Efficiency: Multiprocessing: operating systems; 2 mechanisms

COSC 208, Introduction to Computer Systems, 2021-11-05

Announcements

- Project 2 Part B due date extended to Tues, Nov 9

Outline

- Warm-up
- Cache replacement
- OS overview
- Accessing hardware
- Limited direct execution
- System calls

Warm-up

Q1: *For each of the following instances of caching, indicate whether the caching is motivated by temporal or spatial locality.*

- A CPU caches the first 32 instructions of a function when the function is called
- A CPU caches all of the instructions for a frequently called function
- A web browser caches the Moodle pages for your courses, which you view multiple times per week
- A content distribution network (CDN) caches a video that has gone viral
- A content distribution network (CDN) caches "recommended videos" related to a popular video

Process abstraction

Q2: *Consider building a Lego kit as an analogy for operating systems' process abstraction. Match each component of the analogy with the corresponding component of a real computer system.*

- Analogy
    - Cabinet/drawers for storing Legos
    - Lego bricks
    - Building area (e.g., tabletop)
    - Instruction booklet
    - Following the assembly instructions
    - Current step for the instruction booklet
    - Completed kit
    - You
- Real system
    - CPU
    - persistent storage
    - process
    - program
    - program counter
    - program inputs
    - program outputs
    - registers and main memory

Operating systems (OS) overview

- Purpose of an OS
    - Make computer hardware easy to use—e.g., an OS knows how to load an application's executable code from persistent storage (e.g., solid state drive (SSD)) into main memory, initialize the process's address space (code, heap, stack), and make the CPU execute the application's instructions
    - Support multiprocessing—i.e., running multiple applications concurrently
        - Concurrently == switch between multiple tasks during a window of time—e.g., alternate between cooking and setting the table
        - Parallel == complete two tasks at the same time—e.g., listen to a podcast while cooking
    - Allocate and manage hardware resources
    - Modern OSes also provide a user interface (UI)
- How does the OS fulfill its duties?
    - Mechanisms — fundamental approaches for managing/providing access to hardware resources
    - Policy — specific ways of employing an approach; different policies make different trade-offs

Accessing hardware

- Recall: OS is responsible for allocating/managing the hardware
    - ⇒ applications should **not have unfettered access to hardware**
- Example: execute an instruction on the CPU
    - Allow the application to execute instructions in a "supervised" manner
- Example: accessing the solid state drive (SSD)
    - Ask the OS to access the SSD on the application's behalf—latency of accessing SSD (~1 million CPU cycles) far outweighs the extra instructions required for OS to perform the access on the application's behalf
- Two Mechanisms

Limited Direct Execution

- CPU has two modes of execution: user mode & kernel mode
- *When does a CPU run in user mode?* — when executing application code
- *When does a CPU run in kernel mode?* — when executing OS code
- Allowable operations in user mode are restricted
    - Applications can...
        - Perform arithmetic/logic operations
        - Load/store values in its stack/heap
    - Applications must ask the OS to...
        - Start/terminate applications
        - Create/delete files
        - Display output on screen
        - Read input from user
    - Must transfer control to the OS to perform these operations — How?

System calls

- Invoked via a special assembly instruction: trap (generic) or svc (on ARM)
    - Prior to invoking trap, put system call number for desired operation into a register
- What does the hardware do when a trap instruction is executed?
    1. Save registers to the kernel stack
    2. Switch to kernel mode
    3. Use system call number to index into table of trap handlers — get memory address of first instruction of trap handler
    4. Branch and link to trap handler code (trap handler executes)
    5. Restore registers from the kernel stack
    6. Switch to user mode
    7. Branch to instruction referenced by program counter