CS 347M (Operating Systems Minor)

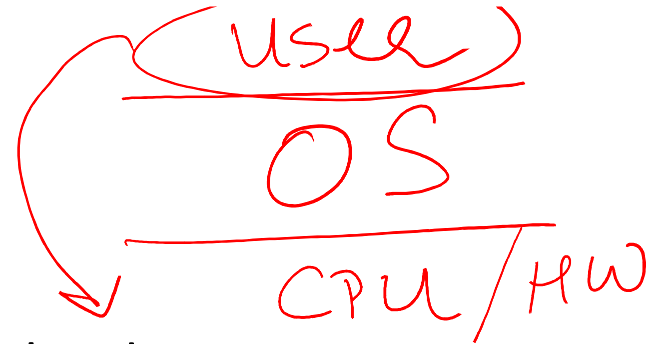
Spring 2022

Lecture 1: Introduction to Operating Systems

Mythili Vutukuru

CSE, IIT Bombay

What is an operating system?



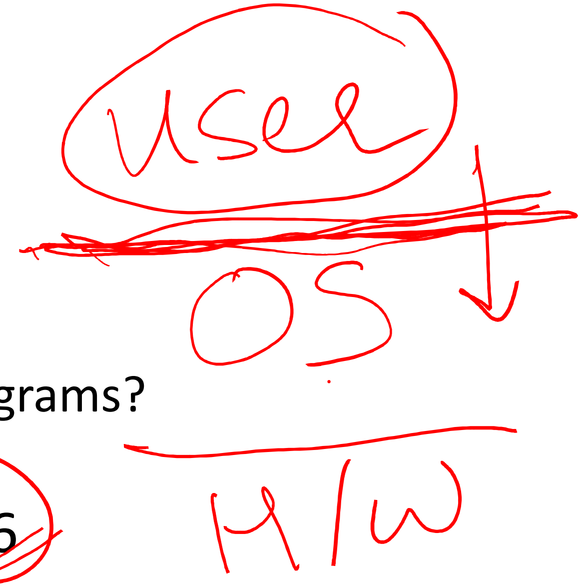
- Middleware between user programs and system hardware
  - Not user application software but system software
  - Example: Linux, Windows, MacOS
- Manages computer hardware: CPU, main memory, I/O devices (hard disk, network card, mouse, keyboard etc.)
  - User applications do not have to worry about system details
- Operating system has kernel + system programs
  - Kernel = the core part of the operating system
  - System programs are useful programs to manage system (e.g., program to list all files in a directory “ls”)

History of operating systems

theory
applications
systems ✓

- Started out as a library to provide common functionality to access hardware, invoked via function calls from user program
  - Convenient to use OS instead of each user writing code to manage hardware
  - Centralized management of hardware resources is more efficient
- Later, computers evolved from running a single program to multiple processes concurrently
  - Multiple untrusted users must share same hardware
- So OS evolved to become trusted system software providing isolation between users, and protecting hardware
  - Multiple users are isolated and protected from each other
  - System hardware and software is protected from unauthorized access by users

Course overview



- How operating systems work
  - What is the functionality provided by OS to user programs?
  - How is the functionality of an OS implemented?
- Examples from a simple “teaching” OS called xv6
  - Read xv6 code to fully understand each OS concept
- Programming assignments to reinforce theory
  - Write user programs using OS functionality
  - Adding new functionality to OS within xv6
- Better understanding of computer systems in general
  - Write better programs for any application in future, by knowing how OS works

Course logistics

- Live classes during regular lecture hours on MS Teams
  - Recording available on Teams for those who cannot attend
  - Attendance to online is strongly encouraged for better learning
  - May move to hybrid class model later on (COVID rules permitting)
- Course Moodle Page
  - Summary of weekly content taught, PDF slides from class
  - Programming assignments problem statement and submission link
  - Discussion forum to ask doubts after class
- References and Textbooks:
  - Course material archive (videos, slides, practice problems etc.) https://www.cse.iitb.ac.in/~mythili/os/
  - Highly recommended online textbook: Operating Systems: Three Easy Pieces (OSTEP) http://pages.cs.wisc.edu/~remzi/OSTEP/

Evaluation and Grading

endsem

- Four modules in the course: processes, memory, concurrency, I/O
- 3 quizzes, one for each module, 10% each
    - End of Jan, end of Feb (midsem week), end of March
    - Online proctored exams
- 3 programming assignments, one for each module, 10% each
    - Due roughly before the corresponding quizzes
    - Evaluation via demo+viva with TAs
- End-semester exam covering the entire syllabus, 40%
    - Online proctored or offline exam (COVID situation permitting)

Programming assignments

- Very important part of the course, to understand the concepts better
- Will require non-trivial amount of programming
  - Please take this course only if you have an interest/aptitude for programming
- Individual assignments, to be solved and submitted by each of you
- Discussions with classmates is ok, but no showing/sharing/copying of code is allowed, all code you submit must be written by you only
  - Plagiarism detection will be run over all submissions
  - Any copying detected will result in FR for all involved students
- Demo of test cases, viva to explain your code and answer questions

Clarifying doubts

- If you have questions or doubts on the course contents:
  - Ask in online class (type in chat box, unmute and speak when asked to)
  - Ask on the Moodle discussion forum. You are encouraged to answer questions from your classmates as well
  - In-person office hours when permitted by COVID guidelines
- If you would like to discuss anything with me in person, please send me email mythili@cse.iitb.ac.in
  - Please avoid using personal email for regular doubts that you can ask in class
  - Please do not send private chat messages on Moodle or Teams
- Please DO NOT hesitate to approach me and talk to me for any question or doubt, no doubt is too stupid to ask

Missing exams and assignment deadlines

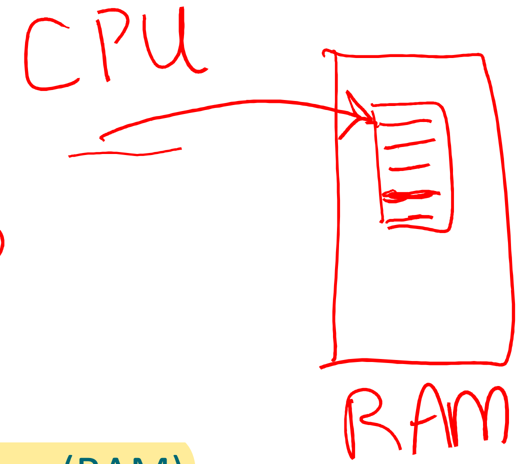
- If you have a medical or any other emergency before a quiz or a programming assignment deadline:
  - Please get in touch with me early before the deadline (not after), with suitable proof (e.g., medical certificate)
  - Send email to me with request, cc TA for programming assignments
- Suitable help will be provided as best as we can
  - Assignment deadlines can be extended by 1 or 2 days on a case-by-case basis
  - Makeup quiz will be conducted at the end of the semester, covering all modules, for those who missed any one of the quizzes

Rest of this lecture

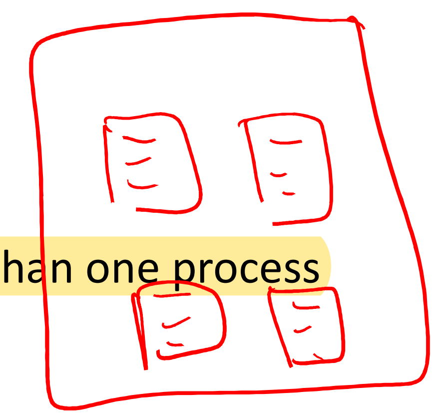
- Overview of computer hardware
  - Not the focus of this course, only high level understanding is expected
- Introduction to OS functionality

CPU and memory

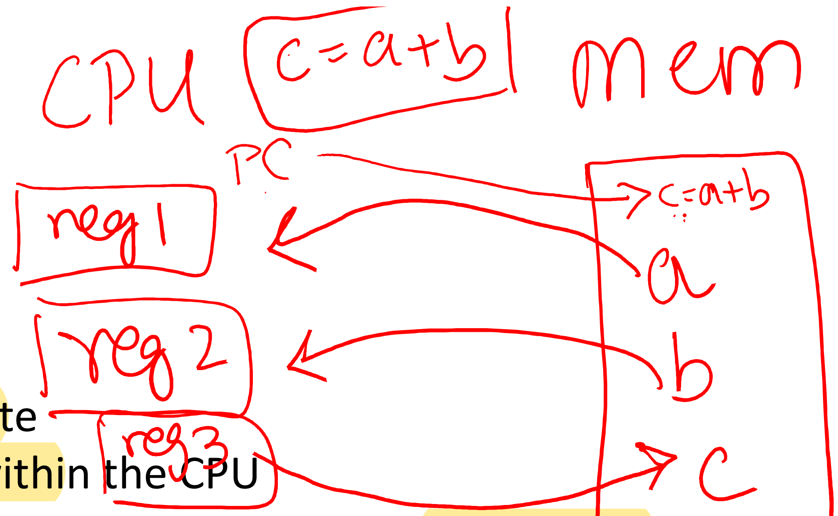
CPU — memory — I/O



- User program = code (instructions for CPU) + data
- Stored program concept
  - User programs stored in main memory or Random Access Memory (RAM)
  - Instructions/data occupy multiple contiguous bytes in memory
  - Memory is byte-addressable: data accessed via memory address / location / byte#
  - CPU fetches code/data from RAM using memory address, and executes instructions
- CPU runs processes = running programs
- Modern CPUs have multiple CPU cores for parallel execution
  - Each CPU core runs one process at a time each
  - Modern CPUs have hyperthreading, where each core can run more than one process also (OS treats hyper-threading cores also as multiple CPU cores)



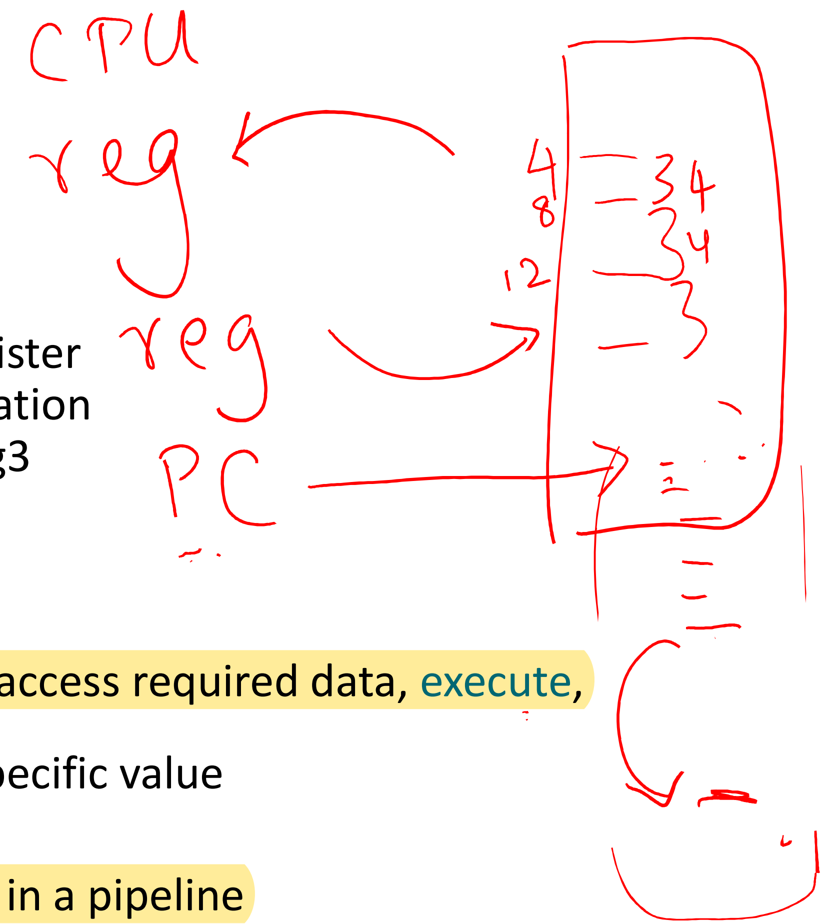
Instructions and registers



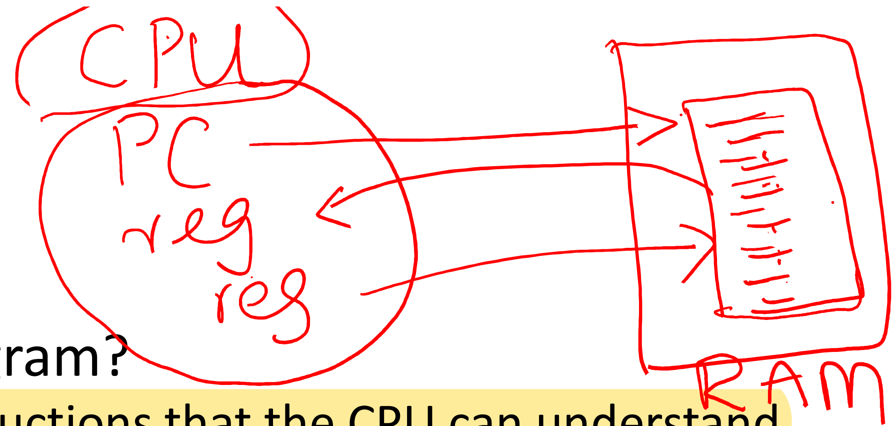
- Every CPU has
  - A set of instructions that the hardware can execute
  - A set of registers for temporary storage of data within the CPU
- High level language (C code) translated into CPU instructions by compiler
  - Can directly write assembly language code, but cumbersome
- Instructions and registers defined by ISA = Instruction Set Architecture
  - Specific to CPU manufacturer (e.g., Intel CPUs follow x86 ISA)
- Registers: special registers (specific purpose) or general purpose
  - Program counter (PC) is special register, has memory address of the next instruction to execute on the CPU
  - General purpose registers can be used for anything, e.g., operands in instructions
- Size of registers defined by architecture (32 bit / 64 bit)

CPU instructions

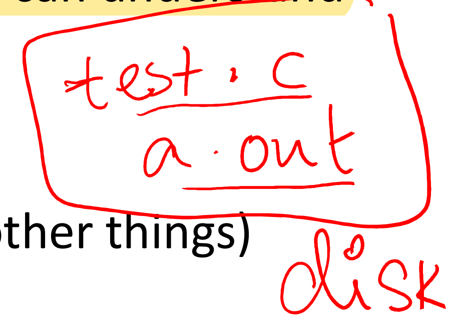
- Some common examples of CPU instructions
  - Load: copy content from memory location → register
  - Store: copy content from register → memory location
  - Arithmetic operations like add: reg1 + reg2 → reg3
  - Logical operations, compare, …
  - Jump: set PC to specific value
- Simple model of CPU
  - Each clock cycle, fetch instruction at PC, decode, access required data, execute, update PC, repeat
  - PC incremented to next instruction, or jump to specific value
- Many optimizations to this simple model
  - Pipelining: run multiple instructions concurrently in a pipeline
  - Many more in modern CPUs to optimize #instructions executed per clock cycle



Running a program



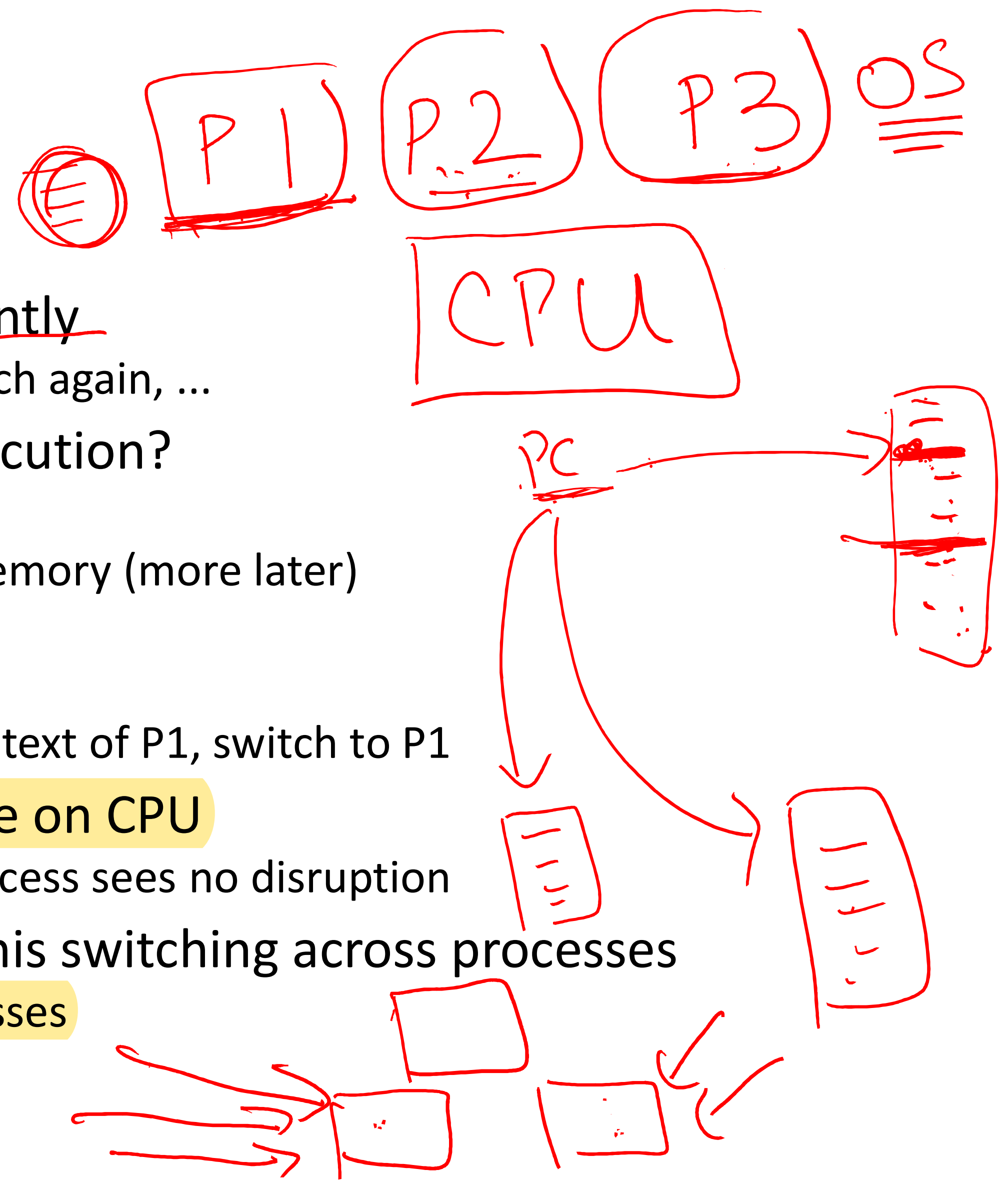
- What happens when you run a C program?
  - C code translated into executable = instructions that the CPU can understand
  - Translation done by program called compiler
  - Executable file stored on hard disk (say, "a.out")
  - When executable is run, a new process is created in RAM
  - Memory image of process in RAM contains code+data (and other things)
  - CPU starts executing the instructions of the program



- When CPU core is running a process, CPU registers contain the execution context of the process
  - PC points to instruction in the program, general purpose registers store data in the program, and so on

Concurrent execution

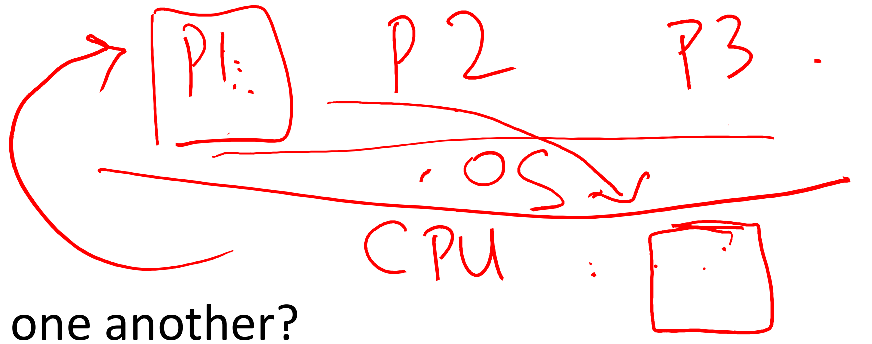
- CPU runs multiple programs concurrently
  - Run one process, switch to another, switch again, ...
- How to ensure correct concurrent execution?
  - Run process P1 for some time
  - Pause P1, save context somewhere in memory (more later)
  - Load context of P2 from memory
  - Run P2 for some time
  - Pause P2, save context of P2, restore context of P1, switch to P1
- Every process thinks it is running alone on CPU
  - Saving and restoring context ensures process sees no disruption
- Operating System (OS) takes care of this switching across processes
  - OS virtualizes CPU across multiple processes



Interrupt handling

- In addition to running user programs, CPU also has to handle external events (e.g., mouse click, keyboard input)
- Interrupt = external signal from I/O device asking for CPU's attention
- How are interrupts handled?
  - CPU is running process P1 and interrupt arrives
  - CPU saves context of P1, runs code to handle interrupt (e.g., read keyboard character)
  - Restore context of P1, resume P1
- Interrupt handling code is part of OS
  - CPU runs interrupt handler of OS and returns back to user code

Isolation



- How to protect processes from one another?
  - Can one process mess up the memory or files of another process?
- Modern CPUs have mechanisms for isolation
- Privileged and unprivileged instructions
  - Privileged instruction = access to sensitive information (e.g., hardware)
  - Regular instructions (e.g., add) are unprivileged
- CPU has multiple modes of operation (Intel x86 CPUs run in 4 rings)
  - Low privilege level (e.g., ring 3) only allows unprivileged instructions
  - High privilege level (e.g., ring 0) allows privileged instructions also
- User code has unprivileged instructions, runs at low privilege level
  - CPU does not execute privileged instructions when in unprivileged user mode
- OS code has privileged instructions, runs at high privilege level
- When user program wants to do privileged operations, it must ask OS
  - CPU shifts to high privilege level, runs OS code, returns to low privilege, back to user code

CPU

reg

CPU caches

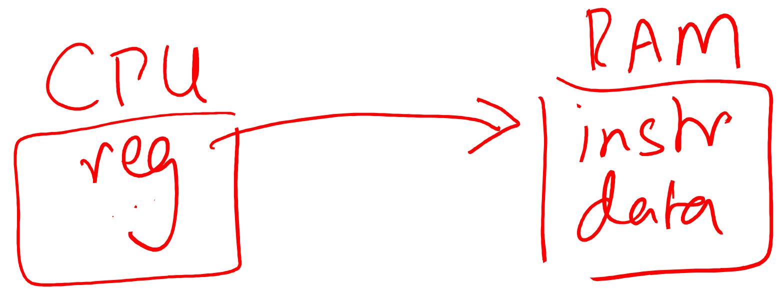
for

caches

RAM

- CPU must access memory to fetch instructions, load data into registers
  - But main memory (DRAM) is very slow (100s of CPU cycles)
  - CPU cannot do useful work while waiting for memory
- To avoid many memory accesses, CPU stores recently accessed instructions and data in CPU caches
  - Multi-level cache hierarchy, some private to cores, some common
  - Example: private L1, L2, common last level cache (LLC or L3)
  - Can be separate for instructions and data, or common (e.g., L1 is separate)
- Caches have low access latency (tens of CPU cycles), faster than DRAM but smaller in size, more expensive
  - Can only store most recently used instructions and data

Memory hierarchy



- Hierarchy of storage elements which store instructions and data
  - CPU registers (small number, <1 nanosec)
  - CPU caches (few MB, 1-10 nanosec)
  - Main memory or RAM (few GB, ~100 nanosec)
  - Hard disk (few TB, ~1 millisec)
- Hard disk is non-volatile storage, rest are volatile
  - Hard disk stores files and other data persistently
- As you go down the hierarchy, memory access technology becomes cheaper, slower, less expensive