Kernel Assignments

CS 256/456

Dept. of Computer Science, University of Rochester

Deadlock Problem and Characterization

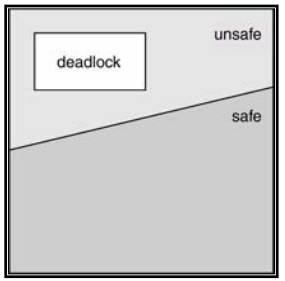
- Definition:
  - A set of blocked processes each holding some resources and waiting to acquire a resource held by another process in the set.
  - None of the processes can proceed or back-off (release resources it owns)
- Characteristics:
  - Mutual exclusion, hold&wait, no preemption, circular wait
- How to systematically deal with deadlock?
  - Do nothing
  - Make sure it never happens in the first place
  - Detect its occurrence and recover from it

Deadlock Avoidance

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a *safe state*.
- System is in safe state if there exists a safe sequence of all processes.
- Sequence $<P_1, P_2, \ldots, P_n>$ is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j<i$.
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Deadlock Avoidance (cont.)

- If a system is in safe state ⇒ no deadlocks.
- If a system is in unsafe state ⇒ possibility of deadlock.
- Deadlock avoidance
  - dynamically examines the resource-allocation state
  - ensure that a system will never enter an unsafe state.

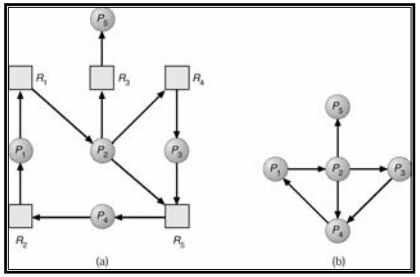


Banker's Algorithm

- Each process must a priori claim the maximum set of resources that might be needed in its execution.
- Safety check
  - repeat
    - pick any process that can finish with existing available resources; finish it and release all its resources
    - until no such process exists
  - all finished → safe; otherwise → unsafe.
- When a resource request is made, the process must wait if:
  - no enough available resource this request
  - granting of such request would result in a unsafe system state

Another Method: Detect Deadlocks and Recover

- Maintain *wait-for* graph
  - Nodes are processes.
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j.
- Periodically search for a cycle in the graph.



Resource-Allocation Graph Corresponding wait-for graph

Recovery from Deadlock

- Recovery through preemption
  - take a resource from some other process
  - depends on nature of the resource
- Recovery through rollback
  - checkpoint a process state periodically
  - rollback a process to its checkpoint state if it is found deadlocked
- Recovery through killing processes
  - kill one or more of the processes in the deadlock cycle
  - the other processes get its resources
- In which order should we choose process to kill?

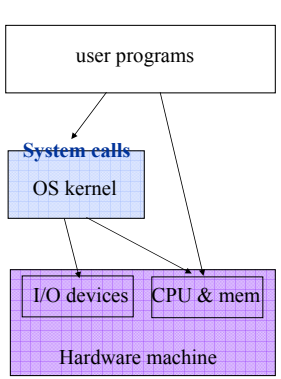
Introduction to Nachos: Our Objective

- Provide a basic understanding of the structure of Nachos
  - not a replacement of your own practice and learning
- Learn design rationale for Nachos
  - you are welcome to suggest improvements to the assignments
- Jump start you on Programming Assignment #3

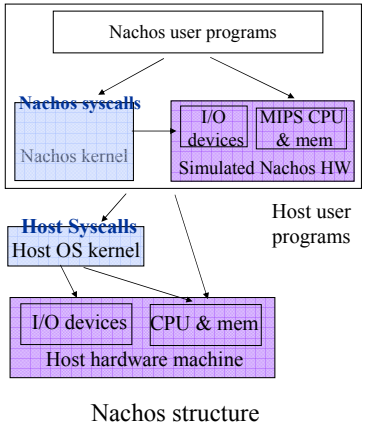
Motivation for Nachos

- Motivation: need good projects for teaching operating systems
- Assignments so far are all at user level
  - OK with understanding OS concepts
  - Not real OS development experience
- Develop an OS from scratch on a real hardware (e.g., Intel IA32)
  - Real hardware is too complex to work with
  - Very hard to support even the most basic user programs
- Making enhancement/adjustment of an existing OS (e.g., Linux)
  - Modern OSes are too complex to understand
  - Not complete view about OS development
- Nachos is a toolkit for teaching operating systems; not an operating system itself

Overall Nachos Structure

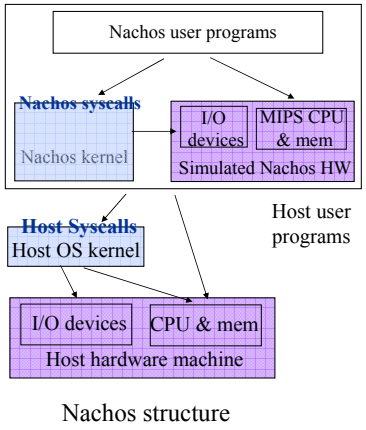


Normal computer structure



Nachos structure

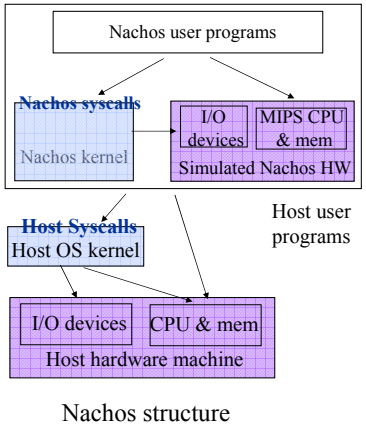
Nachos Kernel and User Programs



Nachos structure

- The Nachos kernel runs directly as a user program on the host machine.
- Kernel code written in C++.
- User programs run on the Nachos kernel and the simulated hardware (with a MIPS CPU).
- User programs access simulated I/O devices through Nachos syscalls.
- User programs written in a stripped-down C, compiled into MIPS executables.

Simulated Nachos Hardware



Nachos structure

- Simulated MIPS CPU
  - A MIPS instruction interpreter
- Simulated memory
  - Basically an array of words
- Simulated I/O devices
  - Console terminal
  - Disk
  - Timer
  - *Network interface*

Nachos as An Instructional Tool

- The Nachos toolkit contains:
  - The simulated hardware (MIPS CPU, memory, I/O devices)
  - A cross-complier/linker that compiles your C user programs into Nachos executables
    - instructions are in MIPS
    - the segment layout and header format can be recognized by the Nachos kernel
  - A skeleton OS kernel supporting
    - a single system call: halt
    - limited synchronization primitives
    - running a single user program at a time
    - no inter-process communication
- Your job is to augment the OS step by step
  - don't touch the simulated hardware
  - shouldn't need to touch the compiler

The Most Common Confusion

- Nachos kernel and Nachos user programs do not run on the same machine (real or virtual)
- Implications:
  - they don't run on the same CPU (real or virtual)
  - they don't share the same memory (real or virtual)
- This is a big weakness
  - why not let Nachos kernel run on the simulated hardware?
  - the projects become too challenging
  - a pain to develop code on the simulated hardware, e.g., can't run gdb

Nachos Assignments

Start from a given skeleton OS, filling the missing pieces step by step

- Assignment #3: threads and synchronization
- Assignment #4: supporting multiple user programs
- Assignment #5: virtual memory
- Assignment #6: file system and disk scheduling

Learning about Nachos

- You cannot learn all about software systems from textbooks
  - read the source code for systems that other people have written
- For Nachos:
  - it is not sufficient to just read the textbook, lectures, and information on the assignment Web pages
  - read over the Nachos source code
  - try to understand where the various pieces of the system live, and how they fit together
- It will take a while to develop an understanding. Start early and be patient!

Traversing the Nachos Files

- starting from the Makefiles
- threads/
  - support threads and synchronization
  - the threads support is fully functional, though some of the synchronization primitives have not been implemented.
- userprog/
  - support the loading and execution of user programs
  - basic memory management
- vm/
  - virtual memory subsystem

Traversing the Nachos Files

- test/
  - some simple Nachos user programs
  - Makefile for compiling these programs and converting them to NOFF
- machine/
  - support machine simulation.
  - You need to read some header files to know how to access the simulated machine.
  - It might also be instructive to look at the implementation of the machine simulation.
  - But **you shouldn't have to modify** anything here.

An Alternative Assignment Track

- Nachos-based assignments:
  - Nachos is specifically designed for OS course projects.
  - The kernels you develop on it are not "real".
- Xen-based assignments
  - Xen, on the other hand, is a "true" virtual machine.
  - You will be working with slightly modified Linux on it.
- Why not choose Xen assignments?
  - More realistic, but much more challenging (the Nachos assignments will be challenging enough).
  - Not able to develop a whole-system understanding as you might for Nachos assignments.
  - We are the first to introduce the Xen assignments so there might be problems and you need to be very patient with us.

Xen: An "True" Virtual Machine

- Real operating systems (with slight modifications) can run on Xen.
  - we use a modified Linux 2.6.10
- For Xen assignments:
  - you will spend most of your time reading, understanding the Linux code.
  - and you will also spend a lot of time on Linux administration.
  - even debugging becomes very challenging.
  - you will get help from us, but you are expected to figure out lots of things on your own.

Xen Assignments

Make adjustment to a fully functional kernel

- Assignment #3: write a system call
- Assignment #4: threads and synchronization
- Assignment #5: CPU scheduling
- Assignment #6: memory management

What Are Other Schools Doing?

- Most do Nachos-based assignments
  - Berkeley, Washington, Duke, …
- Improved Nachos where the kernel runs (along with user programs) on the instructional virtual machine
  - Harvard
- Some have assignments through modifying the Linux kernel on real virtual machines (mostly VMware)
  - Columbia, Washington
- The most challenging is to develop a workable OS (not as complete as Linux) from scratch
  - Princeton