things programs on portal shouldn't do

read other user's files

modify OS's memory

read other user's data in memory

hang the entire system

things programs on portal shouldn't do

read other user's files

modify OS's memory

read other user's data in memory

hang the entire system

privileged instructions

can't let any program run some instructions

example: talk to I/O device

allows machines to be shared between users (e.g. lab servers)

processor has two modes:

- kernel mode — privileged instructions work
- user mode — privileged instructions cause exception instead

only *trusted* OS code runs in kernel mode

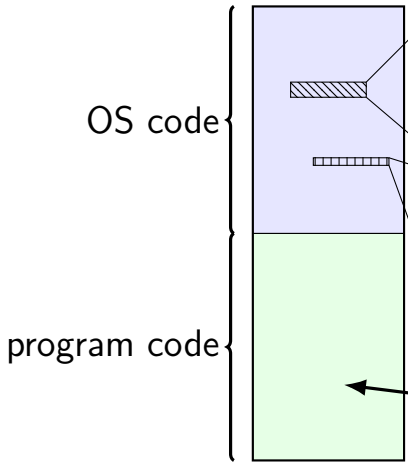
kernel mode

extra one-bit register: “are we in kernel mode”

processor switches to kernel mode to run OS

OS switches processor back to use mode when running normal code

calling the OS?




```
void readFromDiskInto(int diskLocation, char *dest) {
    ...
    runPrivilegedInstruction(...);
    ...
}
```

```
void readFileSafely(const char *name, char *dest) {
    if (canCurrentProgramCanAccessFile(name)) {
        readFromDiskInto(lookupFile(name), dest)
    }
}
```

how do we let this code call `readFromSafely` in kernel mode but not `readFromDisk`?

controlled entry to kernel mode

OS specifies where to start executing code in kernel mode

- typically set at boot
- requires privileged instructions to change

OS makes sure the code it says to start is "safe"

- (hopefully)
- example: checks whether current program is allowed to read file before reading it

Linux x86-64 system calls

special instruction: `syscall`

runs OS specified code in kernel mode

Linux syscall calling convention

before `syscall`:

`%rax` — system call number

`%rdi`, `%rsi`, `%rdx`, `%r10`, `%r8`, `%r9` — args

after `syscall`:

`%rax` — return value

on error: `%rax` contains -1 times “error number”

almost the same as normal function calls

Linux x86-64 hello world

```
.globl _start
.data
hello_str: .asciz "Hello, World!\n"
.text
_start:
  movq $1, %rax # 1 = "write"
  movq $1, %rdi # file descriptor 1 = stdout
  movq $hello_str, %rsi
  movq $15, %rdx # 15 = strlen("Hello, World!\n")
  syscall

  movq $60, %rax # 60 = exit
  movq $0, %rdi
  syscall
```

approx. system call handler

```
sys_call_table:
    .quad handle_read_syscall
    .quad handle_write_syscall
    // ...

handle_syscall:
    ... // save old PC, etc.
    pushq %rcx // save registers
    pushq %rdi
    ...
    call *sys_call_table(,%rax,8)
    ...
    popq %rdi
    popq %rcx
    return_from_exception
```

Linux system call examples

`mmap`, `brk` — allocate memory

`fork` — create new process

`execve` — run a program in the current process

`_exit` — terminate a process

`open`, `read`, `write` — access files

`socket`, `accept`, `getpeername` — socket-related

system call wrappers

can't write C code to generate syscall instruction

solution: call "wrapper" function written in assembly

things programs on portal shouldn't do

read other user's files

modify OS's memory

read other user's data in memory

hang the entire system

things programs on portal shouldn't do

read other user's files

modify OS's memory

read other user's data in memory

hang the entire system

backup slides