CS3210: Isolation Mechanisms

Lecture 4

Instructor: Dr. Tim Andersen

Administrivia

- Lab 2 on Virtual Memory Due Feb 10 (one of the trickiest labs!)
- (Feb 16) Quiz #1. Lab1-3, Ch 0-3, Appendix A/B
- (Feb 20) Final Project Pre-Proposal Due
  - Start forming groups and brainstorming now

Outline

- Kernel Organization: Monolithic vs. Microkernel
- Isolation
- System Calls
- Memory

Kernel Organization: Kernel vs. User Mode

- What runs in kernel mode?

Kernel Organization

- What runs in kernel mode?
  - If the kernel interface is the system call interface, then, in general, all operating system functions run in kernel mode.
  - This is the monolithic kernel design.

Kernel Organization: Monolithic Kernel

In the monolithic "organization the complete operating system runs with full hardware privilege." -- xv6 text

Pros:

- OS designer does not need to determine which parts of the OS need which privilege.
- Easy for parts of OS to cooperate.

Con:

- Mistakes are easier to make and often fatal.

Kernel Organization: Microkernel

Microkernel reduces the number of lines that run in kernel mode to a minimum.

Pros:

- Mistakes are fewer and less fatal

Cons:

- Performance is worse.

Kernel Organization: Monolithic vs. Microkernel

- Linux is a mixture, mostly monolithic, but with many functions performed at the user level
- xv6 is monolithic but so small it is smaller than some microkernels.

Today: isolation

- Isolation vs. protection?

Today: isolation

- Isolation vs. protection?
    - Isolation: user programs cannot interfere with one-another.
    - Protection: user programs cannot access, e.g., memory that is not allocated to them, kernel privilege functions, etc.

Today: isolation

- What is the "unit" of isolation?

The unit of isolation: "The Process"

- Prevent process X from wrecking or spying on process Y
  - (e.g., memory, cpu, FDs, resource exhaustion)
- Prevent a process from wrecking the operating system itself
  - (i.e. from preventing kernel from enforcing isolation)
- In the face of bugs or malice
  - (e.g. a bad process may try to trick the h/w or kernel)
- If one process has a bug, it shouldn't impact others that are not its children.

- Q: can we isolate a process from kernel?

Complete Isolation

- The goal of isolation is to protect processes from one another
- Can we enforce complete isolation?

Complete Isolation

- The goal of isolation is to protect processes from one another
- Can we enforce complete isolation? No.
- The OS must also allow for two other requirements:
    - interaction between processes via pipes, shared mem, etc.
    - multiplexing processes so that all processes can appear to run at the same time even with one CPU, sleep and wakeup based on conditions set by other processes, etc.

Isolation mechanisms in operating systems

1. User/kernel mode flag (aka ring or Privilege Level)
2. Address spaces
3. Timeslicing (later)
4. System call interface

Hardware isolation in x86

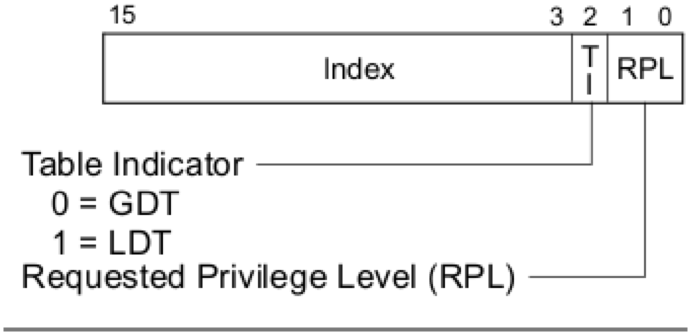

Figure 3-6. Segment Selector

- x86 support: kernel/user mode flag
- CPL (current privilege level): lower 2 bits of `%cs`
  - 0: kernel, privileged
  - 3: user, unprivileged

Hardware isolation in x86 (aka ring)

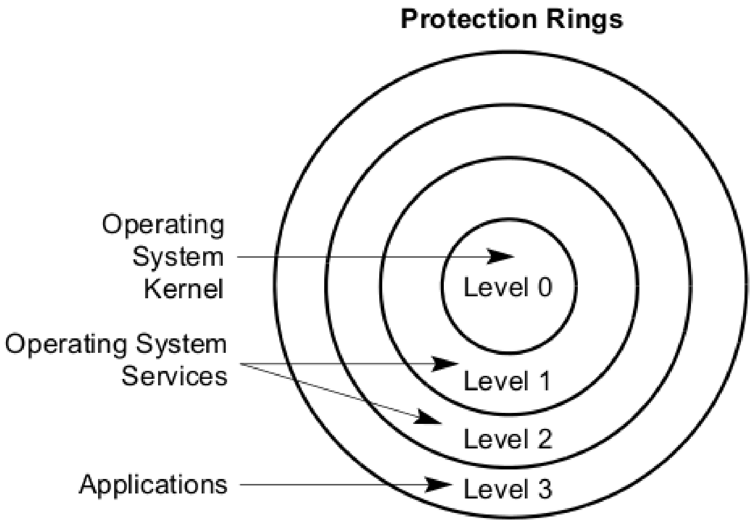

Figure 5-3. Protection Rings

What does "ring 0" protect?

- Protects everything relevant to isolation
  - writes to `%cs` (to defend CPL)
  - every memory read/write is checked for privilege level
  - I/O port accesses are privileged
  - control register accesses (`eflags`, `%cs4`, ...)
- Q: What happens if a user program attempts to execute a privileged instruction?

How to switch b/w rings (ring 0 <-> ring 3)?

- Controlled transfer: system call
  - `int` or `sysenter` instruction set CPL to 0
  - set CPL to 3 before going back to user space
  - E.g., every read or write to screen or disk requires `int` in x86.

System call handling

- Switches to a kernel determined entry point.
- Kernel must:
  - Validate the system call arguments
  - Determine if the process is allowed to perform the operation
  - Deny or execute it.

Making system calls in xv6 (usys.S)

```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name)         \
  .globl name;                \
  name:                       \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL;           \
    ret

SYSCALL(fork)
SYSCALL(exit)
...
```

Returning back to userspace (trapasm.S)

- `syscall()` -> `trapret()` -> `iret`

```
.globl trapret
trapret:
  popal
  popl %gs
  popl %fs
  popl %es
  popl %ds
  addl $0x8, %esp  ## trapno and errcode
  iret
```

How to isolate process memory?

- Idea: "address space"
  - Give each process own memory space
  - Prevent it from accessing other memory (kernel or other processes)
- x86 provides "paging hardware" (next week)
  - MMU: VA -> PA

Virtual address space in xv6

```
       +-=> +------------------+  <= 0xFFFFFFFF
       |    |                  |
       |    |    free memory   |
       |    +------------------+
kernel |    | kernel text/data |  4MB
space  |    +------------------+  <= 0x80100000
(CPL=0)|    |       BIOS       |
       +-=> +------------------+  <= 0x80000000 (KERNBASE)
       |    |       heap       |
       |    +------------------+
user   |    |       stack      |
space  |    +------------------+
(CPL=3)|    |  user text/data  |
       +-=> +------------------+  <= 0x00000000
```

How to isolate CPU?

- Prevent a process from hogging the CPU, e.g. buggy infinite loop
- Cooperative vs. uncooperative scheduling
    - Yield vs. clock driven
- xv6 relies on clock interrupt for context switching (next week)

How to represent a process in xv6 (proc.h)?

```c
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack
  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)
};
```

Code: first kernel code (entry.S)

- entry point of `kernel`
- enable paging
- setup stack
- handover to `main` in `main.c`

Code: the first process (proc.c)

- allocate a proc with `allocproc()`
- setup vm: `setupkvm()` and `inituvm()`
- setup tf to launch `initcode.S`

The first address space in xv6

```
                        +------------------+  <= 0xFFFFFFFF
                        |                  |
                        |    free memory   |
                       ++------------------+
                      / | kernel text/data |  (kernel)
                     /  +------------------+  <= 0x80100000
                     +  |       BIOS       |
   physical mem     /  ++------------------+  <= 0x80000000
                   /  / |       heap       |  (KERNBASE)
+------------------+  | +------------------+
| kernel text/data |  + |       stack      |
+------------------+ /  +------------------+
|       BIOS       |/   |  user text/data  |  (initcode)
+------------------+    +------------------+  <= 0x00000000
```

Code: a new kernel stack (proc.c)

```
+------------------+ <= proc=>kstack + KSTACKSIZE
|       esp        |
|       ...        |
|       eip        |
+------------------+ <= proc=>tf
|     trapret      |
+------------------+
|       eip ---------=> forkret
|       ...        |
+------------------+ <= proc=>context
|     (empty)      |
|                  |
+------------------+ <= proc=>kstack
```

Code: running the first process

- `mpmain()`
- `scheduler()`
- runs `initcode.S`

Code: the first system call (initcode.S)

- handover to "/init" (Q: why not just invoke "/init"?)

```
.globl start
start:
  pushl $argv  // argv[] = {init, 0}
  pushl $init  // init[] = "/init\0"
  pushl $0     // where caller pc would be
  movl $SYS_exec, %eax
  int $T_SYSCALL
```

Code: the /init process (init.c)

```
int main(void) {
  open("console", O_RDWR) // Q1?
  dup(0);                 // Q2?
  dup(0);                 // Q3?
  for(;;) {
    if (!fork())          // Q4?
      exec("sh", argv);   // Q5?
    wait();
  }
}
```

```
$ git clone git@github.gatech.edu:cs3210-spring2017/cs3210-pub
```

or

```
$ cd cs3210-pub
$ git pull
```

References

- Intel Manual
- UW CSE 451
- OSPP
- MIT 6.828
- Wikipedia
- The Internet