Study Set for Lecture 14: I/O Systems

Due Dec 7 at 11:59pm **Points** 10 **Questions** 9
Available Nov 24 at 12pm - Dec 8 at 8pm 14 days **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect14_I_O_Systems.pdf**.

Then answer the questions from this study set and submit them to gain access to the further part of the course.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	9,479 minutes	10 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **10** out of 10 *
Submitted Dec 6 at 9:55am
This attempt took 9,479 minutes.

Question 1 **Not yet graded / 0 pts**

Using bit fields in a C struct design a type for variables that can be used to encode the following controls for an imaginary serial port:

- a choice between full-duplex and half-duplex communication,
- enabling or disabling parity checking with selection for no parity, odd parity, or even parity,

- selection of the word of data length that can range from 5 to 8 bits, and
- selection of one of the 8 speeds supported by the port.

For each field, provide comments that describe the value choices for each of the options.

When the struct is ready, declare a variable of its type, and then using the hex format set the fields to configure the port for communicating over a full-duplex, odd-parity, 6-bit-word, and the highest speed available channel.

Your Answer:

typedef struct serial_r{

//Two selections
unsigned duplex: 1;
//Three selections
unsigned parity: 2;
//5-8 bits -> 4 bits 2^2
unsigned word_len: 2;
//Selection of one of the 8 speeds -> 8 bits 2^3
unsigned speed: 3;

} SERIAL_R

SERIAL_R register= {0x0, 0x1, 0x3, 0x5};

Bit Field Declaration

The declaration of a bit-field has the following form inside a structure −

```
struct {
   type [member_name] : width ;
};
```

The following table describes the variable elements of a bit field −

Sr.No.	Element & Description
1	**type**

	An integer type that determines how a bit-field's value is interpreted. The type may be int, signed int, or unsigned int.
2	**member_name** The name of the bit-field.
3	**width** The number of bits in the bit-field. The width must be less than or equal to the bit width of the specified type.

The variables defined with a predefined width are called **bit fields**. A bit field can hold more than a single bit; for example, if you need a variable to store a value from 0 to 7, then you can define a bit field with a width of 3 bits as follows −

```
struct {
   unsigned int age : 3;
} Age;
```

The above structure definition instructs the C compiler that the age variable is going to use only 3 bits to store the value. If you try to use more than 3 bits, then it will not allow you to do so. Let us try the following example −

Live Demo **(http://tpcg.io/wVJ3IM)**

```
#include <stdio.h>
#include <string.h>

struct {
   unsigned int age : 3;
} Age;

int main( ) {

   Age.age = 4;
   printf( "Sizeof( Age ) : %d\n", sizeof(Age) );
   printf( "Age.age : %d\n", Age.age );

   Age.age = 7;
   printf( "Age.age : %d\n", Age.age );

   Age.age = 8;
   printf( "Age.age : %d\n", Age.age );
```

```
   return 0;
}
```

When the above code is compiled it will compile with a warning and when executed, it produces the following result −

```
Sizeof( Age ) : 4
Age.age : 4
Age.age : 7
Age.age : 0
```

Question 2

Not yet graded / 0 pts

Describe what a device port is and how it is used to communicate with a device.

Describe a sample layout of the port.

Your Answer:

A device port is a 1 to 1 communication channel, common ones are the parallel port and the serial port. A lot of ports interact with a larger bus which can connect to multiple sources in either direction.

When referring to a physical **device** (https://www.computerhope.com/jargon/d/device.htm), a **hardware port** or **peripheral port** is a hole or connection found on the front or back of a computer. Ports allow computers to **access** (https://www.computerhope.com/jargon/a/access.htm) **external** (https://www.computerhope.com/jargon/e/external.htm) devices such as **printers** (https://www.computerhope.com/jargon/p/printer.htm). Below is a short listing of the different computer ports you may find on a computer.

Question 3

Not yet graded / 0 pts

Describe what are block devices and character devices. How do they differ?

What's the difference in how the OS needs to handle each of them?

Your Answer:

Block devices are devices that transfer data in blocks rather than in individual data bits, commands include read, write, and seek. Gets raw I/O or file system access and memory-mapped file access is possible.

Character devices get input character by character, like keyboards and serial ports. Commands include get and put and libraries layered on top allow line editing.

- A Character ('c') Device is one with which the Driver communicates by sending and receiving single characters (bytes, octets).
- A Block ('b') Device is one with which the Driver communicates by sending entire blocks of data.
- Examples for Character Devices: serial ports, parallel ports, sounds cards.
- Examples for Block Devices: hard disks, USB cameras, Disk-On-Key.

Question 4

Not yet graded / 0 pts

What is polling?

How does a polling-based I/O differ from interrupt-based I/O?

Your Answer:

Polling is a way of communicating with a device, where you constantly ask the device what its state is. Can cause busy-wait cycle when waiting for I/O device, but can use a timer to remedy this.

Polling is pervasive throughout CS whenever one program, process, or similar is waiting on another. Basically process A somehow asks process B "do you have anything for me?" repeatedly until it gets something.

This can be anywhere from a very low level (e.g. polling the status of a physical wire looking for a change in voltage) to a very high level (e.g. one server sending a web request to another to see if the price on an item has changed).

Polling is relatively simple to implement and is often (though not always) the right tool for the job. In UNIX-style socket programming, you'll often see poll() compared with select(), which is interrupt-driven (more of a "just wake me up when you have something; I'll sleep for now and let the system do other things" approach).

Interrupt based runs on a CPU interrupt triggered by the I/O device. The interrupts can be masked so they are ignored or delayed. Doesn't cause busy-waiting as long as the I/O eventually goes through

Question 5

Not yet graded / 0 pts

Describe in general terms what buffering is?

Why is buffering needed in dealing with I/O from and to devices?

How is buffering different from caching?

How is buffering different from spooling?

Your Answer:

Buffering is storing data in memory while transferring between devices.

Buffering is needed to cope with the device speed mismatch and the transfer size mismatch and to maintain the collection of coherent chunks.

Caching is used for performance sake since it's main operation is quickly copying data, while the buffer is just for coping with device mismatch.

Spooling is for holding the output of a device if a device can serve only one request at a time, just so it doesn't get overwhelmed with requests.

EXTRA:

Buffers are often used in conjunction with **I/O** (https://en.wikipedia.org/wiki/I/O) to **hardware** (https://en.wikipedia.org/wiki/Computer_hardware), such as **disk drives** (https://en.wikipedia.org/wiki/Disk_drives), sending or receiving data to or from a **network** (https://en.wikipedia.org/wiki/Computer_network), or playing sound on a speaker. A line to a **rollercoaster** (https://en.wikipedia.org/wiki/Rollercoaster) in an amusement park shares many similarities. People who ride the coaster come in at an unknown and often variable pace, but the roller coaster will be able to load people in bursts (as a coaster arrives and is loaded). The **queue area** (https://en.wikipedia.org/wiki/Queue_area) acts as a buffer—a temporary space where those wishing to ride wait until the ride is available. Buffers are usually used in a **FIFO** (https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)) (first in, first out) method, outputting data in the order it arrived.

Buffers can increase application performance by allowing **synchronous** (https://en.wikipedia.org/wiki/Synchronous) operations such as file reads or writes to complete quickly instead of blocking while waiting for hardware interrupts to access a physical disk subsystem; instead, an operating system can immediately return a successful result from an API call, allowing an application to continue processing while the kernel completes the disk operation in the background. Further benefits can be achieved if the application is reading or writing small blocks of data that do not correspond to the block size of the disk subsystem, allowing a buffer to be used to aggregate many smaller read or write operations into

block sizes that are more efficient for the disk subsystem, or in the case of a read, sometimes to completely avoid having to physically access a disk.

Question 6

Not yet graded / 0 pts

Describe what a device driver is and how it helps the operating system to operate with multitude of devices out there.

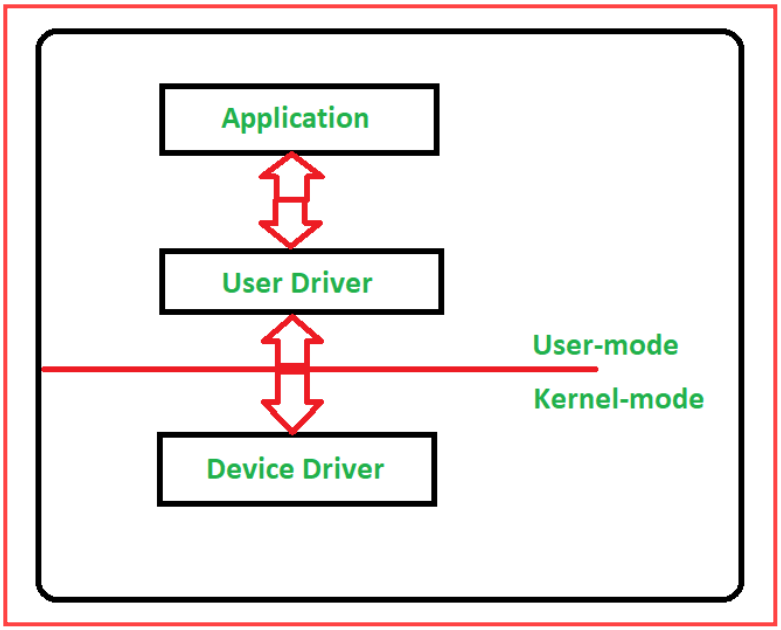
Your Answer:

Device drivers are classes that hide differences among I/O contollers from the kernel. Each class serves similar devices.

Device Driver in computing refers to a special kind of software program or a specific type of software application which controls a specific hardware device that enables different hardware devices for communication with the computer's Operating System **(https://www.geeksforgeeks.org/introduction-of-operating-system-set-1/)** A device driver communicates with the computer hardwares by computer subsystem or computer bus connected to the hardware.

Device Drivers are very essential for a computer system to work properly because without device driver the particular hardware fails to work accordingly means it fails in doing a particular function/action for which it has been created.

In a very common way most term it as only a **Driver** also when someone says **Hardware Driver** that also refers to this **Device Driver.**



Types of Device Driver:
For almost every device associated with the computer system there exist Device Driver for the particular hardware.But it can be broadly classified into two types i.e.,

1. **Kernel-mode Device Driver –**
   This Kernel-mode device driver includes some generic hardwares which loads with operating System as part the OS these are BIOS, motherboard, processor and some other hardwares which are part of kernel software. These includes the minimum system requirement device drivers for each operating system.
2. **User-mode Device Driver –**
   Other than the devices which are brought by kernel for working of the system the user also bring some devices for use during the using of a system that devices needs device drivers to functions those drivers falls under User mode device driver. For example user needs any plug and play action that comes under this.

Question 7

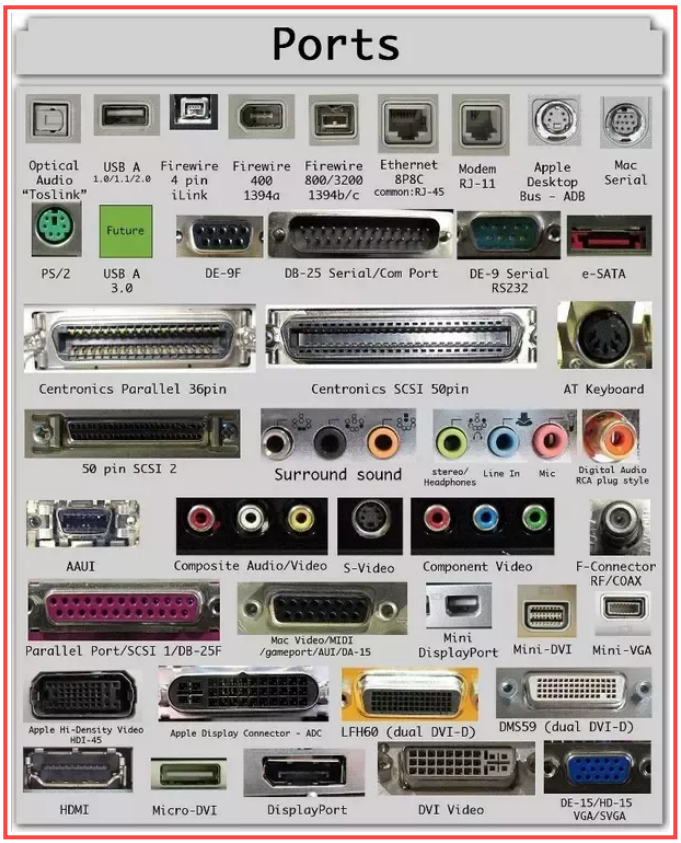
Not yet graded / 0 pts

Compare port-based communication with bus-based communication.

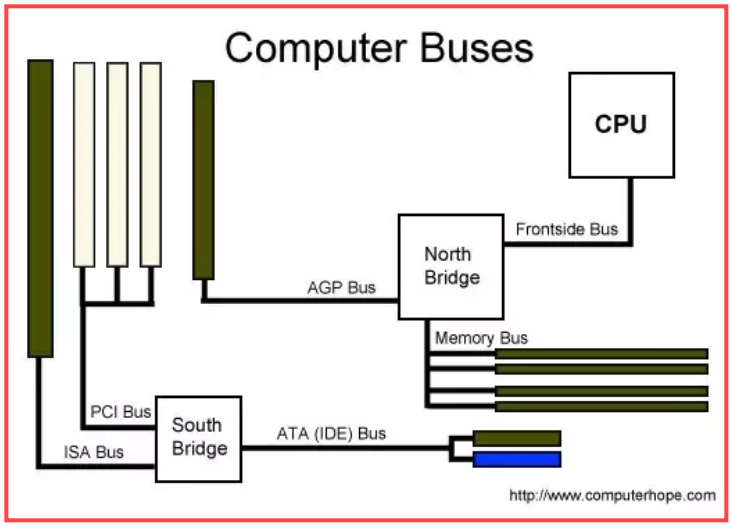
Your Answer:

in port based input output the input output peripheral is connected to the general port of the processor while in bus best input output the input output peripheral is connected via address bus

- A port is commonly defined as a connection point (or interface) between external hardware, like a mouse, keyboard or harddrive.
  - An example is the **serial port (https://en.wikipedia.org/wiki/Serial_port)** .



- A bus is a communication system that transfers data between components inside a computer.
  - An example is the bus that connects your **CPU (https://en.wikipedia.org/wiki/Central_processing_unit)** to your internal memory. It's inside of the motherboard & needs to be very fast.

- 

So to summarise, think of an interface as basically anything that connects up two different parts, this includes a bus or port. A protocol is basically a definition of something, a set of rules everyone agreed upon so communication can be smooth. A bus is a specific kind of interface and it uses an implementation of some protocol to manage communication going through itself, so the CPU knows what to say at what time to access memory and the memory knows how to reply with data when it is requested. A port is basically a kind of external bus, right now way slower than an average bus, since speed is no bottleneck here. A port mostly connects stuff from the outside of the pc and a bus connects things inside.

Question 8

Not yet graded / 0 pts

Describe how CPU communicates with devices using Memory-Mapped I/O (MMIO).

Describe how CPU communicates with devices using Port-Mapped I/O (PMIO).

Explain the key difference between MMIO and PMIO.

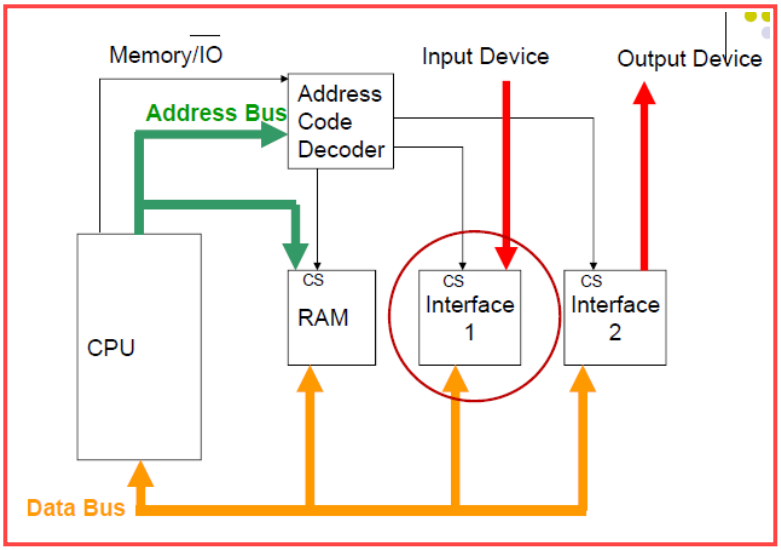
Your Answer:

Memory-mapped I/O and port-mapped I/O are two complementary methods for I/O.

Memory-Mapped I/O

In memory-mapped systems, the I/O device is accessed like it is a part of the memory. `Load` and `Store` commands are executed for reading from and writing to I/O devices, just like they are used for the memory (port-mapped has special commands for I/O). This means I/O devices use the same address bus as memory, meaning that CPU can refer to memory *or* the I/O device based on the value of the address. This approach requires isolation in the address space: that is, addresses reserved for I/O should not be available to physical memory.

Below is an image of a *simple, basic computer system*. The case is much more complicated in contemporary systems.



Port-Mapped I/O

According to **Wikipedia** **(http://en.wikipedia.org/wiki/Port-mapped_IO)**

> Port-mapped I/O often uses a special class of CPU instructions specifically for performing I/O. This is found on Intel microprocessors, with the IN and OUT instructions. These instructions can read and write one to four bytes (outb, outw, outl) to an I/O device. I/O devices have a separate address space from general memory, either accomplished by an extra "I/O" pin on the CPU's physical interface, or an entire bus dedicated to I/O. Because the address space for I/O is isolated from that for main memory, this is sometimes referred to as isolated I/O.

As for the advantages and disadvantages: since the peripheral devices are slower than the memory, sharing data and address buses may slow the memory access. On the other hand, by the I/O simplicity memory-mapped systems provide, CPU requires less internal logic and this helps for faster, cheaper, less power consuming CPUs to be implemented. The logic is similar to that of RISC systems: reduce the complexity, get a more dedicated and a robust system which comes quite handy for embedded systems, for example.

On the contrary (again from Wiki):

> Port-mapped I/O instructions are often very limited, often providing only for simple load and store operations between CPU registers and I/O ports, so that, for example, to add a constant to a port-mapped device register would require three instructions: read the port to a CPU register, add the constant to the CPU register, and write the result back to the port.

Question 9

10 / 10 pts

I have submitted answers to all questions in this study set.

- (●) True
- ( ) False

Quiz Score: **10** out of 10