Discussion Week 4

TA: Kyle Dewey

Overview

- Project #1 debriefing
- System calls
- Project #2

Task 1 Bugs/Ugliness

About that Project...

- I'm not actually grading anything (probably)
- Clarity is a wonderful thing
- I apologize for any incorrect information
  - Output is **not** the same with semaphores
  - Please output information in laundromat

System Calls

Why Operating Systems?

- Multitasking
- Resource management
  - CPU
  - Memory
  - Disks
  - Printers...

Abstraction

- “I just want to print!”
- Lots of different, very similar hardware
- Unify with a common interface

Isolation

- Each process "thinks" it is the only one on the system
- Each has access to resources

Total Resource Access

- Process A prints “Hello world!”
- Process B prints “Goodbye cruel world!”

```
Hello woGoodbye crld!
ruel world!
```

Mediated Access

- Gain access through another entity
- The entity makes sure everything is isolated

Mediated Access

- Process A prints “Hello world!”
- Process B prints “Goodbye cruel world!”

```
Hello world!
Goodbye cruel world!
```

“Entity”

- The entity is the OS
- The pathway for mediation is a system call
- System calls allow processes to communicate with the OS

Syscall Frequency

- Any I/O (network, disk, ...)
- Any process manipulation
- Interprocess communication
- Shared library access
- Essentially access to any shared resource

Tools

- `strace`: Linux tool for intercepting syscalls
- `truss`: Solaris/BSD tool for intercepting syscalls
- **Usage:** `strace ./a.out`

“Useless” C Program

C Hello World

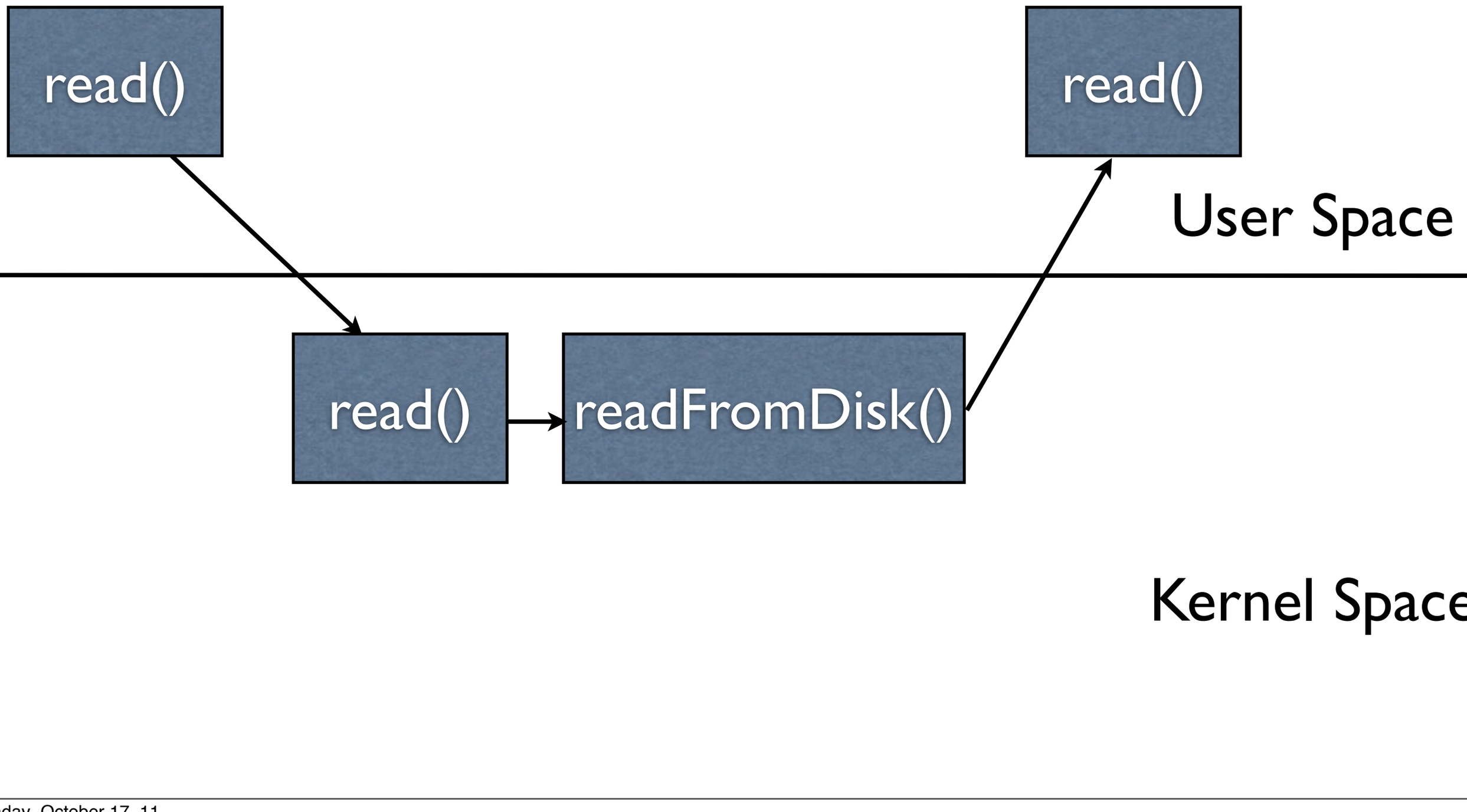
Java Hello World (Note -F -f was needed)

Python Hello World

The Point

- Syscalls are made all over the place
- Rarely, if ever, directly called in actual code
  - Unwieldy
  - Layer of abstraction

System Call Execution



Why Kernel Space?

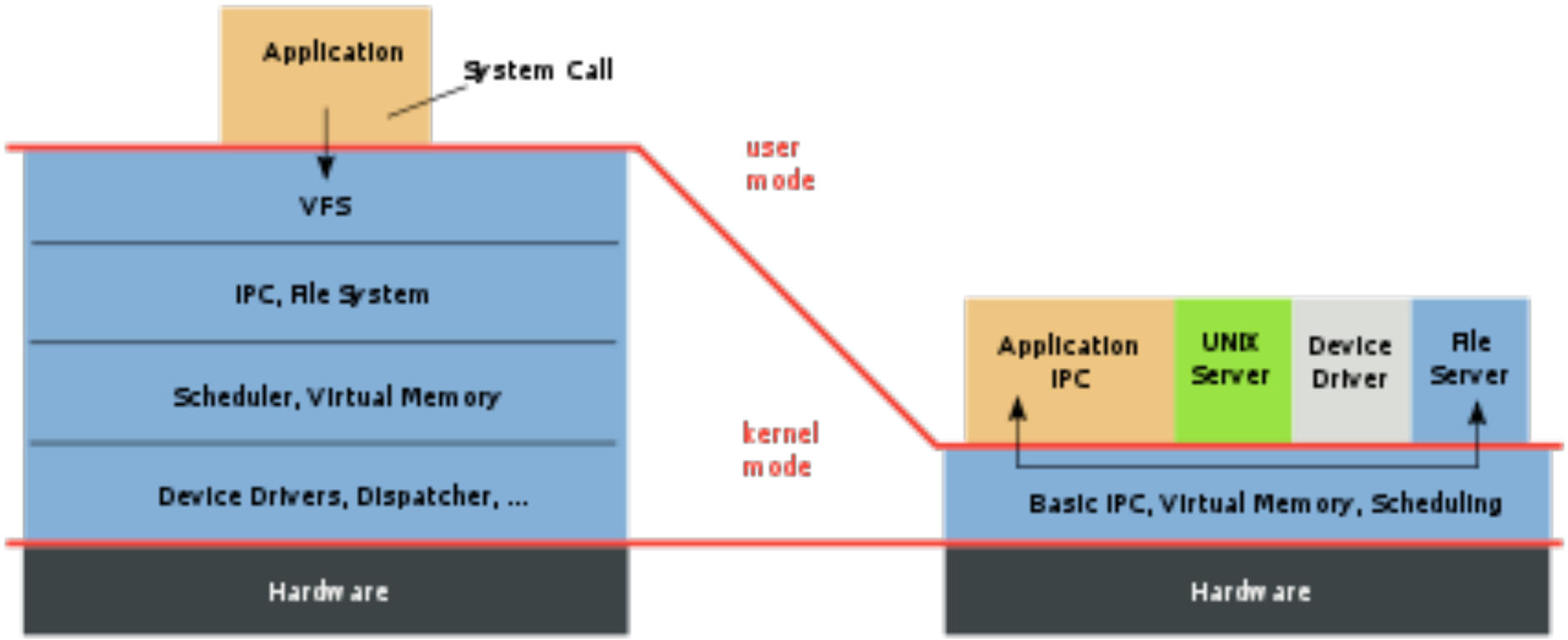
- Kernel executes call "on behalf" of a process
  - If a normal process could do it, then there is no isolation
    - Possible to have more than just "kernel level" and "user level"

Something in Between - Microkernels

- Goal: put as much of the kernel as possible in user space
- Turns out a lot can be done in user space

Microkernel

Monolithic Kernel
based Operating System



Microkernel
based Operating System

Project #2 Part 1

Basic Idea

- Implement system calls for basic process and file manipulation
- Note that “basic” means the base of everything - not simple!

Syscall Naming in NACHOS

- Names shared with threads implementation
- These are **very** different, though you may need the corresponding thread operations

`Fork( func )`

- Copies address space of caller
- Creates a new process in this copied address space
- Executes the given function, with this new process in the new address space
- Note the dissimilarity to UNIX's `fork()`

`Fork()` Example

Yield()

- Temporary yields the calling process to any other processes available to run

`Exit( int )`

- Terminates the calling thread
- The parameter is the exit status (ignored for this project)

`Exec( filename )`

- Spawns a new process
- Executes the code specified in `filename` using the new process
- Note this does not clobber the calling process, as it does with UNIX

`Join( SpaceId )`

- Waits for the process with the given `SpaceId`
- The calling process blocks until the process backing `SpaceId` returns

Project #2 Part 2

NACHOS Filesystem

- Under Linux, it's simply a single big file
- Under NACHOS, it contains NACHOS' directory hierarchy
- Unless otherwise mentioned, the slides refer to the NACHOS hierarchy

UNIX Similarity

- NACHOS is modeled after UNIX
- Some files are actually device interfaces

Filesystem Stubs

- Some stubs are provided that may help
- Extremely basic and limited
- May need to scrap entirely

`Create( name )`

- Create a new, empty file with the given name
- Note that you need to be able to extend file lengths - `FileSystem::Create` indicates an issue with this

`Open( name )`

- Opens the file with the given name
- Returns `NULL` if it does not exist

Close(OpenFileId)

- Closes the file denoted by the given open file ID

`ReadAt( buffer, size, pos )`

- **Reads** `size` **bytes starting at** `pos` **into** `buffer`
- Note that this is a method on the `OpenFile` **object**

`WriteAt( buffer, size, pos )`

- **Write** `size` **bytes from** `buffer`**, starting at** `pos`
- Note that this is a method on the `OpenFile` **object**

Project #2 Notes

- Far more provided detail
- Well-defined outputs
- Due November 8 at midnight
- **Much** more difficult than project #1 (worth twice as much, too)