ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Spring Term 2014

Operating Systems and Networks
Assignment 10

Assigned on: **4th May 2014**
Due by: **9th May 2014**

1 I/O Systems

1.1 General Questions

a) *State three advantages and disadvantages of placing functionality in a device controller, rather than in the kernel.*

Answer: Advantages:

- Bugs are less likely to cause an operating system crash.
- Performance can be improved by utilizing dedicated hardware and hard-coded algorithms.
- The kernel is simplified by moving algorithms out of it.

Disadvantages:

- Bugs are harder to fix - a new firmware version or new hardware is needed.
- Improving algorithms likewise require a hardware update rather than just a kernel or device driver update.
- Embedded algorithms could conflict with application's use of the device, causing decreased performance.

b) *Why might a system use interrupt-driven I/O to manage a single serial port, but polling I/O to manage a high performance network interface card?*

Answer: Polling can be more efficient than interrupt-driven I/O. This is the case when the I/O is frequent and of short duration. Even though a single serial port will perform I/O relatively infrequently and should thus use interrupts, a collection of serial ports such as those in a terminal concentrator can produce a lot of short I/O operations, and interrupting for each one could create a heavy load on the system. A well-timed polling loop could alleviate that load without wasting many resources through looping with no I/O needed.

c) *Polling for an I/O completion can waste a large number of CPU cycles if the processor iterates a busy-waiting loop many times before the I/O completes. But if the I/O device is ready for service, polling can be much more efficient than is catching and dispatching an interrupt. Describe a hybrid strategy that combines polling, sleeping and interrupts for I/O device service. For each of these three strategies (pure polling, pure interrupts, hybrid), describe a computing environment in which that strategy is more efficient than is either of the others.*

Answer: A hybrid approach could switch between polling and interrupts depending on the length of the I/O operation wait. For example, we could poll and loop N times and if the device is still busy at N+1, we could set an interrupt and sleep. This approach would avoid long busy-waiting cycles. This method would be best for very long or very short busy times. It would be inefficient if the I/O completes at N+T (where T is a small number of cycles) due to the overhead of polling plus setting up and catching interrupts. Pure polling is best with very short wait times. Pure interrupts are best with known long wait times.

d) *The Linux operating system differentiates between character and block devices. What is the difference between them?*

Answer: Accesses to block devices are cached and buffered, while character device accesses are not. Block devices must allow random access.

e) *What is the purpose of an IOMMU?*

Answer: An IOMMU has many benefits, for example:

- Memory is protected from malicious devices, a device cannot read or write memory that has not been mapped.
- Virtualized guest operating systems can use devices that do not explicitly support virtualization.
- evices that do not support memory addresses long enough to address the entire physical memory can still address the entire memory through the IOMMU, avoiding overheads associated with bounce buffers.

1.2 DMA

a) *How does DMA increase system concurrency? How does it complicate hardware design?*

Answer: DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses. Hardware design is complicated because the DMA controller must be integrated into the system and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

b) *Although DMA does not use the CPU, the maximum transfer rate is still limited. Consider reading a block from disk. Name three factors that might ultimately limit the file transfer*

Answer: There are four ways that the maximum transfer rate can be limited:

- Limiting speed of the I/O device - in our case, the disk read throughput.
- Limiting speed of the bus. In this case the bus itself is the bottleneck.
- A disk controller with no internal buffers or too small buffer sizes could also limit the performance of the read file operation.
- Erroneous disk or block read (i.e, if the checksum is incorrect). In this case, an error is signaled and no transfer of the block happens. The block has to be retransmitted.

c) *A DMA controller has four channels. The controller is capable of requesting a 32-bit word every 100 nsec. A response takes equally long. How fast does the bus have to be to avoid being a bottleneck?*

Answer: Each bus transaction has a request and a response each taking 100 nsec, or 200 nsec per bus transaction. This gives 5 million bus transactions / sec. If each one is four bytes, the bus should be able to handle 20 MB/sec. The fact that these transactions may be distributed over four I/O devices (four channels) in round-robin fashion is irrelevant. A bus transaction takes 200 nsec, regardless of whether consecutive requests are to the same device or different device, so the number of channels the DMA controller has does not matter.