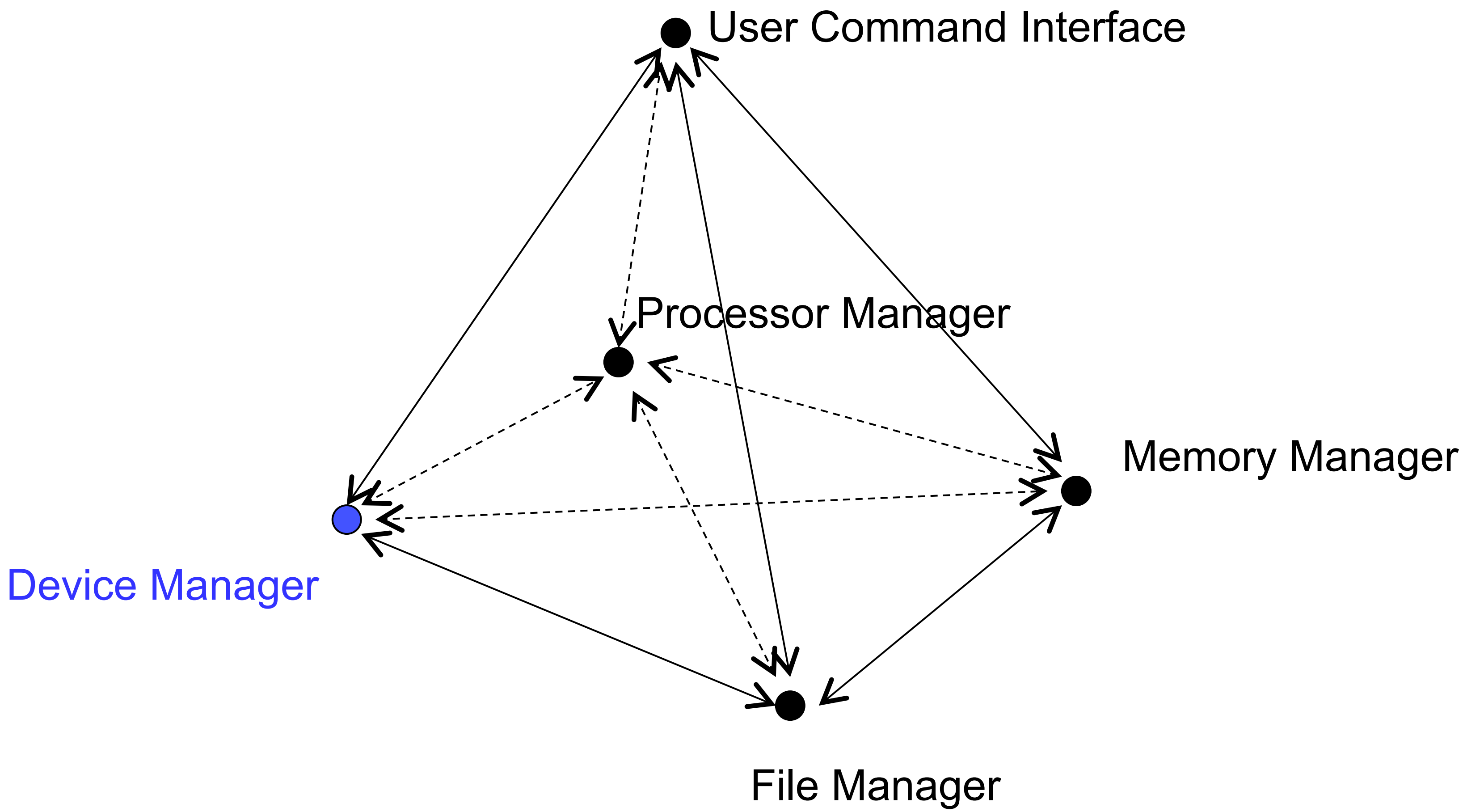
Comp 204: Computer Systems and Their Implementation

Lecture 18: Devices

Today

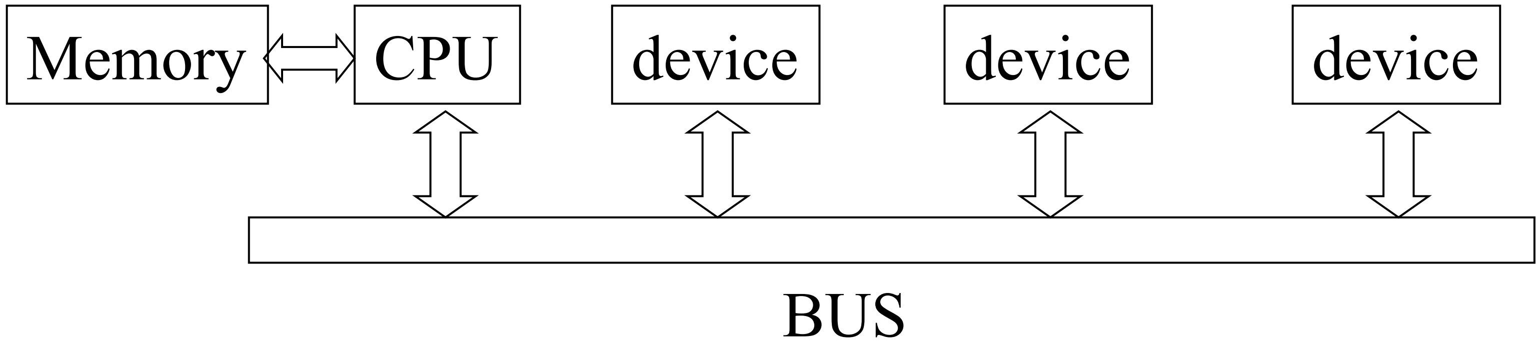
- Devices
  - Introduction
  - Handling I/O
    - Device handling
    - Buffering and caching

Operating System – An Abstract View



Devices

- Peripheral devices connect to ports on the computer
- Data and commands to/from devices may travel along a shared set of wires called a bus (e.g. PCI bus)
  - Devices ignore messages not intended for them
  - Problem of bus contention



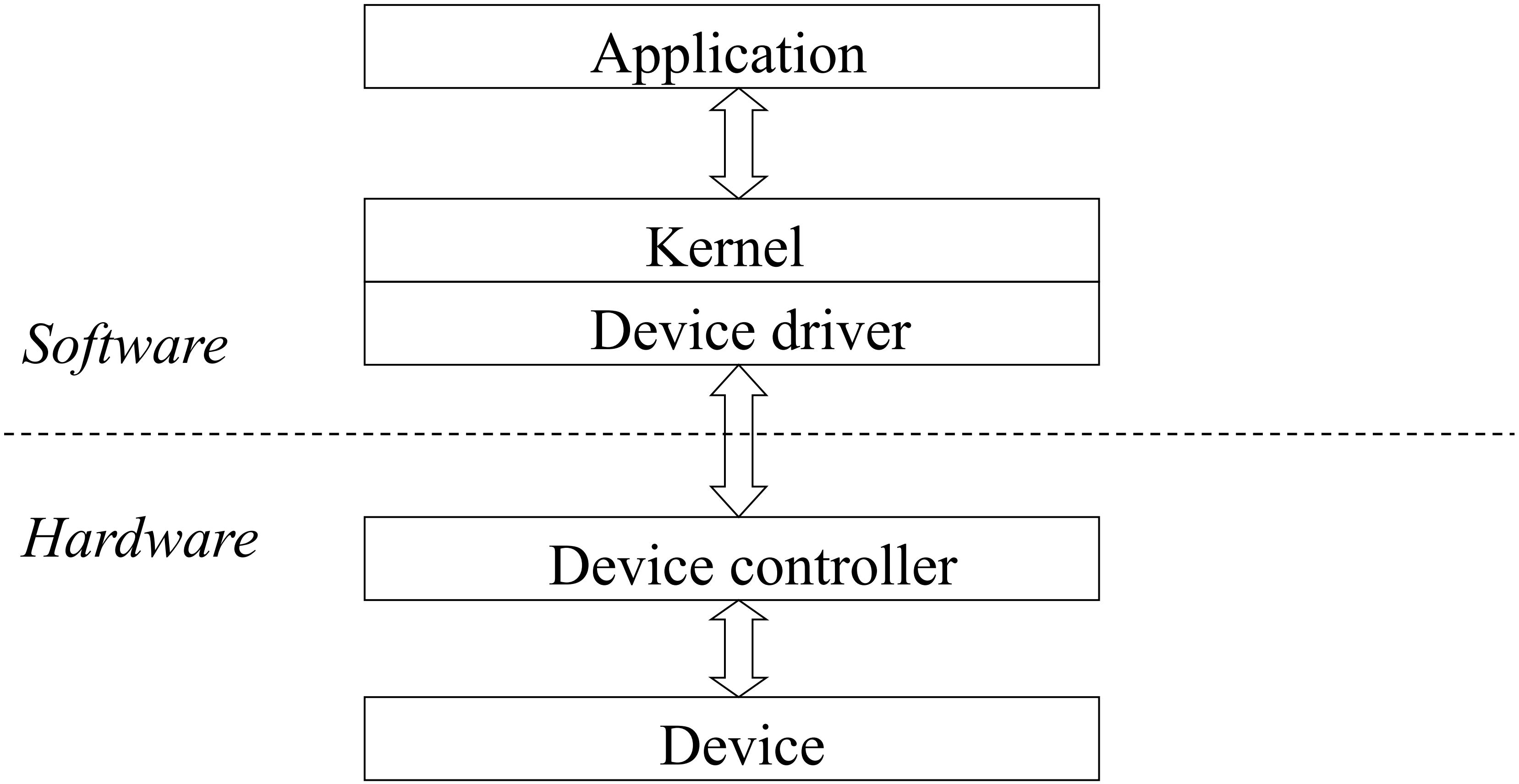
Communication

- Devices usually have several registers:
  - Status reg: indicates busy/ready etc.
  - Command/control reg: to pass commands to device
  - Data regs: to send/receive data
- CPU may have special I/O instructions to alter/ inspect device registers
- Often, registers are mapped onto memory locations
  - e.g. writing to location 100 might send a command to a device

Polling vs. Interrupts

- OS needs to know when device ready for transfers
- Can poll device status
  - Busy-waiting may be inefficient
  - Occasional polling may risk losing data
- Alternative is interrupts
  - CPU interrupted when device has data or is ready to accept data
- e.g. Pentium
  - Interrupts 0-31 non-maskable for error conditions etc.
  - Interrupts 32-255 maskable for devices

I/O Handling



Application
Kernel
Device driver
Software
Hardware
Device controller
Device

Application I/O Interface

- I/O devices can be categorised by their behaviours into generic classes
  - Each general type is accessed through an interface, which is a standard set of functions (though the exact system calls may vary across different OS)
- Device driver layer hides differences among I/O controllers from kernel
- Devices vary on many dimensions
  - Character-stream vs. block
  - Sequential vs. random access
  - Sharable vs. dedicated
  - Speed of operation
  - Read-write, read only, or write only

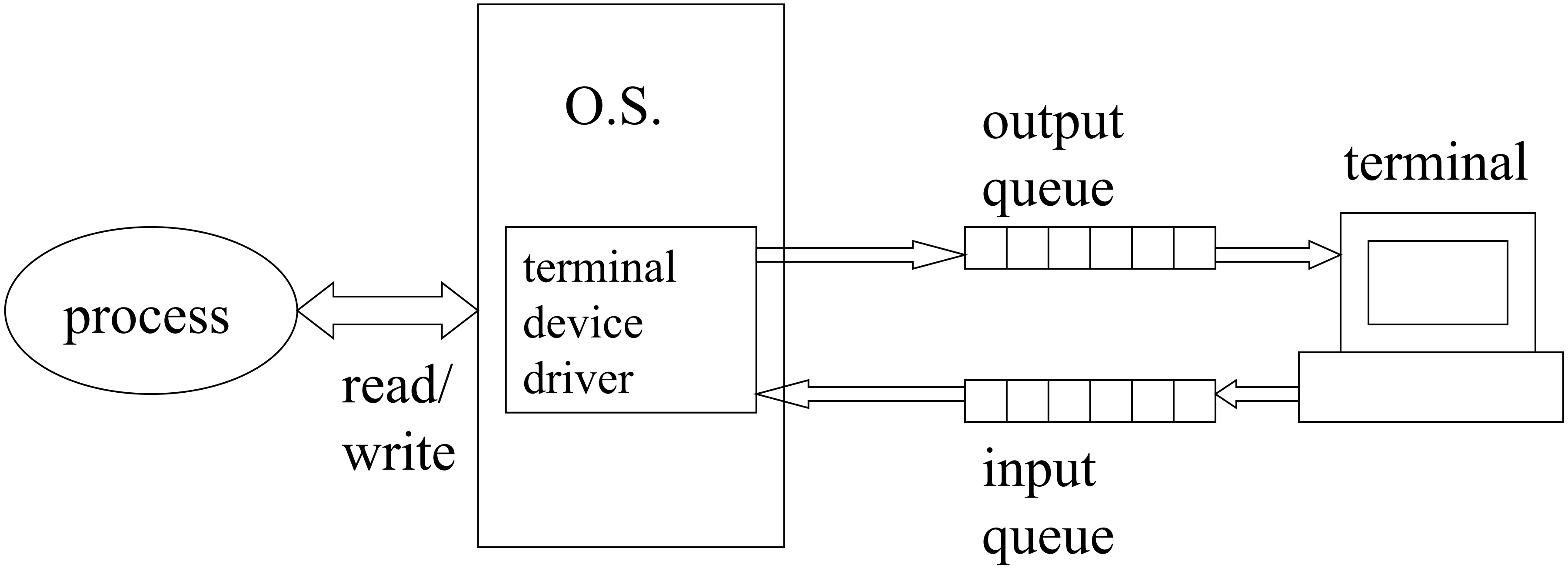
Device Handling

- Device driver converts system calls such as open, read, write, close to low-level commands to control device

- Device controller converts commands to electronic signals operating the hardware

- Application interface
  - e.g. Unix: /dev directory holds special files, one per device (e.g. /dev/tty)
  - accessing special file activates device driver
  - System call ioctl() can be used to pass arbitrary commands to device driver

Example: Unix terminal



Terminal Handling

- Characters typed at keyboard are entered into input queue by device driver
- To echo, driver copies input queue to output queue
- Some characters require further processing by device driver
  - e.g. backspace
    - remove item from input queue
- When read request made, pass contents of input queue to process

Blocking and Non-Blocking I/O

- Blocking: process suspended until I/O completed
  - Process moves from running to waiting
  - Easy to use and understand, but insufficient for some processes' needs
    - e.g. keyboard input and display on screen
- Non-blocking: overlap execution with I/O
  - Can be implemented via multi-threading: some threads block, others continue executing
  - Non-blocking I/O system calls: call returns quickly with value indicating number of bytes read or written
- Asynchronous: system call returns immediately so process runs while I/O executes
  - Process informed when I/O completed at some future time

I/O Scheduling

- I/O requests need to be scheduled to execute in an efficient order
- A good ordering can improve system performance, ensure devices are shared fairly amongst processes and reduce average I/O completion wait time
- Scheduling done via wait queues for each device
  - I/O scheduler may re-arrange the order of the queue to improve efficiency and response
  - Priority may be given to requests requiring a fast response
  - Choice of different scheduling algorithms available for disk I/O
    - e.g. FCFS, Shortest-Seek-Time-First (SSTF), etc.

Buffering

- Consider reading a sequence of characters from a device
  - Making a read request for each char. is costly
- Instead, set up an area of memory called a buffer
  - read a block of chars into buffer in one operation
  - subsequent chars taken directly from buffer
  - only need to access device when buffer empties
- Similarly for writing
  - place each char in a buffer
  - send to device only when buffer full

Buffering

- Double buffering
  - read/write one buffer while other being filled/ emptied
- Buffering may be done by
  - software, e.g. operating system or library routines
  - hardware, e.g. disk drive
- Direct Memory Access (DMA)
  - fast devices (e.g. disk) may write directly into memory buffer, interrupting CPU only when finished
  - CPU might be delayed while DMA controller accesses memory (cycle stealing)
- Buffer writes can cause inconsistency problems
  - may need to flush buffers periodically (e.g. Unix *sync* operation every 30 secs)

Caching

- Similar to buffering, but idea is to speed up access to frequently used items by keeping copies in a faster medium (the cache)

- Other differences:

Buffer	Cache
Items viewed as data in transit	Items viewed as copies of the original
FIFO	Random access
Once item read, viewed as deleted	Items may be read many times

Question

- To assist in locating a bug that is causing a program to crash, a programmer inserts print statements as follows:

```
begin
  ...
  print("Got to point A without crashing");
  ...
  print("Got to B without crashing");
  ...
end
```

a) Too much information will probably be written to the screen to allow location of the bug.
b) The very insertion of the print statements will probably alter the program's behaviour, preventing the bug from occurring.
c) The new diagnostic statements will interfere with the program's existing output, introducing further bugs.
d) The bug will probably make all print statements inoperable.
e) The use of output buffers by the system might prevent some messages from being written.

Answer: e
The buffers may not be flushed, and the program may continue (and crash) giving misleading information.

Spooling

- Some devices non-sharable
  - e.g. printer: multiple processes cannot write to it simultaneously
- Solution is a daemon process called a spooler
  - SPOOL: Simultaneous Peripheral Operations On-Line
- Processes send their printer output via spooler daemon
- Spooler creates a temp file for each process, and writes output to those files
- When process completes, spooler adds file to a queue for printing (de-spooling)

Performance

- I/O is a major factor in system performance: heavy demands are placed on the CPU
  - Device driver code must be executed and processes scheduled efficiently as they block and unblock
  - Involves large amount of context switching
  - Network traffic adds to this

- Measures can be taken to improve performance that include:
  - Reducing the number of context switches
  - Reduce interrupt frequency by using large transfers and polling
  - Making use of DMA

  ...

End of Section

- Files and I/O
  - Files and directory structure
  - Filestore allocation policies
  - Device handling
  - Buffering and caching

- The next section of the module will be Compilers