CS 162 Nachos Tutorial

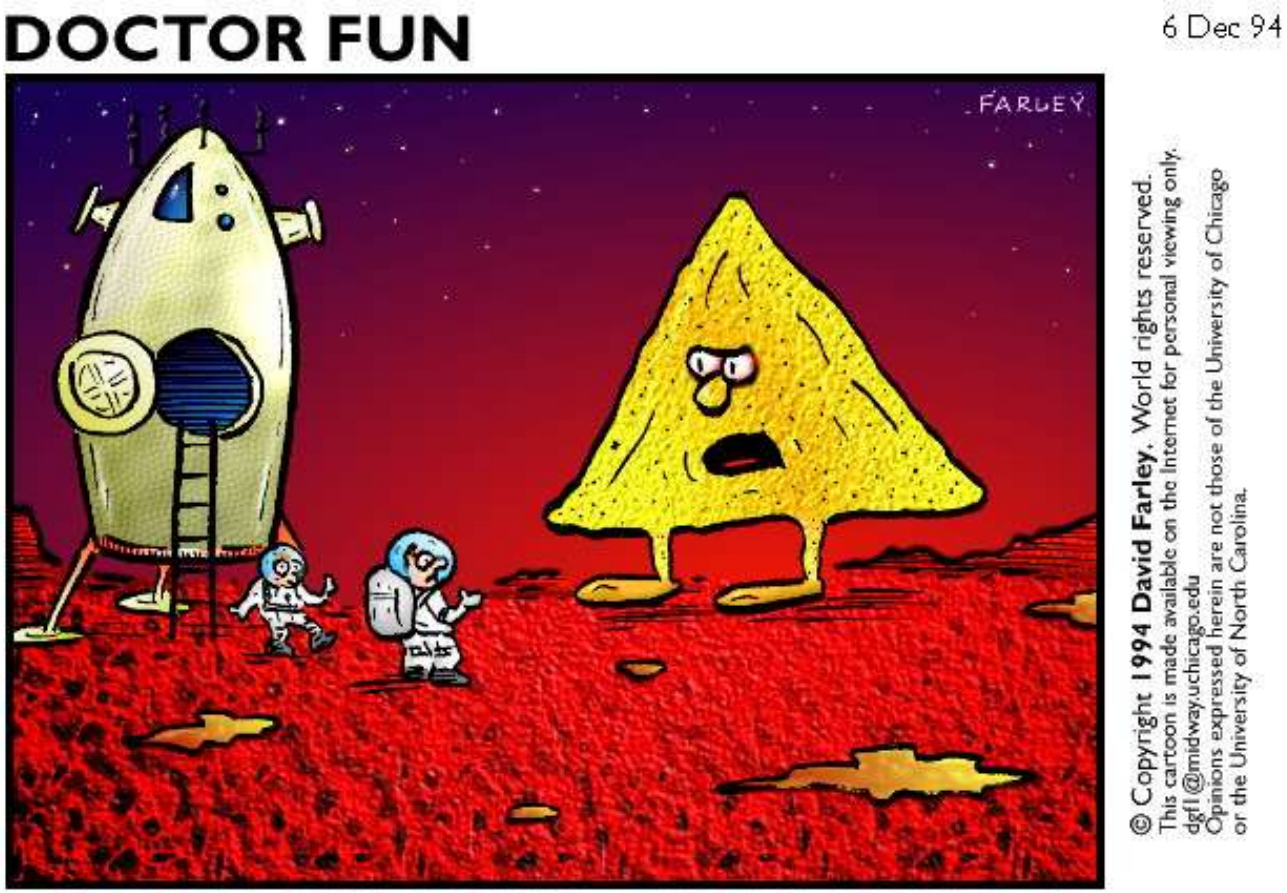


Image courtesy of Thomas Andersen: http://www.cs.washington.edu/homes/tom/nachos/

Outline

- What is Nachos?
  - Capabilities, purpose, history
- How does it work?
- What am I supposed to do?
  - The 4 phases
- How do I get started?

What is Nachos?

- An instructional operating system
- Includes many facets of a real OS:
  - Threads
  - Interrupts
  - Virtual Memory
  - I/O driven by interrupts
- You can (and will) modify and extend it

What else is Nachos?

- Nachos also contains some hardware simulation.
  - MIPS processor
    - Can handle MIPS code in standard COFF, except for floating point instructions
    - You can (and will) write code in C, compile it to MIPS and run it on Nachos.
  - Console
  - Network interface
  - Timer

Why Nachos?

- What better way to learn how an OS works than by building one?
- Much easier and more reasonable to build a simulated OS (in Java)
- Skeleton code allows us to work on, replace, or upgrade one piece at a time.

Why Java?

- Java much simpler than C++
- Java is type-safe – can't write off the end of an array, easier to debug
- Much easier and more reasonable to machine grade a Java project
- More portable

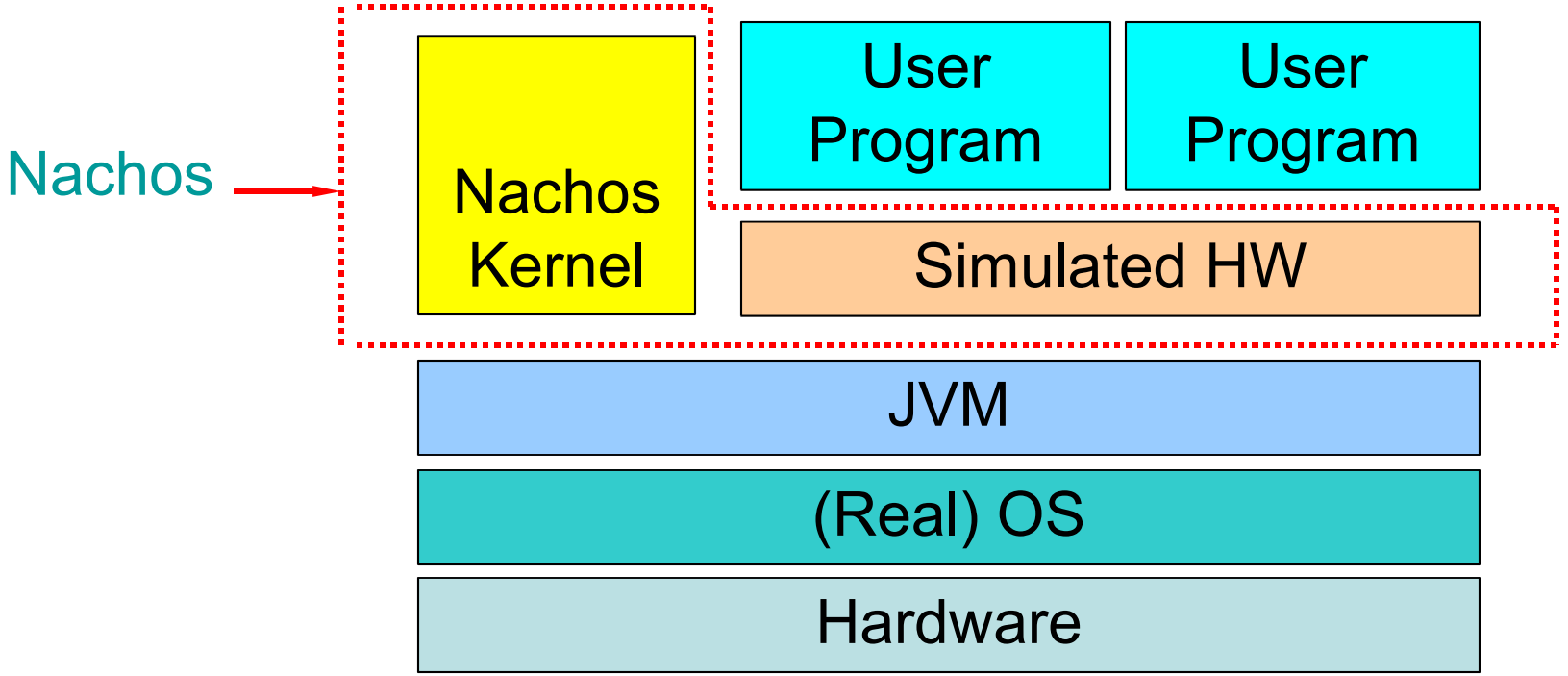
History of Nachos

- Originally created here at Berkeley in 1992 in C++ (and a little assembly)
- By Wayne A. Christopher, Steven J. Procter, and Thomas E. Anderson
- Used at many universities
- Rewritten in Java by Daniel Hettena
  - Now simpler, easier to grade, type-safe, portable, and more students now know Java.

How does Nachos work?

- Entirely written in Java
- Broken into Java packages:
  - nachos.ag (autograder classes)
  - nachos.machine (most of the action)
  - nachos.network (Phase 4)
  - nachos.security (tracks priviledge)
  - nachos.threads (Phase 1)
  - nachos.userprog (Phase 2)
  - nachos.vm (Phase 3)

Nachos Architecture



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)

The Machine!

- nachos.machine.Machine
- Kicks off the system, and provides access to various hardware devices:
  - Machine.interrupt()
  - Machine.timer()
  - Machine.console()
  - Machine.networkLink()

Interrupt Controller

- Kicks off hardware interrupts
- nachos.machine.Interrupt class maintains an event queue, clock
- Clock ticks under two conditions:
  - One tick for executing a MIPS instruction
  - Ten ticks for re-enabling interrupts
- After any tick, Interrupt checks for pending interrupts, and runs them.
- Calls device event handler, not software interrupt handler

Interrupt Controller (cont.)

- Important methods, accessible to other hardware simulation devices:
  - schedule() takes a time, handler
  - tick() takes a boolean (1 or 10 ticks)
  - checkIfDue() invokes due interrupts
  - enable()
  - disable()
- All hardware devices depend on interrupts - they don’t get threads.

Timer

- nachos.machine.Timer
- Hardware device causes interrupts about every 500 ticks (not exact)
- Important methods:
  - getTime() tells many ticks so far
  - setInterruptHandler() tells the timer what to do when it goes off
- Provides preemption

Serial Console

- Java interface nachos.machine.SerialConsole
- Contains methods:
  - readByte() returns one byte (or -1) and waits to interrupt when it has more
  - writeByte() takes one byte and waits to interrupt when its ready for more
  - setInterruptHandlers() tells the console who to call when it receives data or finishes sending data
- Normally implemented by nachos.machine.StandardConsole, hooked up to stdin and stdout
  - Schedules read event every `Stats.ConsoleTime` ticks to polls stdin & invokes interrupt handler

Other Hardware Devices

- Disk
  - Didn't make the jump to Java from C++, we don't use it for our Nachos assignments
- Network Link
  - Similar to console, but packet based.
  - Used for Phase 4.
  - You should be able to figure it out by then.

The Kernel

- Abstract class nachos.machine.Kernel
- Important methods
  - initialize() initializes the kernel, duh!
  - selfTest() performs test (not used by ag)
  - run() runs any user code (none for 1st phase)
  - 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

Nachos Configuration

- nachos.conf file lets you specify many options
  - which classes to use for Kernel, Scheduler
  - whether to be able to run user progs
  - etc.
- Different one for each project.

Creating the First Thread

- ThreadedKernel.initialize

```
public void initialize(String[]      args) {
    ......
    // start threading
    new KThread(null);
    ......
}
```

- What does KThread perform?
- What thread does it create?

Advanced Topics

- The simulated MIPS processor
- Address translation
- User level process
- Syscalls and exception handling
- You will get to know more when we get there

How are we using it?

- Four Nachos assignments - “Phases”
- Phase 1 - Threading
- Phase 2 - Multiprogramming
- Phase 3 - Caching and Virtual Memory
- Phase 4 - Networks and Distributed Systems

Nachos Projects

Extend & Embrace Nachos

- Add features to Nachos (kernel code)
  - Threading
  - File system calls
- Implement user programs (in C)

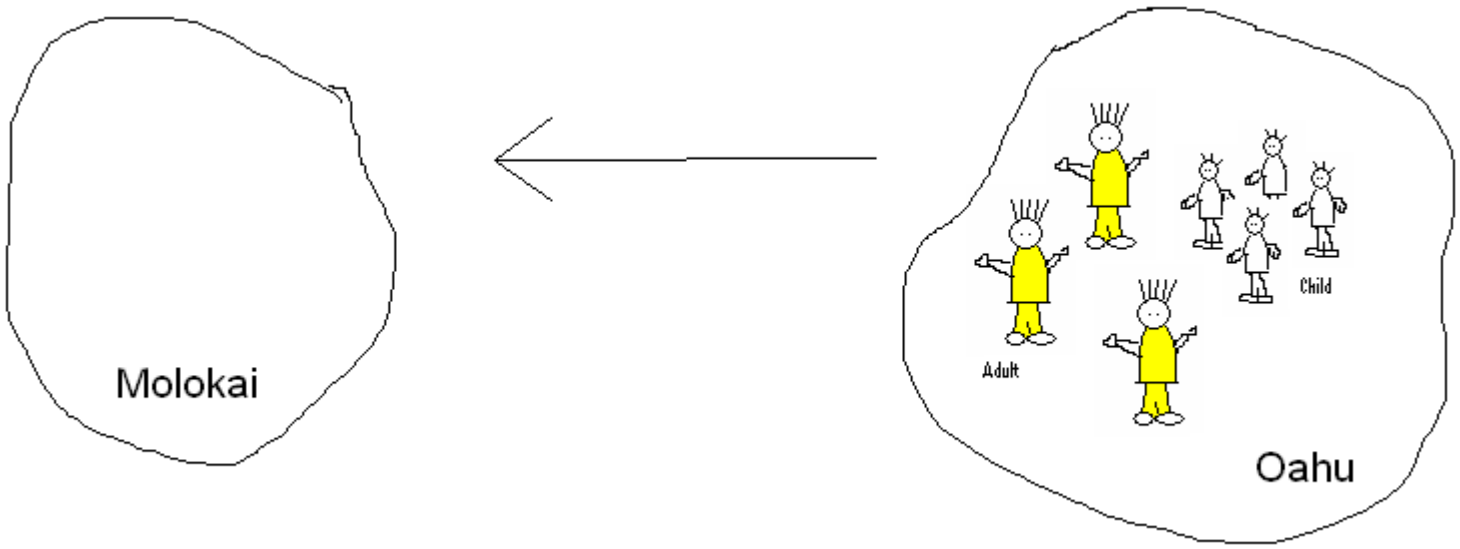
4 Phases

- Phase 1: Thread system (due 2005-02-16)
- Phase 2: Multiprogramming (due 2005-03-07)
- Phase 3: Caching & Virtual Memory (due 2005-03-30)
- Phase 4: Networks & Distributed Systems (due 2005-04-21)

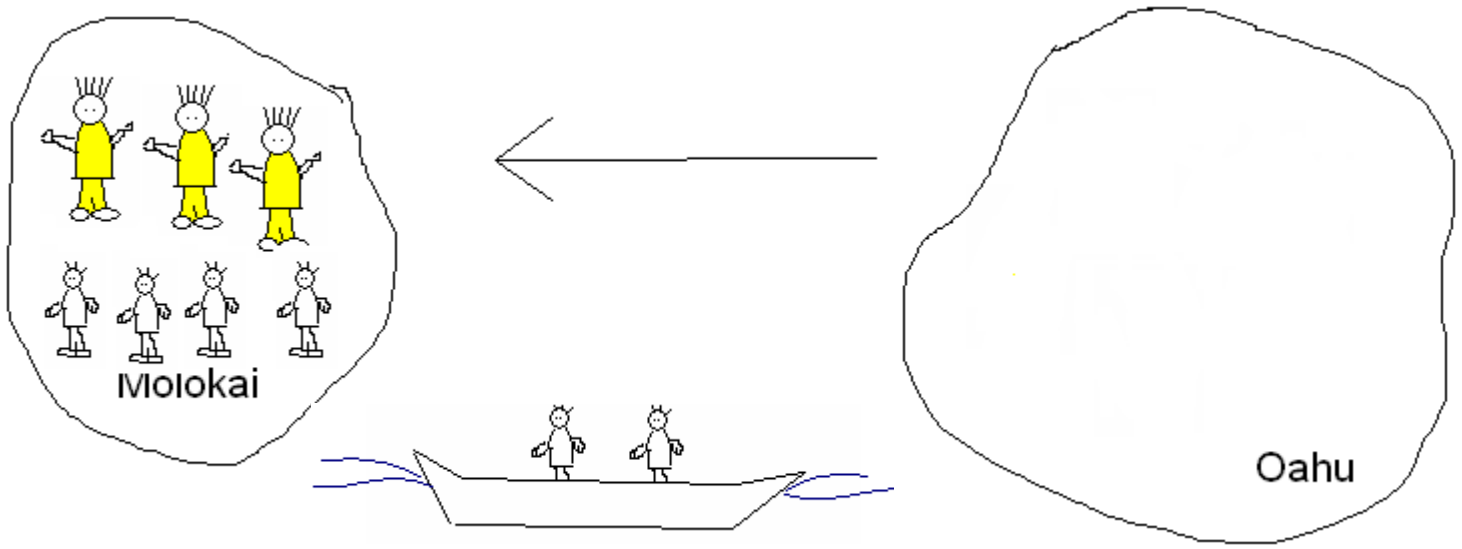
Phase 1: Threading

- 5%: KThread.join
- 5%: Condition Variables (more efficiently)
- 10%: Alarm
- 20%: Communicator
- 35%: PriorityScheduler
- 25%: Rowing Hawaiian kids

Row boat synchronization

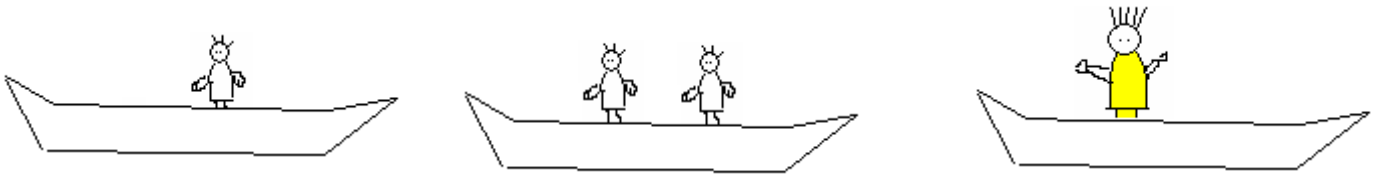


- Get Adults and Children from Oahu to Molokai

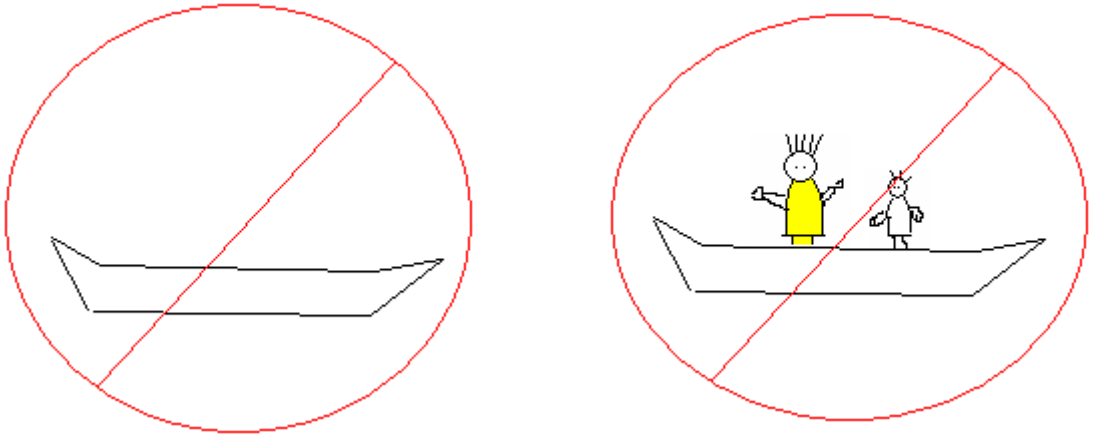


Constraints

- 1 boat
- Boat fits 1 child, or 2 children, or 1 adult



- Pilot required



Phase 2: Multiprogramming

- 30%: File system calls
  - `creat`, `open`, `read`, `write`, `close`, `unlink`
- 25%: Multiprogramming
  - Multiple users/programs at once
- 30%: System calls
  - `exec`, `join`, `exit`
- 15%: LotteryScheduler

Phase 3:Caching & VM

- 30%: Implement TLB, Inverted page table
- 40%: Paged virtual memory
  - Fit large program(s) in memory
- 30%: Lazy loading
  - Don’t load parts of program until needed

Phase 4: Networking

- 75%: Networking syscalls
  - `Connect`, `accept`
- 25%: Chat program
  - Like IRC

- Workload (grading) percentages given
- Divide work fairly
- Projects depend on each other
  - E.g. LotteryScheduler in next project depends on PriorityScheduler

How to get started

- Go to class web page
- Download and install nachos package
- Read the README, make sure you can make proj1 OK
- The first phase is posted – initial design doc due in a week

Advice

- One step at a time. Get a little bit working. Then a little more. Then a little more, etc.
- Find a good tool, including a debugger, and use it. One choice - Eclipse.

For More Information

- README file in the installation has lots of good stuff
- See the Class Web Page for intros, background, and the code itself.
- Read the code! You can see exactly what is going on.

Subversion/CVS

- Allows multiple people to work on code concurrently

Student 1:

```
% emacs threads.java
```

Student 2:

```
% emacs threads.java
% svn commit threads.java
# threads.java committed
```

```
% svn commit threads.java
# ERROR: Student2 already modified threads.java
# Your copy is out of date

% svn update
# svn patches (merges changes into) threads.java

% svn commit threads.java
# threads.java committed
```

Subversion

- Reference: http://svnbook.red-bean.com/
- Windows: http://tortoisesvn.tigris.org/
- Eclipse Plugin available