Welcome!
Partner finding

Searching for a study buddy or partner? Looking to meet a new friend? Are you taking CS, INFO, or ORIE classes?

If so, the CIS Partner Finding Social is for you! This is the PERFECT opportunity to find a partner and meet other students in your classes!

September 1
4-6 pm
Upson 142
Common systems challenges

- Emergent properties
- Propagation of effects
- Incommensurate scaling
- Trade-offs
Propagation of effects: fighting malaria

WHO sprayed villages in N. Borneo with DDT
- Wiped out mosquitos, but...
- Roaches collected DDT in tissue
- Lizard ate roaches, and became slower
- Easy target for cats
- DDT caused cats to die
- Forest rats moved into villages
- Rats carried the bacillus of the plague!
Incommensurate scaling

As the system increases in size or speed, not all components can manage the scale, and things break down

10x higher than Jack!

but also 10x wider and thicker!

About 1000x Jack’s weight — but the cross section of the Giant’s bones was only 100x Jack’s

A human thigh bone breaks at about 10x human weight

The giant would have broken his thighs every time he was taking a step!

noted in “On being the right size”
J.B.S. Haldane
Inevitable Trade-offs

- Speed vs power in processors
- Bandwidth vs computation in compression
- A pawn vs better position in chess
- Space vs time almost everywhere

...
How to Manage Complexity

Modularity
- Good modularity minimizes connections between components

Abstraction
- Separate interface from internals; separate specification from implementation

Hierarchy/Layering
- Constrain interactions so they are easier to understand
What is an OS?

An Operating System implements a virtual machine whose interface is more convenient* that the raw hardware interface.

* easier to use, simpler to code, more reliable, more secure...
Fine. But what is an OS?

A collection of software components that run directly on and manage the hardware and provide services to the user and to application programs.
OS wears many hats

- **Referee**
  - Manages shared resources: CPU, memory, disks, networks, displays, cameras...

- **Illusionist**
  - Clean, easy-to-use abstractions of physical resources
  - Look! Infinite memory! Your own private processor!

- **Glue**
  - Offers a set of common services (e.g., UI routines)
OS as Referee

Resource allocation
- Multiple concurrent tasks... who gets how much?

Isolation
- A faulty app should not disrupt other apps or OS

Communication/Coordination
- Apps need to coordinate and share state
**OS as Illusionist**

Appearance of resources not physically present

- Virtualization
  - Processor, memory, screen space, disk, network

Diagram:

- Operating System
- Hardware
- VM Interface
OS as Illusionist

Appearance of resources not physically present

Virtualization

- Processor, memory, screen space, disk, network
- The entire computer
  - Fooling the OS itself!
  - Eases debugging, portability, isolation
Appearance of resources not physically present

Atomic operations

- HW guarantees atomicity at the word level...
  - What happens during concurrent updates to complex data structures?
  - What is a computer crashes while writing a file block?
- At the hardware level, packets are lost
  - Reliable communication channels
OS as Glue

- Offers standard services to simplify app design and facilitate sharing
  - Send/Receive byte streams
  - Read/Write files
  - Pass messages
  - Share memory
  - UI
- Decouples HW and app development
We need all the help we can get...

Millions of lines of code
We need all the help we can get...

New versions are usually much larger than older versions

Millions of lines of code
We need all the help we can get…

New versions are usually much larger than older versions

Modern systems are really complex!

Millions of lines of code
The Road Ahead

- Virtualizing the CPU
- Virtualizing Memory
- Persistence

- Process Abstraction and API
- Threads and Concurrency
- Scheduling
- Virtual Memory
- Paging
- I/O Devices
- File Systems
Harmony
Your Automated Concurrency Tutor

def diner(which):
    let left, right = (which, (which % N) + 1):
    while choose({ False, True }):
        P(sema);
        lock(forks[left]);
        lock(forks[right]);
        # dine
        unlock(forks[left]);
        unlock(forks[right]);
        V(sema);
        # think

    forks = dict{ Lock() for i in {1..N } };
sema = Semaphore(N - 1);
for i in {1..N}:
    spawn diner(i);
Issues in OS Design

- **Structure**: how is the OS organized?
- **Concurrency**: how are parallel activities created and controlled?
- **Sharing**: how are resources shared?
- **Protection**: how are distrusting parties protected from each other?
- **Naming**: how are resources named by users?
- **Security**: how to authenticate, authorize, and ensure privacy?
- **Performance**: how to make it fast?
More Issues in OS Design

- **Reliability**: how do we deal with failures??
- **Portability**: how to write once, run anywhere?
- **Extensibility**: how do we add new features?
- **Communication**: how do we exchange information?
- **Scale**: what happens as demands increase?
- **Persistence**: how do we make information outlast the processes that created it?
- **Accounting**: who pays the bill and how do we control resource usage?
The Process

Our first abstraction

(Chapters 2-6)
What is a process for?

- It provides a program with the ecosystem it needs to run.
- It is how a program experiences the machine it is running on:
  - Think “The Matrix”
- When a program dreams of a computer, it dreams of a process!
From Program to Process

To make the program’s code and data come alive:

- need a CPU
- need memory
  - for data, code, stack, heap
- need registers
  - PC, SP, regular registers
- need access to I/O
  - list of open files
You’ll Never Walk Alone

Machines run (at least conceptually) multiple programs concurrently (which the OS must manage)

- how should the machine’s resources be mapped to these programs?

OS as a referee...
Machines run (at least conceptually) multiple programs concurrently (which the OS must manage).

- how should the machine’s resources be mapped to these programs?

Enter the illusionist!

- give every program the illusion of running on a private CPU
  - which appears slower than the HW machine’s
- give every program the illusion of running on a private memory
  - which may appear larger (??) than the machine’s

Virtualize the CPU

Virtualize memory
So, what does a process offer programs?

- The illusion of a dedicated CPU - that the OS must somehow “spin” based on the physical processor.
- The illusion of dedicated memory - the process’ address space - that the OS must somehow “spin” based on physical memory.
- A way to access I/O.
- A chance to live!
The OS predicament

- Multiple programs may want to run concurrently
  - OS must support multiple processes

- How should it manage the HW resources at its disposal?
  - must **multiplex**!

- could multiplex in **space**
  - What would it mean for Memory? For the CPU?

- could multiplex in **time**
  - What would it mean for Memory? For the CPU?

- How to keep track of it all?
Process Management by the OS

**Process Control Block (PCB)**

- A per-process data structure held by the OS
- Stores three types of information
  - Process identification
  - Process state (registers, PC, SP, MM Info...)
    - to seamlessly suspend and restart process
  - Process control
    - scheduling status, priority, CPU time used

**PCB**

- PC
- Stack Ptr
- Registers
- PID
- UID
- Priority
- List of open files
- Process status
- Kernel stack ptr
  ...

PC
Stack Ptr
Registers
PID
UID
Priority
List of open files
Process status
Kernel stack ptr
...