Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Practical 2

- Some facts and figures:
  - no. of submissions by the deadline: 116* (86% of class size (135) according to SITS)
  - no. that get full marks (provisionally): 54 (47% approx of submissions)
  - no. that fall short (provisionally): 35 (30% approx of submissions)
  - no. that got zero (provisionally): 27 (23% approx of submisisons)
    - some of these exposed interesting differences between running pthreads on Ubuntu vs. OS X !

* same as for Practical 1 !

Input & Output (I/O)

- A running program often needs to perform I/O operations with one or more devices (e.g. storage, display, network or user input devices)
- For reasons of efficiency, fairness and safety, I/O operations are usually performed by the operating system on behalf of a requesting process

File Storage

- Many different programs need to share file storage space in a structured manner
- File I/O is managed by the operating system to prevent processes from corrupting the data stored by other processes

Communication

- Often one process needs to communicate with another process, either locally on a single computer, or on a remote computer across a network
- In either case, the inter-process communication service is provided by the operating system

Error Detection

- The operating system should handle unexpected error events in an appropriate manner
- Events might include loss of power, prohibited memory access, stack overflow, I/O error, unexpected storage volume removal, …

Accounting

- For many reasons, it is useful to record important system events and gather statistics on a computer system
- For example, to determine the reason behind some unexpected event, to analyse (and charge) the resources used by individual users or to detect performance bottlenecks

Protection

- Processes executing concurrently on a computer system must be prevented from interfering with each other
- Important or sensitive information must be protected and hidden from unauthorized access, either from inside the system or by an outside party

Program's View

- Loosely, programs can now be divided into two categories:
  - Application programs
    - Programs concerned with providing an end-user functionality, e.g.
      - word-processor, telephone exchange, video editing, engine management
  - System programs
    - Programs dealing with aspects of the computer system, e.g.
      - Device drivers, schedulers…

Program View – Application Programs (Apps)

- Apps are written for a standard virtual machine, very close to the real hardware of the system, but standardised to give a uniform and safe program environment.
- Apps have no ability (or right) to do anything except execute their instructions. For everything else, they must ask the OS
  - They become clients of the OS
  - Services are requested via System Calls.
  - The set of services, call formats and data formats becomes the OS's API.
- Apps run in the user mode, and everything else is "walled off"

Program View – System Programs

- System software deals with aspects of the whole computer system.
- Sometimes, therefore, system software needs more "rights"than application software, e.g. to access peripheral interfaces, to access an app's memory, etc.
- Some OSes allow all system software complete access to the computer system
- Other OSes limit the access given to system software depending on the software's role and security, etc., keeping the highest level of access for a small set of core facilities

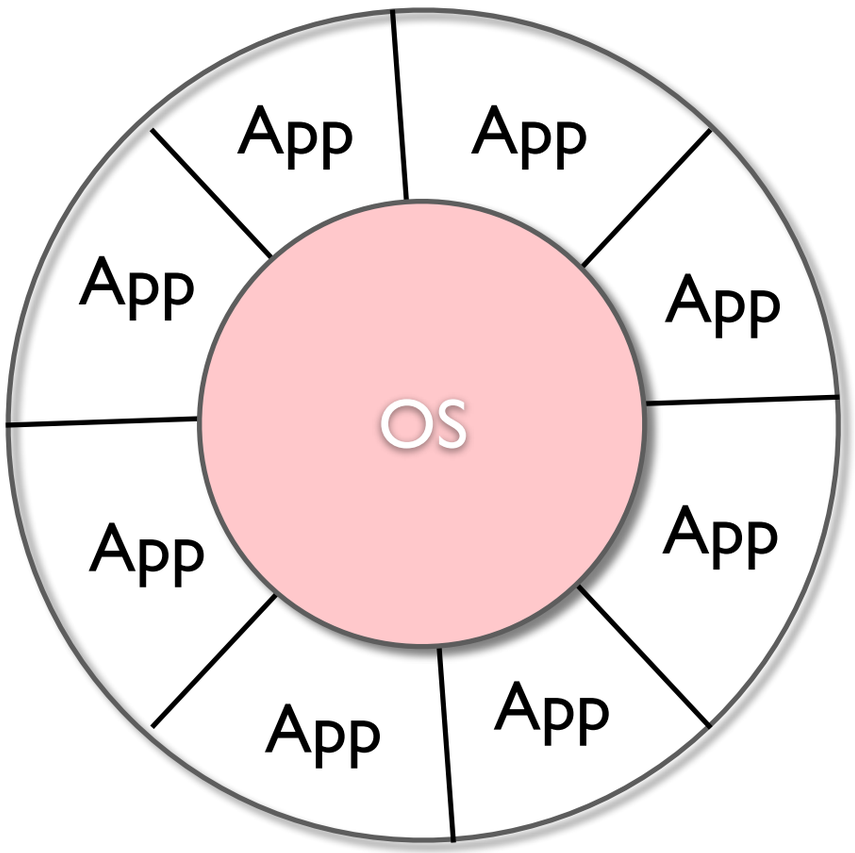
A Programmer's Model – the Process/Kernel Model

- Apps run as one or more *processes*
- A process is a self-contained unit of program code and data. In the process/kernel model, it has the illusion that it's the only process on the machine. When it's running, the machine is in the *process mode*.
- Whenever it makes a System Call, the machine goes into the *kernel mode*. The process is stopped, and the operating system is running. The OS has full access to all the resources of the machine.

Kernel Mode & Privilege

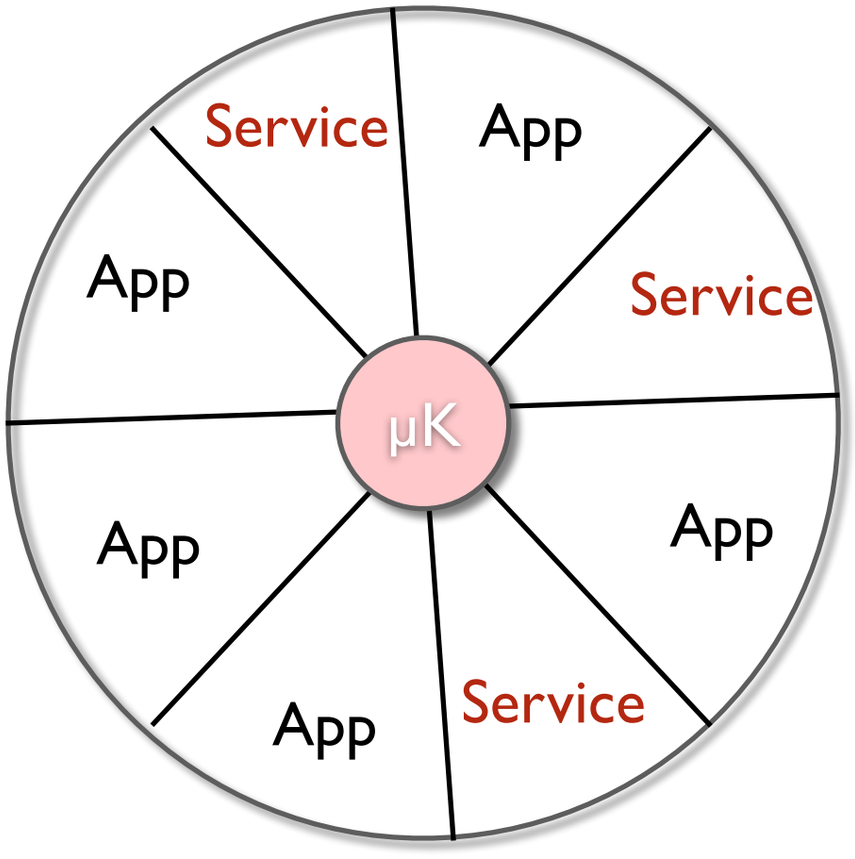
- Normally, the machine has at least two levels of privilege
  - User Mode
  - Kernel Mode (aka Supervisor Mode)
- Some machines implement a hierarchy of privilege.
- A normal application runs at the lowest level of privilege
- OS programs run at a higher level of privilege.
- Privilege ≠ Priority!

Monolithic Kernel



- If all the layers and component parts of the operating system are completely integrated into one another and all run in kernel mode, then the kernel is said to be monolithic. Linux has a monolithic kernel.

Microkernel



- Some OSes have an inner 'core' or *microkernel* in which a very restricted set of facilities is implemented, e.g. memory management, multiprocessing and inter-process communication, which runs at the highest level of protection.
- The other facilities of the "OS" are implemented as user-level services.
- Mach, from CMU, is microkernel based.

Microkernel Basic Concepts

- The core concept of the microkernel is Inter-Process Communication.
  - When a service is requested, by user or by OS services, an IPC message is formed in memory, and the message passed to the callee via the microkernel.
  - The microkernel is responsible for dispatching the message. The IPC is the only method for communicating between processes.
  - The IPC could be based on shared memory and handled by the memory manager and MMU, but also, and transparently, messages could be copied to other processors, giving multiprocessing.

Components of the Microkernel

- The minimum set of components needed to allow the microkernel to run
  - A scheduler
  - A memory manager
  - A set of drivers for the current hardware
  - An implementation of IPC

Microkernel Considered

- Robust
- Easier to add multiprocessing -- well-defined IPC which is exclusive and readily implementable over shared memory, NUMA, networked machines, etc.
- Relatively portable across different architectures.
- Very influential idea in the 1980s and 1990s
- Apparently severe intrinsic performance problems.
- Ongoing story – GNU Hurd et al…
- See also http://en.wikipedia.org/wiki/Tanenbaum–Torvalds_debate

COVER FEATURE

Can We Make Operating Systems Reliable and Secure?

Andrew S. Tanenbaum, Jorrit N. Herder, and Herbert Bos
Vrije Universiteit, Amsterdam

Monolithic

- + Highest performance, as there are no artificial barriers between components of the OS.
- – Less reliable, as more of the OS is running at full privilege, thus more likely to crash the entire system.
- – Less likely to be modular, as the designers are not forced to consider the separate components of their system.

Hybrid Microkernel

- A microkernel that encompasses more than the original minimal set of functions of the microkernel.
- The rest of the OS is implemented at lower levels of privilege.
- Mac OS X is based on a hybrid microkernel.
- Similarly Windows NT was based on a central ‘kernel’ and an accompanying ‘executive’.