COMPSCI 2GA3 A5

Elite Lu

Theory Questions

A

Buffering is useful in I/O operations because system calls are expensive. As a result, using buffering would reduce the number of those calls and performs them simultaneously. The number of system calls can be reduced by a factor of K given that there are K bytes in a buffer. The cases I would see an improvement in performance would be most apparent in cases where there is an equal amount of computational work and I/O operations. For example, reading every char out of a file.

B

The upper half of a device driver is responsible for providing the OS interface to user-space programs. These are the system calls. The lower half runs asynchronously to the top half.
In order to communicate, they use shared variables along with buffers and mutually exclusive execution, which is known as mutex.

C

In general, the main difference between serial and parallel interfaces is that parallel has multiple data lines and the number of lines is equivalent to the interface width whereas serial interfaces only have a single data line that can only transfer one at a time.
For parallel interfaces, the main pro is that there is a lower latency per word as it can sent all at once, thus resulting in the higher output. However, this comes at the downside of having problems via interference of data. With serial interface, having only one data line makes It have higher latency since each word is sent one after the other. However, serial interfaces could have a higher rate of transfer and does not have to deal with interference.
When calculating how long it would take for each part, I will use a table to display my results. The formula I used for calculating the parallel interface would be dividing the total bits by the width and multiplying it by the latency. With the serial interface, I will just be multiplying by total bits by the latency to get the bits per second.

	Parallel	Serial
16-bit	200 ms	320 ms
32-bit	400 ms	640 ms

In order to calculate the throughput of each interface in kilobits per second, I will divide the number of bits in the width by the latency. In order to convert to the values to kilobits, I will divide again by 1000 since there are 1000 bits in a kilobit. In addition to this, I will multiply by one million to convert from per microsecond to second. With this, I am multiplying my resultant answer by a factor of 1000. With this, I would get following:

	Parallel	Serial
Throughput	80 kilobits/second	50 kilobits/second

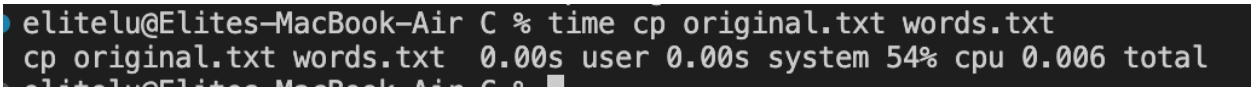
D

The main advantages of Direct Memory Access is that it fetches and stores directly from the memory and only triggers an interrupt when the buffer is full. In addition to this, it improves performance by allowing the transfer of data between a device and memory without using the processor. The buffer and operation chaining can improve the DMA performance since the buffer can reduce the number of system calls. In

addition to this, the processor may have difficulty servicing an interrupt in time to capture the next packet. With buffer and operation chaining, the device can place incoming packets in successive buffers.

Coding Part

Note: I used stdio.h. However, I did not use it for the read and write functions. The only thing I used it for was for the printf function.
For the cp line, this was what happened when I typed in the following:



Here, I have each time I ran the function with the time function when I changed the buffer length:
4: ./a.out original.txt words.txt 0.01s user 0.05s system 18% cpu 0.292 total
16: ./a.out original.txt words.txt 0.00s user 0.02s system 8% cpu 0.269 total
128: ./a.out original.txt words.txt 0.00s user 0.01s system 3% cpu 0.238 total

As you can see, as the buffer length increases, the amount of time decreases. This is because as the buffer length increases, the number of system calls decrease, thus lowering the cost and number of times the read and write operations are called.