Protection: Processes and Kernels

Professor Natacha Crooks
https://cs162.org/

Slides based on prior slide decks from David Culler, Ion Stoica, John Kubiatowicz, Alison Norman and Lorenzo Alvisi
Admistratrivia
Homework and Early Drop Deadline

HW0 Due 31/8

Should be working on Homework 0 already!

cs162-xx account, Github account
Vagrant and VirtualBox – VM environment for the course
  » Consistent, managed environment on your machine

Get familiar with all the cs162 tools, submit to autograder via git
Homework and Early Drop Deadline

HW0 Due 1/9 (Same Day as Early Drop Deadline)

Should be working on Homework 0 already!

Get familiar with all the cs162 tools, submit to autograder via git

HW1 will be released on 2/9
Projects are looming

Group Formation Form (Link on EdStem) is due 2/9.

There is a teammate search functionality on EdStem.

Discussions are starting! First 2 optional but mandatory afterwards

Project 0 will be released on 9/4
An operating system implements a virtual machine for the application whose interface is more convenient than the raw hardware interface (convenient = security, reliability, portability)
Recall: Three main hats

**Referee**
Manage protection, isolation, and sharing of resources

**Illusionist**
Provide clean, easy-to-use abstractions of physical resources

**Glue**
Provides a set of common services
Recall: HW Complex

- Really High Speed I/O (e.g. graphics)
- High-Speed I/O devices (PCI Exp)
- Disks (8 x SATA)
- Slower I/O (USB)
- Integrated Ethernet

Intel Skylake-X I/O Configuration

Memory Channels (High BW DRAM)

- Direct Media Interface (3.93 GBytes/sec)
- HD Audio
- PCI/e Drives
- RAID 0/1/5/10
- Smart Connect (autoupdate)
Recall: Increasing Software Complexity

- Mouse Base Pairs
- Modern Car
- Mac OS X "Tiger"
- Windows 7
- Windows Vista
- Facebook
- Microsoft Office 2013
- Original Unix
- Linux 3.1 (2005)
- Linux 2.2.0 (2000)
- Mars Curiosity Rover
- Firefox
- Android
- Linux 5.6 (2020)
- Linux 5.6 (2020)
# Topic Breakdown

<table>
<thead>
<tr>
<th>Virtualizing the CPU</th>
<th>Process Abstraction and API</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Threads and Concurrency</td>
</tr>
<tr>
<td></td>
<td>Scheduling</td>
</tr>
<tr>
<td>Virtualizing Memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Virtual Memory</td>
</tr>
<tr>
<td></td>
<td>Paging</td>
</tr>
<tr>
<td>Persistence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO devices</td>
</tr>
<tr>
<td>Distributed Systems</td>
<td></td>
</tr>
<tr>
<td></td>
<td>File Systems</td>
</tr>
<tr>
<td></td>
<td>Challenges with distribution</td>
</tr>
<tr>
<td></td>
<td>Data Processing &amp; Storage</td>
</tr>
</tbody>
</table>
Mechanisms vs Policy

**Mechanism**

Low-level methods or protocols that implement a needed piece of functionality

A Brake Pedal!

**Policy**

Algorithms for making decisions within the OS.

Use the mechanism.

“I break when I see a stop sign”
Goals for Today

- What are the requirements of a good VM abstraction?
- What is a process?
- How does the kernel use processes to enforce protection?
- When does one switch from kernel to user mode and back?
Goal 1: Requirements for Virtualization
The OS will protect you

Protection is necessary to preserve the virtualization abstraction

Protect applications from other application’s code (reliability, security, privacy)

Protect OS from the application

Protect applications against inequitable resource utilisation (memory, CPU time)
Goal 2: What is a Process?
A process (simplified)

A process is an **instance** of a running program.

- **CPU**
  - Memory (address space)
    - Store code, data, stack, heap
- **Registers**
  - Program Counter, Stack Pointer, Regular registers
- **IO information**
  - Open files (and others)
From program to process

Executable image, instructions and data

```
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

```
crooks@laptop> ./helloworld
crooks@laptop> ./helloworld
```

Physical Memory
- Code
- Data
- Heap
- Stack

Task Manager
- Processes
  - Google Chrome (41)
    - Status: 0.9% CPU, 3.36 GB Memory, 0.1 MB/s Disk, 0.1 MB/s Network, 1.7% GPU, GPU 0 - Video Decode
      - Power usage: Very low, Power usage time: Very low
  - Microsoft PowerPoint (2)
    - Status: 0.1% CPU, 735.1 MB Memory, 0.1 MB/s Disk, 0.1 MB/s Network, 0% GPU
      - Power usage: Very low, Power usage time: Very low
  - Slack (5)
    - Status: 0% CPU, 413.2 MB Memory, 0.1 MB/s Disk, 0.1 MB/s Network, 0% GPU
      - Power usage: Very low, Power usage time: Very low
A process can be in one of several states: (real OSes have additional variants)
Process Management by the OS

Process Control Block (or process descriptor) in OS stores necessary metadata

- PC
- Stack Ptr
- Registers
- PID
- UID
- List of open files
- Process State
- ...

Crooks CS162 © UCB Fall 2023
Three “Prongs” for the Class

Understanding OS principles

System Programming

Map Concepts to Real Code
Processes in the wild (well, in the kernel)

```c
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };```

// Per-process state
```c
struct proc {
    uint sz;                     // Size of process memory (bytes)
pde_t* pgdir;                // Page table
char *kstack;                // Bottom of kernel stack for this process
enum procstate state;        // Process state
int pid;                     // Process ID
struct proc *parent;         // Parent process
struct trapframe *tf;        // Trap frame for current syscall
struct context *context;     // swtch() here to run process
void *chan;                  // If non-zero, sleeping on chan
int killed;                  // If non-zero, have been killed
struct file *ofile[NOFILE];  // Open files
struct inode *cwd;           // Current directory
char name[16];               // Process name (debugging)
};```

Xv6 Kernel (proc.h)

In Linux: task_struct defined in <linux/sched.h>
Processes in Pintos

```c
struct process {
    /* Owned by process.c. */
    uint32_t* pagedir;   /* Page directory. */
    char process_name[16]; /* Name of the main thread */
    struct thread* main_thread; /* Pointer to main thread */

    /* All the fun data structures you're going to add */
};
```

Pintos
(userprog/process.h)
Many Processes

**Process List** stores all processes

```c
struct {
    struct spinlock lock;
    struct proc proc[NPROC];
} ptable;
```

*Xv6 Kernel (proc.c)*

**Run Queues**

Lists all PCBs in READY state

**Wait Queues**

Lists all PCBs in BLOCKED state
The Illusionist and the Referee are Back

**Illusionist**

Give every process the illusion of running on a private CPU

Give every process the illusion of running on private memory

**Referee**

Manage resources to allocate to each process

Isolate process from all other processes and protect OS
Operating System Kernel

Lowest level of OS running on system. Kernel is trusted with full access to all hardware capabilities.

All other software (OS or applications) is considered untrusted.

<table>
<thead>
<tr>
<th></th>
<th>Untrusted</th>
<th>Trusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rest of OS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating System Kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SPECIAL
The Process, Refined

A executing program with restricted rights

Enforcing mechanism must not hinder functionality or hurt performance
User vs Kernel: Dr Jekyll and Mr Hyde

Application/User Code (Untrusted)

Run all the processor with all potentially dangerous operations disabled

Kernel Code (Trusted)

Runs directly on processor with unlimited rights

Performs any hardware operations

But run on the same machine!
How can the kernel enforce restricted rights?

1) While preserving functionality
2) While preserving performance
3) While preserving control
Attempt 1: Simulation
Recall: CPU Instruction Cycle (from CS61c)
Attempt 1: Simulation

Process

OS

Hardware

Have the Kernel interpret and check every instruction!

Potential Issues:

Extremely slow! Would have to cycle through all operations, switch into the kernel, etc.

Unnecessary. Most operations are perfectly safe!
Attempt 2: Dual Mode Operation

Hardware to the rescue!
Use a bit to enable two modes of execution

In User Mode

Processor checks each instruction before executing it

Executes a limited (safe) set of instructions

In Kernel Mode

OS executes with protection checks off

Can execute any instructions
Hardware must support

1) Privileged Instructions
   Unsafe instructions cannot be executed in user mode

2) Memory Isolation
   Memory accesses outside a process’s address space prohibited

3) Interrupts
   Ensure kernel can regain control from running process

4) Safe Transfers
   Correctly transfer control from user-mode to kernel-mode and back
Req 1/4: Privileged Instructions

Cannot change privilege level (set mode bit)

Cannot change address space

Cannot disable interrupts

Cannot perform IO operations

Cannot halt the processor
How can an application do anything useful ...

Asks for permission to access kernel mode!

**System calls** Transition from user to kernel mode only at specific locations specified by the OS

**Exceptions** User mode code attempts to execute a privileged exception. Generates a processor exception which passes control to kernel at specific locations

More on safe control transfers later
Hardware must support

1) Privileged Instructions
   Unsafe instructions cannot be executed in user mode

2) Memory Isolation
   Memory accesses outside a process's address space prohibited

3) Interrupts
   Ensure kernel can regain control from running process

4) Safe Transfers
   Correctly transfer control from user-mode to kernel-mode and back
Req 2/4: Memory Protection

OS and applications both resident in memory

Application should not read/write kernel memory (or other apps memory)
The character could leave the game area and start overwriting other running programs and kernel memory.

One of the worst bugs I ever had to deal with was in this game. Once the game player made it to the Colony, every so often the system would crash and burn at totally random times. You might be playing for ten minutes when it happened or ten hours, but it would just die in a totally random way.

There was a slow-moving slug like creature that knew how to follow the game player’s trail. When it came across another creature, rather than bouncing off and risk losing the trail, I made it so that it would destroy the other creature and stay on target to find you. This worked great, except that on some rare occasions, this slug could do to a wall what it did to the other creatures. That is, it could delete it. This meant that the virtual door was now open for this creature to explore the rest of the RAM on the Macintosh, deleting and modifying it as it went along. Of course, it was just a matter of time before it found some juicy code. In other words, the bug was a REAL bug.
Super Mario Land 2

Mario could exit a level and explore the entire memory of the system.
Attempt 1: Isolation

Hardware to the rescue! (Again)
Base and Bound registers

<table>
<thead>
<tr>
<th>Code</th>
<th>Data</th>
<th>Heap</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base

Bound

Address Space
Process 1

<table>
<thead>
<tr>
<th>Code</th>
<th>Data</th>
<th>Heap</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base

Bound

Address Space
Process 2
Hardware to the rescue! (Again)

Base and Bound registers

Attempt 1: Isolation

CPU

Memory Reference → OK?

Yes → Continue

No → Generate Exception
Kernel Mode executes without Base and Bound registers

What can the Kernel see?

a) Kernel memory only
b) Kernel memory + application memory of app that “invoked” kernel
c) Everything
Limitations of Isolation

1) Expandable memory
   Static memory allocation

2) Memory Sharing
   Cannot share memory between processes

3) Non-Relative Memory Addresses
   Location of code & data determined at runtime

4) Fragmentation
   Cannot relocate/move programs. Leads to fragmentation
Virtual address space

Set of memory addresses that process can “touch”

Physical address space

Set of memory addresses supported by hardware
Attempt 2: Virtualization

Map from virtual addresses to physical addresses through address translation.
Attempt 2: Virtualization

Continues to provide isolation

Process 1, Virtual Memory Address

Process 2, Virtual Memory Address

Physical for Memory P1

Physical for Memory P2
Benefits of Virtualization

1) Expandable memory
Whole space of virtual address space! Even physical address not resident in memory

2) Memory Sharing
Same virtual address can map to same physical address

3) Relative Memory Addresses
Every process’s memory always starts at 0

4) Fragmentation
Can dynamically change mapping of virtual to physical addresses
What does this program do? (CS61C)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    int *p = malloc(sizeof(int));
    printf("(%d) p: %p\n", getpid(), p);
    *p = 0;
    while (1) {
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p);
    }
    return 0;
}
```

crooks@laptop> gcc -o memory memory.c -Wall
crooks@laptop> ./memory
(120) p: 0x200000
(120) p: 1
(120) p: 2
(120) p: 3
(120) p: 4
crooks@laptop> ./memory & ./memory
(120) p: 0x200000
(254) p: 0x200000

Are these virtual or physical addresses?

Virtual memory provides each process with illusion of own complete (and infinite) memory
**Virtual Memory is Hard!**

**Virtualizing the CPU**
- Process Abstraction and API
- Threads and Concurrency
- Scheduling

**Virtualizing Memory**
- Virtual Memory
- Paging

**Persistence**
- IO devices
- File Systems

**Distributed Systems**
- Challenges with distribution
- Data Processing & Storage

---

Crooks CS162 © UCB Fall 2023
Hardware must support

1) Privileged Instructions
   Unsafe instructions cannot be executed in user mode

2) Memory Isolation
   Memory accesses outside a process’s address space prohibited

3) Interrupts
   Ensure kernel can regain control from running process

4) Safe Transfers
   Correctly transfer control from user-mode to kernel-mode and back
Req 3/4: Interrupts

Kernel must be able to regain control of the processor

Hardware to the rescue! (Again x 2)

Hardware Interrupts

Set to interrupt processor after a specified delay or specified event and transfer control to (specific locations) in Kernel.

Resetting timer is a privileged operation
Hardware must support

1) Privileged Instructions
   Unsafe instructions cannot be executed in user mode

2) Memory Isolation
   Memory accesses outside a process’s address space prohibited

3) Interrupts
   Ensure kernel can regain control from running process

4) Safe Transfers
   Correctly transfer control from user-mode to kernel-mode and back
Req 4/4: Safe Control Transfer

How do safely/correctly transition from executing user process to executing the kernel?

1) System Calls
2) Exceptions
3) Interrupts

Asynchronous
Can be maskable or non-maskable

Synchronous Events (trapping)
User program requests OS service
Transfers to kernel at well-defined location

Synchronous/non-maskable

Read input/write to screen, to files, create new processes, send network packets, get time, etc.

How many system calls in Linux 3.0?

a) 15  b) 336  c) 1021  d) 21121

https://man7.org/linux/man-pages/man2/syscalls.2.html
**System Calls are the “Narrow Waste”**

Simple and powerful interface allows separation of concern
Eases innovation in user space and HW

- Compilers
- Word Processing
- Web Browsers
- Email
- Web Servers
- Databases
- Email
- Word Processing
- Web Browsers
- Web Servers
- Databases

- Portable OS Library
- Application / Service

- User
- System

- Portable OS Kernel
- Platform support, Device Drivers

- Software
- Hardware
- x86
- PowerPC
- ARM
- PCI

- Ethernet (1Gbs/10Gbs)
- 802.11 a/g/n/ac
- SCSI
- Graphics
- Thunderbolt

Crooks CS162 © UCB Fall 2023
System Calls in the Wild (In Linux)

```c
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The __x64_sys_() stubs are created on-the-fly for sys_() system calls
# The abi is "common", "64" or "x32" for this file.

# common read
__x64_sys_read

...
Safe Control Transfer: Exceptions

Any unexpected condition caused by user program behaviour

Stop executing process and enter kernel at specific exception handler

Synchronous and non-maskable

Process missteps (division by zero, writing read-only memory)
Attempts to execute a privileged instruction in user mode
Debugger breakpoints!
Exceptions in the Wild (In Linux)

```c
#define _X86_TRAP_MH

// Interrupts/Exceptions */

#define X86_TRAP_DIVIDE /* Divide-by-zero */
#define X86_TRAP_DB /* Debug */
#define X86_TRAP_NMI /* Non-maskable Interrupt */
#define X86_TRAP_BP /* Breakpoint */
#define X86_TRAP_OF /* Overflow */
#define X86_TRAP_RF /* Round Range Exceeded */
#define X86_TRAP_I/O /* Invalid Opcode */
#define X86_TRAP_OV /* Device Not Available */
#define X86_TRAP_PF /* Double Fault */
#define X86_TRAP_CL/P /* Coprocessor Segment Overrun */
#define X86_TRAP_TS /* Invalid TSS */
#define X86_TRAP_NM /* Segment Not Present */
#define X86_TRAP_SS /* Stack Segment Fault */
#define X86_TRAP_GP /* General Protection Fault */
#define X86_TRAP_XF /* Page Fault */
```
Safe Control Transfer: Interrupts

Asynchronous signal to the processor that some external event has occurred and may require attention.

When process interrupt, stop current process and enter kernel at designated interrupt handler.

Timer Interrupts, IO Interrupts, Interprocessor Interrupts
Safe Control Transfer: Kernel->User

New Process Creation
Kernel instantiates datastructures, sets registers, switches to user mode

Resume after an exception/interrupt/syscall
Resume execution by restoring PC, registers, and unsetting mode

Switching to a different process
Save old process state. Load new process state (restore PC, registers). Unset mode.
Summary: Goals for today

- What are the requirements of a good VM abstraction?
- What is a process?
- How does the kernel use processes to enforce protection?
- When does one switch from kernel to user mode and back?
Summary: Goals for today

• What are the requirements of a good VM abstraction?

• What is a process?

• How does the kernel use processes to enforce protection?

• When does one switch from kernel to user mode and back?

Protection while preserving functionality and performance

Program execution with restricted rights

Dual-Mode operation: privileged instructions, memory protection, control, interrupts, safe control transfer

System Calls, Interrupts, Exceptions