: 0x00000007e (0xffffffffc0000005, 0xffffffff8005fc5574, 0xffffffff8008234108, 0xffffffff8008233960)

*** atikmdag.sys - Address 0xffffffff8005fc5574 base at 0xffffffff88005c0f000 DateStamp 0x4f3b1dc8
OS Responsibilities (5)

- **Error detection and response**

  - A variety of errors can occur while a computer system is running
    - internal hardware errors (memory errors)
    - external hardware errors (device failures)
    - software errors (division by zero or accessing a forbidden memory location)

  - The OS must handle the error with the least impact on running applications by
    - retry the operation
    - end the program that caused the error
    - simply report the error to the application
OS Responsibilities (6)

- **Accounting**

- Operating systems collect many usage and performance statistics

- This data is useful for
  - effectively managing the system
  - anticipating future needs
  - billing purposes (e.g. web hosts)
OS Responsibilities (6)

➥ Accounting

➥ Examples:

➥ system load and resource utilization (CPU, memory)

➥ process execution statistics:

➥ current state (ready, blocked, suspended, etc.)

➥ amount of time running so far

➥ keeping track of allocated resources (file descriptors, memory)

➥ various types of logging (errors, user logins, access attempts, etc.)

➥ tracking of allocated / unallocated memory blocks

➥ maintaining file system data structures
Quick Recall:

How does OS gain the control of CPU (or the computer) and do the accounting?
OS Responsibilities (7)

- **Facilitate program development**

- **The OS is a platform for running programs**
  - The OS provides a variety of utilities that assist the programmer in creating applications for the OS
  - While not strictly part of the core of the OS, they are supplied with the OS and can be viewed as a part of the OS.

- **Examples**
  - Compilers, debuggers, editors, GUI tools
Think-Pair-Share
Q.02
EAT !
Nature of an OS

- Operating systems are just programs!
  - the OS is software, just like the user applications it runs
  - it is a program (or suite of programs) executed by the processor

- Its duty is to efficiently facilitate convenient execution of user programs
  - to do this, it must relinquish control of the processor and cease its own execution
  - for a single CPU, when a user program is running, the OS is not being executed
Nature of an OS

- OS must stay in control of the hardware
  - the OS depends on frequent processor interrupts to periodically regain control
  - it utilizes a privileged set of instructions offered by the hardware
  - user programs are limited to a less-privileged set of instructions

- It is a “resource manager”
  - it must allocate system resources in order to accomplish “useful” work
Nature of an OS
The Kernel

- The OS Kernel
  - the *core functionality* of the operating system
  - the most commonly used functionality
  - but it’s a subjective term...

- The kernel must stay resident in memory
  - it deals with the critical details of operating the computer
    - e.g., resource management, I/O, process scheduling, accounting, etc.

- Other parts of the OS are loaded into memory as needed
Evolution of Operating Systems

- OS evolves for several reasons
  - Hardware upgrades and new types of hardware
  - New services desired by users
  - Fixes ...

- Source lines of code (SLOC) in major OSes

<table>
<thead>
<tr>
<th>Year</th>
<th>Operating System</th>
<th>SLOC (Million)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993</td>
<td>Windows NT 3.1</td>
<td>4–5[2]</td>
</tr>
<tr>
<td>1994</td>
<td>Windows NT 3.5</td>
<td>7–8[2]</td>
</tr>
<tr>
<td>1996</td>
<td>Windows NT 4.0</td>
<td>11–12[2]</td>
</tr>
<tr>
<td>2000</td>
<td>Windows 2000</td>
<td>more than 29[2]</td>
</tr>
<tr>
<td>2001</td>
<td>Windows XP</td>
<td>45[3][4]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Operating System</th>
<th>SLOC (Million)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>Debian 2.2</td>
<td>55–59[6][7]</td>
</tr>
<tr>
<td>2002</td>
<td>Debian 3.0</td>
<td>104[7]</td>
</tr>
<tr>
<td>2005</td>
<td>Debian 3.1</td>
<td>215[7]</td>
</tr>
<tr>
<td>2007</td>
<td>Debian 4.0</td>
<td>283[7]</td>
</tr>
<tr>
<td>2009</td>
<td>Debian 5.0</td>
<td>324[7]</td>
</tr>
<tr>
<td>2012</td>
<td>Debian 7.0</td>
<td>419[8]</td>
</tr>
<tr>
<td>2009</td>
<td>OpenSolaris</td>
<td>9.7</td>
</tr>
<tr>
<td>2009</td>
<td>FreeBSD</td>
<td>8.8</td>
</tr>
<tr>
<td>2005</td>
<td>Mac OS X 10.4</td>
<td>86[9][n 1]</td>
</tr>
<tr>
<td>2001</td>
<td>Linux kernel 2.4.2</td>
<td>2.4[5]</td>
</tr>
<tr>
<td>2003</td>
<td>Linux kernel 2.6.0</td>
<td>5.2</td>
</tr>
<tr>
<td>2009</td>
<td>Linux kernel 2.6.29</td>
<td>11.0</td>
</tr>
<tr>
<td>2009</td>
<td>Linux kernel 2.6.32</td>
<td>12.6[10]</td>
</tr>
<tr>
<td>2010</td>
<td>Linux kernel 2.6.35</td>
<td>13.5[11]</td>
</tr>
<tr>
<td>2012</td>
<td>Linux kernel 3.6</td>
<td>15.9[12]</td>
</tr>
<tr>
<td>2015-06-30</td>
<td>Linux kernel pre-4.2</td>
<td>20.2[13]</td>
</tr>
</tbody>
</table>
Evolution of Operating Systems

- Serial Processing
  - no operating system
  - machines run from a console with display lights, toggle switches, input device, and printer
  - users must schedule time to use the system in advance
  - setup included loading the compiler and source program, saving compiled program, loading and linking
Evolution of Operating Systems

- **Batch systems**
  - implemented a program called a "monitor"
  - software that controls the sequence of events
  - batch jobs together and run them with no downtime

An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.
Evolution of Operating Systems

- The monitor is the first real "operating system"
Soon incorporated several advances

- memory protection: prevent user code from screwing with monitor code
- timers: prevent a single job from monopolizing the system
- privileged instructions: certain machine instructions can only be executed by the OS
- interrupts: provides better flexibility in relinquishing control to and regaining control from user programs
Evolution of Operating Systems

- Systems now commonly implement different “modes” of access
  - User mode
    - certain areas of memory are restricted and inaccessible
    - certain instructions are not usable
    - user programs doing “useful work” run in this mode
  - Kernel mode
    - unrestricted access to memory and all instructions
    - the OS runs in this mode
  - User programs must be prevented from gaining kernel mode
    - if they do, the system is compromised
Uniprogramming

- IO is slow, CPU is fast
- A program containing even a very small number of I/O ops, will spend most of its time waiting for them
- Hence: uniprogramming has poor CPU usage

<table>
<thead>
<tr>
<th>Program A</th>
<th>Run</th>
<th>Wait</th>
<th>Run</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read one record from file</td>
<td>15 μs</td>
</tr>
<tr>
<td>Execute 100 instructions</td>
<td>1 μs</td>
</tr>
<tr>
<td>Write one record to file</td>
<td>15 μs</td>
</tr>
<tr>
<td>Total</td>
<td>31 μs</td>
</tr>
</tbody>
</table>

Percent CPU Utilization = \( \frac{1}{31} = 0.032 = 3.2\% \)
Multiprogramming

- If memory can hold several programs, then CPU can switch to another one whenever a program is awaiting for an I/O to complete
- This is multitasking (multiprogramming)
Multiprogramming

- If memory can hold several programs, then CPU can switch to another one whenever a program is awaiting for an I/O to complete.
Requirements for Multiprogramming

- **Hardware support:**
  - Interrupts
    - in order to execute instructions while I/O device is busy
  - Memory management
    - several ready-to-run jobs must be kept in memory
  - Memory protection (data and programs)

- **Software support from the OS:**
  - Scheduling (which program is to be run next)
  - To manage resource contention
Effects of Multiprogramming

- We assume all jobs arrive at the same time with some order.

<table>
<thead>
<tr>
<th></th>
<th>JOB1</th>
<th>JOB2</th>
<th>JOB3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of job</td>
<td>Heavy compute</td>
<td>Heavy I/O</td>
<td>Heavy I/O</td>
</tr>
<tr>
<td>Duration</td>
<td>5 min</td>
<td>15 min</td>
<td>10 min</td>
</tr>
<tr>
<td>Memory required</td>
<td>50 M</td>
<td>100 M</td>
<td>75 M</td>
</tr>
<tr>
<td>Need disk?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Need terminal?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Need printer?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- For multiprogramming, all three jobs are assumed to run and switch in minimum time unit in a multitasking environment.

- So they are virtually running concurrently.
### JOB1
- **Type of job**: Heavy compute
- **Duration**: 5 min
- **Memory required**: 50 M
- **Need disk?**: No
- **Need terminal?**: No
- **Need printer?**: No

### JOB2
- **Type of job**: Heavy I/O
- **Duration**: 15 min
- **Memory required**: 100 M
- **Need disk?**: No
- **Need terminal?**: Yes
- **Need printer?**: No

### JOB3
- **Type of job**: Heavy I/O
- **Duration**: 10 min
- **Memory required**: 75 M
- **Need disk?**: Yes
- **Need terminal?**: No
- **Need printer?**: Yes
Effects of Multiprogramming

<table>
<thead>
<tr>
<th></th>
<th>Uniprogramming</th>
<th>Multiprogramming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor use</td>
<td>20%</td>
<td>40%</td>
</tr>
<tr>
<td>Memory use</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Disk use</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Printer use</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Elapsed time</td>
<td>30 min</td>
<td>15 min</td>
</tr>
<tr>
<td>Throughput</td>
<td>6 jobs/hr</td>
<td>12 jobs/hr</td>
</tr>
<tr>
<td>Mean response time</td>
<td>18 min</td>
<td>10 min</td>
</tr>
</tbody>
</table>

- Elapsed time: total time used to complete all jobs
- Throughput: number of tasks completed per unit time
- Response time (or turnaround time): time from submission of a job until its completion
Time Sharing Systems (TSS)

- Batch multiprogramming does not support interaction with users
- TSS extends multiprogramming to handle multiple interactive jobs
- Processor’s time is shared among multiple users
- Multiple users simultaneously access the system through terminals
- It is possible because human reaction is slow (comparing to CPU)
25 years of Linux in 5 Minutes

@linuxquestions

#ATO2016
Major OS Advances

... and topics of chapters yet to come!
Major Achievements of OS

To meet the needs of multiprogramming and time sharing, there have been 5 major OS achievements:

- Processes
- Memory management
- Scheduling and resource management
- Information protection and security
- System structure
Process

Many definitions

- a program in execution
- an instance of a program running on a computer
- the entity that can be assigned to and executed on a processor
- a unit of activity characterized by a single sequence of execution, a current state, and an associated set of system resources

Let’s use the following working definition

- An instance of a program that is in execution
Process

- Processes are fundamental to the structure of operating systems
  - The ability to switch between different processes facilitates multiprogramming
  - The OS can switch to a different process while waiting for I/O in another
  - The objective is to improve system utilization

- OS support for processes took a lot of trial and error, because...
Lots of things can go wrong:

- Improper synchronization
  - Consider the case when a program must wait for a signal from another program to be received
  - Improper design of the signaling mechanism can result in loss or duplication of the signal

- Failed mutual exclusion
  - Multiple processes can very easily need to access the same resource at the same time
  - For example, two processes might be writing to the same file at the same time
  - The OS must provide a mechanism to guarantee that only one process can access the resource at a given time
  - Mutual exclusion mechanisms are tricky to get right and hard to test
Lots of things can go wrong:

- **Nondeterministic program operation**
  - Execution of a given program should be deterministic, i.e., not dependent on external factors
  - Program execution is interleaved and memory locations are sometimes shared
  - Result of operations can depend on the order in which the processes are executed

- **Deadlocks**
  - Two or more processes might be hung up waiting for the other two finish (circular wait)
  - For example, two processes each might need two I/O devices to proceed (read from one, write to the other); each process has a lock on one I/O device, so neither can proceed
Memory Management

- The OS has five principle storage management responsibilities
  1. Process isolation:
     - prevent independent processes from interfering with each other’s memory
  2. Automatic allocation and management:
     - memory management should be transparent to the programmer
  3. Support of modular programming:
     - programmers should be able to define program “modules” that can be independently loaded
Memory Management

- The OS has five principle storage management responsibilities

4. Protection and access control:
   - processes may want to share data
   - the OS should facilitate this while still enforcing appropriate limits

5. Long-term storage:
   - many programs require means for storing data for extended periods of time
   - i.e., non-volatile storage that persists after the computer has been turned off
Memory Management

- The key contribution is virtual memory.

- It allows programs to address memory from a logical point of view without regard to the amount that is physically available.

- While a program is running only a portion of the program and data is kept in (real) memory.
  - conceived to meet the requirement of allowing multiple processes to be in memory at the same time.

- Other portions are kept in blocks on disk.
  - the user has access to a memory space that is larger than real memory.
Paging

- Programs are divided into fixed-size blocks, called pages
  - only the referenced pages of a process actually need to be in memory for the process to run
  - before paging, the entire process had to be loaded into memory

Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.
Paging

- Programs reference data by means of a “virtual address”
- Physically, each page may be located anywhere in main memory
- Paging allows a dynamic mapping of virtual addresses to real physical address (by the OS)
As the process is swapped in and out of memory
- the virtual address remains the same
- the real address may change depending on where in memory the pages are placed
Scheduling and Resource Management

- OS is a resource scheduler
- It should be fair
  - ideally, all processes competing for a resource should be given equal access
- It should support differential responsiveness
  - fair is good, but some jobs have different requirements for responsiveness
  - the OS should attempt to satisfy all requests in priority order
Scheduling and Resource Management

- OS is a resource scheduler

- It should be efficient
  - maximize throughput (number of tasks completed per unit time)
  - minimize turnaround time (time from submission of a job until its completion)
  - these can be at odds; must find some happy medium
Security and Protection

- Access control to resources
  - forbid intruders (unauthorized users) to enter the system
  - forbid user processes to access resources which they are not authorized to
Because of its enormous complexity, we view the OS system as a series of levels. Each level performs a related subset of functions. Each level relies on the next lower level to perform more primitive functions. Well defined interfaces: one level can be modified without affecting other levels. This decomposes a problem into a number of more manageable sub problems.
We have discussed the achievements

- Processes
- Memory management
- Scheduling and resource management
- Information protection and security
- System structure
Microkernel Architecture

- Early OS was implemented as monolithic kernel
  - all OS functionality was implemented in a single, large kernel, typically in a single process
  - complexity grows as functionality is added

- Microkernels
  - only a few essential functions to the kernel (like process scheduling and memory management)
  - other OS functions are provided by separate processes, often called servers
  - these separate processes are treated the same as any other process by the kernel
  - It simplifies implementation, provides flexibility, and well suited to distributed environments
Multithreading

- Definition: a technique in which a process can be divided into multiple “threads” that can run concurrently

- A thread
  - a dispatchable unit of work
  - includes its own dedicated context metadata, in addition to overall process context
  - executes sequentially and, like a process, is interruptible
Multithreading

- Then a process becomes
  - a collection of one or more threads and any associated system resources

- Benefits:
  - the programmer has much greater control over the modularity and timing of application events
  - multiple threads in a single process can be executed concurrently if multiple processors
Symmetric Multiprocessing

- We covered symmetric multiprocessors earlier
  - more processors of same capabilities in a single system
- Symmetric multiprocessing:
  - several processes can (physically) run in parallel
- Multiple processors are transparent to the user
Distributed Operating Systems

- Multi-computer systems (like supercomputers)
  - each computer has its own dedicated processors, main memory, and other components

- A distributed OS manages all these computers
  - provides the illusion of a single memory space, file system, and other facilities
  - this entails not just inter-process communication, but inter-computer communication and synchronization
  - Dealing with communication lag is a challenge
Linux Overview