

TCCS 422: OPERATING SYSTEMS

Beyond
Physical Memory

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma



FEEDBACK – 5/30

- In solving HW3 with one producer and two consumer threads with synchronized access to the bounded buffer via single mutex without using signal and conditions, my program is working as expected. I am wondering why would I use signal and condition? It would be a great help if you explain me that a little more...
- With a large bounded buffer, and a small number of tasks, the buffer will never fill to stress the system
- Try:
 - Shrinking bounded buffer to a very small size
 - Dramatically increasing the number (and complexity) of tasks


May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.2

EVALUATING LOCK IMPLEMENTATIONS

- Correctness
 - Does the lock work?
 - Are critical sections mutually exclusive? (atomic-as a unit?)
- Fairness
 - Are threads competing for a lock have a fair chance of acquiring it?
- Overhead



Recall this earlier SLIDE

April 18, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.38

OBJECTIVES – 5/30

- HW3 Questions
- Finish Chapter 20
- Chapter 21 – Swap memory - paging to disk
- Chapter 22 – Page replacement algorithms
- Introduce: Chapter 36 – I/O Devices
- Thursday June 1st
 - Finish Chapter 36 – I/O Devices
 - Chapter 37 – Hard Disk Drives
 - Chapter 39 – File Systems
 - Practice Final Exam

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.4

OBJECTIVES

- Chapter 21
 - Virtual “Swap” Memory
- Chapter 22
 - Page replacement algorithms
 - Replacement algorithm effectiveness

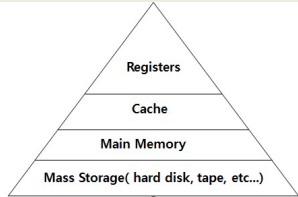
May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.5

MEMORY HIERARCHY

- Disks (HDD, SSD) provide another level of storage in the memory hierarchy



Memory Hierarchy in modern system

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.6

MOTIVATION FOR
EXPANDING THE ADDRESS SPACE

- Can provide illusion of an address space larger than physical RAM
- For a single process
 - Convenience
 - Ease of use
- For multiple processes
 - Large virtual memory space for many concurrent processes

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.7

LATENCY TIMES

- Design considerations
 - SSDs 4x the time of DRAM
 - HDDs 80x the time of DRAM

Action	Latency (ns)	(µs)	
L1 cache reference	0.5ns		
L2 cache reference	7 ns		14x L1 cache
Mutex lock/unlock	25 ns		
Main memory reference	100 ns		20x L2 cache, 200x L1
Read 4K randomly from SSD*	150,000 ns	150 µs	~1GB/sec SSD
Read 1 MB sequentially from memory	250,000 ns	250 µs	
Read 1 MB sequentially from SSD*	1,000,000 ns	1,000 µs	1 ms ~1GB/sec SSD, 4X memory
Read 1 MB sequentially from disk	20,000,000 ns	20,000 µs	20 ms 80x memory, 20X SSD

- Latency numbers every programmer should know
- From: <https://gist.github.com/jboner/2841832#file-latency-txt>

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.8

SWAP SPACE

- Disk space for storing memory pages
- “Swap” them in and out of memory to disk as needed

The diagram illustrates the mapping between physical memory and swap space. Physical memory is divided into four pages (PFN 0-3), each containing a process (Proc 0-3). Swap space is divided into eight blocks (Block 0-7). Block 0 contains Proc 0, Block 1 contains Proc 0, Block 2 is free, Block 3 contains Proc 1, Block 4 contains Proc 1, Block 5 contains Proc 3, Block 6 contains Proc 2, and Block 7 contains Proc 3.

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.9

PAGE LOCATION

- Page table pages are:
 - Stored in memory
 - Swapped to disk
- Present bit
 - In the page table entry (PTE) indicates if page is present
- Page fault
 - Memory page is accessed, but has been swapped to disk

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.10

PAGE FAULT

- OS steps in to handle the page fault
- Loading page from disk requires a free memory page
- Page-Fault Algorithm

```
1: PFN = FindFreePhysicalPage()
2: if (PFN == -1) // no free page found
3:     PFN = EvictPage() // run replacement algorithm
4: DiskRead(PTE.diskaddr, pfn) // sleep (waiting for I/O)
5: PTE.present = True // set PTE bit to present
6: PTE.PFN = PFN // reference new loaded page
7: RetryInstruction() // retry instruction
```

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.11

PAGE REPLACEMENTS


- Page daemon
 - Background threads which monitors swapped pages
- Low watermark (LW)
 - Threshold for when to swap pages to disk
 - Daemon checks: free pages < LW
 - Begin swapping to disk until reaching the highwater mark
- High watermark (HW)
 - Target threshold of free memory pages
 - Daemon free until: free pages >= HW

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.12

REPLACEMENT
POLICIES


POLICY
CHANGES

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.13

CACHE MANAGEMENT

- Replacement policies apply to “any” cache
- Goal is to minimize the number of misses
- Average memory access time can be estimated:

$$AMAT = (P_{hit} * T_M) + (P_{miss} * T_D)$$

Argument	Meaning
T_M	The cost of accessing memory (time)
T_D	The cost of accessing disk (time)
P_{hit}	The probability of finding the data item in the cache(a hit)
P_{miss}	The probability of not finding the data in the cache(a miss)

- Consider $T_M = 100\text{ ns}$, $T_D = 10\text{ms}$
- Consider $P_{hit} = .9\text{ (90\%)}$, $P_{miss} = .1$
- Consider $P_{hit} = .999\text{ (99.9\%)}$, $P_{miss} = .001$

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.14

OPTIMAL REPLACEMENT POLICY

- What if:
 - We could predict the future (... with a magical oracle)
 - All future page accesses are known
 - Always replace the page in the cache used farthest in the future
- Used for a comparison
- Provides a “best case” replacement policy
- Consider a 3-element empty cache with the following page accesses:
0 1 2 0 1 3 0 3 1 2 1

What is the hit/miss ratio?
6 hits

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.15

FIFO REPLACEMENT

- Queue based
- Always replace the oldest element
- Simple to implement
- Doesn't consider importance... just arrival ordering
- Consider a 3-element empty cache with the following page accesses:
0 1 2 0 1 3 0 3 1 2 1

4 hits

LRU incorporates history
- What is the hit/miss ratio?
- How is FIFO different than LRU?

May 30, 2017

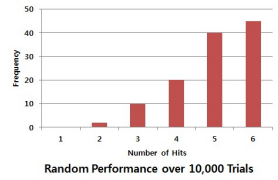
TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.16

RANDOM REPLACEMENT

- Pick a page at random to replace
- Simple and fast implementation
- Performance depends on luck of random choices

0 1 2 0 1 3 0 3 1 2 1


Random Performance over 10,000 Trials

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.17

HISTORY-BASED POLICIES

- LRU: Least recently used
- Always replace page with oldest access time
- Consider when a page was last accessed

0 1 2 0 1 3 0 3 1 2 1

What is the hit/miss ratio?
6 hits

- LFU: Least frequently used
- Always replace page with fewest accesses
- Consider frequency of page accesses

0 1 2 0 1 3 0 3 1 2 1

Hit/miss ratio is=
6 hits

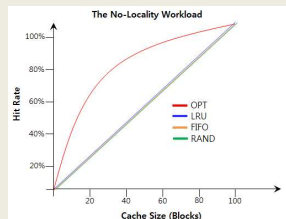
May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.18

WORKLOAD EXAMPLES: NO-LOCALITY

- No-Locality (Random Access) Workload
 - Perform 10,000 random page accesses
 - Across set of 100 memory pages



When the cache is large enough to fit the entire workload, it doesn't matter which policy you use.

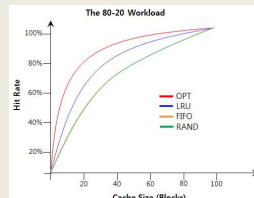
May 30, 2017

TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.19

WORKLOAD EXAMPLES: 80/20

- 80/20 Workload
 - Perform 10,000 page accesses, against set of 100 pages
 - 80% of accesses are to 20% of pages (hot pages)
 - 20% of accesses are to 80% of pages (cold pages)



LRU is more likely to hold onto hot pages (recalls history)

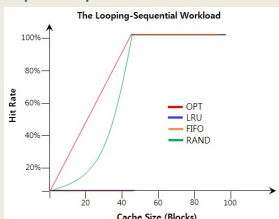
May 30, 2017

TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.20

WORKLOAD EXAMPLES: SEQUENTIAL

- Looping sequential workload
 - Refer to 50 pages in sequence: 0, 1, ..., 49
 - Repeat loop



Random performs better than FIFO and LRU for cache sizes < 50

Algorithms should provide "scan resistance"

May 30, 2017

TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.21

IMPLEMENTING LRU

- Implementing last recently used (LRU) requires tracking access time for all system memory pages
 - Times can be tracked with a list
 - For cache eviction, we must scan an entire list
 - Consider: 4GB memory system (2^{32}), with 4KB pages (2^{12})
 - This requires 2^{20} comparisons !!!
 - Simplification is needed
 - Consider how to approximate the oldest page access

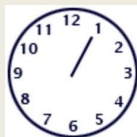
May 30, 2017

TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.22

IMPLEMENTING LRU - 2

- Harness the Page Table Entry (PTE) Use Bit
 - HW sets to 1 when page is used
 - OS sets to 0
- Clock algorithm (approximate LRU)
 - Refer to pages in a circular list
 - Clock hand points to current page
 - Loops around
 - IF USE_BIT=1 set to USE_BIT = 0
 - IF USE_BIT=0 replace page



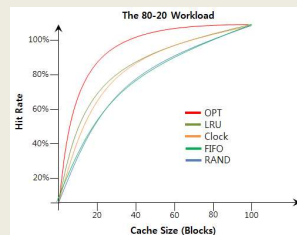
May 30, 2017

TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.23

CLOCK ALGORITHM

- Not as efficient as LRU, but better than other replacement algorithms that do not consider history



May 30, 2017

TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.24

CLOCK ALGORITHM - 2

- Consider dirty pages in cache
- If DIRTY (modified) bit is FALSE
 - No cost to evict page from cache
- If DIRTY (modified) bit is TRUE
 - Cache eviction requires updating memory
 - Contents have changed
- Clock algorithm should favor no cost eviction

May 30, 2017

TCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.25

WHEN TO LOAD PAGES

- On demand → demand paging
- Prefetching
 - Preload pages based on anticipated demand
 - Prediction based on locality
 - Access page P, suggest page P+1 may be used
- What other techniques might help anticipate required memory pages?
 - Prediction models, historical analysis
 - In general: accuracy vs. effort tradeoff
 - High analysis techniques struggle to respond in real time

May 30, 2017

TCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.26

OTHER SWAPPING POLICIES

- Page swaps / writes
 - Group/cluster pages together
 - Collect pending writes, perform as batch
 - Grouping disk writes helps amortize latency costs
- Thrashing
 - Occurs when system runs many memory intensive processes and is low in memory
 - Everything is constantly swapped to-and-from disk

May 30, 2017

TCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.27

OTHER SWAPPING POLICIES - 2

- Working sets
 - Groups of related processes
 - When thrashing: prevent one or more working set(s) from running
 - Temporarily reduces memory burden
 - Allows some processes to run, reduces thrashing

May 30, 2017

TCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.28

I/O DEVICES



May 30, 2017

TCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.29

OBJECTIVES

- Chapter 36
 - Polling vs Interrupts
 - Programmed I/O (PIO)
 - Direct memory Access (DMA)
 - Port-mapped I/O (PMIO)
 - Memory-mapped I/O (MMIO)

May 30, 2017

TCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.30

I/O DEVICES

- Modern computer systems interact with a variety of devices

input **output**

May 30, 2017 TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma L17.31

COMPUTER SYSTEM ARCHITECTURE

Prototypical System Architecture

VERY FAST: CPU is attached to main memory via a **Memory bus** (proprietary).
FAST: High speed devices (e.g. video) are connected via a **General I/O Bus** (e.g. PCI).
SLOWER: Disks are connected via a **Peripheral I/O Bus** (e.g. SCSI, SATA, USB).

May 30, 2017 TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma L17.32

I/O BUSES

- Buses**
 - Buses closer to the CPU are faster
 - Can support fewer devices
 - Further buses are slower, but support more devices
- Physics and costs dictate "levels"
 - Memory bus
 - General I/O bus
 - Peripheral I/O bus
- Tradeoff space: speed vs. locality

May 30, 2017 TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma L17.33

CANONICAL DEVICE

- Consider an arbitrary canonical device

Canonical Device

- Two primary components**
 - Interface (registers for communication)
 - Internals: Local CPU, memory, specific chips, firmware (embedded software)

May 30, 2017 TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma L17.34

CANONICAL DEVICE: HARDWARE INTERFACE

- Status register**
 - Maintains current device status
- Command register**
 - Where commands for interaction are sent
- Data register**
 - Used to send and receive data to the device

General concept:
The OS interacts and controls device behavior by reading and writing the device registers.

May 30, 2017 TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma L17.35

OS DEVICE INTERACTION

- Common example of device interaction

```

while ( STATUS == BUSY)      ← Poll-Is device available?
; //wait until device is not busy
write data to data register      ← Command parameterization
write command to command register      ← Send command
Doing so starts the device and executes the command
while ( STATUS == BUSY)      ← Poll-Is device done?
; //wait until device is done with your request
    
```

May 30, 2017 TCS5422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma L17.36

POLLING

- OS checks if device is *READY* by repeatedly checking the *STATUS* register
 - Simple approach
 - CPU cycles are wasted without doing meaningful work
 - Ok if only a few cycles, for rapid devices that are often *READY*
 - BUT polling, as with “spin locks” we understand is inefficient

CPU utilization by polling

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.37

INTERRUPTS VS POLLING

- For longer waits, put process waiting on I/O to sleep
- Context switch (C/S) to another process
- When I/O completes, fire an interrupt to initiate C/S back
 - Advantage: better multi-tasking and CPU utilization
 - Avoids: unproductive CPU cycles (polling)

Diagram of CPU utilization by interrupt

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.38

INTERRUPTS VS POLLING - 2

What is the tradeoff space ?

- Interrupts are not always the best solution
 - How long does the device I/O require?
 - What is the cost of context switching?

If device I/O is fast → polling is better.
If device I/O is slow → interrupts are better.

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.39

INTERRUPTS VS POLLING - 3

- One solution is a two-phase hybrid approach
 - Initially poll, then sleep and use interrupts
- Deadlock problem
 - Common with network I/O
 - Many arriving packets generate *many many* interrupts
 - Overloads the CPU!
 - No time to execute code, just interrupt handlers !
- Deadlock optimization
 - Coalesce multiple arriving packets (for different processes) into fewer interrupts
 - Must consider number of interrupts a device could generate

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.40

DEVICE I/O

- To interact with a device we must send/receive **DATA**
- There are two general approaches:
 - Programmed I/O (PIO)
 - Direct memory access (DMA)

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.41

Transfer Modes			
Mode	#	Maximum transfer rate (MB/s)	cycle time
PIO	0	3.3	600 ns
	1	5.2	383 ns
	2	8.3	240 ns
	3	11.1	180 ns
	4	16.7	120 ns
Single-word DMA	0	2.1	960 ns
	1	4.2	480 ns
	2	8.3	240 ns
Multi-word DMA	0	4.2	480 ns
	1	13.3	150 ns
	2	16.7	120 ns
	3 ^[34]	20	100 ns
	4 ^[34]	25	80 ns
Ultra DMA	0	16.7	240 ns + 2
	1	25.0	160 ns + 2
	2 (Ultra ATA/33)	33.3	120 ns + 2
	3	44.4	90 ns + 2
	4 (Ultra ATA/66)	66.7	60 ns + 2
	5 (Ultra ATA/100)	100	40 ns + 2
	6 (Ultra ATA/133)	133	30 ns + 2
	7 (Ultra ATA/167) ^[35]	167	24 ns + 2

From https://en.wikipedia.org/wiki/Parallel_ATA

PROGRAMMED I/O (PIO)

- Spend CPU time to perform I/O
- CPU is involved with the data movement (input/output)
- PIO is slow –CPU is occupied with meaningless work

PIO

“over-burdened”

1 : task 1 2 : task 2
C : copy data from memory

CPU: 1 1 1 1 C C C 2 2 2 2 2 2 1 1 1

Disk: 1 1 1 1 1

Diagram of CPU utilization

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.43

PIO DEVICES

- Legacy serial ports
- Legacy parallel ports
- PS/2 keyboard and mouse
- Legacy MIDI, joysticks
- Old network interfaces

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.44

DIRECT MEMORY ACCESS (DMA)

- Copy data in memory by offloading to a “DMA controller”
- Many devices (including CPUs) have DMA controllers
- Give DMA memory address, size, and copy instruction
- DMA performs I/O independent of the CPU

1 : task 1 2 : task 2
C : copy data from memory

CPU: 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1

DMA: C C C

Disk: 1 1 1 1 1

Diagram of CPU utilization by DMA

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.45

DEVICE INTERACTION

- Two primary methods
- Port mapped I/O (PMIO)
- Memory mapped I/O (MMIO)

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.46

PORT MAPPED I/O (PMIO)

- Device specific CPU I/O Instructions
- Follows a CISC model: extra instructions
- x86-x86-64: `in` and `out` instructions
- `outb`, `outw`, `outl`
- 1, 2, 4 byte copy from EAX → device's I/O port

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.47

MEMORY MAPPED I/O (MMIO)

- Device's memory is mapped to CPU memory
- Tenet of RISC CPUs: instructions are eliminated, CPU is simpler
- Old days: 16-bit CPUs didn't have a lot of spare memory space
- Today's CPUs: 32-bit (4GB addr space) & 64-bit (128 TB addr space)
- Regular CPU instructions used to access device: mapped to memory
- Devices monitor CPU address bus and respond to their addresses
- I/O device address areas of memory are **reserved** for I/O
 - Must not be available for normal memory operations.

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.48

DEVICE INTERACTION

- The OS must interact with a variety of devices
- Example: for DISK I/O consider the variety of disks:
- SCSI, IDE, USB flash drive, DVD, etc.
- Device drivers use abstraction to provide general interfaces for vendor specific hardware
- In Linux: block devices

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.49

FILE SYSTEM ABSTRACTION

- Layers of I/O abstraction in Linux
- C functions (open, read, write) issue **block read and write** requests to the generic block layer

The diagram illustrates the File System Stack, showing the flow of data from an application to the hardware. It is divided into two sections: 'user' and 'kernel'. The 'user' section contains the 'Application' layer. The 'kernel' section contains four layers: 'File System', 'Generic Block Interface [block read/write]', 'Generic Block Layer', and 'Device Driver [SCSI, ATA, etc]'. A dashed line separates the user and kernel sections. The 'Application' layer is connected to the 'File System' layer via the 'POSIX API [open, read, write, close, etc]'. The 'File System' layer is connected to the 'Generic Block Interface' layer via the 'Generic Block Interface [block read/write]'. The 'Generic Block Interface' layer is connected to the 'Generic Block Layer' layer via the 'Specific Block Interface [protocol-specific read/write]'. The 'Generic Block Layer' layer is connected to the 'Device Driver' layer via the 'Specific Block Interface [protocol-specific read/write]'. The 'Device Driver' layer is connected to the hardware.

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.50

FILE SYSTEM ABSTRACTION ISSUES

- **Too much abstraction**
- Many devices provide special capabilities
- Example: SCSI Error handling
- SCSI devices provide extra detail which are lost to the OS
- **Buggy device drivers**
- 70% of OS code is in device drivers
- Device drivers are required for every device plugged in
- Drivers are often 3rd party, which is not quality controlled at the same level as the OS (Linux, Windows, MacOS, etc.)

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.51

QUESTIONS

A large, stylized blue question mark icon with a thick black outline, centered on a blue background.

May 30, 2017

TCCS422: Operating Systems [Spring 2017]
Institute of Technology, University of Washington - Tacoma

L17.52