End semester examination

- Open book examination
- You have to bring the xv6 code listing
- Electronic devices are not allowed

Syllabus

- Everything that we have discussed so far

- demand paging and swapping
  - paging assignment, replacement policies, page tables, TLB, memory-mapped files, etc.
- locks
  - spinlocks, ticket spinlocks, readers-writer lock, big-reader lock, read-copy update, Peterson's solution, semaphores, etc.
- Filesystem
  - filesystem structure in xv6, buffer cache, in-memory inode, disk inode, synchronization, crash recovery, logging, etc.

Syllabus

- CPU cache, cache coherence, weak memory model, memory barriers, etc.
- Security in OS, temporal safety, scheduling policies
- calling conventions
- Anything else that is missing here but has been discussed in the class

How does an OS provide process isolation?

- Memory isolation
  - Using page tables

- Disallow execution of privileged instructions
  - Using protection rings

Do we need protection rings if the compiler is trusted?

- Assuming that a “C” compiler never generates a privileged instruction, can we execute the program in ring-0
  - no, because the application can modify a function pointer or the return address on the stack to jump in the middle of an instruction

Do we need protection rings if the compiler is trusted?

- Can we execute a Java program in ring-0

Can we overwrite function pointers in Java

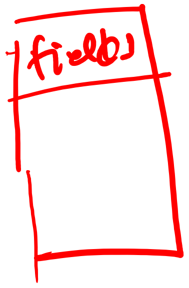
- Functions are the member of the class
    - No concept of pointer arithmetic in Java

- Typecasts of objects are always safe in Java

Typecast in Java

```
class Type1 {
    int field1;
    Type1 ();
    void foo ();
    void bar ();
}
```

```
class Type2 {
    int field1;
    int field2;
    int field3;
    Type2 ();
}
```




```
Type1 obj1 = new Type1();
Type2 obj2 = (Type2)obj1;  ×
obj2.filed3 = 0x8210023;
obj1.foo();       // obj1.foo had been probably overwritten by field3
```

not allowed in Java

How about use-after-free

```
class Type1 {
    int field1;
    Type1 ();
    void foo ();
    void bar ();
}
```

```
class Type2 {
    int field1;
    int field2;
    int field3;
    Type2 ();
}
```

```
Type1 obj1 = new Type1();        0x1234
free (obj1);   x   no free in Java.
Type2 obj2 = new Type2();        0x1234
obj1.foo();      // obj1.foo had been probably overwritten by field3
```

No free in Java

- How does Java detect free objects

· mark and sweep garbage collection
· arbitrary latencies

node

struct list ~~item~~ * node = null ;

struct list * ptr = a List;

node = ptr;

ptr = NULL;

How about overwriting return address

int[] array = new int[32];

array[34] = middle_of_some_instruction;

Not possible!

Java throws a runtime exception on out of bound array access.

We can’t execute privilege instruction in Java because

- No function pointer corruption

- No out of bound array access

How about memory isolation

- Java does not provide virtual address abstraction
- applications can access objects allocated through the memory allocation primitives
- If the allocator is functionally correct, a Java application can never access an arbitrary memory location

Memory isolation in Java

- No hardware support is needed for memory isolation
- Paging can be disabled
  - No TLB pressure, reloading of page tables, no address translation
- All applications are Java applications
- OS is also a Java application

System calls in Java OS

- use context switch API
- allocate objects in the OS address space
- copy arguments to the OS address space
- switch to the OS
  - no cr3 reload, no ring transition, direct jump to the system call handler

IPC in Java OS

- allocate objects in the target process address space
- copy arguments to the target process address space
- switch to the target process
  - no cr3 reload, no ring transition, jump to the receive routine

Device drivers can still access OS objects

- Can we do better?

Device driver isolation

- Each device driver is a sperate Java process
- A device driver can access OS objects by doing system calls similar to other processes

Java OS

- Despite all these advantages why do operating systems still prefer C

- Java is not an efficient language
  - garbage collection can cause arbitrary latencies when memory utilization is high

- An efficient and safe programming language is an important and challenging problem to solve

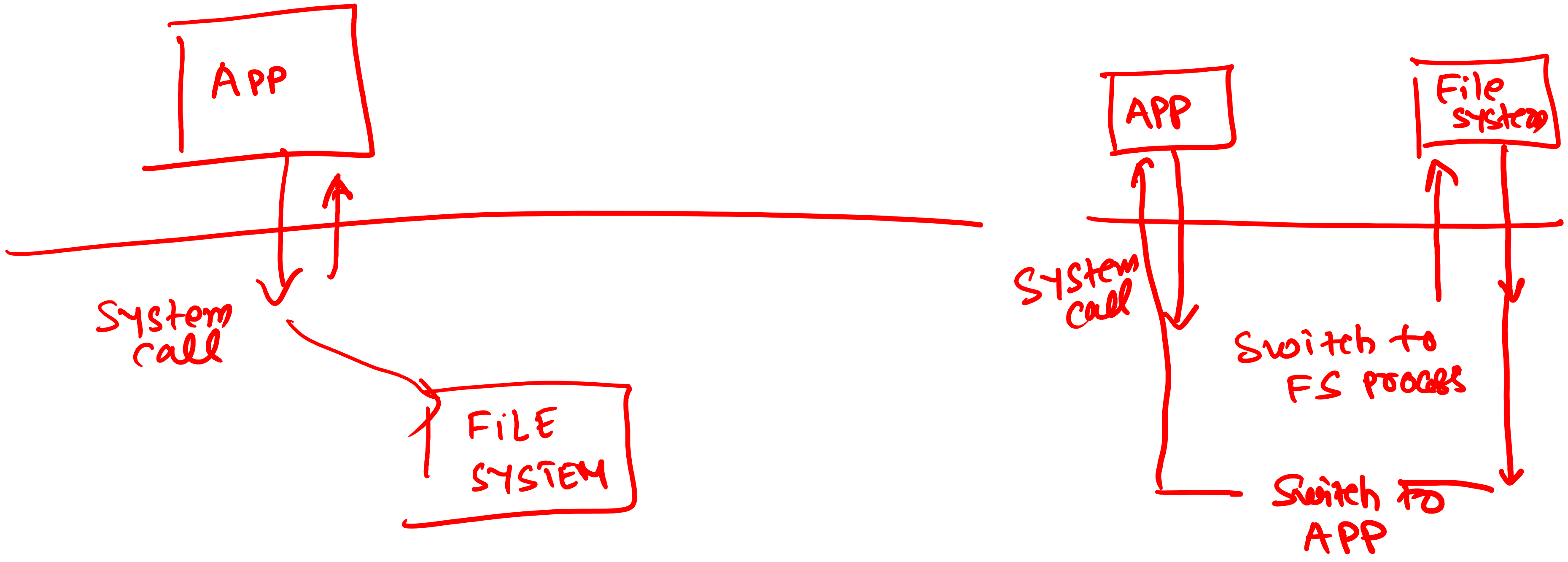
Can we design a relatively secure OS in C

- Problem
  - kernel subsystems, e.g., file system, network stack, device drivers, page fault handler, process manager, etc., can access all the memory
  - all of these together is a huge code base
  - if any of these components are vulnerable to exploits, the whole system can be compromised

Microkernel

- Each subsystem runs as a separate user-mode process
- OS does minimal work
  - address space
  - threads
  - IPC (send and receive)

Microkernel



Pros

- fault tolerance
  - OS can restart the file system process after a null pointer dereference
- security
  - A security bug in the file system won’t affect other kernel services or leak data of kernel

Cons

- System calls are slow
  - page table switch
  - additional two transitions between different privilege levels

Interesting problems

- OS security
- Cloud security
- Scalable OS
- OS for heterogenous architectures
- Distributed OS
- and many more.