Operating Systems 412

Pete Keleher

(some material from Shankar, Agrawala, Youjip Won)

Today

- Administrivia
- Motivation:
  - Why study operating systems?
  - What are operating systems?

Logistics

- Professor: Peter Keleher
  - 5146 Iribe Bldg
  - keleher@umd.edu
  - Class Webpage:
  - https://ceres.cs.umd.edu/412

- Communication:
  - Piazza
  - Office hours
  - Last resort: email me: ***include* 412 *in subject.***
  - **Do not** message me on ELMS, I do not use ELMs.

Logistics

Grading

All grades will be on grades.cs.umd.edu.

45% Projects

We have eight (8) graded projects:

- P0: 5%
- P1: 6%
- P2: 5%
- P3: 5%
- P4a: 6%
- P4b: 6%
- P5a: 6%
- P5b: 6%

All are due **Friday at midnight**. Projects may be submitted up to two days late, 10% off per day.

45% Exams

We have (3) exams, each 15%. *There is no final exam.*

10% Reading Homeworks

We will have approximately 10 weekly reading homeworks.

- The total will be 10%, with weight apportioned equally.
- All are due **Tuesday at noon**. No late homeworks are accepted.

Logistics

- Grading
  - Whole class is curved: avg is B-, stdev up/down for A-, C-
  - Approximate cut-offs ***last year*** (not guaranteed)
    - 85+: A-
    - 75+: B-
    - 65+: C-
    - 60-: D/F
- Most had 40+ points (out of 50) on non-exams last year
  - *Must average a passing grade on the total exam score*

Logistics

- Web site: *https://ceres.cs.umd.edu/412*
- Discussion: *https://piazza.com/umd/spring2025/cmsc412*
- Grades: *https://grades.cs.umd.edu*
- Gradescope: *https://www.gradescope.com/courses/937579*
  - homeworks, assignment submissions, graded exams
- Office Hours
  - Pete (me) IRB 5146, Tues 4:00 - 5:00: *lectures, exams, logistics*
  - TAs (hours TBD): *project questions*
    - Geng Liu ("leo")
    - Tasnim Kabir
- ELMS
  - *Nope!*

Some To-Dos

- Sign up for Piazza !
  - If not already added
- Set up the computing environment (Project Z), and make sure you can run and compile in Docker containers.
- Upcoming:
  - Homework 1 (due Tuesday, Feb 4, at noon),
  - Project Z:
    - set up environment
    - understand structure of GeekOS
    - due this week
  - Project 0: Pipes.
    - This will require a great deal of code and environment exploration.
    - do NOT leave until the last day
    - due Feb 7

"Three Easy Pieces"

- What are we studying?
- Design and implementation of operating systems
- What is an operating system?
- Software layer between hardware and user programs
- Provides useful abstractions:
  - Virtualization
    - processor, memory, storage
  - Concurrency
    - threads, processes, kernel
  - Persistence
    - file systems, hardware

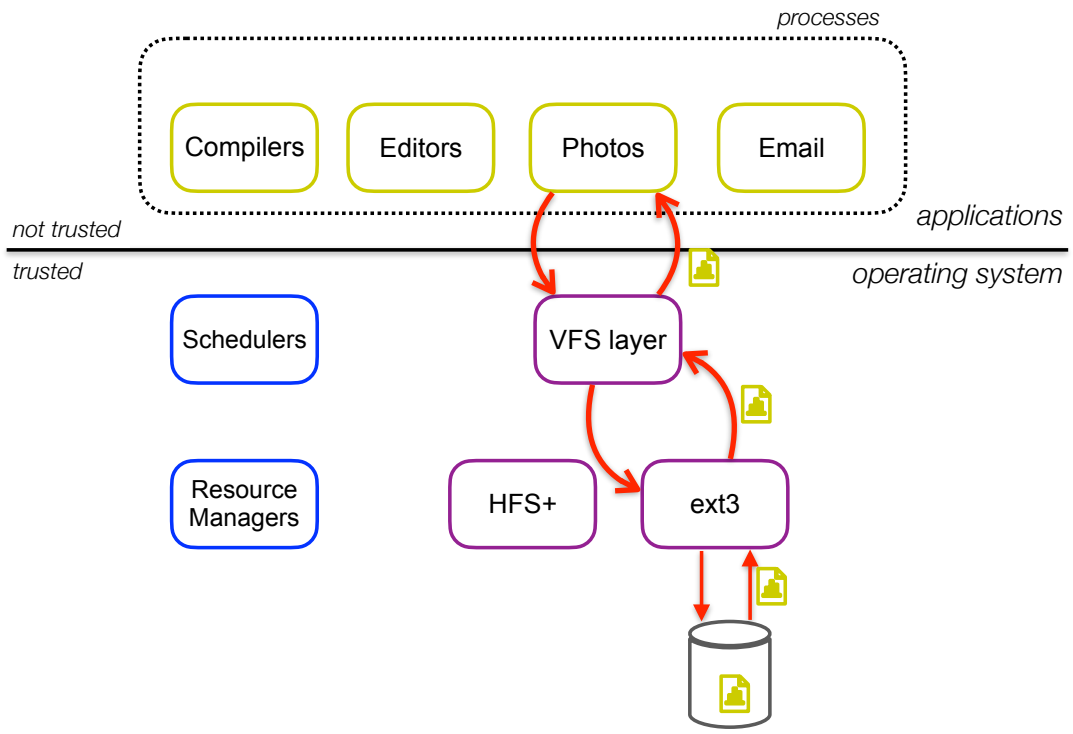
"Three Easy Pieces"

- Operating systems provide useful *abstractions*:
  - Virtualization
    - each process thinks it has exclusive access to processor, memory, storage
  - Concurrency
    - threads / processes have to:
      - work together (synchronize, exchange data)
  - Persistence
    - data is stored on file systems, which rely on a variety of different hardware techniques to make changes *durable*

Processor Virtualization

- Processors usually have many cores
  - …but many more processes
  - need to map multiple process to a core
  - need process to act as if it had full control of core
- CPU virtualization
  - Share a core through *time sharing*
    - give core to a process, let it run
    - context switch to a different process
  - Performance cost

User Space vs Kernel Space



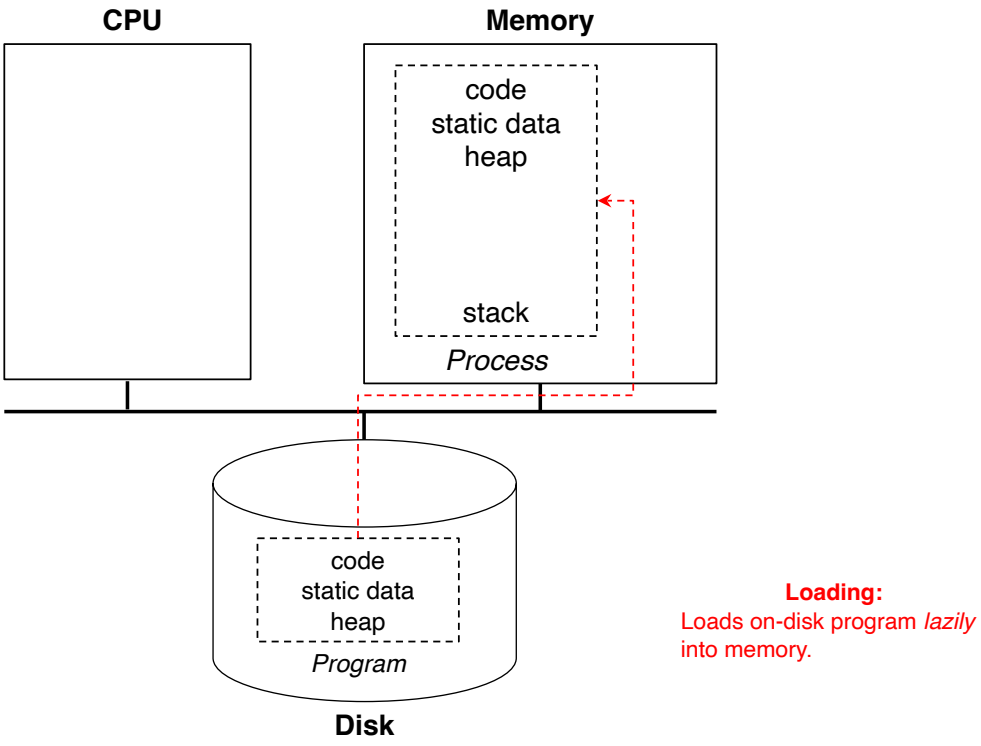
Process Abstraction

- Instance of a running program
  - Memory (address space)
    - Instructions
    - Data
  - Registers
    - Program counter
    - Stack pointer
    - etc…
  - Caches, TLBs….
- API
  - Create
    - create a new process to run a program
  - Destroy
    - halt a runaway process
  - Wait
    - wait for process to terminate
  - Switching support
    - methods to suspend and resume
  - Status

Process Creation

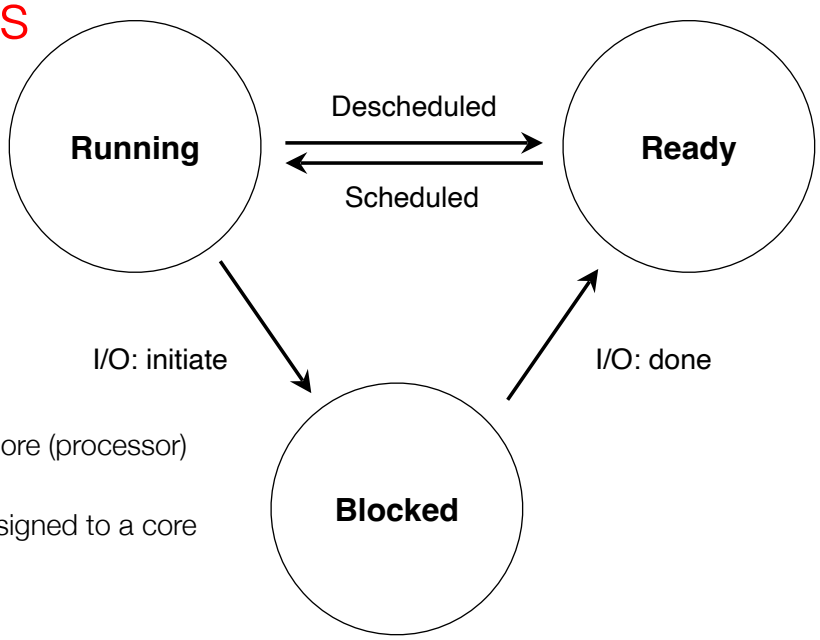
- Load program code into memory
  - Programs on disk in executable format
  - OS performs load *lazily*
    - Program code loaded as needed.
- Allocate run-time *stack*
  - Local vars, function parameters, return addresses
- Program heap
  - Dynamically allocated (*malloc'd*) data
- I/O setup
  - *stdin*, *stdout*, *stderr* file descriptors
- Process start
  - Transfer control

Process Loading



Loading:
Loads on-disk program *lazily* into memory.

Process States



- Running
  - Process occupying the core (processor)
- Ready
  - Ready to run, but not assigned to a core
- Blocked
  - Waiting for an event:
    - message
    - requested data

- Processes assigned to one or more *queues* depending on state

Process State Example

- assume no I/O, no *time-slicing:*

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	$Process_0$ now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	$Process_1$ now done

Figure 4.3: **Tracing Process State: CPU Only**

- processes run to completion before releasing CPU…

Process State Example

- more realistic, with I/O:

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	$Process_0$ initiates I/O
4	Blocked	Running	$Process_0$ is blocked,
5	Blocked	Running	so $Process_1$ runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	$Process_1$ now done
9	Running	–	
10	Running	–	$Process_0$ now done

Figure 4.4: **Tracing Process State: CPU and I/O**

- I/O completion enqueues process on *ready queue*
  - might not run immediately
- still no time-slicing