CSE 120 Discussion

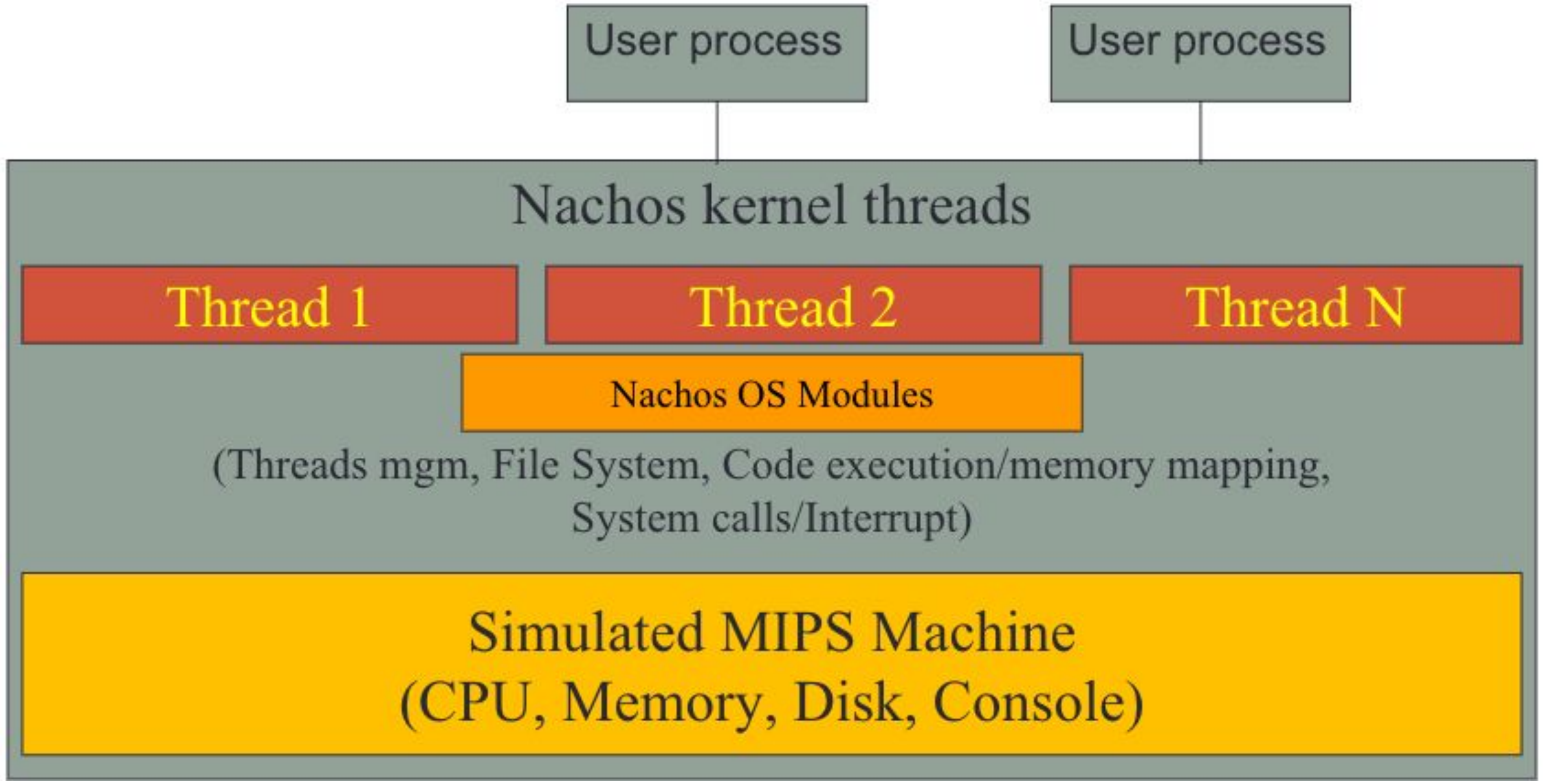
Week 2

What is Nachos?

- It is a simulated Operating System (with simulated Hardware)
- Allows us to deal/work with Operating Systems concepts in a more controlled environment
  - Threads (Proj 1)
  - System Calls (Proj 2)
  - Virtual Memory (Proj 3)

Why Nachos?

- IMO it's kind of similar to real life Software Engineering
  - A large code-base that you start off with no idea about.
- Implementing and working with Operating Systems concepts is a good way to learn about it
  - Kind of similar to Data Structures, you learned about the concepts and then implemented them.


User process
User process
Nachos kernel threads
Thread 1
Thread 2
Thread N
Nachos OS Modules
(Threads mgm, File System, Code execution/memory mapping, System calls/Interrupt)
Simulated MIPS Machine
(CPU, Memory, Disk, Console)

Nachos Packages

- Entirely written in Java
- Broken into Java packages:

Package Name	Purpose	Use in Project #
ag	autograder	-
machine	Basic machine specification	-
threads	Threads Management	0,1
userprog	Binary Code Execution and System Calls	2
vm	Virtual Memory	3
network, security, test, bin	as their name suggests	-

Booting Nachos

- When you run Nachos, it starts in nachos.machine. Machine.main
- Machine.main initializes devices - interrupt controller, timer, MIPS processor, console, file system
- Passes control to the autograder.
- AutoGrader will create a kernel and start it (this starts the OS)

More Info

- nachos.machine.Machine

- Kicks off the system, and provides access to various hardware devices:

  - Machine.interrupt() – Interrupt Management

  - Machine.timer() - Timer

  - Machine.console() – Serial Console

  - Machine.networkLink() – For network communication

The Kernel

- Abstract class nachos.machine.Kernel
- ThreadedKernel inherits from the Kernel

- Important methods
  - initialize() initializes the kernel, duh!
  - selfTest() performs test (not used by ag)
  - run() runs any user code (none for Project 0 and 1)
  - terminate() Game over. Never returns.

- Each Phase will have its own Kernel subclass

Threading

- Happens in package nachos.threads
- All Nachos threads are instances of nachos.thread.KThread (or subclass)
- KThread has status
  - New, Ready, Running, Blocked, Finished
- Every KThread also has a nachos.machine.TCB
- Internally implemented by Java threads

Running threads

- Create a java.lang.Runnable(), make a Kthread, and call fork().
- Example:

```
class Sprinter implements Runnable {
   public void run() {
      // run real fast
  }
}

Sprinter s = new Sprinter();
new KThread(s).fork();
```

Scheduler

- Some subclass of nachos.machine.Scheduler
- Creates ThreadQueue objects which decide what thread to run next.
- Defaults to RoundRobinScheduler
- Specified in Nachos configuration file

Dealing with Large Codebases

1. Don't try to understand every line of code
   a. Picking the right level of abstraction is important!
2. Develop an ability on how to design "experiments" to see how the code works.
   a. We'll go through an example of this.
   b. This can really help you understand what's going on.
3. Do yourself a favor and use an IDE.
   a. Being able to navigate quickly through files/modules can be extremely helpful.
   b. Breakpoints are nice too.
   c. Syntax highlighting is also nice.

Biggest Advice

Starting early really **really** *really* really makes a difference.

Even if it just means reading the README.

Lets practice working in Nachos

You remember project 0 (“AWESOME” thread)?

Let’s go through undoing the changes to proj0 and go through how nachos runs.

*grep -rnw * -e 'TEXTYOUWANTTOFIND'*

1. How does Nachos start running?
   a. Machine.main > Autograder -> Kernel -> ThreadedKernel -> KThread
   b. Why ThreadedKernel? Why not UserKernel?
      i. nachos.conf
2. Why/How does PingTest run?
3. Demo

Project 1 - Threads

Implementing Join. What is join though?

- Lets assume there are 2 threads.
- Let’s call them Thread A and Thread B.
- What happens if Thread A executes the line “B.join()”?
- It means Thread A will wait until Thread B has finished.
- So how do we implement that?

High Level Idea

1. If Thread A calls “B.join()”, we want a way to BLOCK thread A until thread B has finished.
2. Once Thread B finishes, we want to unblock Thread A

Thats it!

Pieces of the Puzzle

Question 1: How do we block a process?

Answer 1: Let’s take a look at KThread.sleep()

Question 2: How do we unblock a process?

Answer 2: Let’s take a look at ready()

Question 3: How do I know when a process is finished?

Answer 3: KThread.finish()

Pieces of the Puzzle

Question 4: If A calls B.join. What is the currentThread? What is ‘this’?

Answer: ‘this’ = B, currentThread = A

Question 5: What if A calls B.join and B is a thread that has already finished?

Some Other Details

Whats the deal with "boolean intStatus = Machine.interrupt().disable();" and "Machine.interrupt().restore(intStatus);"? Let's take a look at sleep()

Also, keep in mind that for any given Thread X. "X.join()" can be executed at most ONE time.

We have to associate a thread A with a thread B where A executed "B.join", so that when B finishes, we can unblock the thread A. How do we do that?