Supporting User Program

Chunzhi Su

ACM Honored Class
Shanghai Jiao Tong University

User Code vs Kernel Code

In Nachos (not in real world)

- All codes written in your java project are kernel codes (except the "virtual machine"). They are executed by JVM.
- User programs are written in assembly codes, executed by the "CPU" : a MIPS simulator written in JAVA.

So kernel codes are not executed by "CPU"!!
This is one reason why Nachos is only a toy, not a real OS.

Why we need this phase

Although user codes are mainly executed(simulated) by "CPU", they also need supports from OS

- System Call
- Handle CPU Exception
- Virtual-to-physical Address Translation
- FileSystem

Goal of this phase is to implement these supports for user program.

What you have ...

You have got a “virtual machine”.

- nachos.machine.Processor
  - It is the “CPU”.
  - It includes a “main memory”.
  - It supports a subset of R3000 instruction set.

What you have ...

You have also got a naive filesystem available.

- nachos.machine.FileSystem & nachos.machine.OpenFile
    - Interface of filesystem and file.
    - Filesystem creates OpenFile.
- nachos.machine.StubFileSystem
    - A basic implementation of filesystem and file.

What you deal with ...

- nachos.userprog.UserKernel
  - Extends ThreadedKernel.
  - A kernel that support multiple user processes.
  - Contains global algorithm and data for the OS.

What you deal with ...

- nachos.userprog.Uthread
  - Extends Kthread.
  - Executes user codes inside a user process
    - call “CPU” to run user program.
    - context switch, etc.
- nachos.userprog.UserProcess
  - Contains local algorithm and data for a process, such as page table, file descriptors.
  - much work to do here.

How things work ...

- Booting
    - Kernel does the initialization.
    - Call UserProcess.execute(shellProg, args) to start shell.
- Launching a user process (UserProcess.execute())
    - Load binary code from file.
    - Create and initialize a UThread.
    - UThread calls “CPU” to simulate instructions.

How things work ...

- Address Translation
  - Page table is an array of nachos.machine.TranslationEntry.
  - In this phase, “CPU” supports a “hard-wired” page table.
  - Call Processor.setPageTable() before process runs.
  - In later phases, “CPU” wont support this any longer.

How things work ...

- Handle Exception
    - When "CPU" throws exception, exception handler in UserKernel is invoked.
    - Kernel then calls UserProcess.handleException(cause).
    - Write codes to handle exceptions in UserProcess.handleException(cause).

Task 1

Implement syscalls for file management.

- creat, open, read, write, close, unlink.
- See syscall.h for details.
- **You are not asked to implement a filesystem**
  - In this phase, use StubFileSystem directly.
  - For each syscall, invoke corresponding method of filesystem interface.

Task 2

Implement simple paging using page table.

- UserProcess.readVirtualMemory() & UserProcess.writeVirtualMemory()
  - They are used to copy data between kernel and user's virtual address space
    - Kernel space : JVM memory.
    - User space : "Memory" in nachos.machine.Processor.
  - Remember: **in Nachos**, user code and kernel code dont share CPU and memory.

Task 3

Implement syscalls for process management.

- exec, join, exit.
- Again, see syscall.h for details.
- **Bullet-proof all the syscalls**
  - There should be **nothing** a user program can do to crash your OS.
  - Even if user program passes wrong args in syscall.

Task 4

Implement a lottery scheduler.

- If you made a good design before implementing PriorityScheduler, this will be very easy.

Console

- OpenForReading()
    - Returns an OpenFile.
    - Standard input for user program.
- OpenForWriting()
    - Returns an OpenFile.
    - Standard output for user program.

Thank you. Q&A