CS 4414 — Operating Systems

Changelog

changes since lecture:

21 Jan 2022: add explanation slide for exception counting exercise

lectures

recordings available afterwards

- attendance not required

if you aren't watching live, I recommend writing down questions...

- you can ask via Piazza, office hours

course webpage

`https://www.cs.virginia.edu/~cr4bd/4414/S2022/`

linked off Collab

office hours

via Discord and in-person

will be indicated on calendar on website

Discord OH will use online queue

in-person OH may or may not use online queue

homeworks

there will be programming assignments

...mostly in C or C++; one in Python

one or two weeks

- if two weeks "checkpoint" submission after first week

two week assignments worth more

xv6

some assignments will use xv6, a teaching operating system

simplified OS based on an old Unix version

- built by some people at MIT
- (though they currently use a RISC V version instead of the x86-32 version we'll use)

theoretically actually boots on real 32-bit x86 hardware

...and supports multicore!

- (but we'll run it only single-core, in an emulator)

quizzes

there will be online quizzes after each week of lecture

...starting after next week

same interface as CS 3330, but no time limit

(haven't seen it? we'll talk more next Thursday)

quizzes are open notes, open book, open compiler, open Internet

exams

final exam

current plan: take-home, 24 hours, overlapping official final time

(subject to change, will announce later)

late policy

there is a late policy on the website

textbook

recommended textbook:
Anderson and Dahlin, *Operating Systems: Principles and Practice*

no required textbook

alt: Arpaci-Dusseau, *Operating Systems: Three Easy Pieces* **(free PDFs!)**

- some topics we'll cover where this may be primary textbook

alternative: Silberchartz (used in previous semesters)

- full version: Operating System Concepts, Ninth Edition

cheating: homeworks

don't

homeworks are individual

no code from prior semesters (other than your own)

no sharing code, pesudocode, detailed descriptions of code

no using code from Internet/etc., with limited exceptions

- tiny things solving problems that aren't point of assignment
- ...*credited* where used in your code
- e.g. code to split string into array for non-text-parsing assignment
- exception: something explicitly referred to by the assignment writeup
- in doubt: ask

citation

if using small amount of code *clearly not point of assignment*

- e.g. split string into array for non-text-parsing assignment
- e.g. filling arrays of pointers from vectors of strings

not sure what counts? ask

then make sure you cite where you got it in your code

- should not be other student, etc. — no sharing code

if using code clearly part of major objective of assignment

then don't

- e.g. if you find a shell online, don't use it solve the shell assignment

cheating: quizzes

don't

quizzes: also individual

don't share answers

don't IM people for answers

don't ask on StackOverflow for answers

getting help

Piazza

TA and my office hours (will be posted soon)

emailing me

OS definition ambiguity

different exact defintions

'part of OS' v. 'just a program/library'

- example: code to allow moving windows on the screen part of the OS?
- example: code to support printers is part of the OS?

we'll not sweat the details — give general, common principles

history: computer operator

via National Library of Medicine; computer operators operating an Honeywell 800

history: computer operator

"computer operator" was a job

operating system = program that automates (part of) that job?

via National Library of Medicine; computer operators operating an Honeywell 800

what is an operating system?

software that:

Anderson-Dahlin manages a computer's resources

Arpaci-Dusseau provides 'virtual machine': more convenient than real machine

what is an operating system?

software that:

Anderson-Dahlin manages a computer's resources

Arpaci-Dusseau provides 'virtual machine': more convenient than real machine

not contradictory — just different emphases

common theme: better (?) interface than "raw" machine

OS roles

Anderson-Dahlin's taxonomy of things OS's do

referee — resource sharing, protection

protection: program can't interfere with other programs/OS

illusionist — clean, easy abstractions

glue — common services

storage, window systems, authorization, networking, ...

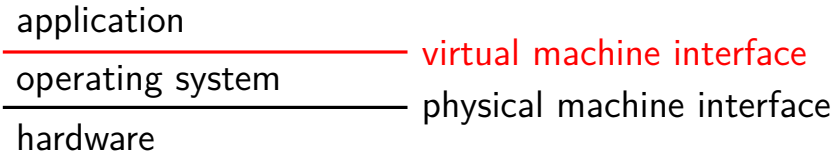
exercise: examples of views

exercise: come up with two examples of
how you (as a user or programmer) notice the OS taking...

the “referee” role (resource sharing; protection)

the “illusionist” role (clean abstractions)

the virtual machine interface



system virtual machine
(VirtualBox, VMWare, Hyper-V, ...)

process virtual machine
(typical operating systems)

imitate physical interface
(of some real hardware)

chosen for convenience
(of applications)

system virtual machines

run entire operating systems

- for OS development, portability

interface $\approx$ hardware interface (but maybe not the real hardware)

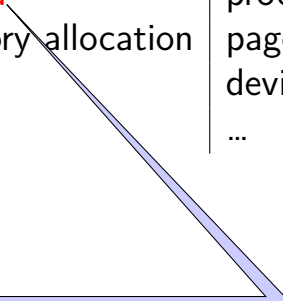
- aid reusing existing raw hardware-targeted code
- different "application programmer"

process virtual machine

process VM	real hardware
thread	processors
memory allocation	page tables
files	devices
...	...

process virtual machine

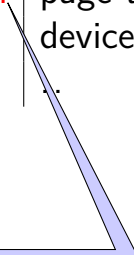
process VM	real hardware
thread	processors
memory allocation	page tables
files	devices
...	...



(virtually) infinite "threads" ($\sim$ virtual CPUs)
no matter number of CPUs

process virtual machine

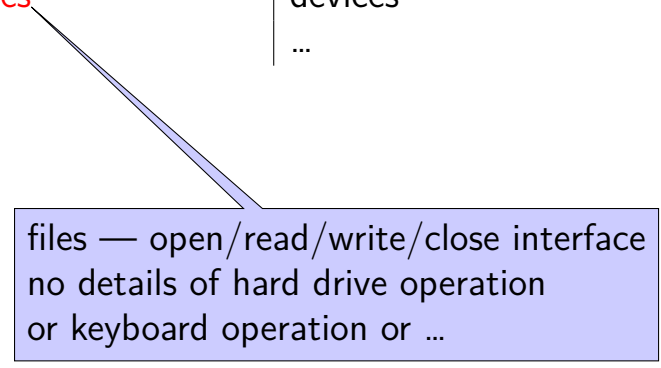
process VM	real hardware
thread	processors
memory allocation	page tables
files	devices
...	...



memory allocation functions
no worries about organization of “real” memory

process virtual machine

process VM	real hardware
thread	processors
memory allocation	page tables
files	devices
...	...



The Process

process = thread(s) + address space + ...

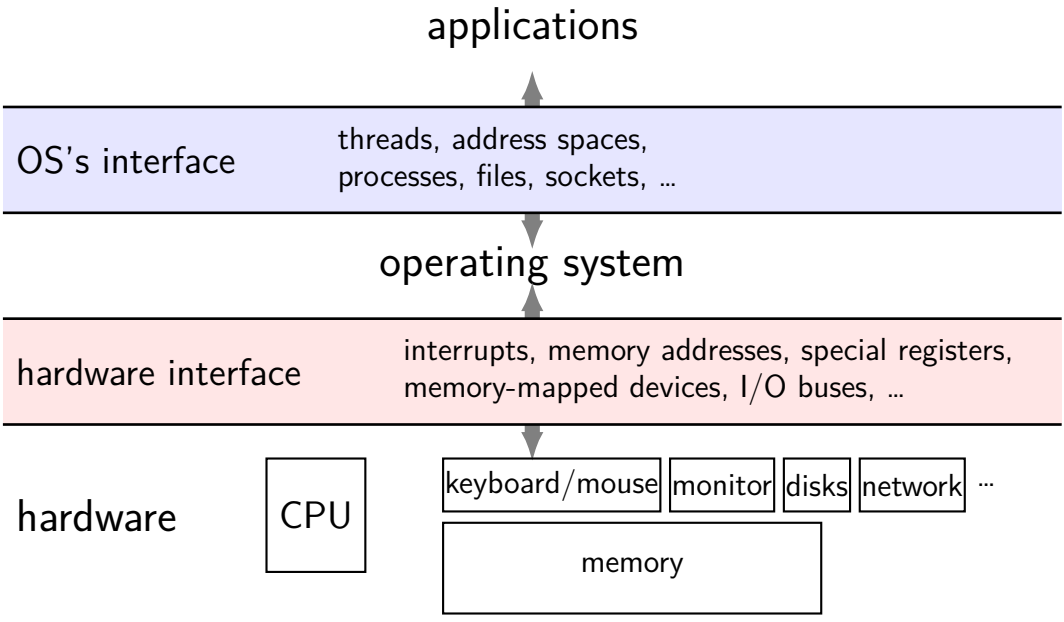
illusion of dedicated machine:

thread = illusion of own CPU

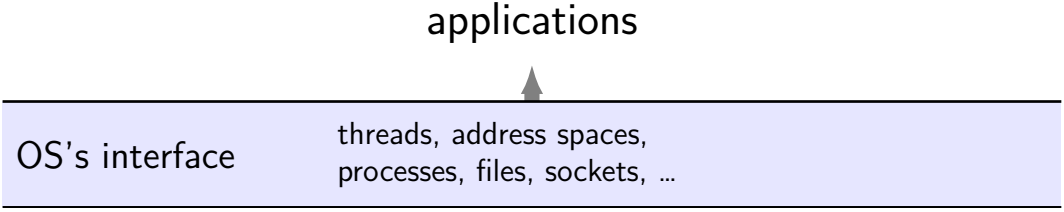
- own program counter and "stream" of instructions to run
- own registers
- ...

address space = illusion of own memory

the abstract virtual machine



abstract VM: application view

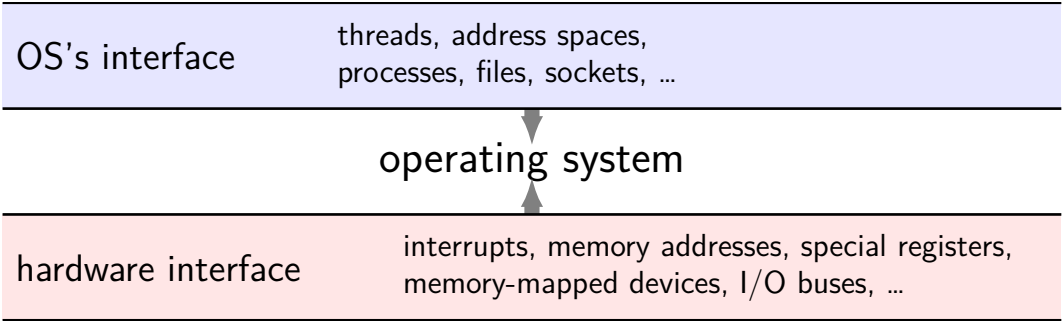


the application's "machine" is the operating system

no hardware I/O details visible — future-proof

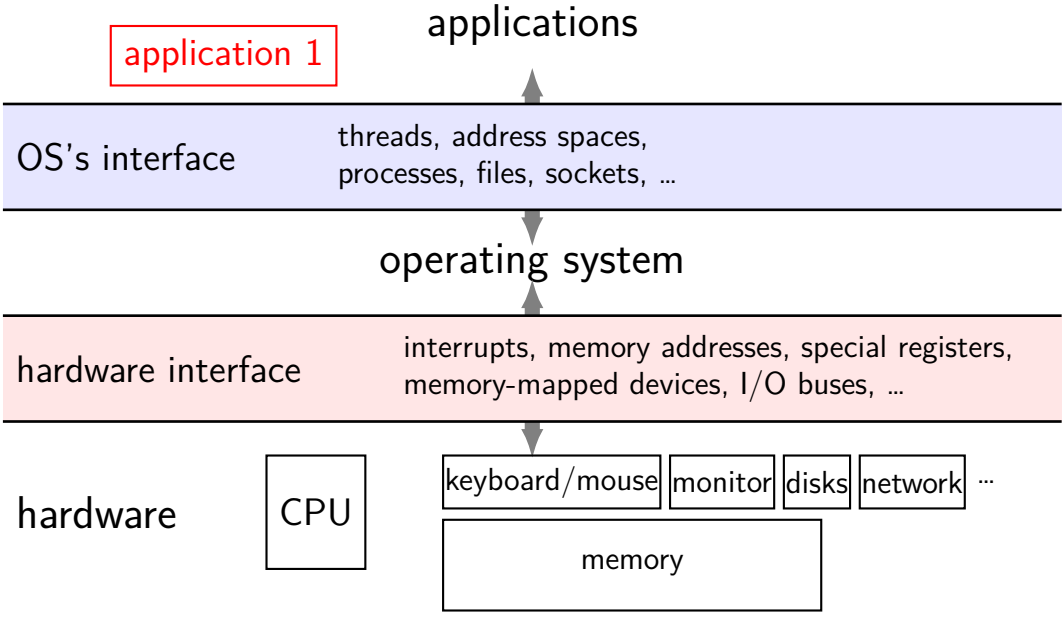
more featureful interfaces than real hardware

abstract VM: OS view

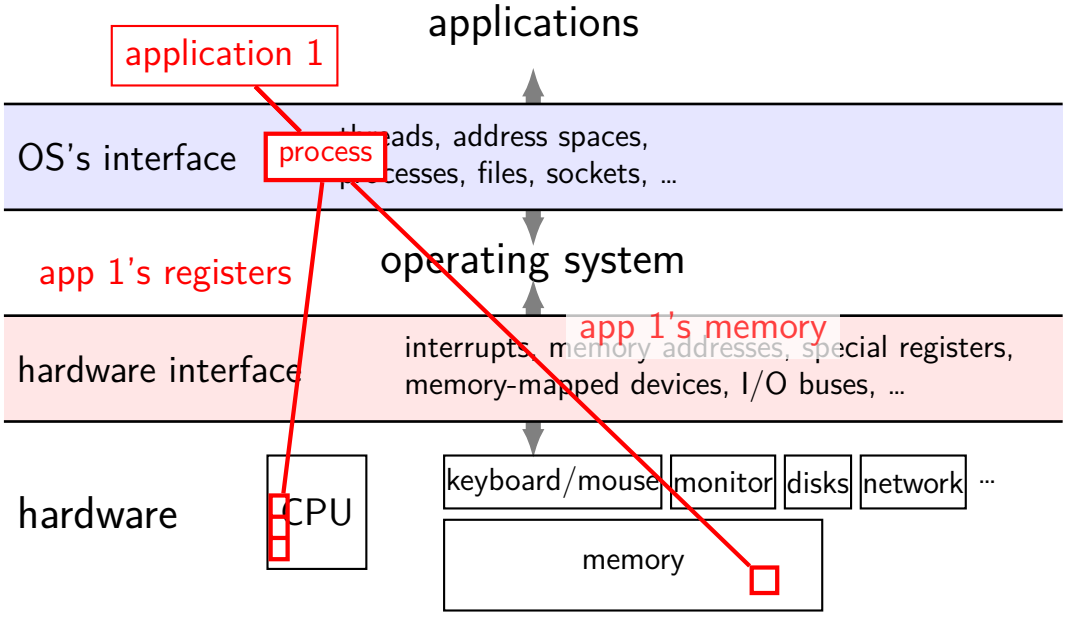


operating system's job: translate one interface to another

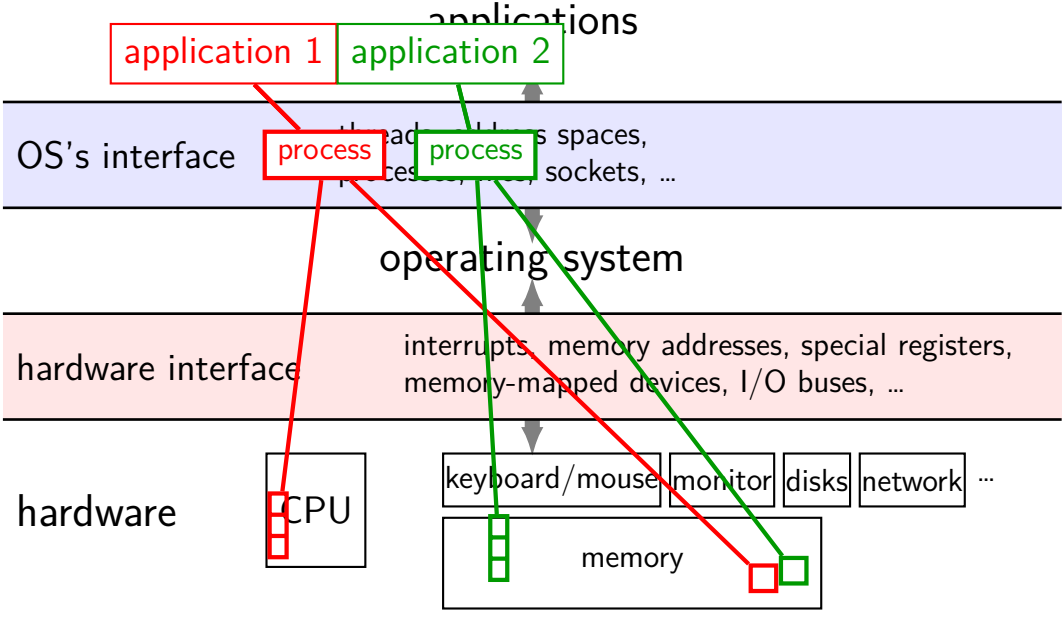
program → process → CPU and memory



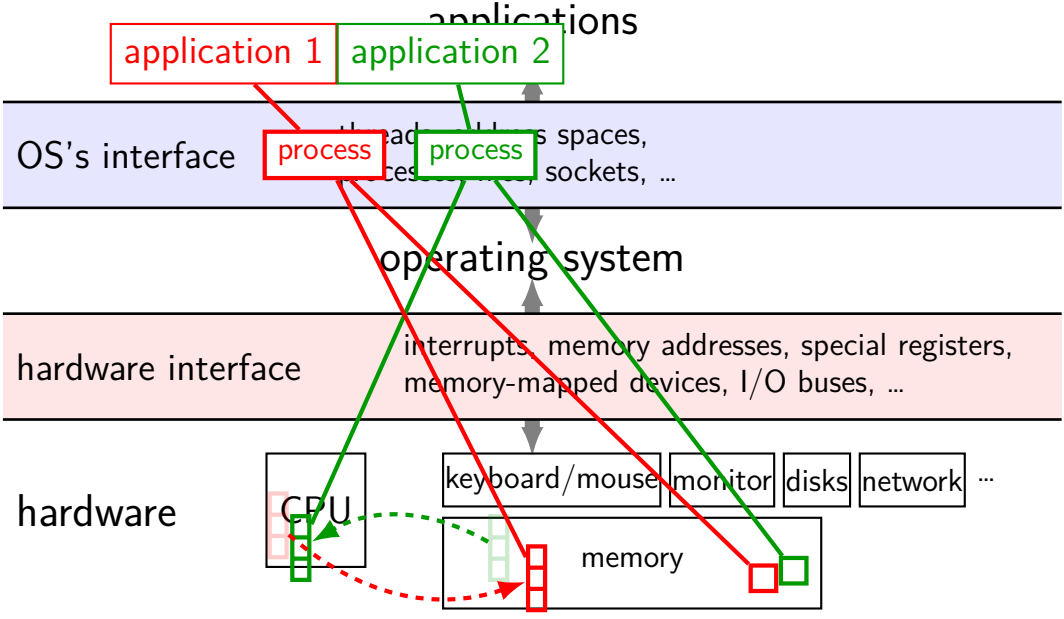
program → process → CPU and memory



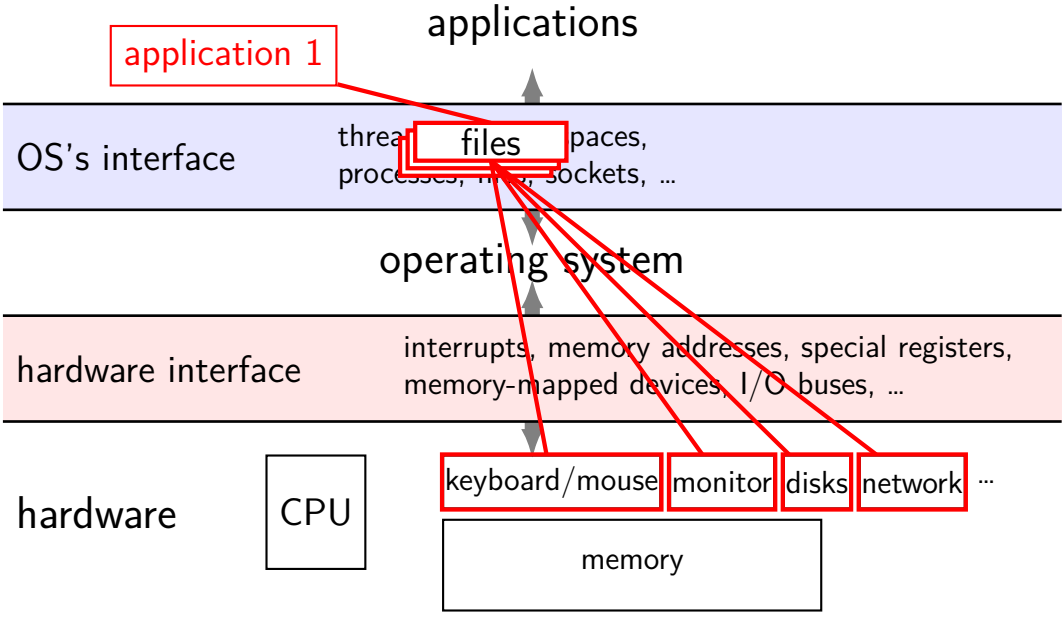
program → process → CPU and memory



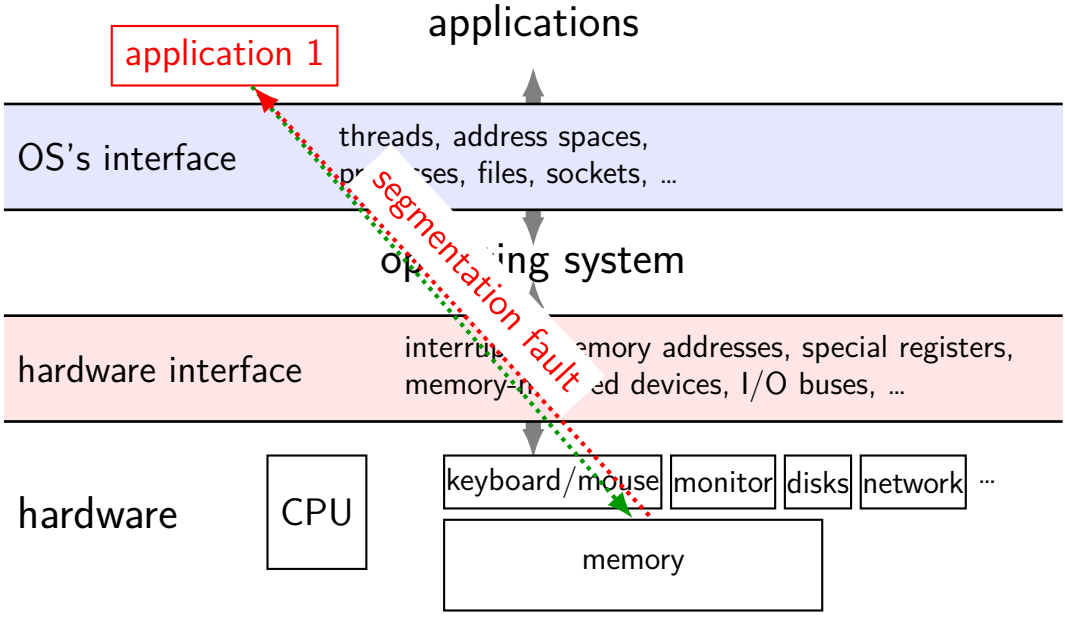
program → process → CPU and memory



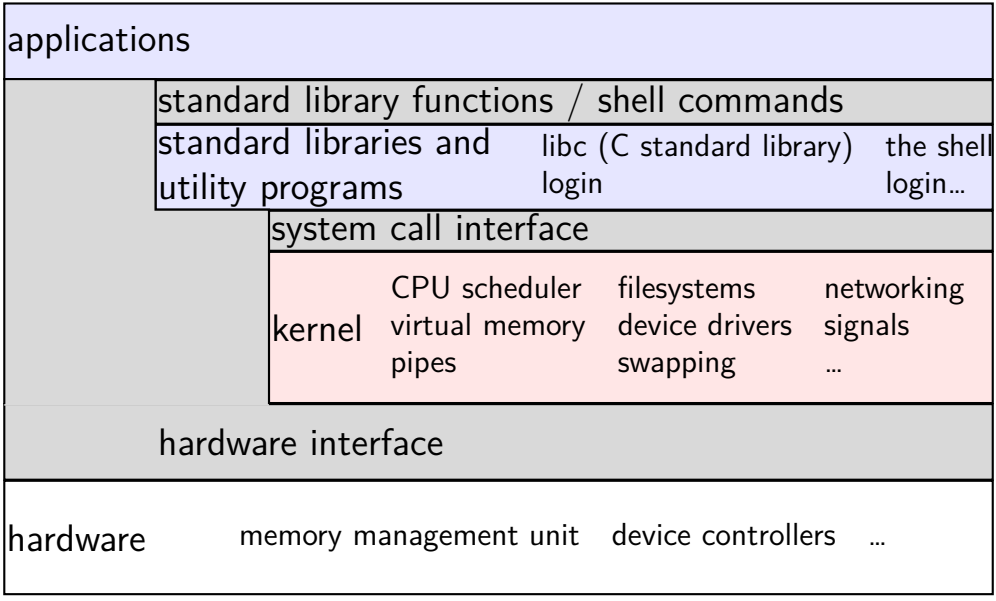
files → input/output



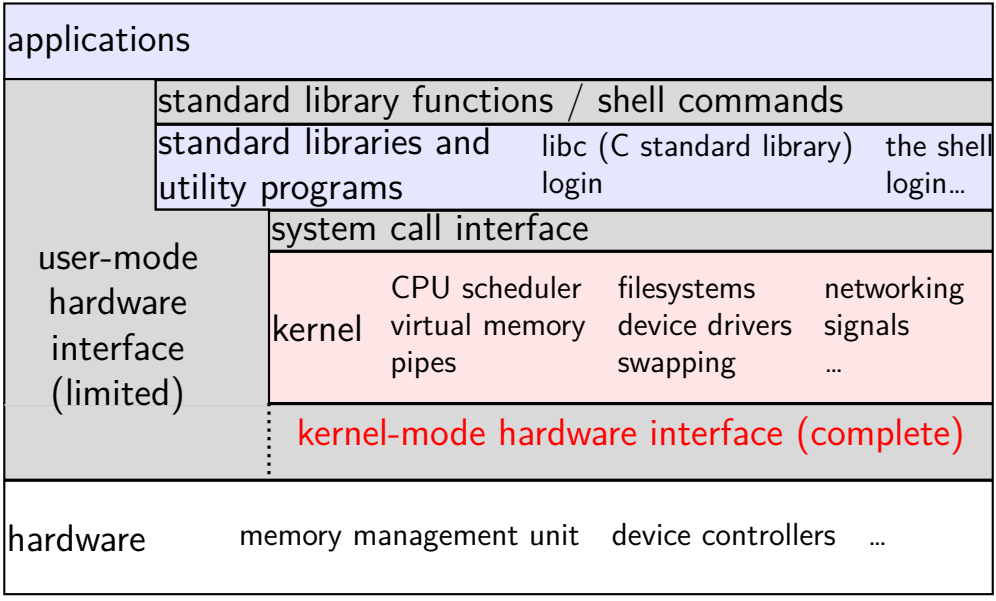
security and protection



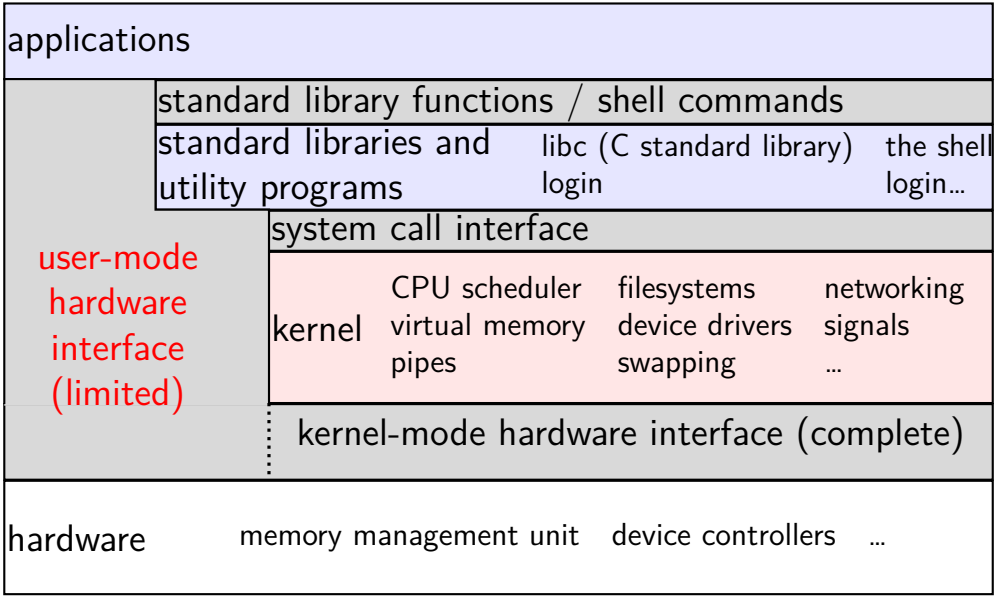
the classic Unix design



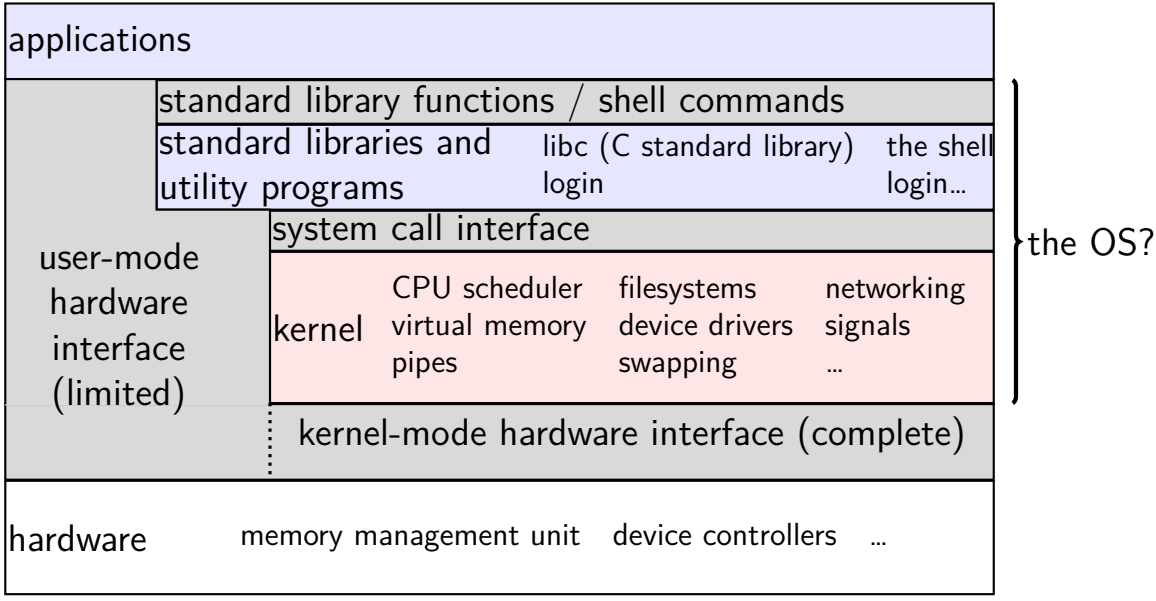
the classic Unix design



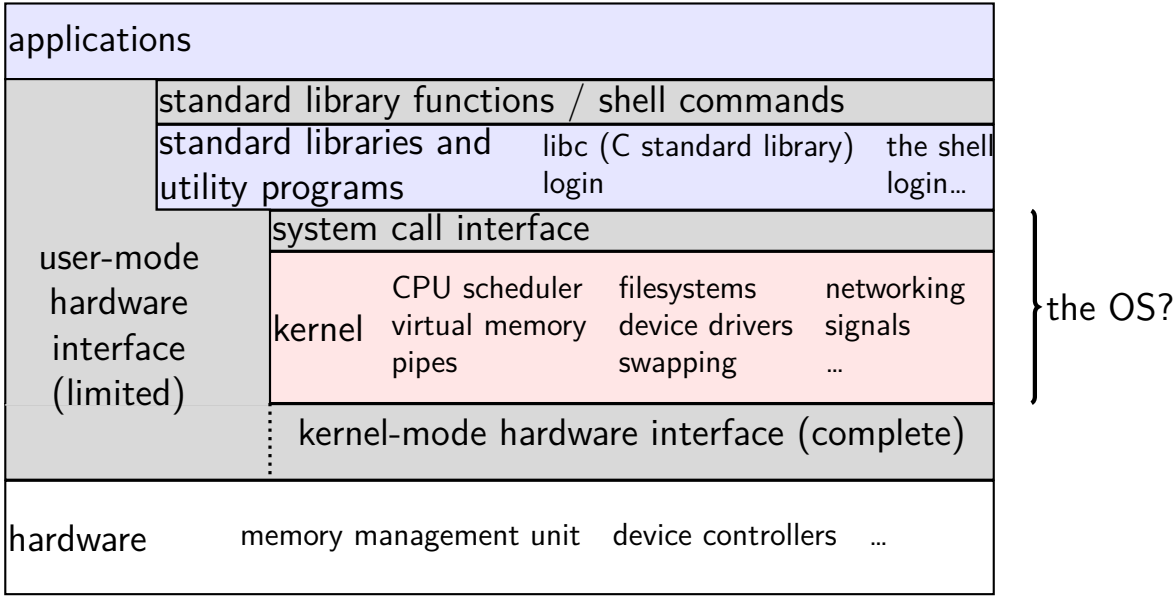
the classic Unix design



the classic Unix design



the classic Unix design



aside: is the OS the kernel?

OS = stuff that runs in kernel mode?

OS = stuff that runs in kernel mode + libraries to use it?

OS = stuff that runs in kernel mode + libraries + utility programs (e.g. shell, finder)?

OS = everything that comes with machine?

no consensus on where the line is

each piece can be replaced separately...

OS roles

Anderson-Dahlin's taxonomy of things OS's do

referee — resource sharing, protection

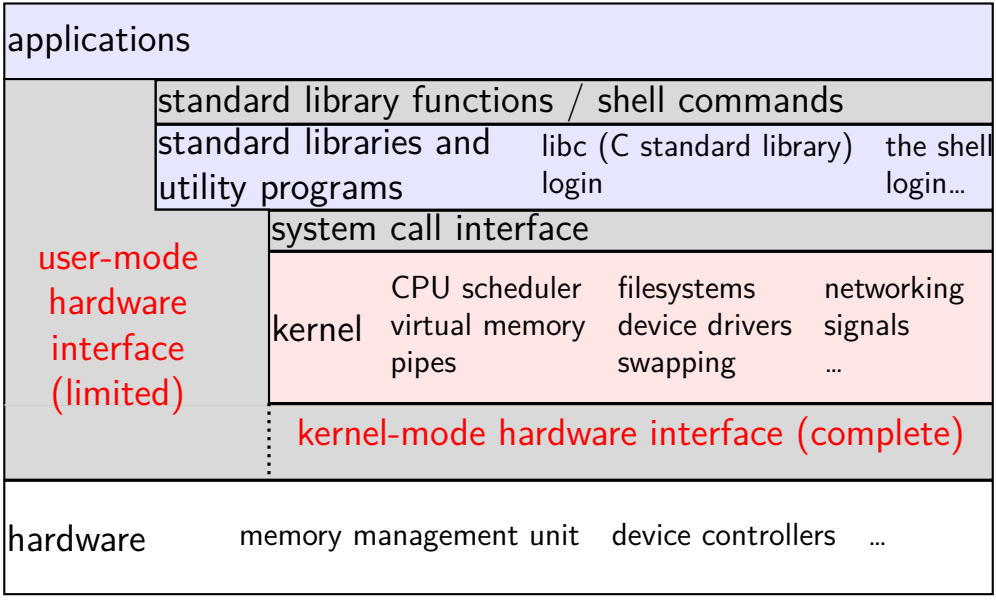
protection: program can't interfere with other programs/OS

illusionist — clean, easy abstractions

glue — common services

storage, window systems, authorization, networking, ...

the classic Unix design



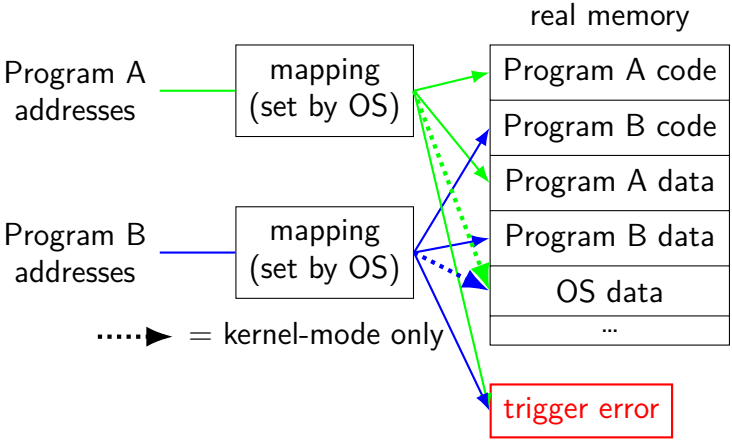
mechanism 1: dual-mode operation

processor has two modes: kernel (privileged) and user

some operations require kernel mode

OS controls what runs in kernel mode

mechanism 2: address translation



aside: alternate mechanisms

dual mode operation and address translation are common today

...so we'll talk about them a lot

not the only ways to implement operating system features

(plausibly not even the most efficient...)

problem: OS needs to respond to events

keypress happens?

program using CPU for too long?

...

problem: OS needs to respond to events

keypress happens?

program using CPU for too long?

...

hardware support for running OS: *exception*

need hardware support because CPU is running application instructions

exceptions and dual-mode operation

rule: user code always runs in user mode

rule: only OS code ever runs in kernel mode

on *exception*: changes from user mode to kernel mode

...and is only mechanism for doing so

how OS controls what runs in kernel mode

exception terminology

CS 3330 terms:

interrupt: triggered by external event

- timer, keyboard, network, ...

fault: triggered by program doing something "bad"

- invalid memory access, divide-by-zero, ...

traps: triggered by explicit program action

- system calls

aborts: something in the hardware broke

xv6 exception terms

everything is a called a trap

or sometimes an interrupt

no real distinction in *name* about kinds

real world exception terms

it's all over the place...

context clues

kernel services

allocating memory? (change address space)

reading/writing to file? (communicate with hard drive)

read input? (communicate with keyboard)

all need privileged (kernel mode) instructions!

need to run code in kernel mode

hardware mechanism: deliberate exceptions

some instructions exist to trigger exceptions

still works like normal exception

- starts executing OS-chosen handler
- ...in kernel mode

allows program requests privilieged instructions

- OS handler decides what program can request
- OS handler decides format of requests

exercise: how many exceptions?

single-core OS with processes A, B, C

already running process A

A prompts for input, then

A waits to read a keypress

while A is waiting for the keypress the OS runs B, then C

then keypress happens, and OS switches to A immediately

then A exits

exercise: how many exceptions?

exercise: how many exceptions?

A prompts for input

> likely first exception: OS writes prompt to screen, then resumes running A

A waits to read a keypress

> likely second exception: OS records that A wants a keypresss, then switches to B (since A can't keep running)

while A is waiting for the keypress the OS runs B, then C

> likely third exception: OS runs to switch from B to C

then keypress happens, and OS switches to A immediately

> likely fourth exception: OS runs to handle keypress, switches to A as part of handling it

then A exits

> likely fifth exception: OS runs to switch from A to something else

xv6

we will be using an teaching OS called "xv6"

- several (not all) programming assignments

based on Sixth Edition Unix

modified to be multicore and use 32-bit x86 (not PDP-11)

- (there's also a (more recent) RISC V version, but we cover x86 in CS 2150...)

xv6 setup/assignment

first assignment — adding two simple xv6 system calls

includes xv6 download instructions

and link to xv6 book

xv6 technical requirements

you will need a Linux environment

- we will supply one (VM on website), or get your own

some non-Linux environments have worked well for students, but we are limited in how much tech support we can do for them

- the Windows Subsystem for Linux
- OS X natively (needs a cross-compiler)

...with qemu installed

- qemu (for us) = emulator for 32-bit x86 system
- Ubuntu/Debian package qemu-system-i386

first assignment

released a week from Friday; due the following week

get compiled and xv6 working

...toolkit uses an emulator

- could run on real hardware or a standard VM, but a lot of details
- also, emulator lets you use GDB

xv6: what's included

Unix-like kernel

- very small set of syscalls
- some less featureful (e.g. exit without exit status)

userspace library

- very limited

userspace programs

- command line, ls, mkdir, echo, cat, etc.
- some self-testing programs

xv6: echo.c

```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int i;

  for(i = 1; i < argc; i++)
    printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");
  exit();
}
```

xv6: echo.c

```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int i;

  for(i = 1; i < argc; i++)
    printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");
  exit();
}
```

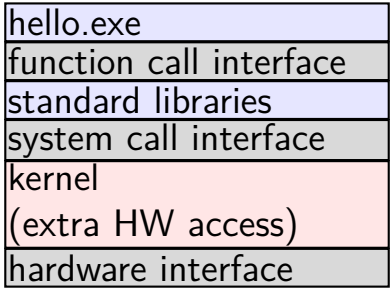
xv6: echo.c

```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int i;

  for(i = 1; i < argc; i++)
    printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");
  exit();
}
```

xv6 demo

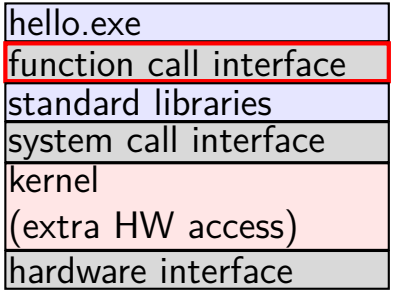
hello.c

```c
#include "user.h"
int main(void) {
  write(1, "Hello, World!", 13);
  exit();
}
```



hello.c

```c
#include "user.h"
int main(void) {
  write(1, "Hello, World!", 13);
  exit();
}
```



backup slides

common goal: hide complexity

hiding complexity

common goal: hide complexity

hiding complexity

competing applications — failures, malicious applications

- text editor shouldn't need to know if browser is running

varying hardware — diverse and changing interfaces

- different keyboard interfaces, disk interfaces, video interfaces, etc.
- applications shouldn't change

common goal: for application programmer

write once for lots of hardware

avoid reimplementing common functionality

don't worry about other programs

OS as abstraction layer