# SDEV-385 Homework Week 12

Jarred Glaser

11/21/2020

1. *Consider the following I/O scenarios on a single-user PC:*

* *A mouse used with a graphical user interface*
* *A tape drive on a multitasking operating system (assume no device preallocation is available)*
* *A disk drive containing user files*
* *A graphics card with direct bus connection, accessible through memory-mapped I/O*

*For each of these I/O scenarios, would you design the operating system to use buffering, spooling, caching, or a combination? Would you use polled I/O, or interrupt-driven I/O? Give reasons for your choices.*

**A mouse used with a graphical user interface**

* Buffering - may be used to store operations from the mouse when other operations that are higher in priority are taking place.
* Spooling - no need for spooling as only one user will be using a single mouse at a given time
* Caching - we don't need to store "copies" of the data from the mouse so caching is not needed either.
* Interrupt-driven I/O should be used here, as it would be inefficient to constantly poll the mouse for input, rather than to just have the mouse alert the system when an action is needed

**A tape drive on a multitasking operating system (assume no device preallocation is available)**

* Buffering - may be used to increase throughput time between the tape drive and source/destination of I/O
* Spooling - Since this is a multitasking OS, spooling may be useful to handle multiple users that want to read from or write to the tape drive
* Caching - may be useful for quicker access to data stored on the tape drive
* Interrupt-driven I/O should be used here for the same reason as the mouse from above.

**A disk drive containing user files**

* Buffering - may be used to increase throughput time between the disk and source/destination of I/O (same reason as tape drive)
* Spooling - Not necessary because multiple users can access a disk at once
* Caching - may be helpful to cache commonly accessed data from disk for improved access speeds
* Interrupt-driven I/O is best used here

**A graphics card with direct bus connection, accessible through memory-mapped I/O**

* Buffering - can be used to increase performance
* Spooling - not needed because the direct bus connection already makes this a very fast device
* Caching - not needed for the same reason as spooling
* Since this is a memory-mapped device with a direct bus connection, neither interrupt-driven I/O or polling is necessary

1. *In most multi-programmed systems, user programs access memory through virtual addresses, while the operating system uses raw physical addresses to access memory. What are the implications of this design for the initiation of I/O operations by the user program and their execution by the operating system?*

When the kernel needs to run an I/O operation from a user application, it must translate the virtual address space to actual address space. This can cause overhead and slow down the I/O operation.

1. *Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their device is ready?*

Blocking I/O - good for when operating system is waiting for a single event from a device and only one operation from the device can be accepted at a time

* e.g. keyboard, mouse, hard drive

Non-blocking I/O - good for when there are multiple operations to/from a device that can be accepted at one time.

* e.g. network server sockets accepting many input operations at the same time, applications that accept multiple keyboard commands at the same time, logging system accepting input from multiple applications at the same time

Nonblocking I/O is usually more complex to implement for the programmer and busy-wait operations can cause overhead compared to simple blocking I/O

**References**

Operating System Concepts Essentials, 2nd Edition, Silberschatz, Galvin, Gagne, Wiley Press.

(2020). Asynchronous programming. Blocking I/O and non-blocking I/O. Retrieved from <https://luminousmen.com/post/asynchronous-programming-blocking-and-non-blocking#:~:text=However%2C%20there%20is%20also%20a,processed%20at%20some%20later%20point>.