CS 3423 Operating Systems

Fall Semester 2021

Prof. Pai H. Chou

Weekly Review 13

Scope: Chapter 12, I/O Systems and   
(part of) Chapter 13, File System Interface

## 1. Definitions and Short Answers

1. In terms of I/O systems,
   1. what is a **port** of a device?

A: A port is an endpoint for a device to connect to another device.

* 1. what is a **bus**?

A: A bus is a shared medium between two or possibly more connected devices. Shared group of wires for connecting ports.

* 1. what is a **daisy chain**?

A: A daisy chain is a wiring scheme in which multiple devices are wired together in a sequence or ring.

* 1. what does a **controller** do?

A: Its job is to deal with the interface specific aspects of the device. Operates on port, bus, or device.

1. What are four kinds of **device registers** that need to be accessed by the host for the following operations?

A: data-in (Read by host), data-out (Written by host), status (read by host to find I/O status, error) and control (full/half duplex, parity, baud rate).

* 1. How does a host receive data?

A: Read from the data-in register.

* 1. How does a host send data?

A: Write to the data-out register.

* 1. How does a host find the status or error of a device?

A: Read the status register.

* 1. How does a host change settings on a device?

A: Host must write the desired configuration to the control register.

1. If a processor supports **I/O instructions**, what kind of I/O is it called?  
   A: These instructions are called direct I/O instructions and are part of the ISA. They cause a waveform to be generated by I/O.
2. If a processor does **not** support I/O instructions, can it perform I/O? If so, what is it called, and what kind of instruction does it use to perform I/O? Or if not, why not?

A: Yes, it can through memory mapped I/O. In this case we map device control registers into the address space of CPU. Load/store instructions like regular memory but we access device data and command registers.

1. Does the processor in a personal computer "see" the PCI or PCIe bus directly? If not, what does it have to go through?

A: The CPU doesn’t see the PCIe bus directly every single cycle. It must go through a bridge or a memory controller

1. How is a hard drive with a SATA interface connected to a PC with a PCIe bus?

A: Through a SATA PCIe expansion card. It is also known as “SATA Host Bus Adapter (HBA)”.

1. How is a USB keyboard connected to a PC with a PCIe bus?

A: The USB port is first connected to an expansion bus and the expansion

Bus itself is connected to an expansion bus interface.

1. Is **polling** more suitable for slow or fast devices? Why?

A: Polling is reasonable for fast devices.

1. Can polling be made more efficient by *context switching* to another process between two status checks? What potential problems may happen?

A: We can assume it will be more efficient if the other useful work to be done fast, otherwise we could miss a cycle and overwrite/lose data. In general, it is not a good idea and Interrupts are preferred.

1. What is the meaning of **interrupt chaining**? What is a good reason for it?  
   A: In interrupt chaining multiple devices share the same IRQ and ISR. Once invoked, the ISR must query each shared device. It is a good idea because it provides a compromise between the overhead of a huge interrupt table and the inefficiency of dispatching to a single interrupt handler.
2. What are examples of **exceptions** that an OS handles? What kind of mechanism is used for an OS to handle an exception?

A: Examples include divide by zero, segmentation fault, attempt to execute a privileged instruction from user mode, etc.

An ISR is used for the OS to handle an exception. The memory addresses of these ISR’s are contained in an interrupt vector.

1. What is the meaning of **split interrupt management**? What is the reason for it?  
   A: Split interrupt management is performed because interrupt handling is in many cases time and resource constrained and therefore difficult to implement. It is performed in two levels:

1. First-level interrupt handler (FLIH) – Actual ISR to do I/O.

2. Second-level interrupt handler (SLIH) – Separately scheduled routine to process the data (without I/0) for the OS.

1. When a DMA controller and a processor both must access the main memory, what happens? Which one gets priority? Or can both access memory simultaneously? Does it slow down the CPU? Does the DMA controller update the processor's data cache?

A: CPU gets the higher priority. No, both cannot access simultaneously.

Yes, some slowdown is observed but in practice not much.

1. Which of the following Unix calls are for which types of devices?

|  |  |  |
| --- | --- | --- |
| API | block device or  character device | synchronous, asynchronous or nonblocking |
| read() | block | Blocking – synchronous |
| write() | block | Blocking - synchronous |
| seek() | block |  |
| get() | character |  |
| put() | character |  |
| select() | block | synchronous (blocking) |

1. What is the purpose of ioctl()?

A: In Unix, It’s purpose is to control specific I/O devices throughout I/O specific commands. A “backdoor” to an I/O device.

1. What is the meaning of **vectored I/O**, and why is it a good idea?

A: In vectored I/O, one system call is allowed to perform multiple I/O operations. It is a good idea because it decreases changes between user and kernel modes and context switching at the user level is reduced.

1. Does every I/O system call cause a device driver to be invoked? Why or why not?

A: Not necessarily. If the data is readily available in the buffer cache, it is not necessary to invoke a device driver. Otherwise, physical I/O must be performed.

--- Chapter 13 ---

1. Where are the metadata such as file name of a file stored and where is it kept?

A: It is stored in disk and kept in a registry or metadata file.

1. What is the **file pointer** and what are different ways it can be moved?

A: A file pointer points to the last read/write location in file. Kernel location within the file. Every written unit increases the file pointer. seek() may also be called to point to a different location (random access) of the file.

1. What does truncate() do and why is it a better idea than delete() and (re-)create the file.

A: Truncate writes over a file and update. It is a better idea to truncate rather than delete because there is no need to recreate the file attributes and thus the OS does not need to reallocate the metadata of a newly created file.

1. Is it a good idea for an OS to define the formats for most types of files? Why or why not?

A: No, it is not a good idea because it makes the OS large and cumbersome. It is better to let each application program interpret the input files and use the appropriate file structure. The OS must support at least one structure -executable file- in order to load and run programs.

## 2. Programming Exercise

There is no programming exercise this week, but be sure you are caught up with your project checkpoint.