System Programming

An Introduction

Uni Karlsruhe - Systems Chair / IBDS
Christian Ceelen

C/S: Education so far

- Computer Science #1-#4
  - Algorithms, Theory
  - Development of Algorithms
- Computer Engineering #1+2
  - Hardware Basics and Development
  - Processor Basics and Programming
- Mathematics

Congratulation for comming this far!

Programming Experience

- High-Level Language (Java)
- Functional Language (Gopher)
- Machine-Level Language (Assembler)

- Who had no bugs?
- Who had lots of bugs?

Complexity in LoC

O(10):	Shell-Scripts
O(100):	Info 1-2-3
O(1000):	Info 4
O(10^6):	Emacs (20.2)
O(10^7):	Linux Kernel (2.4.x), GCC(2.95)
O(10^8):	Linux Kernel (2.6.x)

And complexity is still growing

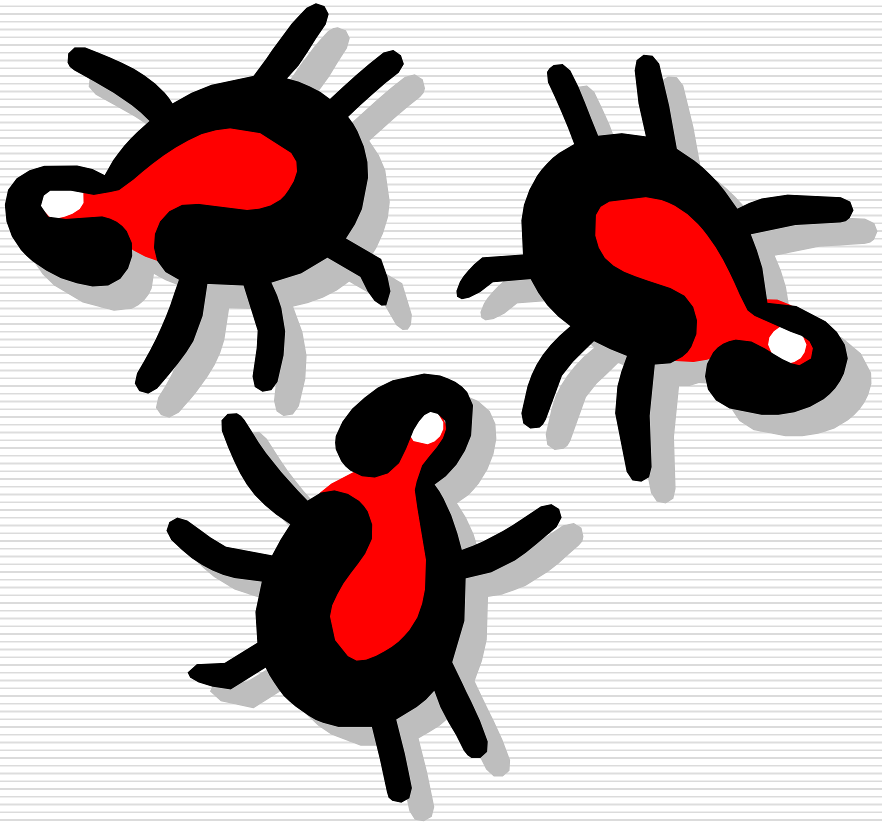
Complexity

- What are you going to do?
  - Avoid it?
  - Fight it?
  - Life with it?
  - Manage it?
- But how?
- Bug free?

Sad but true

Destilled Realizations of a System Programmer from Murphy‘s Law

Everything is about Bugs



C/S: Everything is about Bugs

- Algorithm Engineering
  - Develop buggy algorithms
  - Make your bugs fast
  - Avoid obvious bugs
- Formal Systems
  - Define bugs
  - Prove abscence of bugs
- Software Engineering
  - Organize your bugs
  - Test against Spec.
  - Track bugs during implementation
- System Architecture
  - Cope with bugs
  - Design with bugs in mind
  - Avoid Bugs in Design

C/S: Living with Bugs

- Bugs [all]
- Design [Sys Arch]
  - Architecture
  - Performance
  - Scaleability
- Development [SWT]
  - Specification [Formal Systems]
  - Documentation
  - Implementation

C/S: Living with Bugs

- Programming Style [SWT]
  - Testing
  - Error handling
  - Maintaince
  - Development cycle (version control)
- Customor focus [SysArch, SWT]
  - Useability
  - Stability

Why are we here?

"Programming is understanding"

-Kristen Nygaard

Programming Assignments

- Provide introduction into „System Programming“
- Provide deeper understanding of **Operating Systems** and **Systems Programming** through practical experience.

- Approach: Participate in the design and implementation of a **simple operating system**.

Prerequisits

- Programming with C/C++ (and ASM)
  - see paper at our homepag[illegible]
  - deeper knowledge o[illegible]++/gasm/ld not necessary
- Use differe[illegible]gramming tools
  - Dee[illegible]owledge of „make“ is not [illegible]sary
  - Versioning system
    - See lecture SWT next week: CVS & Co.

You should work this out yourself!

Experience You Gain Here

- Debugging a System
  - Tools
  - Techniques
- Narrowing complexity gap between your experience and production system
- Work in a team
- Programming with the whole system in mind

Participation

- subscribe mailinglist at
  - http://lists.ira.uka.de/mailman/listinfo/nachos
  - Send mails to: nachos@ira.uka.de
- form a group with 2-3 persons
- registration will close with the 3rd assignment (1st of December)
  - Further announcements in the lecture and on the mailing list

Help! I need somebody! Help!

- Ask at nachos@ira.uka.de
- Personal Contact (R154, R155)
  - Daniel Kirchner kirchner@ira.uka.de
  - Sebastian Biemüller biemueller@ira.uka.de
  - Christian Ceelen ceelen@ira.uka.de
- Consultation Time (SB, DK)
  - Fr. 15:45

NachOS

An Educational Operating System

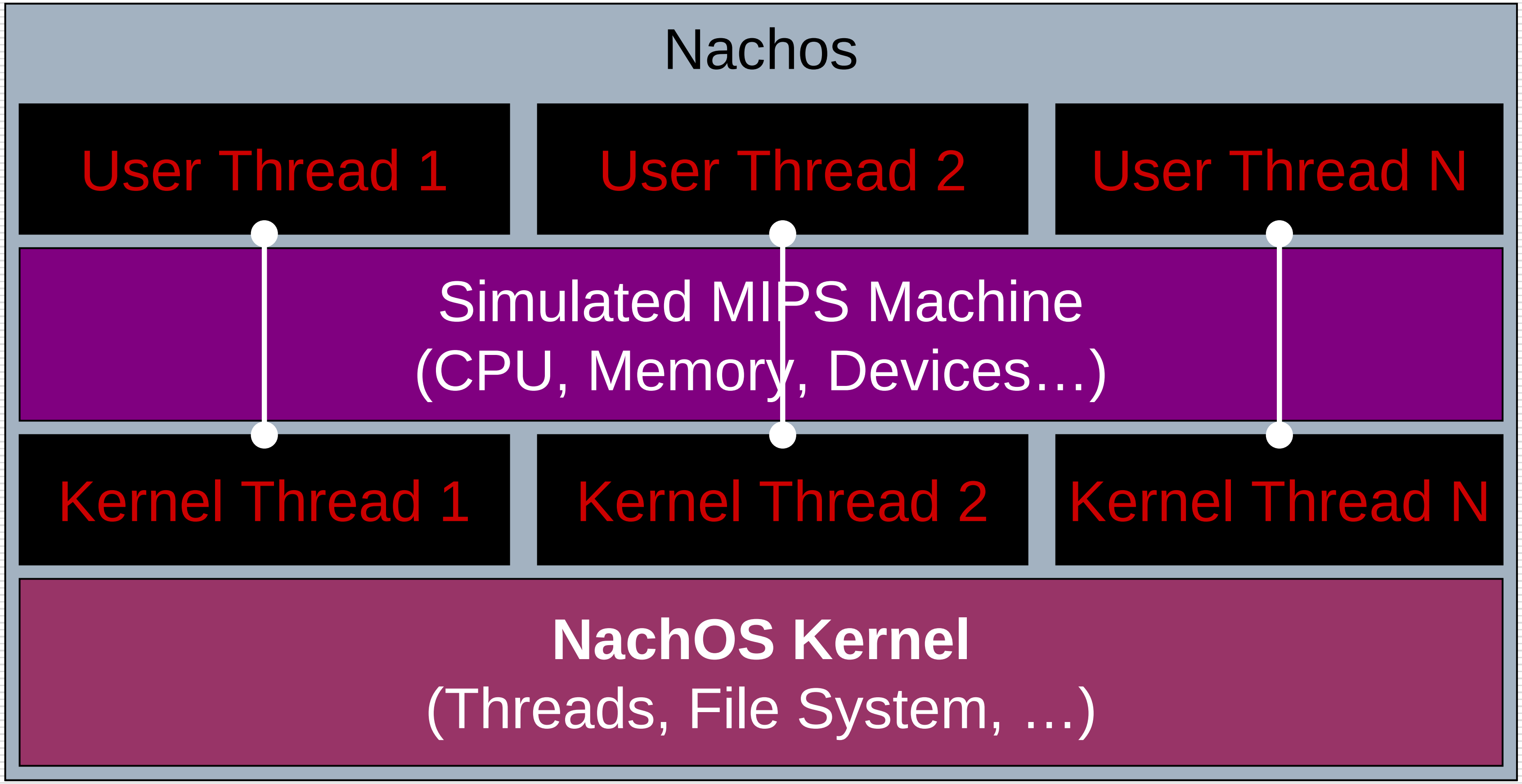
NachOS

- NachOS:
  - Not Another Completely Heuristic Operating System
- Written by Tom Anderson and his students at UC Berkeley
  - http://www.cs.washington.edu/homes/tom/nachos/

NachOS

- An educational OS used to
  - teach monolithic kernel design and implementation
  - do experiments
- Fact:
  - Real hardware is difficult to handle.
  - May break if handled wrong.
- Approach:
  - Use a virtual MIPS machine
  - Provide some basic OS elements

NachOS V4.1



Base Operating System

NachOS V. 4.1

- Simulates MIPS architecture on host system (Unix / Windows / MacOS X ?)
  - User programs need a cross-compiler (target MIPS)
- Runs multiple NachOS threads as one Unix process
- Nachos appears as a single threaded process to the host operating system

Exercises

- Ex0 – Shell, Tools & Library
- Ex1 – Thread Synchronization
- Ex2 – System Call
- Ex3 – Virtual Memory Management
- Ex4 – File System

Challenge:

- Ex5 – Optimization

What to do?

- Assignments 0-4
  - due in the last week of this term
  - short code review on design and specification (incl. documentation)
- Assignment 5
  - due in the last week before the exam
  - short code review
  - better performance than any other group

Supported Infrastructure

- Linux (x86)
  - Almost any flavor will do
- Windows (x86)
  - Cygwin (→ http://www.cygwin.com)
- MacOS X (PPC)

- Provided at our homepage:
  - Cross compiler for decstation-mips

Other Architecture

- Sorry you are on your own, pal!
  - GCC 2.95
  - Make
  - Build cross compiler yourself

  - A little tricky, but should work on any current Linux or BSD system and architecture.

Building a Cross Compiler

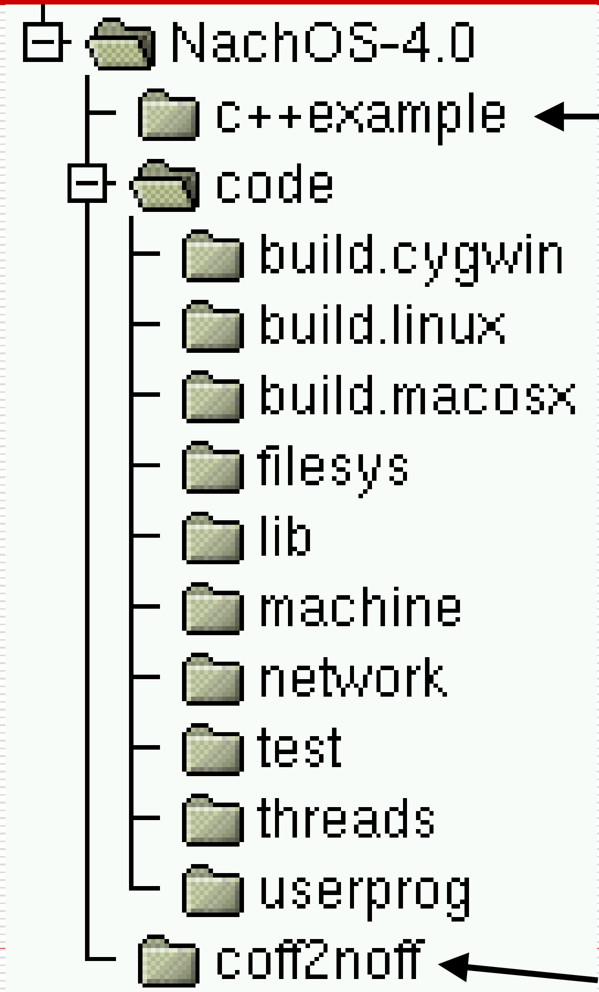
- Get binutils and gcc source from gnu.org
- Create a build directory
- „configure --target=decstation-mips --prefix=$Install-path“
- „make“ and „make install“ (first binutils, then gcc)
- Get 2.95.x, because GCC 3.x will cause zillions of troubls in NachOS

Setup your System

Get archive from our homepage

- Get NachOS-4.1.tgz
- Get GCC
  - 2.95 works best
  - 3.x causes trouble
- Get Cross-Compiler (mips-x86.*-xgcc.tgz)
  - Install as root in /usr/local/nachos....
  - 2.7.2 is venerable and proven
  - 2.95.x is ok

NachOS content



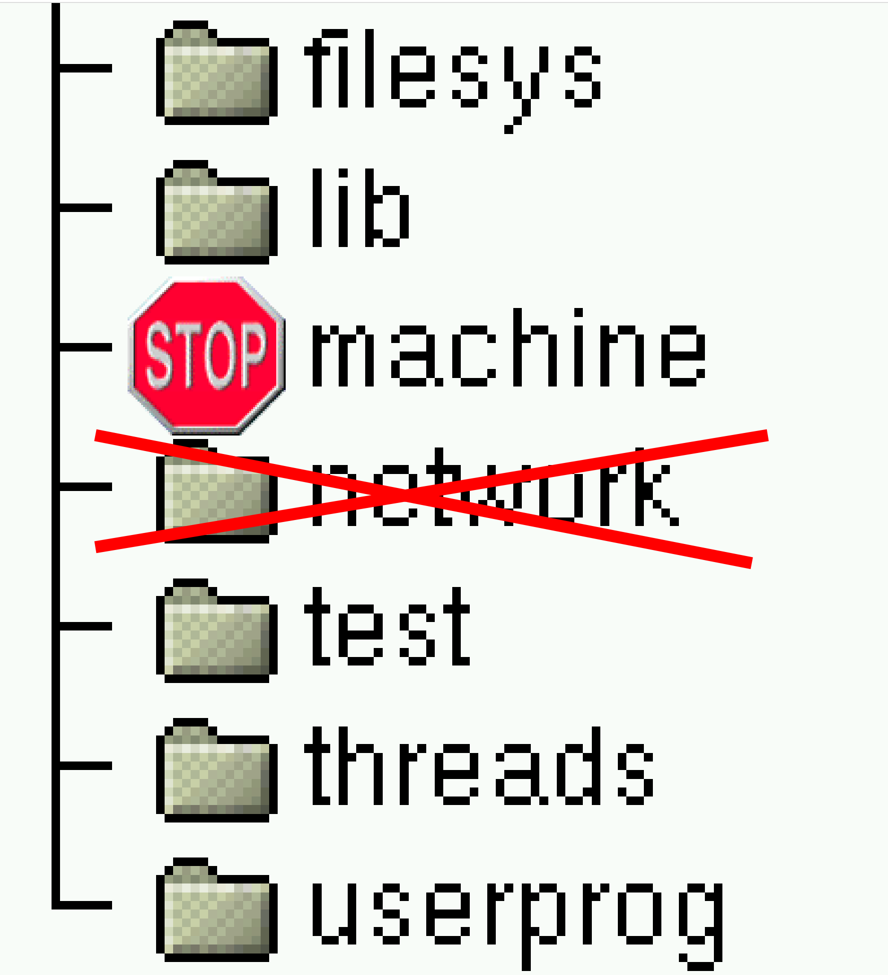
C++ introduction

Build directories:
make depend
make

Actual NachOS source of the kernel and the simulator

Tool needed to convert user programs

NachOS Code



Assignment 4

Some helper functions.

Simulator! Do not change!

Don‘t bother about this.

NachOS test applications

Assignment 1

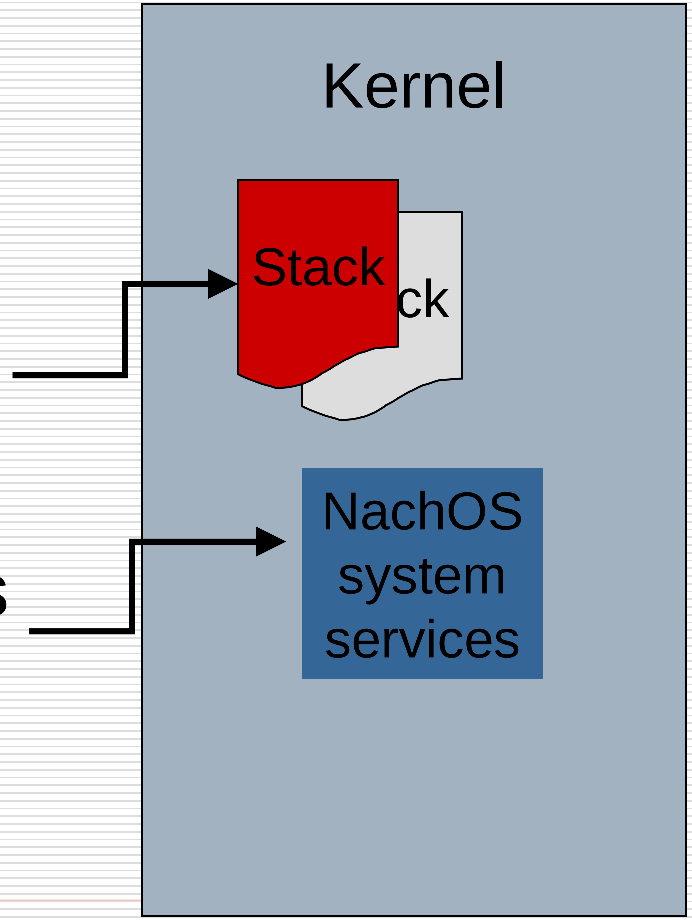
Assignment 2 & 3

How does it work?

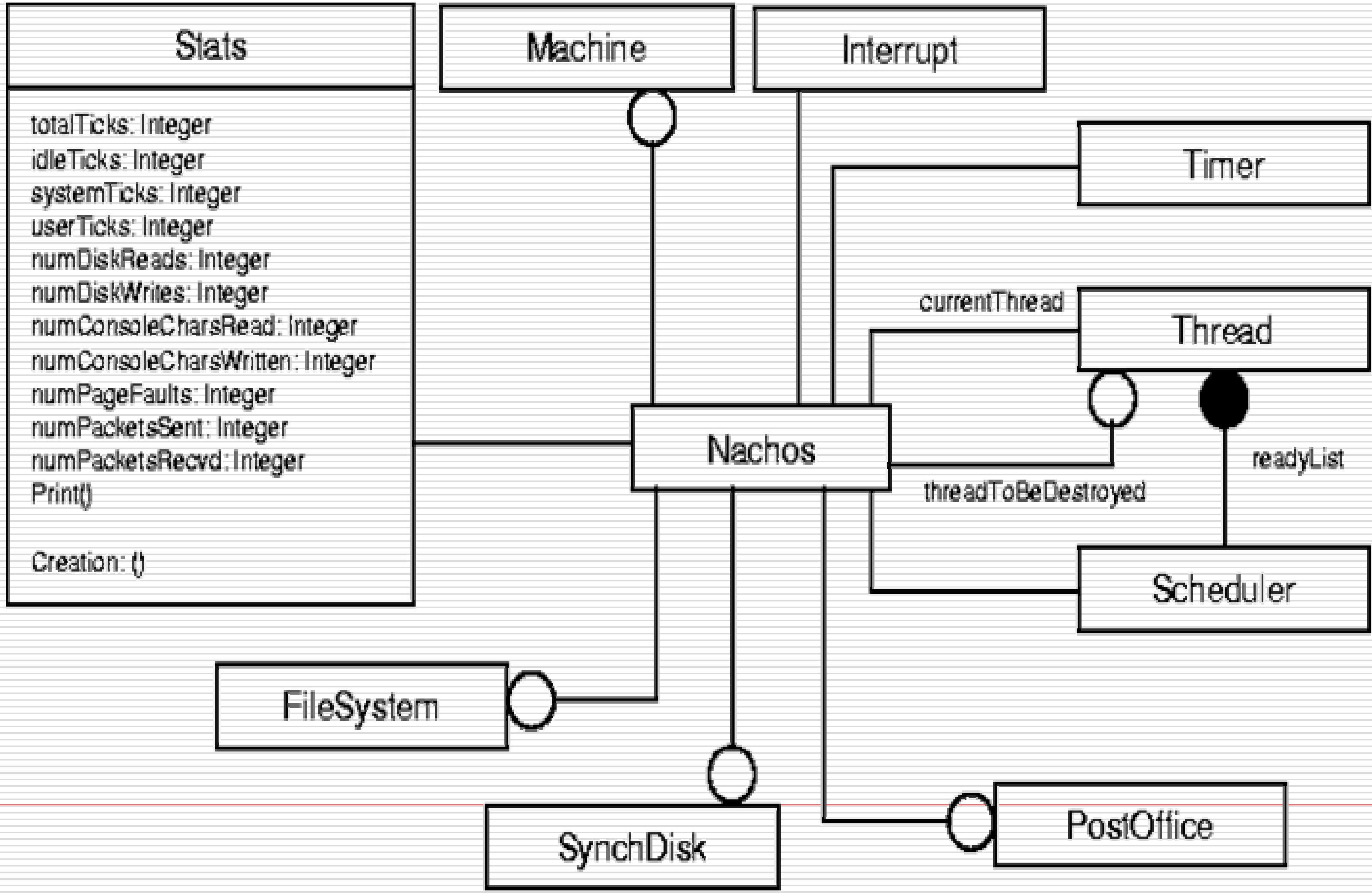
NachOS kernel is a native application on your host OS.

Implements its own user thread library and scheduler

Implements the system calls used by the user program

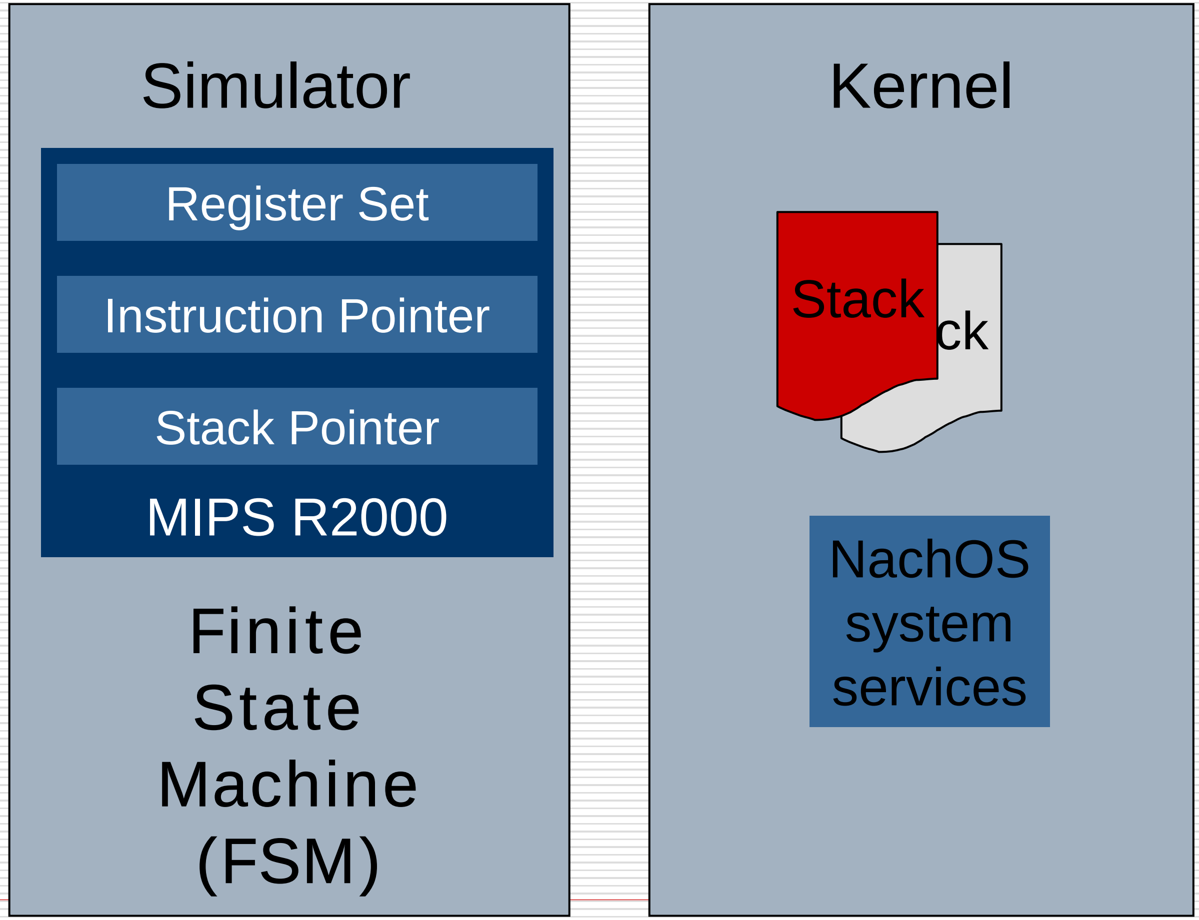


NachOS Architecture

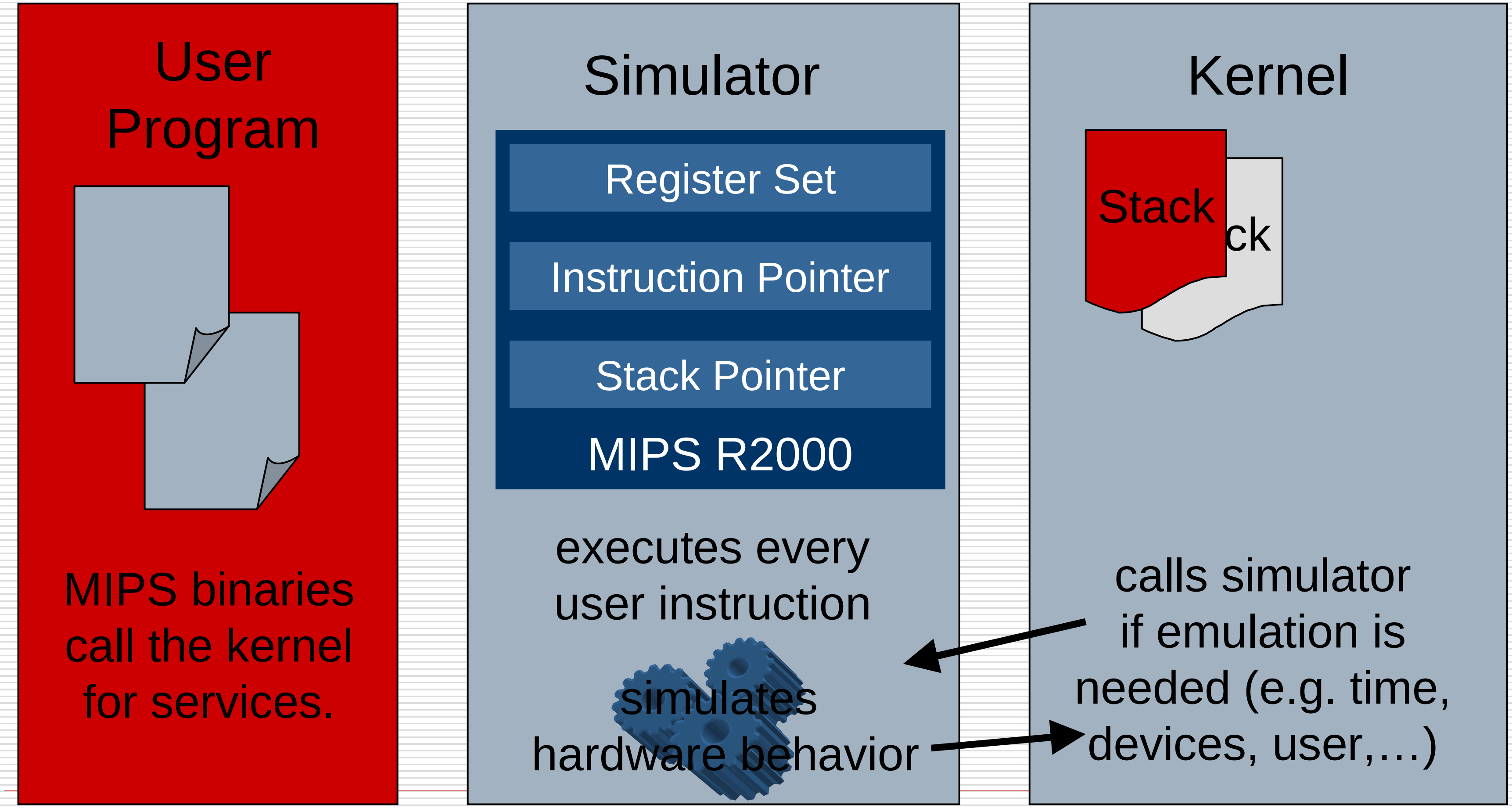


How does it work?

Essentially just a big *Switch/Case* structure, that interpretates the instructions of a MIPS Binary.

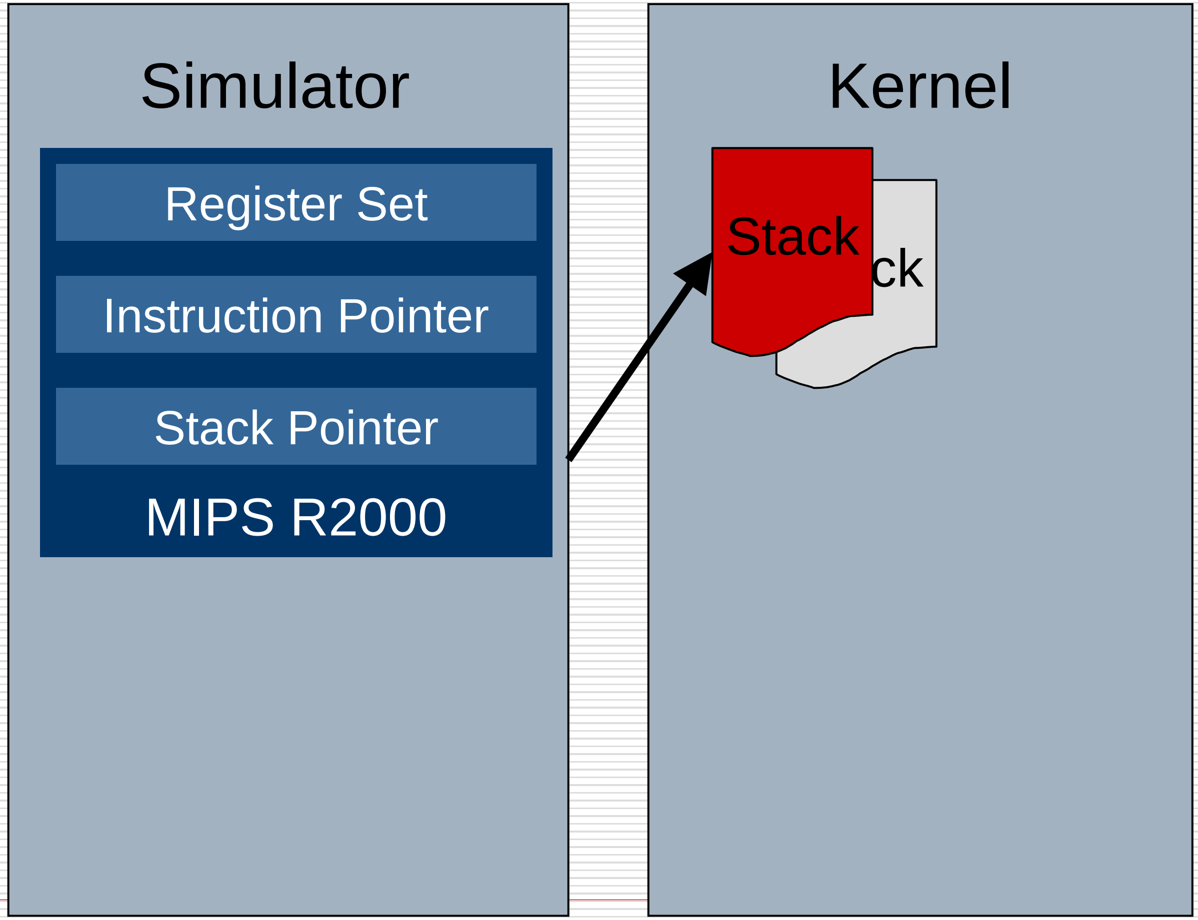


How does it work?

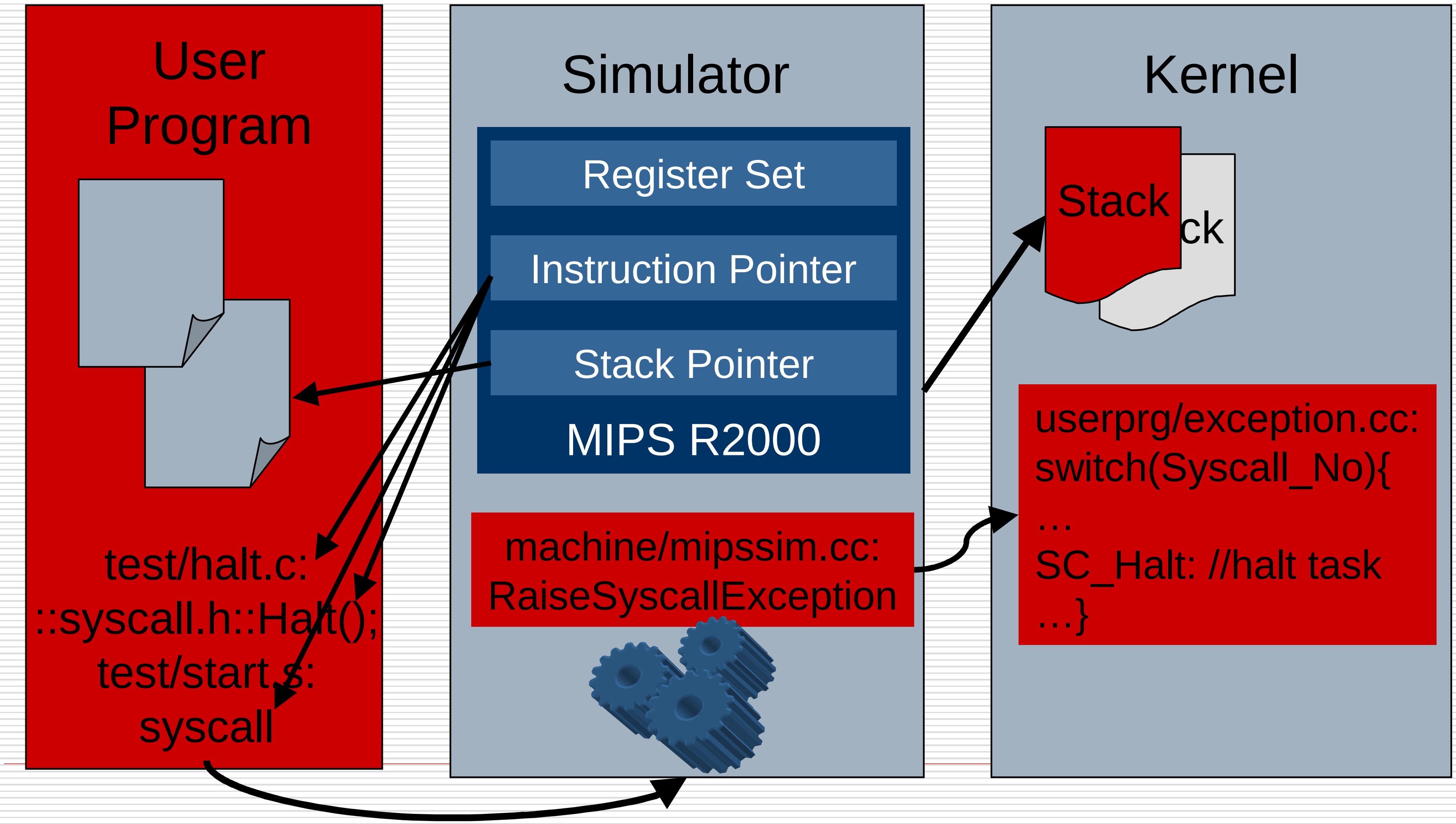


How does ot work?

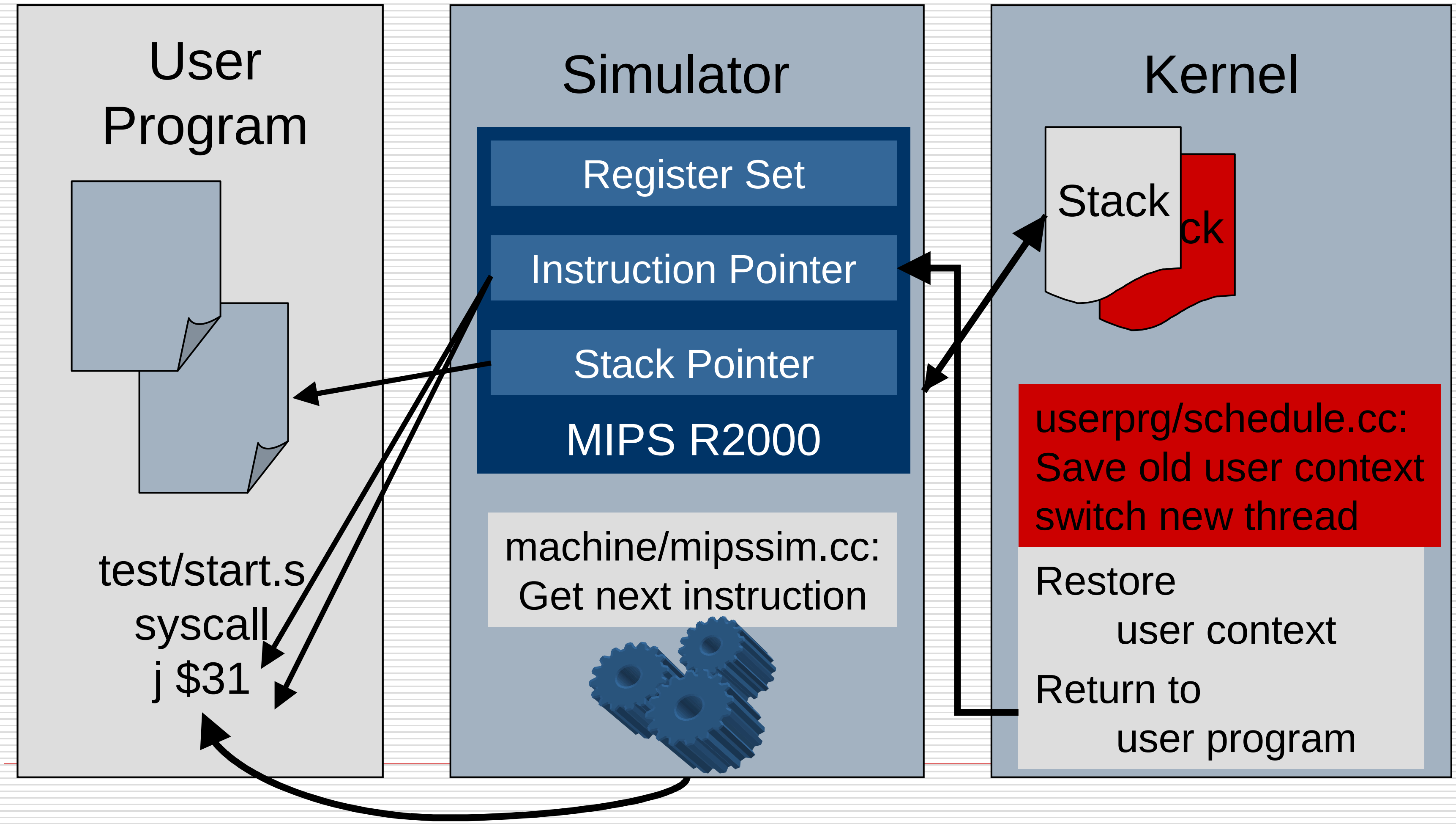
State of the simulator is tied to the current user application.



How does it work?

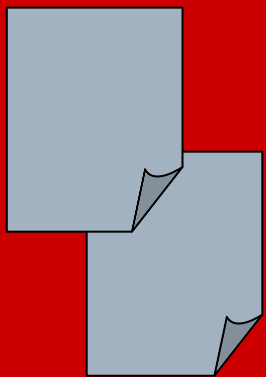


How does it work?



How does it work?

User Program



MIPS binaries
Needs
cross compiler
and NOFF-tool

Simulator

Kernel

- Runs natively on host OS. Needs gcc & binutils (included in Cygwin or Linux)

- Basic OS services are already implemented
- Simulates hardware behavior (don't need raw device access!)

test/start.s

- MIPS dependend assembler code for userlevel bindings
- **Do not modify!**
- How does it work:
  - gets parameter via registers (C-calling convention)
  - loads syscall number into first register
  - does a syscall exception (enter the kernel)
  - jump back

threads/switch.s

- Basic user level thread system
- Host Machine dependent assembler code for context switches
- **Do not modify!**
- How to perform a context switch:
  - SWITCH(oldThread, newThread);
- How does it work:
  - Saves current “register set” to stack
  - Changes stack pointer
  - Loads register set from new stack and returns

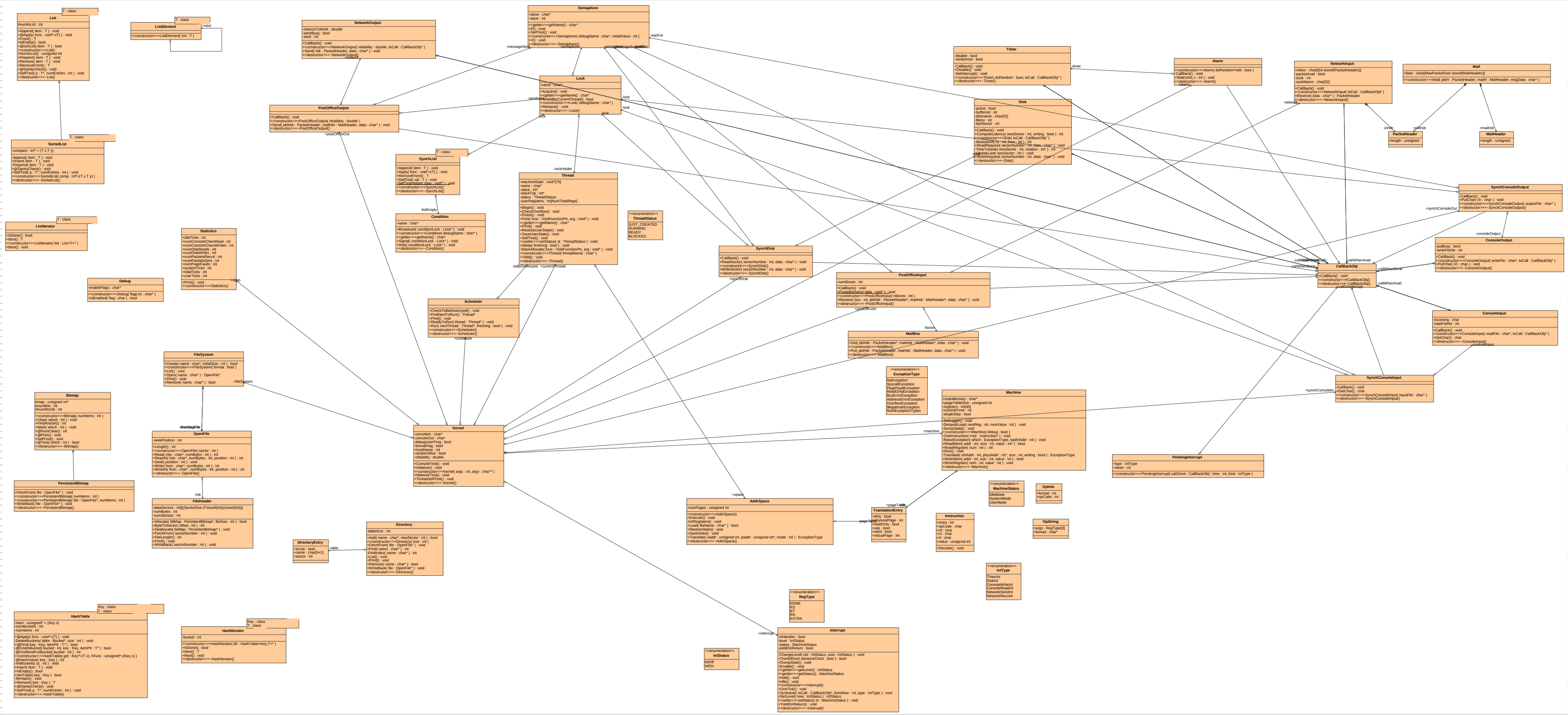
userprog/syscall.h

- Defines C-calling signature of every system call
- Has to be included into every user program

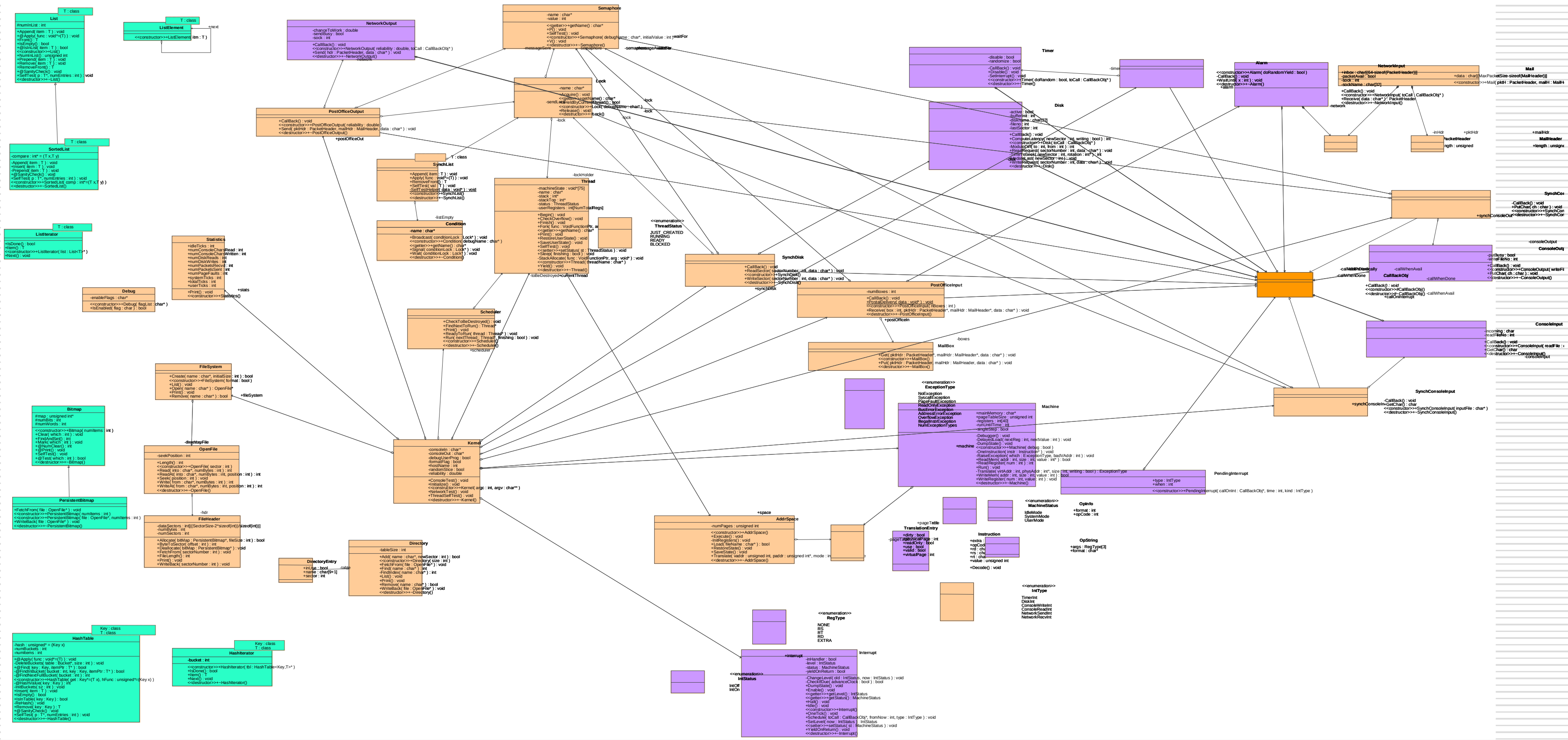
NachOS Architecture

- Complexity:
  - Vanilla this year: ~14500 LOC
  - Vanilla last year: ~12500 LOC
  - Last years best: ~20200 LOC
  - Last years worst: ~13300 LOC
  - Winner Challange: ~22200 LOC
- Reverse engineered UML diagram is on the NachOS homepage

NachOS Architecture



NachOS Architecture



System Development

- Use system architecture to design
  - Adapt design for new challanges
  - Add new components/features
- Use software engineering methods to develop
  - Write a specification
  - Use revision/version control
  - Write good unit tests while coding
  - Use building tool
  - Generate code (e.g. FSM, UML, StateCharts)

Coding Philosophy

- Write everything at once, the… art debugging
  - Big Bang
  - Write test co…
- Write e… and ch…
  - …tep
  - …ode while programming each

Both approaches work!
But which does work for you?
First is very painful!
Second is very structured!

General Fool Protection

This System is unstable.
Internal Windows Error #231
Please change the user!

System Programming

- Test your implementation
  - For instance with unit tests (e.g. lib/TUT.h)

- Test if errors are handled
- Test if wrong parameters handled
- Test if a wrong state occures
- Fault injection
- Write your own debugger to query your system state

System Programming

- Document your code!
- Write reentrant functions! (avoid global variables and state)
- Do you know what your tools are doing?

System Programming

- Optimization:

  "Premature optimization
  is the root of all evil"

  -D.E. Knuth

  - Don't do it on the first run.
  - Get the design and abstraction right in the first place!
  - ***"A good, small design yields a good performance without optimizations."***

  -J. Liedtke

Debugging

- Included debugger supports:
  - Single stepping (option –s)
  - Tracing (option –d td prints debugging messages of thread „t“ and disk „d“ emulation)
- Using gdb with NachOS
  - needs compiler option (–ggdb)
  - http://www.student.math.uwaterloo.ca/~cs354/misc/debug.html

Common Problems

- Implementation started too early
- Not enough time in interface design
- No specification
- Insufficient documentation
- Too much hand work
- Too much art work
- No design

Questions ?

Think first,
code later!

Follow the instructions given to you!
They are meant as a help!

Exercises

- Ex0 – Shell, Tools and Library
- Ex1 – Thread Synchronization
- Ex2 – System Call
- Ex3 – Virtual Memory Management
- Ex4 – File System

Challenge:

- Ex5 – Optimization

#0 Shell, Tools & Library (2 weeks)

- Write a small shell
- Write a user library
- Write some tools for file management

- Few design decisions
- Try to specify the user API clearly

#1 Thread System

- Given basic semaphores and thread switch
- Topic fields:
  - basic thread management
  - scheduler
  - further synchronization problems
- Things to learn:
  - write synchronized code and synchronization mechanisms
  - scheduling algorithms
  - thread switch
  - C/C++ calling convention

#1 Plan (2 weeks)

Detailed tasks:

- Implement Locks and Condition Variables
- Implement a non-preemptive priority-driven scheduler
- Implement Producer-Consumer, Alarm Clock, Barber Shop, etc.
- Test all synchronization mechanisms with these problems
  - Note: The mechanisms should work independently from the chosen scheduler.
  - Optional: Implement further scheduling algorithms and further test cases (dining philosophers).

#2 System Call and KDebug

Topic fields:

- Kernel Interface (system call & exceptions)
- Basic File I/O
- Kernel / User mode

Things to learn:

- Mode changes
- Copy in / copy out
- Kernel-User interaction
- Threads and Stacks
- Reentrant functions

#2 Plan (3 weeks)

Detailed tasks:

- Implement syscalls (halt, exec, exit, open, read, write, create, close)
- Implement synchronized file and console access
- Implement multitasking
- Implement preemptive multitasking
- Implement multithreaded tasks (fork, join)

#3 Caching and Virtual Memory

Topics

- Protection
- Virtual Memory
- Swapping

Things to learn:

- Address space management
- Soft-TLB management
- Pageing
- Communication

#3 Plan (4 weeks)

Detailed tasks:

- Implement Management for Soft-TLB
- Implement Address Spaces
  - Page Tables
- Implement Virtual Memory / Swapping
  - Frame Table
  - Swap to file in NachOS file system
- Look out for Protection
- Implement synchronous Inter Process Communication (simplified IPC path from exercise)
- Optional: Parameter Passing

#4 File System

Given #1to #3 and simulated disc

Topics

- File System
- Synchronization and Persistence of Files

Things to learn:

- Hardware/Software interaction
- Basic file system properties
- Persistence of a file
- Consistence of a file

#4 Plan (4 weeks)

Detailed Tasks:

- Complete the NachOS file system to run with your disc driver, watch out for synchronization
- Implement a hierarchical name space (directory based)
- Optional: Improve disc driver (rotational delay, disc seek)
- Optional: Avoid disc corruption or write tool to restore a corrupted disc

#5 Performance

Given #1 to #4

Topics:

- Memory Hierarchy
- Working Set
- Buffering
- Cache Conflicts and Cache Effects

Things to learn

- Performance Monitoring
- Profiling
- Working Set Strategy
- Stress Test of Implementation
- 1st & 2nd level cache performance ☺

#5 Plan (8 weeks)

Detailed Tasks:

- Implement Buffering for your disc access (ro)
- Implement Applications with high memory footprint (matrix multiplication)
- Measure Performance with different Paging Strategies and Page Replacement Algorithms
- Improve your System
- Whose system is the fastest
- Optional: Improve your Application
- Optional: Who has the fastest combination?