Protection: Processes and Kernels

Professor Natacha Crooks
https://cs162.org/

Slides based on prior slide decks from David Culler, Ion Stoica, John Kubitawociz, Alison Norman and Lorenzo Alvisi
Admistratrivia
Both on Friday Sept 2\textsuperscript{nd}

Should be working on Homework 0 already!

\texttt{cs162-xx account, Github account\textbackslash Vagrant and VirtualBox – VM environment for the course}

\hspace{1em} » Consistent, managed environment on your machine

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

Group Formation Form (Link on EdStem) is due 4/9.

There is a teammate search functionality on EdStem.

Discussions are starting! First 2 optional but mandatory afterwards
Recall: Operating System

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)
- Direct Media Interface (3.93 GBytes/sec)
- Memory Channels (High BW DRAM)
- High-Speed I/O devices (PCI Exp)
- Disks (8 x SATA)
- Slower I/O (USB)
- Integrated Ethernet

Intel Skylake-X I/O Configuration
Recall: Increasing Software Complexity

![Bar Chart: Increasing Software Complexity Over Time]

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

Virtualizing the CPU

Process Abstraction and API

Virtualizing Memory

Threads and Concurrency

Persistence

Scheduling

Distributed Systems

Virtual Memory

Challenges with distribution

Paging

Data Processing & Storage

IO devices

File Systems
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 is an instance of a running program

- CPU
- Memory (address space)
- Registers
- IO information

- Store code, data, stack, heap
- Program Counter, Stack Pointer, Regular registers
- Open files (and others)
From program to process

Executable image, instructions and data

./helloworld

crooks@laptop> ./helloworld

crooks@laptop> ./helloworld
Process Life Cycle

A process can be in one of several states:
(real OSes have additional variants)
Process Control Block (or process descriptor)
in OS stores necessary metadata

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

Process Management by the OS
Three “Prongs” for the Class

- Understanding OS principles
- System Programming
- Map Concepts to Real Code
Processes in the wild (well, in the kernel)

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

// Per-process state
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>`
Many Processes

Process List stores all processes

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

Xv6 Kernel (proc.c)

Run Queues

Wait Queues

Lists all PCBs in **READY state**

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

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

Address Space

Process 1

Process 2
Attempt 1: Isolation

Hardware to the rescue! (Again)

**Base and Bound** registers

Diagram:

- **CPU** -> **OK?**
  - **Yes** -> **Continue**
  - **No** -> **Generate Exception**
Attempt 1: Isolation

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
Attempt 2: Virtualization

Virtual address space
Set of memory addresses that process can “touch”

Physical address space
Set of memory addresses supported by hardware
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 Memory for P1

Physical Memory for 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;
}
```

```bash
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
(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!

<table>
<thead>
<tr>
<th>Virtualizing the CPU</th>
<th>Process Abstraction and API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtualizing Memory</td>
<td>Threads and Concurrency</td>
</tr>
<tr>
<td>Persistence</td>
<td>Scheduling</td>
</tr>
<tr>
<td>Distributed Systems</td>
<td>Virtual Memory</td>
</tr>
<tr>
<td></td>
<td>Paging</td>
</tr>
<tr>
<td></td>
<td>IO devices</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>
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
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
How do safely/correctly transition from executing user process to executing the kernel?

1) System Calls
2) Exceptions
3) Interrupts

Synchronous Events (trapping)

Asynchronous
Can be maskable or non-maskable
Safe Control Transfer: System Calls

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

Portable OS Library
System Call Interface
Portable OS Kernel
Platform support, Device Drivers

User
System
Software
Hardware

x86
PowerPC
ARM

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

Compilers
Word Processing
Web Browsers
Email
Web Servers
Databases

Application / Service

Crooks CS162 © UCB Fall 2022
System Calls in the Wild (In Linux)
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-maskeable

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)
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?
  Protection while preserving functionality and performance

• What is a process?
  Program execution with restricted rights

• How does the kernel use processes to enforce protection?
  Dual-Mode operation: privileged instructions, memory protection, control, interrupts, safe control transfer

• When does one switch from kernel to user mode and back?
  System Calls, Interrupts, Exceptions