Discussion Week 8

TA: Kyle Dewey

Overview

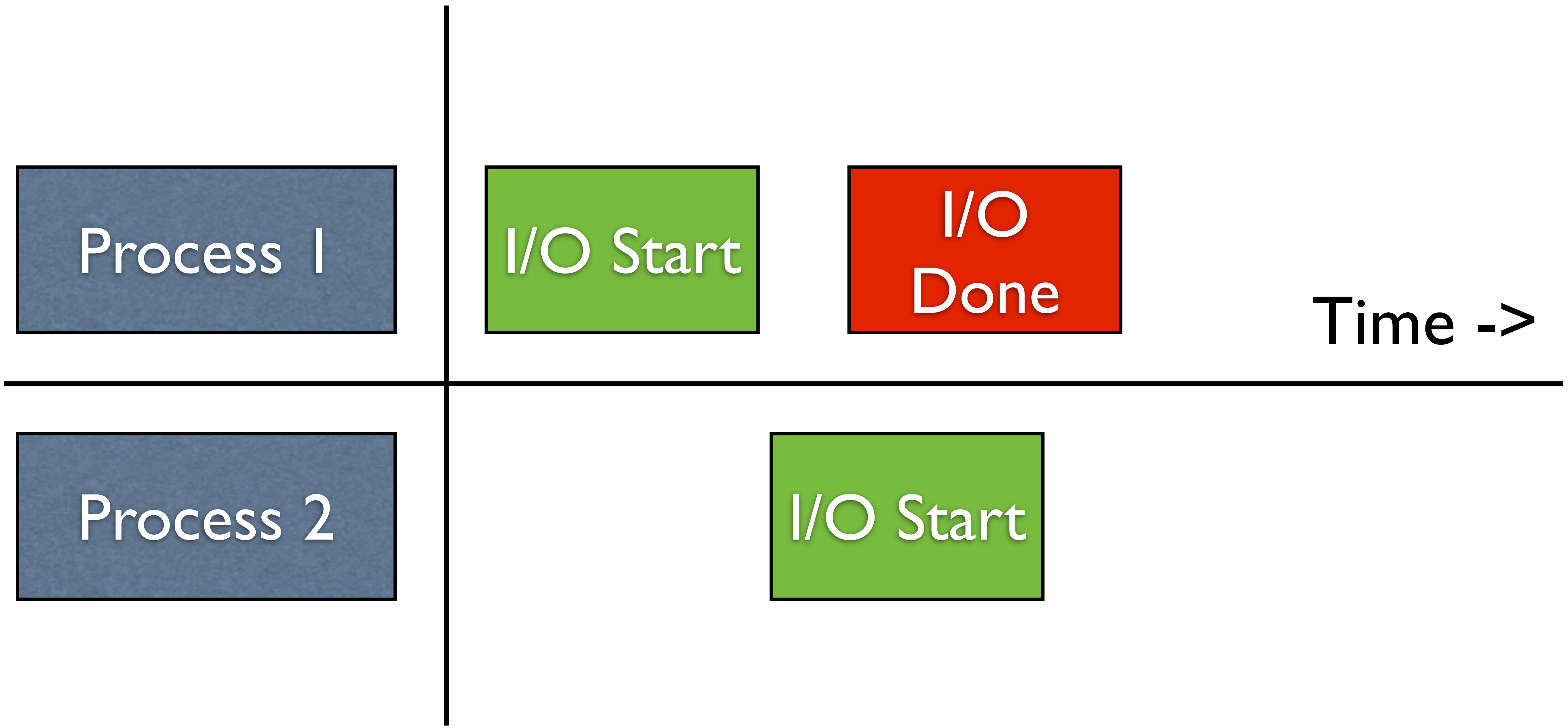
- Exams
- Interrupt priority
- Direct memory access (DMA)
- Different kinds of I/O calls
- Caching
- What I/O looks like

Exams

Interrupt Priority

- Process 1 makes an I/O request
- Process 2 makes an I/O request
- While setting up Process 2's request, Process 1's request finishes

Interrupt Priority



Prioritizing

- While servicing one interrupt, another interrupt occurred
- Question: should we finish our ISR or allow the other ISR to run immediately?

Prioritizing

- Which ISRs preempt which is potentially complex
- Preemption within the kernel can be complex
  - Similar issues as with process preemption

Performing I/O

- "I/O takes forever, so while we do I/O, we schedule something else."
    - ...so how do we do I/O if we're doing something else?

DMA Controller

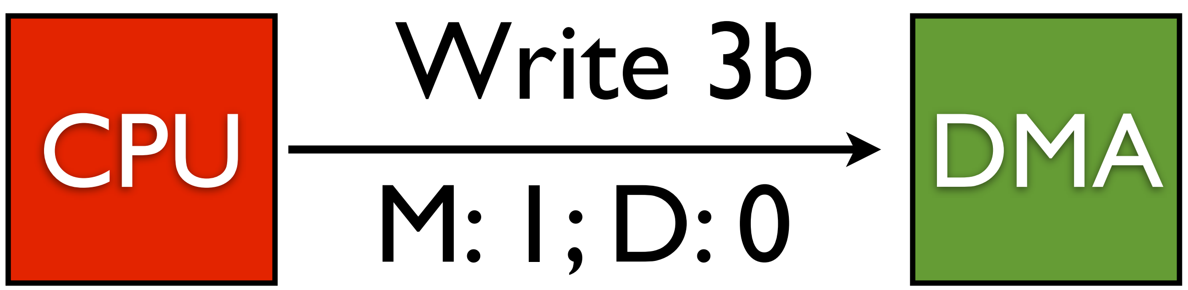
- Answer: special hardware
- Dedicated processor just for handling I/O
- The processor directly manipulates memory and the I/O device, bypassing the CPU

DMA Messages

- Read X bytes starting at Y part of disk into Z part of memory
- Write X bytes to disk starting at part Y of disk from Z part of memory

DMA in Action

Time 1



Memory

x	f	o	o

Disk

a	b	c	d

Time 2

CPU Schedules Process

DMA Controller Does I/O

Memory

x	f	o	o

Disk

f	b	c	d

DMA in Action

Times 3-4

CPU Runs Process

DMA Controller Does I/O

Memory

x	f	o	o

Disk

f	o	o	d

Time 5

CPU ← I/O Done — DMA

Memory

x	f	o	o

Disk

f	b	c	d

DMA Issues

- Question: How does this work in with virtual memory?
- Question: How does this work with swappable pages?

I/O Types

Blocking/Nonblocking

- Blocking: wait until I/O is complete until returning
- Nonblocking: on each call, return what has been done so far

Nonblocking

- Question: Why is this useful?
- Consider a program that gets the sum of integers from a 1 GB file

Synchronous/ Asynchronous

- Synchronous: wait for I/O to complete before returning
- Asynchronous: do not wait for I/O to complete before returning
  - Ideally, there is a callback to signal when the I/O is complete

Asynchronous vs. Nonblocking

- DMA is asynchronous
- Asynchronous is still all or nothing, like a typical synchronous blocking I/O
- Nonblocking can get partial results
- Note: Wikipedia lies

I/O Calls

	Synchronous	Asynchronous
Blocking	Common	Depends...
Nonblocking	Nonsensical	Possible

But I/O is still slow...

Caching

- Idea: Do as much “I/O” in memory as possible

```
file = open( fileName, O_RDWR );
read( file, buffer, 50 );
... // more code that changes buffer
lseek( file, 0, SEEK_SET );
write( file, buffer, 50 );
close( file )
```

Caching

- Massive performance gains possible
- Real systems use this all the time
- Can be quite complex

UNIX Semantics

- A write() is immediately available to anything that calls read()
- How does this work with caching?

Consistency

- Cache is by definition a copy of the actual resource
- Any access to that resource must behave as if caching is not being performed
- They **will** be out of sync, but they **cannot** look out of sync

Example Revisited

- How does the OS handle this?

```
file = open( fileName, O_RDWR );
read( file, buffer, 50 );
... // more code that changes buffer
lseek( file, 0, SEEK_SET );
write( file, buffer, 50 );
strcpy( buffer, "Poisoned buffer" );
close( file )
```

The face of I/O

Variation

Variation

- “I/O” is extremely broad
- ...yet we try to make it fit into a consistent interface
- Major differences in OSes are often seen in how they treat I/O

Broad Classes

- Character based: One character at a time is manipulated
    - Examples: keyboards, mice, network cards
- Block based: Blocks of characters are manipulated at a time
    - Examples: Hard drives, flash drives

Broad Classes

- Some manage not to fit into even these
- Example: tape drives
  - A seek can take a **long** time - impractical under essentially any circumstance
  - Character devices have no notion of seeking
  - However, data is stored in blocks...

Floppy Drives

- Often used the same interface as tape drives
- More naturally a block device

Responding to I/O

- Interrupts or polling can be used
- Some support both
- Which is best depends on usage scenario
  - Polling makes little sense for hard drives - why?
  - Interrupts are not usually used for mice - why?

I/O Space

- I/O devices often have a limited amount of space they can work with internally
- If the OS fails to respond before this space fills, data can be lost
  - Example: keyboards

Buffering

- Buffer: a place in memory to put this data
- Not *quite* the same as a cache
  - Not a copy
  - Note that a cache can also be a buffer, but a buffer is by definition not a cache