


# TCSS 422: OPERATING SYSTEMS

## Beyond Physical Memory, I/O Devices



**Wes J. Lloyd**  
School of Engineering and Technology,  
University of Washington - Tacoma

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

## FEEDBACK FROM 12/3

- Program 3
- Write to a proc file?
- Once we have a reference to a process, we then traverse pages on that process?

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.2

## FEEDBACK - 2

- Which I/O Devices work better with interrupts (other than keyboard)?
- **Interrupt driven I/O - - is off-loaded from the CPU**
  - Via Directory Memory Access (DMA) controller
  - CPU non involved in the data transfer
  - Interrupts enable a context-switch to notify data is available
  - Examples: ISA, PCI bus
- **Polled I/O is - - programmed I/O**
- Data transfers fully occupy CPU for entire data transfer
- CPU unavailable for other work
- Examples: ATA (parallel ports), legacy serial/parallel ports, PS/2 keyboard/mouse, MIDI, joysticks

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.3

## FEEDBACK - 3

- Does the mouse use interrupts, polling, or a hybrid of both?
  - Interrupts
  - Where is the polling (BUSY) process? (see top -d .1)

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma


L19.4

# CLOUD AND DISTRIBUTED SYSTEMS RESEARCH




## CLOUD AND DISTRIBUTED SYSTEMS LAB

WES LLOYD, [WLLOYD@UW.EDU](mailto:WLLOYD@UW.EDU),  
[HTTP://FACULTY.WASHINGTON.EDU/WLLOYD](http://FACULTY.WASHINGTON.EDU/WLLOYD)



- **Serverless Computing (FaaS):**
  - *How should cloud native applications be composed from microservices to optimize performance and cost? Code structure directly influences hosting costs.*
    - Service composition, performance and cost optimization/modeling/analytics, Application migration, Mitigation of Platform limitations, Influencing infrastructure, Lambda@Edge
- **Containerization (Docker):**
  - *How should containers and container platforms be leveraged and managed to optimize performance, reduce costs, and maximize server utilization?*
    - Containers, container orchestration frameworks, resource allocation, checkpointing
- **Infrastructure-as-a-Service (IaaS) Cloud:**
  - *How should applications and workloads be deployed to optimize performance and cost? There are many “knobs”, configuration options to consider.*
    - Application/workload deployment, performance and cost optimization/modeling/analytics, infrastructure management, resource contention detection/mitigation, HW heterogeneity



## OBJECTIVES

- Review Quiz 5
- Program 3 Questions
- Practice Final – 12/5
  
- Device I/O
- Chapter 36 – I/O Devices
- Chapter 37 – Hard Disk Drives

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.7

## CHAPTER 36: I/O DEVICES



December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.8

## OBJECTIVES

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

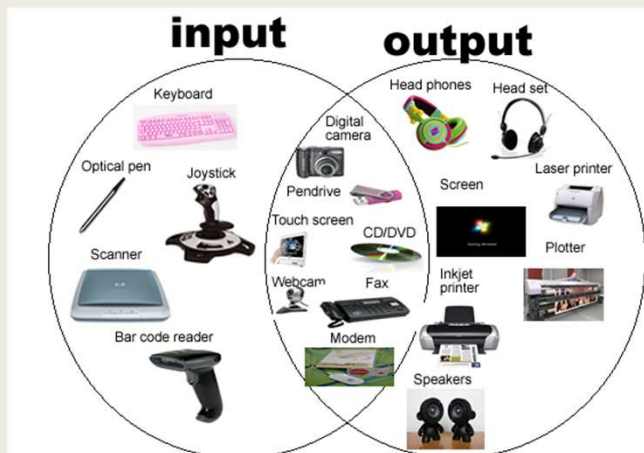
December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.9

## I/O DEVICES

- Modern computer systems interact with a variety of devices



December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.10

## COMPUTER SYSTEM ARCHITECTURE

The diagram illustrates a prototypical system architecture. At the top, a CPU and Memory are connected to a Memory Bus (proprietary). Below this, a Graphics component is connected to a General I/O Bus (e.g., PCI). At the bottom, four disks are connected to a Peripheral I/O Bus (e.g., SCSI, SATA, USB). The buses are shown as horizontal lines with arrows indicating bidirectional communication.

**Prototypical System Architecture**

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

December 5, 2018	TCCS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.11
------------------	---	--------

## 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

December 5, 2018	TCCS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.12
------------------	---	--------

## CANONICAL DEVICE

- Consider an arbitrary canonical **“standard/generic”** device

Registers:    Status    Command    Data    interface

---

Micro-controller(CPU)  
Memory (DRAM or SRAM or both)  
Other Hardware-specific Chips    internals

**Canonical Device**

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

December 5, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.13
------------------	---	--------

## 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.

December 5, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.14
------------------	---	--------

## 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
    
```

December 5, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.15
------------------	---	--------

## 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

The diagram shows a timeline of CPU cycles. The top row is labeled 'CPU' and contains a sequence of boxes: five '1's, five 'p's, and five '1's. Above the 'p's, a red dashed arrow labeled "waiting IO" spans from the first 'p' to the last 'p'. To the right, a legend shows a box with '1' labeled ': task 1' and a box with 'p' labeled ': polling'. Below the CPU row is a row labeled 'Disk' with five '1's. Below the diagram is the caption 'CPU utilization by polling'.

December 5, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.16
------------------	---	--------



## 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)

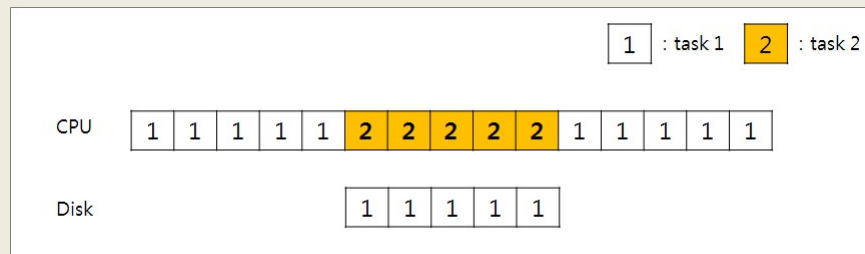


Diagram of CPU utilization by interrupt

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.17

## 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.

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.18

## INTERRUPTS VS POLLING - 3

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

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.19

## 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)

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.20

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 <sup>[34]</sup>	20	100 ns
	4 <sup>[34]</sup>	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) <sup>[35]</sup>	167	24 ns + 2

From [https://en.wikipedia.org/wiki/Parallel\\_ATA](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**

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

CPU    1 1 1 1 C C C 2 2 2 2 2 1 1 1

Disk                    1 1 1 1 1

"over-burdened"

**Diagram of CPU utilization**

December 5, 2018

TCCS422: Operating Systems [Fall 2018]  
 School of Engineering and Technology, University of Washington - Tacoma

L19.22

## PIO DEVICES

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

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.23

## PROGRAMMED I/O DEVICE (PIO) INTERACTION

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

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.24

## 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

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.25

## 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.

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.26

## DIRECT MEMORY ACCESS (DMA)

- Copy data in memory by **offloading** to “DMA controller”
- Many devices (including CPUs) integrate DMA controllers
- CPU gives DMA: memory address, size, and copy instruction
- DMA performs I/O independent of the CPU
- DMA controller generates CPU interrupt when I/O completes

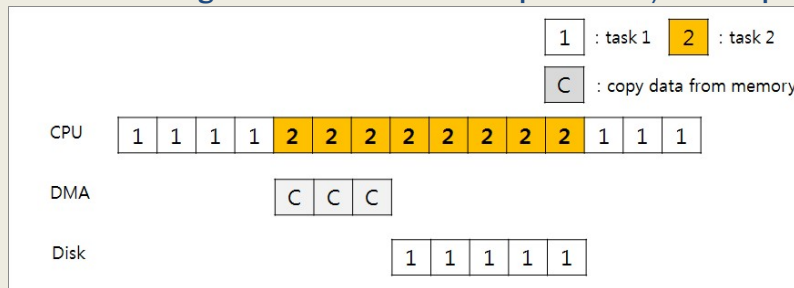


Diagram of CPU utilization by DMA

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.27

## DIRECTORY MEMORY ACCESS – 2

- Many devices use DMA
  - HDD/SSD controllers (ISA/PCI)
  - Graphics cards
  - Network cards
  - Sound cards
  - Intra-chip memory transfer for multi-core processors
- DMA allows computation and data transfer time to proceed in parallel

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.28

## 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

December 5, 2018	TCCS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.29
------------------	---	--------

## 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 user space to kernel space and back. A dashed horizontal line separates the user space (top) from the kernel space (bottom). In the user space, there is an 'Application' box. In the kernel space, there are four stacked boxes: 'File System', 'Generic Block Layer', 'Specific Block Interface [protocol-specific read/write]', and 'Device Driver [SCSI, ATA, etc]'. A 'POSIX API [open, read, write, close, etc]' box is positioned between the Application and File System layers. A 'Generic Block Interface [block read/write]' box is positioned between the File System and Generic Block Layer layers. A 'Specific Block Interface [protocol-specific read/write]' box is positioned between the Generic Block Layer and Device Driver layers. Arrows indicate the flow of data from the Application through the POSIX API, File System, Generic Block Layer, Specific Block Interface, and Device Driver.

**The File System Stack**

December 5, 2018	TCCS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L19.30
------------------	---	--------

## 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 3<sup>rd</sup> party, which is not quality controlled at the same level as the OS (Linux, Windows, MacOS, etc.)

December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.31

## QUESTIONS



December 5, 2018

TCSS422: Operating Systems [Fall 2018]  
School of Engineering and Technology, University of Washington - Tacoma

L19.3  
2